



MCUXpresso SDK Documentation

Release 25.12.00-pvw1



NXP
Oct 20, 2025



Table of contents

1	KW47-LOC	3
1.1	Overview	3
1.2	Getting Started with MCUXpresso SDK Package	3
1.2.1	Getting Started with Package	3
1.3	Getting Started with MCUXpresso SDK GitHub	18
1.3.1	Getting Started with MCUXpresso SDK Repository	19
1.4	Release Notes	31
1.4.1	MCUXpresso SDK Release Notes	31
1.5	ChangeLog	43
1.5.1	MCUXpresso SDK Changelog	43
1.6	Driver API Reference Manual	135
1.7	Middleware Documentation	135
1.7.1	Wireless Bluetooth LE host stack and applications	135
1.7.2	Wireless Connectivity Framework	135
1.7.3	Multicore	135
1.7.4	FreeMASTER	136
1.7.5	FreeRTOS	136
2	KW47B42ZB7	137
2.1	CACHE: LPCAC CACHE Memory Controller	137
2.2	CCM32K: 32kHz Clock Control Module	138
2.3	CE Basic Functions	148
2.4	CE Command Functions	148
2.5	CE CMSIS Functions	150
2.6	CE Initialization Functions	150
2.7	CE Matrix Functions	151
2.8	CE Transform Functions	155
2.9	Clock Driver	157
2.10	CMC: Core Mode Controller Driver	176
2.11	CRC: Cyclic Redundancy Check Driver	189
2.12	EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	191
2.13	ELEMU: Edgelock Messaging unit driver	210
2.14	EWM: External Watchdog Monitor Driver	210
2.15	FGPIO Driver	213
2.16	C40ESP3 Flash Driver	213
2.17	FlexCAN: Flex Controller Area Network Driver	217
2.18	FlexCAN Driver	217
2.19	FlexCAN eDMA Driver	264
2.20	FlexIO: FlexIO Driver	267
2.21	FlexIO Driver	267
2.22	FlexIO eDMA SPI Driver	284
2.23	FlexIO eDMA UART Driver	288
2.24	FlexIO I2C Master Driver	291
2.25	FlexIO I2S Driver	299
2.26	FlexIO SPI Driver	310
2.27	FlexIO UART Driver	323

2.28	GPIO: General-Purpose Input/Output Driver	333
2.29	GPIO Driver	336
2.30	I3C: I3C Driver	342
2.31	I3C Common Driver	344
2.32	I3C Master Driver	346
2.33	I3C Slave Driver	372
2.34	IMU: Inter CPU Messaging Unit	385
2.35	Common Driver	392
2.36	LIN: Local Interconnect Network Driver	403
2.37	LIN Driver	403
2.38	LIN LPUART Driver	419
2.39	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	426
2.40	LPCMP: Low Power Analog Comparator Driver	445
2.41	LPI2C: Low Power Inter-Integrated Circuit Driver	451
2.42	LPI2C Master Driver	453
2.43	LPI2C Master DMA Driver	467
2.44	LPI2C Slave Driver	469
2.45	LPIT: Low-Power Interrupt Timer	480
2.46	LPSPi: Low Power Serial Peripheral Interface	486
2.47	LPSPi Peripheral driver	486
2.48	LPSPi eDMA Driver	508
2.49	LPTMR: Low-Power Timer	515
2.50	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	520
2.51	LPUART Driver	520
2.52	LPUART eDMA Driver	539
2.53	LTC: LP Trusted Cryptography	542
2.54	LTC AES driver	543
2.55	LTC DES driver	548
2.56	LTC HASH driver	556
2.57	LTC PKHA driver	558
2.58	LTC Blocking APIs	567
2.59	MCM: Miscellaneous Control Module	567
2.60	MSCM: Miscellaneous System Control	572
2.61	MU: Messaging Unit	573
2.62	PORT: Port Control and Interrupts	585
2.63	RTC: Real Time Clock	593
2.64	SEMA42: Hardware Semaphores Driver	599
2.65	SFA: Signal Frequency Analyser	603
2.66	SMSCM: Secure Miscellaneous System Control Module	612
2.67	SPC: System Power Control driver	616
2.68	SYSPM: System Performance Monitor	644
2.69	TPM: Timer PWM Module	647
2.70	TRDC: Trusted Resource Domain Controller	663
2.71	TRGMUX: Trigger Mux Driver	684
2.72	TSTMR: Timestamp Timer Driver	685
2.73	VBAT: Smart Power Switch	686
2.74	VREF: Voltage Reference Driver	693
2.75	WDOG32: 32-bit Watchdog Timer	696
2.76	WUU: Wakeup Unit driver	702
3	Middleware	707
3.1	Motor Control	707
3.1.1	FreeMASTER	707
3.2	MultiCore	744
3.2.1	Multicore SDK	744
3.3	Wireless	840
3.3.1	NXP Wireless Framework and Stacks	840

4	RTOS	1537
4.1	FreeRTOS	1537
4.1.1	FreeRTOS kernel	1537
4.1.2	FreeRTOS drivers	1543
4.1.3	backoffalgorithm	1543
4.1.4	corehttp	1546
4.1.5	corejson	1548
4.1.6	coremqtt	1551
4.1.7	corepkcs11	1554
4.1.8	freertos-plus-tcp	1557
Index		1561

This documentation contains information specific to the kw47loc board.

Chapter 1

KW47-LOC

1.1 Overview

The KW47 LOC is an evaluation kit for KW47 Automotive MCU with 2.4 GHz Bluetooth Low Energy and generic FSK wireless connectivity and dual CAN/LIN connectivity. This localization platform is dedicated to Bluetooth Channel Sounding Ranging solution development with a dedicated on-chip Localization Compute Engine to reduce ranging latency. The KW47's highly sensitive, optimized 2.4 GHz radio features is featured with dual antenna diversity via RF switch which can be bypassed to test via SMA RF connection. The board includes an advanced MCU-Link debug probe, Power and low power option, dual CAN transceiver, buttons, switches, LEDs and a MikroE Click connector.



MCU device and part on board is shown below:

- Device: KW47B42ZB7
- PartNumber: KW47B42ZB7AFTA

1.2 Getting Started with MCUXpresso SDK Package

1.2.1 Getting Started with Package

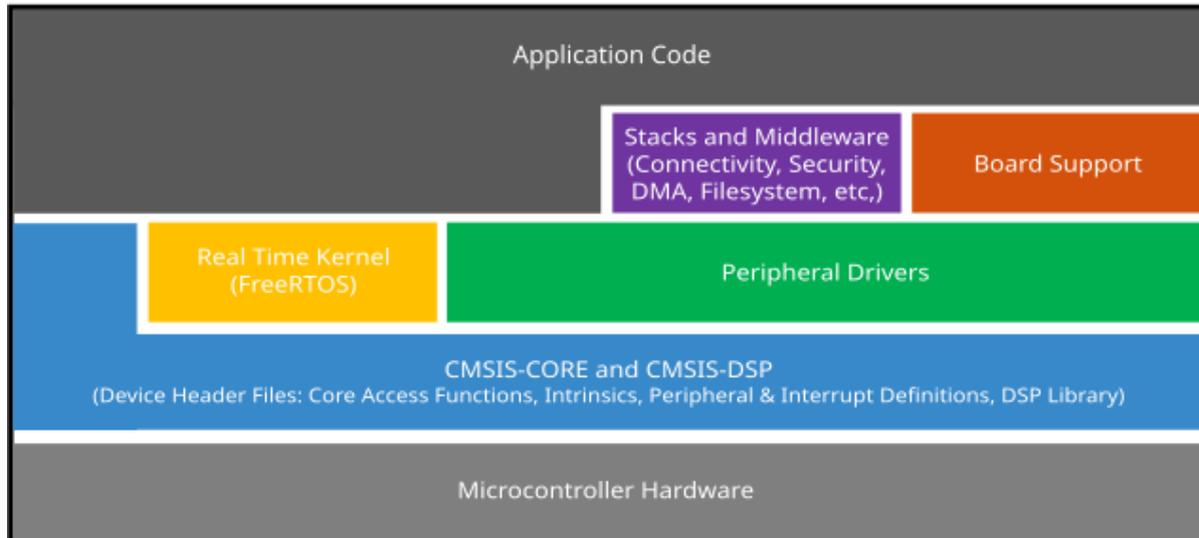
Overview

The NXP MCUXpresso software and tools offer comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on general purpose, crossover, and Bluetooth-enabled MCUs from NXP. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications, which can be used standalone or collaboratively with the A cores running another Operating System (such as Linux OS Kernel). Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to demo applications. The MCUXpresso SDK

also contains optional RTOS integrations such as FreeRTOS and Azure RTOS, device stack, and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for KW47-LOC*.

For the latest version of this and other MCUXpresso SDK documents, see [MCUXpresso Software Development Kit \(SDK\)](#).



MCUXpresso SDK board support package folders

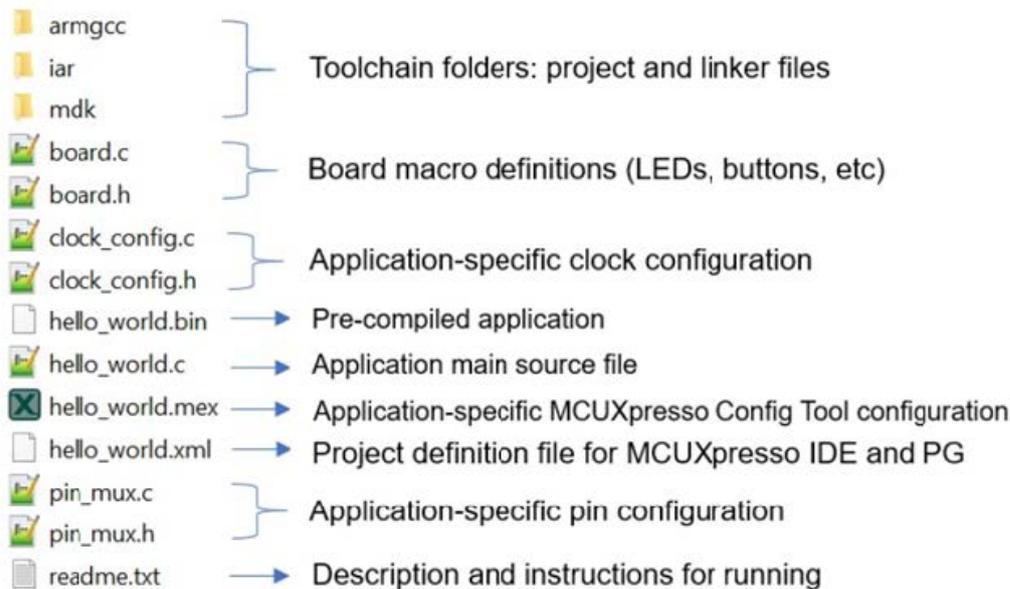
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each `<board_name>` folder, there are various sub-folders to classify the type of examples it contains. These include (but are not limited to):

- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may use stacks and middleware.
- `driver_examples`: Simple applications that show how to use the peripheral drivers of the MCUXpresso SDK for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- `rtos_examples`: Basic FreeRTOS™ OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the RTOS drivers for the MCUXpresso SDK.
- `wireless_examples`: Applications that use the Wireless stacks.

Example application structure This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual* (document MCUXSDKAPIRM).

Each `<board_name>` folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the `hello_world` example (part of the `demo_apps` folder), the same general rules apply to any type of example in the `<board_name>` folder.

In the `hello_world` application folder, you see the following contents:



All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

Note: To prevent compilation errors, do not use special characters in the path of the SDK such as `{!,@,#,$,&,%}^` and space.

Parent topic: [MCUXpresso SDK board support package folders](#)

Locating example application source files When opening an example application in any of the supported IDEs, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- `devices/<device_name>`: CMSIS header file of the device, MCUXpresso SDK feature file, and a few other files
- `devices/<device_name>/drivers`: All the peripheral drivers for your specific MCU
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code, including vector table definitions
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications
- `devices/<device_name>/project_template`: Project template used by MCUXpresso IDE to create projects

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the `rtos` folder. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

Parent topic: [MCUXpresso SDK board support package folders](#)

Running a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the **kw47loc** hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

Build an example application Do the following steps to build the `hello_world` example application..

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/iar
```

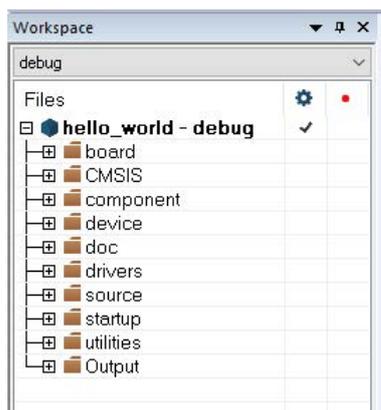
Using the kw47loc hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/kw47loc/demo_apps/hello_world/iar/hello_world.eww
```

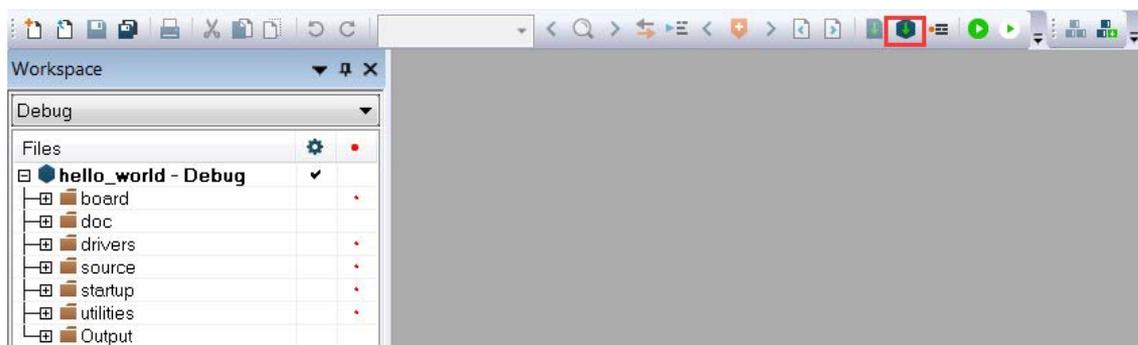
Other example applications may have additional folders in their path.

2. Select the desired build target from the drop-down menu.

For this example, select **hello_world – debug**.



3. To build the demo application, click **Make**, highlighted in red in *Figure 2*.

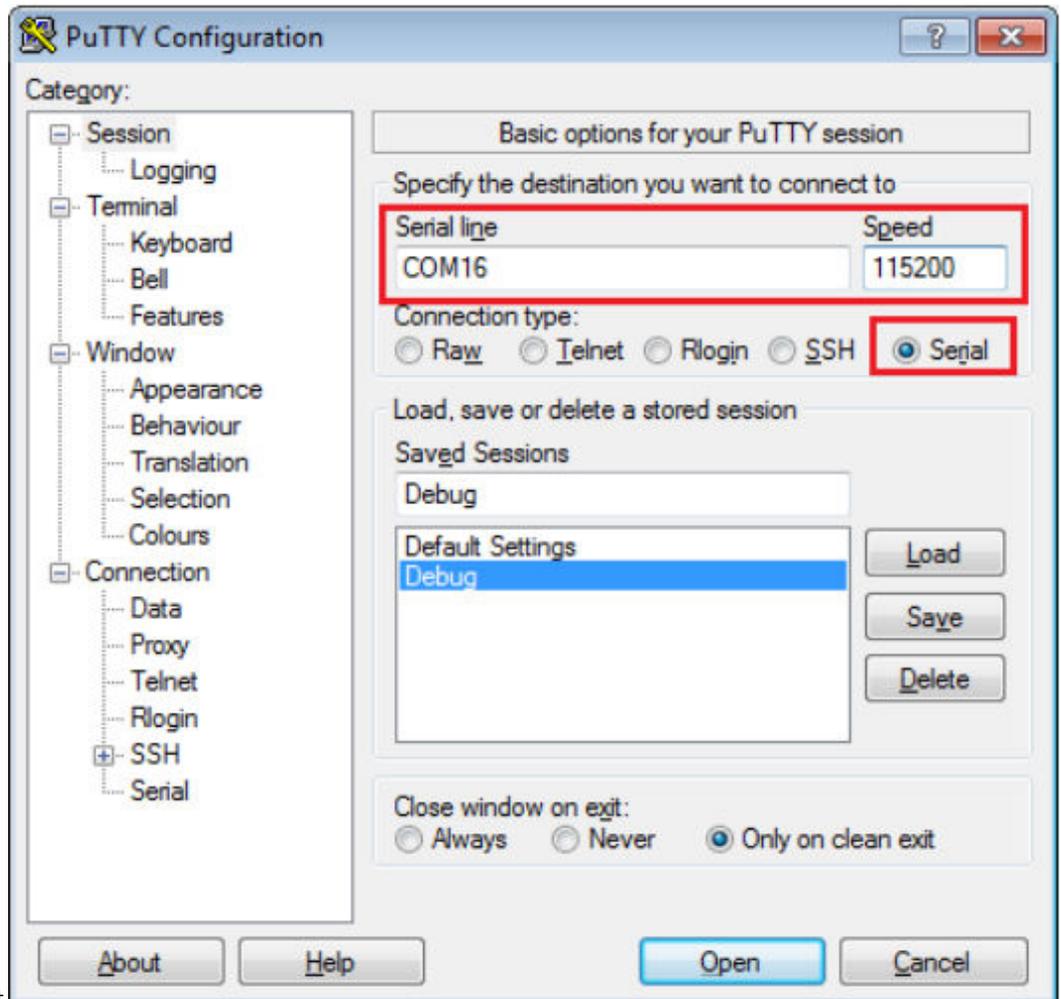


4. The build completes without errors.

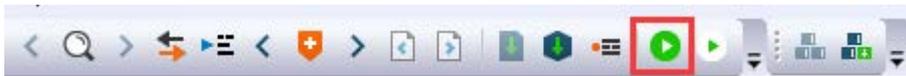
Parent topic: [Running a demo application using IAR](#)

Run an example application To download and run the application, perform these steps:

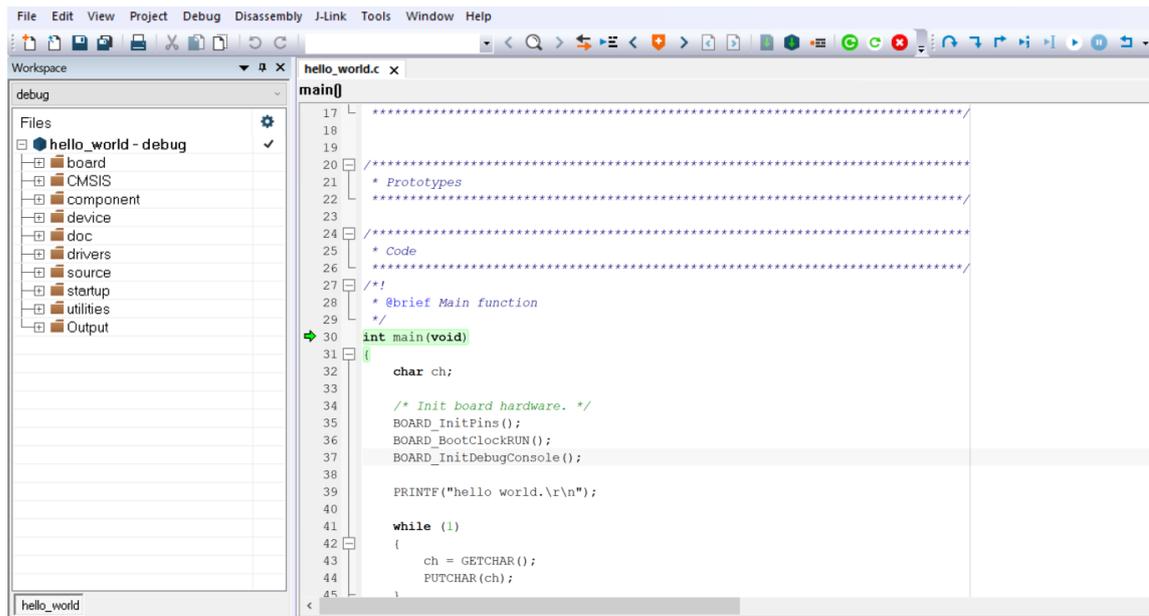
1. Connect the development platform to your PC via USB cable between the USB connector (J14) and the PC USB connector.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port. To determine the COM port number, see [How to determine COM Port](#). Configure the terminal with these settings:
 1. 115200 baud rate, depending on your settings (reference the BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
 2. No parity
 3. 8 data bits



4. 1 stop bit
3. In IAR, click the **Download and Debug** button to download the application to the target.



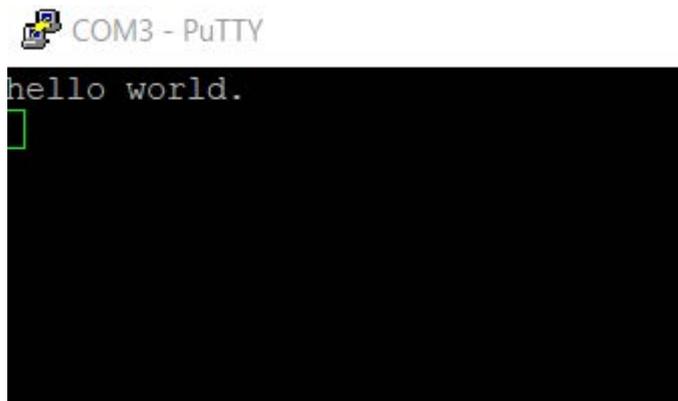
4. The application is then downloaded to the target and automatically runs to the main() function.



5. Run the code by clicking the **Go** button.



6. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



Parent topic:[Running a demo application using IAR](#)

Run a demo using MCUXpresso IDE

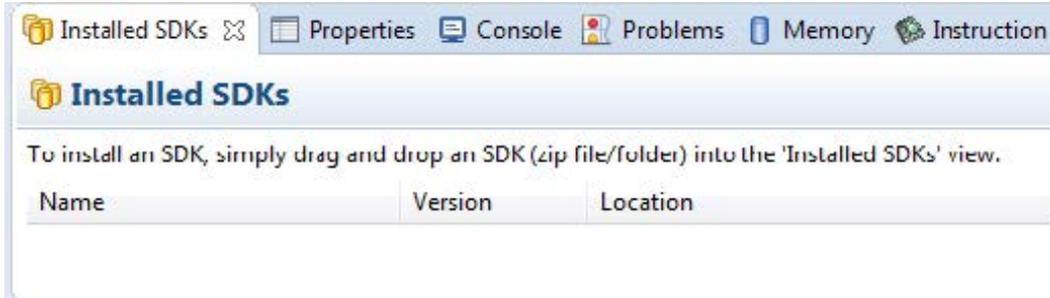
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello_world demo application targeted for the KW47-LOC hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

Selecting the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be located outside of the MCUXpresso SDK tree.

Parent topic:[Run a demo using MCUXpresso IDE](#)

Building an example application To build an example application, follow these steps.

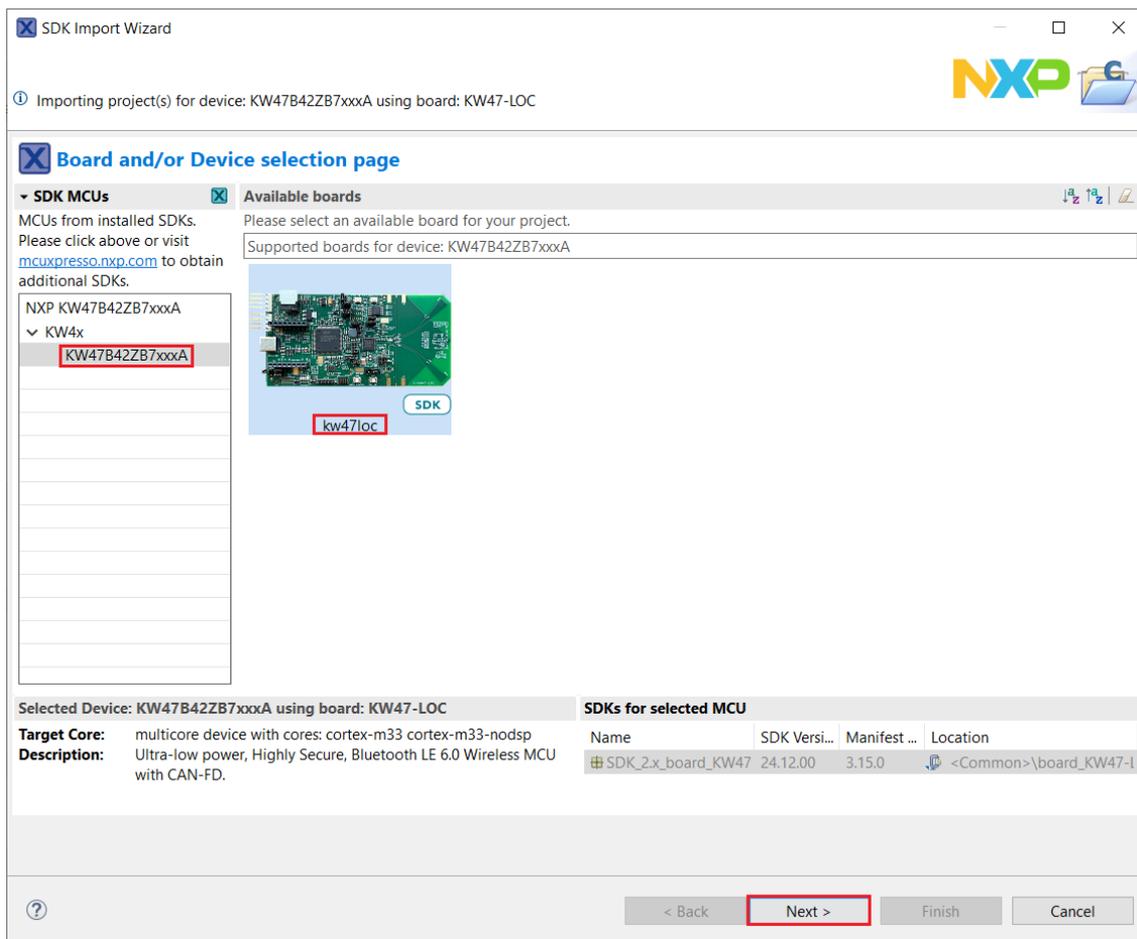
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



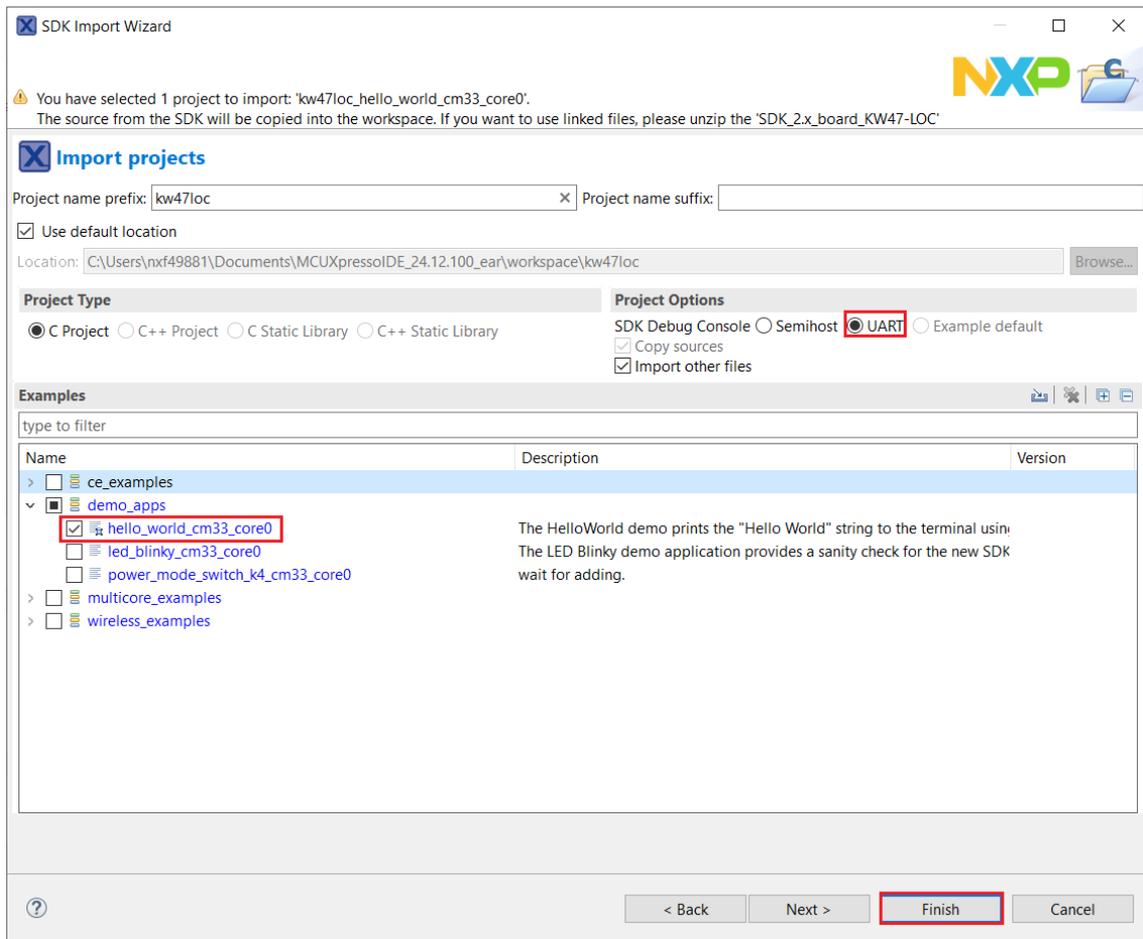
2. On the **Quickstart Panel**, click **Import SDK example(s)...**



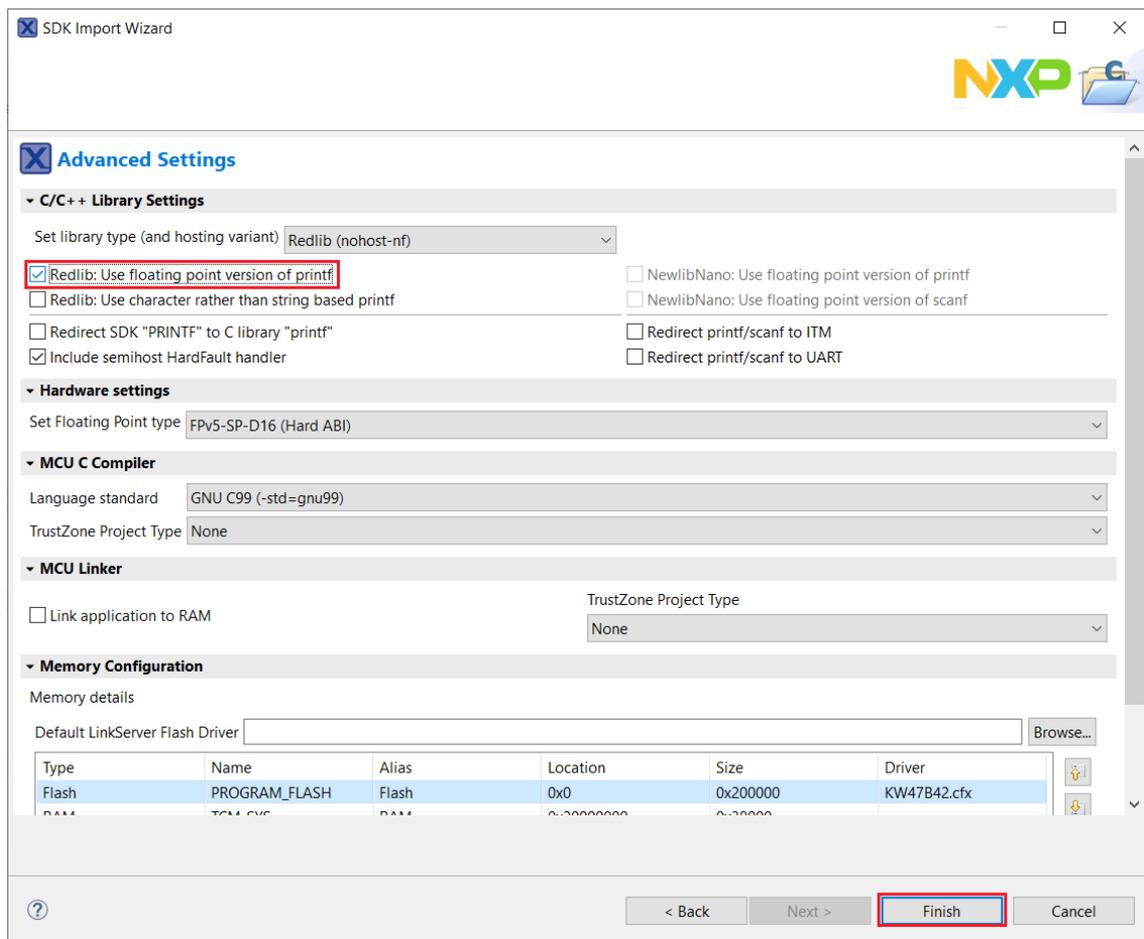
3. In the window that appears, expand the **KW4x** folder and select **KW47**. Then, select **kw47loc** and click **Next**.



4. Expand the demo_apps folder and select hello_world. Then, click **Next**.



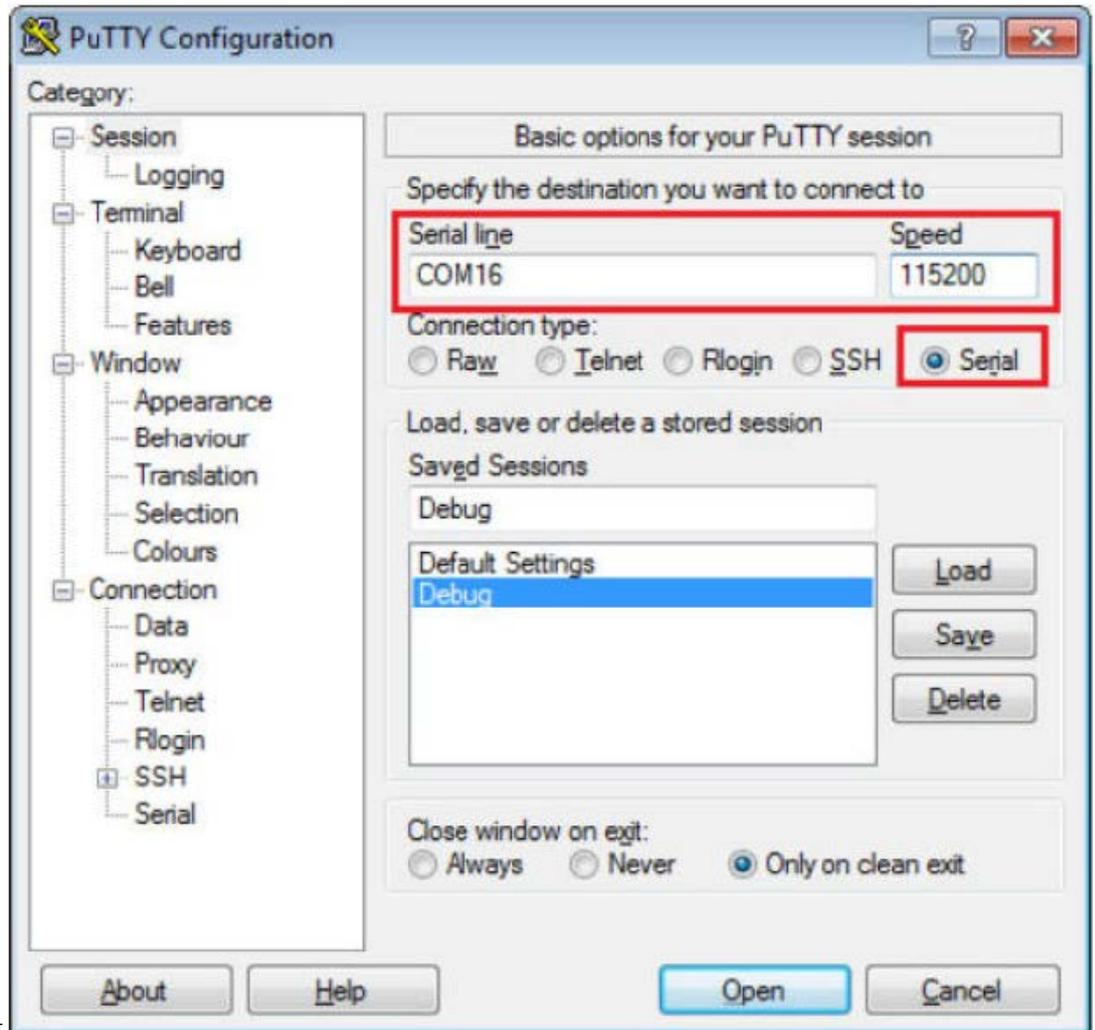
5. Ensure **Redlib: Use floating point version of printf** is selected if the example prints floating point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.



Parent topic: [Run a demo using MCUXpresso IDE](#)

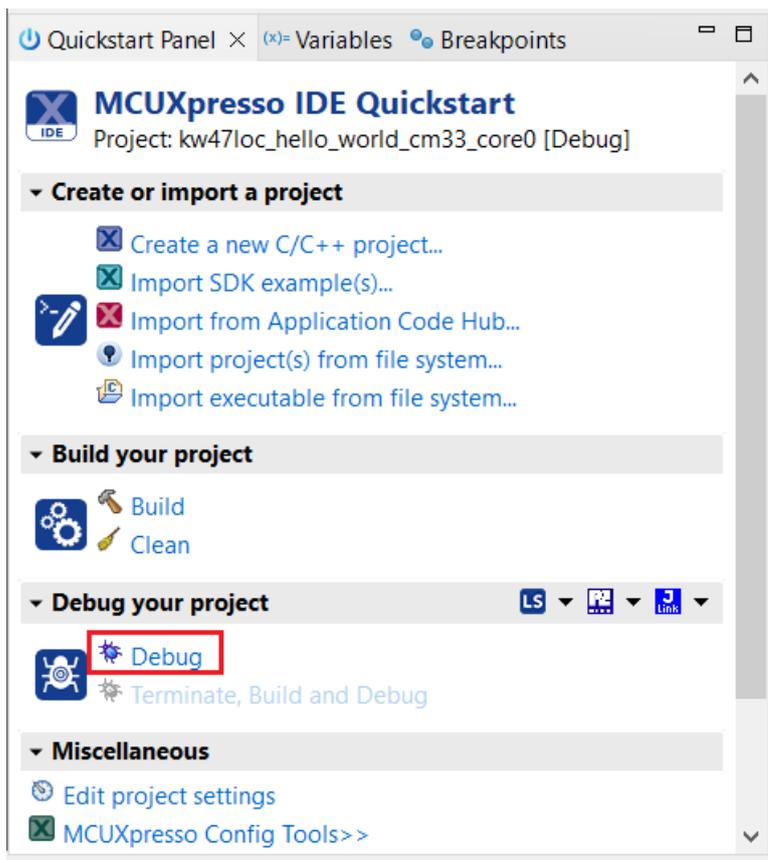
Running an example application To download and run the application, perform the following steps:

1. Connect the development platform to your PC via USB cable between the USB connector (J14) and the PC USB connector.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see [How to determine COM Port](#)). Configure the terminal with these settings:
 1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)
 2. No parity
 3. 8 data bits

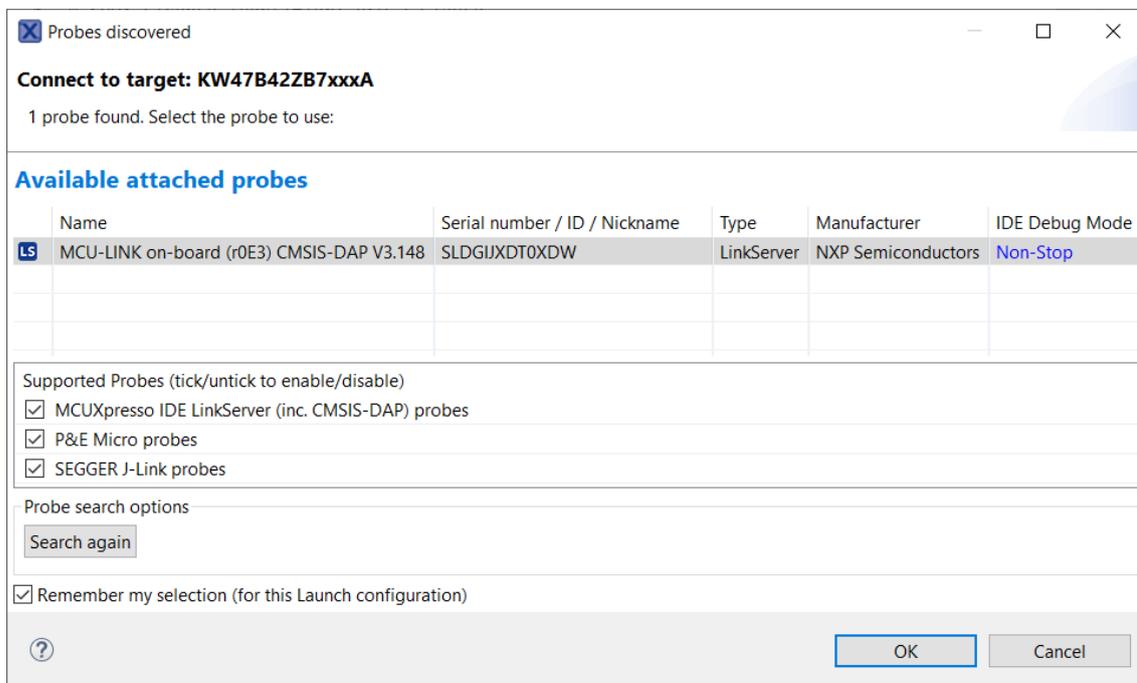


4. 1 stop bit

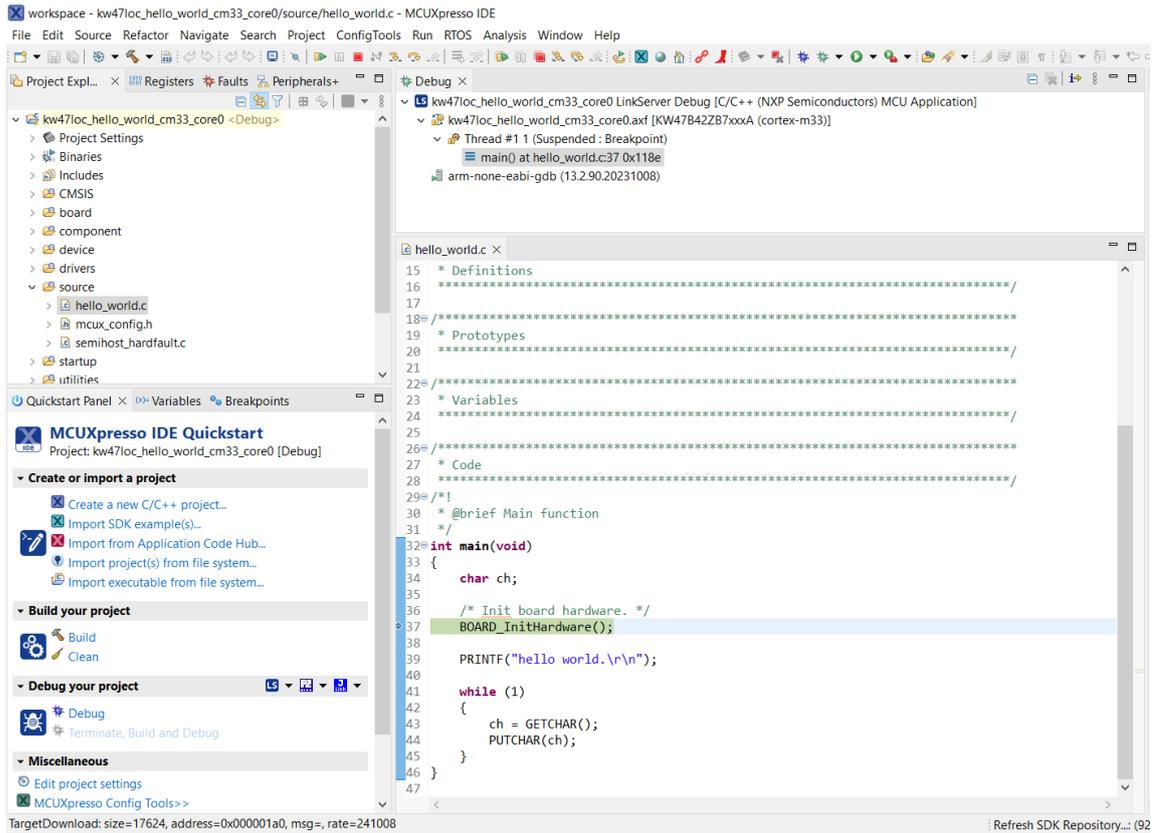
3. On the **Quickstart Panel**, click on **Debug**.



4. The first time you debug a project, the **Debug Emulator Selection** dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)



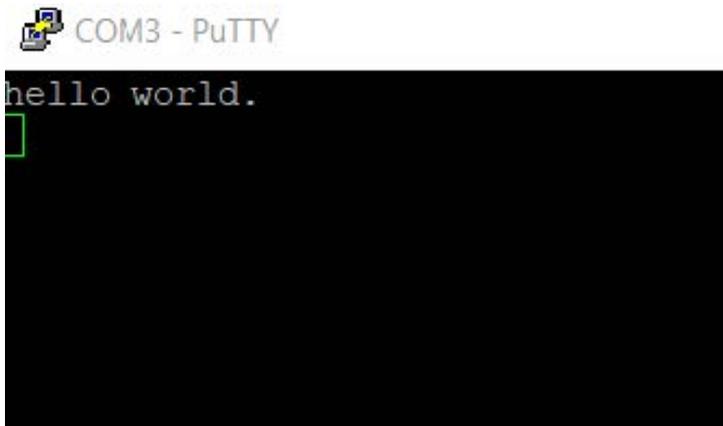
5. The application is downloaded to the target and automatically runs to `main()`.



6. Start the application by clicking **Resume**.



The hello_world application is now running and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.

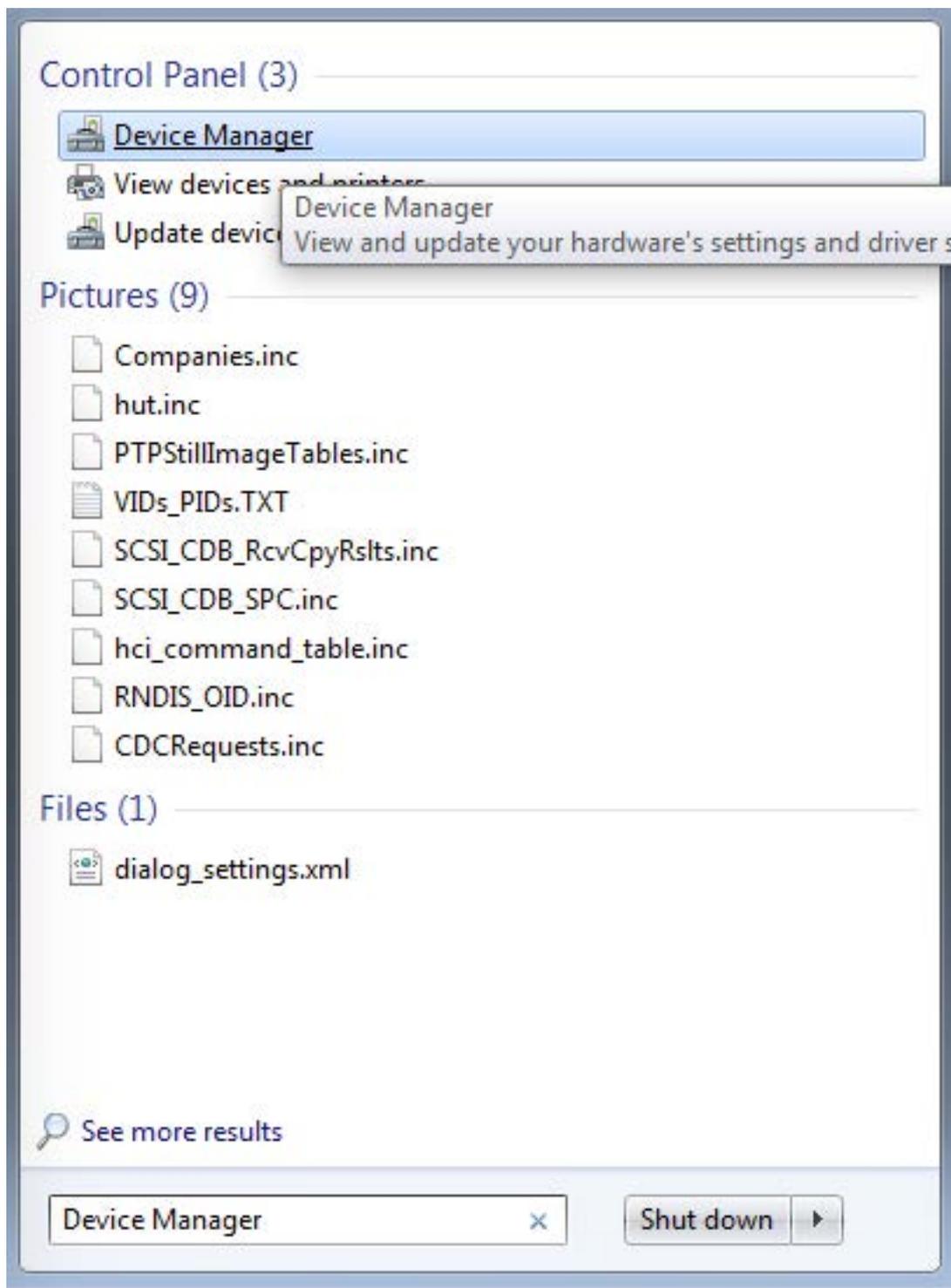


Parent topic: [Run a demo using MCUXpresso IDE](#)

How to determine COM Port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform. All NXP boards ship with a factory programmed, on-board debug interface MCU-LINK.

1. To determine the COM port, open the Windows operating system **Device Manager**. This can be achieved by going to the Windows operating system **Start** menu and typing **Device Manager** in the search bar, as shown in *Figure 1*.



2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports.

How to set the board to the Bootloader ISP mode

On the KW47-LOC board:

1. Press and hold the BOOT CONFIG switch, **SW4**.

2. Connect the KW47-LOC board via the micro-USB connector to the PC.
3. Release the switch **SW4**.
4. Check the associated USB port number on the PC (such as **COM10**).
5. While the KW47 is in bootloader ISP mode, navigate to the folder where *blhost.exe* is located.
6. Type the command `.\blhost.exe -p COM10 -- get-property 1` to make sure that the KW47 is in Bootloader ISP mode, you should see:
 - Ping responded in 1 attempt(s)
 - Inject command 'get-property'
 - Response status = 0 (0x0) Success.
 - Response word 1 = 1258488064 (0x4b030100)
 - Current Version = K3.1.0

On a custom board:

1. Short the `BOOT_CFG` pin to VDD while reset.

Updating debugger firmware

The KW47-LOC board comes with a CMSIS-DAP-compatible debug interface (known as MCU-Link). This firmware in this debug interface may be updated using the host computer scripts. This firmware is typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

NXP provides the MCU-Link utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link. The utility can be downloaded from [MCU-Link Debug Probe](#).

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in MCU-Link user guide, [MCU-Link Debug Probe](#).

1. Install the MCU-Link utility.
2. Unplug the USB cable of the board.
3. Install the jumper on JP12.
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the MCU-Link installation directory, `<MCU-Link install dir>`.
 1. To program CMSIS-DAP debug firmware: `<MCU-Link install dir>/scripts/program_CMSIS`.
 2. To program J-Link debug firmware: `<MCU-Link install dir>/scripts/program_JLINK`.
6. Remove the jumper on JP12.
7. Re-power the board by removing the USB cable and plugging it in again.

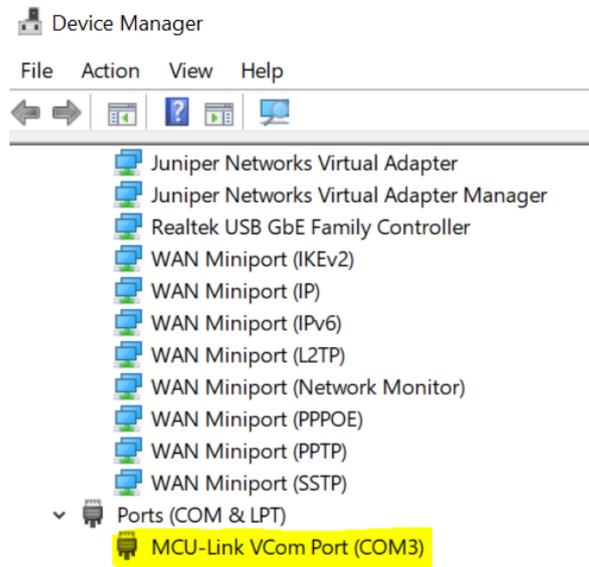
Updating NBU for Wireless Examples

The Narrow Band Unit (NBU) firmware is included in the SDK folder as a signed and encrypted FW.

```
middleware\wireless\ble_controller\bin\kw47_nbu_ble_hosted.bin
```

To program the NBU software for the KW47, perform the following steps:

1. While holding pressed the **SW4** on the KW47-LOC board, attach the USB connector J3 to your computer. Then, release the **SW4** after you plugged the USB cable on your computer.
2. Verify which COM port is assigned to your KW47-LOC board. To check the assigned COM port, in the Windows **Device Manager** program, search for Ports (**COM & LPT**) and save the COM port number. In this example, the assigned COM port is **COM3**.



```

KW47-EVK_NBUProgram.cmd - Notepad
File Edit Format View Help
::: Script to program the .bin file with the NBU image to device.
..\bin\blhost.exe -p COMX get-property 1
ECHO off
ECHO WARNING!!! Verify that the KW47-EVK board has been entered programming mode properly. Press enter to continue.
ECHO on
pause
::: Script uploads .bin file with NBU image to device. based in the SDK v2.16.00
..\bin\blhost.exe -p COMX flash-erase-all 2
..\bin\blhost.exe -p COMX write-memory 0x48800000 kw47_nbu_ble_hosted.bin
pause
    
```

3. Open the *04. Tools* folder that comes together with this document and go to the *script* folder. Locate the *KW47-LOC_NBUProgram.cmd* and open it using any text editor. Replace all the references to **COMX** with the COM Port assigned to your KW47-LOC board. Save the changes applied to the script.

Note: To run the script, place the *blhost.exe* and the *KW47-LOC_NBUProgram.cmd* script in the same folder. Both scripts can be found in the *04. Tools* folder

4. Double click on the *KW47-LOC_NBUProgram.cmd* script to program the NBU and load the FAT software. This script first executes a command to make sure the **KW47** has entered ISP mode properly. If the device did not enter in ISP mode, it cannot be programmed. The following examples show when the device did not enter in ISP (left) and when the device was programmed properly (right).

<pre> Error: Initial ping failure: No response received for ping command. </pre>	<pre> Ping responded in 1 attempt(s) Inject command 'get-property' Response status = 0 (0x0) Success. Response word 1 = 1258488064 (0x4b030100) Current Version = K3.1.0 </pre>
--	---

1.3 Getting Started with MCUXpresso SDK GitHub

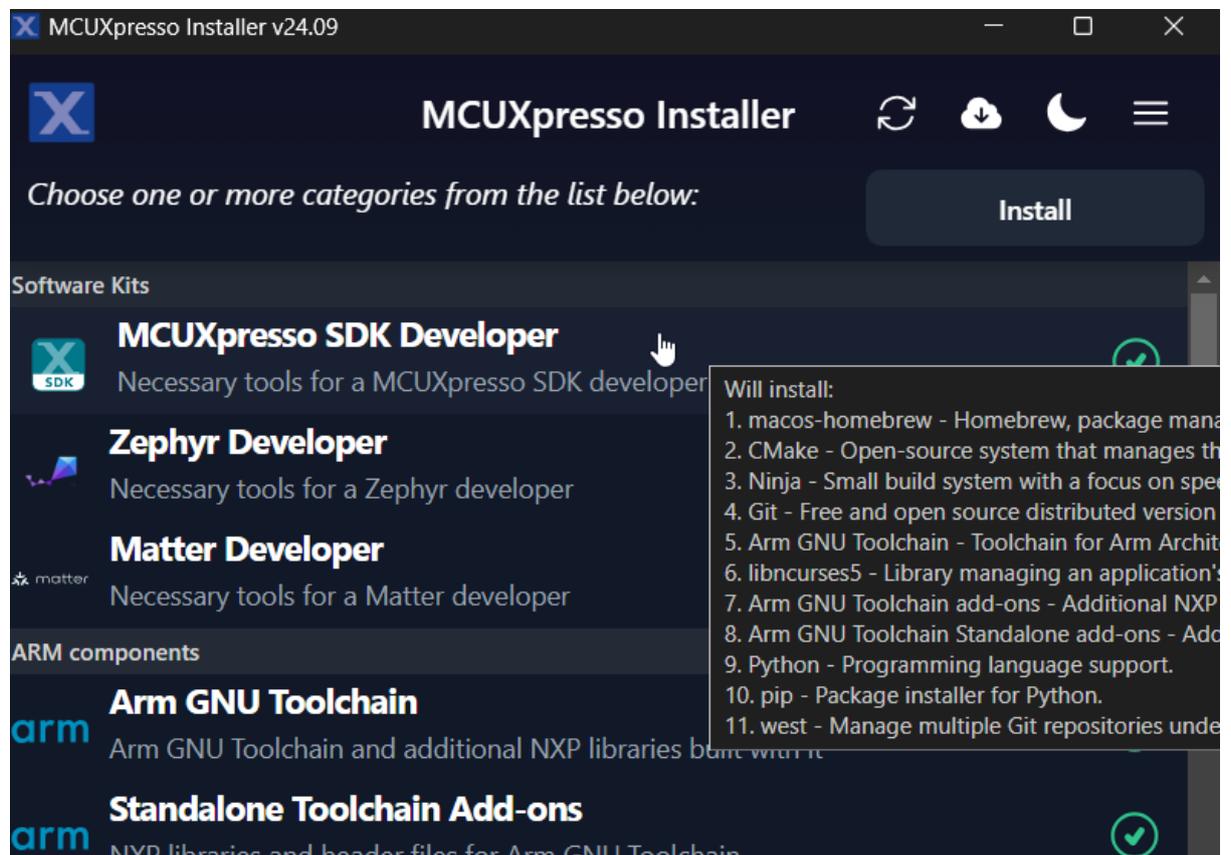
1.3.1 Getting Started with MCUXpresso SDK Repository

Installation

NOTE

If the installation instruction asks/selects whether to have the tool installation path added to the PATH variable, agree/select the choice. This option ensures that the tool can be used in any terminal in any path. *Verify the installation* after each tool installation.

Install Prerequisites with MCUXpresso Installer The MCUXpresso Installer offers a quick and easy way to install the basic tools needed. The MCUXpresso Installer can be obtained from <https://github.com/nxp-mcuxpresso/vscode-for-mcux/wiki/Dependency-Installation>. The MCUXpresso Installer is an automated installation process, simply select MCUXpresso SDK Developer from the menu and click install. If you prefer to install the basic tools manually, refer to the next section.



Alternative: Manual Installation

Basic tools

Git Git is a free and open source distributed version control system. Git is designed to handle everything from small to large projects with speed and efficiency. To install Git, visit the official [Git website](https://git-scm.com/). Download the appropriate version (you may use the latest one) for your operating system (Windows, macOS, Linux). Then run the installer and follow the installation instructions.

User `git --version` to check the version if you have a version installed.

Then configure your username and email using the commands:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

Python Install python 3.10 or latest. Follow the [Python Download](#) guide.

Use `python --version` to check the version if you have a version installed.

West Please use the west version equal or greater than 1.2.0

```
# Note: you can add option '--default-timeout=1000' if you meet connection issue. Or you may set a different ↵
↵source using option '-i'.
# for example, in China you could try: pip install -U west -i https://pypi.tuna.tsinghua.edu.cn/simple
pip install -U west
```

Build And Configuration System

CMake It is strongly recommended to use CMake version equal or later than 3.30.0. You can get latest CMake distributions from [the official CMake download page](#).

For Windows, you can directly use the .msi installer like `cmake-3.31.4-windows-x86_64.msi` to install.

For Linux, CMake can be installed using the system package manager or by getting binaries from [the official CMake download page](#).

After installation, you can use `cmake --version` to check the version.

Ninja Please use the ninja version equal or later than 1.12.1.

By default, Windows comes with the Ninja program. If the default Ninja version is too old, you can directly download the [ninja binary](#) and register the ninja executor location path into your system path variable to work.

For Linux, you can use your [system package manager](#) or you can directly download the [ninja binary](#) to work.

After installation, you can use `ninja --version` to check the version.

Kconfig MCUXpresso SDK uses Kconfig python implementation. We customize it based on our needs and integrate it into our build and configuration system. The Kconfiglib sources are placed under `mcuxsdk/scripts/kconfig` folder.

Please make sure [python](#) environment is setup ready then you can use the Kconfig.

Ruby Our build system supports IDE project generation for iar, mdk, codewarrior and xtensa to provide OOB from build to debug. This feature is implemented with ruby. You can follow the [guide ruby environment setup](#) to setup the ruby environment. Since we provide a built-in portable ruby, it is just a simple one cmd installation.

If you only work with CLI, you can skip this step.

Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

Toolchain	Download and Installation Guide	Note
Armgcc	Arm GNU Toolchain Install Guide	ARMGCC is default toolchain
IAR	IAR Installation and Licensing quick reference guide	
MDK	MDK Installation	
Armclang	Installing Arm Compiler for Embedded	
Zephyr	Zephyr SDK	
Codewarrior	NXP CodeWarrior	
Xtensa	Tensilica Tools	
NXP S32Compiler RISC-V Zen-V	NXP Website	

After you have installed the toolchains, register them in the system environment variables. This will allow the west build to recognize them:

Toolchain	Environment Variable	Example	Cmd Line Argument
Armgcc	ARM-MGCC_DIR	C:\armgcc for windows/usr for Linux. Typically arm-none-eabi-* is installed under /usr/bin	- toolchain armgcc
IAR	IAR_DIR	C:\iar\ewarm-9.60.3 for Windows/opt/iarsystems/bxarm-9.60.3 for Linux	- toolchain iar
MDK	MDK_DIR	C:\Keil_v5 for Windows.MDK IDE is not officially supported with Linux.	- toolchain mdk
Armclang	ARM-CLANG_DIR	C:\ArmCompilerforEmbedded6.22 for Windows/opt/ArmCompilerforEmbedded6.21 for Linux	- toolchain mdk
Zephyr	ZEPHYR_SE	c:\NXP\zephyr-sdk-<version> for windows/opt/zephyr-sdk-<version> for Linux	- toolchain zephyr
CodeWarrior	CW_DIR	C:\Freescale\CW MCU v11.2 for windowsCodeWarrior is not supported with Linux	- toolchain code-warrior
Xtensa	XCC_DIR	C:\xtensa\XtDevTools\install\tools\RI-2023.11-win32\XtensaTools for windows/opt/xtensa/XtDevTools/install/tools/RI-2023.11-Linux/XtensaTools for Linux	- toolchain xtensa
NXP S32Compiler RISC-V Zen-V	RISCV-LVM_DIR	C:\riscv-llvm-win32_b298_b298_2024.08.12 for Windows/opt/riscv-llvm-Linux-x64_b298_b298_2024.08.12 for Linux	- toolchain riscv-llvm

- The <toolchain>_DIR is the root installation folder, not the binary location folder. For IAR, it is directory containing following installation folders:

-  arm
-  common
-  install-info

- MDK IDE using armclang toolchain only officially supports Windows. In Linux, please directly use armclang toolchain by setting ARMCLANG_DIR. In Windows, since most Keil users will install MDK IDE instead of standalone armclang toolchain, the MDK_DIR has higher priority than ARMCLANG_DIR.
- For Xtensa toolchain, please set the XTENSA_CORE environment variable. Here's an example list:

Device Core	XTENSA_CORE
RT500 fusion1	nxp_rt500_RI23_11_newlib
RT600 hifi4	nxp_rt600_RI23_11_newlib
RT700 hifi1	rt700_hifi1_RI23_11_nlib
RT700 hifi4	t700_hifi4_RI23_11_nlib
i.MX8ULP fusion1	fusion_nxp02_dsp_prod

- In Windows, the short path is used in environment variables. If any toolchain is using the long path, you can open a command window from the toolchain folder and use below command to get the short path: for %i in (.) do echo %~fsi

Tool installation check Once installed, open a terminal or command prompt and type the associated command to verify the installation.

If you see the version number, you have successfully installed the tool. Else, check whether the tool's installation path is added into the PATH variable. You can add the installation path to the PATH with the commands below:

- Windows: Open command prompt or powershell, run below command to show the user PATH variable.

```
reg query HKEY_CURRENT_USER\Environment /v PATH
```

The tool installation path should be C:\Users\xxx\AppData\Local\Programs\Git\cmd. If the path is not seen in the output from above, append the path value to the PATH variable with the command below:

```
reg add HKEY_CURRENT_USER\Environment /v PATH /d "%PATH%;C:\Users\xxx\AppData\Local\Programs\Git\cmd"
```

Then close the command prompt or powershell and verify the tool command again.

- Linux:
 1. Open the \$HOME/.bashrc file using a text editor, such as vim.
 2. Go to the end of the file.
 3. Add the line which appends the tool installation path to the PATH variable and export PATH at the end of the file. For example, export PATH="/Directory1:\$PATH".
 4. Save and exit.

5. Execute the script with `source .bashrc` or reboot the system to make the changes live. To verify the changes, run `echo $PATH`.
- macOS:
 1. Open the `$HOME/.bash_profile` file using a text editor, such as `nano`.
 2. Go to the end of the file.
 3. Add the line which appends the tool installation path to the `PATH` variable and export `PATH` at the end of the file. For example, `export PATH="/Directory1:$PATH"`.
 4. Save and exit.
 5. Execute the script with `source .bash_profile` or reboot the system to make the changes live. To verify the changes, run `echo $PATH`.

Get MCUXpresso SDK Repo

Establish SDK Workspace To get the MCUXpresso SDK repository, use the `west` tool to clone the manifest repository and checkout all the west projects.

```
# Initialize west with the manifest repository
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests/ mcuxpresso-sdk

# Update the west projects
cd mcuxpresso-sdk
west update

# Allow the usage of west extensions provided by MCUXpresso SDK
west config commands.allow_extensions true
```

Install Python Dependency(If do tool installation manually) To create a Python virtual environment in the west workspace core repo directory `mcuxsdk`, follow these steps:

1. Navigate to the core directory:

```
cd mcuxsdk
```

2. [Optional] Create and activate the virtual environment: If you don't want to use the python virtual environment, skip this step. **We strongly suggest you use `venv` to avoid conflicts with other projects using python.**

```
python -m venv .venv

# For Linux/MacOS
source .venv/bin/activate

# For Windows
.\.venv\Scripts\activate
# If you are using powershell and see the issue that the activate script cannot be run.
# You may fix the issue by opening the powershell as administrator and run below command:
powershell Set-ExecutionPolicy RemoteSigned
# then run above activate command again.
```

Once activated, your shell will be prefixed with `(.venv)`. The virtual environment can be deactivated at any time by running `deactivate` command.

Remember to activate the virtual environment every time you start working in this directory. If you are using some modern shell like `zsh`, there are some powerful plugins to help you auto switch `venv` among workspaces. For example, `zsh-autoswitch-virtualenv`.

3. Install the required Python packages:

```
# Note: you can add option '--default-timeout=1000' if you meet connection issue. Or you may set a
↳different source using option '-i'.
# for example, in China you could try: pip3 install -r mcuxsdk/scripts/requirements.txt -i https://pypi.
↳tuna.tsinghua.edu.cn/simple
pip install -r scripts/requirements.txt
```

Explore Contents

This section helps you build basic understanding of current fundamental project content and guides you how to build and run the provided example project in whole SDK delivery.

Folder View The whole MCUXpresso SDK project, after you have done the west init and west update operations follow the guideline at [Getting Started Guide](#), have below folder structure:

Folder	Description
mani-fests	Manifest repo, contains the manifest file to initialize and update the west workspace.
mcuxsdk	The MCUXpresso SDK source code, examples, middleware integration and script files.

All the projects record in the [Manifest repo](#) are checked out to the folder mcuxsdk/, the layout of mcuxsdk folder is shown as below:

Folder	Description
arch	Arch related files such as ARM CMSIS core files, RISC-V files and the build files related to the architecture.
cmake	The cmake modules, files which organize the build system.
com-ponents	Software components.
de-vices	Device support package which categorized by device series. For each device, header file, feature file, startup file and linker files are provided, also device specific drivers are included.
docs	Documentation source and build configuration for this sphinx built online documentation.
drivers	Peripheral drivers.
ex-amples	Various demos and examples, support files on different supported boards. For each board support, there are board configuration files.
mid-dle-ware	Middleware components integrated into SDK.
rtos	Rtos components integrated into SDK.
scripts	Script files for the west extension command and build system support.
svd	Svd files for devices, this is optional because of large size. Customers run west manifest config group.filter +optional and west update mcux-soc-svd to get this folder.

Examples Project The examples project is part of the whole SDK delivery, and locates in the folder mcuxsdk/examples of west workspace.

Examples files are placed in folder of <example_category>, these examples include (but are not limited to)

- `demo_apps`: Basic demo set to start using SDK, including `hello_world` and `led_blinky`.
- `driver_examples`: Simple applications that show how to use the peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI transfer using DMA).

Board porting layers are placed in folder of `_boards/<board_name>` which aims at providing the board specific parts for examples code mentioned above.

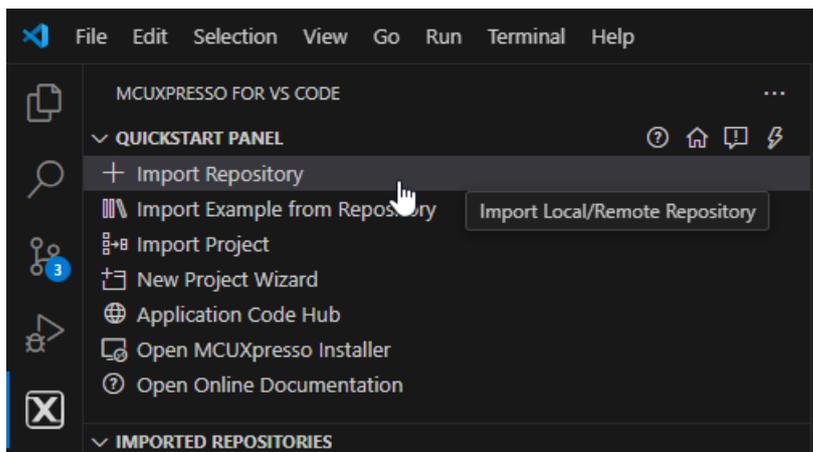
Run a demo using MCUXpresso for VS Code

This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these steps can be applied to any example application in the MCUXpresso SDK.

Build an example application This section assumes that the user has already obtained the SDK as outlined in [Get MCUXpresso SDK Repo](#).

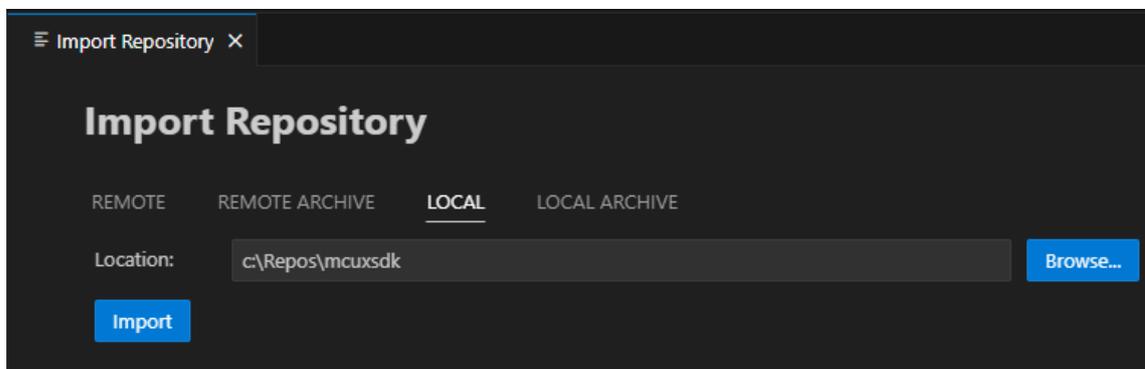
To build an example application:

1. Import the SDK into your workspace. Click **Import Repository** from the **QUICKSTART PANEL**.



Note: You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

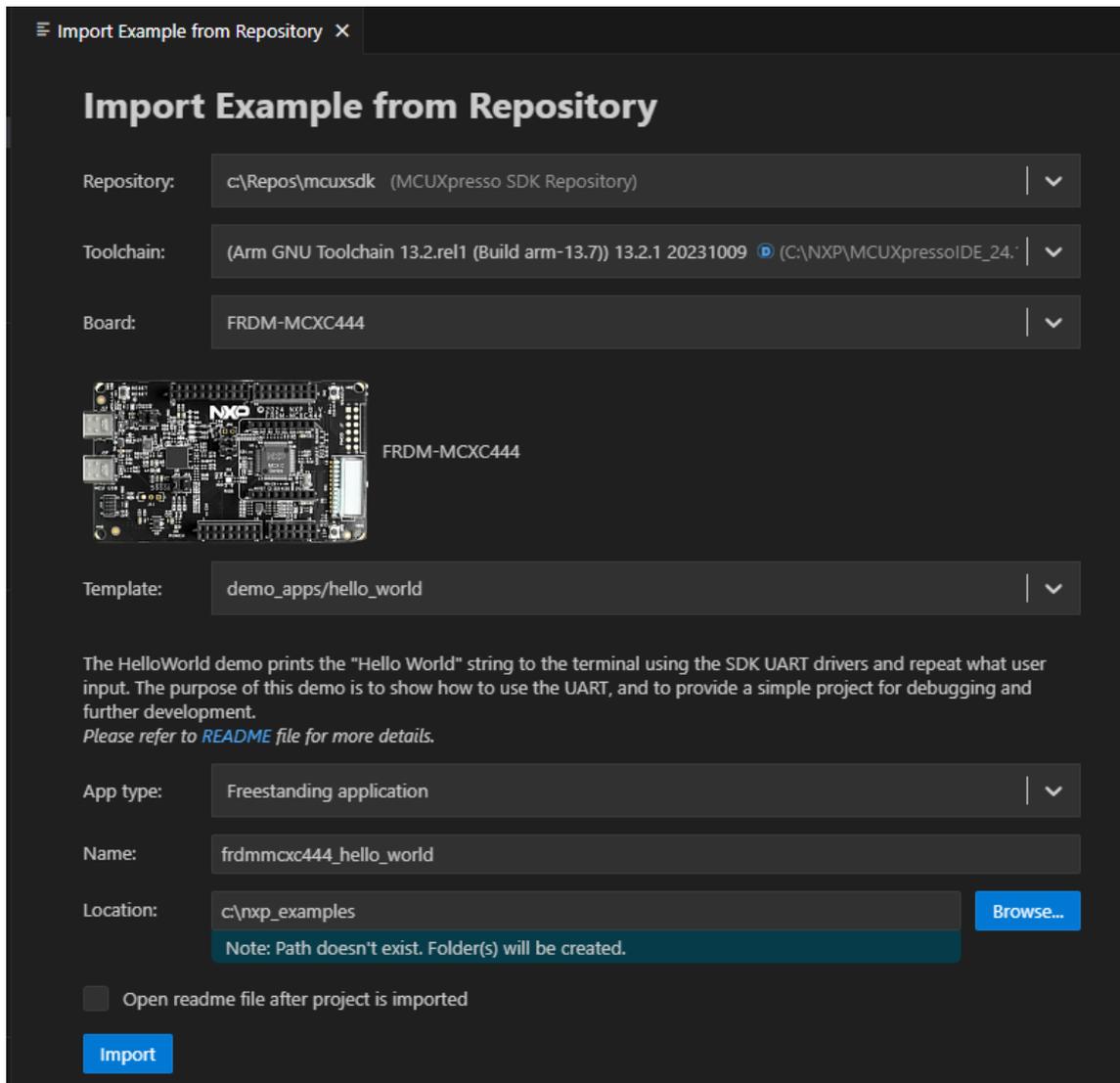
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



2. Click **Import Example from Repository** from the **QUICKSTART PANEL**.

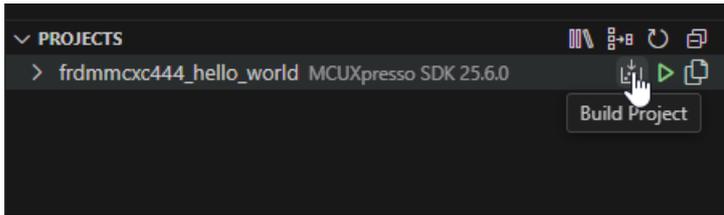


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

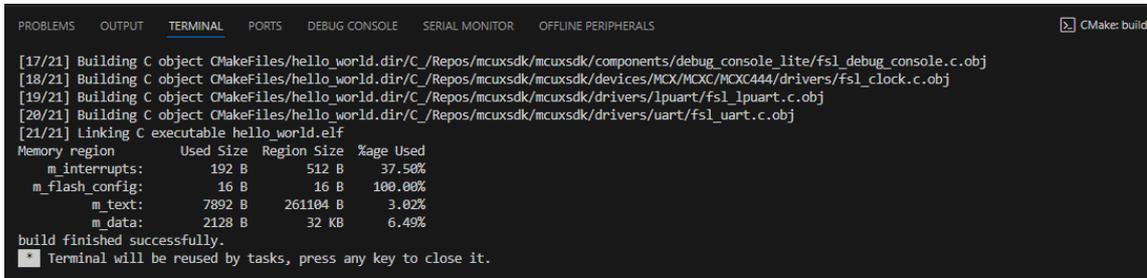


Note: The MCUXpresso SDK projects can be imported as **Repository applications** or **Freestanding applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Freestanding examples can be imported to a user-defined location. Select between these by designating your selection in the **App type** dropdown menu.

3. VS Code will prompt you to confirm if the imported files are trusted. Click **Yes**.
4. Navigate to the **PROJECTS** view. Find your project and click the **Build Project** icon.

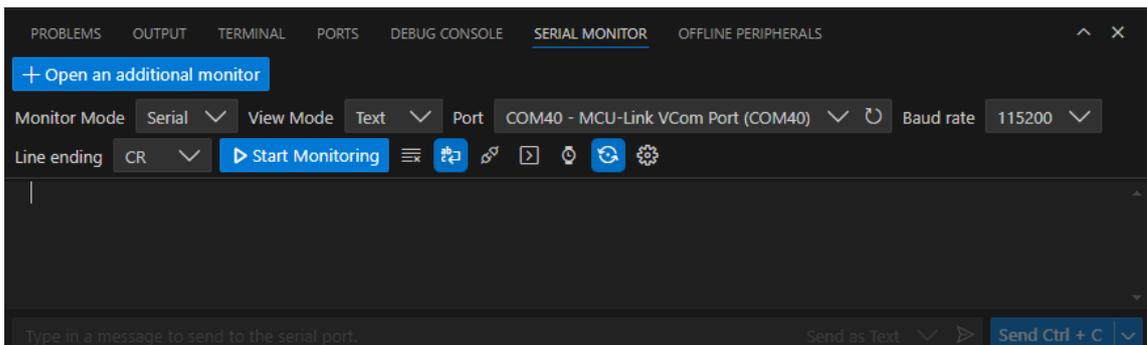


The integrated terminal will open at the bottom and will display the build output.

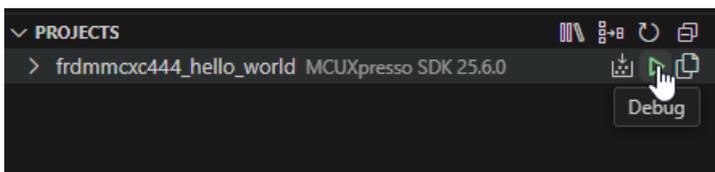


Run an example application **Note:** for full details on MCUXpresso for VS Code debug probe support, see [MCUXpresso for VS Code Wiki](#).

1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



2. Navigate to the **PROJECTS** view and click the play button to initiate a debug session.



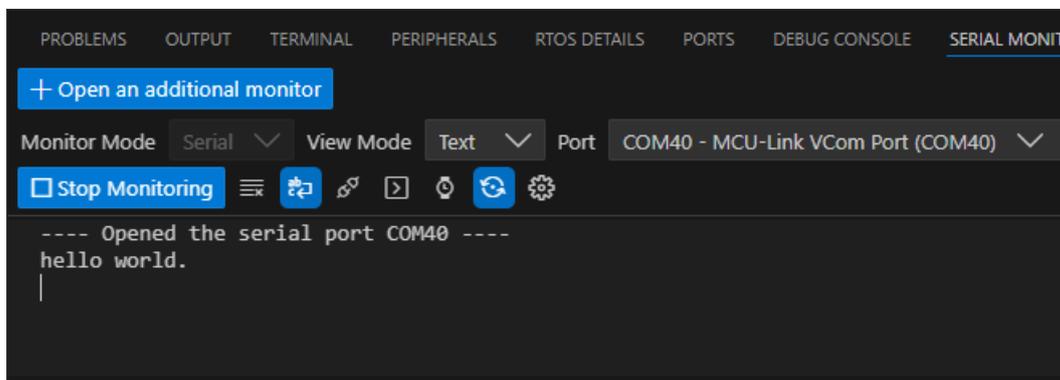
The debug session will begin. The debug controls are initially at the top.

```

18  /*****
21
22  /*****
23  * Variables
24  *****/
25
26  /*****
27  * Code
28  *****/
29  /*!
30  * @brief Main function
31  */
32  int main(void)
33  {
34      char ch;
35
36      /* Init board hardware. */
37      BOARD_InitHardware();
38
39      PRINTF("hello world.\r\n");
40
41      while (1)
42      {
43          ch = GETCHAR();
44          PUTCHAR(ch);
45      }
46  }
47

```

3. Click **Continue** on the debug controls to resume execution of the code. Observe the output on the **Serial Monitor**.



Running a demo using ARMGCC CLI/IAR/MDK

Supported Boards Use the west extension `west list_project` to understand the board support scope for a specified example. All supported build command will be listed in output:

```

west list_project -p examples/demo_apps/hello_world [-t armgcc]

INFO: [ 1][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↳ evk9mimx8ulp -Dcore_id=cm33]
INFO: [ 2][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↳ evkbimxrt1050]
INFO: [ 3][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_

```

(continues on next page)

(continued from previous page)

```

↪ evkbnimxrt1060]
INFO: [ 4][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm4]
INFO: [ 5][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm7]
INFO: [ 6][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1060]
INFO: [ 7][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimx7ulp]
...

```

The supported toolchains and build targets for an example are decided by the example-self example.yml and board example.yml, please refer Example Toolchains and Targets for more details.

Build the project Use `west build -h` to see help information for west build command. Compared to zephyr's west build, MCUXpresso SDK's west build command provides following additional options for mcux examples:

- `--toolchain`: specify the toolchain for this build, default `armgcc`.
- `--config`: value for `CMAKE_BUILD_TYPE`. If not provided, build system will get all the example supported build targets and use the first debug target as the default one. Please refer Example Toolchains and Targets for more details about example supported build targets.

Here are some typical usages for generating a SDK example:

```

# Generate example with default settings, default used device is the mainset MK22F51212
west build -b frdmk22f examples/demo_apps/hello_world

# Just print cmake commands, do not execute it
west build -b frdmk22f examples/demo_apps/hello_world --dry-run

# Generate example with other toolchain like iar, default armgcc
west build -b frdmk22f examples/demo_apps/hello_world --toolchain iar

# Generate example with other config type
west build -b frdmk22f examples/demo_apps/hello_world --config release

# Generate example with other devices with --device
west build -b frdmk22f examples/demo_apps/hello_world --device MK22F12810 --config release

```

For multicore devices, you shall specify the corresponding core id by passing the command line argument `-Dcore_id`. For example

```

west build -b evkbnimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config_
↪ flexspi_nor_debug

```

For shield, please use the `--shield` to specify the shield to run, like

```

west build -b mimxrt700evk --shield a8974 examples/issdk_examples/sensors/fxls8974cf/fxls8974cf_poll -
↪ Dcore_id=cm33_core0

```

Sysbuild(System build) To support multicore project building, we ported Sysbuild from Zephyr. It supports combine multiple projects for compilation. You can build all projects by adding `--sysbuild` for main application. For example:

```

west build -b evkbnimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪ id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always

```

For more details, please refer to System build.

Config a Project Example in MCUXpresso SDK is configured and tested with pre-defined configuration. You can follow steps blow to change the configuration.

1. Run cmake configuration

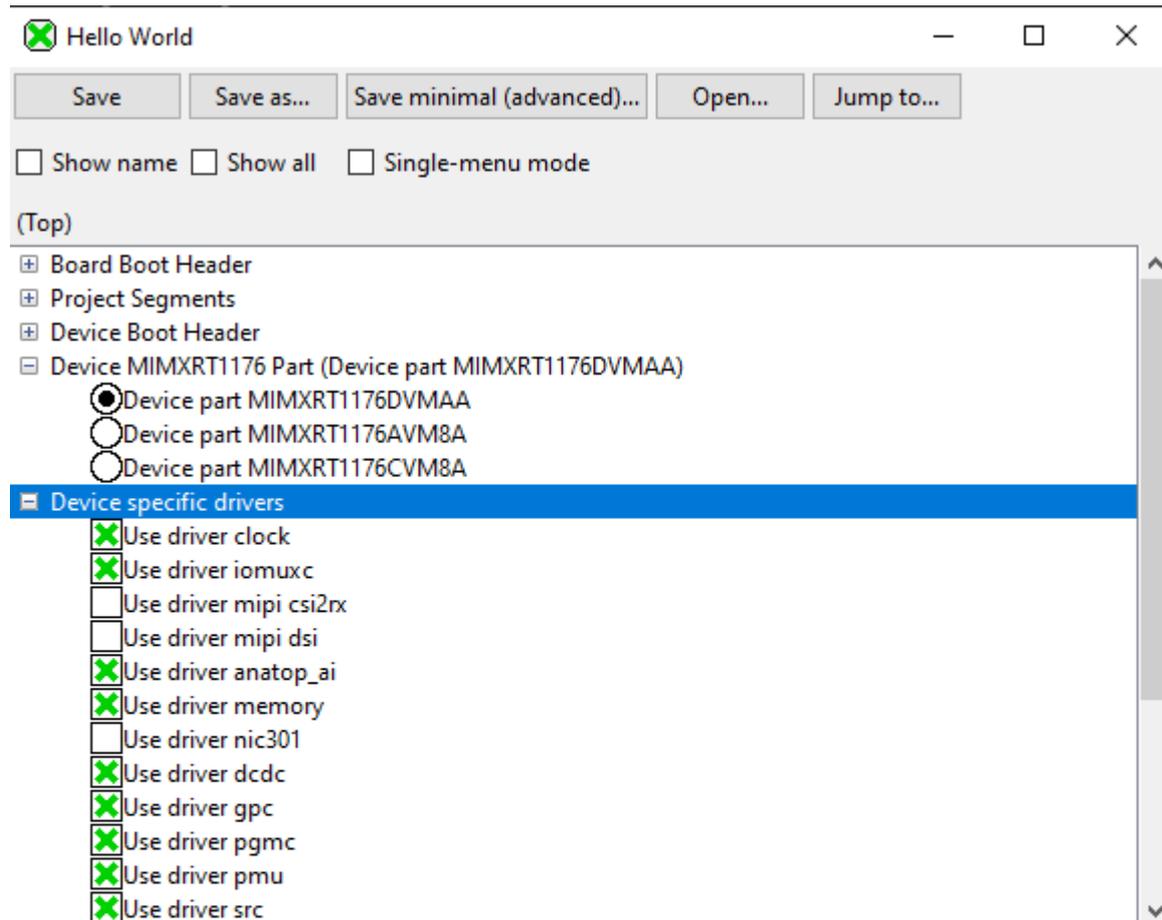
```
west build -b evkbnimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

Please note the project will be built without --cmake-only parameter.

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like



```
Kconfig definition, with parent deps. propagated to 'depends on'
=====
At D:/sdk_next/mcuxsdk\devices\../devices/RT/RT1170/MIMXRT1176\drivers/Kconfig: 5
Included via D:/sdk_next/mcuxsdk/examples/demo_apps/hello_world/Kconfig: 6 ->
D:/sdk_next/mcuxsdk/Kconfig.mcuxpresso: 9 -> D:/sdk_next/mcuxsdk\devices/Kconfig: 1
-> D:/sdk_next/mcuxsdk\devices\../devices/RT/RT1170/MIMXRT1176/Kconfig: 8
Menu path: (Top)

menu "Device specific drivers"
```

You can reconfigure the project by selecting/deselecting Kconfig options.

After saving and closing the Kconfig GUI, you can directly run `west build` to build with the new configuration.

Flash *Note:* Please refer Flash and Debug The Example to enable west flash/debug support.

Flash the hello_world example:

```
west flash -r linkserver
```

Debug Start a gdb interface by following command:

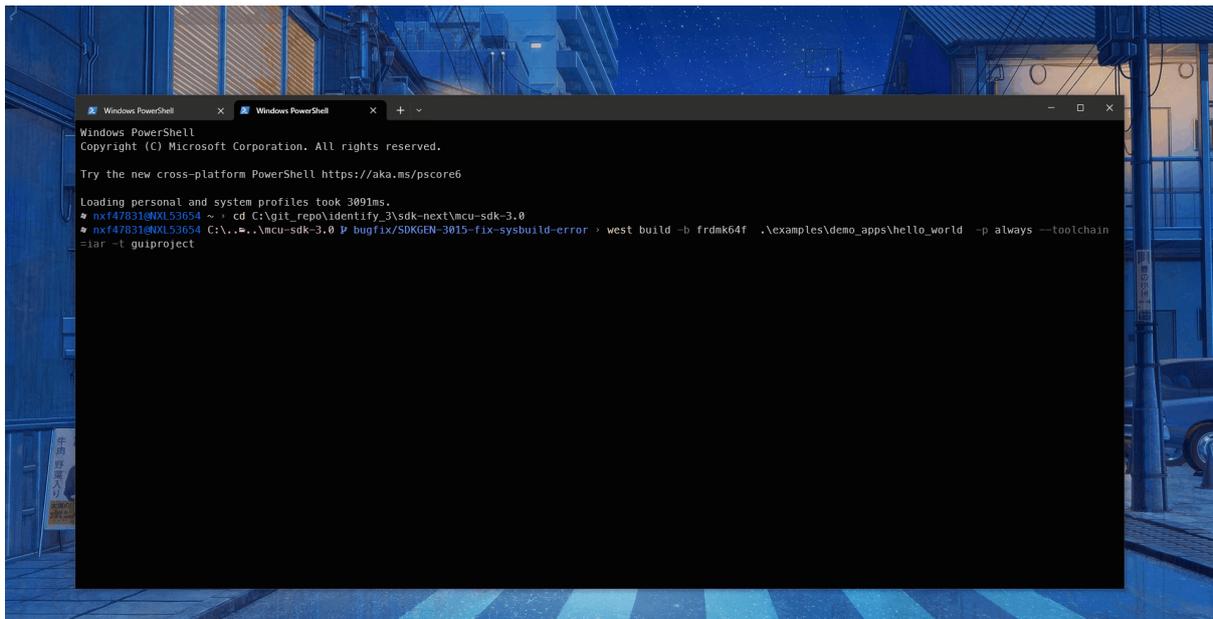
```
west debug -r linkserver
```

Work with IDE Project The above build functionalities are all with CLI. If you want to use the toolchain IDE to work to enjoy the better user experience especially for debugging or you are already used to develop with IDEs like IAR, MDK, Xtensa and CodeWarrior in the embedded world, you can play with our IDE project generation functionality.

This is the cmd to generate the evkbmimxrt1170 hello_world IAR IDE project files.

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config_
↵ flexspi_nor_debug -p always -t guiproject
```

By default, the IDE project files are generated in mcuxsdk/build/<toolchain> folder, you can open the project file with the IDE tool to work:



Note, please follow the [Installation](#) to setup the environment especially make sure that *ruby* has been installed.

1.4 Release Notes

This is an early adopter release provided as preview for development with pre-production devices.

1.4.1 MCUXpresso SDK Release Notes

Overview

The MCUXpresso SDK is a comprehensive software enablement package designed to simplify and accelerate application development with Arm Cortex-M-based devices from NXP, including its general purpose, crossover and Bluetooth-enabled MCUs. MCUXpresso SW and Tools for DSC further extends the SDK support to current 32-bit Digital Signal Controllers. The MCUXpresso SDK includes production-grade software with integrated RTOS (optional), integrated enabling software technologies (stacks and middleware), reference software, and more.

In addition to working seamlessly with the MCUXpresso IDE, the MCUXpresso SDK also supports and provides example projects for various toolchains. The Development tools chapter in the associated Release Notes provides details about toolchain support for your board. Support for the MCUXpresso Config Tools allows easy cloning of existing SDK examples and demos, allowing users to leverage the existing software examples provided by the SDK for their own projects.

Underscoring our commitment to high quality, the MCUXpresso SDK is MISRA compliant and checked with Coverity static analysis tools. For details on MCUXpresso SDK, see [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#).

MCUXpresso SDK

As part of the MCUXpresso software and tools, MCUXpresso SDK is the evolution of Kinetis SDK, includes support for LPC, DSC, PN76, and i.MX System-on-Chip (SoC). The same drivers, APIs, and middleware are still available with support for Kinetis, LPC, DSC, and i.MX silicon. The MCUXpresso SDK adds support for the MCUXpresso IDE, an Eclipse-based toolchain that works with all MCUXpresso SDKs. Easily import your SDK into the new toolchain to access to all of the available components, examples, and demos for your target silicon. In addition to the MCUXpresso IDE, support for the MCUXpresso Config Tools allows easy cloning of existing SDK examples and demos, allowing users to leverage the existing software examples provided by the SDK for their own projects.

In order to maintain compatibility with legacy Freescale code, the filenames and source code in MCUXpresso SDK containing the legacy Freescale prefix FSL has been left as is. The FSL prefix has been redefined as the NXP Foundation Software Library.

Development tools

The MCUXpresso SDK was tested with following development tools. Same versions or above are recommended.

- IAR Embedded Workbench for Arm, version is 9.60.4
- MCUXpresso IDE, Rev. 25.06.xx
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

This release supports board and devices listed in following table. The board and devices in bold were tested in this release.

Development boards	MCU devices
KW47-LOC	KW47B42ZB7AFTA

MCUXpresso SDK release package

The MCUXpresso SDK release package content is aligned with the silicon subfamily it supports. This includes the boards, CMSIS, devices, middleware, and RTOS support.

Device support The device folder contains the whole software enablement available for the specific System-on-Chip (SoC) subfamily. This folder includes clock-specific implementation, device register header files, device register feature header files, and the system configuration source files. Included with the standard SoC support are folders containing peripheral drivers, toolchain support, and a standard debug console. The device-specific header files provide a direct access to the microcontroller peripheral registers. The device header file provides an overall SoC memory mapped register definition. The folder also includes the feature header file for each peripheral on the microcontroller. The toolchain folder contains the startup code and linker files for each supported toolchain. The startup code efficiently transfers the code execution to the main() function.

Board support The boards folder provides the board-specific demo applications, driver examples, and middleware examples.

Demo application and other examples The demo applications demonstrate the usage of the peripheral drivers to achieve a system level solution. Each demo application contains a readme file that describes the operation of the demo and required setup steps. The driver examples demonstrate the capabilities of the peripheral drivers. Each example implements a common use case to help demonstrate the driver functionality.

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Middleware

GenFSK link layer The Generic FSK protocol enables radio operation using a custom GFSK/GMSK or MSK modulation format.

Main Features supported:

- Highly configurable packet structure
- Optimized Sequence Command Set
- High-precision timebase to maintain network timing
- Two timer-compare mechanisms for Interrupt Generation and Sequence Launching
- Hardware automation for packet transmit and receive, CRC and Whitening
- Up to four network addresses to synchronize to, can be 8-bit, 16-bit or 32-bit
- Packet Lengths up to 2047 Bytes
- Support complex auto-sequence, like CCA before TX, Auto-ACK, TR.
- Many operating modes can support the sending and receiving of multiple protocol packets, such as Bluetooth LE.

Wireless XCVR The XCVR component provides a base Transceiver Driver for the 2.4 GHz narrowband radio.

Bluetooth LE Controller

- Main features supported:
 - Peripheral Role
 - Central Role
 - Multiple PHYs (1 Mbps, 2 Mbps, Coded PHY)
 - Asymmetric Connections
 - Public/Random/Static Addresses
 - Network/Device Privacy Modes
 - Extended Advertising
 - Extended Scanning
 - Passive/Active Scanning
 - LE Encryption
 - LE Ping Procedure
 - HCI Test Interface
 - UART Test Interface
 - Randomized Advertising Channel Indexing
 - Sleep Clock Accuracy Update - Mechanism
 - ADI Field in Scan Response Data
 - HCI Support for Debug Keys in LE - Secure Connections
- Main capabilities supported:
 - Simultaneous scanning 1 Mbps and Long Range
 - Scanning and advertising in parallel
 - 24 connections as a central role
 - 24 connections as a peripheral role
 - Any combination of central and peripheral roles (24 connections maximum)
 - 8 connections with a 7.5 ms connection interval
 - Two advertising sets in parallel (Four adv set as Early Access Release).
 - 26 Accept List entries
 - 36 Resolvable Private Address (RPA) entries
 - Up to two Chain Packets per Extended Advertising set
 - Enhanced Notification on end of - Scanning/Advertising/Connection events
 - Connection event counters associated to Bluetooth LE packet reception
 - Timestamp associated to Bluetooth LE packet reception
 - RF channel info associated to Bluetooth LE packet reception
 - NXP proprietary Bluetooth LE Handover feature
 - Decision Based Advertising Filtering (DBAF)
 - Advertising Coding Selection (ACS)
 - Periodic Advertising with Responses (PAWR) Additional features supported for KW47 and MCX W72 devices:

- Channel Sounding Additional features supported as EAR (Early Access Release) in the experimental builds:
- Unified Test Protocol (UTP).
- LL Extended Feature Set.
- Monitoring Advertisers.

Note: Project configuration enabling Experimental features on KW45 and MCX W71 requires the Radio Subsystem (NBU) Firmware to be reprogrammed with the firmware provided in the SDK under `middleware\wireless\ble_controller\bin\experimental\`. For NBU programming steps, see the *EVK Quick Start Guide* and Secure Provisioning SDK (SPSDK) documentation.

Project configurations that require usage of the Bluetooth LE controller including all Bluetooth LE examples require the Radio Subsystem (NBU) Firmware to be re-programmed with the firmware provided in the SDK under `middleware\wireless\ble_controller\bin`.

Wireless Bluetooth LE host stack and applications The Bluetooth LE Host Stack component provides an implementation for a Bluetooth LE mandatory and some optional, proprietary, and experimental features. The Bluetooth LE Host Stack component provides application examples, services, and profiles.

Main features supported:

- Automotive Compliance
- MISRA Compliance
- HIS CCM <= 20
- Advanced Secure Mode
- Enhanced ATT
- GATT Caching
- Bluetooth LE Host GCC Libraries
- Bluetooth LE Host IAR Libraries
- Bluetooth LE Host Peripheral Libraries
- Bluetooth LE Central Libraries
- Bluetooth LE Host Full Host Features Libraries
- Bluetooth LE Host Optional Features Libraries
- Bluetooth LE Host Mandatory Features Libraries
- Bare-metal and FreeRTOS Support
- Bluetooth LE Privacy Support
- CCC Sample Applications
- Enhanced Notifications
- Dynamic Database
- OTA Support - Sample Applications
- Decision based Advertising Filtering (DBAF) - Experimental feature
- Advertising Coding Selection (ACS) - Experimental feature
- Channel Sounding - Experimental feature with controlled access (contact your NXP representative for access)

- Bluetooth LE Controller main and experimental features and capabilities described below are supported by the Bluetooth LE Host.

Note: For evaluating DBAF and ACS experimental features, replace the Bluetooth LE Host default example projects libraries with the libraries from the SDK folder `..\middleware\wireless\bluetooth\host\lib_exp` and enable the features in the application. The Radio Subsystem (NBU) Firmware with experimental features is required.

Wireless Localization Localization

Wireless Connectivity Framework The Connectivity Framework is a software component that provides hardware abstraction modules to the upper layer connectivity stacks and components. It also provides a list of services and APIs, such as, Low power, Over the Air (OTA) Firmware update, File System, Security, Sensors, Serial Connectivity Interface (FSCI), and others. The Connectivity Framework modules are located in the `middleware\wireless\framework` SDK folder.

CMSIS DSP Library The MCUXpresso SDK is shipped with the standard CMSIS development pack, including the prebuilt libraries.

NXP PSA CRYPTO DRIVER PSA crypto driver for crypto library integration via driver wrappers

EdgeLock SE050 Plug and Trust Middleware Secure subsystem library - SSS APIs

Multicore Multicore Software Development Kit

mbedTLS mbedtls SSL/TLS library v3.x

mbedTLS mbedtls SSL/TLS library v2.x

LittleFS LittleFS filesystem stack

LIN Stack LIN Stack middleware

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

Release contents

Provides an overview of the MCUXpresso SDK release package contents and locations.

Deliverable	Location
Boards	INSTALL_DIR/boards
Demo Applications	INSTALL_DIR/boards/<board_name>/demo_apps
Driver Examples	INSTALL_DIR/boards/<board_name>/driver_examples
eIQ examples	INSTALL_DIR/boards/<board_name>/eiq_examples
Board Project Template for MCUXpresso IDE NPW	INSTALL_DIR/boards/<board_name>/project_template
Driver, SoC header files, extension header files and feature header files, utilities	INSTALL_DIR/devices/<device_name>
CMSIS drivers	INSTALL_DIR/devices/<device_name>/cmsis_drivers
Peripheral drivers	INSTALL_DIR/devices/<device_name>/drivers
Toolchain linker files and startup code	INSTALL_DIR/devices/<device_name>/<toolchain_name>
Utilities such as debug console	INSTALL_DIR/devices/<device_name>/utilities
Device Project Template for MCUXpresso IDE NPW	INSTALL_DIR/devices/<device_name>/project_template
CMSIS Arm Cortex-M header files, DSP library source	INSTALL_DIR/CMSIS
Components and board device drivers	INSTALL_DIR/components
RTOS	INSTALL_DIR/rtos
Release Notes, Getting Started Document and other documents	INSTALL_DIR/docs
Tools such as shared cmake files	INSTALL_DIR/tools
Middleware	INSTALL_DIR/middleware

What is new

The following changes have been implemented compared to the previous SDK release version (25.09.00).

- **Bluetooth LE Host Stack and Applications**

Added

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

Improved

- Miscellaneous application updates.

Fixed

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

Changed

- Merged **Gap_SetPeriodicAdvParameters** and **Gap_SetPeriodicAdvParametersV2** into **Gap_SetPeriodicAdvParameters**.
- Details can be found in github repository [nxp-mcuxpresso/mcuxsdk-middleware-bluetooth-host/CHANGELOG.md](https://github.com/nxp-mcuxpresso/mcuxsdk-middleware-bluetooth-host/CHANGELOG.md).

- **Bluetooth LE controller**

- Channel sounding updates:

- * Fixed the RF-PHY receiver testcases (RF-PHY/RCV-LE/CA/BV-7-C, RF-PHY/RCV-LE/CA/BV-13-C, RF-PHY/RCV-LE/CA/BV-30-C, RF-PHY/RCV-LE/CA/BV-31-C).
 - * Fix provided to set bit in pHadmConn->uiFlags for rxed LL_CS_CAPABILITIES_REQ/RSP, allow multiple LE CS Write Cached Remote Supported Capabilities.
 - * RTT bias compensation : Updated software to align with Chip IFR version 2.5
 - PAwR/PAST updates:
 - * Fixed multiple responses formatting in the Periodic Advertising Responses report.
 - * Fixed the double message allocation causing a memory leak.
 - DBAF update:
 - * Fixed case where sometimes AUX_ADV_IND with LE Coded Phy was not scheduled.
 - Fixed handling of duplicate Connection Request using different Peer RPA (LL/SEC/ADV/BI-01-C & LL/SEC/INI/BI-01-C conformance tests).
 - Fixed the missing chained packet TX when coded PHY is used on multiple Advertising sets and the Advertising interval is small (LL/DDI/ADV/BV-30-C conformance test).
 - Fixed issue with packet time restrictions after PHY update (packet payload too long). This fixes the LL/CON/PER/BV-53-C, LL/CON/PER/BV-56-C and LL/CON/PER/BV-59-C conformance tests.
 - Fixed case where address resolution was stuck.
- **Transceiver drivers (XCVR)**
 - Added support for Bluetooth LE Channel Sounding.
 - Added API to control PA ramp type and duration.
 - **Connectivity framework (compared to 25.06.00 release)**
 - **Major Changes**
 - * [wireless_mcu] Replaced the ICS RX linked list with message queue to eliminate memory allocation in the 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.
 - * [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.
 - * [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.
 - * [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_HSL0T) is set to 2 while BOARD_RADIO_DOMAIN_WAKE_UP_DELAY is 0x16.

– Minor Changes (no impact on application)

- * [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.
- * [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_d are disable
- * [Common] Added MDK compatibility for the errno framework header.
- * [OTA] Enabled calling OTA_GetImgState() prior to OTA_Initialize().

- * [wireless_mcu] Fixed PLATFORM_IsExternalFlashSectorBlank() to check the entire sector instead of just one page.
- * [OTA] Removed gUseInternalStorageLink_d linker flag definition when external OTA storage is used.
- * [wireless_mcu] Resolved counter wrap issue in PLATFORM_GetDeltaTimeStamp().
- * [kw47_mcxw72] Updated shared memory allocation for RPMsg adapter.
- * [kw45_mcxw71][kw47_mcxw72] Moved RAM bank definitions from the connectivity framework to device-specific definitions.
- * [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 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.
- * [SFC] Remove obsolete flag gNbuJtagCapability.
- * [wireless_mcu] Introduced new API PLATFORM_GetRadioIdleDuration32K(). Deprecated PLATFORM_CheckNextBleConnectivityActivity() API.
- * [DBG] Cleaned up fwk_fault_handler.c.

Details can be found in *CHANGELOG.md*

Known issues

This section lists the known issues, limitations, and/or workarounds.

Maximum file path length in Windows 7 operating system

The Windows 7 operating system imposes a 260-character maximum length for file paths. When installing the MCUXpresso SDK, place it in a directory close to the root to prevent file paths from exceeding the maximum character length specified by the Windows operating system. The recommended location is the C:\nxp folder.

Only FreeRTOS is tested for RTOS support

This release only supports the FreeRTOS kernel and a bare-metal non-preemptive task scheduler.

Bluetooth LE

Most sensor applications have pairing and bonding disabled to allow a faster interaction with mobile applications. These two security features can be enabled in the app_preinclude.h header file.

Bluetooth LE controller:

- The maximum Advertising data length is limited to 800 bytes.
- Missing chained packets during scanning.
- Need to add a dummy nbu request to wake-up NBU core during asynchronous wakeup to avoid low power race condition issue (ie: for NBU ramlog dump from shared memory).

Periodic Advertising with Responses (PAwR):

- Periodic Advertising with Response (PAwR) is not supported with the configuration “Subevent Interval = Number of Response Slots * Response Slot Spacing”.
- Unoptimized Periodic Advertising placement with Connection events.

KW45/MCXW71: No specific issues.

KW47/MCXW72: Channel Sounding (CS): Limitations:

- RTT with Sounding Sequence is not supported.
- TX SNR is not supported.
- LE 2M 2BT PHY is not supported.
- Maximum 6 Channel Sounding procedures are supported in parallel.
- Scheduling of activities may be non optimal when multiple Channel Sounding procedures are running in parallel.
- Phase measurement bias is within certification range ($<1.7 \times 2\pi$ ns) with KW47 EVK board. However, if different PCB or antenna matching is used, some bias may appear due to increased delay.
- Channel Sounding is not supported with internal FRO32. Known issues:
 - When CS_SYNC=2Mbps, sensitivity drops at -85dBm (vs -95dBm for 1Mbps).
 - NADM may report false positive attacks when using localization board at 2Mbps.
 - When CS Subevents are configured very close from each other ($<700\mu$ s), some Subevents may be aborted with reason 0x3.
 - When CS offset is configured too close from ACL anchor point, the anchor point may not be served (TX on central or RX on peripheral will not happen). Ideally, CS Offset should be configured greater than 1ms.
 - Wireless_ranging stability issue in test mode 2Mbps with RTT random sequence payload 128 bits.
- RTT bias compensation:
 - For parts not properly configured at production (IFR blank), RTT bias may not be compensated properly. Consequently, an inaccuracy of +/-2m may be observed.
 - Temperature variation: RTT variation of +/-2m is observed based on temperature.

Other limitations

- The following Connectivity Framework configurations are Experimental and not recommended for mass production:
 - Power down on application power domain.
- A hardfault can be encountered when using `fsl_component_mem_manager_light.c` memory allocator and shutting down some unused RAM banks in low power. It is due to a wrong reinitialization of ECC RAM banks. To be sure not to reproduce the issue, `gPlatformShutdownEccRamInLowPower` should be set to 0.

- GenFSK Connectivity_test application is not operational with Low Power enabled.
- Serial manager is only supported on UART (not I2C nor SPI).
- The --no-warn-rwx-segments cannot be recognized on legacy MCUXpresso IDE versions.

The --no-warn-rwx-segments option in MCUXpresso projects should be manually removed from the project settings if someone needs to use legacy (< 11.8.0) MCUXpresso IDE versions

- If the FRO32K is configured as the clock source of the CM33 Core then the debug session will block in both IAR, MCUX CMSIS-DAP while debugging. Use a lower debug wire speed, for example 1 MHz instead of the default one.

In IAR, the option is in **Runtime Checking -> Debugger -> CMSIS DAP -> Interface -> Interface speed.**

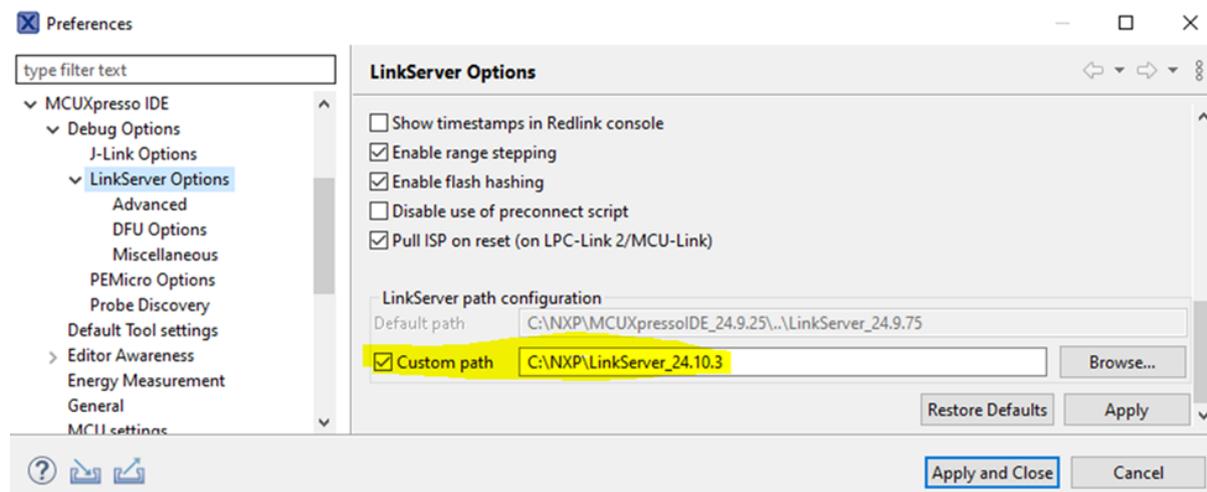
In MCUXpresso IDE, the option is in **LinkServer Debugger -> Advanced Settings -> Wire-speed (Hz).**

- Low power reference design applications are not supported for the armgcc toolchain from zip archives. Please use MCUXpresso IDE or IAR toolchains for development using these applications.

Latest MCUX IDE 24.9.25 cannot support KW47 multicore compiling

The latest MCUX IDE 24.9.25 cannot support KW47 multicore compiling, users need to upgrade to the Linkserver_24.10.22 or higher version, and change the LinkServer path configuration in the MCUX IDE. Two ways to change the LinkServer path in the MCUX IDE.

Option 1 (recommended): Using the custom Path



Option 2: Using the command-line to change the settings

1. Close the MCUXpresso IDE if it is open.
2. Execute the following command:

```
<path_to_MCUXpressoIDE_installation_folder>\ide\mcuxpressoide -application com.nxp.mcuxpresso.
->headless.application -nosplash -run set.config.preference com.nxp.mcuxpresso.core.debug.support.
->linkserver:linkserver.path.default_path=<path_to_LinkServer_installation_folder>
```

where:

- <path_to_MCUXpressoIDE_installation_folder> is the folder where the MCUXpresso IDE is installed.

- <path_to_LinkServer_installation_folder> is the folder where the new/custom LinkServer is installed.

Example:

```
C:\NXP\MCUXpressoIDE_24.9.25\ide\mcuxpressoide -application com.nxp.mcuxpresso.headless.application -
↳ nosplash -run set.config.preference com.nxp.mcuxpresso.core.debug.support.linkserver:linkserver.path.
↳ default_path=c:\NXP\LinkServer_24.10.15
```

1.5 ChangeLog

1.5.1 MCUXpresso SDK Changelog

Board Support Files

board

[25.06.00]

- Initial version

clock_config

[25.06.00]

- Initial version

pin_mux

[25.06.00]

- Initial version
-

CACHE LPCAC

[2.1.2]

- Improvements
 - Add memory barrier when enabling/disabling cache.

[2.1.1]

- Bug Fixes
 - MISRA C-2012 issue fixed: rule 10.1.

[2.1.0]

- Bug Fixes
 - Deleted L1CACHE_EnableCodeCacheWriteBuffer function because of no enable bit in register CPCR2.

[2.0.0]

- Initial version.
-

CCM32K

[2.2.0]

- Improvements
 - Removed wait loops in CCM32K's APIs.

[2.1.1]

- Improvements
 - Updated the judgment conditions in the CCM32K_Enable32kFro() API to avoid code stuck in the while loop.

[2.1.0]

- New Features
 - Added clock monitor control function group and clock gate control function group to support some devices that support clock monitor and clock gate control.

[2.0.0]

- Initial version.
-
-

CLOCK

[2.2.6]

- Bug Fixes:
 - Fixed violations of the CERT INT31-C.

[2.2.5]

- Improvements:
 - Added comments for ERR052742: FRO6M clock is not stable.

[2.2.4]

- Improvements:
 - Fixed the MSG issue that overrunning array fro192mFreq of 5 4-byte elements at element index 7.

[2.2.3]

- New Features:
 - Provide the new API CLOCK_IsFIRCAutoTrimLocked to check whether FIRC auto trim locked to target frequency range.

[2.2.2]

- Bug fix:
 - Updated the SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY for Core1.

[2.2.1]

- Bug fix:
 - Fixed the GPIO_PinInit function may cause the hardfault in clock driver when pass the GPIOD as the fist parameter.

[2.2.0]

- Improvements
 - Added the support for TPM2.

[2.1.0]

- Improvements
 - Provide the FSL_SDK_FORCE_CLK_DRIVER_NS_ACCESS macro to force use the none secure address even the trustzone secure mode is enabled.

[2.0.0]

- Fixed the MISRA issues.

[1.1.0]

- Some minor fixes.

[1.0.0]

- Initial version.
-

CMC

[2.4.3]

- Improvements
 - Add timeout for while loop code.

[2.4.2]

- Improvements
 - Add support for specific KW4x device variants
 - * Let kCMC_SecurityViolationReset depend on whether CMC_SRS_SECVIO_MASK definition exists or not.
 - * CMC_LockWriteOperationToBootRomStatusReg and CMC_CheckBootRomStatusRegWriteLocked exist only if FSL_FEATURE_CMC_HAS_NO_BOOTROM_LOCK_REGISTER is not defined

[2.4.1]

- Improvements
 - Clear SCB SCR[SLEEPDEEP] bitfield after wakeup.

[2.4.0]

- New Features
 - Added new interface to support CMC[DIER] register.
- Improvements
 - Updated _cmc_system_sram_arrays enumeration to support some devices that provide more sram bank.

[2.3.1]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.7;

[2.3.0]

- Improvements
 - Added new functions to support controls of BLR register.
 - Updated CMC_GetBootRomStatus() to support some devices that equipped multiple BootROM status registers.
 - Added CMC_WriteBootRomStatusReg() to write BootRom Status register.

[2.2.1]

- Improvements
 - For some devices, SRS_JTAG bit is reserved. Added a feature macro to adapt to different devices.

[2.2.0]

- Improvements
 - Updated `_cmc_system_sram_arrays` enumeration, make it more universal.
 - Updated SRAM related APIs(`CMC_PowerOffSRAMAllMode()`, `CMC_PowerOffSRAMLowPowerOnly()`, `CMC_PowerOnSRAMAllMode()`, `CMC_PowerOnSRAMLowPowerOnly()`), due to updates of registers' names.
 - Renamed `CMC_GetBootConfigPinLogic()` to `CMC_GetISPMODEPinLogic()`.
 - Renamed `CMC_ClearBootConfig()` to `CMC_ClearISPMODEPinLogic()`.
 - Updated enumeration `_cmc_power_mode_protection` due to some macros are deleted in header file.

[2.1.0]

- Improvements
 - Added some macros to separate the scenes that some registers are reserved in some devices.

[2.0.0]

- Initial version.
-

COMMON

[2.6.0]

- Bug Fixes
 - Fix CERT-C violations.

[2.5.0]

- New Features
 - Added new APIs `InitCriticalSectionMeasurementContext`, `DisableGlobalIRQEx` and `EnableGlobalIRQEx` so that user can measure the execution time of the protected sections.

[2.4.3]

- Improvements
 - Enable irqs that mount under `irqsteer` interrupt extender.

[2.4.2]

- Improvements
 - Add the macros to convert peripheral address to secure address or non-secure address.

[2.4.1]

- Improvements
 - Improve for the macro redefinition error when integrated with `zephyr`.

[2.4.0]

- New Features
 - Added EnableIRQWithPriority, IRQ_SetPriority, and IRQ_ClearPendingIRQ for ARM.
 - Added MSDK_EnableCpuCycleCounter, MSDK_GetCpuCycleCount for ARM.

[2.3.3]

- New Features
 - Added NETC into status group.

[2.3.2]

- Improvements
 - Make driver aarch64 compatible

[2.3.1]

- Bug Fixes
 - Fixed MAKE_VERSION overflow on 16-bit platforms.

[2.3.0]

- Improvements
 - Split the driver to common part and CPU architecture related part.

[2.2.10]

- Bug Fixes
 - Fixed the ATOMIC macros build error in cpp files.

[2.2.9]

- Bug Fixes
 - Fixed MISRA C-2012 issue, 5.6, 5.8, 8.4, 8.5, 8.6, 10.1, 10.4, 17.7, 21.3.
 - Fixed SDK_Malloc issue that not allocate memory with required size.

[2.2.8]

- Improvements
 - Included stddef.h header file for MDK tool chain.
- New Features:
 - Added atomic modification macros.

[2.2.7]

- Other Change
 - Added MECC status group definition.

[2.2.6]

- Other Change
 - Added more status group definition.
- Bug Fixes
 - Undef __VECTOR_TABLE to avoid duplicate definition in cmsis_clang.h

[2.2.5]

- Bug Fixes
 - Fixed MISRA C-2012 rule-15.5.

[2.2.4]

- Bug Fixes
 - Fixed MISRA C-2012 rule-10.4.

[2.2.3]

- New Features
 - Provided better accuracy of SDK_DelayAtLeastUs with DWT, use macro SDK_DELAY_USE_DWT to enable this feature.
 - Modified the Cortex-M7 delay count divisor based on latest tests on RT series boards, this setting lets result be closer to actual delay time.

[2.2.2]

- New Features
 - Added include RTE_Components.h for CMSIS pack RTE.

[2.2.1]

- Bug Fixes
 - Fixed violation of MISRA C-2012 Rule 3.1, 10.1, 10.3, 10.4, 11.6, 11.9.

[2.2.0]

- New Features
 - Moved SDK_DelayAtLeastUs function from clock driver to common driver.

[2.1.4]

- New Features
 - Added OTFAD into status group.

[2.1.3]

- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fixed the rule: rule-10.3.

[2.1.2]

- Improvements
 - Add SUPPRESS_FALL_THROUGH_WARNING() macro for the usage of suppressing fallthrough warning.

[2.1.1]

- Bug Fixes
 - Deleted and optimized repeated macro.

[2.1.0]

- New Features
 - Added IRQ operation for XCC toolchain.
 - Added group IDs for newly supported drivers.

[2.0.2]

- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fixed the rule: rule-10.4.

[2.0.1]

- Improvements
 - Removed the implementation of LPC8XX Enable/DisableDeepSleepIRQ() function.
 - Added new feature macro switch “FSL_FEATURE_HAS_NO_NONCACHEABLE_SECTION” for specific SoCs which have no noncacheable sections, that helps avoid an unnecessary complex in link file and the startup file.
 - Updated the align(x) to **attribute**(aligned(x)) to support MDK v6 armclang compiler.

[2.0.0]

- Initial version.
-

CRC

[2.0.4]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.0.3]

- Bug fix:
 - Fix MISRA issues.

[2.0.2]

- Bug fix:
 - Fix MISRA issues.

[2.0.1]

- Bug fix:
 - DATA and DATALL macro definition moved from header file to source file.

[2.0.0]

- Initial version.
-

EDMA (DMA3)

[2.5.1]

- Bug Fixes
 - enables the auto stop request feature in API EDMA_ResetChannel

[2.5.0]

- Improvements
 - Add API EDMA_GetTransferSize for EDMA_PrepareTransferConfig to reduce HIS CCM value.

[2.4.0]

- Improvements
 - Added peripheral to peripheral support in DMA3 driver.
- Bug Fixes
 - Fixed the ERQ bit reading error issue.

[2.3.2]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.3.1]

- Bug Fixes
 - Added clear TCD_CITER_ELINKNO and TCD_BITER_ELINKNO registers in EDMA_AbortTransfer to make sure the TCD registers in a correct state for next calling of EDMA_SubmitTransfer.

[2.3.0]

- Improvements
 - Added feature FSL_FEATURE_EDMA_HAS_NO_SBR_ATTR_BIT to separate DMA without ATTR bitfield.

[2.2.7]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.8, 5.6.

[2.2.6]

- Bug Fixes
 - Fixed the TCD overwrite issue when submit transfer request in the callback if there is a active TCD in hardware.

[2.2.5]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.1, 10.4.

[2.2.4]

- Bug Fixes
 - Fix the issue that EDMA_AbortTransfer not reset edma handle(tcdUsed, tail, header) and fix EDMA_InstallTCMemory(handle->header = 1)

[2.2.3]

- Improvements
 - Added feature FSL_FEATURE_EDMA_MODULE_CHANNEL_IRQ_ENTRY_SUPPORT_PARAMETER to support driver IRQ handler with parameters.
 - Added feature FSL_FEATURE_EDMA_HAS_COMMON_CLOCK_GATE to improve clock gate control in dma driver.

[2.2.2]

- Bug Fixes
 - Fixed the issue of EDMA_SubmitTransfer return busy when calling EDMA_EnableChannelInterrupts before submit transfer.
 - Fixed violations of MISRA C-2012 rule 10.4, 10.1, 9.2, 10.4, 10.6, 14.4, 10.7, 14.3, 11.6.

[2.2.1]

- Improvements
 - Removed channel MUX reset from `EDMA_ResetChannel`, since channel mux should be constant while channel is alive.

[2.2.0]

- Improvements
 - Added new API `EDMA_SetChannelMux` to support channel mux feature.
 - Added new API `EDMA_PrepareTransferConfig` to expose parameters source offset and destination offset.
 - Exposed `EDMA_InstallTCD` function to application.
 - Added source/destination address alignment check.

[2.1.1]

- Improvements
 - Added 8bytes transfer width feature support in driver.

[2.1.0]

- Bug Fixes
 - Added const type for parameter configuration in `EDMA_SubmitTransfer` and `EDMA_HandleTransferConfig` API.
 - Added configurations for `srcAddr` and `destAddr` in `EDMA_PrepareTransfer` API.

[2.0.2]

- Improvements
 - Updated eDMA driver to support `MP_CR` bit GMRC.
 - Updated eDMA instance name for i.MX 8QM.
 - Used instance number as factor to calculate channel number for different instance instead of hard code.

[2.0.1]

- New Features
 - Added control macro to enable/disable the `CLOCK` code in current driver.
 - Added `s_EDMAEnabledChannel` to record enabled channel to merge all the channel IRQ handler into driver IRQ handler.
 - Added feature macro for bits `EMI` and `EBW` in `MP_CSR`.
- Improvements
 - Removed all the separated channel IRQ handler in DMA driver.

[2.0.0]

- Initial version.
-

ELEMU

- 2.1.2 Fix macro BIT redefined issue when compiling with Zephyr.
 - 2.1.1 Fix MISRA issues.
 - 2.1.0 Cleanup of unused legacy FW. Updated ELEMU_loadFw function.
 - 2.0.0 Renamed MU to ELEMU to avoid confusion with other Messaging units.
 - 0.0.4 Updated register name according to latest device header file.
 - 0.0.3 Fix MISRA issues.
 - 0.0.2 The name of SNT has changed to ELEMU.
 - 0.0.1 Initial version of SNT driver.
-

EWM

[2.0.4]

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.0.3]

- Bug Fixes
 - Fixed violation of MISRA C-2012 rules: 10.1, 10.3.

[2.0.2]

- Bug Fixes
 - Fixed violation of MISRA C-2012 rules: 10.3, 10.4.

[2.0.1]

- Bug Fixes
 - Fixed the hard fault in EWM_Deinit.

[2.0.0]

- Initial version.
-

FLASH

[2.3.3]

- Improvements
 - Added timeout for while loop in flash_command_pre_sequence, flash_command_sequence, flash_erase_sequence, flash_pgm_sequence and flash_command_pre_sequence API.

[2.3.2]

- Improvement
- Enabled the kFLASH_PropertyPflash1SectorSize property support in FLASH_GetProperty API.

[2.3.1]

- Improvement
- Clear flash cache before every erase to prevent the possibility of returning stale data.

[2.3.0]

- New features
- Add support for MCX N series

[2.2.1]

- Improvement
 - To fix the potential unsigned integer overflow in the operation, added a check to ensure that the addition does not exceed the maximum value.

[2.2.0]

- New Features
 - Add the support for MW30.

[2.1.2]

- Improvement
 - Add the support for KW47/MCXW72 phantoms.

[2.1.1]

- Improvement
 - Add the conditional compiling flag '#if defined(RF_FMU)' to make the driver be compatible with the non-radio phantoms.

[2.1.0]

- Bug Fixes
 - Fix flash driver run error in flash memory

[2.0.0]

- Initial version.
-

FLEXCAN

[2.14.5]

- Improvements
 - Make API FLEXCAN_GetFDMailboxOffset **public**.
 - Add API FLEXCAN_SetMbID and FLEXCAN_SetFDMbID to configure Message Buffer ID individually.
- Bug Fixes
 - Fixed violations of the CERT INT30-C INT31-C.
 - Fixed violations of the CERT ARR30-C.

[2.14.4]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.4, 18.1.

[2.14.3]

- Improvements
 - Add unhandled interrupt events check for following API:
 - * FLEXCAN_MbHandleIRQ
 - * FLEXCAN_EhancedRxFifoHandleIRQ
- Bug Fixes
 - Remove FLEXCAN_MemoryErrorHandleIRQ on some platform without memory error interrupt.
 - Add conditional compile for CTRL2[ISOCANFDEN] because some platform do not have this bit.

[2.14.2]

- Improvements
 - Add Coverage Justification for uncovered code.
 - Adjust API FLEXCAN_TransferAbortReceive order.
 - Update FLEXCAN_Enable to enter Freeze Mode first when enter Disable mode on some platform.
 - Added while loop timeout for following API:
 - * FLEXCAN_EnterFreezeMode
 - * FLEXCAN_ExitFreezeMode
 - * FLEXCAN_Enable
 - * FLEXCAN_Reset

- * FLEXCAN_TransferSendBlocking
 - * FLEXCAN_TransferReceiveBlocking
 - * FLEXCAN_TransferFDSendBlocking
 - * FLEXCAN_TransferFDReceiveBlocking
 - * FLEXCAN_TransferReceiveFifoBlocking
 - * FLEXCAN_TransferReceiveEnhancedFifoBlocking
- Bug Fixes
 - Remove remote frame feature in CANFD mode because there is no remote frame in the CANFD format.
 - Remove legacy Rx FIFO disabled branch in FLEXCAN_SubHandlerForLegacyRxFIFO and FLEXCAN_SubHandlerForDataTransferred.

[2.14.1]

- Bug Fixes
 - Fixed register IMASK2-4 IFLAG2-4 HR_TIME_STAMP_n access issue on FlexCAN instances with different number of MBs.
 - Fixed bit field MBDSR1-3 access issue on FlexCAN instances with different number of MBs.
- Improvements
 - Unified following API as same parameter and return value type:
 - * FLEXCAN_GetMbStatusFlags
 - * FLEXCAN_ClearMbStatusFlags
 - * FLEXCAN_EnableMbInterrupts
 - * FLEXCAN_DisableMbInterrupts
 - Add workaround for ERR050443 and ERR052403.
 - Update message buffer read process in API FLEXCAN_ReadRxMb and FLEXCAN_ReadFDRxMb to make critical section as short as possible.
 - Simplify API FLEXCAN_DriverDataIRQHandler implementation by remove parameter type.

[2.14.0]

- Improvements
 - Support external time tick feature.
 - Support high resolution timestamp feature.
 - Enter Freeze Mode first when enter Disable Mode on some platform.
 - Add feature macro for Pretended Networking because some FlexCAN instance do not have this feature.
 - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
 - Add new FlexCAN IRQ Handler FLEXCAN_DriverDataIRQHandler and FLEXCAN_DriverEventIRQHandler. Thses IRQ Handlers are used on soc which FlexCAN interrupts are grouped by specific function and assigned to different vector.

- Update macro FLEXCAN_WAKE_UP_FLAG and FLEXCAN_PNWAKE_UP_FLAG to simplify code.
- Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKMSK_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_NO_SLFWAK_SUPPORT.
- Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKSRC_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_GLITCH_FILTER.
- Bug Fixes
 - Fixed wrong interrupt and status flag helper macro in enumeration flexcan_flags and API FLEXCAN_DisableInterrupts.
 - Fixed interrupt flag helper macro typo issue.
 - Remove flags which will be unassociated with interrupt in macro FLEXCAN_MEMORY_ERROR_INT_FLAG.
 - Remove flags which will be unassociated with interrupt in macro FLEXCAN_ERROR_AND_STATUS_INT_FLAG.
 - Fixed array out-of-bounds access when read enhanced Rx FIFO.

[2.13.1]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.13.0]

- Improvements
 - Support payload endianness selection feature.

[2.12.0]

- Improvements
 - Support automatic Remote Response feature.
 - Add API FLEXCAN_SetRemoteResponseMbConfig() to configure automatic Remote Response mailbox.

[2.11.8]

- Improvements
 - Synchronize flexcan driver update on s32z platform.

[2.11.7]

- Bug Fixes
 - Fixed FLEXCAN_TransferReceiveEnhancedFifoEDMA() compatibility with edma5.

[2.11.6]

- Bug Fixes
 - Fixed ERRATA_9595 FLEXCAN_EnterFreezeMode() may result to bus fault on some platform.

[2.11.5]

- Bug Fixes
 - Fixed flexcan_memset() crash under high optimization compilation.

[2.11.4]

- Improvements
 - Update CANFD max bitrate to 10Mbps on MCXNx3x and MCXNx4x.
 - Release peripheral from reset if necessary in init function.

[2.11.3]

- Bug Fixes
 - Fixed FLEXCAN_TransferReceiveEnhancedFifoEDMA() compile error with DMA3.

[2.11.2]

- Bug Fixes
 - Fixed bug that timestamp in flexcan_handle_t not updated when RX overflow happens.

[2.11.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1.

[2.11.0]

- Bug Fixes
 - Fixed wrong base address argument in FLEXCAN2 IRQ Handler.
- Improvements
 - Add API to determine if the instance supports CAN FD mode at run time.

[2.10.1]

- Bug Fixes
 - Fixed HIS CCM issue.
 - Fixed RTOS issue by adding protection to read-modify-write operations on interrupt enable/disable API.

[2.10.0]

- Improvements
 - Update driver to make it able to support devices which has more than 64 8bytes MBs.
 - Update CAN FD transfer APIs to make them set/get edl bit according to frame content, which can make them compatible with classic CAN.

[2.9.2]

- Bug Fixes
 - Fixed the issue that FLEXCAN_CheckUnhandleInterruptEvents() can't detecting the exist enhanced RX FIFO interrupt status.
 - Fixed the issue that FLEXCAN_ReadPNWakeUpMB() does not return fail even no existing valid wake-up frame.
 - Fixed the issue that FLEXCAN_ReadEnhancedRxFifo() may clear bits other than the data available bit.
 - Fixed violations of the MISRA C-2012 rules 10.4, 10.8.
- Improvements
 - Return kStatus_FLEXCAN_RxFifoDisabled instead of kStatus_Fail when read FIFO fail during IRQ handler.
 - Remove unreachable code from timing calculates APIs.
 - Update Enhanced Rx FIFO handler to make it deal with underflow/overflow status first.

[2.9.1]

- Bug Fixes
 - Fixed the issue that FLEXCAN_TransferReceiveEnhancedFifoBlocking() API clearing Fifo data available flag more than once.
 - Fixed the issue that entering FLEXCAN_SubHandlerForEnhancedRxFifo() even if Enhanced Rx fifo interrupts are not enabled.
 - Fixed the issue that FLEXCAN_TransferReceiveEnhancedFifoEDMA() update handle even if previous Rx FIFO receive not finished.
 - Fixed the issue that FLEXCAN_SetEnhancedRxFifoConfig() not configure the ERFCR[NFE] bits to the correct value.
 - Fixed the issue that FLEXCAN_ReceiveFifoEDMACallback() can't differentiate between Rx fifo and enhanced rx fifo.
 - Fixed the issue that FLEXCAN_TransferHandleIRQ() can't report Legacy Rx FIFO warning status.

[2.9.0]

- Improvements
 - Add public set bit rate API to make driver easier to use.
 - Update Legacy Rx FIFO transfer APIs to make it support received multiple frames during one API call.
 - Optimized FLEXCAN_SubHandlerForDataTransferred() API in interrupt handling to reduce the probability of packet loss.

[2.8.7]

- Improvements
- Initialized the EDMA configuration structure in the FLEXCAN EDMA driver.

[2.8.6]

- Bug Fixes
- Fix Coverity overrun issues in fsl_flexcan_edma driver.

[2.8.5]

- Improvements
 - Make driver aarch64 compatible.

[2.8.4]

- Bug Fixes
 - Fixed FlexCan_Errata_6032 to disable all interrupts.

[2.8.3]

- Bug Fixes
 - Fixed an issue with the FLEXCAN_EnableInterrupts and FLEXCAN_DisableInterrupts interrupt enable bits in the CTRL1 register.

[2.8.2]

- Bug Fixes
 - Fixed errors in timing calculations and simplify the calculation process.
 - Fixed issue of CBT and FDCBT register may write failure.

[2.8.1]

- Bug Fixes
 - Fixed the issue of CAN FD three sampling points.
 - Added macro to support the devices that no MCR[SUPV] bit.
 - Remove unnecessary clear WMB operations.

[2.8.0]

- Improvements
 - Update config configuration.
 - * Added enableSupervisorMode member to support enable/disable Supervisor mode.
 - Simplified the algorithm in CAN FD improved timing APIs.

[2.7.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.3, 10.7.

[2.7.0]

- Improvements
 - Update config configuration.
 - * Added enablePretendedNetworking member to support enable/disable Pretended Networking feature.
 - * Added enableTransceiverDelayMeasure member to support enable/disable Transceiver Delay MeasurementPretended feature.
 - * Added bitRate/bitRateFD member to work as baudRate/baudRateFD member union.
 - Rename all “baud” in code or comments to “bit” to align with the CAN spec.
 - Added Pretended Networking mode related APIs.
 - * FLEXCAN_SetPNConfig
 - * FLEXCAN_GetPNMatchCount
 - * FLEXCAN_ReadPNWakeUpMB
 - Added support for Enhanced Rx FIFO.
 - Removed independent memory error interrupt/status APIs and put all interrupt/status control operation into FLEXCAN_EnableInterrupts/FLEXCAN_DisableInterrupts and FLEXCAN_GetStatusFlags/FLEXCAN_ClearStatusFlags APIs.
 - Update improved timing APIs to make it calculate improved timing according to CiA doc recommended.
 - * FLEXCAN_CalculateImprovedTimingValues.
 - * FLEXCAN_FDCalculateImprovedTimingValues.
 - Update FLEXCAN_SetBitRate/FLEXCAN_SetFDBitRate to added the use of enhanced timing registers.

[2.6.2]

- Improvements
 - Add CANFD frame data length enumeration.

[2.6.1]

- Bug Fixes
 - Fixed the issue of not fully initializing memory in FLEXCAN_Reset() API.

[2.6.0]

- Improvements
 - Enable CANFD ISO mode in FLEXCAN_FDInit API.
 - Enable the transceiver delay compensation feature when enable FD operation and set bitrate switch.

- Implementation memory error control in FLEXCAN_Init API.
- Improve FLEXCAN_FDCalculateImprovedTimingValues API to get same value for FPRESDIV and PRESDIV.
- Added memory error configuration for user.
 - * enableMemoryErrorControl
 - * enableNonCorrectableErrorEnterFreeze
- Added memory error related APIs.
 - * FLEXCAN_GetMemoryErrorReportStatus
 - * FLEXCAN_GetMemoryErrorStatusFlags
 - * FLEXCAN_ClearMemoryErrorStatusFlags
 - * FLEXCAN_EnableMemoryErrorInterrupts
 - * FLEXCAN_DisableMemoryErrorInterrupts
- Bug Fixes
 - Fixed the issue of sent duff CAN frame after call FLEXCAN_FDInit() API.

[2.5.2]

- Bug Fixes
 - Fixed the code error issue and simplified the algorithm in improved timing APIs.
 - * The bit field in CTRL1 register couldn't calculate higher ideal SP, we set it as the lowest one(75%)
 - FLEXCAN_CalculateImprovedTimingValues
 - FLEXCAN_FDCalculateImprovedTimingValues
 - Fixed MISRA-C 2012 Rule 17.7 and 14.4.
- Improvements
 - Pass EsrStatus to callback function when kStatus_FLEXCAN_ErrorStatus is coming.

[2.5.1]

- Bug Fixes
 - Fixed the non-divisible case in improved timing APIs.
 - * FLEXCAN_CalculateImprovedTimingValues
 - * FLEXCAN_FDCalculateImprovedTimingValues

[2.5.0]

- Bug Fixes
 - MISRA C-2012 issue check.
 - * Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-11.8, rule-12.2, rule-13.4, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-16.4, rule-17.3, rule-5.8, rule-8.3, rule-8.5.
 - Fixed the issue that API FLEXCAN_SetFDRxMbConfig lacks inactive message buff.
 - Fixed the issue of Pa082 warning.

- Fixed the issue of dead lock in the function of interruption handler.
- Fixed the issue of Legacy Rx Fifo EDMA transfer data fail in evkmimxrt1060 and evkmimxrt1064.
- Fixed the issue of setting CANFD Bit Rate Switch.
- Fixed the issue of operating unknown pointer risk.
 - * when used the pointer “handle->mbFrameBuf[mbIdx]” to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
- Added assert to check current CAN clock source affected by other clock gates in current device.
 - * In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).
- Improvements
 - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
 - Optimized the function of interruption handler.
 - Added two APIs for FLEXCAN EDMA driver.
 - * FLEXCAN_PrepareTransfConfiguration
 - * FLEXCAN_StartTransferDatafromRxFIFO
 - Added new API for FLEXCAN driver.
 - * FLEXCAN_GetTimeStamp
 - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
 - * FLEXCAN_EnterFreezeMode
 - * FLEXCAN_ExitFreezeMode
 - Added new configuration for user.
 - * disableSelfReception
 - * enableListenOnlyMode
 - Renamed the two clock source enum macros based on CLKSRC bit field value directly.
 - * The CLKSRC bit value had no property about Oscillator or Peripheral type in lots of devices, it acted as two different clock input source only, but the legacy enum macros name contained such property, that misled user to select incorrect CAN clock source.
 - Created two new enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrc0
 - * kFLEXCAN_ClkSrc1
 - Deprecated two legacy enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrcOsc

- * kFLEXCAN_ClkSrcPeri
- Changed the process flow for Remote request frame response..
 - * Created a new enum macro for the FLEXCAN driver.
 - kStatus_FLEXCAN_RxRemote
- Changed the process flow for kFLEXCAN_StateRxRemote state in the interrupt handler.
 - * Should the TX frame not register to the pointer of frame handle, interrupt handler would not be able to read the remote response frame from the mail box to ram, so user should read the frame by manual from mail box after a complete remote frame transfer.

[2.4.0]

- Bug Fixes
 - MISRA C-2012 issue check.
 - * Fixed rules, containing: rule-12.1, rule-17.7, rule-16.4, rule-11.9, rule-8.4, rule-14.4, rule-10.8, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-11.6, rule-13.5, rule-11.3, rule-8.3, rule-12.2 and rule-16.1.
 - Fixed the issue that CANFD transfer data fail when bus baudrate is 30Khz.
 - Fixed the issue that ERR009595 does not follow the ERRATA document.
 - Fixed code error for ERR006032 work around solution.
 - Fixed the Coverity issue of BAD_SHIFT in FLEXCAN.
 - Fixed the Repo build warning issue for variable without initial.
- Improvements
 - Fixed the run fail issue of FlexCAN RemoteRequest UT Case.
 - Implementation all TX and RX transferring Timestamp used in FlexCAN demos.
 - Fixed the issue of UT Test Fail for CANFD payload size changed from 64BperMB to 8PerMB.
 - Implementation for improved timing API by baud rate.

[2.3.2]

- Improvements
 - Implementation for ERR005959.
 - Implementation for ERR005829.
 - Implementation for ERR006032.

[2.3.1]

- Bug Fixes
 - Added correct handle when kStatus_FLEXCAN_TxSwitchToRx is coming.

[2.3.0]

- Improvements
 - Added self-wakeup support for STOP mode in the interrupt handling.

[2.2.3]

- Bug Fixes
 - Fixed the issue of CANFD data phase's bit rate not set as expected.

[2.2.2]

- Improvements
 - Added a time stamp feature and enable it in the interrupt_transfer example.

[2.2.1]

- Improvements
 - Separated CANFD initialization API.
 - In the interrupt handling, fix the issue that the user cannot use the normal CAN API when with an FD.

[2.2.0]

- Improvements
 - Added `FSL_FEATURE_FLEXCAN_HAS_SUPPORT_ENGINE_CLK_SEL_REMOVE` feature to support SoCs without CAN Engine Clock selection in FlexCAN module.
 - Added FlexCAN Serial Clock Operation to support i.MX SoCs.

[2.1.0]

- Bug Fixes
 - Corrected the spelling error in the function name `FLEXCAN_XXX()`.
 - Moved Freeze Enable/Disable setting from `FLEXCAN_Enter/ExitFreezeMode()` to `FLEXCAN_Init()`.
 - Corrected wrong helper macro values.
- Improvements
 - Hid `FLEXCAN_Reset()` from user.
 - Used `NDEBUG` macro to wrap `FLEXCAN_IsMbOccupied()` function instead of `DEBUG` macro.

[2.0.0]

- Initial version.
-

FLEXCAN_EDMA

[2.12.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 18.1.

[2.12.0]

- Improvements
 - Support high resolution timestamp feature in enhanced Rx FIFO EDMA.
 - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
- Bug Fixes
 - Fixed array out-of-bounds access when read enhanced Rx FIFO in EDMA.

[2.11.7]

- Refer FLEXCAN driver change log 2.7.0 to 2.11.7
-

FLEXIO

[2.3.0]

- Improvements
 - Supported platforms which don't have DOZE mode control.
 - Added more pin control functions.

[2.2.3]

- Improvements
 - Adapter the FLEXIO driver to platforms which don't have system level interrupt controller, such as NVIC.

[2.2.2]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.2.1]

- Improvements
 - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

- New Features
 - Added new APIs to support FlexIO pin register.

[2.1.0]

- Improvements
 - Added API FLEXIO_SetClockMode to set flexio channel counter and source clock.

[2.0.4]

- Bug Fixes
 - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.2]

- Improvements
 - Split FLEXIO component which combines all flexio/flexio_uart/flexio_i2c/flexio_i2s drivers into several components: FlexIO component, flexio_uart component, flexio_i2c_master component, and flexio_i2s component.
- Bug Fixes
 - Fixed MISRA issues
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.0.1]

- Bug Fixes
 - Fixed the doze mode configuration error in FLEXIO_Init API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
-

FLEXIO_I2C

[2.6.2]

- Improvements
 - Added timeout for while loop in FLEXIO_I2C_MasterTransferBlocking().
- Bug Fixes
 - Fixed build issues related to I2C_RETRY_TIMES.

[2.6.1]

- Bug Fixes
 - Fixed coverity issues

[2.6.0]

- Improvements
 - Supported platforms which don't have DOZE mode control.

[2.5.1]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.0]

- Improvements
 - Split some functions, fixed CCM problem in file `fsl_flexio_i2c_master.c`.

[2.4.0]

- Improvements
 - Added delay of 1 clock cycle in `FLEXIO_I2C_MasterTransferRunStateMachine` to ensure that bus would be idle before next transfer if master is nacked.
 - Fixed issue that the restart setup time is less than the time in I2C spec by adding delay of 1 clock cycle before restart signal.

[2.3.0]

- Improvements
 - Used 3 timers instead of 2 to support transfer which is more than 14 bytes in single transfer.
 - Improved `FLEXIO_I2C_MasterTransferGetCount` so that the API can check whether the transfer is still in progress.
- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.2.0]

- New Features
 - Added timeout mechanism when waiting certain state in transfer API.
 - Added an API for checking bus pin status.
- Bug Fixes
 - Fixed COVERITY issue of useless call in `FLEXIO_I2C_MasterTransferRunStateMachine`.
 - Fixed MISRA issues
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.
 - Added codes in `FLEXIO_I2C_MasterTransferCreateHandle` to clear pending NVIC IRQ, disable internal IRQs before enabling NVIC IRQ.
 - Modified code so that during master's nonblocking transfer the start and slave address are sent after interrupts being enabled, in order to avoid potential issue of sending the start and slave address twice.

[2.1.7]

- Bug Fixes
 - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking did not wait for STOP bit sent.
 - Fixed COVERITY issue of useless call in FLEXIO_I2C_MasterTransferRunStateMachine.
 - Fixed the issue that I2C master did not check whether bus was busy before transfer.

[2.1.6]

- Bug Fixes
 - Fixed the issue that I2C Master transfer APIs(blocking/non-blocking) did not support the situation of master transfer with subaddress and transfer data size being zero, which means no data followed the subaddress.

[2.1.5]

- Improvements
 - Unified component full name to FLEXIO I2C Driver.

[2.1.4]

- Bug Fixes
 - The following modifications support FlexIO using multiple instances:
 - * Removed FLEXIO_Reset API in module Init APIs.
 - * Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
 - * Updated module Enable APIs to only support enable operation.

[2.1.3]

- Improvements
 - Changed the prototype of FLEXIO_I2C_MasterInit to return kStatus_Success if initialized successfully or to return kStatus_InvalidArgument if “(srcClock_Hz / masterConfig->baudRate_Bps) / 2 - 1” exceeds 0xFFU.

[2.1.2]

- Bug Fixes
 - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
 - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
 - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
 - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
 - Fixed the Dozen mode configuration error in FLEXIO_I2C_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.

- Fixed the issue that FLEXIO_I2C_MasterTransferBlocking API called FLEXIO_I2C_MasterTransferCreateHandle, which lead to the s_flexioHandle/s_flexioIsr/s_flexioType variable being written. Then, if calling FLEXIO_I2C_MasterTransferBlocking API multiple times, the s_flexioHandle/s_flexioIsr/s_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

[2.1.1]

- Bug Fixes
 - Implemented the FLEXIO_I2C_MasterTransferBlocking API which is defined in header file but has no implementation in the C file.

[2.1.0]

- New Features
 - Added Transfer prefix in transactional APIs.
 - Added transferSize in handle structure to record the transfer size.
-

FLEXIO_I2S

[2.2.2]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 12.4.

[2.2.1]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.2.0]

- New Features
 - Added timeout mechanism when waiting certain state in transfer API.
- Bug Fixes
 - Fixed IAR Pa082 warnings.
 - Fixed violations of the MISRA C-2012 rules 10.4, 14.4, 11.8, 11.9, 10.1, 17.7, 11.6, 10.3, 10.7.

[2.1.6]

- Bug Fixes
 - Added reset flexio before flexio i2s init to make sure flexio status is normal.

[2.1.5]

- Bug Fixes
 - Fixed the issue that I2S driver used hard code for bitwidth setting.

[2.1.4]

- Improvements
 - Unified component's full name to FLEXIO I2S (DMA/EDMA) driver.

[2.1.3]

- Bug Fixes
 - The following modifications support FLEXIO using multiple instances:
 - * Removed FLEXIO_Reset API in module Init APIs.
 - * Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
 - * Updated module Enable APIs to only support enable operation.

[2.1.2]

- New Features
 - Added configure items for all pin polarity and data valid polarity.
 - Added default configure for pin polarity and data valid polarity.

[2.1.1]

- Bug Fixes
 - Fixed FlexIO I2S RX data read error and eDMA address error.
 - Fixed FlexIO I2S slave timer compare setting error.

[2.1.0]

- New Features
 - Added Transfer prefix in transactional APIs.
 - Added transferSize in handle structure to record the transfer size.
-

FLEXIO_SPI

[2.4.3]

- Improvements
 - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

- Bug Fixes
 - Fixed FLEXIO_SPI_MasterTransferBlocking and FLEXIO_SPI_MasterTransferNonBlocking issue in CS continuous mode, the CS might not be continuous.

[2.4.1]

- Bug Fixes
 - Fixed coverity issues

[2.4.0]

- Improvements
 - Supported platforms which don't have DOZE mode control.

[2.3.5]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.3.4]

- Bug Fixes
 - Fixed the txData from void * to const void * in transmit API

[2.3.3]

- Bugfixes
 - Fixed cs-continuous mode.

[2.3.2]

- Improvements
 - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

- Bugfixes
 - Fixed IRQ SHIFTBUF overrun issue when one FLEXIO instance used as multiple SPIs.

[2.3.0]

- New Features
 - Supported FLEXIO_SPI slave transfer with continuous master CS signal and CPHA=0.
 - Supported FLEXIO_SPI master transfer with continuous CS signal.
 - Support 32 bit transfer width.
- Bug Fixes

- Fixed wrong timer compare configuration for dma/edma transfer.
- Fixed wrong byte order of rx data if transfer width is 16 bit, since the we use shifter buffer bit swapped/byte swapped register to read in received data, so the high byte should be read from the high bits of the register when MSB.

[2.2.1]

- Bug Fixes
 - Fixed bug in FLEXIO_SPI_MasterTransferAbortEDMA that when aborting EDMA transfer EDMA_AbortTransfer should be used rather than EDMA_StopTransfer.

[2.2.0]

- Improvements
 - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
 - Fixed MISRA 10.4 issues.
 - Added codes in FLEXIO_SPI_MasterTransferCreateHandle and FLEXIO_SPI_SlaveTransferCreateHandle to clear pending NVIC IRQ before enabling NVIC IRQ, to fix issue of pending IRQ interfering the on-going process.

[2.1.3]

- Improvements
 - Unified component full name to FLEXIO SPI(DMA/EDMA) Driver.
- Bug Fixes
 - Fixed MISRA issues
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.1.2]

- Bug Fixes
 - The following modification support FlexIO using multiple instances:
 - * Removed FLEXIO_Reset API in module Init APIs.
 - * Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
 - * Updated module Enable APIs to only support enable operation.

[2.1.1]

- Bug Fixes
 - Fixed bug where FLEXIO SPI transfer data is in 16 bit per frame mode with eDMA.
 - Fixed bug when FLEXIO SPI works in eDMA and interrupt mode with 16-bit per frame and Lsbfirst.
 - Fixed the Dozen mode configuration error in FLEXIO_SPI_MasterInit/FLEXIO_SPI_SlaveInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.

- Improvements
 - Added `#ifndef/#endif` to allow users to change the default TX value at compile time.

[2.1.0]

- New Features
 - Added Transfer prefix in transactional APIs.
 - Added `transferSize` in handle structure to record the transfer size.
 - Bug Fixes
 - Fixed the error register address return for 16-bit data write in `FLEXIO_SPI_GetTxDataRegisterAddress`.
 - Provided independent `IRQHandler/transfer` APIs for Master and slave to fix the baudrate limit issue.
-

FLEXIO_UART

[2.6.4]

- Improvements
 - Make `UART_RETRY_TIMES` configurable by `CONFIG_UART_RETRY_TIMES`.

[2.6.3]

- Bug Fixes
 - Fixed coverity issues

[2.6.2]

- Bug Fixes
 - Fixed coverity issues

[2.6.1]

- Improvements
 - Improve baudrate calculation method, to support higher frequency FlexIO clock source.

[2.6.0]

- Improvements
 - Supported platforms which don't have DOZE mode control.

[2.5.1]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.0]

- Improvements
 - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

- Improvements
 - Use separate data for TX and RX in flexio_uart_transfer_t.
- Bug Fixes
 - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling FLEXIO_UART_TransferReceiveNonBlocking, the received data count returned by FLEXIO_UART_TransferGetReceiveCount is wrong.

[2.3.0]

- Improvements
 - Added check for baud rate's accuracy that returns kStatus_FLEXIO_UART_BaudrateNotSupport when the best achieved baud rate is not within 3% error of configured baud rate.
- Bug Fixes
 - Added codes in FLEXIO_UART_TransferCreateHandle to clear pending NVIC IRQ before enabling NVIC IRQ, to fix issue of pending IRQ interfering the on-going process.

[2.2.0]

- Improvements
 - Added timeout mechanism when waiting for certain states in transfer driver.
- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.1.6]

- Bug Fixes
 - Fixed IAR Pa082 warnings.
 - Fixed MISRA issues
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.1.5]

- Improvements
 - Triggered user callback after all the data in ringbuffer were received in FLEXIO_UART_TransferReceiveNonBlocking.

[2.1.4]

- Improvements
 - Unified component full name to FLEXIO UART(DMA/EDMA) Driver.

[2.1.3]

- Bug Fixes
 - The following modifications support FLEXIO using multiple instances:
 - * Removed FLEXIO_Reset API in module Init APIs.
 - * Updated module Deinit APIs to reset the shifter/timer configuration instead of disabling module and clock.
 - * Updated module Enable APIs to only support enable operation.

[2.1.2]

- Bug Fixes
 - Fixed the transfer count calculation issue in FLEXIO_UART_TransferGetReceiveCount, FLEXIO_UART_TransferGetSendCount, FLEXIO_UART_TransferGetReceiveCountDMA, FLEXIO_UART_TransferGetSendCountDMA, FLEXIO_UART_TransferGetReceiveCountEDMA and FLEXIO_UART_TransferGetSendCountEDMA.
 - Fixed the Doze mode configuration error in FLEXIO_UART_Init API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
 - Added code to report errors if the user sets a too-low-baudrate which FLEXIO cannot reach.
 - Disabled FLEXIO_UART receive interrupt instead of all NVICs when reading data from ring buffer. If ring buffer is used, receive nonblocking will disable all NVIC interrupts to protect the ring buffer. This had negative effects on other IPs using interrupt.

[2.1.1]

- Bug Fixes
 - Changed the API name FLEXIO_UART_StopRingBuffer to FLEXIO_UART_TransferStopRingBuffer to align with the definition in C file.

[2.1.0]

- New Features
 - Added Transfer prefix in transactional APIs.
 - Added txSize/rxSize in handle structure to record the transfer size.
- Bug Fixes
 - Added an error handle to handle the situation that data count is zero or data buffer is NULL.

FLEXIO_UART_EDMA

[2.3.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules.

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0
-

GPIO

[2.8.2]

- Bug Fixes
 - Fixed COVERITY issue that GPIO_GetInstance could return clock array overflow values due to GPIO base and clock being out of sync.

[2.8.1]

- Bug Fixes
 - Fixed CERT INT31-C issues.

[2.8.0]

- Improvements
 - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.

[2.8.0]

- Improvements
 - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.
 - Remove support for API GPIO_GetPinsDMARequestFlags with GPIO_ISFR_COUNT <= 1.

[2.7.3]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.7.2]

- New Features
 - Support devices without PORT module.

[2.7.1]

- Bug Fixes
 - Fixed MISRA C-2012 rule 10.4 issues in GPIO_GpioGetInterruptChannelFlags() function and GPIO_GpioClearInterruptChannelFlags() function.

[2.7.0]

- New Features
 - Added API to support Interrupt select (IRQS) bitfield.

[2.6.0]

- New Features
 - Added API to get GPIO version information.
 - Added API to control a pin for general purpose input.
 - Added some APIs to control pin in secure and previlage status.

[2.5.3]

- Bug Fixes
 - Correct the feature macro typo: FSL_FEATURE_GPIO_HAS_NO_INDEP_OUTPUT_CONTORL.

[2.5.2]

- Improvements
 - Improved GPIO_PortSet/GPIO_PortClear/GPIO_PortToggle functions to support devices without Set/Clear/Toggle registers.

[2.5.1]

- Bug Fixes
 - Fixed wrong macro definition.
 - Fixed MISRA C-2012 rule issues in the FGPIO_CheckAttributeBytes() function.
 - Defined the new macro to separate the scene when the width of registers is different.
 - Removed some redundant macros.
- New Features
 - Added some APIs to get/clear the interrupt status flag when the port doesn't control pins' interrupt.

[2.4.1]

- Improvements
 - Improved GPIO_CheckAttributeBytes() function to support 8 bits width GACR register.

[2.4.0]

- Improvements
 - API interface added:
 - * New APIs were added to configure the GPIO interrupt clear settings.

[2.3.2]

- Bug Fixes
 - Fixed the issue for MISRA-2012 check.
 - * Fixed rule 3.1, 10.1, 8.6, 10.6, and 10.3.

[2.3.1]

- Improvements
 - Removed deprecated APIs.

[2.3.0]

- New Features
 - Updated the driver code to adapt the case of interrupt configurations in GPIO module. New APIs were added to configure the GPIO interrupt settings if the module has this feature on it.

[2.2.1]

- Improvements
 - API interface changes:
 - * Refined naming of APIs while keeping all original APIs by marking them as deprecated. The original APIs will be removed in next release. The main change is updating APIs with prefix of `_PinXXX()` and `_PortXXX`.

[2.1.1]

- Improvements
 - API interface changes:
 - * Added an API for the check attribute bytes.

[2.1.0]

- Improvements
 - API interface changes:
 - * Added “pins” or “pin” to some APIs’ names.
 - * Renamed “_PinConfigure” to “GPIO_PinInit”.
-

I3C

[2.14.3]

- Improvements
 - Fixed Coverity CERT-C violations.
 - Used I3C_RSTS instead of I3C special feature macro.
 - Adapted the driver to support new platform.

[2.14.2]

- Improvements
 - Added timeout for ENTDAAs process API.
 - Added build system macro to control the timeout setting.

[2.14.1]

- Improvements
 - Split the function `I3C_MasterTransferBlocking` to meet the HIS-CCM requirement.

[2.14.0]

- Improvements
 - Added the choice to set fast start header with push-pull speed when all targets addresses have MSB 0 instead of forcing to set it.
 - Deleted duplicated busy check in `I3C_MasterStart` function.

[2.13.1]

- Bug Fixes
 - Disabled Rx auto-termination in repeated start interrupt event while transfer API doesn't enable it.
 - Waited the completion event after loading all Tx data in Tx FIFO.
- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.13.0]

- New features
 - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
 - Set read termination with START at the same time in case unknown issue.
 - Set `MCTRL[TYPE]` as 0 for DDR force exit.
- Improvements
 - Added the API to reset device count assigned by ENTDAAs.
 - Provided the method to set global macro `I3C_MAX_DEVCNT` to determine how many device addresses ENTDAAs can allocate at one time.
 - Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

[2.12.0]

- Improvements
 - Added the slow clock parameter for Controller initialization function to calculate accurate timeout.
- Bug Fixes
 - Fixed the issue that BAMATCH field can't be 0. BAMATCH should be 1 for 1MHz slow clock.

[2.11.1]

- Bug Fixes
 - Fixed the issue that interrupt API transmits extra byte when subaddress and data size are null.
 - Fixed the slow clock calculation issue.

[2.11.0]

- New features
 - Added the START/ReSTART SCL delay setting for the Soc which supports this feature.
- Bug Fixes
 - Fixed the issue that ENTDA process waits Rx pending flag which causes problem when Rx watermark isn't 0. Just check the Rx FIFO count.

[2.10.8]

- Improvements
 - Support more instances.

[2.10.7]

- Improvements
 - Fixed the potential compile warning.

[2.10.6]

- New features
 - Added the I3C private read/write with 0x7E address as start.

[2.10.5]

- New features
 - Added I3C HDR-DDR transfer support.

[2.10.4]

- Improvements
 - Added one more option for master to not set RDTERM when doing I3C Common Command Code transfer.

[2.10.3]

- Improvements
 - Masked the slave IBI/MR/HJ request functions with feature macro.

[2.10.2]

- Bug Fixes
 - Added workaround for errata ERR051617: I3C working with I2C mode creates the unintended Repeated START before actual STOP on some platforms.

[2.10.1]

- Bug Fixes
 - Fixed the issue that DAA function doesn't wait until all Rx data is read out from FIFO after master control done flag is set.
 - Fixed the issue that DAA function could return directly although the disabled interrupts are not enabled back.

[2.10.0]

- New features
 - Added I3C extended IBI data support.

[2.9.0]

- Improvements
 - Added adaptive termination for master blocking transfer. Set termination with start signal when receiving bytes less than 256.

[2.8.2]

- Improvements
 - Fixed the build warning due to armgcc strict check.

[2.8.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 17.7.

[2.8.0]

- Improvements
 - Added API I3C_MasterProcessDAASpecifiedBaudrate for temporary baud rate adjustment when I3C master assigns dynamic address.

[2.7.1]

- Bug Fixes
 - Fixed the issue that I3C slave handle STOP event before finishing data transmission.

[2.7.0]

- Fixed the CCM problem in file fsl_i3c.c.
- Fixed the FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND usage issue in I3C_GetDefaultConfig and I3C_Init.

[2.6.0]

- Fixed the FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND usage issue in fsl_i3c.h.
- Changed some static functions in fsl_i3c.c as non-static and define the functions in fsl_i3c.h to make I3C DMA driver reuse:
 - I3C_GetIBIType
 - I3C_GetIBIAddress
 - I3C_SlaveCheckAndClearError
- Changed the handle pointer parameter in IRQ related functions to void * type to make it reuse in I3C DMA driver.
- Added new API I3C_SlaveRequestIBIWithSingleData for slave to request single data byte, this API could be used regardless slave is working in non-blocking interrupt or non-blocking dma.
- Added new API I3C_MasterGetDeviceListAfterDAA for master application to get the device information list built up in DAA process.

[2.5.4]

- Improved I3C driver to avoid setting state twice in the SendCommandState of I3C_RunTransferStateMachine.
- Fixed MISRA violation of rule 20.9.
- Fixed the issue that I3C_MasterEmitRequest did not use Type I3C SDR.

[2.5.3]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH and FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND.

[2.5.2]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_MERRWARN_TERM.
- Fixed the issue that call to I3C_MasterTransferBlocking API did not generate STOP signal when NAK status was returned.

[2.5.1]

- Improved the receive terminate size setting for interrupt transfer read, now it's set at beginning of transfer if the receive size is less than 256 bytes.

[2.5.0]

- Added new API `I3C_MasterRepeatedStartWithRxSize` to send repeated start signal with receive terminate size specified.
- Fixed the status used in `I3C_RunTransferStateMachine`, changed to use pending interrupts as status to be handled in the state machine.
- Fixed MISRA 2012 violation of rule 10.3, 10.7.

[2.4.0]

- Bug Fixes
 - Fixed `kI3C_SlaveMatchedFlag` interrupt is not properly handled in `I3C_SlaveTransferHandleIRQ` when it comes together with interrupt `kI3C_SlaveBusStartFlag`.
 - Fixed the inaccurate I2C baudrate calculation in `I3C_MasterSetBaudRate`.
 - Added new API `I3C_MasterGetIBIRules` to get registered IBI rules.
 - Added new variable `isReadTerm` in struct `_i3c_master_handle` for transfer state routine to check if `MCTRL.RDTERM` is configured for read transfer.
 - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
 - Fixed the slave `maxWriteLength` and `maxReadLength` does not be configured into `SMAXLIMITS` register issue.
 - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
 - Added `isHotJoin` in `i3c_slave_config_t` to request hot-join event during slave init.

[2.3.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 8.4, 17.7.
 - Fixed incorrect HotJoin event index in `I3C_GetIBIType`.

[2.3.1]

- Bug Fixes
 - Fixed the issue that call of `I3C_MasterTransferBlocking/I3C_MasterTransferNonBlocking` fails for the case which receive length 1 byte of data.
 - Fixed the issue that STOP signal is not sent when NAK status is detected during execution of `I3C_MasterTransferBlocking` function.

[2.3.0]

- Improvements
 - Added I3C common driver APIs to initialize I3C with both master and slave configuration.
 - Updated I3C master transfer callback to function set structure to include callback invoke for IBI event and slave2master event.
 - Updated I3C master non-blocking transfer model and always enable the interrupts to be able to re-act to the slave start event and handle slave IBI.

[2.2.0]

- Bug Fixes
 - Fixed the issue that I3C transfer size limit to 255 bytes.

[2.1.2]

- Bug Fixes
 - Reset default `hkeep` value to `kI3C_MasterHighKeeperNone` in `I3C_MasterGetDefaultConfig`

[2.1.1]

- Bug Fixes
 - Fixed incorrect FIFO reset operation in I3C Master Transfer APIs.
 - Fixed i3c slave IRQ handler issue, slave transmit could be underrun because tx FIFO is not filled in time right after start flag detected.

[2.1.0]

- Added definitions and APIs for I3C slave functionality, updated previous I3C APIs to support I3C functionality.

[2.0.0]

- Initial version.
-

IMU

[2.2.0]

- New Features
 - Added `IMU_BUSY_POLL_COUNT` parameter to prevent infinite polling loops in IMU operations.
 - Added timeout mechanism to all polling loops in IMU driver code.
- Improvements
 - Enhanced error handling in blocking functions to return timeout status.
 - Updated documentation to clarify timeout behavior and return values.
 - Added `IMU_ERR_TIMEOUT` error code for timeout conditions.

[2.1.1]

- Bug Fixes
 - Fix MISRA C-2012 violations.
 - Fixed `IMU_GetStatusFlags` bug that returns wrong RX FIFO status.

[2.1.0]

- Improvements:
 - Updated API prototype, remove CIU2_Type from parameters.

[2.0.0]

- Initial version.
-

LIN

[2.2.4]

- Improvements
 - Added Kconfig item LIN_LPUART_TRANSMISSION_COMPLETE_TIMEOUT, user can configure it to none zero value for the while loop to break after retrying LIN_LPUART_TRANSMISSION_COMPLETE_TIMEOUT times.

[2.2.3]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.2.2]

- Bug Fixes
 - Fixed bug in LIN_LPUART_Init, while disabling the LPUART ERROR IRQ, the variable g_linLpuartErrIrqId should be used instead of g_linLpuartRxTxIrqId. This is a typo issue.

[2.2.1]

- Bug Fixes
 - Fixed bug in LIN_LPUART_GetTransmitStatus and LIN_LPUART_GetReceiveStatus, if the current event ID is timeout, then the timeout service has already been executed, LIN_TIMEOUT should be returned.
 - Fixed bug in LIN_LPUART_AutobaudTimerValEval, if the calculated MasterBaudrate is 0 which means the baudrate is not supported, then there is no need to update the baudrate configuration, just set the node to idle state.
 - Fixed issue in LIN_LPUART_CheckWakeupSignal, the check on the timerGetTimeIntervalCallback should be executed after the first falling edge rather than the second.

[2.2.0]

- Bug Fixes
 - Fixed bug that for lin slave once break field is detected before completing the last frame no error status is raised.

[2.1.1]

- Bug Fixes
 - Fixed the MISRA C-2012 issue.
 - * Fixed rules, containing: rule-10.8, 10.3.
 - Fixed bug in LIN_MakeChecksumByte, when frame ID is 0x3D the PID should be 0x7D.

[2.1.0]

- Improvements
 - Updated the driver to support LIN stack.
 - * For LIN_LPUART_Init() function, if run successfully, the return is LIN_SUCCESS instead of LIN_INITIALIZED.
 - * Changed the LIN event ID name from “LIN_RECV_SYNC_OK” to “LIN_SYNC_OK”.
 - * Changed the LIN event ID name from “LIN_RECV_SYNIV_ERROR” to “LIN_SYNIV_ERROR”.
 - * Changed the definition macro name “LIN_MAKE_PARITY” to “MAKE_PARITY”.
 - * Changed the definition macro name “LIN_CHECK_PARITY” to “CHECK_PARITY”.
- Bug Fixes
 - Fixed the MISRA C-2012 issue.
 - * Fixed rules, containing: rule-17.7, rule-14.4, rule-11.9, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-8.6, rule-8.4, rule-11.3, rule-10.8, and rule-16.1.

[2.0.1]

- Bug Fixes
 - Fixed bug where resting LPUART FIFO initializes the LPUART module.

[2.0.0]

- Initial version.
-

LPADC

[2.9.3]

- Improvements
 - Add timeout for while loop code.

[2.9.2]

- Improvements
 - Fixed CERT-C issues.

[2.9.1]

- Bug Fixes
 - Fixed incorrect channel B FIFO selection logic.

[2.9.0]

- Bug Fixes
 - Add code to handle the case where GCC[GAIN_CAL] is a signed number.
 - Split LPADC_FinishAutoCalibration function into two functions.
 - Improved LPADC driver.

[2.8.4]

- Bug Fixes
 - Remove function 'LPADC_SetOffsetValue' assert statement, this statement may cause runtime errors in existing code.

[2.8.3]

- Bug Fixes
 - Fixed SDK lpadc driver examples compile issue, move condition 'commandId < ADC_CV_COUNT' to a more appropriate location.

[2.8.2]

- Bug Fixes
 - Fixed the violations of MISRA C-2012 rule 18.1, 10.3, 10.1 and 10.4.

[2.8.1]

- Bug Fixes
 - Fixed LPADC sample mode enum name mistake.

[2.8.0]

- Improvements
 - Release peripheral from reset if necessary in init function.
- Bug Fixes
 - Fixed function LPADC_GetConvResult() issue.
 - Fixed function LPADC_SetConvCommandConfig() bugs.

[2.7.2]

- Improvements
 - Use feature macros instead of header file macros.
- Bug Fixes
 - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.1]

- Improvements
 - Corrected descriptions of several functions.
 - Improved function LPADC_GetOffsetValue and LPADC_SetOffsetValue.
 - Revert changes of feature macros for lpadc.
 - Use feature macros instead of header file macros.
- Bug Fixes
 - Fixed the violations of MISRA C-2012 rule 10.8.
 - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.0]

- Improvements
 - Added supports of CFG2 register.
 - Removed some useless macros.

[2.6.2]

- Bug Fixes
 - Fixed the violations of MISRA C-2012 rules.
 - Fixed LPADC driver code compile error issue.

[2.6.1]

- Improvements
 - Updated the use of macros in the driver code.

[2.6.0]

- Improvements
 - Added the API LPADC_SetOffset12BitValue() to configure 12bit ADC conversion offset trim value manually.
 - Added the API LPADC_SetOffset16BitValue() to configure 16bit ADC conversion offset trim value manually.
 - Added API to set offset calibration mode.
 - Added configuration of alternate channel.
 - Updated auto calibration API and added calibration value conversion API.
- New feature
 - Added API LPADC_EnableHardwareTriggerCommandSelection() to enable trigger commands controlled by ADC_ETC.
 - Updated LPADC_DoAutoCalibration() to allow doing something else before the ADC initialization to be totally complete. Enhance initialization duration time of the ADC.
 - Added two new APIs to get/set calibration value.

[2.5.2]

- Improvements
 - Added while loop, LPADC_GetConvResult() will return only when the FIFO will not be empty.

[2.5.1]

- Bug Fixes
 - Fixed some typos in Lpadc driver comments.

[2.5.0]

- Improvements
 - Added missing items to enable trigger interrupts.

[2.4.0]

- New features
 - Added APIs to get/clear trigger status flags.

[2.3.0]

- Improvements
 - Removed LPADC_MeasureTemperature() function for the LPADC supports different temperature sensor calculation equations.

[2.2.1]

- Improvements
 - Optimized LPADC_MeasureTemperature() function to support the specific series with flash solidified calibration value.
 - Clean doxygen warnings.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.3, rule 10.8 and rule 17.7.

[2.2.0]

- New Feature
 - Added API LPADC_MeasureTemperature() to get correct temperature from the internal sensor.
- Improvements
 - Separated lpadc_conversion_resolution_mode_t with related feature macro.
- Bug Fixes
 - Fixed the violations of MISRA C-2012 rules:
 - * Rule 10.3, 10.4, 10.6, 10.7 and 17.7.

[2.1.1]

- Improvements
 - Updated the gain calibration formula.
 - Used feature to segregate the new item kLPADC_TriggerPriorityPreemptSubsequently.

[2.1.0]

- New Features
 - Added the API LPADC_SetOffsetValue() to support configure offset trim value manually.
 - Added the API LPADC_DoOffsetCalibration() to do offset calibration independently.
- Improvements
 - Improved the usage of macros and removed invalid macros.

[2.0.2]

- Improvements
 - Added support for platforms with 2 FIFOs and different calibration measures.

[2.0.1]

- Bug Fixes
 - Ensured the API LPADC_SetConvCommandConfig configure related registers correctly.

[2.0.0]

- Initial version.
-

LPCMP

[2.3.2]

- Improvements
 - Fixed LPCMP CERT-C issues.

[2.3.1]

- Improvements
 - Update LPCMP driver to be compatible with platforms that do not support LPCMP nano power mode selection.

[2.3.0]

- New Feature
 - Added some new features for platforms which support
 - * Plus input source selection.
 - * Minus input source selection.
 - * CMP to DAC link.
- Improvements
 - Removed some new features for platforms which doesn't support
 - * Functional clock source selection.
 - * DAC high power mode selection.
 - * Round Robin clock source selection.
 - * Round Robin trigger source selection.
 - * Round Robin channel sample numbers setting.
 - * Round Robin channel sample time threshold setting.
 - * Round Robin internal trigger configuration.

[2.2.0]

- Improvements
 - Change `FSL_FEATURE_LPCMP_HAS_NO_CCR0_CMP_STOP_EN` to `FSL_FEATURE_LPCMP_HAS_CCR0_CMP_STOP_EN`.

[2.1.3]

- New Feature
 - Added new macro to handle the case where some instances do not have the CCR0 `CMP_STOP_EN` bit field.

[2.1.2]

- New Feature
 - Add macros to be compatible with some platforms that do not have the CCR0 `CMP_STOP_EN` bitfield.

[2.1.1]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.1.0]

- New Features:
 - Supported round robin mode and window mode feature.

[2.0.3]

- Bug Fixes:
 - Fixed the violation of MISRA-2012 rule 17.7.

[2.0.2]

- Bug Fixes:
 - The current API LPCMP_ClearStatusFlags has to check w1c bits.

[2.0.1]

- Added control macro to enable/disable the CLOCK code in current driver.

[2.0.0]

- Initial version.
-

LPI2C

[2.6.2]

- Improvements
 - Added timeout for while loop in LPI2C_TransferStateMachineSendCommand().

[2.6.1]

- Bug Fixes
 - Fixed coverity issues.

[2.6.0]

- New Feature
 - Added common IRQ handler entry LPI2C_DriverIRQHandler.

[2.5.7]

- Improvements
 - Added support for separated IRQ handlers.

[2.5.6]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.5]

- Bug Fixes
 - Fixed LPI2C_SlaveInit() - allow to disable SDA/SCL glitch filter.

[2.5.4]

- Bug Fixes
 - Fixed LPI2C_MasterTransferBlocking() - the return value was sometime affected by call of LPI2C_MasterStop().

[2.5.3]

- Improvements
 - Added handler for LPI2C7 and LPI2C8.

[2.5.2]

- Bug Fixes
 - Fixed ERR051119 to ignore the nak flag when IGNACK=1 in LPI2C_MasterCheckAndClearError.

[2.5.1]

- Bug Fixes
 - Added bus stop incase of bus stall in LPI2C_MasterTransferBlocking.
- Improvements
 - Release peripheral from reset if necessary in init function.

[2.5.0]

- New Features
 - Added new function LPI2C_SlaveEnableAckStall to enable or disable ACKSTALL.

[2.4.1]

- Improvements
 - Before master transfer with transactional APIs, enable master function while disable slave function and vise versa for slave transfer to avoid the one affecting the other.

[2.4.0]

- Improvements
 - Split some functions, fixed CCM problem in file fsl_lpi2c.c.
- Bug Fixes
 - Fixed bug in LPI2C_MasterInit that the MCFGR2's value set in LPI2C_MasterSetBaudRate may be overwritten by mistake.

[2.3.2]

- Improvements
 - Initialized the EDMA configuration structure in the LPI2C EDMA driver.

[2.3.1]

- Improvements
 - Updated LPI2C_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes
 - Fixed bug in LPI2C_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
 - Fixed bug in LPI2C_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.
 - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
 - Fixed bug in the usage of LPI2C_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.
 - Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
 - Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

[2.3.0]

- New Features
 - Supported reading more than 256 bytes of data in one transfer as master.
 - Added API LPI2C_GetInstance.
- Bug Fixes
 - Fixed bug in LPI2C_MasterTransferAbortEDMA, LPI2C_MasterTransferAbort and LPI2C_MasterTransferHandleIRQ that before sending stop signal whether master is active and whether stop signal has been sent should be checked, to make sure no FIFO error or bus error will be caused.
 - Fixed bug in LPI2C master EDMA transactional layer that the bus error cannot be caught and returned by user callback, by monitoring bus error events in interrupt handler.
 - Fixed bug in LPI2C_GetCyclesForWidth that the parameter used to calculate clock cycle should be $2^{\text{prescaler}}$ rather than prescaler.
 - Fixed bug in LPI2C_MasterInit that timeout value should be configured after baudrate, since the timeout calculation needs prescaler as parameter which is changed during baudrate configuration.

- Fixed bug in LPI2C_MasterTransferHandleIRQ and LPI2C_RunTransferStateMachine that when master writes with no stop signal, need to first make sure no data remains in the tx FIFO before finishes the transfer.

[2.2.0]

- Bug Fixes
 - Fixed issue that the SCL high time, start hold time and stop setup time do not meet I2C specification, by changing the configuration of data valid delay, setup hold delay, clock high and low parameters.
 - MISRA C-2012 issue fixed.
 - * Fixed rule 8.4, 13.5, 17.7, 20.8.

[2.1.12]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.1.11]

- Bug Fixes
 - Fixed the bug that, during master non-blocking transfer, after the last byte is sent/received, the kLPI2C_MasterNackDetectFlag is expected, so master should not check and clear kLPI2C_MasterNackDetectFlag when remainingBytes is zero, in case FIFO is emptied when stop command has not been sent yet.
 - Fixed the bug that, during non-blocking transfer slave may nack master while master is busy filling tx FIFO, and NDF may not be handled properly.

[2.1.10]

- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fixed rule 10.3, 14.4, 15.5.
 - Fixed unaligned access issue in LPI2C_RunTransferStateMachine.
 - Fixed uninitialized variable issue in LPI2C_MasterTransferHandleIRQ.
 - Used linked TCD to disable tx and enable rx in read operation to fix the issue that for platform sharing the same DMA request with tx and rx, during LPI2C read operation if interrupt with higher priority happened exactly after command was sent and before tx disabled, potentially both tx and rx could trigger dma and cause trouble.
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 11.6, 11.9, 14.4, 17.7.
 - Fixed the waitTimes variable not re-assignment issue for each byte read.
- New Features
 - Added the IRQHandler for LPI2C5 and LPI2C6 instances.
- Improvements
 - Updated the LPI2C_WAIT_TIMEOUT macro to unified name I2C_RETRY_TIMES.

[2.1.9]

- Bug Fixes
 - Fixed Coverity issue of unchecked return value in I2C_RTOS_Transfer.
 - Fixed Coverity issue of operands did not affect the result in LPI2C_SlaveReceive and LPI2C_SlaveSend.
 - Removed STOP signal wait when NAK detected.
 - Cleared slave repeat start flag before transmission started in LPI2C_SlaveSend/LPI2C_SlaveReceive. The issue was that LPI2C_SlaveSend/LPI2C_SlaveReceive did not handle with the reserved repeat start flag. This caused the next slave to send a break, and the master was always in the receive data status, but could not receive data.

[2.1.8]

- Bug Fixes
 - Fixed the transfer issue with LPI2C_MasterTransferNonBlocking, kLPI2C_TransferNoStopFlag, with the wait transfer done through callback in a way of not doing a blocking transfer.
 - Fixed the issue that STOP signal did not appear in the bus when NAK event occurred.

[2.1.7]

- Bug Fixes
 - Cleared the stopflag before transmission started in LPI2C_SlaveSend/LPI2C_SlaveReceive. The issue was that LPI2C_SlaveSend/LPI2C_SlaveReceive did not handle with the reserved stop flag and caused the next slave to send a break, and the master always stayed in the receive data status but could not receive data.

[2.1.6]

- Bug Fixes
 - Fixed driver MISRA build error and C++ build error in LPI2C_MasterSend and LPI2C_SlaveSend.
 - Reset FIFO in LPI2C Master Transfer functions to avoid any byte still remaining in FIFO during last transfer.
 - Fixed the issue that LPI2C_MasterStop did not return the correct NAK status in the bus for second transfer to the non-existing slave address.

[2.1.5]

- Bug Fixes
 - Extended the Driver IRQ handler to support LPI2C4.
 - Changed to use ARRAY_SIZE(kLpi2cBases) instead of FEATURE_COUNT to decide the array size for handle pointer array.

[2.1.4]

- Bug Fixes
 - Fixed the LPI2C_MasterTransferEDMA receive issue when LPI2C shared same request source with TX/RX DMA request. Previously, the API used scatter-gather method, which handled the command transfer first, then the linked TCD which was pre-set with the receive data transfer. The issue was that the TX DMA request and the RX DMA request were both enabled, so when the DMA finished the first command TCD transfer and handled the receive data TCD, the TX DMA request still happened due to empty TX FIFO. The result was that the RX DMA transfer would start without waiting on the expected RX DMA request.
 - Fixed the issue by enabling IntMajor interrupt for the command TCD and checking if there was a linked TCD to disable the TX DMA request in LPI2C_MasterEDMACallback API.

[2.1.3]

- Improvements
 - Added LPI2C_WATI_TIMEOUT macro to allow the user to specify the timeout times for waiting flags in functional API and blocking transfer API.
 - Added LPI2C_MasterTransferBlocking API.

[2.1.2]

- Bug Fixes
 - In LPI2C_SlaveTransferHandleIRQ, reset the slave status to idle when stop flag was detected.

[2.1.1]

- Bug Fixes
 - Disabled the auto-stop feature in eDMA driver. Previously, the auto-stop feature was enabled at transfer when transferring with stop flag. Since transfer was without stop flag and the auto-stop feature was enabled, when starting a new transfer with stop flag, the stop flag would be sent before the new transfer started, causing unsuccessful sending of the start flag, so the transfer could not start.
 - Changed default slave configuration with address stall false.

[2.1.0]

- Improvements
 - API name changed:
 - * LPI2C_MasterTransferCreateHandle -> LPI2C_MasterCreateHandle.
 - * LPI2C_MasterTransferGetCount -> LPI2C_MasterGetTransferCount.
 - * LPI2C_MasterTransferAbort -> LPI2C_MasterAbortTransfer.
 - * LPI2C_MasterTransferHandleIRQ -> LPI2C_MasterHandleInterrupt.
 - * LPI2C_SlaveTransferCreateHandle -> LPI2C_SlaveCreateHandle.
 - * LPI2C_SlaveTransferGetCount -> LPI2C_SlaveGetTransferCount.
 - * LPI2C_SlaveTransferAbort -> LPI2C_SlaveAbortTransfer.

* LPI2C_SlaveTransferHandleIRQ -> LPI2C_SlaveHandleInterrupt.

[2.0.0]

- Initial version.
-

LPI2C_EDMA

[2.4.4]

- Improvements
 - Added support for 2KB data transfer

[2.4.3]

- Improvements
 - Added support for separated IRQ handlers.

[2.4.2]

- Improvements
 - Add EDMA ext API to accommodate more types of EDMA.

[2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1
-

LPIT

[2.1.2]

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.1.1]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.1.0]

- Improvements
 - Add new function LPIT_SetTimerValue to set timeout period.

[2.0.2]

- Improvements
 - Improved LPIT_SetTimerPeriod implementation, configure timeout value with LPIT ticks minus 1 generate more correct interval.
 - Added timeout value configuration check for LPIT_SetTimerPeriod, at least input 3 ticks for calling LPIT_SetTimerPeriod.
- Bug Fixes
 - Fixed MISRA C-2012 rule 17.7 violations.

[2.0.1]

- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.

[2.0.0]

- Initial version.
-

LPSPI

[2.7.3]

- Improvements
 - Added timeout for while loop in LPSPI_MasterTransferWriteAllTxData().
 - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.7.2]

- Bug Fixes
 - Fixed coverity issues.

[2.7.1]

- Bug Fixes
 - Workaround for errata ERR050607
 - Workaround for errata ERR010655

[2.7.0]

- New Feature
 - Added common IRQ handler entry LPSPI_DriverIRQHandler.

[2.6.10]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.6.9]

- Bug Fixes
 - Fixed reading of TCR register
 - Workaround for errata ERR050606

[2.6.8]

- Bug Fixes
 - Fixed build error when SPI_RETRY_TIMES is defined to non-zero value.

[2.6.7]

- Bug Fixes
 - Fixed the txData from void * to const void * in transmit API `_lpspi_master_handle` and `_lpspi_slave_handle`.

[2.6.6]

- Bug Fixes
 - Added LPSPI register init in `LPSPI_MasterInit` incase of LPSPI register exist.

[2.6.5]

- Improvements
 - Introduced `FSL_FEATURE_LPSPI_HAS_NO_PCSCFG` and `FSL_FEATURE_LPSPI_HAS_NO_MULTI_WIDTH` for conditional compile.
 - Release peripheral from reset if necessary in init function.

[2.6.4]

- Bug Fixes
 - Added `LPSPI6_DriverIRQHandler` for LPSPI6 instance.

[2.6.3]

- Hot Fixes
 - Added macro switch in function `LPSPI_Enable` about ERRATA051472.

[2.6.2]

- Bug Fixes
 - Disabled `lpspi` before `LPSPI_MasterSetBaudRate` incase of LPSPI opened.

[2.6.1]

- Bug Fixes
 - Fixed return value while calling LPSPI_WaitTxFifoEmpty in function LPSPI_MasterTransferNonBlocking.

[2.6.0]

- Feature
 - Added the new feature of multi-IO SPI .

[2.5.3]

- Bug Fixes
 - Fixed 3-wire txmask of handle vaule reentrant issue.

[2.5.2]

- Bug Fixes
 - Workaround for errata ERR051588 by clearing FIFO after transmit underrun occurs.

[2.5.1]

- Bug Fixes
 - Workaround for errata ERR050456 by resetting the entire module using LPSPI_CR[RST] bit.

[2.5.0]

- Bug Fixes
 - Workaround for errata ERR011097 to wait the TX FIFO to go empty when writing TCR register and TCR[TXMSK] value is 1.
 - Added API LPSPI_WaitTxFifoEmpty for wait the txfifo to go empty.

[2.4.7]

- Bug Fixes
 - Fixed bug that the SR[REF] would assert if software disabled or enabled the LPSPI module in LPSPI_Enable.

[2.4.6]

- Improvements
 - Moved the configuration of registers for the 3-wire lpspi mode to the LPSPI_MasterInit and LPSPI_SlaveInit function.

[2.4.5]

- Improvements
 - Improved LPSPI_MasterTransferBlocking send performance when frame size is 1-byte.

[2.4.4]

- Bug Fixes
 - Fixed LPSPI_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

[2.4.3]

- Bug Fixes
 - Fixed bug that the ISR response speed is too slow on some platforms, resulting in the first transmission of overflow, Set proper RX watermarks to reduce the ISR response times.

[2.4.2]

- Bug Fixes
 - Fixed bug that LPSPI_MasterTransferBlocking will modify the parameter txbuff and rxbuff pointer.

[2.4.1]

- Bug Fixes
 - Fixed bug that LPSPI_SlaveTransferNonBlocking can't detect RX error.

[2.4.0]

- Improvements
 - Split some functions, fixed CCM problem in file fsl_lpspi.c.

[2.3.1]

- Improvements
 - Initialized the EDMA configuration structure in the LPSPI EDMA driver.
- Bug Fixes
 - Fixed bug that function LPSPI_MasterTransferBlocking should return after the transfer complete flag is set to make sure the PCS is re-asserted.

[2.3.0]

- New Features
 - Supported the master configuration of sampling the input data using a delayed clock to improve slave setup time.

[2.2.1]

- Bug Fixes
 - Fixed bug in LPSPI_SetPCSContinuous when disabling PCS continuous mode.

[2.2.0]

- Bug Fixes
 - Fixed bug in 3-wire polling and interrupt transfer that the received data is not correct and the PCS continuous mode is not working.

[2.1.0]

- Improvements
 - Improved LPSPI_SlaveTransferHandleIRQ to fill up TX FIFO instead of write one data to TX register which improves the slave transmit performance.
 - Added new functional APIs LPSPI_SelectTransferPCS and LPSPI_SetPCSContinuous to support changing PCS selection and PCS continuous mode.
- Bug Fixes
 - Fixed bug in non-blocking and EDMA transfer APIs that kStatus_InvalidArgument is returned if user configures 3-wire mode and full-duplex transfer at the same time, but transfer state is already set to kLPSPI_Busy by mistake causing following transfer can not start.
 - Fixed bug when LPSPI slave using EDMA way to transfer, tx should be masked when tx data is null, otherwise in 3-wire mode which tx/rx use the same pin, the received data will be interfered.

[2.0.5]

- Improvements
 - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
 - Fixed the bug that LPSPI can not transfer large data using EDMA.
 - Fixed MISRA 17.7 issues.
 - Fixed variable overflow issue introduced by MISRA fix.
 - Fixed issue that rxFifoMaxBytes should be calculated according to transfer width rather than FIFO width.
 - Fixed issue that completion flag was not cleared after transfer completed.

[2.0.4]

- Bug Fixes
 - Fixed in LPSPI_MasterTransferBlocking that master rxfifo may overflow in stall condition.
 - Eliminated IAR Pa082 warnings.
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 10.6, 11.9, 14.2, 14.4, 15.7, 17.7.

[2.0.3]

- Bug Fixes
 - Removed LPSPI_Reset from LPSPI_MasterInit and LPSPI_SlaveInit, because this API may glitch the slave select line. If needed, call this function manually.

[2.0.2]

- New Features
 - Added dummy data set up API to allow users to configure the dummy data to be transferred.
 - Enabled the 3-wire mode, SIN and SOUT pins can be configured as input/output pin.

[2.0.1]

- Bug Fixes
 - Fixed the bug that the clock source should be divided by the PRESCALE setting in LP-SPI_MasterSetDelayTimes function.
 - Fixed the bug that LPSPI_MasterTransferBlocking function would hang in some corner cases.
- Optimization
 - Added `#ifndef/#endif` to allow user to change the default TX value at compile time.

[2.0.0]

- Initial version.
-

LPSPI_EDMA

[2.4.9]

- Improvements
 - Removed unused code from `LPSPI_SeparateEdmaReadData()`.

[2.4.8]

- Improvements
 - Added timeout for while loop in `EDMA_LpspiMasterCallback()` and `EDMA_LpspiSlaveCallback()`.

[2.4.7]

- Bug Fixes
 - Add macro `LPSPI_ALIGN_TCD_SIZE_MASK` to align an address to `edma_tcd_t` size.

[2.4.6]

- Improvements
 - Increased transmit FIFO watermark to ensure whole transmit FIFO will be used during data transfer.

[2.4.5]

- Bug Fixes
 - Fixed reading of TCR register
 - Workaround for errata ERR050606

[2.4.4]

- Improvements
 - Add EDMA ext API to accommodate more types of EDMA.

[2.4.3]

- Improvements
 - Supported 32K bytes transmit in DMA, improve the max datasize in LP-SPI_MasterTransferEDMALite.

[2.4.2]

- Improvements
 - Added callback status in EDMA_LpspiMasterCallback and EDMA_LpspiSlaveCallback to check transferDone.

[2.4.1]

- Improvements
 - Add the TXMSK wait after TCR setting.

[2.4.0]

- Improvements
 - Separated LPSPI_MasterTransferEDMA functions to LP-SPI_MasterTransferPrepareEDMA and LPSPI_MasterTransferEDMALite to optimize the process of transfer.
-

LPTMR

[2.2.1]

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.2.0]

- Improvements
 - Updated lptmr_prescaler_clock_select_t, only define the valid options.

[2.1.1]

- Improvements
 - Updated the characters from “PTMR” to “LPTMR” in “FSL_FEATURE_PTMR_HAS_NO_PRESCALER_CLOCK_SOURCE_1_SUPPORT” feature definition.

[2.1.0]

- Improvements
 - Implement for some special devices’ not supporting for all clock sources.
- Bug Fixes
 - Fixed issue when accessing CMR register.

[2.0.2]

- Bug Fixes
 - Fixed MISRA-2012 issues.
 - * Rule 10.1.

[2.0.1]

- Improvements
 - Updated the LPTMR driver to support 32-bit CNR and CMR registers in some devices.

[2.0.0]

- Initial version.
-

LPUART

[2.10.0]

- New Feature
 - Added support to configure RTS watermark.

[2.9.4]

- Improvements
 - Merged duplicate code.

[2.9.3]

- Improvements
 - Added timeout for while loops in LPUART_Deinit().

[2.9.2]

- Bug Fixes
 - Fixed coverity issues.

[2.9.1]

- Bug Fixes
 - Fixed coverity issues.

[2.9.0]

- New Feature
 - Added support for swap TXD and RXD pins.
 - Added common IRQ handler entry LPUART_DriverIRQHandler.

[2.8.3]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.8.2]

- Bug Fix
 - Fixed the bug that LPUART_TransferEnable16Bit controlled by wrong feature macro.

[2.8.1]

- Bug Fixes
 - Fixed issue for MISRA-2012 check.
 - * Fixed rule-5.3, rule-5.8, rule-10.4, rule-11.3, rule-11.8.

[2.8.0]

- Improvements
 - Added support of DATA register for 9bit or 10bit data transmit in write and read API. Such as: LPUART_WriteBlocking16bit, LPUART_ReadBlocking16bit, LPUART_TransferEnable16Bit, LPUART_WriteNonBlocking16bit, LPUART_ReadNonBlocking16bit.

[2.7.7]

- Bug Fixes
 - Fixed the bug that baud rate calculation overflow when srcClock_Hz is 528MHz.

[2.7.6]

- Bug Fixes
 - Fixed LPUART_EnableInterrupts and LPUART_DisableInterrupts bug that blocks if the LPUART address doesn't support exclusive access.

[2.7.5]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.7.4]

- Improvements
 - Added support for atomic register accessing in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.7.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 15.7.

[2.7.2]

- Bug Fix
 - Fixed the bug that the OSR calculation error when lpuart init and lpuart set baud rate.

[2.7.1]

- Improvements
 - Added support for LPUART_BASE_PTRS_NS in security mode in file fsl_lpuart.c.

[2.7.0]

- Improvements
 - Split some functions, fixed CCM problem in file fsl_lpuart.c.

[2.6.0]

- Bug Fixes
 - Fixed bug that when there are multiple lpuart instance, unable to support different ISR.

[2.5.3]

- Bug Fixes
 - Fixed comments by replacing unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag with kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag.

[2.5.2]

- Bug Fixes
 - Fixed bug that when setting watermark for TX or RX FIFO, the value may exceed the maximum limit.
- Improvements
 - Added check in LPUART_TransferDMAHandleIRQ and LPUART_TransferEdmaHandleIRQ to ensure if user enables any interrupts other than transfer complete interrupt, the dma transfer is not terminated by mistake.

[2.5.1]

- Improvements
 - Use separate data for TX and RX in lpuart_transfer_t.
- Bug Fixes
 - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling LPUART_TransferReceiveNonBlocking, the received data count returned by LPUART_TransferGetReceiveCount is wrong.

[2.5.0]

- Bug Fixes
 - Added missing interrupt enable masks kLPUART_Match1InterruptEnable and kLPUART_Match2InterruptEnable.
 - Fixed bug in LPUART_EnableInterrupts, LPUART_DisableInterrupts and LPUART_GetEnabledInterrupts that the BAUD[LBKDIE] bit field should be soc specific.
 - Fixed bug in LPUART_TransferHandleIRQ that idle line interrupt should be disabled when rx data size is zero.
 - Deleted unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag, since firstly their function are the same as kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag, secondly to obtain them one data word must be read out thus interfering with the receiving process.
 - Fixed bug in LPUART_GetStatusFlags that the STAT[LBKDIF], STAT[MA1F] and STAT[MA2F] should be soc specific.
 - Fixed bug in LPUART_ClearStatusFlags that tx/rx FIFO is reset by mistake when clearing flags.
 - Fixed bug in LPUART_TransferHandleIRQ that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
 - Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
 - Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
 - Added APIs LPUART_GetRxFifoCount/LPUART_GetTxFifoCount to get rx/tx FIFO data count.
 - Added APIs LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
 - Fixed MISRA advisory 17.7 issues.

[2.4.0]

- New Features
 - Added APIs to configure 9-bit data mode, set slave address and send address.

[2.3.1]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.3.0]

- Improvements
 - Modified LPUART_TransferHandleIRQ so that txState will be set to idle only when all data has been sent out to bus.
 - Modified LPUART_TransferGetSendCount so that this API returns the real byte count that LPUART has sent out rather than the software buffer status.
 - Added timeout mechanism when waiting for certain states in transfer driver.

[2.2.8]

- Bug Fixes
 - Fixed issue for MISRA-2012 check.
 - * Fixed rule-10.3, rule-14.4, rule-15.5.
 - Eliminated Pa082 warnings by assigning volatile variables to local variables and using local variables instead.
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 10.8, 14.4, 11.6, 17.7.
- Improvements
 - Added check for kLPUART_TransmissionCompleteFlag in LPUART_WriteBlocking, LPUART_TransferHandleIRQ, LPUART_TransferSendDMACallback and LPUART_SendEDMACallback to ensure all the data would be sent out to bus.
 - Rounded up the calculated sbr value in LPUART_SetBaudRate and LPUART_Init to achieve more accurate baudrate setting. Changed osr from uint32_t to uint8_t since osr's biggest value is 31.
 - Modified LPUART_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.

[2.2.7]

- Bug Fixes
 - Fixed issue for MISRA-2012 check.
 - * Fixed rule-12.1, rule-17.7, rule-14.4, rule-13.3, rule-14.4, rule-10.4, rule-10.8, rule-10.3, rule-10.7, rule-10.1, rule-11.6, rule-13.5, rule-11.3, rule-13.2, rule-8.3.

[2.2.6]

- Bug Fixes
 - Fixed the issue of register's being in repeated reading status while dealing with the IRQ routine.

[2.2.5]

- Bug Fixes
 - Do not set or clear the TIE/RIE bits when using LPUART_EnableTxDMA and LPUART_EnableRxDMA.

[2.2.4]

- Improvements
 - Added hardware flow control function support.
 - Added idle-line-detecting feature in LPUART_TransferNonBlocking function. If an idle line is detected, a callback is triggered with status kStatus_LPUART_IdleLineDetected returned. This feature may be useful when the received Bytes is less than the expected received data size. Before triggering the callback, data in the FIFO (if has FIFO) is read out, and no interrupt will be disabled, except for that the receive data size reaches 0.
 - Enabled the RX FIFO watermark function. With the idle-line-detecting feature enabled, users can set the watermark value to whatever you want (should be less than the RX FIFO size). Data is received and a callback will be triggered when data receive ends.

[2.2.3]

- Improvements
 - Changed parameter type in LPUART_RTOS_Init struct from rtos_lpuart_config to lpuart_rtos_config_t.
- Bug Fixes
 - Disabled LPUART receive interrupt instead of all NVICs when reading data from ring buffer. Otherwise when the ring buffer is used, receive nonblocking method will disable all NVICs to protect the ring buffer. This may has a negative effect on other IPs that are using the interrupt.

[2.2.2]

- Improvements
 - Added software reset feature support.
 - Added software reset API in LPUART_Init.

[2.2.1]

- Improvements
 - Added separate RX/TX IRQ number support.

[2.2.0]

- Improvements
 - Added support of 7 data bits and MSB.

[2.1.1]

- Improvements
 - Removed unnecessary check of event flags and assert in LPUART_RTOS_Receive.
 - Added code to always wait for RX event flag in LPUART_RTOS_Receive.

[2.1.0]

- Improvements
 - Update transactional APIs.
-

LPUART_EDMA

[2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0
-

LTC

[2.0.17]

- Bug fix:
 - Fix CMAC for payloads over one block, and if BRIC is present on the device, remove XCBC and “decrypt key” functionality

[2.0.16]

- Bug fix:
 - Fix uninitialized GCC warning in LTC_AES_GenerateDecryptKey().

[2.0.15]

- Bug fix:
 - Fix MISRA-2012 issues.

[2.0.14]

- Improvements:
 - Add feature macro `FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT` into `LTC_Deinit` function.

[2.0.13]

- Improvements:
 - Add feature macro `FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT` into `LTC_init` function.

[2.0.12]

- Bug fix:
 - Fix AES Decrypt in CBC modes fail when used `kLTC_DeCryptKey`.

[2.0.11]

- Bug fix:
 - Fix MISRA-2012 issues.

[2.0.10]

- Bug fix:
 - Fix MISRA-2012 issues.

[2.0.9]

- Bug fix:
 - Fix sign-compare warning in `ltc_set_context` and in `ltc_get_context`.

[2.0.8]

- Bug fix:
 - Fixed coverity issues introduced in 2.0.7

[2.0.7]

- Bug fix:
 - Fixed MISRA-2012 issues
 - * Fixed rule 15.7. 10.1. 8.6. 10.7. 10.3, 11.6, 11.3, 10.8, 16.1.

[2.0.6]

- Bug fix:
 - Fixed [KPSDK-23603][LTC] AES Decrypt in ECB and CBC modes fail when ciphertext size > 0xff0 bytes.

[2.0.5]

- Improvements:
 - Fixed MISRA issues
 - * Fixed rule 15.7, rule 14.4, rule 10.4, rule 18.4, rule 17.7, 5.7, 12.2.

[2.0.4]

- Improvements:
 - Constant LTC_PKHA_CompareBigNum() processing time.

[2.0.3]

- Bug fix:
 - Fixed LTC_PKHA_CompareBigNum() in case an integer argument is an array of all zeros.

[2.0.2]

- Bug fix:
 - Fixed [KPSDK-10932][LTC][SHA] LTC_HASH() blocks indefinitely when message size exceeds 4080 Bytes.

[2.0.1]

- Bug fix:
 - Fixed c++ build warning in block_to_ififo() functions.

[2.0.0]

- Initial version.
-

MCM

[2.2.0]

- Improvements
 - Support platforms with less features.

[2.1.0]

- Others
 - Remove byteID from mcm_lmem_fault_attribute_t for document update.

[2.0.0]

- Initial version.
-

MSCM

[2.0.0]

- Initial version.
-

MU

[2.8.0]

- New Features
 - Added MU1_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
 - Added timeout mechanism to all polling loops in MU driver code.
- Improvements
 - Updated function signatures to return status codes for better error handling:
 - * Changed MU_ResetBothSides to return status_t instead of void
 - * Updated MU_SendMsg to return status_t for timeout indication
 - * Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
 - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.7.0]

- New Features
 - Added API MU_GetRxStatusFlags.

[2.6.0]

- New Features
 - Added API MU_GetInterruptsPending.

[2.5.1]

- Bug Fixes
 - Fixed the bug that MU_TriggerGeneralPurposeInterrupts and MU_TriggerInterrupts may trigger previous triggered general purpose interrupts again by mistake.

[2.5.0]

- New Features
 - Supported more than 4 general purpose interrupts.
 - Added separate APIs for general purpose interrupts.

[2.4.0]

- Improvements
 - Supported the case that some features only available with specific instances. These features include Hardware Reset, Boot Peer Core, Hold Reset. When using the features with instances which don't support them, driver will report error.

[2.3.3]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.3.2]

- Improvements
 - Supported platforms which don't have CCR0[RSTH], CCR0[CLKE], CCR0[HR], CCR0[HRM].

[2.3.1]

- Bug Fixes
 - Fixed build error for platforms which have CCR0[RSTH], but no CCR0[NMI].

[2.3.0]

- New features
 - Added support for i.MX RT7xx.

[2.2.1]

- Bug Fixes
 - Fixed issue that MU_GetInstance() is defined but never used.

[2.2.0]

- New features
 - Added support for i.MX RT118x.
- Bug Fixes
 - Fixed general purpose interrupt bug.
- Other Changes
 - Change _mu_interrupt_trigger item value.

[2.1.2]

- Bug Fixes
 - Fixed bug that general purpose interrupt can't be configured.

[2.1.1]

- Bug Fixes
 - Fixed MISRA C-2012 issues.

[2.1.0]

- Improvements
 - Added new enum `mu_msg_reg_index_t`.

[2.0.0]

- Initial version.
-

PORT

[2.5.1]

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.5.0]

- Bug Fixes
 - Correct the `kPORT_MuxAsGpio` for some platforms.

[2.4.1]

- Bug Fixes
 - Fixed the violations of MISRA C-2012 rules: 10.1, 10.8 and 14.4.

[2.4.0]

- New Features
 - Updated `port_pin_config_t` to support input buffer and input invert.

[2.3.0]

- New Features
 - Added new APIs for Electrical Fast Transient(EFT) detect.
 - Added new API to configure port voltage range.

[2.2.0]

- New Features
 - Added new api `PORT_EnablePinDoubleDriveStrength`.

[2.1.1]

- Bug Fixes
 - Fixed the violations of MISRA C-2012 rules: 10.1, 10.4□11.3□11.8, 14.4.

[2.1.0]

- New Features
 - Updated the driver code to adapt the case of the interrupt configurations in GPIO module. Will move the pin configuration APIs to GPIO module.

[2.0.2]

- Other Changes
 - Added feature guard macros in the driver.

[2.0.1]

- Other Changes
 - Added “const” in function parameter.
 - Updated some enumeration variables’ names.
-

RTC

[2.4.0]

- New features
 - Add support for RTC clock output.
 - Add support for RTC time seconds interrupt configuration.

[2.3.3]

- Bug Fixes
 - Fix RTC_GetDatetime function validating datetime issue.

[2.3.2]

- Improvements
 - Handle errata 010716: Disable the counter before setting alarm register and then reen-able the counter.

[2.3.1]

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.3.0]

- Improvements
 - Added API RTC_EnableLPOClock to set 1kHz LPO clock.
 - Added API RTC_EnableCrystalClock to replace API RTC_SetClockSource.

[2.2.2]

- Improvements
 - Refine _rtc_interrupt_enable order.

[2.2.1]

- Bug Fixes
 - Fixed the issue of Pa082 warning.
 - Fixed the issue of bit field mask checking.
 - Fixed the issue of hard code in RTC_Init.

[2.2.0]

- Bug Fixes
 - Fixed MISRA C-2012 issue.
 - * Fixed rule contain: rule-17.7, rule-14.4, rule-10.4, rule-10.7, rule-10.1, rule-10.3.
 - Fixed central repository code formatting issue.
- Improvements
 - Added an API for enabling wakeup pin.

[2.1.0]

- Improvements
 - Added feature macro check for many features.

[2.0.0]

- Initial version.
-

SEMA42

[2.1.1]

- Improvements
 - Updated SEMA42_TryLock function to avoid unsigned integer operations wrap issue.

[2.1.0]

- New Features
 - Added SEMA42_BUSY_POLL_COUNT parameter to prevent infinite polling loops in SEMA42 operations.
 - Added timeout mechanism to all polling loops in SEMA42 driver code.
- Improvements
 - Updated SEMA42_Lock function to return status_t instead of void for better error handling.
 - Enhanced documentation to clarify timeout behavior and return values.

[2.0.4]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.0.3]

- Improvements
 - Changed to implement SEMA42_Lock base on SEMA42_TryLock.

[2.0.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 17.7.

[2.0.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 18.1.

[2.0.0]

- Initial version.
-

SFA

[2.1.3]

- Improvements
 - Add timeout for APIs with dfmea issues.

[2.1.2]

- Improvements
 - Updated SFA_ClearStatusFlag() function to support clearing all status flags.

[2.1.1]

- Improvements
 - Updated SFA driver code to reduce code redundancy.
 - Updated the use of macros in the driver code to enable the CUT pin function.

[2.1.0]

- Improvements
 - Updated how the clock frequency under test is calculated.
 - Supported configuration reference clock source.
 - Added clock count limit APIs.
 - Added macros to support fields that are configurable in different instances.

[2.0.1]

- Bug Fixes
 - Fixed the issue in SFA_Mode0Calculate function.
- Improvements
 - For the devices that support RF SFA, cast the peripheral structure to SFA_Type.

[2.0.0]

- Initial version.
-

SMSCM

[2.0.0]

- Initial version.
-

SPC

[2.8.0]

- Improvements
 - Added feature macros to support some platforms which do not have system LDO, VDD_SYS, DCDC_BURST_CFG.

[2.7.1]

- Improvements
 - Make configuration structure pointers const in SPC driver API for Higher Power mode

[2.7.0]

- Improvements
 - Added new APIs to control VDD Core Glitch Detector.

[2.6.1]

- Improvements
 - Add timeout for while loop code.

[2.6.0]

- New Features
 - Added new feature FSL_FEATURE_SPC_HAS_SC_SPC_LP_REQ_BIT for compatibility with new platforms that do not support the SPC SC[SPC_LP_REQ] bitfield.

[2.5.0]

- New Features
 - Added new function group to support high power feature.

[2.4.1]

- Improvements
 - Added the new feature FSL_FEATURE_SPC_HAS_SC_REG_BUSY to support the device which has the SC[REG_BUSY] bitfield.

[2.4.0]

- Improvements
 - Set functions of VD_SYS_CFG[LVSEL] configuration as deprecated, since this bit field is reserved for all devices.

[2.3.0]

- Improvements
 - Added SPC_SetSRAMOperateVoltage() to set SRAM operate voltage.
- Bug Fixes
 - Fixed the issue in SPC_SetDCDCBurstConfig() function.
 - Fixed a bug that is SPC has VDD_DS the voltage is not configured correctly.

[2.2.0]

- Bug Fixes
 - The default value of ACTIVE_CFG[CORELDO_VDD_DS] bit is 1, removed code of set that bit.
 - Fixed an issue of SPC_SetDCDCBurstConfig() to avoid program stuck in a while loop.
 - Removed some duplicate code that never reached.

- Improvements
 - Added two APIs to control Vdd core glitch detect feature in Active/lowpower modes.

[2.1.0]

- Improvements
 - Improved the `spc_dcdc_voltage_level_t` enumeration.
 - Added some low power request check APIs.
 - Added `SPC_SoftwareGatePowerSwitch()` and `SPC_PowerModeControlPowerSwitch()` function.
 - If APIs' parameters is in type of structure pointer, set it to const structure pointer.
- Bug Fixes
 - Fixed the issue in `SPC_SetDCDCRefreshCount()` function.
 - Fixed some issues in voltage detect enable APIs.
 - Fixed some issues in Bandgap control APIs.
 - Fixed some issues in regulator active/lowpower mode configuration APIs.

[2.0.2]

- Bug Fixes
 - Fixed the doxygen issue by adding missed comments.

[2.0.1]

- Bug Fixes
 - The enumerator `spc_core_ldo_voltage_level_t` may different for different devices, added the macro to separate different scenes.

[2.0.0]

- Initial version.
-

SYSPM

[2.3.1]

- Improvements
 - Add timeout for while loop.

[2.3.0]

- New Features
 - Supported instruction counter.

[2.2.0]

- New Features
 - Supported platforms which only have one monitor.
 - Supported platforms whose instruction counter can't be cleared.
 - Supported platforms whose counter can't be disabled.

[2.1.0]

- New Features
 - Added new APIs SYSPM_Init and SYSPM_Deinit.

[2.0.0]

- Initial version.
-

TPM

[2.4.1]

- Improvements
 - Add Coverage Justification for uncovered code.

[2.4.0]

- New Feature
 - Added while loop timeout for MOD CnV CnSC and SC register write sequence.
 - Change the return type from void to status_t for following API:
 - * TPM_DisableChannel
 - * TPM_EnableChannel
 - * TPM_SetupOutputCompare
 - * TPM_SetTimerPeriod
 - * TPM_StopTimer

[2.3.6]

- Bug Fixes
 - Fixed CERT INT30-C INT31-C issue for TPM_SetupDualEdgeCapture.

[2.3.5]

- New Feature
 - Added IRQ handler entry for TPM2.

[2.3.4]

- New Feature
 - Added common IRQ handler entry TPM_DriverIRQHandler.

[2.3.3]

- Improvements
 - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.3.2]

- Bug Fixes
 - Fixed ERR008085 TPM writing the TPMx_MOD or TPMx_CnV registers more than once may fail when the timer is disabled.

[2.3.1]

- Bug Fixes
 - Fixed compilation error when macro FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is 1.

[2.3.0]

- Improvements
 - Create callback feature for TPM match and timer overflow interrupts.

[2.2.4]

- Improvements
 - Add feature macros(FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_EN, FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_SYNC).

[2.2.3]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.2.2]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.4.

[2.2.1]

- Bug Fixes
 - Fixed CCM issue by splitting function from TPM_SetupPwm() function to reduce function complexity.
 - Fixed violations of MISRA C-2012 rule 17.7.

[2.2.0]

- Improvements
 - Added TPM_SetChannelPolarity to support select channel input/output polarity.
 - Added TPM_EnableChannelExtTrigger to support enable external trigger input to be used by channel.
 - Added TPM_CalculateCounterClkDiv to help calculates the counter clock prescaler.
 - Added TPM_GetChannelValue to support get TPM channel value.
 - Added new TPM configuration.
 - * syncGlobalTimeBase
 - * extTriggerPolarity
 - * chnlPolarity
 - Added new PWM signal configuration.
 - * secPauseLevel
- Bug Fixes
 - Fixed TPM_SetupPwm can't configure 0% combined PWM issues.

[2.1.1]

- Improvements
 - Add feature macro for PWM pause level select feature.

[2.1.0]

- Improvements
 - Added TPM_EnableChannel and TPM_DisableChannel APIs.
 - Added new PWM signal configuration.
 - * pauseLevel - Support select output level when counter first enabled or paused.
 - * enableComplementary - Support enable/disable generate complementary PWM signal.
 - * deadTimeValue - Support deadtime insertion for each pair of channels in combined PWM mode.
- Bug Fixes
 - Fixed issues about channel MSnB:MSnA and ELSnB:ELSnA bit fields and CnV register change request acknowledgement. Writes to these bits are ignored when the interval between successive writes is less than the TPM clock period.

[2.0.8]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.1, 10.4 ,10.7 and 14.4.

[2.0.7]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.4 and 17.7.

[2.0.6]

- Bug Fixes
 - Fixed Out-of-bounds issue.

[2.0.5]

- Bug Fixes
 - Fixed MISRA-2012 rules.
 - * Rule 10.6, 10.7

[2.0.4]

- Bug Fixes
 - Fixed ERR050050 in functions TPM_SetupPwm/TPM_UpdatePwmDutycycle. When TPM was configured in EPWM mode as PS = 0, the compare event was missed on the first reload/overflow after writing 1 to the CnV register.

[2.0.3]

- Bug Fixes
 - MISRA-2012 issue fixed.
 - * Fixed rules: rule-12.1, rule-17.7, rule-16.3, rule-14.4, rule-1.3, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-10.6, and rule-18.1.

[2.0.2]

- Bug Fixes
 - Fixed issues in functions TPM_SetupPwm/TPM_UpdateChnlEdgeLevelSelect/TPM_SetupInputCapture/TPM_SetupOutputCompare/TPM_SetupDualEdgeCapture, wait acknowledgement when the channel is disabled.

[2.0.1]

- Bug Fixes
 - Fixed TPM_UpdateChnlEdgeLevelSelect ACK wait issue.
 - Fixed the issue that TPM_SetupDualEdgeCapture could not set FILTER register.
 - Fixed TPM_UpdateChnlEdgeLevelSelect ACK wait issue.

[2.0.0]

- Initial version.
-

TRDC

[2.3.0]

- New Features
 - Added API TRDC_EnableProcessorDomainAssignment to disable/enable DAC.
- Bug Fixes
 - Fixed MISRA violation of missing function declaration.
 - Fixed wrong operation of domain mask in TRDC_MbcNseClearAll and TRDC_MrcDomainNseClear.

[2.2.1]

- Bug Fixes
 - Fixed MISRA violations of rule 10.3.

[2.2.0]

- New Features
 - Added new APIs to support SoC with more than one processor core.

[2.1.1]

- Bug Fixes
 - Fixed MISRA violations of rule 10.3, 10.6, 10.7 and 18.1.

[2.1.0]

- Improvements
 - Modified some of the addressing method according to the device header file's update.
 - Modified flash address configuration structure and default setting.
- New Features
 - Added API to set flash logic window array address.
- Bug Fixes
 - Fixed wrong return value of TRDC_GetFlashLogicalWindowPbase.
 - Fixed bug in TRDC_GetAndClearFirstSpecificDomainError that the error status is cleared by mistake before obtained by software.
 - Fixed MISRA violations of rule 10.3, 10.4, 10.6 and 10.8.

[2.0.0]

- Initial version.
-

TRGMUX

[2.0.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.8.

[2.0.0]

- Initial version.
-

TSTMR

[2.1.0]

- New Features
 - Support configured clock frequency.
 - Add TSTMR_Init and TSTMR_init APIs.
- Improvements
 - Change TSTMR_DelayUs from static inline function to normal function.

[2.0.4]

- Bugfix
 - Fix MISRA C-2012 Rule 10.4 and 14.4 issues.

[2.0.3]

- Bugfix
 - Fix CERT INT30-C that Unsigned integer operation TSTMR_ReadTimeStamp(base) - startTime may wrap.

[2.0.2]

- Improvements
 - Support 24MHz clock source.
- Bugfix
 - Fix MISRA C-2012 Rule 10.4 issue.
 - Read of TSTMR HIGH must follow TSTMR LOW atomically: require masking interrupt around 2 LSB / MSB accesses.

[2.0.1]

- Bugfix
 - Restrict to read with 32-bit accesses only.
 - Restrict that TSTMR LOW read occurs first, followed by the TSTMR HIGH read.

[2.0.0]

- Initial version.
-

VBAT

[2.1.1]

- Improvements
 - Added timeout for while loop.

[2.1.0]

- Improvements
 - Both FROCFGA and LDOCFGA registers are set as internal register, removed corresponding interfaces.

[2.0.0]

- Initial version.
-

VREF

[2.4.0]

- Improvements
 - Support TEST_UNLOCK and TRIM0 registers read and write.

[2.3.0]

- Improvements
 - Add feature macros for compatibility with some platforms that don't have CSR[LDBG_BUF_EN] and UTRIM[TRIM2V1].

[2.2.2]

- Bug Fixes
 - Fixed typo in vref_buffer_mode_t enumerator.

[2.2.1]

- Improvements
 - Updated VREF driver code to reduce code redundancy.

[2.2.0]

- Improvements
 - Remove API VREF_SetTrimVal and VREF_GetTrimVal
 - Add new API VREF_SetVrefTrimVal,VREF_SetTrim21Val,VREF_GetVrefTrimVal and VREF_GetTrim21Val
 - Updated VREF driver code to conform to VREF IP architecture.

[2.1.0]

- Improvements
 - Remove API VREF_SetTrimVal and VREF_GetTrimVal
 - Add new API VREF_SetVrefTrimVal,VREF_SetTrim21Val,VREF_GetVrefTrimVal and VREF_GetTrim21Val
 - Updated VREF driver code to conform to VREF IP architecture.
- Bug Fixes
 - Fix issue which progeam stuck in VREF_Init

[2.0.0]

- Initial version.
-

WDOG32

[2.2.1]

- Bug Fixes
 - Fix CERT INT31-C that the bool value shall be converted to unsigned int 0 or 1 then passed to registers.
 - Fix MISRA 2012 20.3 vilation.

[2.2.0]

- Improvements
 - Added while loop timeout config value for WDOG32 reconfiguration and unlock sequence.
 - Change the return type of WDOG32_Init, WDOG32_Deinit and WDOG32_Unlock from void to status_t.

[2.1.0]

- Improvements
 - Release peripheral from reset if necessary in init function.

[2.0.4]

- Improvements
 - To ensure that the reconfiguration is inside 128 bus clocks unlock window, put all re-configuration APIs in quick access code section.

[2.0.3]

- Bug Fixes
 - Fixed the noncompliance issue of the reference document.
 - * Waited until for new configuration to take effect by checking the RCS bit field.
 - * Waited until for registers to be unlocked by checking the ULK bit field.
- Improvements
 - Added 128 bus clocks delay ensures a smooth transition before restarting the counter with the new configuration when there is no RCS status bit.

[2.0.2]

- Bug Fixes
 - MISRA C-2012 issue fixed.
 - * Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.
 - Fixed the issue of the inseparable process interrupted by other interrupt source.
 - * WDOG32_Refresh

[2.0.1]

- Bug Fixes
 - WDOG must be configured within its configuration time period.
 - * Added WDOG32_Init API to quick access section.
 - * Defined register variable in WDOG32_Init API.

[2.0.0]

- Initial version.
-

WUU

[2.4.0]

- New Features
 - Added WUU_ClearExternalWakeupPinsConfig() to clear settings of PDC and PE register.

[2.3.0]

- New Features
 - Added WUU_ClearInternalWakeUpModulesConfig() to clear settings of DM and ME register.

[2.2.1]

- Bug Fixes
 - Fixed WUU_SetPinFilterConfig() unable to set edge detection of pin filter config.
 - Fixed wrong macro used in WUU_GetPinFilterFlag() function.

[2.2.0]

- New Features
 - Added the WUU_GetExternalWakeupPinFlag() and WUU_ClearExternalWakeupPinFlag() function .

[2.1.0]

- New Features
 - Added the WUU_GetModuleInterruptFlag() function to support the devices that equipped MF register.

[2.0.0]

- Initial version.
-

1.6 Driver API Reference Manual

This section provides a link to the Driver API RM, detailing available drivers and their usage to help you integrate hardware efficiently.

[KW47B42ZB7](#)

1.7 Middleware Documentation

Find links to detailed middleware documentation for key components. While not all onboard middleware is covered, this serves as a useful reference for configuration and development.

1.7.1 Wireless Bluetooth LE host stack and applications

[NXP Bluetooth LE Host and Sample Applications](#)

1.7.2 Wireless Connectivity Framework

[framework](#)

1.7.3 Multicore

[Multicore SDK](#)

1.7.4 FreeMASTER

freemaster

1.7.5 FreeRTOS

FreeRTOS

Chapter 2

KW47B42ZB7

2.1 CACHE: LPCAC CACHE Memory Controller

`static inline void L1CACHE_EnableCodeCache(void)`

Enables the processor code bus cache.

`static inline void L1CACHE_DisableCodeCache(void)`

Disables the processor code bus cache.

`static inline void L1CACHE_InvalidateCodeCache(void)`

Invalidates the processor code bus cache.

`void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)`

Invalidates L1 instrument cache by range.

Parameters

- `address` – The start address of the memory to be invalidated.
- `size_byte` – The memory size.

`static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)`

Invalidates L1 data cache by range.

Parameters

- `address` – The start address of the memory to be invalidated.
- `size_byte` – The memory size.

`static inline void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)`

Cleans L1 data cache by range.

The cache is write through mode, so there is nothing to do with the cache flush/clean operation.

Parameters

- `address` – The start address of the memory to be cleaned.
- `size_byte` – The memory size.

`static inline void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)`

Cleans and Invalidates L1 data cache by range.

Parameters

- `address` – The start address of the memory to be clean and invalidated.

- size_byte – The memory size.

```
static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)
```

Invalidates instruction cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

```
static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)
```

Invalidates data cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

```
static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)
```

Clean data cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned.

```
static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)
```

Cleans and Invalidates data cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated.

```
FSL_CACHE_DRIVER_VERSION
```

cache driver version 2.1.2.

2.2 CCM32K: 32kHz Clock Control Module

```
void CCM32K_Enable32kFro(CCM32K_Type *base, bool enable)
```

Enable/Disable 32kHz free-running oscillator.

Note: There is a start up time before clocks are output from the FRO.

Note: To enable FRO32k and set it as 32kHz clock source please follow steps:

```
CCM32K_Enable32kFro(base, true); //Enable FRO analog oscillator.
CCM32K_DisableCLKOutToPeripherals(base, mask); //Disable clock out.
CCM32K_SelectClockSource(base, kCCM32K_ClockSourceSelectFro32k); //Select FRO32k as clock_
↔source.
while(CCM32K_GetStatus(base) != kCCM32K_32kFroActiveStatusFlag); //Check FOR32k is active_
↔and in used.
CCM32K_EnableCLKOutToPeripherals(base, mask); //Enable clock out if needed.
```

Parameters

- base – CCM32K peripheral base address.

- `enable` – Boolean value to enable or disable the 32kHz free-running oscillator. `true` — Enable 32kHz free-running oscillator. `false` — Disable 32kHz free-running oscillator.

```
static inline void CCM32K_Lock32kFroWriteAccess(CCM32K_Type *base)
```

Lock all further write accesses to the FRO32K_CTRL register until a POR occurs.

Parameters

- `base` – CCM32K peripheral base address.

```
static inline uint16_t CCM32K_Get32kFroTrimValue(CCM32K_Type *base)
```

Get frequency trim value of 32kHz free-running oscillator.

Parameters

- `base` – CCM32K peripheral base address.

Returns

The current trim value.

```
void CCM32K_Set32kFroTrimValue(CCM32K_Type *base, uint16_t trimValue)
```

Set the frequency trim value of 32kHz free-running oscillator by software.

Note: The frequency is decreased monotonically when the trimValue is changed progressively from 0x0U to 0x7FFU.

Note: If the FRO32 is enabled before invoking this function, then in this function the FRO32 will be disabled, after updating trim value the FRO32 will be re-enabled.

Parameters

- `base` – CCM32K peripheral base address.
- `trimValue` – The frequency trim value.

```
static inline void CCM32K_Disable32kFroIFRLoad(CCM32K_Type *base, bool disable)
```

Disable/Enable the function of setting 32kHz free-running oscillator trim value when IFR value gets loaded in the SOC.

Parameters

- `base` – CCM32K peripheral base address.
- `disable` – Boolean value to disable or enable IFR loading function. `true` — Disable IFR loading function. `false` — Enable IFR loading function.

```
static inline void CCM32K_Lock32kFroTrimWriteAccess(CCM32K_Type *base)
```

Lock all further write accesses to the FRO32K_TRIM register until a POR occurs.

Parameters

- `base` – CCM32K peripheral base address.

```
void CCM32K_Set32kOscConfig(CCM32K_Type *base, ccm32k_osc_mode_t mode, const  
                           ccm32k_osc_config_t *config)
```

Config 32k Crystal Oscillator.

Note: When the mode selected as `kCCM32K_Disable32kHzCrystalOsc` or `kCCM32K_Bypass32kHzCrystalOsc` the parameter config is useless, so it can be set as

“NULL”.

Note: To enable OSC32K and select it as clock source of 32kHz please follow steps:

```
CCM32K_Set32kOscConfig(base, kCCM32K_Enable32kHzCrystalOsc, config); //Enable OSC32k and
↪set config.
while((CCM32K_GetStatus(base) & kCCM32K_32kOscReadyStatusFlag) == 0UL); //Check if
↪OSC32K is stable.
CCM32K_DisableCLKOutToPeripherals(base, mask); //Disable clock out.
CCM32K_SelectClockSource(base, kCCM32K_ClockSourceSelectOsc32k); //Select OSC32k as clock
↪source.
while((CCM32K_GetStatus(base) & kCCM32K_32kOscActiveStatusFlag) == 0UL); //Check if
↪OSC32K is used as clock source.
CCM32K_EnableCLKOutToPeripherals(base, mask); //Enable clock out.
```

Parameters

- base – CCM32K peripheral base address.
- mode – The mode of 32k crystal oscillator.
- config – The pointer to the structure `ccm32k_osc_config_t`.

```
static inline void CCM32K_Lock32kOscWriteAccess(CCM32K_Type *base)
```

Lock all further write accesses to the OSC32K_CTRL register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

```
void CCM32K_EnableClockMonitor(CCM32K_Type *base, bool enable)
```

Enable/disable clock monitor.

Parameters

- base – CCM32K peripheral base address.
- enable – Used to enable/disable clock monitor.
 - **turn** Enable clock monitor.
 - **false** Disable clock monitor.

```
static inline void CCM32K_SetClockMonitorFreqTrimValue(CCM32K_Type *base,
                                                         ccm32k_clock_monitor_freq_trim_value_t
                                                         trimValue)
```

Set clock monitor frequency trim value.

Parameters

- base – CCM32K peripheral base address.
- trimValue – Clock minitor frequency trim value, please refer to `ccm32k_clock_monitor_freq_trim_value_t`.

```
static inline void CCM32K_SetClockMonitorDivideTrimValue(CCM32K_Type *base,
                                                          ccm32k_clock_monitor_divide_trim_value_t
                                                          trimValue)
```

Set clock monitor divide trim value.

Parameters

- base – CCM32K peripheral base address.

- trimValue – Clock minitor divide trim value, please refer to `ccm32k_clock_monitor_divide_trim_value_t`.

```
void CCM32K_SetClockMonitorConfig(CCM32K_Type *base, const
                                ccm32k_clock_monitor_config_t *config)
```

Config clock monitor one time, including frequency trim value, divide trim value.

Parameters

- base – CCM32K peripheral base address.
- config – Pointer to `ccm32k_clock_monitor_config_t` structure.

```
static inline void CCM32K_LockClockMonitorWriteAccess(CCM32K_Type *base)
```

Lock all further write accesses to the CLKMON_CTRL register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

```
static inline void CCM32K_EnableCLKOutToPeripherals(CCM32K_Type *base, uint8_t
                                                    peripheralMask)
```

Enable 32kHz clock output to selected peripherals.

Parameters

- base – CCM32K peripheral base address.
- peripheralMask – The mask of peripherals to enable 32kHz clock output, should be the OR'ed value of `ccm32k_clock_output_peripheral_t`.

```
static inline void CCM32K_DisableCLKOutToPeripherals(CCM32K_Type *base, uint8_t
                                                    peripheralMask)
```

Disable 32kHz clock output to selected peripherals.

Parameters

- base – CCM32K peripheral base address.
- peripheralMask – The mask of peripherals to disable 32kHz clock output, should be the OR'ed value of `ccm32k_clock_output_peripheral_t`.

```
static inline void CCM32K_SelectClockSource(CCM32K_Type *base,
                                           ccm32k_clock_source_select_t clockSource)
```

Select CCM32K module's clock source which will be provide to the device.

Parameters

- base – CCM32K peripheral base address.
- clockSource – Used to select clock source, please refer to `ccm32k_clock_source_select_t` for details.

```
static inline void CCM32K_LockClockGateWriteAccess(CCM32K_Type *base)
```

Lock all further write access to the CGC32K register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

```
static inline uint32_t CCM32K_GetStatusFlag(CCM32K_Type *base)
```

Get the status flag.

Parameters

- base – CCM32K peripheral base address.

Returns

The status flag of the current node. The enumerator of status flags have been provided, please see the Enumerations title for details.

`ccm32k_state_t` CCM32K_GetCurrentState(CCM32K_Type *base)

Get current state.

Parameters

- base – CCM32K peripheral base address.

Returns

The CCM32K's current state, please refer to `ccm32k_state_t` for details.

`ccm32k_clock_source_t` CCM32K_GetClockSource(CCM32K_Type *base)

Return current clock source.

Parameters

- base – CCM32K peripheral base address.

Return values

- `kCCM32K_ClockSourceNone` – The none clock source is selected.
- `kCCM32K_ClockSource32kFro` – 32kHz free-running oscillator is selected as clock source.
- `kCCM32K_ClockSource32kOsc` – 32kHz crystal oscillator is selected as clock source..

FSL_CCM32K_DRIVER_VERSION

CCM32K driver version 2.2.0.

enum `_ccm32k_osc_xtal_cap`

The enumerator of internal capacitance of OSC's XTAL pin.

Values:

enumerator `kCCM32K_OscXtal0pFCap`

The internal capacitance for XTAL pin is 0pF.

enumerator `kCCM32K_OscXtal2pFCap`

The internal capacitance for XTAL pin is 2pF.

enumerator `kCCM32K_OscXtal4pFCap`

The internal capacitance for XTAL pin is 4pF.

enumerator `kCCM32K_OscXtal6pFCap`

The internal capacitance for XTAL pin is 6pF.

enumerator `kCCM32K_OscXtal8pFCap`

The internal capacitance for XTAL pin is 8pF.

enumerator `kCCM32K_OscXtal10pFCap`

The internal capacitance for XTAL pin is 10pF.

enumerator `kCCM32K_OscXtal12pFCap`

The internal capacitance for XTAL pin is 12pF.

enumerator `kCCM32K_OscXtal14pFCap`

The internal capacitance for XTAL pin is 14pF.

enumerator `kCCM32K_OscXtal16pFCap`

The internal capacitance for XTAL pin is 16pF.

enumerator kCCM32K_OscXtal18pFCap

The internal capacitance for XTAL pin is 18pF.

enumerator kCCM32K_OscXtal20pFCap

The internal capacitance for XTAL pin is 20pF.

enumerator kCCM32K_OscXtal22pFCap

The internal capacitance for XTAL pin is 22pF.

enumerator kCCM32K_OscXtal24pFCap

The internal capacitance for XTAL pin is 24pF.

enumerator kCCM32K_OscXtal26pFCap

The internal capacitance for XTAL pin is 26pF.

enumerator kCCM32K_OscXtal28pFCap

The internal capacitance for XTAL pin is 28pF.

enumerator kCCM32K_OscXtal30pFCap

The internal capacitance for XTAL pin is 30pF.

enum _ccm32k_osc_extal_cap

The enumerator of internal capacitance of OSC's EXTAL pin.

Values:

enumerator kCCM32K_OscExtal0pFCap

The internal capacitance for EXTAL pin is 0pF.

enumerator kCCM32K_OscExtal2pFCap

The internal capacitance for EXTAL pin is 2pF.

enumerator kCCM32K_OscExtal4pFCap

The internal capacitance for EXTAL pin is 4pF.

enumerator kCCM32K_OscExtal6pFCap

The internal capacitance for EXTAL pin is 6pF.

enumerator kCCM32K_OscExtal8pFCap

The internal capacitance for EXTAL pin is 8pF.

enumerator kCCM32K_OscExtal10pFCap

The internal capacitance for EXTAL pin is 10pF.

enumerator kCCM32K_OscExtal12pFCap

The internal capacitance for EXTAL pin is 12pF.

enumerator kCCM32K_OscExtal14pFCap

The internal capacitance for EXTAL pin is 14pF.

enumerator kCCM32K_OscExtal16pFCap

The internal capacitance for EXTAL pin is 16pF.

enumerator kCCM32K_OscExtal18pFCap

The internal capacitance for EXTAL pin is 18pF.

enumerator kCCM32K_OscExtal20pFCap

The internal capacitance for EXTAL pin is 20pF.

enumerator kCCM32K_OscExtal22pFCap

The internal capacitance for EXTAL pin is 22pF.

enumerator kCCM32K_OscExtal24pFCap

The internal capacitance for EXTAL pin is 24pF.

enumerator kCCM32K_OscExtal26pFCap

The internal capacitance for EXTAL pin is 26pF.

enumerator kCCM32K_OscExtal28pFCap

The internal capacitance for EXTAL pin is 28pF.

enumerator kCCM32K_OscExtal30pFCap

The internal capacitance for EXTAL pin is 30pF.

enum _ccm32k_osc_fine_adjustment_value

The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by modifying the automatic gain control (AGC).

Values:

enumerator kCCM32K_OscFineAdjustmentRange0

enum _ccm32k_osc_coarse_adjustment_value

The enumerator of osc amplifier coarse fine adjustment. Tunes the internal transconductance (gm) by increasing the current.

Values:

enumerator kCCM32K_OscCoarseAdjustmentRange0

enumerator kCCM32K_OscCoarseAdjustmentRange1

enumerator kCCM32K_OscCoarseAdjustmentRange2

enumerator kCCM32K_OscCoarseAdjustmentRange3

enum _ccm32k_osc_mode

The enumerator of 32kHz oscillator.

Values:

enumerator kCCM32K_Disable32kHzCrystalOsc

Disable 32kHz Crystal Oscillator.

enumerator kCCM32K_Enable32kHzCrystalOsc

Enable 32kHz Crystal Oscillator.

enumerator kCCM32K_Bypass32kHzCrystalOsc

Bypass 32kHz Crystal Oscillator, use the 32kHz Oscillator or external 32kHz clock.

The enumerator of CCM32K status flag.

Values:

enumerator kCCM32K_32kOscReadyStatusFlag

Indicates the 32kHz crystal oscillator is stable.

enumerator kCCM32K_32kOscActiveStatusFlag

Indicates the 32kHz crystal oscillator is active and in use.

enumerator kCCM32K_32kFroActiveStatusFlag

Indicates the 32kHz free running oscillator is active and in use.

enumerator kCCM32K_ClockDetectStatusFlag

Indicates the clock monitor has detected an error.

enum `_ccm32k_state`

The enumerator of module state.

Values:

enumerator `kCCM32K_Both32kFro32kOscDisabled`

Indicates both 32kHz free running oscillator and 32kHz crystal oscillator are disabled.

enumerator `kCCM32K_Only32kFroEnabled`

Indicates only 32kHz free running oscillator is enabled.

enumerator `kCCM32K_Only32kOscEnabled`

Indicates only 32kHz crystal oscillator is enabled.

enumerator `kCCM32K_Both32kFro32kOscEnabled`

Indicates both 32kHz free running oscillator and 32kHz crystal oscillator are enabled.

enum `_ccm32k_clock_source`

The enumerator of clock source.

Values:

enumerator `kCCM32K_ClockSourceNone`

None clock source.

enumerator `kCCM32K_ClockSource32kFro`

32kHz free running oscillator is the clock source.

enumerator `kCCM32K_ClockSource32kOsc`

32kHz crystal oscillator is the clock source.

enum `_ccm32k_clock_monitor_freq_trim_value`

Clock monitor frequency trim values.

Values:

enumerator `kCCM32K_ClockMonitor2CycleAssert`

Clock monitor asserts 2 cycle after expected edge (assert after 10 cycles with no edge).

enumerator `kCCM32K_ClockMonitor4CycleAssert`

Clock monitor asserts 4 cycle after expected edge (assert after 12 cycles with no edge).

enumerator `kCCM32K_ClockMonitor6CycleAssert`

Clock monitor asserts 6 cycle after expected edge (assert after 14 cycles with no edge).

enumerator `kCCM32K_ClockMonitor8CycleAssert`

Clock monitor asserts 8 cycle after expected edge (assert after 16 cycles with no edge).

enum `_ccm32k_clock_monitor_divide_trim_value`

Clock monitor divide trim values.

Values:

enumerator `kCCM32K_ClockMonitor_1kHzFro32k_1kHzOsc32k`

Clock monitor operates at 1 kHz for both FRO32K and OSC32K.

enumerator `kCCM32K_ClockMonitor_64HzFro32k_1kHzOsc32k`

Clock monitor operates at 64 Hz for FRO32K and clock monitor operates at 1 kHz for OSC32K.

enumerator `kCCM32K_ClockMonitor_1KHzFro32k_64HzOsc32k`

Clock monitor operates at 1K Hz for FRO32K and clock monitor operates at 64 Hz for OSC32K.

enumerator kCCM32K_ClockMonitor_64HzFro32k_64HzOsc32k

Clock monitor operates at 64 Hz for FRO32K and clock monitor operates at 64 Hz for OSC32K.

enum _ccm32k_clock_source_select

CCM32K clock source enumeration.

Values:

enumerator kCCM32K_ClockSourceSelectFro32k

FRO32K clock output is selected as clock source.

enumerator kCCM32K_ClockSourceSelectOsc32k

OSC32K clock output is selected as clock source.

enum _ccm32k_clock_output_peripheral

32kHz clock output peripheral bit map.

Values:

enumerator kCCM32K_ClockOutToRtc

32kHz clock output to RTC.

enumerator kCCM32K_ClockOutToRfmc

32kHz clock output to Rfmc.

enumerator kCCM32K_ClockOutToNbu

32kHz clock output to NBU.

enumerator kCCM32K_ClockOutToWuuRmcPortD

32kHz clock output to WUU/RMC/PORTD.

enumerator kCCM32K_ClockOutToOtherModules

32kHz clock output to Other modules.

typedef enum _ccm32k_osc_xtal_cap ccm32k_osc_xtal_cap_t

The enumerator of internal capacitance of OSC's XTAL pin.

typedef enum _ccm32k_osc_extal_cap ccm32k_osc_extal_cap_t

The enumerator of internal capacitance of OSC's EXTAL pin.

typedef enum _ccm32k_osc_fine_adjustment_value ccm32k_osc_fine_adjustment_value_t

The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by modifying the automatic gain control (AGC).

typedef enum _ccm32k_osc_coarse_adjustment_value ccm32k_osc_coarse_adjustment_value_t

The enumerator of osc amplifier coarse fine adjustment. Tunes the internal transconductance (gm) by increasing the current.

typedef enum _ccm32k_osc_mode ccm32k_osc_mode_t

The enumerator of 32kHz oscillator.

typedef enum _ccm32k_state ccm32k_state_t

The enumerator of module state.

typedef enum _ccm32k_clock_source ccm32k_clock_source_t

The enumerator of clock source.

typedef enum _ccm32k_clock_monitor_freq_trim_value

ccm32k_clock_monitor_freq_trim_value_t

Clock monitor frequency trim values.

```
typedef enum _ccm32k_clock_monitor_divide_trim_value
```

```
ccm32k_clock_monitor_divide_trim_value_t
```

Clock monitor divide trim values.

```
typedef struct _ccm32k_clock_monitor_config ccm32k_clock_monitor_config_t
```

Clock monitor configuration structure.

```
typedef enum _ccm32k_clock_source_select ccm32k_clock_source_select_t
```

CCM32K clock source enumeration.

```
typedef enum _ccm32k_clock_output_peripheral ccm32k_clock_output_peripheral_t
```

32kHz clock output peripheral bit map.

```
typedef struct _ccm32k_osc_config ccm32k_osc_config_t
```

The structure of oscillator configuration.

```
CCM32K_OSC32K_CTRL_OSC_MODE_MASK
```

```
CCM32K_OSC32K_CTRL_OSC_MODE_SHIFT
```

```
CCM32K_OSC32K_CTRL_OSC_MODE(x)
```

```
struct _ccm32k_clock_monitor_config
```

```
#include <fsl_ccm32k.h> Clock monitor configuration structure.
```

Public Members

```
bool enableClockMonitor
```

Used to enable/disable clock monitor.

```
ccm32k_clock_monitor_freq_trim_value_t freqTrimValue
```

Clock minitor frequency trim value.

```
ccm32k_clock_monitor_divide_trim_value_t divideTrimValue
```

Clock minitor divide trim value.

```
struct _ccm32k_osc_config
```

```
#include <fsl_ccm32k.h> The structure of oscillator configuration.
```

Public Members

```
bool enableInternalCapBank
```

enable/disable the internal capacitance bank.

```
ccm32k_osc_xtal_cap_t xtalCap
```

The internal capacitance for the OSC XTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.

```
ccm32k_osc_extal_cap_t extalCap
```

The internal capacitance for the OSC EXTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.

```
ccm32k_osc_fine_adjustment_value_t fineAdjustment
```

32kHz crystal oscillator amplifier fine adjustment value.

```
ccm32k_osc_coarse_adjustment_value_t coarseAdjustment
```

32kHz crystal oscillator amplifier coarse adjustment value.

2.3 CE Basic Functions

`int32_t CE_ExecCmd(void)`

Executes commands in the command queue.

Returns

Command execution status.

`int32_t CE_NullCmd(void)`

Sends a basic command to verify that ZV2117 is operational and that the command interface is functioning properly.

This function passes through the command processing interface and returns a success status.

Returns

Command execution status.

`int32_t CE_Copy(int32_t *pDst, int32_t *pSrc, const int32_t N)`

Copies data between buffers in the ZV2117 memory section.

Copies a specified number of 32-bit words from the source to the destination buffer. Both buffers must reside in the ZV2117 data memory section.

Parameters

- `pDst` – **[out]** Pointer to destination buffer
- `pSrc` – **[in]** Pointer to source buffer
- `N` – **[in]** Number of 32-bit words to copy

Returns

Command execution status.

2.4 CE Command Functions

`int32_t CE_CmdInitBuffer(ce_cmdbuffer_t *psCmdBuffer, volatile uint32_t cmdbuffer[], volatile int32_t statusbuffer[], ce_cmd_mode_t cmdmode)`

Initializes the CM33-ZV2117 command buffer.

This function must be called once after power-up or reset, or when the command queue mode needs to be changed. Once configured, the command mode remains unchanged till reset or re-initialization.

Parameters

- `psCmdBuffer` – **[out]** Pointer to the command buffer structure. Must be allocated in ARM memory (not ZV2117).
- `cmdbuffer` – **[in]** Command buffer. Must be 256 words in ZV2117 data memory.
- `statusbuffer` – **[in]** Status buffer. Must be 134 words in ZV2117 data memory.
- `cmdmode` – **[in]** Command mode. One of:
 - `kCE_CmdModeOneNonBlocking`
 - `kCE_CmdModeMultipleNonBlocking`
 - `kCE_CmdModeOneBlocking`
 - `kCE_CmdModeMultipleBlocking`

Return values

0 – Initialization is successful.

int32_t CE_CmdReset(void)

Resets the CM33-ZV2117 command queue.

Any pending commands in the queue will be flushed.

Return values

0 – Reset is successful.

int32_t CE_CmdAdd(ce_cmd_t cmd, ce_cmdstruct_t *cmdargs)

Adds a command to the command queue.

Parameters

- cmd – **[in]** Command name. Choose from the enum description in fsl_ce_if.h. Not all of the cmd are implemented in the current release.
- cmdargs – **[in]** Arguments structure detailing the arguments for the function/command.

Return values

- 0 – Command added successfully.
- -1 – Command not added (queue is full).

int32_t CE_CmdLaunch(int32_t force_launch)

Launches the ZV2117 with the current command queue.

Parameters

- force_launch – **[in]**
 - 1: Launches the queue regardless of the command mode.
 - 0: Launches only if in single-command mode. Otherwise, does nothing.

Return values

0 – Launch is successful.

int32_t CE_CmdLaunchBlocking(void)

Launches the current command queue and waits for completion.

Return values

0 – Launch is successful.

int32_t CE_CmdLaunchNonBlocking(void)

Launches the current command queue and returns immediately.

ZV2117 will send an interrupt via MUA->GCR to ARM upon task completion. Alternatively, the user can poll for completion.

If using interrupt, the user must call CE_CmdReset() in the IRQ handler. IRQ::DSP_IRQn must be enabled.

The user can optionally also poll to figure out the command queue execution status.

Return values

0 – Launch is successful.

int32_t CE_CmdCheckStatus(void)

Checks the execution status of the current command queue. Only applicable in non-blocking mode.

Returns

- CE_STATUS_BUSY ZV2117 is still executing.

- CE_STATUS_IDLE Execution completed; ZV2117 is ready for new commands.

CE_COMPUTE_TIMEOUT

Maximum loop wait time for CE computation.

When CE is computing, driver will wait for the computation to complete. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

2.5 CE CMSIS Functions

```
void ce_arm_cfft_f32(const arm_cfft_instance_f32 *S, float *p1, uint8_t ifftFlag, uint8_t bitReverseFlag, float *pOut, float *pScratch)
```

CMSIS-compatible FFT implementation using ZV2117 backend.

This API is designed to be compatible with the ARM CM33 FFT interface. It delegates the FFT computation to the ZV2117 CE module. Please refer to CM33 documentation for details.

Note: This API only support float32 FFTs.

Returns

Command execution status.

```
struct arm_cfft_instance_f32
```

```
#include <fsl_ce_cmsis.h> FFT/IFFT float32.
```

Public Members

```
uint16_t fftLen
```

Length of the FFT.

```
const float *pTwiddle
```

Points to the twiddle factor table.

```
const uint16_t *pBitRevTable
```

Points to the bit reversal table.

```
uint16_t bitRevLength
```

Length of the bit reversal table.

2.6 CE Initialization Functions

```
FSL_CE_DRIVER_VERSION
```

CE driver version.

```
typedef struct ce_copy_image ce_copy_image_t
```

Structure for CE copy image to destination address.

Defines source and destination address for copying image with given size.

```
void CE_Init(ce_copy_image_t *ceCopyImage)
```

Initializes the CE.

Parameters

- *ceCopyImage* – **[in]** The information about the CE image to copy.

```
void CE_InstallFirmware(ce_copy_image_t *ceCopyImage)
```

Installs CE firmware by given image info.

Parameters

- *ceCopyImage* – **[in]** The information about the CE image to copy.

```
void CE_InitWithoutFirmware(void)
```

Initializes the CE.

This function is similar to `CE_Init`, but it does not install the firmware, the firmware can be installed using `CE_InstallFirmware`.

```
struct _ce_copy_image
```

#include <fsl_ce.h> Structure for CE copy image to destination address.

Defines source and destination address for copying image with given size.

2.7 CE Matrix Functions

```
int32_t CE_MatrixAdd_Q15(int16_t *pDst, int16_t *pA, int16_t *pB, int32_t M, int32_t N)
```

Calculates the sum of two real 16-bit integer (Q15) matrices.

Computes $C = A + B$, where A, B, and C are an $M \times N$ real `int16_t` matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- *pDst* – **[out]** Pointer to the output matrix C (size $M \times N$).
- *pA* – **[in]** Pointer to the input matrix A (size $M \times N$).
- *pB* – **[in]** Pointer to the input matrix B (size $M \times N$).
- *M* – **[in]** Number of rows of matrices A, B, and C.
- *N* – **[in]** Number of columns of matrices A, B, and C.

Returns

Command execution status.

```
int32_t CE_MatrixAdd_Q31(int32_t *pDst, int32_t *pA, int32_t *pB, int32_t M, int32_t N)
```

Calculates the sum of two real 32-bit integer (Q31) matrices.

Computes $C = A + B$, where A, B, and C are an $M \times N$ real `int32_t` matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- *pDst* – **[out]** Pointer to the output matrix C (size $M \times N$).
- *pA* – **[in]** Pointer to the input matrix A (size $M \times N$).
- *pB* – **[in]** Pointer to the input matrix B (size $M \times N$).
- *M* – **[in]** Number of rows of matrices A, B, and C.
- *N* – **[in]** Number of columns of matrices A, B, and C.

Returns

Command execution status.

int32_t CE_MatrixAdd_F32(float *pDst, float *pA, float *pB, int32_t M, int32_t N)

Calculates the sum of two real 32-bit floating-point matrices.

Computes $C = A + B$, where A, B, and C are an $M \times N$ real float32 matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size $M \times N$).
- pA – **[in]** Pointer to the input matrix A (size $M \times N$).
- pB – **[in]** Pointer to the input matrix B (size $M \times N$).
- M – **[in]** Number of rows of matrices A, B, and C.
- N – **[in]** Number of columns of matrices A, B, and C.

Returns

Command execution status.

int32_t CE_MatrixElemMul_F32(float *pDst, float *pA, float *pB, int32_t M, int32_t N)

Calculates the element-wise product of two real 32-bit floating-point matrices.

Computes $C = A \times B$, where A, B, and C are an $M \times N$ real float32 matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size $M \times N$).
- pA – **[in]** Pointer to the input matrix A (size $M \times N$).
- pB – **[in]** Pointer to the input matrix B (size $M \times N$).
- M – **[in]** Number of rows of matrices A, B, and C.
- N – **[in]** Number of columns of matrices A, B, and C.

Returns

Command execution status.

int32_t CE_MatrixMul_F32(float *pDst, float *pA, float *pB, int32_t M, int32_t N, int32_t P)

Calculates the matrix product of two real 32-bit floating-point matrices.

Computes $C = A \times B$, where:

- A is an $M \times N$ real float32 matrix,
- B is an $N \times P$ real float32 matrix,
- C is the resulting $M \times P$ real float32 matrix.

All matrices are assumed to be in row-major format.

Note: Data precision and format is as defined by the argument type. Limits on max value of N:

- For F32: $N < 128$;
 - For CF32: $N < 64$.
-

Parameters

- pDst – **[out]** Pointer to the output matrix C (size $M \times P$)
- pA – **[in]** Pointer to the input matrix A (size $M \times N$).
- pB – **[in]** Pointer to the input matrix B (size $N \times P$).

- M – **[in]** Number of rows of matrix A.
- N – **[in]** Number of columns in matrix A (and rows in matrix B).
- P – **[in]** Number of columns of matrix B.

Returns

Command execution status.

`int32_t CE_MatrixMul_CF32(float *pDst, float *pA, float *pB, int32_t M, int32_t N, int32_t P)`

Calculates the matrix product of two complex 32-bit floating point matrices.

Computes $C = A \times B$, where:

- A is an $M \times N$ complex float32 matrix,
- B is an $N \times P$ complex float32 matrix,
- C is the resulting $M \times P$ complex float32 matrix.

All matrices are assumed to be in row-major format.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size $M \times P$).
- pA – **[in]** Pointer to the input matrix A (size $M \times N$).
- pB – **[in]** Pointer to the input matrix B (size $N \times P$).
- M – **[in]** Number of rows of matrix A.
- N – **[in]** Number of columns in matrix A (and rows in matrix B).
- P – **[in]** Number of columns of matrix B.

Returns

Command execution status.

`int32_t CE_MatrixInvHerm_CF32(float *pAinv, float *pA, float *pScratch, int32_t M, uint8_t flag_packedInput, uint8_t flag_cholInv)`

Calculates the inverse or Cholesky inverse of a complex Hermitian matrix (in float32 precision).

Computes one of the following:

- $A_{inv} = \text{inv}(A)$
- $A_{inv} = \text{inv}(\text{chol}(A))$

where $\text{chol}(A)$ is the lower triangular Cholesky decomposition of matrix A.

The input matrix must be in row-major format and can be either:

- A full $M \times M$ matrix, or
- A packed upper triangular matrix (containing only the upper triangle).

The output matrix is written in row-major format, and only the lower triangle is stored. The total number of output elements is:

- $M_c = (M + 1) \times M / 2$

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- pAinv – **[out]** Pointer to the output matrix inverse. M_c elements are written (complex float32).

- pA – **[in]** Pointer to the input matrix A (complex float32, size $M \times M$).
- pScratch – **[in]** Pointer to scratch buffer (minimum size $M_c \times 3 \times 8$ bytes).
- M – **[in]** Number of rows/columns in matrix A.
- flag_packedInput – **[in]** Format of input matrix:
 - 0: full matrix
 - 1: packed upper triangular matrix
- flag_cholInv – **[in]** Type of inverse to compute:
 - 0: inv(A)
 - 1: inv(chol(A))

Returns

Command execution status.

```
int32_t CE_MatrixEvdHerm_CF32(float *pLambdaOut, float *pUout, float *pUin, float
                             *pScratch, int32_t M, float tol, int32_t max_iter, uint8_t
                             flag_packedInput)
```

Calculates the Eigenvalue Decomposition (EVD) of a complex Hermitian matrix (float32 precision).

Performs EVD on an $M \times M$ complex Hermitian matrix A:

- [U, T] = eig(A)

The input matrix must be in column-major format and can be either:

- A full $M \times M$ matrix, or
- A packed upper triangular matrix.

The decomposition uses an iterative QR algorithm to compute:

- U: matrix of eigenvectors ($M \times M$, column-major)
- T: vector of eigenvalues ($M \times 1$)

Maximum supported matrix size: $M \leq 40$.

Note: If input and output buffers overlap, the output U matrix will overwrite the input matrix A.

Parameters

- pLambdaOut – **[out]** Pointer to the output vector of eigenvalues (size $M \times 1$, complex float32).
- pUout – **[out]** Pointer to the output matrix of eigenvectors (size $M \times M$, column-major, complex float32).
- pUin – **[in]** Pointer to the input matrix A (size $M \times M$, column-major, complex float32).
- pScratch – **[in]** Pointer to scratch buffer (minimum size: $(M \times M \times 4 + 360) \times 4$ bytes).
- M – **[in]** Number of rows/columns in matrix A.
- tol – **[in]** Tolerance value for QR convergence. Smaller values yield better accuracy but require more iterations.
- max_iter – **[in]** Maximum number of QR iterations. If exceeded, the function terminates regardless of convergence.

- `flag_packedInput` – **[in]** Format of input matrix:
 - 0: full matrix
 - 1: packed upper triangular part only

Returns

Command execution status. The number of QR iterations is returned in `status[3]` register.

```
int32_t CE_MatrixChol_CF32(float *pL, float *pA, float *pScratch, int32_t M, uint8_t
                           flag_packedInput)
```

Calculates the Cholesky Decomposition of a complex Hermitian matrix (float32 precision).

Calculates $L = \text{chol}(A)$, where A is a complex Hermitian $M \times M$ matrix and L is a lower triangular matrix such that $A = L \times L^H$.

The input matrix must be in row-major format and can be either:

- A full $M \times M$ matrix, or
- A packed format containing only the upper triangular elements.

The output matrix is also in row-major format, and only the lower triangular part is written. The total number of output elements is M_c , where:

- $M_c = (M + 1) \times M / 2$

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- `pL` – **[out]** Pointer to the output lower triangular matrix L . Only M_c elements are written (complex-valued float32 data in row-major format).
- `pA` – **[in]** Pointer to the input matrix A (full or packed format). Must be complex-valued float32 data in row-major format.
- `pScratch` – **[in]** Pointer to a scratch buffer (at least $M_c \times 8$ bytes).
- `M` – **[in]** Number of rows/columns of matrix A .
- `flag_packedInput` – **[in]** Format flag for input matrix:
 - 0: full matrix
 - 1: packed upper triangular matrix

Returns

Command execution status.

2.8 CE Transform Functions

```
int32_t CE_TransformCFFT_F16(float *pY, float *pX, float *pScratch, int32_t log2N)
```

Computes the FFT of complex-valued 16-bit floating point data.

Performs an N -point Fast Fourier Transform (FFT) on a complex-valued float16 input stream. The FFT size N must be a power of 2, with valid range: $16 \leq N \leq 16384$.

Note:

- Data precision and format follow the argument types.

- `float*` is used to represent float16 pointers.
 - The input, output and scratch buffers must be separately allocated and non-overlapping.
-

Parameters

- `pY` – **[out]** Pointer to the FFT output buffer (complex-valued float16 data).
- `pX` – **[in]** Pointer to the FFT input buffer (complex-valued float16 data).
- `pScratch` – **[in]** Pointer to a scratch buffer (at least $N \times 4$ bytes).
- `log2N` – **[in]** Base-2 logarithm of the FFT size N .

Returns

Command execution status.

`int32_t CE_TransformCFFT_F32(float *pY, float *pX, float *pScratch, int32_t log2N)`

Computes the FFT of complex-valued 32-bit floating point data.

Performs an N -point Fast Fourier Transform (FFT) on a complex-valued float32 input stream. The FFT size N must be a power of 2, with valid range: $32 \leq N \leq 16384$.

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- `pY` – **[out]** Pointer to the FFT output buffer (complex-valued float32).
- `pX` – **[in]** Pointer to the FFT input buffer (complex-valued float32).
- `pScratch` – **[in]** Pointer to a scratch buffer (at least $N \times 8$ bytes).
- `log2N` – **[in]** Base-2 logarithm of the FFT size N .

Returns

Command execution status.

`int32_t CE_TransformIFFT_F16(float *pY, float *pX, float *pScratch, int32_t log2N)`

Computes the IFFT of complex-valued 16-bit floating point data.

Performs an N -point Inverse Fast Fourier Transform (IFFT) on a complex-valued float16 input stream. The IFFT size N must be a power of 2, with valid range: $16 \leq N \leq 16384$.

Note:

- Data precision and format follow the argument types.
 - `float*` is used to represent float16 pointers.
 - The input, output and scratch buffers must be separately allocated and non-overlapping.
-

Parameters

- `pY` – **[out]** Pointer to the IFFT output buffer (complex-valued float16).
- `pX` – **[in]** Pointer to the IFFT input buffer (complex-valued float16).
- `pScratch` – **[in]** Pointer to a scratch buffer (at least $N \times 4$ bytes).
- `log2N` – **[in]** Base-2 logarithm of the IFFT size N .

Returns

Command execution status.

`int32_t CE_TransformIFFT_F32(float *pY, float *pX, float *pScratch, int32_t log2N)`

Computes the IFFT of complex-valued 32-bit floating point data.

Performs an N-point Inverse Fast Fourier Transform (IFFT) on a complex-valued float32 input stream. The IFFT size N must be a power of 2, with valid range: $32 \leq N \leq 16384$.

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- `pY` – **[out]** Pointer to the IFFT output buffer (complex-valued float32).
- `pX` – **[in]** Pointer to the IFFT input buffer (complex-valued float32).
- `pScratch` – **[in]** Pointer to a scratch buffer (at least $N \times 8$ bytes).
- `log2N` – **[in]** Base-2 logarithm of the IFFT size N.

Returns

Command execution status.

2.9 Clock Driver

`enum _clock_name`

Clock name used to get clock frequency.

These clocks source would be generated from SCG module.

Values:

enumerator `kCLOCK_CoreSysClk`
Cortex M33 clock.

enumerator `kCLOCK_SlowClk`
SLOW_CLK with DIVSLOW.

enumerator `kCLOCK_PlatformClk`
PLAT_CLK.

enumerator `kCLOCK_SysClk`
SYS_CLK.

enumerator `kCLOCK_BusClk`
BUS_CLK with DIVBUS.

enumerator `kCLOCK_ScgSysOscClk`
SCG system OSC clock.

enumerator `kCLOCK_ScgSircClk`
SCG SIRC clock.

enumerator `kCLOCK_ScgFircClk`
SCG FIRC clock.

enumerator `kCLOCK_RtcOscClk`
RTC OSC clock.

enum `_clock_ip_control`

Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[CC]

Values:

enumerator `kCLOCK_IpClkControl_fun0`

Peripheral clocks are disabled, module does not stall low power mode entry.

enumerator `kCLOCK_IpClkControl_fun1`

Peripheral clocks are enabled, module does not stall low power mode entry.

enumerator `kCLOCK_IpClkControl_fun2`

Peripherals clocks are enabled unless peripheral is idle, low power mode entry will stall until peripheral is idle.

enumerator `kCLOCK_IpClkControl_fun3`

Peripheral clocks are enabled unless in SLEEP mode (or lower), low power mode entry will stall until peripheral is idle Peripheral functional clocks that remain enabled in SLEEP mode are enabled and do not stall low power mode entry unless entering DEEPSLEEP mode (or lower)

enum `_clock_ip_src`

Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[MUX].

Values:

enumerator `kCLOCK_IpSrcFro6M`

FRO 6M clock.

enumerator `kCLOCK_IpSrcFro192M`

FRO 192M clock.

enumerator `kCLOCK_IpSrcSoscClk`

OSC RF clock.

enumerator `kCLOCK_IpSrc32kClk`

32k Clk clock.

enum `_tpm2_ip_src`

Clock source for TPM2.

These options are for RF_CMC1->TPM2_CFG[CLK_MUX_SEL].

Values:

enumerator `kCLOCK_Tpm2SrcCoreClk`

Core Clock.

enumerator `kCLOCK_Tpm2SrcSoscClk`

Radio Oscillator.

enum `_clock_ip_name`

Clock IP name.

Values:

enumerator `kCLOCK_NOGATE`

No clock gate for the IP in MRCC

enumerator `kCLOCK_Ewm0`

Clock ewm0

enumerator kCLOCK_Syspm0
Clock syspm0

enumerator kCLOCK_Wdog0
Clock wdog0

enumerator kCLOCK_Wdog1
Clock wdog1

enumerator kCLOCK_Sfa0
Clock sfa0

enumerator kCLOCK_Crc0
Clock crc0

enumerator kCLOCK_Secsubsys
Clock secsubsys

enumerator kCLOCK_Lpit0
Clock lpit0

enumerator kCLOCK_Tstmr0
Clock tstmr0

enumerator kCLOCK_Tpm0
Clock tpm0

enumerator kCLOCK_Tpm1
Clock tpm1

enumerator kCLOCK_Lpi2c0
Clock lpi2c0

enumerator kCLOCK_Lpi2c1
Clock lpi2c1

enumerator kCLOCK_I3c0
Clock i3c

enumerator kCLOCK_Lpspi0
Clock lpspi0

enumerator kCLOCK_Lpspi1
Clock lpspi1

enumerator kCLOCK_Lpuart0
Clock lpuart0

enumerator kCLOCK_Lpuart1
Clock lpuart1

enumerator kCLOCK_Flexio0
Clock Flexio0

enumerator kCLOCK_Can0
Clock Can0

enumerator kCLOCK_Sema0
Clock Sema0

enumerator kCLOCK_Data_stream_2p4
Clock data_stream_2p4

enumerator kCLOCK_PortA
Clock portA

enumerator kCLOCK_PortB
Clock portB

enumerator kCLOCK_PortC
Clock portC

enumerator kCLOCK_Lpadc0
Clock lpadc0

enumerator kCLOCK_Lpcmp0
Clock lpcmp0

enumerator kCLOCK_Lpcmp1
Clock lpcmp1

enumerator kCLOCK_Vref0
Clock verf0

enumerator kCLOCK_Mtr_master
Clock mtr_master

enumerator kCLOCK_Can1
Clock Can1

enumerator kCLOCK_GpioA
Clock gpioA

enumerator kCLOCK_GpioB
Clock gpioB

enumerator kCLOCK_GpioC
Clock gpioC

enumerator kCLOCK_Dma0
Clock dma0

enumerator kCLOCK_Pflexnvm
Clock pflexnvm

enumerator kCLOCK_Sram0
Clock Sram0

enumerator kCLOCK_Sram1
Clock Sram1

enumerator kCLOCK_Sram2
Clock Sram2

enumerator kCLOCK_Sram3
Clock Sram3

enumerator kCLOCK_Sram0_NOECC
Clock Sram0 NOECC

enumerator kCLOCK_DSP0
Clock DSPV

enumerator kCLOCK_DSP0_MUA
Clock DSPV MUA

enumerator kCLOCK_Sram1_NOECC
Clock Sram1 NOECC

enumerator kCLOCK_Rf_2p4ghz_bist
Clock rf_2p4ghz_bist

enumerator kCLOCK_Lptmr0_wake2vsys
Clock LPTMR0 IPG clk erclk wake2vsys

enumerator kCLOCK_Lptmr1_wake2vsys
Clock LPTMR1 IPG clk erclk wake2vsys

enumerator kCLOCK_Lptmr2_wake2vsys
Clock LPTMR2 IPG clk erclk wake2vsys

enumerator kCLOCK_Sirc_vsys_gating
Clock SIRC vsys gating

enum _scg_status

SCG status return codes.

Values:

enumerator kStatus_SCG_Busy
Clock is busy.

enumerator kStatus_SCG_InvalidSrc
Invalid source.

enum _scg_sys_clk

SCG system clock type.

Values:

enumerator kSCG_SysClkSlow
System slow clock.

enumerator kSCG_SysClkBus
Bus clock.

enumerator kSCG_SysClkPlatform
Platform clock.

enumerator kSCG_SysClkCore
Core clock.

enum _scg_sys_clk_src

SCG system clock source.

ERR052742: FRO6M clock(kSCG_SysClkSrcSirc) is not stable. The FRO6M clock is not stable on some parts. FRO6M outputs lower frequency signal instead of 6MHz when device is reset or wakes up from low power. It can impact peripherals using it as a clock source. Please use clock source other than the FRO6M. For example, use FRO192M instead of FRO6M as clock source for peripherals.

Values:

enumerator kSCG_SysClkSrcSysOsc
System OSC.

enumerator kSCG_SysClkSrcSirc
Slow IRC.

enumerator kSCG_SysClkSrcFirc
Fast IRC.

enumerator kSCG_SysClkSrcRosc
RTC OSC.

enum _scg_sys_clk_div
SCG system clock divider value.

Values:

enumerator kSCG_SysClkDivBy1
Divided by 1.

enumerator kSCG_SysClkDivBy2
Divided by 2.

enumerator kSCG_SysClkDivBy3
Divided by 3.

enumerator kSCG_SysClkDivBy4
Divided by 4.

enumerator kSCG_SysClkDivBy5
Divided by 5.

enumerator kSCG_SysClkDivBy6
Divided by 6.

enumerator kSCG_SysClkDivBy7
Divided by 7.

enumerator kSCG_SysClkDivBy8
Divided by 8.

enumerator kSCG_SysClkDivBy9
Divided by 9.

enumerator kSCG_SysClkDivBy10
Divided by 10.

enumerator kSCG_SysClkDivBy11
Divided by 11.

enumerator kSCG_SysClkDivBy12
Divided by 12.

enumerator kSCG_SysClkDivBy13
Divided by 13.

enumerator kSCG_SysClkDivBy14
Divided by 14.

enumerator kSCG_SysClkDivBy15
Divided by 15.

enumerator kSCG_SysClkDivBy16
Divided by 16.

enum _clock_clkout_src
SCG clock out configuration (CLKOUTSEL).

Values:

enumerator kClockClkoutSelScgSlow
SCG Slow clock.

enumerator kClockClkoutSelSosc
System OSC.

enumerator kClockClkoutSelSirc
Slow IRC.

enumerator kClockClkoutSelFirc
Fast IRC.

enumerator kClockClkoutSelScgRtcOsc
SCG RTC OSC clock.

enum _scg_sosc_monitor_mode
SCG system OSC monitor mode.

Values:

enumerator kSCG_SysOscMonitorDisable
Monitor disabled.

enumerator kSCG_SysOscMonitorInt
Interrupt when the SOSC error is detected.

enumerator kSCG_SysOscMonitorReset
Reset when the SOSC error is detected.

SOSC enable mode.

Values:

enumerator kSCG_SoscDisable
Disable SOSC clock.

enumerator kSCG_SoscEnable
Enable SOSC clock.

enumerator kSCG_SoscEnableInSleep
Enable SOSC in sleep mode.

enum _scg_rosc_monitor_mode
SCG ROSC monitor mode.

Values:

enumerator kSCG_RoscMonitorDisable
Monitor disabled.

enumerator kSCG_RoscMonitorInt
Interrupt when the RTC OSC error is detected.

enumerator kSCG_RoscMonitorReset
Reset when the RTC OSC error is detected.

enum _scg_sirc_enable_mode
SIRC enable mode.

Values:

enumerator kSCG_SircDisableInSleep
Disable SIRC clock.

enumerator kSCG_SircEnableInSleep
Enable SIRC in sleep mode.

enum _scg_firc_trim_mode
SCG fast IRC trim mode.

Values:

enumerator kSCG_FircTrimNonUpdate
FIRC trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure scg_firc_trim_config_t.

enumerator kSCG_FircTrimUpdate
FIRC trim enable and trim value update enable. In this mode, the trim value is auto update.

enum _scg_firc_trim_src
SCG fast IRC trim source.

Values:

enumerator kSCG_FircTrimSrcSysOsc
System OSC.

enumerator kSCG_FircTrimSrcRtcOsc
RTC OSC (32.768 kHz).

FIRC enable mode.

Values:

enumerator kSCG_FircDisable
Disable FIRC clock.

enumerator kSCG_FircEnable
Enable FIRC clock.

enumerator kSCG_FircEnableInSleep
Enable FIRC in sleep mode.

enum _scg_firc_range
SCG fast IRC clock frequency range.

Values:

enumerator kSCG_FircRange48M
Fast IRC is trimmed to 48 MHz.

enumerator kSCG_FircRange64M
Fast IRC is trimmed to 64 MHz.

enumerator kSCG_FircRange96M
Fast IRC is trimmed to 96 MHz.

enumerator kSCG_FircRange192M
Fast IRC is trimmed to 192 MHz.

enum _fro192m_rf_range
FRO192M RF clock frequency range.

Values:

enumerator kFro192M_Range16M
FRO192M output frequenc 16 MHz.

enumerator kFro192M_Range24M
FRO192M output frequenc 24 MHz.

enumerator kFro192M_Range32M
FRO192M output frequenc 32 MHz.

enumerator kFro192M_Range48M
FRO192M output frequenc 48 MHz.

enumerator kFro192M_Range64M
FRO192M output frequenc 64 MHz.

enum _fro192m_rf_clk_div
RF Flash APB and RF_CMC clock divide.

Values:

enumerator kFro192M_ClkDivBy1
Divided by 1.

enumerator kFro192M_ClkDivBy2
Divided by 2.

enumerator kFro192M_ClkDivBy4
Divided by 4.

enumerator kFro192M_ClkDivBy8
Divided by 8.

typedef enum _clock_name clock_name_t
Clock name used to get clock frequency.

These clocks source would be generated from SCG module.

typedef enum _clock_ip_control clock_ip_control_t
Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[CC]

typedef enum _clock_ip_src clock_ip_src_t
Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[MUX].

typedef enum _tpm2_ip_src tpm2_src_t
Clock source for TPM2.

These options are for RF_CMC1->TPM2_CFG[CLK_MUX_SEL].

typedef enum _clock_ip_name clock_ip_name_t
Clock IP name.

typedef enum _scg_sys_clk scg_sys_clk_t
SCG system clock type.

typedef enum _scg_sys_clk_src scg_sys_clk_src_t
SCG system clock source.

ERR052742: FRO6M clock(kSCG_SysClkSrcSirc) is not stable. The FRO6M clock is not stable on some parts. FRO6M outputs lower frequency signal instead of 6MHz when device is reset or wakes up from low power. It can impact peripherals using it as a clock source. Please use clock source other than the FRO6M. For example, use FRO192M instead of FRO6M as clock source for peripherals.

typedef enum *_scg_sys_clk_div* scg_sys_clk_div_t
SCG system clock divider value.

typedef struct *_scg_sys_clk_config* scg_sys_clk_config_t
SCG system clock configuration.

typedef enum *_clock_clkout_src* clock_clkout_src_t
SCG clock out configuration (CLKOUTSEL).

typedef enum *_scg_sosc_monitor_mode* scg_sosc_monitor_mode_t
SCG system OSC monitor mode.

typedef struct *_scg_sosc_config* scg_sosc_config_t
SCG system OSC configuration.

typedef enum *_scg_rosc_monitor_mode* scg_rosc_monitor_mode_t
SCG ROSC monitor mode.

typedef struct *_scg_rosc_config* scg_rosc_config_t
SCG ROSC configuration.

typedef enum *_scg_sirc_enable_mode* scg_sirc_enable_mode_t
SIRC enable mode.

typedef struct *_scg_sirc_config* scg_sirc_config_t
SCG slow IRC clock configuration.

typedef enum *_scg_firc_trim_mode* scg_firc_trim_mode_t
SCG fast IRC trim mode.

typedef enum *_scg_firc_trim_src* scg_firc_trim_src_t
SCG fast IRC trim source.

typedef struct *_scg_firc_trim_config* scg_firc_trim_config_t
SCG fast IRC clock trim configuration.

typedef enum *_scg_firc_range* scg_firc_range_t
SCG fast IRC clock frequency range.

typedef struct *_scg_firc_config_t* scg_firc_config_t
SCG fast IRC clock configuration.

typedef enum *_fro192m_rf_range* fro192m_rf_range_t
FRO192M RF clock frequency range.

typedef enum *_fro192m_rf_clk_div* fro192m_rf_clk_div_t
RF Flash APB and RF_CMC clock divide.

typedef struct *_fro192m_rf_clk_config* fro192m_rf_clk_config_t
FRO192M RF clock configuration.

volatile uint32_t g_xtal0Freq
External XTAL0 (OSC0/SYSOSC) clock frequency.

The XTAL0/EXTAL0 (OSC0/SYSOSC) clock frequency in Hz. When the clock is set up, use the function `CLOCK_SetXtal0Freq` to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
CLOCK_InitSysOsc(...);  
CLOCK_SetXtal0Freq(8000000);
```

This is important for the multicore platforms where only one core needs to set up the OSC0/SYSOSC using `CLOCK_InitSysOsc`. All other cores need to call the `CLOCK_SetXtal0Freq` to get a valid clock frequency.

```
volatile uint32_t g_xtal32Freq
```

External XTAL32/EXTAL32 clock frequency.

The XTAL32/EXTAL32 clock frequency in Hz. When the clock is set up, use the function `CLOCK_SetXtal32Freq` to set the value in the clock driver.

This is important for the multicore platforms where only one core needs to set up the clock. All other cores need to call the `CLOCK_SetXtal32Freq` to get a valid clock frequency.

```
static inline void CLOCK_EnableClock(clock_ip_name_t name)
```

Enable the clock for specific IP.

Parameters

- `name` – Which clock to enable, see `clock_ip_name_t`.

```
static inline void CLOCK_EnableTPM2(void)
```

Enable the TPM2 clock.

```
static inline void CLOCK_EnableClockLPMode(clock_ip_name_t name, clock_ip_control_t control)
```

Enable the clock for specific IP in low power mode.

Parameters

- `name` – Which clock to enable, see `clock_ip_name_t`.
- `control` – Clock Config, see `clock_ip_control_t`.

```
static inline void CLOCK_DisableClock(clock_ip_name_t name)
```

Disable the clock for specific IP.

Parameters

- `name` – Which clock to disable, see `clock_ip_name_t`.

```
static inline void CLOCK_DisableTPM2(void)
```

Disable the TPM2 clock.

```
static inline void CLOCK_SetIpSrc(clock_ip_name_t name, clock_ip_src_t src)
```

Set the clock source for specific IP module.

Set the clock source for specific IP, not all modules need to set the clock source, should only use this function for the modules need source setting. ERR052742: FRO6M clock is not stable. The FRO6M clock is not stable on some parts. FRO6M outputs lower frequency signal instead of 6MHz when device is reset or wakes up from low power. It can impact peripherals using it as a clock source. Please use clock source other than the FRO6M. For example, use FRO192M instead of FRO6M as clock source for peripherals.

Parameters

- `name` – Which peripheral to check, see `clock_ip_name_t`.
- `src` – Clock source to set.

```
static inline void CLOCK_SetTpm2Src(tpm2_src_t src)
```

Set the clock source for TPM2.

Parameters

- `src` – Clock source to set.

```
static inline void CLOCK_SetIpSrcDiv(clock_ip_name_t name, uint8_t divValue)
```

Set the clock source and divider for specific IP module.

Set the clock source and divider for specific IP, not all modules need to set the clock source and divider, should only use this function for the modules need source and divider setting.

Divider output clock = Divider input clock / (divValue+1)].

Parameters

- name – Which peripheral to check, see clock_ip_name_t.
- divValue – The divider value.

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_name_t.

Parameters

- clockName – Clock names defined in clock_name_t

Returns

Clock frequency value in hertz

uint32_t CLOCK_GetCoreSysClkFreq(void)

Get the core clock or system clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetPlatClkFreq(void)

Get the platform clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetBusClkFreq(void)

Get the bus clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetFlashClkFreq(void)

Get the flash clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetIpFreq(*clock_ip_name_t* name)

Gets the functional clock frequency for a specific IP module.

This function gets the IP module's functional clock frequency based on MRCC registers. It is only used for the IP modules which could select clock source by MRCC[PCS].

Parameters

- name – Which peripheral to get, see clock_ip_name_t.

Returns

Clock frequency value in Hz

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.2.6.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

EDMA_CLOCKS

Clock ip name array for EDMA.

SYSPM_CLOCKS

Clock ip name array for SYSPM.

SFA_CLOCKS

Clock ip name array for SFA.

CRC_CLOCKS

Clock ip name array for CRC.

TPM_CLOCKS

Clock ip name array for TPM.

LPI2C_CLOCKS

Clock ip name array for LPI2C.

I3C_CLOCKS

Clock ip name array for I3C.

LPSPI_CLOCKS

Clock ip name array for LPSPI.

LPUART_CLOCKS

Clock ip name array for LPUART.

PORT_CLOCKS

Clock ip name array for PORT.

LPADC_CLOCKS

Clock ip name array for LPADC.

LPCMP_CLOCKS

Clock ip name array for LPCMP.

VREF_CLOCKS

Clock ip name array for VREF.

GPIO_CLOCKS

Clock ip name array for GPIO.

LPIT_CLOCKS

Clock ip name array for LPIT.

RF_CLOCKS

Clock ip name array for RF.

WDOG_CLOCKS

Clock ip name array for WDOG.

FLEXCAN_CLOCKS

Clock ip name array for FLEXCAN.

FLEXIO_CLOCKS

Clock ip name array for FLEXIO.

TSTMR_CLOCKS

Clock ip name array for TSTMR.

EWM_CLOCKS

Clock ip name array for EWM.

SEMA42_CLOCKS

Clock ip name array for SEMA42.

MU_CLOCKS

Clock ip name array for MU.

MAKE_MRCC_REGADDR(base, offset)

“IP Connector name definition used for clock gate, clock source and clock divider setting. It is defined as the corresponding register address.

CLOCK_REG(name)

uint32_t CLOCK_GetSysClkFreq(*scg_sys_clk_t* type)

Gets the SCG system clock frequency.

This function gets the SCG system clock frequency. These clocks are used for core, platform, external, and bus clock domains.

Parameters

- type – Which type of clock to get, core clock or slow clock.

Returns

Clock frequency.

static inline void CLOCK_SetRunModeSysClkConfig(const *scg_sys_clk_config_t* *config)

Sets the system clock configuration for RUN mode.

This function sets the system clock configuration for RUN mode.

Parameters

- config – Pointer to the configuration.

static inline void CLOCK_GetCurSysClkConfig(*scg_sys_clk_config_t* *config)

Gets the system clock configuration in the current power mode.

This function gets the system configuration in the current power mode.

Parameters

- config – Pointer to the configuration.

static inline void CLOCK_SetClkOutSel(*clock_clkout_src_t* setting)

Sets the clock out selection.

This function sets the clock out selection (CLKOUTSEL).

Parameters

- setting – The selection to set.

status_t CLOCK_InitSysOsc(const *scg_sosc_config_t* *config)

Initializes the SCG system OSC.

This function enables the SCG system OSC clock according to the configuration.

Note: This function can't detect whether the system OSC has been enabled and used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – System OSC is initialized.
- kStatus_SCG_Busy – System OSC has been enabled and is used by the system clock.
- kStatus_ReadOnly – System OSC control register is locked.

status_t CLOCK_DeinitSysOsc(void)

De-initializes the SCG system OSC.

This function disables the SCG system OSC clock.

Note: This function can't detect whether the system OSC is used by an IP.

Return values

- *kStatus_Success* – System OSC is deinitialized.
- *kStatus_SCG_Busy* – System OSC is used by the system clock.
- *kStatus_ReadOnly* – System OSC control register is locked.

uint32_t CLOCK_GetSysOscFreq(void)

Gets the SCG system OSC clock frequency (SYSOSC).

Returns

Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK_IsSysOscErr(void)

Checks whether the system OSC clock error occurs.

Returns

True if the error occurs, false if not.

static inline void CLOCK_ClearSysOscErr(void)

Clears the system OSC clock error.

static inline void CLOCK_SetSysOscMonitorMode(*scg_sosc_monitor_mode_t* mode)

Sets the system OSC monitor mode.

This function sets the system OSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- *mode* – Monitor mode to set.

static inline bool CLOCK_IsSysOscValid(void)

Checks whether the system OSC clock is valid.

Returns

True if clock is valid, false if not.

static inline void CLOCK_UnlockSysOscControlStatusReg(void)

Unlock the SOSCSR control status register.

static inline void CLOCK_LockSysOscControlStatusReg(void)

Lock the SOSCSR control status register.

status_t CLOCK_InitSirc(const *scg_sirc_config_t* *config)

Initializes the SCG slow IRC clock.

This function enables the SCG slow IRC clock according to the configuration.

Note: This function can't detect whether the system OSC has been enabled and used by an IP.

Parameters

- *config* – Pointer to the configuration structure.

Return values

- `kStatus_Success` – SIRC is initialized.
- `kStatus_SCG_Busy` – SIRC has been enabled and is used by system clock.
- `kStatus_ReadOnly` – SIRC control register is locked.

`status_t` `CLOCK_DeinitSirc(void)`

De-initializes the SCG slow IRC.

This function disables the SCG slow IRC.

Note: This function can't detect whether the SIRC is used by an IP.

Return values

- `kStatus_Success` – SIRC is deinitialized.
- `kStatus_SCG_Busy` – SIRC is used by system clock.
- `kStatus_ReadOnly` – SIRC control register is locked.

`uint32_t` `CLOCK_GetSircFreq(void)`

Gets the SCG SIRC clock frequency.

Returns

Clock frequency; If the clock is invalid, returns 0.

`static inline bool` `CLOCK_IsSircValid(void)`

Checks whether the SIRC clock is valid.

Returns

True if clock is valid, false if not.

`static inline void` `CLOCK_UnlockSircControlStatusReg(void)`

Unlock the SIRCCSR control status register.

`static inline void` `CLOCK_LockSircControlStatusReg(void)`

Lock the SIRCCSR control status register.

`status_t` `CLOCK_InitFirc(const scg_firc_config_t *config)`

Initializes the SCG fast IRC clock.

This function enables the SCG fast IRC clock according to the configuration.

Note: This function can't detect whether the FIRC has been enabled and used by an IP.

Parameters

- `config` – Pointer to the configuration structure.

Return values

- `kStatus_Success` – FIRC is initialized.
- `kStatus_SCG_Busy` – FIRC has been enabled and is used by the system clock.
- `kStatus_ReadOnly` – FIRC control register is locked.

status_t CLOCK_DeinitFirc(void)

De-initializes the SCG fast IRC.

This function disables the SCG fast IRC.

Note: This function can't detect whether the FIRC is used by an IP.

Return values

- `kStatus_Success` – FIRC is deinitialized.
- `kStatus_SCG_Busy` – FIRC is used by the system clock.
- `kStatus_ReadOnly` – FIRC control register is locked.

uint32_t CLOCK_GetFircFreq(void)

Gets the SCG FIRC clock frequency.

Returns

Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK_IsFircErr(void)

Checks whether the FIRC clock error occurs.

Returns

True if the error occurs, false if not.

static inline void CLOCK_ClearFircErr(void)

Clears the FIRC clock error.

static inline bool CLOCK_IsFircValid(void)

Checks whether the FIRC clock is valid.

Returns

True if clock is valid, false if not.

static inline void CLOCK_UnlockFircControlStatusReg(void)

Unlock the FIRCCSR control status register.

static inline void CLOCK_LockFircControlStatusReg(void)

Lock the FIRCCSR control status register.

static inline bool CLOCK_IsFIRCAutoTrimLocked(void)

Check whether FIRC auto trim locked to target frequency range.

When FIRCTREN and FIRCTRUP are enabled, TRIM_LOCK will indicate when auto trimming is complete and output FIRC frequency has locked to target FIRC range. TRIM_LOCK will automatically get cleared if FIRCTREN and FIRCTRUP are not set.

Returns

True if FIRC trim locked to target frequency range, false if not.

status_t CLOCK_InitRosc(const *scg_rosc_config_t* *config)

brief Initializes the SCG ROSC.

This function enables the SCG ROSC clock according to the configuration.

param config Pointer to the configuration structure. retval `kStatus_Success` ROSC is initialized. retval `kStatus_SCG_Busy` ROSC has been enabled and is used by the system clock. retval `kStatus_ReadOnly` ROSC control register is locked.

note This function can't detect whether the system OSC has been enabled and used by an IP.

status_t CLOCK_DeinitRosc(void)

brief De-initializes the SCG ROSC.

This function disables the SCG ROSC clock.

retval *kStatus_Success* System OSC is deinitialized. *retval kStatus_SCG_Busy* System OSC is used by the system clock. *retval kStatus_ReadOnly* System OSC control register is locked.

note This function can't detect whether the ROSC is used by an IP.

uint32_t CLOCK_GetRtcOscFreq(void)

Gets the SCG RTC OSC clock frequency.

Returns

Clock frequency; If the clock is invalid, returns 0.

status_t CLOCK_InitRfFro192M(const *fro192m_rf_clk_config_t* *config)

Initializes the FRO192M clock for the Radio Mode Controller.

This function configure the RF FRO192M clock according to the configuration.

Parameters

- *config* – Pointer to the configuration structure.

Return values

kStatus_Success – RF FRO192M is configured.

uint32_t CLOCK_GetRfFro192MFreq(void)

Gets the FRO192M clock frequency.

Returns

Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK_IsRoscErr(void)

Checks whether the ROSC clock error occurs.

Returns

True if the error occurs, false if not.

static inline void CLOCK_ClearRoscErr(void)

Clears the ROSC clock error.

static inline void CLOCK_SetRoscMonitorMode(*scg_rosc_monitor_mode_t* mode)

Sets the ROSC monitor mode.

This function sets the ROSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- *mode* – Monitor mode to set.

static inline bool CLOCK_IsRoscValid(void)

Checks whether the ROSC clock is valid.

Returns

True if clock is valid, false if not.

static inline void CLOCK_UnlockRoscControlStatusReg(void)

Unlock the ROSCCSR control status register.

static inline void CLOCK_LockRoscControlStatusReg(void)

Lock the ROSCCSR control status register.

```
static inline void CLOCK_SetXtal0Freq(uint32_t freq)
```

Sets the XTAL0 frequency based on board settings.

Parameters

- freq – The XTAL0/EXTAL0 input clock frequency in Hz.

```
static inline void CLOCK_SetXtal32Freq(uint32_t freq)
```

Sets the XTAL32 frequency based on board settings.

Parameters

- freq – The XTAL32/EXTAL32 input clock frequency in Hz.

```
uint32_t divSlow
```

Slow clock divider, see `scg_sys_clk_div_t`.

```
uint32_t divBus
```

Bus clock divider, see `scg_sys_clk_div_t`.

```
uint32_t ___pad0___
```

Reserved.

```
uint32_t divCore
```

Core clock divider, see `scg_sys_clk_div_t`.

```
uint32_t ___pad1___
```

Reserved.

```
uint32_t src
```

System clock source, see `scg_sys_clk_src_t`.

```
uint32_t ___pad2___
```

reserved.

```
uint32_t freq
```

System OSC frequency.

```
uint32_t enableMode
```

Enable mode, OR'ed value of `_scg_sosc_enable_mode`.

```
scg_sosc_monitor_mode_t monitorMode
```

Clock monitor mode selected.

```
scg_rosc_monitor_mode_t monitorMode
```

Clock monitor mode selected.

```
scg_sirc_enable_mode_t enableMode
```

Enable mode, OR'ed value of `_scg_sirc_enable_mode`.

```
scg_firc_trim_mode_t trimMode
```

FIRC trim mode.

```
scg_firc_trim_src_t trimSrc
```

Trim source.

```
uint16_t trimDiv
```

Divider of SOSC for FIRC.

```
uint8_t trimCoar
```

Trim coarse value; Irrelevant if trimMode is `kSCG_FircTrimUpdate`.

```
uint8_t trimFine
```

Trim fine value; Irrelevant if trimMode is `kSCG_FircTrimUpdate`.

uint32_t enableMode

Enable mode.

scg_firc_range_t range

Fast IRC frequency range.

const scg_firc_trim_config_t *trimConfig

Pointer to the FIRC trim configuration; set NULL to disable trim.

fro192m_rf_range_t range

FRO192M RF clock frequency range.

fro192m_rf_clk_div_t apb_rfcmc_div

RF Flash APB and RF_CMC clock divide.

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL

Configure whether driver controls clock.

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note: All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

struct _scg_sys_clk_config

#include <fsl_clock.h> SCG system clock configuration.

struct _scg_sosc_config

#include <fsl_clock.h> SCG system OSC configuration.

struct _scg_rosc_config

#include <fsl_clock.h> SCG ROSC configuration.

struct _scg_sirc_config

#include <fsl_clock.h> SCG slow IRC clock configuration.

struct _scg_firc_trim_config

#include <fsl_clock.h> SCG fast IRC clock trim configuration.

struct _scg_firc_config_t

#include <fsl_clock.h> SCG fast IRC clock configuration.

struct _fro192m_rf_clk_config

#include <fsl_clock.h> FRO192M RF clock configuration.

2.10 CMC: Core Mode Controller Driver

void CMC_SetClockMode(CMC_Type *base, cmc_clock_mode_t mode)

Sets clock mode.

This function config the amount of clock gating when the core asserts Sleeping due to WFI, WFE or SLEEPONEXIT.

Parameters

- base – CMC peripheral base address.
- mode – System clock mode.

```
static inline void CMC_LockClockModeSetting(CMC_Type *base)
```

Locks the clock mode setting.

After invoking this function, any clock mode setting will be blocked.

Parameters

- base – CMC peripheral base address.

```
static inline cmc_core_clock_gate_status_t CMC_GetCoreClockGatedStatus(CMC_Type *base)
```

Gets the core clock gated status.

This function get the status to indicate whether the core clock is gated. The core clock gated status can be cleared by software.

Parameters

- base – CMC peripheral base address.

Returns

The status to indicate whether the core clock is gated.

```
static inline void CMC_ClearCoreClockGatedStatus(CMC_Type *base)
```

Clears the core clock gated status.

This function clear clock status flag by software.

Parameters

- base – CMC peripheral base address.

```
static inline uint8_t CMC_GetWakeupSource(CMC_Type *base)
```

Gets the Wakeup Source.

This function gets the Wakeup sources from the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Wakeup sources from the previous low power mode entry. See `_cmc_wakeup_sources` for details.

```
static inline cmc_clock_mode_t CMC_GetClockMode(CMC_Type *base)
```

Gets the Clock mode.

This function gets the clock mode of the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Low Power status.

```
static inline uint32_t CMC_GetSystemResetStatus(CMC_Type *base)
```

Gets the System reset status.

This function returns the system reset status. Those status updates on every MAIN Warm Reset to indicate the type/source of the most recent reset.

Parameters

- base – CMC peripheral base address.

Returns

The most recent system reset status. See `_cmc_system_reset_sources` for details.

static inline uint32_t CMC_GetStickySystemResetStatus(CMC_Type *base)

Gets the sticky system reset status since the last WAKE Cold Reset.

This function gets all source of system reset that have generated a system reset since the last WAKE Cold Reset, and that have not been cleared by software.

Parameters

- base – CMC peripheral base address.

Returns

System reset status that have not been cleared by software. See `_cmc_system_reset_sources` for details.

static inline void CMC_ClearStickySystemResetStatus(CMC_Type *base, uint32_t mask)

Clears the sticky system reset status flags.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the sticky system reset status to be cleared.

static inline uint8_t CMC_GetResetCount(CMC_Type *base)

Gets the number of reset sequences completed since the last WAKE Cold Reset.

Parameters

- base – CMC peripheral base address.

Returns

The number of reset sequences.

static inline uint32_t CMC_GetResetInitStatusFlags(CMC_Type *base)

Gets status flags to indicate if any errors occurred during the reset initialization sequence.

Parameters

- base – CMC peripheral base address.

Returns

Status flags that indicate errors occurred during the reset initialization sequence.

void CMC_SetPowerModeProtection(CMC_Type *base, uint32_t allowedModes)

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes. This should be done before set the lowPower mode for each power doamin.

The allowed lowpower modes are passed as bit map. For example, to allow Sleep and DeepSleep, use `CMC_SetPowerModeProtection(CMC_base, kCMC_AllowSleepMode|kCMC_AllowDeepSleepMode)`. To allow all low power modes, use `CMC_SetPowerModeProtection(CMC_base, kCMC_AllowAllLowPowerModes)`.

Parameters

- base – CMC peripheral base address.
- allowedModes – Bitmaps of the allowed power modes. See `_cmc_power_mode_protection` for details.

static inline void CMC_LockPowerModeProtectionSetting(CMC_Type *base)

Locks the power mode protection.

This function locks the power mode protection. After invoking this function, any power mode protection setting will be ignored.

Parameters

- base – CMC peripheral base address.

```
static inline void CMC_SetGlobalPowerMode(CMC_Type *base, cmc_low_power_mode_t  
                                         lowPowerMode)
```

Config the same lowPower mode for all power domain.

This function configures the same low power mode for MAIN power domain and WAKE power domain.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See `cmc_low_power_mode_t` for details.

```
static inline void CMC_SetMAINPowerMode(CMC_Type *base, cmc_low_power_mode_t  
                                         lowPowerMode)
```

Configures entry into low power mode for the MAIN Power domain.

This function configures the low power mode for the MAIN power domain, when the core executes WFI/WFE instruction. The available lowPower modes are defined in the `cmc_low_power_mode_t`.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See `cmc_low_power_mode_t` for details.

```
static inline cmc_low_power_mode_t CMC_GetMAINPowerMode(CMC_Type *base)
```

Gets the power mode of the MAIN Power domain.

Parameters

- base – CMC peripheral base address.

Returns

The power mode of MAIN Power domain. See `cmc_low_power_mode_t` for details.

```
static inline void CMC_SetWAKEPowerMode(CMC_Type *base, cmc_low_power_mode_t  
                                         lowPowerMode)
```

Configure entry into low power mode for the WAKE Power domain.

This function configures the low power mode for the WAKE power domain, when the core executes WFI/WFE instruction. The available lowPower mode are defined in the `cmc_low_power_mode_t`.

Note: The lowPower Mode for the WAKE domain must not be configured to a lower power mode than any other power domain.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See `cmc_low_power_mode_t` for details.

```
static inline cmc_low_power_mode_t CMC_GetWAKEPowerMode(CMC_Type *base)
```

Gets the power mode of the WAKE Power domain.

Parameters

- base – CMC peripheral base address.

Returns

The power mode of WAKE Power domain. See `cmc_low_power_mode_t` for details.

```
void CMC_ConfigResetPin(CMC_Type *base, const cmc_reset_pin_config_t *config)
```

Configure reset pin.

This function configures reset pin. When enabled, the low power filter is enabled in both Active and Low power modes, the reset filter is only enabled in Active mode. When both filters are enabled, they operate in series.

Parameters

- `base` – CMC peripheral base address.
- `config` – Pointer to the reset pin config structure.

```
static inline void CMC_EnableSystemResetInterrupt(CMC_Type *base, uint32_t mask)
```

Enable system reset interrupts.

This function enables the system reset interrupts. The assertion of non-fatal warm reset can be delayed for 258 cycles of the 32K_CLK clock while an enabled interrupt is generated. Then Software can perform a graceful shutdown or abort the non-fatal warm reset provided the pending reset source is cleared by resetting the reset source and then clearing the pending flag.

Parameters

- `base` – CMC peripheral base address.
- `mask` – System reset interrupts. See `_cmc_system_reset_interrupt_enable` for details.

```
static inline void CMC_DisableSystemResetInterrupt(CMC_Type *base, uint32_t mask)
```

Disable system reset interrupts.

This function disables the system reset interrupts.

Parameters

- `base` – CMC peripheral base address.
- `mask` – System reset interrupts. See `_cmc_system_reset_interrupt_enable` for details.

```
static inline uint32_t CMC_GetSystemResetInterruptFlags(CMC_Type *base)
```

Gets System Reset interrupt flags.

This function returns the System reset interrupt flags.

Parameters

- `base` – CMC peripheral base address.

Returns

System reset interrupt flags. See `_cmc_system_reset_interrupt_flag` for details.

```
static inline void CMC_ClearSystemResetInterruptFlags(CMC_Type *base, uint32_t mask)
```

Clears System Reset interrupt flags.

This function clears system reset interrupt flags. The pending reset source can be cleared by resetting the source of the reset and then clearing the pending flags.

Parameters

- `base` – CMC peripheral base address.
- `mask` – System Reset interrupt flags. See `_cmc_system_reset_interrupt_flag` for details.

static inline void CMC_EnableNonMaskablePinInterrupt(CMC_Type *base, bool enable)
Enable/Disable Non maskable Pin interrupt.

Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Non maskable pin interrupt. true - enable Non-maskable pin interrupt. false - disable Non-maskable pin interrupt.

static inline uint8_t CMC_GetISPMODEPinLogic(CMC_Type *base)
Gets the logic state of the ISPMODE_n pin.

This function returns the logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

Parameters

- base – CMC peripheral base address.

Returns

The logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

static inline void CMC_ClearISPMODEPinLogic(CMC_Type *base)
Clears ISPMODE_n pin state.

Parameters

- base – CMC peripheral base address.

static inline void CMC_ForceBootConfiguration(CMC_Type *base, bool assert)
Set the logic state of the BOOT_CONFIGn pin.

This function force the logic state of the Boot_Config pin to assert on next system reset.

Parameters

- base – CMC peripheral base address.
- assert – Assert the corresponding pin or not. true - Assert corresponding pin on next system reset. false - No effect.

static inline void CMC_LockWriteOperationToBootRomStatusReg(CMC_Type *base, uint8_t index)

Lock write operation to BootROM status register and BootROM Lock register.

Note: If locked, BootROM status register cannot be written.

Note: Once locked, only cold reset can reset related register.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.

static inline bool CMC_CheckBootRomStatusRegWriteLocked(CMC_Type *base, uint8_t index)
Check if BootROM status register can be written.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.

Return values

- `true` – The selected BootRom status register is locked and cannot be written.
- `false` – The selected BootRom Status register is unlocked and cannot be written.

static inline uint32_t CMC_GetBootRomStatus(CMC_Type *base, uint8_t index)

Gets the information written by the BootROM.

Parameters

- `base` – CMC peripheral base address.
- `index` – The index of BootROM status register, ranges from 0.

Returns

The status information written by the BootROM.

static inline void CMC_WriteBootRomStatusReg(CMC_Type *base, uint8_t index, uint32_t value)

Writes value to BootROM status register, in this way, BootROM status registers are used as general purpose register.

Note: Value in BootROM status registers are reset in cold reset.

Parameters

- `base` – CMC peripheral base address.
- `index` – The index of BootROM status register, ranges from 0.
- `value` – Value to write.

void CMC_PowerOffSRAMAllMode(CMC_Type *base, uint32_t mask)

Power off the selected system SRAM always.

This function power off the selected system SRAM always. The SRAM arrays should not be accessed while they are shut down. SRAM array contents are not retained if they are powered off.

Parameters

- `base` – CMC peripheral base address.
- `mask` – Bitmap of the SRAM arrays to be powered off all modes. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC_PowerOnSRAMAllMode(CMC_Type *base, uint32_t mask)

Power on SRAM during all mode.

Parameters

- `base` – CMC peripheral base address.
- `mask` – Bitmap of the SRAM arrays to be powered on all modes. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC_PowerOffSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)

Power off the selected system SRAM during low power mode only.

This function power off the selected system SRAM only during low power mode. SRAM array contents are not retained if they are power off.

Parameters

- `base` – CMC peripheral base address.

- `mask` – Bitmap of the SRAM arrays to be power off during low power mode only. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

```
static inline void CMC_PowerOnSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)
```

Power on the selected system SRAM during low power mode only.

This function power on the selected system SRAM. The SRAM array contents are retained in low power modes.

Parameters

- `base` – CMC peripheral base address.
- `mask` – Bitmap of the SRAM arrays to be power on during low power mode only. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

```
void CMC_ConfigFlashMode(CMC_Type *base, bool wake, bool doze, bool disable)
```

Config the low power mode of the on-chip flash memory.

This function config the low power mode of the on-chip flash memory.

Parameters

- `base` – CMC peripheral base address.
- `wake` – true: Flash will exit low power state during the flash memory accesses. false: No effect.
- `doze` – true: Flash is disabled while core is sleeping false: No effect.
- `disable` – true: Flash memory is placed in low power state. false: No effect.

```
static inline void CMC_EnableDebugOperation(CMC_Type *base, bool enable)
```

Enables/Disables debug Operation when the core sleep.

This function config what happens to debug when core sleeps.

Parameters

- `base` – CMC peripheral base address.
- `enable` – Enable or disable Debug when Core is sleeping. true - Debug remains enabled when the core is sleeping. false - Debug is disabled when the core is sleeping.

```
void CMC_PreEnterLowPowerMode(void)
```

Prepares to enter low power modes.

This function should be called before entering low power modes.

```
void CMC_PostExitLowPowerMode(void)
```

Recovers after wake up from low power modes.

This function should be called after wake up from low power modes. This function should be used with `CMC_PreEnterLowPowerMode()`

```
void CMC_GlobalEnterLowPowerMode(CMC_Type *base, cmc_low_power_mode_t lowPowerMode)
```

Config the entry into the same low power mode for each power domains.

This function provides the feature to entry into the same low power mode for each power domains. Before invoking this function, please ensure the selected power mode have been allowed.

Parameters

- `base` – CMC peripheral base address.

- `lowPowerMode` – The low power mode to be entered. See `cmc_low_power_mode_t` for the details.

`void CMC_EnterLowPowerMode(CMC_Type *base, const cmc_power_domain_config_t *config)`
Configures the entry into different low power modes for each power domains.

This function provides the feature to entry into different low power modes for each power domains. Before invoking this function please ensure the selected modes are allowed.

Parameters

- `base` – CMC peripheral base address.
- `config` – Pointer to the `cmc_power_domain_config_t` structure.

`FSL_CMC_DRIVER_VERSION`

CMC driver version 2.4.3.

`CMC_SRAM_BUSY_TIMEOUT`

Max loops to wait for CMC SRAM operation complete.

When configuring the SRAM, driver will wait for the completion of new settings. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

`enum _cmc_power_mode_protection`

CMC power mode Protection enumeration.

Values:

enumerator `kCMC_AllowSleepMode`

Allow Sleep mode.

enumerator `kCMC_AllowDeepSleepMode`

Allow Deep Sleep mode.

enumerator `kCMC_AllowPowerDownMode`

Allow Power Down mode.

enumerator `kCMC_AllowDeepPowerDownMode`

Allow Deep Power Down mode.

enumerator `kCMC_AllowAllLowPowerModes`

Allow all low power modes.

`enum _cmc_wakeup_sources`

Wake up sources from the previous low power mode entry.

Values:

enumerator `kCMC_WakeupFromResetInterruptOrPowerDown`

Wakeup source is reset interrupt, or wake up from [Deep] Power Down.

enumerator `kCMC_WakeupFromDebugRequest`

Wakeup source is debug request.

enumerator `kCMC_WakeupFromInterrupt`

Wakeup source is interrupt.

enumerator `kCMC_WakeupFromDMAWakeup`

Wakeup source is DMA Wakeup.

enumerator `kCMC_WakeupFromWUURquest`

Wakeup source is WUU request.

enumerator kCMC_WakeupFromBusMaster

Wakeup source is Bus master.

enum _cmc_system_reset_interrupt_enable

System Reset Interrupt enable enumeration.

Values:

enumerator kCMC_PinResetInterruptEnable

Pin Reset interrupt enable.

enumerator kCMC_DAPResetInterruptEnable

DAP Reset interrupt enable.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetInterruptEnable

Low Power Acknowledge Timeout Reset interrupt enable.

enumerator kCMC_SystemClockGenerationResetInterruptEnable

System Clock Generation Reset interrupt enable.

enumerator kCMC_Watchdog0ResetInterruptEnable

Watchdog 0 Reset interrupt enable.

enumerator kCMC_SoftwareResetInterruptEnable

Software Reset interrupt enable.

enumerator kCMC_LockupResetInterruptEnable

Lockup Reset interrupt enable.

enumerator kCMC_Watchdog1ResetInterruptEnable

Watchdog 1 Reset interrupt enable

enum _cmc_system_reset_interrupt_flag

CMC System Reset Interrupt Status flag.

Values:

enumerator kCMC_PinResetInterruptFlag

Pin Reset interrupt flag.

enumerator kCMC_DAPResetInterruptFlag

DAP Reset interrupt flag.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetFlag

Low Power Acknowledge Timeout Reset interrupt flag.

enumerator kCMC_Watchdog0ResetInterruptFlag

Watchdog 0 Reset interrupt flag.

enumerator kCMC_SoftwareResetInterruptFlag

Software Reset interrupt flag.

enumerator kCMC_LockupResetInterruptFlag

Lock up Reset interrupt flag.

enumerator kCMC_Watchdog1ResetInterruptFlag

Watchdog 1 Reset interrupt flag.

enum _cmc_system_sram_arrays

CMC System SRAM arrays low power mode enable enumeration.

Values:

enumerator kCMC_SRAMBank0

Power off SRAM Bank0, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank1

Power off SRAM Bank1, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank2

Power off SRAM Bank2, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank3

Power off SRAM Bank3, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank4

Power off SRAM Bank4, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank5

Power off SRAM Bank5, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank6

Power off SRAM Bank6, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank7

Power off SRAM Bank7, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank8

Power off SRAM Bank8, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank9

Power off SRAM Bank9, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank10

Power off SRAM Bank10, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_AllSramArrays

Mask of all system SRAM arrays.

enum _cmc_system_reset_sources

System reset sources enumeration.

Values:

enumerator kCMC_WakeUpReset

The reset caused by a wakeup from Power Down or Deep Power Down mode.

enumerator kCMC_PORReset

The reset caused by power on reset detection logic.

enumerator kCMC_LVDRReset

The reset caused by a Low Voltage Detect.

enumerator kCMC_HVDRReset

The reset caused by a High voltage Detect.

enumerator kCMC_WarmReset

The last reset source is a warm reset source.

enumerator kCMC_FatalReset

The last reset source is a fatal reset source.

enumerator kCMC_PinReset

The reset caused by the RESET_b pin.

enumerator kCMC_DAPReset

The reset caused by a reset request from the Debug Access port.

enumerator kCMC_ResetTimeout

The reset caused by a timeout or other error condition in the system reset generation.

enumerator kCMC_LowPowerAcknowledgeTimeoutReset

The reset caused by a timeout in low power mode entry logic.

enumerator kCMC_SCGRReset

The reset caused by a loss of clock or loss of lock event in the SCG.

enumerator kCMC_Watchdog0Reset

The reset caused by a WatchDog 0 timeout.

enumerator kCMC_SoftwareReset

The reset caused by a software reset request.

enumerator kCMC_LockUpReset

The reset caused by the ARM core indication of a LOCKUP event.

enumerator kCMC_Watchdog1Reset

The reset caused by a WatchDog 1 timeout.

enum _cmc_core_clock_gate_status

Indicate the core clock was gated.

Values:

enumerator kCMC_CoreClockNotGated

Core clock not gated.

enumerator kCMC_CoreClockGated

Core clock was gated due to low power mode entry.

enum _cmc_clock_mode

CMC clock mode enumeration.

Values:

enumerator kCMC_GateNoneClock

No clock gating.

enumerator kCMC_GateCoreClock

Gate Core clock.

enumerator kCMC_GateCorePlatformClock

Gate Core clock and platform clock.

enumerator kCMC_GateAllSystemClocks

Gate all System clocks, without getting core entering into low power mode.

enumerator kCMC_GateAllSystemClocksEnterLowPowerMode

Gate all System clocks, with core entering into low power mode.

enum _cmc_low_power_mode

CMC power mode enumeration.

Values:

enumerator kCMC_ActiveMode

Select Active mode.

enumerator kCMC_SleepMode

Select Sleep mode when a core executes WFI or WFE instruction.

enumerator kCMC_DeepSleepMode

Select Deep Sleep mode when a core executes WFI or WFE instruction.

enumerator kCMC_PowerDownMode

Select Power Down mode when a core executes WFI or WFE instruction.

enumerator kCMC_DeepPowerDown

Select Deep Power Down mode when a core executes WFI or WFE instruction.

typedef enum *_cmc_core_clock_gate_status* cmc_core_clock_gate_status_t

Indicate the core clock was gated.

typedef enum *_cmc_clock_mode* cmc_clock_mode_t

CMC clock mode enumeration.

typedef enum *_cmc_low_power_mode* cmc_low_power_mode_t

CMC power mode enumeration.

typedef struct *_cmc_reset_pin_config* cmc_reset_pin_config_t

CMC reset pin configuration.

typedef struct *_cmc_power_domain_config* cmc_power_domain_config_t

power mode configuration for each power domain.

CMC_BLR_LOCK_FIELD_WIDTH

CMC_BLR_LOCK_IDX_MASK(index)

CMC_BLR_LOCK_IDX_SHIFT(index)

CMC_BLR_LOCK_IDX(index, value)

struct *_cmc_reset_pin_config*

#include <fsl_cmc.h> CMC reset pin configuration.

Public Members

bool lowpowerFilterEnable

Low Power Filter enable.

bool resetFilterEnable

Reset Filter enable.

uint8_t resetFilterWidth

Width of the Reset Filter.

struct *_cmc_power_domain_config*

#include <fsl_cmc.h> power mode configuration for each power domain.

Public Members

cmc_clock_mode_t clock_mode

Clock mode for each power domain.

cmc_low_power_mode_t main_domain

The low power mode of the MAIN power domain.

cmc_low_power_mode_t wake_domain

The low power mode of the WAKE power domain.

2.11 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- Version 2.0.4
 - Release peripheral from reset if necessary in init function.
- Version 2.0.3
 - Fix MISRA issues
- Version 2.0.2
 - Fix MISRA issues
- Version 2.0.1
 - move DATA and DATALL macro definition from header file to source file

enum `_crc_bits`

CRC bit width.

Values:

enumerator `kCrcBits16`

Generate 16-bit CRC code

enumerator `kCrcBits32`

Generate 32-bit CRC code

enum `_crc_result`

CRC result type.

Values:

enumerator `kCrcFinalChecksum`

CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

enumerator `kCrcIntermediateChecksum`

CRC data register read value is intermediate checksum (raw value). Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for `CRC_Init()` to continue adding data to this checksum.

typedef enum `_crc_bits` `crc_bits_t`

CRC bit width.

typedef enum `_crc_result` `crc_result_t`

CRC result type.

typedef struct `_crc_config` `crc_config_t`

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void `CRC_Init(CRC_Type *base, const crc_config_t *config)`

Enables and configures the CRC peripheral module.

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

- base – CRC peripheral address.
- config – CRC module configuration structure.

static inline void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

This function disables the clock gate in the SIM module for the CRC peripheral.

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

Loads default values to the CRC protocol configuration structure. The default values are as follows.

```
config->polynomial = 0x1021;
config->seed = 0xFFFF;
config->reflectIn = false;
config->reflectOut = false;
config->complementChecksum = false;
config->crcBits = kCrcBits16;
config->crcResult = kCrcFinalChecksum;
```

Parameters

- config – CRC protocol configuration structure.

void CRC_WriteData(CRC_Type *base, const uint8_t *data, size_t dataSize)

Writes data to the CRC module.

Writes input data buffer bytes to the CRC data register. The configured type of transpose is applied.

Parameters

- base – CRC peripheral address.
- data – Input data stream, MSByte in data[0].
- dataSize – Size in bytes of the input data buffer.

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 32-bit checksum, after configured transpose and complement operations.

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 16-bit checksum, after configured transpose and complement operations.

`CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT`

Default configuration structure filled by `CRC_GetDefaultConfig()`. Use `CRC16-CCIT-FALSE` as default.

`struct _crc_config`

`#include <fsl_crc.h>` CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

`uint32_t` polynomial

CRC Polynomial, MSBit first. Example polynomial: $0x1021 = 1_0000_0010_0001 = x^{12} + x^5 + 1$

`uint32_t` seed

Starting checksum value

`bool` reflectIn

Reflect bits on input.

`bool` reflectOut

Reflect bits on output.

`bool` complementChecksum

True if the result shall be complement of the actual checksum.

`crc_bits_t` crcBits

Selects 16- or 32- bit CRC protocol.

`crc_result_t` crcResult

Selects final or intermediate checksum return from `CRC_Get16bitResult()` or `CRC_Get32bitResult()`

2.12 EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

`void EDMA_Init(DMA_Type *base, const edma_config_t *config)`

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure.

Note: This function enables the minor loop map feature.

Parameters

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see “`edma_config_t`”.

`void EDMA_Deinit(DMA_Type *base)`
Deinitializes the eDMA peripheral.
This function gates the eDMA clock.

Parameters

- `base` – eDMA peripheral base address.

`void EDMA_InstallTCD(DMA_Type *base, uint32_t channel, edma_tcd_t *tcd)`
Push content of TCD structure into hardware TCD register.

Parameters

- `base` – EDMA peripheral base address.
- `channel` – EDMA channel number.
- `tcd` – Point to TCD structure.

`void EDMA_GetDefaultConfig(edma_config_t *config)`
Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values:

```
config.enableMasterIdReplication = true;  
config.enableHaltOnError = true;  
config.enableRoundRobinArbitration = false;  
config.enableDebugMode = false;  
config.enableBufferedWrites = false;
```

Parameters

- `config` – A pointer to the eDMA configuration structure.

`static inline void EDMA_EnableAllChannelLink(DMA_Type *base, bool enable)`
Enables/disables all channel linking.

This function enables/disables all channel linking in the management page. For specific channel linking enablement & configuration, please refer to `EDMA_SetChannelLink` and `EDMA_TcdSetChannelLink` APIs.

For example, to disable all channel linking in the DMA0 management page:

```
EDMA_EnableAllChannelLink(DMA0, false);
```

Parameters

- `base` – eDMA peripheral base address.
- `enable` – Switcher of the channel linking feature for all channels. “true” means to enable. “false” means not.

`void EDMA_ResetChannel(DMA_Type *base, uint32_t channel)`
Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

Note: This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
void EDMA_SetTransferConfig(DMA_Type *base, uint32_t channel, const edma_transfer_config_t
                          *config, edma_tcd_t *nextTcd)
```

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_config_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

Note: If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA_ResetChannel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_SetMinorOffsetConfig(DMA_Type *base, uint32_t channel, const
                              edma_minor_offset_config_t *config)
```

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

```
static inline void EDMA_SetChannelArbitrationGroup(DMA_Type *base, uint32_t channel,
                                                  uint32_t group)
```

Configures the eDMA channel arbitration group.

This function configures the channel arbitration group. The arbitration group priorities are evaluated by numeric value from highest group number to lowest.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- group – Fixed-priority arbitration group number for the channel.

```
static inline void EDMA_SetChannelPreemptionConfig(DMA_Type *base, uint32_t channel, const
                                                    edma_channel_Preemption_config_t
                                                    *config)
```

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

```
static inline uint32_t EDMA_GetChannelSystemBusInformation(DMA_Type *base, uint32_t
                                                         channel)
```

Gets the eDMA channel identification and attribute information on the system bus interface.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of the channel system bus information. Users need to use the `_edma_channel_sys_bus_info` type to decode the return variables.

```
void EDMA_SetChannelLink(DMA_Type *base, uint32_t channel, edma_channel_link_type_t
                        type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
 - kEDMA_LinkNone
 - kEDMA_MinorLink
 - kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
void EDMA_SetBandWidth(DMA_Type *base, uint32_t channel, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
 - kEDMABandwidthStallNone
 - kEDMABandwidthStall4Cycle
 - kEDMABandwidthStall8Cycle

```
void EDMA_SetModulo(DMA_Type *base, uint32_t channel, edma_modulo_t srcModulo,
                   edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA_EnableAsyncRequest(DMA_Type *base, uint32_t channel, bool enable)
```

Enables an async request for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

```
static inline void EDMA_EnableAutoStopRequest(DMA_Type *base, uint32_t channel, bool
                                              enable)
```

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

```
void EDMA_EnableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)
```

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA_DisableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)
```

Disables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined `edma_interrupt_enable_t` type.

```
static inline void EDMA_SetChannelMux(DMA_Type *base, uint32_t channel, uint32_t mux)
```

Set channel mux source.

Note:When the peripheral is no longer needed, the mux configuration for that channel should be written to 0, thus releasing the resource.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mux – the mux source value is SOC specific, please reference the SOC for detail.

```
void EDMA_TcdReset(edma_tcd_t *tcd)
```

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

```
void EDMA_TcdSetTransferConfig(edma_tcd_t *tcd, const edma_transfer_config_t *config,  
                               edma_tcd_t *nextTcd)
```

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The STCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_config_t config = {  
    ...  
}  
edma_tcd_t tcd __aligned(32);  
edma_tcd_t nextTcd __aligned(32);  
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

Parameters

- `tcd` – Pointer to the TCD structure.
- `config` – Pointer to eDMA transfer configuration structure.
- `nextTcd` – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_TcdSetMinorOffsetConfig(edma_tcd_t *tcd, const edma_minor_offset_config_t
                                *config)
```

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- `tcd` – A point to the TCD structure.
- `config` – A pointer to the minor offset configuration structure.

```
void EDMA_TcdSetChannelLink(edma_tcd_t *tcd, edma_channel_link_type_t type, uint32_t
                           linkedChannel)
```

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- `tcd` – Point to the TCD structure.
- `type` – Channel link type, it can be one of:
 - `kEDMA_LinkNone`
 - `kEDMA_MinorLink`
 - `kEDMA_MajorLink`
- `linkedChannel` – The linked channel number.

```
static inline void EDMA_TcdSetBandWidth(edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
 - `kEDMABandwidthStallNone`
 - `kEDMABandwidthStall4Cycle`
 - `kEDMABandwidthStall8Cycle`

```
void EDMA_TcdSetModulo(edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t
destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA_TcdEnableAutoStopRequest(edma_tcd_t *tcd, bool enable)
```

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

```
void EDMA_TcdEnableInterrupts(edma_tcd_t *tcd, uint32_t mask)
```

Enables the interrupt source for the eDMA TCD.

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined *edma_interrupt_enable_t* type.

```
void EDMA_TcdDisableInterrupts(edma_tcd_t *tcd, uint32_t mask)
```

Disables the interrupt source for the eDMA TCD.

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined *edma_interrupt_enable_t* type.

```
static inline void EDMA_EnableChannelRequest(DMA_Type *base, uint32_t channel)
```

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
static inline void EDMA_DisableChannelRequest(DMA_Type *base, uint32_t channel)
```

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
static inline void EDMA_TriggerChannelStart(DMA_Type *base, uint32_t channel)
```

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
uint32_t EDMA_GetRemainingMajorLoopCount(DMA_Type *base, uint32_t channel)
```

Gets the Remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

- a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTES(initially configured)
-

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

Major loop count which has not been transferred yet for the current TCD.

```
static inline uint32_t EDMA_GetErrorStatusFlags(DMA_Type *base)
```

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

The mask of error status flags. Users need to use the `_edma_error_status_flags` type to decode the return variables.

```
uint32_t EDMA_GetChannelStatusFlags(DMA_Type *base, uint32_t channel)
```

Gets the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

void EDMA_ClearChannelStatusFlags(DMA_Type *base, uint32_t channel, uint32_t mask)

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined `_edma_channel_status_flags` type.

void EDMA_CreateHandle(*edma_handle_t* *handle, DMA_Type *base, uint32_t channel)

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

void EDMA_InstallTCDMemory(*edma_handle_t* *handle, *edma_tcd_t* *tcdPool, uint32_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the EDMA_CreateHandle to use scatter/gather feature.

Parameters

- handle – eDMA handle pointer.
- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

void EDMA_SetCallback(*edma_handle_t* *handle, *edma_callback* callback, void *userData)

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.
- userData – A parameter for the callback function.

void EDMA_PrepareTransferConfig(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- `config` – The user configuration structure of type `edma_transfer_config_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – eDMA transfer source address offset
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `destOffset` – eDMA transfer destination address offset
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.

```
void EDMA_PrepareTransfer(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth,  
                        void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest,  
                        uint32_t transferBytes, edma_transfer_type_t transferType)
```

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- `config` – The user configuration structure of type `edma_transfer_config_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `transferType` – eDMA transfer type.

```
status_t EDMA_SubmitTransfer(edma_handle_t *handle, const edma_transfer_config_t *config)
```

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. If submitting the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

- `handle` – eDMA handle pointer.
- `config` – Pointer to eDMA transfer configuration structure.

Return values

- `kStatus_EDMA_Success` – It means submit transfer request succeed.
- `kStatus_EDMA_QueueFull` – It means TCD queue is full. Submit transfer request is not allowed.
- `kStatus_EDMA_Busy` – It means the given channel is busy, need to submit request later.

void EDMA_StartTransfer(*edma_handle_t* *handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

void EDMA_StopTransfer(*edma_handle_t* *handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call EDMA_StartTransfer() again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

void EDMA_AbortTransfer(*edma_handle_t* *handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns

The next TCD address.

static inline *edma_transfer_size_t* EDMA_GetTransferSize(uint32_t width)

Get the transfer size.

This function gets the transfer size.

Parameters

- width – transfer width(bytes).

Returns

The transfer size.

```
void EDMA_HandleIRQ(edma_handle_t *handle)
```

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Parameters

- handle – eDMA handle pointer.

```
FSL_EDMA_DRIVER_VERSION
```

eDMA driver version

Version 2.5.1.

```
enum _edma_transfer_size
```

eDMA transfer configuration

Values:

```
enumerator kEDMA_TransferSize1Bytes
```

Source/Destination data transfer size is 1 byte every time

```
enumerator kEDMA_TransferSize2Bytes
```

Source/Destination data transfer size is 2 bytes every time

```
enumerator kEDMA_TransferSize4Bytes
```

Source/Destination data transfer size is 4 bytes every time

```
enumerator kEDMA_TransferSize8Bytes
```

Source/Destination data transfer size is 8 bytes every time

```
enumerator kEDMA_TransferSize16Bytes
```

Source/Destination data transfer size is 16 bytes every time

```
enumerator kEDMA_TransferSize32Bytes
```

Source/Destination data transfer size is 32 bytes every time

```
enumerator kEDMA_TransferSize64Bytes
```

Source/Destination data transfer size is 64 bytes every time

```
enum _edma_modulo
```

eDMA modulo configuration

Values:

```
enumerator kEDMA_ModuloDisable
```

Disable modulo

```
enumerator kEDMA_Modulo2bytes
```

Circular buffer size is 2 bytes.

```
enumerator kEDMA_Modulo4bytes
```

Circular buffer size is 4 bytes.

```
enumerator kEDMA_Modulo8bytes
```

Circular buffer size is 8 bytes.

```
enumerator kEDMA_Modulo16bytes
```

Circular buffer size is 16 bytes.

```
enumerator kEDMA_Modulo32bytes
```

Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes

Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes

Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes

Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes

Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes

Circular buffer size is 2 G bytes.

enum _edma_bandwidth

Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone

No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle

eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle

eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type

Channel link type.

Values:

enumerator kEDMA_LinkNone

No channel link

enumerator kEDMA_MinorLink

Channel link after each minor loop

enumerator kEDMA_MajorLink

Channel link while major loop count exhausted

eDMA channel status flags, _edma_channel_status_flags

Values:

enumerator kEDMA_DoneFlag

DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag

eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag

eDMA interrupt flag, set while an interrupt occurred of this channel

eDMA channel error status flags, _edma_error_status_flags

Values:

enumerator kEDMA_DestinationBusErrorFlag

Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag
Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_ValidFlag
No error occurred, this bit is 0. Otherwise, it is 1.

eDMA channel system bus information, `_edma_channel_sys_bus_info`

Values:

enumerator kEDMA_AttributeOutput
DMA's AHB system bus attribute output value.

enumerator kEDMA_PrivilegedAccessLevel
Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA_MasterId
DMA's master ID when channel is active and master ID replication is enabled.

enum `_edma_interrupt_enable`

eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum `_edma_transfer_type`

eDMA transfer type

Values:

enumerator `kEDMA_MemoryToMemory`
Transfer from memory to memory

enumerator `kEDMA_PeripheralToMemory`
Transfer from peripheral to memory

enumerator `kEDMA_MemoryToPeripheral`
Transfer from memory to peripheral

enumerator `kEDMA_PeripheralToPeripheral`
Transfer from Peripheral to peripheral

eDMA transfer status, `_edma_transfer_status`

Values:

enumerator `kStatus_EDMA_QueueFull`
TCD queue is full.

enumerator `kStatus_EDMA_Busy`
Channel is busy and can't handle the transfer request.

typedef enum `_edma_transfer_size` `edma_transfer_size_t`
eDMA transfer configuration

typedef enum `_edma_modulo` `edma_modulo_t`
eDMA modulo configuration

typedef enum `_edma_bandwidth` `edma_bandwidth_t`
Bandwidth control.

typedef enum `_edma_channel_link_type` `edma_channel_link_type_t`
Channel link type.

typedef enum `_edma_interrupt_enable` `edma_interrupt_enable_t`
eDMA interrupt source

typedef enum `_edma_transfer_type` `edma_transfer_type_t`
eDMA transfer type

typedef struct `_edma_config` `edma_config_t`
eDMA global configuration structure.

typedef struct `_edma_transfer_config` `edma_transfer_config_t`
eDMA transfer configuration
This structure configures the source/destination transfer attribute.

typedef struct `_edma_channel_Preemption_config` `edma_channel_Preemption_config_t`
eDMA channel priority configuration

typedef struct `_edma_minor_offset_config` `edma_minor_offset_config_t`
eDMA minor offset configuration

typedef struct `_edma_tcd` `edma_tcd_t`
eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

typedef void (*`edma_callback`)(struct `_edma_handle` *`handle`, void *`userData`, bool `transferDone`, uint32_t `tcds`)

Define callback function for eDMA.

```
typedef uint32_t (*edma_memorymap_callback)(uint32_t addr)
```

Memory map function callback for DMA.

```
typedef struct _edma_handle edma_handle_t
```

eDMA transfer handle structure

```
struct _edma_config
```

#include <fsl_edma.h> eDMA global configuration structure.

Public Members

```
bool enableMasterIdReplication
```

Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for DMA transfers is used.

```
bool enableHaltOnError
```

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

```
bool enableRoundRobinArbitration
```

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection

```
bool enableDebugMode
```

Enable(true) eDMA debug mode. When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

```
struct _edma_transfer_config
```

#include <fsl_edma.h> eDMA transfer configuration

This structure configures the source/destination transfer attribute.

Public Members

```
uint32_t srcAddr
```

Source data address.

```
uint32_t destAddr
```

Destination data address.

```
edma_transfer_size_t srcTransferSize
```

Source data transfer size.

```
edma_transfer_size_t destTransferSize
```

Destination data transfer size.

```
int16_t srcOffset
```

Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

```
int16_t destOffset
```

Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

```
uint32_t minorLoopBytes
```

Bytes to transfer in a minor loop

```
uint32_t majorLoopCounts
```

Major loop iteration count.

```
struct _edma_channel_Preemption_config
```

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

`bool enableChannelPreemption`

If true: a channel can be suspended by other channel with higher priority

`bool enablePreemptAbility`

If true: a channel can suspend other channel with low priority

`uint8_t channelPriority`

Channel priority

`struct _edma_minor_offset_config`

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

`bool enableSrcMinorOffset`

Enable(true) or Disable(false) source minor loop offset.

`bool enableDestMinorOffset`

Enable(true) or Disable(false) destination minor loop offset.

`uint32_t minorOffset`

Offset for a minor loop mapping.

`struct _edma_tcd`

#include <fsl_edma.h> eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

Public Members

`__IO uint32_t SADDR`

SADDR register, used to save source address

`__IO uint16_t SOFF`

SOFF register, save offset bytes every transfer

`__IO uint16_t ATTR`

ATTR register, source/destination transfer size and modulo

`__IO uint32_t NBYTES`

Nbytes register, minor loop length in bytes

`__IO uint32_t SLAST`

SLAST register

`__IO uint32_t DADDR`

DADDR register, used for destination address

`__IO uint16_t DOFF`

DOFF register, used for destination offset

`__IO uint16_t CITER`

CITER register, current minor loop numbers, for unfinished minor loop.

`__IO uint32_t DLAST_SGA`

DLASTSGA register, next stcd address used in scatter-gather mode

__IO uint16_t CSR
CSR register, for TCD control status

__IO uint16_t BITER
BITER register, begin minor loop count.

struct _edma_handle
#include <fsl_edma.h> eDMA transfer handle structure

Public Members

edma_callback callback
Callback function for major count exhausted.

void *userData
Callback function parameter.

DMA_Type *base
eDMA peripheral base address.

edma_tcd_t *tcdPool
Pointer to memory stored TCDs.

uint8_t channel
eDMA channel number.

volatile int8_t header
The first TCD index.

volatile int8_t tail
The last TCD index.

volatile int8_t tcdUsed
The number of used TCD slots.

volatile int8_t tcdSize
The total number of TCD slots in the queue.

uint8_t flags
The status of the current channel.

2.13 ELEMU: Edgelock Messaging unit driver

2.14 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const *ewm_config_t* *config)
Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

[ewm_config_t](#)

Parameters

- config – Pointer to the EWM configuration structure.

static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined
 - kEWM_InterruptEnable

static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address

- mask – The interrupts to disable The parameter can be combination of the following source if defined
 - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;  
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

`_ewm_status_flags_t`

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)

Services the EWM.

This function resets the EWM counter to zero.

Parameters

- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION

EWM driver version 2.0.4.

enum _ewm_lpo_clock_source

Describes EWM clock source.

Values:

enumerator kEWM_LpoClockSource0
EWM clock sourced from lpo_clk[0]

enumerator kEWM_LpoClockSource1
EWM clock sourced from lpo_clk[1]

enumerator kEWM_LpoClockSource2
EWM clock sourced from lpo_clk[2]

enumerator kEWM_LpoClockSource3
EWM clock sourced from lpo_clk[3]

enum _ewm_interrupt_enable_t

EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable
Enable the EWM to generate an interrupt

enum `_ewm_status_flags_t`

EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator `kEWM_RunningFlag`

Running flag, set when EWM is enabled

typedef enum `_ewm_lpo_clock_source` `ewm_lpo_clock_source_t`

Describes EWM clock source.

typedef struct `_ewm_config` `ewm_config_t`

Data structure for EWM configuration.

This structure is used to configure the EWM.

struct `_ewm_config`

#include `<fsl_ewm.h>` Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

bool `enableEwm`

Enable EWM module

bool `enableEwmInput`

Enable EWM_in input

bool `setInputAssertLogic`

EWM_in signal assertion state

bool `enableInterrupt`

Enable EWM interrupt

`ewm_lpo_clock_source_t` `clockSource`

Clock source select

uint8_t `prescaler`

Clock prescaler value

uint8_t `compareLowValue`

Compare low-register value

uint8_t `compareHighValue`

Compare high-register value

2.15 FGPIO Driver

2.16 C40ESP3 Flash Driver

enum `_flash_driver_version_constants`

Flash driver version for ROM.

Values:

enumerator kFLASH_DriverVersionName
Flash driver version name.

enumerator kFLASH_DriverVersionMajor
Major flash driver version.

enumerator kFLASH_DriverVersionMinor
Minor flash driver version.

enumerator kFLASH_DriverVersionBugfix
Bugfix for flash driver version.

enum _flash_property_tag
Enumeration for various flash properties.

Values:

enumerator kFLASH_PropertyPflash0SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash0TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash0BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash0BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash0BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash0FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash0AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash0AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyPflash1SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash1TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash1BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash1BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash1BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash1FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash1AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash1AccessSegmentCount

Pflash access segment count property.

enumerator kFLASH_PropertyFlexRamBlockBaseAddr

FlexRam block base address property.

enumerator kFLASH_PropertyFlexRamTotalSize

FlexRam total size property.

typedef enum *flash_property_tag* flash_property_tag_t

Enumeration for various flash properties.

FSL_FLASH_DRIVER_VERSION

Flash driver version for SDK.

Version 2.3.3.

FLASH_ADDR_MASK

enum *_flash_driver_api_keys*

Enumeration for Flash driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFLASH_ApiEraseKey

Key value used to validate all flash erase APIs.

status_t FLASH_Init(*flash_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.

status_t FLASH_Erase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

status_t FLASH_EraseAll(FMU_Type *base, uint32_t key)

Erases entire flash and ifr.

status_t FLASH_Program(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash phrases with data at locations passed in through parameters.

status_t FLASH_ProgramPage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash pages with data at locations passed in through parameters.

status_t FLASH_VerifyErasePhrase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash phrases are erased.

status_t FLASH_VerifyErasePage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash pages are erased.

status_t FLASH_VerifyEraseSector(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash sectors are erased.

status_t FLASH_VerifyEraseAll(FMU_Type *base)

Verify that all flash and IFR space is erased.

status_t FLASH_VerifyEraseBlock(*flash_config_t* *config, FMU_Type *base, uint32_t blockaddr)

Verify that a flash block is erased.

status_t FLASH_VerifyEraseIFRPhrase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr phrases are erased.

status_t FLASH_VerifyEraseIFRPage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr pages are erased.

status_t FLASH_VerifyEraseIFRSector(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr sectors are erased.

status_t FLASH_GetProperty(*flash_config_t* *config, *flash_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flash property.

status_t Read_Into_MISR(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t ending, uint32_t *seed, uint32_t *signature)

Read into MISR.

The Read into MISR operation generates a signature based on the contents of the selected flash memory using an embedded MISR.

status_t Read_IFR_Into_MISR(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t ending, uint32_t *seed, uint32_t *signature)

Read IFR into MISR.

The Read IFR into MISR operation generates a signature based on the contents of the selected IFR space using an embedded MISR.

typedef struct *flash_mem_descriptor* flash_mem_desc_t

Flash memory descriptor.

typedef struct *flash_ifr_desc* flash_ifr_desc_t

typedef struct *msf1_config* msf1_config_t

```
typedef struct _flash_config flash_config_t
```

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

```
struct _flash_mem_descriptor
```

```
#include <fsl_k4_flash.h> Flash memory descriptor.
```

Public Members

```
uint32_t blockBase
```

Base address of the flash block

```
uint32_t totalSize
```

The size of the flash block.

```
uint32_t blockCount
```

A number of flash blocks.

```
struct _flash_ifr_desc
```

```
#include <fsl_k4_flash.h>
```

```
struct _msf1_config
```

```
#include <fsl_k4_flash.h>
```

```
struct _flash_config
```

```
#include <fsl_k4_flash.h> Flash driver state information.
```

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

2.17 FlexCAN: Flex Controller Area Network Driver

2.18 FlexCAN Driver

```
bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)
```

Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN_FDInit to enable CAN FD mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)

Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080. For 32 bytes payload frame, the second mailbox is $((1/12)*512 + 1\%12*40)/4 = 10$, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – Mailbox index.

Returns

Mailbox address offset in word.

status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

uint32_t FLEXCAN_GetInstance(CAN_Type *base)

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN instance.

bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t sourceClock_Hz, flexcan_timing_config_t *pTimingConfig)

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.

- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

void FLEXCAN_Init(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz)
Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the *flexcan_config_t* parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate    = 1000000U;
flexcanConfig.maxMbNum   = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.enableDoze = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.timingConfig = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 4000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t bitRateFD, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.
- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

void FLEXCAN_FDInit(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz, *flexcan_mb_size_t* dataSize, bool brs)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the `flexcan_config_t` parameters and how to call the `FLEXCAN_FDIInit` function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate    = 1000000U;
flexcanConfig.bitRateFD  = 2000000U;
flexcanConfig.maxMbNum   = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze = false;
flexcanConfig.timingConfig = timingConfig;
FLEXCAN_FDIInit(CAN0, &flexcanConfig, 8000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- `base` – FlexCAN peripheral base address.
- `pConfig` – Pointer to the user-defined configuration structure.
- `sourceClock_Hz` – FlexCAN Protocol Engine clock source frequency in Hz.
- `dataSize` – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- `brs` – True if bit rate switch is enabled in FD mode.

`void FLEXCAN_Deinit(CAN_Type *base)`

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- `base` – FlexCAN peripheral base address.

`void FLEXCAN_GetDefaultConfig(flexcan_config_t *pConfig)`

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. `flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enablePretendededeNetworking = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian; flexcanConfig.timingConfig = timingConfig;`

Parameters

- `pConfig` – Pointer to the FlexCAN configuration structure.

`void FLEXCAN_SetTimingConfig(CAN_Type *base, const flexcan_timing_config_t *pConfig)`

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the `FLEXCAN_SetBitRate()` instead.

Note: Calling `FLEXCAN_SetTimingConfig()` overrides the bit rate set in `FLEXCAN_Init()` or `FLEXCAN_SetBitRate()`.

Parameters

- `base` – FlexCAN peripheral base address.
- `pConfig` – Pointer to the timing configuration structure.

status_t `FLEXCAN_SetBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)`

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on `FLEXCAN_CalculateImprovedTimingValues()` API calculated timing values.

Note: Calling `FLEXCAN_SetBitRate()` overrides the bit rate set in `FLEXCAN_Init()`.

Parameters

- `base` – FlexCAN peripheral base address.
- `sourceClock_Hz` – Source Clock in Hz.
- `bitRate_Bps` – Bit rate in Bps.

Returns

`kStatus_Success` - Set CAN baud rate (only Nominal phase) successfully.

`void` `FLEXCAN_SetFDTimingConfig(CAN_Type *base, const flexcan_timing_config_t *pConfig)`

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the `FLEXCAN_SetFDBitRate()` to set both Nominal/Data bit Rate instead.

Note: Calling `FLEXCAN_SetFDTimingConfig()` overrides the data phase bit rate set in `FLEXCAN_FDInit()/FLEXCAN_SetFDBitRate()`.

Parameters

- `base` – FlexCAN peripheral base address.
- `pConfig` – Pointer to the timing configuration structure.

status_t `FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRateN_Bps, uint32_t bitRateD_Bps)`

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on `FLEXCAN_FDCalculateImprovedTimingValues()` API calculated timing values.

Parameters

- `base` – FlexCAN peripheral base address.
- `sourceClock_Hz` – Source Clock in Hz.
- `bitRateN_Bps` – Nominal bit Rate in Bps.

- bitRateD_Bps – Data bit Rate in Bps.

Returns

kStatus_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN_SetTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
 - true: Enable Tx Message Buffer.
 - false: Disable Tx Message Buffer.

```
void FLEXCAN_SetRxMbConfig(CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t
                          *pRxMbConfig, bool enable)
```

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbEmpty, kFLEXCAN_RxMbFull or kFLEXCAN_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
 - true: Enable Rx Message Buffer.
 - false: Disable Rx Message Buffer.

```
static inline void FLEXCAN_SetMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)
```

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

```
void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)
```

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
 - true: Enable Tx Message Buffer.
 - false: Disable Tx Message Buffer.

```
void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const
                             flexcan_rx_mb_config_t *pRxMbConfig, bool enable)
```

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.

- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
 - true: Enable Rx Message Buffer.
 - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)
Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const
flexcan_frame_t *pFrame)

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const *flexcan_rx_fifo_config_t* *pRxFifoConfig,
bool enable)

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note: Legacy Rx FIFO only can receive classic CAN message.

Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Legacy Rx FIFO.
 - true: Enable Legacy Rx FIFO.
 - false: Disable Legacy Rx FIFO.

void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const
flexcan_enhanced_rx_fifo_config_t *pConfig, bool
enable)

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

Note: Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
 - true: Enable Enhanced Rx FIFO.
 - false: Disable Enhanced Rx FIFO.

```
void FLEXCAN_SetPNConfig(CAN_Type *base, const flexcan_pn_config_t *pConfig)
```

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

```
static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)
```

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators `_flexcan_flags`. To check the specific status, compare the return value with enumerators in `_flexcan_flags`.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the `_flexcan_flags`.

```
static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of `_flexcan_flags`.

```
static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)
```

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base,
flexcan_memory_error_report_status_t
*errorStatus)

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of received wake up messages.

```
static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)
```

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

```
static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)
```

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of `_flexcan_interrupt_enable`.

```
static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)
```

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to disable. Logical OR of `_flexcan_interrupt_enable`.

```
static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)
```

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)
```

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)
```

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.

- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- enable – true to enable, false to disable.

static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN Rx FIFO Head address.

static inline *status_t* FLEXCAN_Enable(CAN_Type *base, bool enable)

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

Parameters

- base – FlexCAN base pointer.
- enable – true to enable, false to disable.

Returns

kStatus_Success Enable FlexCAN module successful
kStatus_Timeout Timeout when wait for Low-Power Mode Acknowledge

status_t FLEXCAN_WriteTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pTxFrame)

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success -- Write Tx Message Buffer Successfully.

- `kStatus_Fail` -- Tx Message Buffer is currently in use.

`status_t` FLEXCAN_ReadRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – The FlexCAN Message Buffer index.
- `pRxFrame` – Pointer to CAN message frame structure for reception.

Return values

- `kStatus_Success` -- Rx Message Buffer is full and has been read successfully.
- `kStatus_FLEXCAN_RxOverflow` -- Rx Message Buffer is already overflowed and has been read successfully.
- `kStatus_Fail` -- Rx Message Buffer is empty.

`status_t` FLEXCAN_WriteFDTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_fd_frame_t* *pTxFrame)

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – The FlexCAN FD Message Buffer index.
- `pTxFrame` – Pointer to CAN FD message frame to be sent.

Return values

- `kStatus_Success` -- Write Tx Message Buffer Successfully.
- `kStatus_Fail` -- Tx Message Buffer is currently in use.

`status_t` FLEXCAN_ReadFDRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – The FlexCAN FD Message Buffer index.
- `pRxFrame` – Pointer to CAN FD message frame structure for reception.

Return values

- `kStatus_Success` -- Rx Message Buffer is full and has been read successfully.
- `kStatus_FLEXCAN_RxOverflow` -- Rx Message Buffer is already overflowed and has been read successfully.
- `kStatus_Fail` -- Rx Message Buffer is empty.

status_t FLEXCAN_ReadRxFifo(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – Read Message from Rx FIFO successfully.
- kStatus_Fail – Rx FIFO is not enabled.

status_t FLEXCAN_ReadEnhancedRxFifo(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – Read Message from Rx FIFO successfully.
- kStatus_Fail – Rx FIFO is not enabled.

status_t FLEXCAN_ReadPNWakeUpMB(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.
- mbIdx – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

Return values

- kStatus_Success – Read Message from Wake up Message Buffer successfully.
- kStatus_Fail – Wake up Message Buffer has no valid content.

status_t FLEXCAN_TransferFDSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – Write Tx Message Buffer Successfully.
- kStatus_Fail – Tx Message Buffer is currently in use.
- kStatus_Timeout – Failed to send frames within specific time.

status_t FLEXCAN_TransferFDReceiveBlocking(CAN_Type *base, uint8_t mbIdx, flexcan_fd_frame_t *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – Rx Message Buffer is empty.
- kStatus_Timeout – Failed to receive frames within specific time.

status_t FLEXCAN_TransferFDSendNonBlocking(CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferFDReceiveNonBlocking(CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.

- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

void FLEXCAN_TransferFDAbortSend(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

void FLEXCAN_TransferFDAbortReceive(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

status_t FLEXCAN_TransferSendBlocking(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success -- Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow -- Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail -- Rx Message Buffer is empty.
- kStatus_Timeout -- Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveFifoBlocking(CAN_Type *base, *flexcan_frame_t* *pRxFrame)
Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success -- Read Message from Rx FIFO successfully.
- kStatus_Fail -- Rx FIFO is not enabled.
- kStatus_Timeout -- Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveEnhancedFifoBlocking(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success -- Read Message from Rx FIFO successfully.
- kStatus_Fail -- Rx FIFO is not enabled.
- kStatus_Timeout -- Failed to receive frames within specific time.

void FLEXCAN_TransferCreateHandle(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_transfer_callback_t* callback, void *userData)

Initializes the FlexCAN handle.

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t FLEXCAN__TransferSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN__TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN__TransferReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success -- Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN__RxBusy -- Rx Message Buffer is in use.

status_t FLEXCAN__TransferReceiveFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the *flexcan_fifo_transfer_t*.

Return values

- kStatus_Success -- Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy -- Rx FIFO is currently in use.

status_t FLEXCAN_TransferGetReceiveFifoCount(CAN_Type *base, flexcan_handle_t *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

status_t FLEXCAN_TransferReceiveEnhancedFifoNonBlocking(CAN_Type *base, flexcan_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the ref flexcan_fifo_transfer_t.@

Return values

- kStatus_Success -- Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy -- Rx FIFO is currently in use.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCount(CAN_Type *base, flexcan_handle_t *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

uint32_t FLEXCAN_GetTimeStamp(flexcan_handle_t *handle, uint8_t mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback

function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

```
void FLEXCAN_TransferAbortSend(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
```

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

```
void FLEXCAN_TransferAbortReceive(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)
```

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

```
void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, flexcan_handle_t *handle)
```

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_TransferAbortReceiveEnhancedFifo(CAN_Type *base, flexcan_handle_t *handle)
```

Aborts the interrupt driven message receive from Enhanced Rx FIFO process.

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_TransferHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)
```

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – FlexCAN peripheral base address.

- handle – FlexCAN handle pointer.

void FLEXCAN_MbHandleIRQ(CAN_Type *base, flexcan_handle_t *handle, uint32_t startMbIdx, uint32_t endMbIdx)

FlexCAN Message Buffer IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

void FLEXCAN_EhancedRxFifoHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)

FlexCAN Enhanced Rx FIFO IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_BusoffErrorHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)

FlexCAN Bus Off, Error and Warning IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_PNWakeUpHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)

FlexCAN Pretended Networking Wake-up IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MemoryErrorHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)

FlexCAN Memory Error IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

FSL_FLEXCAN_DRIVER_VERSION

FlexCAN driver version.

FlexCAN transfer status.

Values:

enumerator kStatus_FLEXCAN_TxBusy

Tx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_TxIdle

Tx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_TxSwitchToRx

Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus_FLEXCAN_RxBusy
Rx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_RxIdle
Rx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_RxOverflow
Rx Message Buffer is Overflowed.

enumerator kStatus_FLEXCAN_RxFifoBusy
Rx Message FIFO is Busy.

enumerator kStatus_FLEXCAN_RxFifoIdle
Rx Message FIFO is Idle.

enumerator kStatus_FLEXCAN_RxFifoOverflow
Rx Message FIFO is overflowed.

enumerator kStatus_FLEXCAN_RxFifoWarning
Rx Message FIFO is almost overflowed.

enumerator kStatus_FLEXCAN_RxFifoDisabled
Rx Message FIFO is disabled during reading.

enumerator kStatus_FLEXCAN_ErrorStatus
FlexCAN Module Error and Status.

enumerator kStatus_FLEXCAN_WakeUp
FlexCAN is waken up from STOP mode.

enumerator kStatus_FLEXCAN_UnHandled
UnHandled Interrupt asserted.

enumerator kStatus_FLEXCAN_RxRemote
Rx Remote Message Received in Mail box.

enumerator kStatus_FLEXCAN_RxFifoUnderflow
Enhanced Rx Message FIFO is underflow.

enumerator kStatus_FLEXCAN_MemoryError
FlexCAN Memory Error.

enum _flexcan_frame_format
FlexCAN frame format.

Values:

enumerator kFLEXCAN_FrameFormatStandard
Standard frame format attribute.

enumerator kFLEXCAN_FrameFormatExtend
Extend frame format attribute.

enum _flexcan_frame_type
FlexCAN frame type.

Values:

enumerator kFLEXCAN_FrameTypeData
Data frame type attribute.

enumerator kFLEXCAN_FrameTypeRemote
Remote frame type attribute.

enum `_flexcan_clock_source`
FlexCAN clock source.

Deprecated:

Do not use the `kFLEXCAN_ClkSrcOs`. It has been superceded `kFLEXCAN_ClkSrc0`

Do not use the `kFLEXCAN_ClkSrcPeri`. It has been superceded `kFLEXCAN_ClkSrc1`

Values:

enumerator `kFLEXCAN_ClkSrcOsc`
FlexCAN Protocol Engine clock from Oscillator.

enumerator `kFLEXCAN_ClkSrcPeri`
FlexCAN Protocol Engine clock from Peripheral Clock.

enumerator `kFLEXCAN_ClkSrc0`
FlexCAN Protocol Engine clock selected by user as SRC == 0.

enumerator `kFLEXCAN_ClkSrc1`
FlexCAN Protocol Engine clock selected by user as SRC == 1.

enum `_flexcan_wake_up_source`
FlexCAN wake up source.

Values:

enumerator `kFLEXCAN_WakeupSrcUnfiltered`
FlexCAN uses unfiltered Rx input to detect edge.

enumerator `kFLEXCAN_WakeupSrcFiltered`
FlexCAN uses filtered Rx input to detect edge.

enum `_flexcan_endianness`
FlexCAN payload endianness.

Values:

enumerator `kFLEXCAN_bigEndian`
Transmit frame with MSB first, receive frame with big-endian format.

enumerator `kFLEXCAN_littleEndian`
Transmit frame with LSB first, receive frame with little-endian format.

enum `_flexcan_rx_fifo_filter_type`
FlexCAN Rx Fifo Filter type.

Values:

enumerator `kFLEXCAN_RxFifoFilterTypeA`
One full ID (standard and extended) per ID Filter element.

enumerator `kFLEXCAN_RxFifoFilterTypeB`
Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

enumerator `kFLEXCAN_RxFifoFilterTypeC`
Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

enumerator `kFLEXCAN_RxFifoFilterTypeD`
All frames rejected.

enum `_flexcan_mb_size`

FlexCAN Message Buffer Payload size.

Values:

enumerator `kFLEXCAN_8BperMB`

Selects 8 bytes per Message Buffer.

enumerator `kFLEXCAN_16BperMB`

Selects 16 bytes per Message Buffer.

enumerator `kFLEXCAN_32BperMB`

Selects 32 bytes per Message Buffer.

enumerator `kFLEXCAN_64BperMB`

Selects 64 bytes per Message Buffer.

enum `_flexcan_fd_frame_length`

FlexCAN CAN FD frame supporting data length (available DLC values).

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

Values:

enumerator `kFLEXCAN_0BperFrame`

Frame contains 0 valid data bytes.

enumerator `kFLEXCAN_1BperFrame`

Frame contains 1 valid data bytes.

enumerator `kFLEXCAN_2BperFrame`

Frame contains 2 valid data bytes.

enumerator `kFLEXCAN_3BperFrame`

Frame contains 3 valid data bytes.

enumerator `kFLEXCAN_4BperFrame`

Frame contains 4 valid data bytes.

enumerator `kFLEXCAN_5BperFrame`

Frame contains 5 valid data bytes.

enumerator `kFLEXCAN_6BperFrame`

Frame contains 6 valid data bytes.

enumerator `kFLEXCAN_7BperFrame`

Frame contains 7 valid data bytes.

enumerator `kFLEXCAN_8BperFrame`

Frame contains 8 valid data bytes.

enumerator `kFLEXCAN_12BperFrame`

Frame contains 12 valid data bytes.

enumerator `kFLEXCAN_16BperFrame`

Frame contains 16 valid data bytes.

enumerator `kFLEXCAN_20BperFrame`

Frame contains 20 valid data bytes.

enumerator kFLEXCAN_24BperFrame
Frame contains 24 valid data bytes.

enumerator kFLEXCAN_32BperFrame
Frame contains 32 valid data bytes.

enumerator kFLEXCAN_48BperFrame
Frame contains 48 valid data bytes.

enumerator kFLEXCAN_64BperFrame
Frame contains 64 valid data bytes.

enum _flexcan_efifo_dma_per_read_length
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

Values:

enumerator kFLEXCAN_1WordPerRead
Transfer 1 32-bit words (CS).

enumerator kFLEXCAN_2WordPerRead
Transfer 2 32-bit words (CS + ID).

enumerator kFLEXCAN_3WordPerRead
Transfer 3 32-bit words (CS + ID + 1~4 bytes data).

enumerator kFLEXCAN_4WordPerRead
Transfer 4 32-bit words (CS + ID + 5~8 bytes data).

enumerator kFLEXCAN_5WordPerRead
Transfer 5 32-bit words (CS + ID + 9~12 bytes data).

enumerator kFLEXCAN_6WordPerRead
Transfer 6 32-bit words (CS + ID + 13~16 bytes data).

enumerator kFLEXCAN_7WordPerRead
Transfer 7 32-bit words (CS + ID + 17~20 bytes data).

enumerator kFLEXCAN_8WordPerRead
Transfer 8 32-bit words (CS + ID + 21~24 bytes data).

enumerator kFLEXCAN_9WordPerRead
Transfer 9 32-bit words (CS + ID + 25~28 bytes data).

enumerator kFLEXCAN_10WordPerRead
Transfer 10 32-bit words (CS + ID + 29~32 bytes data).

enumerator kFLEXCAN_11WordPerRead
Transfer 11 32-bit words (CS + ID + 33~36 bytes data).

enumerator kFLEXCAN_12WordPerRead
Transfer 12 32-bit words (CS + ID + 37~40 bytes data).

enumerator kFLEXCAN_13WordPerRead
Transfer 13 32-bit words (CS + ID + 41~44 bytes data).

enumerator kFLEXCAN_14WordPerRead
Transfer 14 32-bit words (CS + ID + 45~48 bytes data).

enumerator kFLEXCAN_15WordPerRead
Transfer 15 32-bit words (CS + ID + 49~52 bytes data).

enumerator kFLEXCAN_16WordPerRead
Transfer 16 32-bit words (CS + ID + 53~56 bytes data).

enumerator kFLEXCAN_17WordPerRead
Transfer 17 32-bit words (CS + ID + 57~60 bytes data).

enumerator kFLEXCAN_18WordPerRead
Transfer 18 32-bit words (CS + ID + 61~64 bytes data).

enumerator kFLEXCAN_19WordPerRead
Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT).

enum _flexcan_rx_fifo_priority
FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

Values:

enumerator kFLEXCAN_RxFifoPrioLow
Matching process start from Rx Message Buffer first.

enumerator kFLEXCAN_RxFifoPrioHigh
Matching process start from Enhanced/Legacy Rx FIFO first.

enum _flexcan_interrupt_enable
FlexCAN interrupt enable enumerations.

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

Note: FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Values:

enumerator kFLEXCAN_BusOffInterruptEnable
Bus Off interrupt, use bit 15.

enumerator kFLEXCAN_ErrorInterruptEnable
CAN Error interrupt, use bit 14.

enumerator kFLEXCAN_TxWarningInterruptEnable
Tx Warning interrupt, use bit 11.

enumerator kFLEXCAN_RxWarningInterruptEnable
Rx Warning interrupt, use bit 10.

enumerator kFLEXCAN_WakeUpInterruptEnable
Self Wake Up interrupt, use bit 26.

enumerator kFLEXCAN_FDErrorInterruptEnable
CAN FD Error interrupt, use bit 31.

enumerator kFLEXCAN_PNMatchWakeUpInterruptEnable
PN Match Wake Up interrupt, use high word bit 17.

enumerator kFLEXCAN_PNTimeoutWakeUpInterruptEnable
PN Timeout Wake Up interrupt, use high word bit 16. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

enumerator kFLEXCAN_ERxFifoUnderflowInterruptEnable

Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

enumerator kFLEXCAN_ERxFifoOverflowInterruptEnable

Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

enumerator kFLEXCAN_ERxFifoWatermarkInterruptEnable

Enhanced Rx FIFO Data Available interrupt, use high word bit 28.

enumerator kFLEXCAN_ERxFifoDataAvlInterruptEnable

enumerator kFLEXCAN_HostAccessNCErrInterruptEnable

Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN_FlexCanAccessNCErrInterruptEnable

FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN_HostOrFlexCanCErrInterruptEnable

Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum _flexcan_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note: The CPU read action clears the bits corresponding to the FLEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using _flexcan_error_flags enumerations.

Values:

enumerator kFLEXCAN_ErrorOverrunFlag

Error Overrun Status.

enumerator kFLEXCAN_FDErrorIntFlag

CAN FD Error Interrupt Flag.

enumerator kFLEXCAN_BusoffDoneIntFlag

Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN_SynchFlag

CAN Synchronization Status.

enumerator kFLEXCAN_TxWarningIntFlag

Tx Warning Interrupt Flag.

enumerator kFLEXCAN_RxWarningIntFlag

Rx Warning Interrupt Flag.

enumerator kFLEXCAN_IdleFlag

FlexCAN In IDLE Status.

enumerator kFLEXCAN_FaultConfinementFlag

FlexCAN Fault Confinement State.

enumerator kFLEXCAN_TransmittingFlag

FlexCAN In Transmission Status.

enumerator kFLEXCAN_ReceivingFlag

FlexCAN In Reception Status.

enumerator kFLEXCAN_BusOffIntFlag
Bus Off Interrupt Flag.

enumerator kFLEXCAN_ErrorIntFlag
CAN Error Interrupt Flag.

enumerator kFLEXCAN_WakeUpIntFlag
Self Wake-Up Interrupt Flag.

enumerator kFLEXCAN_ErrorFlag

enumerator kFLEXCAN_PNMatchIntFlag
PN Matching Event Interrupt Flag.

enumerator kFLEXCAN_PNTimeoutIntFlag
PN Timeout Event Interrupt Flag.

enumerator kFLEXCAN_ERxFifoUnderflowIntFlag
Enhanced Rx FIFO underflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoOverflowIntFlag
Enhanced Rx FIFO overflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoWatermarkIntFlag
Enhanced Rx FIFO watermark Interrupt Flag.

enumerator kFLEXCAN_ERxFifoDataAvlIntFlag
Enhanced Rx FIFO data available Interrupt Flag.

enumerator kFLEXCAN_ERxFifoEmptyFlag
Enhanced Rx FIFO empty status.

enumerator kFLEXCAN_ERxFifoFullFlag
Enhanced Rx FIFO full status.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorIntFlag
Host Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag
FlexCAN Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_CorrectableErrorIntFlag
Correctable Error Interrupt Flag.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag
Host Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag
FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_CorrectableErrorOverrunFlag
Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_AllMemoryErrorIntFlag
All Memory Error Interrupt Flags.

enumerator kFLEXCAN_AllMemoryErrorFlag
All Memory Error Flags.

enum `_flexcan_error_flags`

FlexCAN error status flags.

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with `KFLEXCAN_ErrorFlag` in `_flexcan_flags` enumerations to determine which error is generated.

Values:

enumerator `kFLEXCAN_FDStuffingError`

Stuffing Error.

enumerator `kFLEXCAN_FDFormError`

Form Error.

enumerator `kFLEXCAN_FDCrcError`

Cyclic Redundancy Check Error.

enumerator `kFLEXCAN_FDBit0Error`

Unable to send dominant bit.

enumerator `kFLEXCAN_FDBit1Error`

Unable to send recessive bit.

enumerator `kFLEXCAN_TxErrorWarningFlag`

Tx Error Warning Status.

enumerator `kFLEXCAN_RxErrorWarningFlag`

Rx Error Warning Status.

enumerator `kFLEXCAN_StuffingError`

Stuffing Error.

enumerator `kFLEXCAN_FormError`

Form Error.

enumerator `kFLEXCAN_CrcError`

Cyclic Redundancy Check Error.

enumerator `kFLEXCAN_AckError`

Received no ACK on transmission.

enumerator `kFLEXCAN_Bit0Error`

Unable to send dominant bit.

enumerator `kFLEXCAN_Bit1Error`

Unable to send recessive bit.

FlexCAN Legacy Rx FIFO status flags.

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupiees more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Values:

enumerator `kFLEXCAN_RxFifoOverflowFlag`

Rx FIFO overflow flag.

enumerator `kFLEXCAN_RxFifoWarningFlag`

Rx FIFO almost full flag.

enumerator kFLEXCAN_RxFifoFrameAvlFlag

Frames available in Rx FIFO flag.

enum _flexcan_memory_error_type

FlexCAN Memory Error Type.

Values:

enumerator kFLEXCAN_CorrectableError

The memory error is correctable which means on bit error.

enumerator kFLEXCAN_NonCorrectableError

The memory error is non-correctable which means two bit errors.

enum _flexcan_memory_access_type

FlexCAN Memory Access Type.

Values:

enumerator kFLEXCAN_MoveOutFlexCanAccess

The memory error was detected during move-out FlexCAN access.

enumerator kFLEXCAN_MoveInAccess

The memory error was detected during move-in FlexCAN access.

enumerator kFLEXCAN_TxArbitrationAccess

The memory error was detected during Tx Arbitration FlexCAN access.

enumerator kFLEXCAN_RxMatchingAccess

The memory error was detected during Rx Matching FlexCAN access.

enumerator kFLEXCAN_MoveOutHostAccess

The memory error was detected during Rx Matching Host (CPU) access.

enum _flexcan_byte_error_syndrome

FlexCAN Memory Error Byte Syndrome.

Values:

enumerator kFLEXCAN_NoError

No bit error in this byte.

enumerator kFLEXCAN_ParityBits0Error

Parity bit 0 error in this byte.

enumerator kFLEXCAN_ParityBits1Error

Parity bit 1 error in this byte.

enumerator kFLEXCAN_ParityBits2Error

Parity bit 2 error in this byte.

enumerator kFLEXCAN_ParityBits3Error

Parity bit 3 error in this byte.

enumerator kFLEXCAN_ParityBits4Error

Parity bit 4 error in this byte.

enumerator kFLEXCAN_DataBits0Error

Data bit 0 error in this byte.

enumerator kFLEXCAN_DataBits1Error

Data bit 1 error in this byte.

enumerator kFLEXCAN_DataBits2Error

Data bit 2 error in this byte.

enumerator kFLEXCAN_DataBits3Error

Data bit 3 error in this byte.

enumerator kFLEXCAN_DataBits4Error

Data bit 4 error in this byte.

enumerator kFLEXCAN_DataBits5Error

Data bit 5 error in this byte.

enumerator kFLEXCAN_DataBits6Error

Data bit 6 error in this byte.

enumerator kFLEXCAN_DataBits7Error

Data bit 7 error in this byte.

enumerator kFLEXCAN_AllZeroError

All-zeros non-correctable error in this byte.

enumerator kFLEXCAN_AllOneError

All-ones non-correctable error in this byte.

enumerator kFLEXCAN_NonCorrectableErrors

Non-correctable error in this byte.

enum _flexcan_pn_match_source

FlexCAN Pretended Networking match source selection.

Values:

enumerator kFLEXCAN_PNMatSrcID

Message match with ID filtering.

enumerator kFLEXCAN_PNMatSrcIDAndData

Message match with ID filtering and payload filtering.

enum _flexcan_pn_match_mode

FlexCAN Pretended Networking mode match type.

Values:

enumerator kFLEXCAN_PNMatModeEqual

Match upon ID/Payload contents against an exact target value.

enumerator kFLEXCAN_PNMatModeGreater

Match upon an ID/Payload value greater than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeSmaller

Match upon an ID/Payload value smaller than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeRange

Match upon an ID/Payload value inside a range, greater than or equal to a specified lower limit, and smaller than or equal to a specified upper limit

typedef enum _flexcan_frame_format flexcan_frame_format_t

FlexCAN frame format.

typedef enum _flexcan_frame_type flexcan_frame_type_t

FlexCAN frame type.

typedef enum *_flexcan_clock_source* flexcan_clock_source_t
FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

typedef enum *_flexcan_wake_up_source* flexcan_wake_up_source_t
FlexCAN wake up source.

typedef enum *_flexcan_endianness* flexcan_endianness_t
FlexCAN payload endianness.

typedef enum *_flexcan_rx_fifo_filter_type* flexcan_rx_fifo_filter_type_t
FlexCAN Rx Fifo Filter type.

typedef enum *_flexcan_mb_size* flexcan_mb_size_t
FlexCAN Message Buffer Payload size.

typedef enum *_flexcan_efifo_dma_per_read_length* flexcan_efifo_dma_per_read_length_t
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

typedef enum *_flexcan_rx_fifo_priority* flexcan_rx_fifo_priority_t
FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

typedef enum *_flexcan_memory_error_type* flexcan_memory_error_type_t
FlexCAN Memory Error Type.

typedef enum *_flexcan_memory_access_type* flexcan_memory_access_type_t
FlexCAN Memory Access Type.

typedef enum *_flexcan_byte_error_syndrome* flexcan_byte_error_syndrome_t
FlexCAN Memory Error Byte Syndrome.

typedef struct *_flexcan_memory_error_report_status* flexcan_memory_error_report_status_t
FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN_GetMemoryErrorReportStatus() function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

typedef struct *_flexcan_frame* flexcan_frame_t
FlexCAN message frame structure.

typedef struct *_flexcan_fd_frame* flexcan_fd_frame_t
CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see *_flexcan_fd_frame_length*.

typedef struct *_flexcan_timing_config* flexcan_timing_config_t
FlexCAN protocol timing characteristic configuration structure.

typedef struct *flexcan_config* flexcan_config_t
FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

typedef struct *flexcan_rx_mb_config* flexcan_rx_mb_config_t
FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

typedef enum *flexcan_pn_match_source* flexcan_pn_match_source_t
FlexCAN Pretended Networking match source selection.

typedef enum *flexcan_pn_match_mode* flexcan_pn_match_mode_t
FlexCAN Pretended Networking mode match type.

typedef struct *flexcan_pn_config* flexcan_pn_config_t
FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

typedef struct *flexcan_rx_fifo_config* flexcan_rx_fifo_config_t
FlexCAN Legacy Rx FIFO configuration structure.

typedef struct *flexcan_enhanced_rx_fifo_std_id_filter* flexcan_enhanced_rx_fifo_std_id_filter_t
FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

typedef struct *flexcan_enhanced_rx_fifo_ext_id_filter* flexcan_enhanced_rx_fifo_ext_id_filter_t
FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

typedef struct *flexcan_enhanced_rx_fifo_config* flexcan_enhanced_rx_fifo_config_t
FlexCAN Enhanced Rx FIFO configuration structure.

typedef struct *flexcan_mb_transfer* flexcan_mb_transfer_t
FlexCAN Message Buffer transfer.

typedef struct *flexcan_fifo_transfer* flexcan_fifo_transfer_t
FlexCAN Rx FIFO transfer.

typedef struct *flexcan_handle* flexcan_handle_t
FlexCAN handle structure definition.

typedef void (*flexcan_transfer_callback_t)(CAN_Type *base, flexcan_handle_t *handle, status_t status, uint64_t result, void *userData)

FLEXCAN_WAIT_TIMEOUT

FLEXCAN_POLLING_TIMEOUT

Max loops to wait for polling transfer.

FLEXCAN_MODULE_TIMEOUT

Max loops to wait for FlexCAN register access complete.

DLC_LENGTH_DECODE(dlc)

FlexCAN frame length helper macro.

FLEXCAN_ID_STD(id)

FlexCAN Frame ID helper macro.

Standard Frame ID helper macro.

FLEXCAN_ID_EXT(id)

Extend Frame ID helper macro.

FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)

FlexCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)

Extend Rx Message Buffer Mask helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)

FlexCAN Legacy Rx FIFO Mask helper macro.

Standard Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id)

Standard Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id)

Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id)

Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)

Standard Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)

Extend Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)

Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)

Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)

Extend Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)

FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)
Standard Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)
Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)
Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)
Standard Rx FIFO Filter helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)
Extend Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)
Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)
Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id)
Extend Rx FIFO Filter helper macro Type C lower part helper macro.

ENHANCED_RX_FIFO_FSCH(x)
FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.

RTR_STD_HIGH(x)

RTR_STD_LOW(x)

RTR_EXT(x)

ID_STD_LOW(id)

ID_STD_HIGH(id)

ID_EXT(id)

FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask)
Standard ID filter element with filter + mask scheme.

FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE(id_upper, rtr, id_lower, rtr_mask)
Standard ID filter element with filter range.

FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS(id1, rtr1, id2, rtr2)
Standard ID filter element with two filters without masks.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW(id, rtr)

Extended ID filter element with filter + mask scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH(id_mask, rtr_mask)

Extended ID filter element with filter + mask scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW(id_upper, rtr)

Extended ID filter element with range scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH(id_lower, rtr_mask)

Extended ID filter element with range scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW(id2, rtr2)

Extended ID filter element with two filters without masks low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH(id1, rtr1)

Extended ID filter element with two filters without masks high word.

FLEXCAN_PN_STD_MASK(id, rtr)

FlexCAN Pretended Networking ID Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_PN_EXT_MASK(id, rtr)

Extend Rx Message Buffer Mask helper macro.

FLEXCAN_PN_INT_MASK(x)

FlexCAN interrupt/status flag helper macro.

FLEXCAN_PN_INT_UNMASK(x)

FLEXCAN_PN_STATUS_MASK(x)

FLEXCAN_PN_STATUS_UNMASK(x)

FLEXCAN_EFIFO_INT_MASK(x)

FLEXCAN_EFIFO_INT_UNMASK(x)

FLEXCAN_EFIFO_STATUS_MASK(x)

FLEXCAN_EFIFO_STATUS_UNMASK(x)

FLEXCAN_MECR_INT_MASK(x)

FLEXCAN_MECR_INT_UNMASK(x)

FLEXCAN_MECR_STATUS_MASK(x)

FLEXCAN_MECR_STATUS_UNMASK(x)

FLEXCAN_ERROR_AND_STATUS_INT_FLAG

FLEXCAN_PNWAKE_UP_FLAG

FLEXCAN_WAKE_UP_FLAG

FLEXCAN_MEMORY_ERROR_INT_FLAG

FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG

FlexCAN Enhanced Rx FIFO base address helper macro.

E_RX_FIFO(base)

FLEXCAN_CALLBACK(x)

FlexCAN transfer callback function.

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_FLEXCAN_ErrorStatus`, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

struct `_flexcan_memory_error_report_status`

#include <fsl_flexcan.h> FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of `FLEXCAN_GetMemoryErrorReportStatus()` function. And user can use `FLEXCAN_GetMemoryErrorReportStatus` to get the status of the last memory error access.

Public Members

flexcan_memory_error_type_t errorType

The type of memory error that giving rise to the report.

flexcan_memory_access_type_t accessType

The type of memory access that giving rise to the memory error.

`uint16_t` accessAddress

The address where memory error detected.

`uint32_t` errorData

The raw data word read from memory with error.

struct `_flexcan_frame`

#include <fsl_flexcan.h> FlexCAN message frame structure.

struct `_flexcan_fd_frame`

#include <fsl_flexcan.h> CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see `_flexcan_fd_frame_length`.

Public Members

`uint32_t` idhit

Note: ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using `FLEXCAN_FixEnhancedRx FifoFrameIdHit` API is recommended to ensure this idhit value is correct. CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).

struct `_flexcan_timing_config`

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

`uint32_t preDivider`
Classic CAN or CAN FD nominal phase bit rate prescaler.

`uint32_t rJumpwidth`
Classic CAN or CAN FD nominal phase Re-sync Jump Width.

`uint32_t phaseSeg1`
Classic CAN or CAN FD nominal phase Segment 1.

`uint32_t phaseSeg2`
Classic CAN or CAN FD nominal phase Segment 2.

`uint32_t propSeg`
Classic CAN or CAN FD nominal phase Propagation Segment.

`uint32_t fpreDivider`
CAN FD data phase bit rate prescaler.

`uint32_t frJumpwidth`
CAN FD data phase Re-sync Jump Width.

`uint32_t fphaseSeg1`
CAN FD data phase Phase Segment 1.

`uint32_t fphaseSeg2`
CAN FD data phase Phase Segment 2.

`uint32_t fpropSeg`
CAN FD data phase Propagation Segment.

`struct _flexcan_config`
#include <fsl_flexcan.h> FlexCAN module configuration structure.

Deprecated:

Do not use the `baudRate`. It has been superceded `bitRate`
Do not use the `baudRateFD`. It has been superceded `bitRateFD`

Public Members

`flexcan_clock_source_t clkSrc`
Clock source for FlexCAN Protocol Engine.

`flexcan_wake_up_source_t wakeupSrc`
Wake up source selection.

`uint8_t maxMbNum`
The maximum number of Message Buffers used by user.

`bool enableLoopBack`
Enable or Disable Loop Back Self Test Mode.

`bool enableTimerSync`
Enable or Disable Timer Synchronization.

`bool enableSelfWakeup`
Enable or Disable Self Wakeup Mode.

`bool enableIndividMask`
Enable or Disable Rx Individual Mask and Queue feature.

- `bool disableSelfReception`
Enable or Disable Self Reflection.
- `bool enableListenOnlyMode`
Enable or Disable Listen Only Mode.
- `bool enableDoze`
Enable or Disable Doze Mode.
- `bool enablePretendedNetworking`
Enable or Disable the Pretended Networking mode.
- `bool enableMemoryErrorControl`
Enable or Disable the memory errors detection and correction mechanism.
- `bool enableNonCorrectableErrorEnterFreeze`
Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.
- `bool enableTransceiverDelayMeasure`
Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.
- `bool enableRemoteRequestFrameStored`
true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response Frame.
- `flexcan_endianness_t payloadEndianness`
Selects the byte order for the payload of transmit and receive frames, see `flexcan_endianness_t`.

`struct _flexcan_rx_mb_config`

`#include <fsl_flexcan.h>` FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of `FLEXCAN_SetRxMbConfig()` function. The `FLEXCAN_SetRxMbConfig()` function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

Public Members

`uint32_t id`

CAN Message Buffer Frame Identifier, should be set using `FLEXCAN_ID_EXT()` or `FLEXCAN_ID_STD()` macro.

`flexcan_frame_format_t format`

CAN Frame Identifier format(Standard of Extend).

`flexcan_frame_type_t type`

CAN Frame Type(Data or Remote for classical CAN only).

`struct _flexcan_pn_config`

`#include <fsl_flexcan.h>` FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of `FLEXCAN_SetPNConfig()` function. The `FLEXCAN_SetPNConfig()` function is used to configure FlexCAN Networking work mode.

Public Members

`bool enableTimeout`

Enable or Disable timeout event trigger wakeup.

`uint16_t timeoutValue`

The timeout value that generates a wakeup event, the counter timer is incremented based on 64 times the CAN Bit Time unit.

`bool enableMatch`

Enable or Disable match event trigger wakeup.

`flexcan_pn_match_source_t matchSrc`

Selects the match source (ID and/or data match) to trigger wakeup.

`uint8_t matchNum`

The number of times a given message must match the predefined ID and/or data before generating a wakeup event, range in 0x1 ~ 0xFF.

`flexcan_pn_match_mode_t idMatchMode`

The ID match type.

`flexcan_pn_match_mode_t dataMatchMode`

The data match type.

`uint32_t idLower`

The ID target values 1 which used either for ID match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in ID match “range detection”.

`uint32_t idUpper`

The ID target values 2 which used only as the upper limit value in ID match “range detection” or used to store the ID mask in “equal to”.

`uint8_t lengthLower`

The lower limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

`uint8_t lengthUpper`

The upper limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

`struct _flexcan_rx_fifo_config`

`#include <fsl_flexcan.h>` FlexCAN Legacy Rx FIFO configuration structure.

Public Members

`uint32_t *idFilterTable`

Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

`uint8_t idFilterNum`

The FlexCAN Legacy Rx FIFO Filter elements quantity.

`flexcan_rx_fifo_filter_type_t idFilterType`

The FlexCAN Legacy Rx FIFO Filter type.

`flexcan_rx_fifo_priority_t priority`

The FlexCAN Legacy Rx FIFO receive priority.

`struct _flexcan_enhanced_rx_fifo_std_id_filter`

`#include <fsl_flexcan.h>` FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length ≤ 8 , it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct `_flexcan_enhanced_rx_fifo_ext_id_filter`

`#include <fsl_flexcan.h>` FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length ≤ 8 , it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct `_flexcan_enhanced_rx_fifo_config`

`#include <fsl_flexcan.h>` FlexCAN Enhanced Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to `idFilterNum`. There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupie 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

uint8_t idFilterPairNum

`idFilterPairNum` is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

`uint8_t` `extendIdFilterNum`

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, `extendIdFilterNum` need less than or equal to `idFilterPairNum`.

`uint8_t` `fifoWatermark`

(`fifoWatermark + 1`) is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

`flexcan_efifo_dma_per_read_length_t` `dmaPerReadLength`

Define the length of each read of the Enhanced RX FIFO element by the DAM, see `_flexcan_fd_frame_length`.

`flexcan_rx_fifo_priority_t` `priority`

The FlexCAN Enhanced Rx FIFO receive priority.

`struct` `_flexcan_mb_transfer`

`#include <fsl_flexcan.h>` FlexCAN Message Buffer transfer.

Public Members

`flexcan_frame_t` `*frame`

The buffer of CAN Message to be transfer.

`uint8_t` `mbIdx`

The index of Message buffer used to transfer Message.

`struct` `_flexcan_fifo_transfer`

`#include <fsl_flexcan.h>` FlexCAN Rx FIFO transfer.

Public Members

`flexcan_fd_frame_t` `*framefd`

The buffer of CAN Message to be received from Enhanced Rx FIFO.

`flexcan_frame_t` `*frame`

The buffer of CAN Message to be received from Legacy Rx FIFO.

`size_t` `frameNum`

Number of CAN Message need to be received from Legacy or Enhanced Rx FIFO.

`struct` `_flexcan_handle`

`#include <fsl_flexcan.h>` FlexCAN handle structure.

Public Members

`flexcan_transfer_callback_t` `callback`

Callback function.

`void` `*userData`

FlexCAN callback function parameter.

`flexcan_frame_t` `*volatile` `mbFrameBuf[CAN_WORD1_COUNT]`

The buffer for received CAN data from Message Buffers.

`flexcan_fd_frame_t` `*volatile` `mbFDFrameBuf[CAN_WORD1_COUNT]`

The buffer for received CAN FD data from Message Buffers.

flexcan_frame_t *volatile rxFifoFrameBuf

The buffer for received CAN data from Legacy Rx FIFO.

flexcan_fd_frame_t *volatile rxFifoFDFrameBuf

The buffer for received CAN FD data from Enhanced Rx FIFO.

size_t rxFifoFrameNum

The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.

size_t rxFifoTransferTotalNum

Total CAN Message number need to be received from Legacy or Enhanced Rx FIFO.

volatile uint8_t mbState[CAN_WORD1_COUNT]

Message Buffer transfer state.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

volatile uint32_t timestamp[CAN_WORD1_COUNT]

Mailbox transfer timestamp.

struct byteStatus

Public Members

bool byteIsRead

The byte n (0~3) was read or not. The type of error and which bit in byte (n) is affected by the error.

struct __unnamed14__

Public Members

uint32_t timestamp

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length

CAN frame data length in bytes (Range: 0~8).

uint32_t type

CAN Frame Type(DATA or REMOTE).

uint32_t format

CAN Frame Identifier(STD or EXT format).

uint32_t __pad0__

Reserved.

uint32_t idhit

CAN Rx FIFO filter hit id(This value is only used in Rx FIFO receive mode).

struct __unnamed16__

Public Members

uint32_t id

CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

```
uint32_t __pad0__  
    Reserved.  
union __unnamed18__
```

Public Members

```
struct __flexcan_frame  
struct __flexcan_frame  
struct __unnamed20__
```

Public Members

```
uint32_t dataWord0  
    CAN Frame payload word0.  
uint32_t dataWord1  
    CAN Frame payload word1.  
struct __unnamed22__
```

Public Members

```
uint8_t dataByte3  
    CAN Frame payload byte3.  
uint8_t dataByte2  
    CAN Frame payload byte2.  
uint8_t dataByte1  
    CAN Frame payload byte1.  
uint8_t dataByte0  
    CAN Frame payload byte0.  
uint8_t dataByte7  
    CAN Frame payload byte7.  
uint8_t dataByte6  
    CAN Frame payload byte6.  
uint8_t dataByte5  
    CAN Frame payload byte5.  
uint8_t dataByte4  
    CAN Frame payload byte4.  
struct __unnamed24__
```

Public Members

```
uint32_t timestamp  
    FlexCAN internal Free-Running Counter Time Stamp.
```

uint32_t length

CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length ≤ 8 , it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32_t type

CAN Frame Type(DATA only).

uint32_t format

CAN Frame Identifier(STD or EXT format).

uint32_t srr

Substitute Remote request.

uint32_t esi

Error State Indicator.

uint32_t brs

Bit Rate Switch.

uint32_t edl

Extended Data Length.

struct `__unnamed26__`

Public Members

uint32_t id

CAN Frame Identifier, should be set using `FLEXCAN_ID_EXT()` or `FLEXCAN_ID_STD()` macro.

uint32_t `__pad0__`

Reserved.

union `__unnamed28__`

Public Members

struct `_flexcan_fd_frame`

struct `_flexcan_fd_frame`

struct `__unnamed30__`

Public Members

uint32_t dataWord[16]

CAN FD Frame payload, 16 double word maximum.

struct `__unnamed32__`

Public Members

uint8_t dataByte3

CAN Frame payload byte3.

uint8_t dataByte2
CAN Frame payload byte2.

uint8_t dataByte1
CAN Frame payload byte1.

uint8_t dataByte0
CAN Frame payload byte0.

uint8_t dataByte7
CAN Frame payload byte7.

uint8_t dataByte6
CAN Frame payload byte6.

uint8_t dataByte5
CAN Frame payload byte5.

uint8_t dataByte4
CAN Frame payload byte4.

union __unnamed34__

Public Members

struct __flexcan_config

struct __flexcan_config

struct __unnamed36__

Public Members

uint32_t baudRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t baudRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

struct __unnamed38__

Public Members

uint32_t bitRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t bitRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

union __unnamed40__

Public Members

struct __flexcan_pn_config

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

```
struct _flexcan_pn_config
```

```
struct __unnamed44__
```

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

Public Members

```
uint32_t lowerWord0
    CAN Frame payload word0.
```

```
uint32_t lowerWord1
    CAN Frame payload word1.
```

```
struct __unnamed46__
```

Public Members

```
uint8_t lowerByte3
    CAN Frame payload byte3.
```

```
uint8_t lowerByte2
    CAN Frame payload byte2.
```

```
uint8_t lowerByte1
    CAN Frame payload byte1.
```

```
uint8_t lowerByte0
    CAN Frame payload byte0.
```

```
uint8_t lowerByte7
    CAN Frame payload byte7.
```

```
uint8_t lowerByte6
    CAN Frame payload byte6.
```

```
uint8_t lowerByte5
    CAN Frame payload byte5.
```

```
uint8_t lowerByte4
    CAN Frame payload byte4.
```

```
union __unnamed42__
```

Public Members

```
struct _flexcan_pn_config
```

< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

```
struct _flexcan_pn_config
```

```
struct __unnamed48__
```

< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

Public Members

uint32_t upperWord0
CAN Frame payload word0.

uint32_t upperWord1
CAN Frame payload word1.

struct __unnamed50 __

Public Members

uint8_t upperByte3
CAN Frame payload byte3.

uint8_t upperByte2
CAN Frame payload byte2.

uint8_t upperByte1
CAN Frame payload byte1.

uint8_t upperByte0
CAN Frame payload byte0.

uint8_t upperByte7
CAN Frame payload byte7.

uint8_t upperByte6
CAN Frame payload byte6.

uint8_t upperByte5
CAN Frame payload byte5.

uint8_t upperByte4
CAN Frame payload byte4.

2.19 FlexCAN eDMA Driver

```
void FLEXCAN_TransferCreateHandleEDMA(CAN_Type *base, flexcan_edma_handle_t *handle,  
                                       flexcan_edma_transfer_callback_t callback, void  
                                       *userData, edma_handle_t *rxFifoEdmaHandle)
```

Initializes the FlexCAN handle, which is used in transactional functions.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

```
void FLEXCAN_PrepareTransfConfiguration(CAN_Type *base, flexcan_fifo_transfer_t *pFifoXfer,  
                                       edma_transfer_config_t *pEdmaConfig)
```

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

status_t FLEXCAN_StartTransferDatafromRxFIFO(CAN_Type *base, flexcan_edma_handle_t *handle, edma_transfer_config_t *pEdmaConfig)

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer configuration and Start eDMA transfer

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferGetReceiveFifoCountEMDA(CAN_Type *base, flexcan_edma_handle_t *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

```
void FLEXCAN_TransferAbortReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t
                                         *handle)
```

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.

```
status_t FLEXCAN_TransferReceiveEnhancedFifoEDMA(CAN_Type *base, flexcan_edma_handle_t
                                                *handle, flexcan_fifo_transfer_t
                                                *pFifoXfer)
```

Receives the CAN FD Message from the Enhanced Rx FIFO using eDMA.

This function receives the CAN FD Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

```
static inline status_t FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA(CAN_Type *base,
                                                                        flex-
                                                                        can_edma_handle_t
                                                                        *handle, size_t
                                                                        *count)
```

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

```
FSL_FLEXCAN_EDMA_DRIVER_VERSION
```

FlexCAN EDMA driver version.

```
typedef struct flexcan_edma_handle flexcan_edma_handle_t
```

```
typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, flexcan_edma_handle_t
*handle, status_t status, void *userData)
```

FlexCAN transfer callback function.

```
struct flexcan_edma_handle
```

```
#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.
```

Public Members

flexcan_edma_transfer_callback_t callback

Callback function.

void *userData

FlexCAN callback function parameter.

edma_handle_t *rxFifoEdmaHandle

The EDMA handler for Rx FIFO.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

size_t frameNum

The number of messages that need to be received.

flexcan_fd_frame_t *framefd

Point to the buffer of CAN Message to be received from Enhanced Rx FIFO.

2.20 FlexIO: FlexIO Driver

2.21 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

- userConfig – pointer to flexio_config_t structure

void FLEXIO_Init(FLEXIO_Type *base, const *flexio_config_t* *userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_GetDefaultConfig().

Example

```
flexio_config_t config = {
.enableFlexio = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio_config_t structure

```
void FLEXIO_Deinit(FLEXIO_Type *base)
```

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

Note: After calling this API, call the FLEXIO_Init to use the FlexIO module.

Parameters

- base – FlexIO peripheral base address

```
uint32_t FLEXIO_GetInstance(FLEXIO_Type *base)
```

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

```
void FLEXIO_Reset(FLEXIO_Type *base)
```

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

```
static inline void FLEXIO_Enable(FLEXIO_Type *base, bool enable)
```

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

```
static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)
```

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns

FlexIO pin input data

```
static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)
```

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns

current State pointer

```
void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
```

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {  
.timerSelect = 0,  
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,  
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
```

(continues on next page)

(continued from previous page)

```
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

```
void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t
                          *timerConfig)
```

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFThnSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

```
static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index,
                                       flexio_timer_decrement_source_t clocksource)
```

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)
Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)
Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)
Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)
Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)
Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)
Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$
- enable – True to enable, false to disable.

uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, flexio_shifter_buffer_type_t type, uint8_t index)

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio_shifter_buffer_type_t

- index – Shifter index

Returns

Corresponding shifter buffer index

`status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, flexio_isr_t isr)`

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`status_t FLEXIO_UnregisterHandleIRQ(void *base)`

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)`

Sets the output level of the multiple FLEXIO pins to the logic 0.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

`static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)`

Sets the output level of the multiple FLEXIO pins to the logic 1.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

`static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)`

Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

`static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)`

Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- output – FLEXIO pin output logic level.

- 0: corresponding pin output low-logic level.
- 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

```
static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)
```

Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

```
static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)
```

Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

```
FSL_FLEXIO_DRIVER_VERSION
```

FlexIO driver version.

```
enum _flexio_timer_trigger_polarity
```

Define time of timer trigger polarity.

Values:

```
enumerator kFLEXIO_TimerTriggerPolarityActiveHigh  
Active high.
```

```
enumerator kFLEXIO_TimerTriggerPolarityActiveLow  
Active low.
```

```
enum _flexio_timer_trigger_source
```

Define type of timer trigger source.

Values:

```
enumerator kFLEXIO_TimerTriggerSourceExternal  
External trigger selected.
```

```
enumerator kFLEXIO_TimerTriggerSourceInternal  
Internal trigger selected.
```

```
enum _flexio_pin_config
```

Define type of timer/shifter pin configuration.

Values:

```
enumerator kFLEXIO_PinConfigOutputDisabled  
Pin output disabled.
```

```
enumerator kFLEXIO_PinConfigOpenDrainOrBidirection  
Pin open drain or bidirectional output enable.
```

```
enumerator kFLEXIO_PinConfigBidirectionOutputData  
Pin bidirectional output data.
```

```
enumerator kFLEXIO_PinConfigOutput  
Pin output.
```

```
enum _flexio_pin_polarity
```

Definition of pin polarity.

Values:

enumerator kFLEXIO_PinActiveHigh
Active high.

enumerator kFLEXIO_PinActiveLow
Active low.

enum _flexio_timer_mode
Define type of timer work mode.

Values:

enumerator kFLEXIO_TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO_TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO_TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO_TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO_TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum _flexio_timer_output
Define type of timer initial output or timer reset condition.

Values:

enumerator kFLEXIO_TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO_TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum _flexio_timer_decrement_source
Define type of timer decrement.

Values:

enumerator kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput
Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum _flexio_timer_reset_condition
Define type of timer reset condition.

Values:

enumerator kFLEXIO_TimerResetNever

Timer never reset.

enumerator kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput

Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput

Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerPinRisingEdge

Timer reset on Timer Pin rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerRisingEdge

Timer reset on Trigger rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerBothEdge

Timer reset on Trigger rising or falling edge.

enum _flexio_timer_disable_condition

Define type of timer disable condition.

Values:

enumerator kFLEXIO_TimerDisableNever

Timer never disabled.

enumerator kFLEXIO_TimerDisableOnPreTimerDisable

Timer disabled on Timer N-1 disable.

enumerator kFLEXIO_TimerDisableOnTimerCompare

Timer disabled on Timer compare.

enumerator kFLEXIO_TimerDisableOnTimerCompareTriggerLow

Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO_TimerDisableOnPinBothEdge

Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh

Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO_TimerDisableOnTriggerFallingEdge

Timer disabled on Trigger falling edge.

enum _flexio_timer_enable_condition

Define type of timer enable condition.

Values:

enumerator kFLEXIO_TimerEnabledAlways

Timer always enabled.

enumerator kFLEXIO_TimerEnableOnPrevTimerEnable

Timer enabled on Timer N-1 enable.

enumerator kFLEXIO_TimerEnableOnTriggerHigh

Timer enabled on Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerHighPinHigh

Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO_TimerEnableOnPinRisingEdge

Timer enabled on Pin rising edge.

enumerator kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh
Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerRisingEdge
Timer enabled on Trigger rising edge.

enumerator kFLEXIO_TimerEnableOnTriggerBothEdge
Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition
Define type of timer stop bit generate condition.

Values:

enumerator kFLEXIO_TimerStopBitDisabled
Stop bit disabled.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompare
Stop bit is enabled on timer compare.

enumerator kFLEXIO_TimerStopBitEnableOnTimerDisable
Stop bit is enabled on timer disable.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompareDisable
Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition
Define type of timer start bit generate condition.

Values:

enumerator kFLEXIO_TimerStartBitDisabled
Start bit disabled.

enumerator kFLEXIO_TimerStartBitEnabled
Start bit enabled.

enum _flexio_timer_output_state
FlexIO as PWM channel output state.

Values:

enumerator kFLEXIO_PwmLow
The output state of PWM channel is low

enumerator kFLEXIO_PwmHigh
The output state of PWM channel is high

enum _flexio_shifter_timer_polarity
Define type of timer polarity for shifter control.

Values:

enumerator kFLEXIO_ShifterTimerPolarityOnPositive
Shift on positive edge of shift clock.

enumerator kFLEXIO_ShifterTimerPolarityOnNegative
Shift on negative edge of shift clock.

enum _flexio_shifter_mode
Define type of shifter working mode.

Values:

enumerator kFLEXIO__ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO__ShifterModeReceive

Receive mode.

enumerator kFLEXIO__ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO__ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO__ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO__ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO__ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio_shifter_input_source

Define type of shifter input source.

Values:

enumerator kFLEXIO__ShifterInputFromPin

Shifter input from pin.

enumerator kFLEXIO__ShifterInputFromNextShifterOutput

Shifter input from Shifter N+1.

enum _flexio_shifter_stop_bit

Define of STOP bit configuration.

Values:

enumerator kFLEXIO__ShifterStopBitDisable

Disable shifter stop bit.

enumerator kFLEXIO__ShifterStopBitLow

Set shifter stop bit to logic low level.

enumerator kFLEXIO__ShifterStopBitHigh

Set shifter stop bit to logic high level.

enum _flexio_shifter_start_bit

Define type of START bit configuration.

Values:

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnEnable

Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnShift

Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO__ShifterStartBitLow

Set shifter start bit to logic low level.

enumerator kFLEXIO__ShifterStartBitHigh

Set shifter start bit to logic high level.

enum *_flexio_shifter_buffer_type*

Define FlexIO shifter buffer type.

Values:

enumerator *kFLEXIO_ShifterBuffer*

Shifter Buffer N Register.

enumerator *kFLEXIO_ShifterBufferBitSwapped*

Shifter Buffer N Bit Byte Swapped Register.

enumerator *kFLEXIO_ShifterBufferByteSwapped*

Shifter Buffer N Byte Swapped Register.

enumerator *kFLEXIO_ShifterBufferBitByteSwapped*

Shifter Buffer N Bit Swapped Register.

enumerator *kFLEXIO_ShifterBufferNibbleByteSwapped*

Shifter Buffer N Nibble Byte Swapped Register.

enumerator *kFLEXIO_ShifterBufferHalfWordSwapped*

Shifter Buffer N Half Word Swapped Register.

enumerator *kFLEXIO_ShifterBufferNibbleSwapped*

Shifter Buffer N Nibble Swapped Register.

enum *_flexio_gpio_direction*

FLEXIO gpio direction definition.

Values:

enumerator *kFLEXIO_DigitalInput*

Set current pin as digital input

enumerator *kFLEXIO_DigitalOutput*

Set current pin as digital output

enum *_flexio_pin_input_config*

FLEXIO gpio input config.

Values:

enumerator *kFLEXIO_InputInterruptDisabled*

Interrupt request is disabled.

enumerator *kFLEXIO_InputInterruptEnable*

Interrupt request is enable.

enumerator *kFLEXIO_FlagRisingEdgeEnable*

Input pin flag on rising edge.

enumerator *kFLEXIO_FlagFallingEdgeEnable*

Input pin flag on falling edge.

typedef enum *_flexio_timer_trigger_polarity* *flexio_timer_trigger_polarity_t*

Define time of timer trigger polarity.

typedef enum *_flexio_timer_trigger_source* *flexio_timer_trigger_source_t*

Define type of timer trigger source.

typedef enum *_flexio_pin_config* *flexio_pin_config_t*

Define type of timer/shifter pin configuration.

`typedef enum _flexio_pin_polarity flexio_pin_polarity_t`
Definition of pin polarity.

`typedef enum _flexio_timer_mode flexio_timer_mode_t`
Define type of timer work mode.

`typedef enum _flexio_timer_output flexio_timer_output_t`
Define type of timer initial output or timer reset condition.

`typedef enum _flexio_timer_decrement_source flexio_timer_decrement_source_t`
Define type of timer decrement.

`typedef enum _flexio_timer_reset_condition flexio_timer_reset_condition_t`
Define type of timer reset condition.

`typedef enum _flexio_timer_disable_condition flexio_timer_disable_condition_t`
Define type of timer disable condition.

`typedef enum _flexio_timer_enable_condition flexio_timer_enable_condition_t`
Define type of timer enable condition.

`typedef enum _flexio_timer_stop_bit_condition flexio_timer_stop_bit_condition_t`
Define type of timer stop bit generate condition.

`typedef enum _flexio_timer_start_bit_condition flexio_timer_start_bit_condition_t`
Define type of timer start bit generate condition.

`typedef enum _flexio_timer_output_state flexio_timer_output_state_t`
FlexIO as PWM channel output state.

`typedef enum _flexio_shifter_timer_polarity flexio_shifter_timer_polarity_t`
Define type of timer polarity for shifter control.

`typedef enum _flexio_shifter_mode flexio_shifter_mode_t`
Define type of shifter working mode.

`typedef enum _flexio_shifter_input_source flexio_shifter_input_source_t`
Define type of shifter input source.

`typedef enum _flexio_shifter_stop_bit flexio_shifter_stop_bit_t`
Define of STOP bit configuration.

`typedef enum _flexio_shifter_start_bit flexio_shifter_start_bit_t`
Define type of START bit configuration.

`typedef enum _flexio_shifter_buffer_type flexio_shifter_buffer_type_t`
Define FlexIO shifter buffer type.

`typedef struct _flexio_config flexio_config_t`
Define FlexIO user configuration structure.

`typedef struct _flexio_timer_config flexio_timer_config_t`
Define FlexIO timer configuration structure.

`typedef struct _flexio_shifter_config flexio_shifter_config_t`
Define FlexIO shifter configuration structure.

`typedef enum _flexio_gpio_direction flexio_gpio_direction_t`
FLEXIO gpio direction definition.

`typedef enum _flexio_pin_input_config flexio_pin_input_config_t`
FLEXIO gpio input config.

```
typedef struct flexio_gpio_config flexio_gpio_config_t
```

The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

```
typedef void (*flexio_isr_t)(void *base, void *handle)
```

typedef for FlexIO simulated driver interrupt handler.

```
FLEXIO_Type *const s_flexioBases[]
```

Pointers to flexio bases for each instance.

```
const clock_ip_name_t s_flexioClocks[]
```

Pointers to flexio clocks for each instance.

```
void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, flexio_gpio_config_t *config)
```

Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the FLEXIO_SetPinConfig() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalInput,
    0U,
    kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalOutput,
    0U,
    0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

```
FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)
```

Calculate FlexIO timer trigger.

```
FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(x)
```

```
FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)
```

```
struct flexio_config_
```

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members

```
bool enableFlexio
```

Enable/disable FlexIO module

`bool enableInDoze`
Enable/disable FlexIO operation in doze mode

`bool enableInDebug`
Enable/disable FlexIO operation in debug mode

`bool enableFastAccess`
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

`struct _flexio_timer_config`
#include <fsl_flexio.h> Define FlexIO timer configuration structure.

Public Members

`uint32_t triggerSelect`
The internal trigger selection number using MACROs.

`flexio_timer_trigger_polarity_t triggerPolarity`
Trigger Polarity.

`flexio_timer_trigger_source_t triggerSource`
Trigger Source, internal (see 'trgsel') or external.

`flexio_pin_config_t pinConfig`
Timer Pin Configuration.

`uint32_t pinSelect`
Timer Pin number Select.

`flexio_pin_polarity_t pinPolarity`
Timer Pin Polarity.

`flexio_timer_mode_t timerMode`
Timer work Mode.

`flexio_timer_output_t timerOutput`
Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

`flexio_timer_decrement_source_t timerDecrement`
Configures the source of the Timer decrement and the source of the Shift clock.

`flexio_timer_reset_condition_t timerReset`
Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

`flexio_timer_disable_condition_t timerDisable`
Configures the condition that causes the Timer to be disabled and stop decrementing.

`flexio_timer_enable_condition_t timerEnable`
Configures the condition that causes the Timer to be enabled and start decrementing.

`flexio_timer_stop_bit_condition_t timerStop`
Timer STOP Bit generation.

`flexio_timer_start_bit_condition_t timerStart`
Timer STRAT Bit generation.

`uint32_t timerCompare`
Value for Timer Compare N Register.

```
struct _flexio_shifter_config
```

```
    #include <fsl_flexio.h> Define FlexIO shifter configuration structure.
```

Public Members

```
uint32_t timerSelect
```

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

```
flexio_shifter_timer_polarity_t timerPolarity
```

Timer Polarity.

```
flexio_pin_config_t pinConfig
```

Shifter Pin Configuration.

```
uint32_t pinSelect
```

Shifter Pin number Select.

```
flexio_pin_polarity_t pinPolarity
```

Shifter Pin Polarity.

```
flexio_shifter_mode_t shifterMode
```

Configures the mode of the Shifter.

```
uint32_t parallelWidth
```

Configures the parallel width when using parallel mode.

```
flexio_shifter_input_source_t inputSource
```

Selects the input source for the shifter.

```
flexio_shifter_stop_bit_t shifterStop
```

Shifter STOP bit.

```
flexio_shifter_start_bit_t shifterStart
```

Shifter START bit.

```
struct _flexio_gpio_config
```

```
    #include <fsl_flexio.h> The FLEXIO pin configuration structure.
```

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

```
flexio_gpio_direction_t pinDirection
```

FLEXIO pin direction, input or output

```
uint8_t outputLogic
```

Set a default output logic, which has no use in input

```
uint8_t inputConfig
```

Set an input config

2.22 FlexIO eDMA SPI Driver

```
status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,
                                                    flexio_spi_master_edma_handle_t
                                                    *handle,
                                                    flexio_spi_master_edma_transfer_callback_t
                                                    callback, void *userData,
                                                    edma_handle_t *txHandle,
                                                    edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI master eDMA handle.

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

```
status_t FLEXIO_SPI_MasterTransferEDMA(FLEXIO_SPI_Type *base,
                                        flexio_spi_master_edma_handle_t *handle,
                                        flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

```
void FLEXIO_SPI_MasterTransferAbortEDMA(FLEXIO_SPI_Type *base,
                                         flexio_spi_master_edma_handle_t *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

```
status_t FLEXIO_SPI_MasterTransferGetCountEDMA(FLEXIO_SPI_Type *base,  
                                              flexio_spi_master_edma_handle_t *handle,  
                                              size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

```
static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_slave_edma_handle_t  
                                                         *handle,  
                                                         flexio_spi_slave_edma_transfer_callback_t  
                                                         callback, void *userData,  
                                                         edma_handle_t *txHandle,  
                                                         edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

```
status_t FLEXIO_SPI_SlaveTransferEDMA(FLEXIO_SPI_Type *base,  
                                       flexio_spi_slave_edma_handle_t *handle,  
                                       flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.

- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(FLEXIO_SPI_Type *base,
                                                    flexio_spi_slave_edma_handle_t
                                                    *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to *FLEXIO_SPI_Type* structure.
- handle – Pointer to *flexio_spi_slave_edma_handle_t* structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCountEDMA(FLEXIO_SPI_Type *base,
                                                           flexio_spi_slave_edma_handle_t
                                                           *handle, size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- base – Pointer to *FLEXIO_SPI_Type* structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION

FlexIO SPI EDMA driver version.

```
typedef struct flexio_spi_master_edma_handle flexio_spi_master_edma_handle_t
typedef for flexio_spi_master_edma_handle_t in advance.
```

```
typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t
Slave handle is the same with master handle.
```

```
typedef void (*flexio_spi_master_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_master_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_slave_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
struct flexio_spi_master_edma_handle
```

#include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch the content of the handle.

Public Members

size_t transferSize

Total bytes to be transferred.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

bool txInProgress

Send transfer in progress

bool rxInProgress

Receive transfer in progress

edma_handle_t *txHandle
DMA handler for SPI send

edma_handle_t *rxHandle
DMA handler for SPI receive

flexio_spi_master_edma_transfer_callback_t callback
Callback for SPI DMA transfer

void *userData
User Data for SPI DMA callback

2.23 FlexIO eDMA UART Driver

status_t FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base,
flexio_uart_edma_handle_t *handle,
flexio_uart_edma_transfer_callback_t
callback, void *userData, *edma_handle_t*
*txEdmaHandle, *edma_handle_t*
*rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_UART_Type.
- handle – Pointer to flexio_uart_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base,
flexio_uart_edma_handle_t *handle,
flexio_uart_transfer_t *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

```
status_t FLEXIO_UART_TransferReceiveEDMA(FLEXIO_UART_Type *base,
                                          flexio_uart_edma_handle_t *handle,
                                          flexio_uart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

```
void FLEXIO_UART_TransferAbortSendEDMA(FLEXIO_UART_Type *base,
                                        flexio_uart_edma_handle_t *handle)
```

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
void FLEXIO_UART_TransferAbortReceiveEDMA(FLEXIO_UART_Type *base,
                                           flexio_uart_edma_handle_t *handle)
```

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
status_t FLEXIO_UART_TransferGetSendCountEDMA(FLEXIO_UART_Type *base,
                                               flexio_uart_edma_handle_t *handle,
                                               size_t *count)
```

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

```
status_t FLEXIO_UART_TransferGetReceiveCountEDMA(FLEXIO_UART_Type *base,  
                                                flexio_uart_edma_handle_t *handle,  
                                                size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- base – Pointer to *FLEXIO_UART_Type*
- handle – Pointer to *flexio_uart_edma_handle_t* structure
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- *kStatus_NoTransferInProgress* – transfer has finished or no transfer in progress.
- *kStatus_Success* – Successfully return the count.

```
FSL_FLEXIO_UART_EDMA_DRIVER_VERSION
```

FlexIO UART EDMA driver version.

```
typedef struct flexio_uart_edma_handle flexio_uart_edma_handle_t
```

```
typedef void (*flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base,  
flexio_uart_edma_handle_t *handle, status_t status, void *userData)
```

UART transfer callback function.

```
struct flexio_uart_edma_handle
```

```
#include <fsl_flexio_uart_edma.h> UART eDMA handle.
```

Public Members

```
flexio_uart_edma_transfer_callback_t callback
```

Callback function.

```
void *userData
```

UART callback function parameter.

```
size_t txDataSizeAll
```

Total bytes to be sent.

```
size_t rxDataSizeAll
```

Total bytes to be received.

```
edma_handle_t *txEdmaHandle
```

The eDMA TX channel used.

```
edma_handle_t *rxEdmaHandle
```

The eDMA RX channel used.

```
uint8_t nbytes
```

eDMA minor byte transfer count initially configured.

```
volatile uint8_t txState
```

TX transfer state.

```
volatile uint8_t rxState
```

RX transfer state

2.24 FlexIO I2C Master Driver

`status_t FLEXIO_I2C_CheckForBusyBus(FLEXIO_I2C_Type *base)`

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure..

Return values

- `kStatus_Success` –
- `kStatus_FLEXIO_I2C_Busy` –

`status_t FLEXIO_I2C_MasterInit(FLEXIO_I2C_Type *base, flexio_i2c_master_config_t *masterConfig, uint32_t srcClock_Hz)`

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.
- `masterConfig` – Pointer to `flexio_i2c_master_config_t` structure.
- `srcClock_Hz` – FlexIO source clock in Hz.

Return values

- `kStatus_Success` – Initialization successful
- `kStatus_InvalidArgument` – The source clock exceed upper range limitation

`void FLEXIO_I2C_MasterDeinit(FLEXIO_I2C_Type *base)`

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifter and timer config, module can't work unless the `FLEXIO_I2C_MasterInit` is called.

Parameters

- `base` – pointer to `FLEXIO_I2C_Type` structure.

`void FLEXIO_I2C_MasterGetDefaultConfig(flexio_i2c_master_config_t *masterConfig)`

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the `FLEXIO_I2C_MasterInit()`.

Example:

```
flexio_i2c_master_config_t config;  
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)
Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)
Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND `_flexio_i2c_master_status_flags` can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)
Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_I2C_RxFullFlag
 - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)
Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
 - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)
Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps,
uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ

- srcClock_Hz – source clock in HZ

```
void FLEXIO_I2C_MasterStart(FLEXIO_I2C_Type *base, uint8_t address, flexio_i2c_direction_t
                             direction)
```

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
 - kFLEXIO_I2C_Write: Transmit
 - kFLEXIO_I2C_Read: Receive

```
void FLEXIO_I2C_MasterStop(FLEXIO_I2C_Type *base)
```

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

```
void FLEXIO_I2C_MasterRepeatedStart(FLEXIO_I2C_Type *base)
```

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

```
void FLEXIO_I2C_MasterAbortStop(FLEXIO_I2C_Type *base)
```

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

```
void FLEXIO_I2C_MasterEnableAck(FLEXIO_I2C_Type *base, bool enable)
```

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

```
status_t FLEXIO_I2C_MasterSetTransferCount(FLEXIO_I2C_Type *base, uint16_t count)
```

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

- `count` – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- `kStatus_Success` – Successfully configured the count.
- `kStatus_InvalidArgument` – Input argument is invalid.

`static inline void FLEXIO_I2C_MasterWriteByte(FLEXIO_I2C_Type *base, uint32_t data)`

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the `TxEEmptyFlag` is asserted before calling this API.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.
- `data` – a byte of data.

`static inline uint8_t FLEXIO_I2C_MasterReadByte(FLEXIO_I2C_Type *base)`

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.

Returns

data byte read.

`status_t FLEXIO_I2C_MasterWriteBlocking(FLEXIO_I2C_Type *base, const uint8_t *txBuff, uint8_t txSize)`

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.
- `txBuff` – The data bytes to send.
- `txSize` – The number of data bytes to send.

Return values

- `kStatus_Success` – Successfully write data.
- `kStatus_FLEXIO_I2C_Nak` – Receive NAK during writing data.
- `kStatus_FLEXIO_I2C_Timeout` – Timeout polling status flags.

`status_t FLEXIO_I2C_MasterReadBlocking(FLEXIO_I2C_Type *base, uint8_t *rxBuff, uint8_t rxSize)`

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_transfer_t *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_handle_t *handle,
flexio_i2c_master_transfer_callback_t
callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_handle_t *handle,
flexio_i2c_master_transfer_t *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the

transfer is finished. If the return status is not `kStatus_FLEXIO_I2C_Busy`, the transfer is finished.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state
- `xfer` – pointer to `flexio_i2c_master_transfer_t` structure

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_FLEXIO_I2C_Busy` – FlexIO I2C is not idle, is running another transfer.

`status_t` `FLEXIO_I2C_MasterTransferGetCount`(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_handle_t *handle, *size_t*
*count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.
- `kStatus_Success` – Successfully return the count.

`void` `FLEXIO_I2C_MasterTransferAbort`(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t*
*handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state

`void` `FLEXIO_I2C_MasterTransferHandleIRQ`(*void* *i2cType, *void* *i2cHandle)

Master interrupt handler.

Parameters

- `i2cType` – Pointer to `FLEXIO_I2C_Type` structure
- `i2cHandle` – Pointer to `flexio_i2c_master_transfer_t` structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy
I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle
I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak
NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout
Timeout polling status flags.

enum _flexio_i2c_master_interrupt
Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable
Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable
Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags
Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag
Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag
Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag
Receive NAK flag.

enum _flexio_i2c_direction
Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write
Master send to slave.

enumerator kFLEXIO_I2C_Read
Master receive from slave.

typedef enum *flexio_i2c_direction* flexio_i2c_direction_t
Direction of master transfer.

typedef struct *flexio_i2c_type* FLEXIO_I2C_Type
Define FlexIO I2C master access structure typedef.

typedef struct *flexio_i2c_master_config* flexio_i2c_master_config_t
Define FlexIO I2C master user configuration structure.

```
typedef struct _flexio_i2c_master_transfer flexio_i2c_master_transfer_t
```

Define FlexIO I2C master transfer structure.

```
typedef struct _flexio_i2c_master_handle flexio_i2c_master_handle_t
```

FlexIO I2C master handle typedef.

```
typedef void (*flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base,  
flexio_i2c_master_handle_t *handle, status_t status, void *userData)
```

FlexIO I2C master transfer callback typedef.

```
I2C_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _flexio_i2c_type
```

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

```
FLEXIO_Type *flexioBase
```

FlexIO base pointer.

```
uint8_t SDAPinIndex
```

Pin select for I2C SDA.

```
uint8_t SCLPinIndex
```

Pin select for I2C SCL.

```
uint8_t shifterIndex[2]
```

Shifter index used in FlexIO I2C.

```
uint8_t timerIndex[3]
```

Timer index used in FlexIO I2C.

```
uint32_t baudrate
```

Master transfer baudrate, used to calculate delay time.

```
struct _flexio_i2c_master_config
```

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

```
bool enableMaster
```

Enables the FlexIO I2C peripheral at initialization time.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode.

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode.

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
uint32_t baudRate_Bps
```

Baud rate in Bps.

```
struct _flexio_i2c_master_transfer
```

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags

Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress

7-bit slave address.

flexio_i2c_direction_t direction

Transfer direction, read or write.

uint32_t subaddress

Sub address. Transferred MSB first.

uint8_t subaddressSize

Size of sub address.

uint8_t volatile *data

Transfer buffer.

volatile size_t dataSize

Transfer size.

struct *flexio_i2c_master_handle*

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer

FlexIO I2C master transfer copy.

size_t transferSize

Total bytes to be transferred.

uint8_t state

Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback

Callback function called at transfer event. Callback function called at transfer event.

void *userData

Callback parameter passed to callback function.

bool needRestart

Whether master needs to send re-start signal.

2.25 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO_I2S_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in FLEXIO_I2S_Init(). Users may use the initialized structure unchanged in FLEXIO_I2S_Init() or modify some fields of the structure before calling FLEXIO_I2S_Init().

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXIO_I2S_Init to use the FlexIO I2S module.

Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO_I2S_Enable(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S module operation.

Parameters

- base – Pointer to FLEXIO_I2S_Type
- enable – True to enable, false dose not have any effect.

uint32_t FLEXIO_I2S_GetStatusFlags(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S status flags.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

Status flag, which are ORed by the enumerators in the `_flexio_i2s_status_flags`.

void FLEXIO_I2S_EnableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

void FLEXIO_I2S_DisableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

```
static inline void FLEXIO_I2S_TxEnableDMA(FLEXIO_I2S_Type *base, bool enable)
```

Enables/disables the FlexIO I2S Tx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline void FLEXIO_I2S_RxEnableDMA(FLEXIO_I2S_Type *base, bool enable)
```

Enables/disables the FlexIO I2S Rx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s send data register address.

```
static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s receive data register address.

```
void FLEXIO_I2S_MasterSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format,  
                                uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – I2S master clock source frequency in Hz.

```
void FLEXIO_I2S_SlaveSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format)
```

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.

`status_t FLEXIO_I2S_WriteBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *txData, size_t size)`

Sends data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- `base` – FlexIO I2S base pointer.
- `bitWidth` – How many bits in a audio word, usually 8/16/24/32 bits.
- `txData` – Pointer to the data to be written.
- `size` – Bytes to be written.

Return values

- `kStatus_Success` – Successfully write data.
- `kStatus_FLEXIO_I2C_Timeout` – Timeout polling status flags.

`static inline void FLEXIO_I2S_WriteData(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint32_t data)`

Writes data into a data register.

Parameters

- `base` – FlexIO I2S base pointer.
- `bitWidth` – How many bits in a audio word, usually 8/16/24/32 bits.
- `data` – Data to be written.

`status_t FLEXIO_I2S_ReadBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *rxData, size_t size)`

Receives a piece of data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- `base` – FlexIO I2S base pointer
- `bitWidth` – How many bits in a audio word, usually 8/16/24/32 bits.
- `rxData` – Pointer to the data to be read.
- `size` – Bytes to be read.

Return values

- `kStatus_Success` – Successfully read data.
- `kStatus_FLEXIO_I2C_Timeout` – Timeout polling status flags.

`static inline uint32_t FLEXIO_I2S_ReadData(FLEXIO_I2S_Type *base)`

Reads a data from the data register.

Parameters

- `base` – FlexIO I2S base pointer

Returns

Data read from data register.

```
void FLEXIO_I2S_TransferTxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,
                                       flexio_i2s_callback_t callback, void *userData)
```

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- handle – Pointer to *flexio_i2s_handle_t* structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

```
void FLEXIO_I2S_TransferSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,
                                  flexio_i2s_format_t *format, uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

```
void FLEXIO_I2S_TransferRxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,
                                       flexio_i2s_callback_t callback, void *userData)
```

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.
- handle – Pointer to *flexio_i2s_handle_t* structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

```
status_t FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t
                                             *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call *FLEXIO_I2S_GetRemainingBytes* to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXIO_I2S_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t FLEXIO_I2S_TransferReceiveNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data receive.
- kStatus_FLEXIO_I2S_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

void FLEXIO_I2S_TransferAbortSend(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)
Aborts the current send.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

void FLEXIO_I2S_TransferAbortReceive(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)
Aborts the current receive.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

status_t FLEXIO_I2S_TransferGetSendCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be sent.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t FLEXIO_I2S_TransferGetReceiveCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be received.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes recieved.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

Returns

count Bytes received.

void FLEXIO_I2S_TransferTxHandleIRQ(*void* *i2sBase, *void* *i2sHandle)

Tx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure

void FLEXIO_I2S_TransferRxHandleIRQ(*void* *i2sBase, *void* *i2sHandle)

Rx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure.

FSL_FLEXIO_I2S_DRIVER_VERSION

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator kStatus_FLEXIO_I2S_Idle
FlexIO I2S is in idle state

enumerator kStatus_FLEXIO_I2S_TxBusy
FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_RxBusy
FlexIO I2S Rx is busy

enumerator kStatus_FLEXIO_I2S_Error
FlexIO I2S error occurred

enumerator kStatus_FLEXIO_I2S_QueueFull
FlexIO I2S transfer queue is full.

enumerator kStatus_FLEXIO_I2S_Timeout
FlexIO I2S timeout polling status flags.

enum _flexio_i2s_master_slave

Master or slave mode.

Values:

enumerator kFLEXIO_I2S_Master
Master mode

enumerator kFLEXIO_I2S_Slave
Slave mode

_flexio_i2s_interrupt_enable Define FlexIO FlexIO I2S interrupt mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_I2S_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

_flexio_i2s_status_flags Define FlexIO FlexIO I2S status mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_I2S_RxDataRegFullFlag
Receive buffer full flag.

enum _flexio_i2s_sample_rate

Audio sample rate.

Values:

enumerator kFLEXIO_I2S_SampleRate8KHz
Sample rate 8000Hz

```

enumerator kFLEXIO_I2S_SampleRate11025Hz
    Sample rate 11025Hz
enumerator kFLEXIO_I2S_SampleRate12KHz
    Sample rate 12000Hz
enumerator kFLEXIO_I2S_SampleRate16KHz
    Sample rate 16000Hz
enumerator kFLEXIO_I2S_SampleRate22050Hz
    Sample rate 22050Hz
enumerator kFLEXIO_I2S_SampleRate24KHz
    Sample rate 24000Hz
enumerator kFLEXIO_I2S_SampleRate32KHz
    Sample rate 32000Hz
enumerator kFLEXIO_I2S_SampleRate44100Hz
    Sample rate 44100Hz
enumerator kFLEXIO_I2S_SampleRate48KHz
    Sample rate 48000Hz
enumerator kFLEXIO_I2S_SampleRate96KHz
    Sample rate 96000Hz
enum _flexio_i2s_word_width
    Audio word width.
    Values:
enumerator kFLEXIO_I2S_WordWidth8bits
    Audio data width 8 bits
enumerator kFLEXIO_I2S_WordWidth16bits
    Audio data width 16 bits
enumerator kFLEXIO_I2S_WordWidth24bits
    Audio data width 24 bits
enumerator kFLEXIO_I2S_WordWidth32bits
    Audio data width 32 bits
typedef struct flexio_i2s_type FLEXIO_I2S_Type
    Define FlexIO I2S access structure typedef.
typedef enum flexio_i2s_master_slave flexio_i2s_master_slave_t
    Master or slave mode.
typedef struct flexio_i2s_config flexio_i2s_config_t
    FlexIO I2S configure structure.
typedef struct flexio_i2s_format flexio_i2s_format_t
    FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.
typedef enum flexio_i2s_sample_rate flexio_i2s_sample_rate_t
    Audio sample rate.
typedef enum flexio_i2s_word_width flexio_i2s_word_width_t
    Audio word width.

```

```
typedef struct _flexio_i2s_transfer flexio_i2s_transfer_t
```

Define FlexIO I2S transfer structure.

```
typedef struct _flexio_i2s_handle flexio_i2s_handle_t
```

```
typedef void (*flexio_i2s_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
status_t status, void *userData)
```

FlexIO I2S xfer callback prototype.

```
I2S_RETRY_TIMES
```

Retry times for waiting flag.

```
FLEXIO_I2S_XFER_QUEUE_SIZE
```

FlexIO I2S transfer queue size, user can refine it according to use case.

```
struct _flexio_i2s_type
```

```
#include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.
```

Public Members

```
FLEXIO_Type *flexioBase
```

FlexIO base pointer

```
uint8_t txPinIndex
```

Tx data pin index in FlexIO pins

```
uint8_t rxPinIndex
```

Rx data pin index

```
uint8_t bclkPinIndex
```

Bit clock pin index

```
uint8_t fsPinIndex
```

Frame sync pin index

```
uint8_t txShifterIndex
```

Tx data shifter index

```
uint8_t rxShifterIndex
```

Rx data shifter index

```
uint8_t bclkTimerIndex
```

Bit clock timer index

```
uint8_t fsTimerIndex
```

Frame sync timer index

```
struct _flexio_i2s_config
```

```
#include <fsl_flexio_i2s.h> FlexIO I2S configure structure.
```

Public Members

```
bool enableI2S
```

Enable FlexIO I2S

```
flexio_i2s_master_slave_t masterSlave
```

Master or slave

```
flexio_pin_polarity_t txPinPolarity
```

Tx data pin polarity, active high or low

flexio_pin_polarity_t rxPinPolarity
Rx data pin polarity

flexio_pin_polarity_t bclkPinPolarity
Bit clock pin polarity

flexio_pin_polarity_t fsPinPolarity
Frame sync pin polarity

flexio_shifter_timer_polarity_t txTimerPolarity
Tx data valid on bclk rising or falling edge

flexio_shifter_timer_polarity_t rxTimerPolarity
Rx data valid on bclk rising or falling edge

struct *_flexio_i2s_format*

#include <fsl_flexio_i2s.h> FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

Public Members

uint8_t bitWidth
Bit width of audio data, always 8/16/24/32 bits

uint32_t sampleRate_Hz
Sample rate of the audio data

struct *_flexio_i2s_transfer*

#include <fsl_flexio_i2s.h> Define FlexIO I2S transfer structure.

Public Members

uint8_t *data
Data buffer start pointer

size_t dataSize
Bytes to be transferred.

struct *_flexio_i2s_handle*

#include <fsl_flexio_i2s.h> Define FlexIO I2S handle structure.

Public Members

uint32_t state
Internal state

flexio_i2s_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32bits

flexio_i2s_transfer_t queue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
 Data bytes need to transfer

volatile uint8_t queueUser
 Index for user to queue transfer

volatile uint8_t queueDriver
 Index for driver to get the transfer data and size

2.26 FlexIO SPI Driver

```
void FLEXIO_SPI_MasterInit(FLEXIO_SPI_Type *base, flexio_spi_master_config_t  
                          *masterConfig, uint32_t srcClock_Hz)
```

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Example

```
FLEXIO_SPI_Type spiDev = {  
.flexioBase = FLEXIO,  
.SDOPinIndex = 0,  
.SDIPinIndex = 1,  
.SCKPinIndex = 2,  
.CSnPinIndex = 3,  
.shifterIndex = {0,1},  
.timerIndex = {0,1}  
};  
flexio_spi_master_config_t config = {  
.enableMaster = true,  
.enableInDoze = false,  
.enableInDebug = true,  
.enableFastAccess = false,  
.baudRate_Bps = 500000,  
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,  
.direction = kFLEXIO_SPI_MsbFirst,  
.dataMode = kFLEXIO_SPI_8BitMode  
};  
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $2*2=4$. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by $(1.5+2.5)*2=8$.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

```
void FLEXIO_SPI_MasterDeinit(FLEXIO_SPI_Type *base)
```

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

```
void FLEXIO_SPI_MasterGetDefaultConfig(flexio_spi_master_config_t *masterConfig)
```

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

```
void FLEXIO_SPI_SlaveInit(FLEXIO_SPI_Type *base, flexio_spi_slave_config_t *slaveConfig)
```

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1. Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3. For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $3*2=6$. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by $(1.5+2.5)*2=8$. Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0}
};
flexio_spi_slave_config_t config = {
    .enableSlave = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type*.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the *FLEXIO_SPI_SlaveConfigure()*. Example:

```
flexio_spi_slave_config_t slaveConfig;  
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the *flexio_spi_slave_config_t* structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- mask – status flag The parameter can be any combination of the following values:
 - kFLEXIO_SPI_TxEmptyFlag
 - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- mask – interrupt source. The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_DisableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `mask` – interrupt source The parameter can be any combination of the following values:
 - `kFLEXIO_SPI_RxFullInterruptEnable`
 - `kFLEXIO_SPI_TxEmptyInterruptEnable`

```
void FLEXIO_SPI_EnableDMA(FLEXIO_SPI_Type *base, uint32_t mask, bool enable)
```

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the `kFLEXIO_SPI_TxEmptyFlag` does/doesn't trigger the DMA request.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `mask` – SPI DMA source.
- `enable` – True means enable DMA, false means disable DMA.

```
static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(FLEXIO_SPI_Type *base,
                                                         flexio_spi_shift_direction_t
                                                         direction)
```

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `direction` – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

```
static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(FLEXIO_SPI_Type *base,
                                                         flexio_spi_shift_direction_t
                                                         direction)
```

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `direction` – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

```
static inline void FLEXIO_SPI_Enable(FLEXIO_SPI_Type *base, bool enable)
```

Enables/disables the FlexIO SPI module operation.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type`.
- `enable` – True to enable, false does not have any effect.

```
void FLEXIO_SPI_MasterSetBaudRate(FLEXIO_SPI_Type *base, uint32_t baudRate_Bps,
                                  uint32_t srcClockHz)
```

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

```
static inline void FLEXIO_SPI_WriteData(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t
                                         direction, uint32_t data)
```

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

```
static inline uint32_t FLEXIO_SPI_ReadData(FLEXIO_SPI_Type *base,
                                           flexio_spi_shift_direction_t direction)
```

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

```
status_t FLEXIO_SPI_WriteBlocking(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t
                                   direction, const uint8_t *buffer, size_t size)
```

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, *uint8_t* *buffer, *size_t* size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- *kStatus_Success* – Successfully create the handle.
- *kStatus_FLEXIO_SPI_Timeout* – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – pointer to *FLEXIO_SPI_Type* structure
- xfer – FlexIO SPI transfer structure, see *flexio_spi_transfer_t*.

Return values

- *kStatus_Success* – Successfully create the handle.
- *kStatus_FLEXIO_SPI_Timeout* – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, *void* *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- handle – Pointer to the *flexio_spi_master_handle_t* structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base,
flexio_spi_master_handle_t *handle,
flexio_spi_transfer_t *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t*
*handle)

Aborts the master data transfer, which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base,
flexio_spi_master_handle_t *handle, *size_t*
*count)

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_MasterTransferHandleIRQ(*void* *spiType, *void* *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

```
status_t FLEXIO_SPI_SlaveTransferCreateHandle(FLEXIO_SPI_Type *base,
                                             flexio_spi_slave_handle_t *handle,
                                             flexio_spi_slave_transfer_callback_t callback,
                                             void *userData)
```

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

```
status_t FLEXIO_SPI_SlaveTransferNonBlocking(FLEXIO_SPI_Type *base,
                                             flexio_spi_slave_handle_t *handle,
                                             flexio_spi_transfer_t *xfer)
```

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbort(FLEXIO_SPI_Type *base,
                                                flexio_spi_slave_handle_t *handle)
```

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCount(FLEXIO_SPI_Type *base,
                                                       flexio_spi_slave_handle_t *handle,
                                                       size_t *count)
```

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)`

FlexIO SPI slave IRQ handler function.

Parameters

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.
- `spiHandle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

`FSL_FLEXIO_SPI_DRIVER_VERSION`

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator `kStatus_FLEXIO_SPI_Busy`

FlexIO SPI is busy.

enumerator `kStatus_FLEXIO_SPI_Idle`

SPI is idle

enumerator `kStatus_FLEXIO_SPI_Error`

FlexIO SPI error.

enumerator `kStatus_FLEXIO_SPI_Timeout`

FlexIO SPI timeout polling status flags.

enum `_flexio_spi_clock_phase`

FlexIO SPI clock phase configuration.

Values:

enumerator `kFLEXIO_SPI_ClockPhaseFirstEdge`

First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.

enumerator `kFLEXIO_SPI_ClockPhaseSecondEdge`

First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

enum `_flexio_spi_shift_direction`

FlexIO SPI data shifter direction options.

Values:

enumerator `kFLEXIO_SPI_MsbFirst`

Data transfers start with most significant bit.

enumerator `kFLEXIO_SPI_LsbFirst`

Data transfers start with least significant bit.

enum `_flexio_spi_data_bitcount_mode`

FlexIO SPI data length mode options.

Values:

enumerator `kFLEXIO_SPI_8BitMode`

8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags
Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag
Receive buffer full flag.

enum _flexio_spi_dma_enable
Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable
Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable
Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable
All DMA request source

enum _flexio_spi_transfer_flags
Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb
FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb
FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb
FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb
FlexIO SPI 16-bit LSB first

```

enumerator kFLEXIO_SPI_32bitMsb
    FlexIO SPI 32-bit MSB first
enumerator kFLEXIO_SPI_32bitLsb
    FlexIO SPI 32-bit LSB first
enumerator kFLEXIO_SPI_csContinuous
    Enable the CS signal continuous mode
typedef enum flexio_spi_clock_phase flexio_spi_clock_phase_t
    FlexIO SPI clock phase configuration.
typedef enum flexio_spi_shift_direction flexio_spi_shift_direction_t
    FlexIO SPI data shifter direction options.
typedef enum flexio_spi_data_bitcount_mode flexio_spi_data_bitcount_mode_t
    FlexIO SPI data length mode options.
typedef struct flexio_spi_type FLEXIO_SPI_Type
    Define FlexIO SPI access structure typedef.
typedef struct flexio_spi_master_config flexio_spi_master_config_t
    Define FlexIO SPI master configuration structure.
typedef struct flexio_spi_slave_config flexio_spi_slave_config_t
    Define FlexIO SPI slave configuration structure.
typedef struct flexio_spi_transfer flexio_spi_transfer_t
    Define FlexIO SPI transfer structure.
typedef struct flexio_spi_master_handle flexio_spi_master_handle_t
    typedef for flexio_spi_master_handle_t in advance.
typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t
    Slave handle is the same with master handle.
typedef void (*flexio_spi_master_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_master_handle_t *handle, status_t status, void *userData)
    FlexIO SPI master callback for finished transmit.
typedef void (*flexio_spi_slave_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_slave_handle_t *handle, status_t status, void *userData)
    FlexIO SPI slave callback for finished transmit.
FLEXIO_SPI_DUMMYDATA
    FlexIO SPI dummy transfer data, the data is sent while txData is NULL.
SPI_RETRY_TIMES
    Retry times for waiting flag.
FLEXIO_SPI_XFER_DATA_FORMAT(flag)
    Get the transfer data format of width and bit order.
struct flexio_spi_type
    #include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

```

Public Members

```

FLEXIO_Type *flexioBase
    FlexIO base pointer.

```

uint8_t SDOPinIndex

Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex

Pin select for data input.

uint8_t SCKPinIndex

Pin select for clock.

uint8_t CSnPinIndex

Pin select for enable.

uint8_t shifterIndex[2]

Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]

Timer index used in FlexIO SPI.

struct `_flexio_spi_master_config`

`#include <fsl_flexio_spi.h>` Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster

Enable/disable FlexIO SPI master after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct `_flexio_spi_slave_config`

`#include <fsl_flexio_spi.h>` Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave

Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct *_flexio_spi_transfer*

#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

size_t dataSize

Transfer bytes.

uint8_t flags

FlexIO SPI control flag, MSB first or LSB first.

struct *_flexio_spi_master_handle*

#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData

Transfer buffer.

uint8_t *rxData

Receive buffer.

size_t transferSize

Total bytes to be transferred.

volatile size_t txRemainingBytes

Send data remaining in bytes.

volatile size_t rxRemainingBytes

Receive data remaining in bytes.

volatile uint32_t state

FlexIO SPI internal state.

uint8_t bytePerFrame

SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction

Shift direction.

flexio_spi_master_transfer_callback_t callback

FlexIO SPI callback.

`void *userData`
 Callback parameter.

`bool isCsContinuous`
 Is current transfer using CS continuous mode.

`uint32_t timer1Cfg`
 TIMER1 TIMCFG register value backup.

2.27 FlexIO UART Driver

`status_t FLEXIO_UART_Init(FLEXIO_UART_Type *base, const flexio_uart_config_t *userConfig, uint32_t srcClock_Hz)`

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by `FLEXIO_UART_GetDefaultConfig()`.

Example

```
FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxFPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
    .bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `userConfig` – Pointer to the `flexio_uart_config_t` structure.
- `srcClock_Hz` – FlexIO source clock in Hz.

Return values

- `kStatus_Success` – Configuration success.
- `kStatus_FLEXIO_UART_BaudrateNotSupport` – Baudrate is not supported for current clock source frequency.

`void FLEXIO_UART_Deinit(FLEXIO_UART_Type *base)`
 Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the `FLEXIO_UART_Init` to use the FlexIO UART module.

Parameters

- `base` – Pointer to `FLEXIO_UART_Type` structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;  
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_UART_TxDataRegEmptyFlag
 - kFLEXIO_UART_RxEmptyFlag
 - kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

```
static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(FLEXIO_UART_Type *base)
```

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.

Returns

FlexIO UART receive data register address.

```
static inline void FLEXIO_UART_EnableTxDMA(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the *kFLEXIO_UART_TxDataRegEmptyFlag* does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_EnableRxDMA(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting *kFLEXIO_UART_RxDataRegFullFlag* does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_Enable(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- enable – True to enable, false does not have any effect.

```
static inline void FLEXIO_UART_WriteByte(FLEXIO_UART_Type *base, const uint8_t *buffer)
```

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the *TxEmptyFlag* is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- buffer – The data bytes to send.

```
static inline void FLEXIO_UART_ReadByte(FLEXIO_UART_Type *base, uint8_t *buffer)
```

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the *RxFullFlag* is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.

- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

```
void FLEXIO_UART_TransferStartRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle, uint8_t *ringBuffer, size_t
                                         ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_ReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `ringBuffer` – Start address of ring buffer for background receiving. Pass `NULL` to disable the ring buffer.
- `ringBufferSize` – Size of the ring buffer.

```
void FLEXIO_UART_TransferStopRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.

```
status_t FLEXIO_UART_TransferSendNonBlocking(FLEXIO_UART_Type *base,
                                              flexio_uart_handle_t *handle,
                                              flexio_uart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the `kStatus_FLEXIO_UART_TxIdle` as status parameter.

Note: The `kStatus_FLEXIO_UART_TxIdle` is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `xfer` – FlexIO UART transfer structure. See `flexio_uart_transfer_t`.

Return values

- `kStatus_Success` – Successfully starts the data transmission.
- `kStatus_UART_TxBusy` – Previous transmission still not finished, data not written to the TX register.

```
void FLEXIO_UART_TransferAbortSend(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the `remainBytes` to find out how many bytes are still not sent out.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetSendCount(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `count` – Number of bytes sent so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

```
status_t FLEXIO_UART_TransferReceiveNonBlocking(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer, size_t *receivedBytes)
```

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to `xfer->data`. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the

xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See flexio_uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

```
void FLEXIO_UART_TransferAbortReceive(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetReceiveCount(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

```
void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)
```

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)
Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy
Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy
Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle
UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle
UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR
ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun
UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun
UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout
UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport
Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char

FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar
7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar
8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar
9-bit data characters

enum _flexio_uart_interrupt_enable

Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

```

    enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
        Receive buffer full interrupt enable.
enum _flexio_uart_status_flags
    Define FlexIO UART status mask.
    Values:
    enumerator kFLEXIO_UART_TxDataRegEmptyFlag
        Transmit buffer empty flag.
    enumerator kFLEXIO_UART_RxDataRegFullFlag
        Receive buffer full flag.
    enumerator kFLEXIO_UART_RxOverRunFlag
        Receive buffer over run flag.
typedef enum _flexio_uart_bit_count_per_char flexio_uart_bit_count_per_char_t
    FlexIO UART bit count per char.
typedef struct _flexio_uart_type FLEXIO_UART_Type
    Define FlexIO UART access structure typedef.
typedef struct _flexio_uart_config flexio_uart_config_t
    Define FlexIO UART user configuration structure.
typedef struct _flexio_uart_transfer flexio_uart_transfer_t
    Define FlexIO UART transfer structure.
typedef struct _flexio_uart_handle flexio_uart_handle_t
typedef void (*flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t
*handle, status_t status, void *userData)
    FlexIO UART transfer callback function.
UART_RETRY_TIMES
    Retry times for waiting flag.
struct _flexio_uart_type
    #include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members
FLEXIO_Type *flexioBase
    FlexIO base pointer.
uint8_t TxPinIndex
    Pin select for UART_Tx.
uint8_t RxPinIndex
    Pin select for UART_Rx.
uint8_t shifterIndex[2]
    Shifter index used in FlexIO UART.
uint8_t timerIndex[2]
    Timer index used in FlexIO UART.
struct _flexio_uart_config
    #include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

```

Public Members

bool enableUart

Enable/disable FlexIO UART TX & RX.

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

flexio_uart_bit_count_per_char_t bitCountPerChar

number of bits, 7/8/9 -bit

struct *_flexio_uart_transfer*

#include <fsl_flexio_uart.h> Define FlexIO UART transfer structure.

Public Members

size_t dataSize

Transfer size

struct *_flexio_uart_handle*

#include <fsl_flexio_uart.h> Define FLEXIO UART handle structure.

Public Members

const uint8_t *volatile txData

Address of remaining data to send.

volatile size_t txDataSize

Size of the remaining data to send.

uint8_t *volatile rxData

Address of remaining data to receive.

volatile size_t rxDataSize

Size of the remaining data to receive.

size_t txDataSizeAll

Total bytes to be sent.

size_t rxDataSizeAll

Total bytes to be received.

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

size_t rxRingBufferSize

Size of the ring buffer.

volatile uint16_t rxRingBufferHead

Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
 Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
 Callback function.

void *userData
 UART callback function parameter.

volatile uint8_t txState
 TX transfer state.

volatile uint8_t rxState
 RX transfer state

union __unnamed78__

Public Members

uint8_t *data
 The buffer of data to be transfer.

uint8_t *rxData
 The buffer to receive data.

const uint8_t *txData
 The buffer of data to be sent.

2.28 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION
 GPIO driver version.

enum _gpio_pin_direction
 GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput
 Set current pin as digital input

enumerator kGPIO_DigitalOutput
 Set current pin as digital output

enum _gpio_checker_attribute
 GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW
 User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW
 User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW
 User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO_IgnoreAttributeCheck
Ignores the attribute check

enum _gpio_interrupt_config
Configures the interrupt generation condition.

Values:

enumerator kGPIO_InterruptStatusFlagDisabled
Interrupt status flag is disabled.

enumerator kGPIO_DMARisingEdge
ISF flag and DMA request on rising edge.

enumerator kGPIO_DMAFallingEdge
ISF flag and DMA request on falling edge.

enumerator kGPIO_DMAEitherEdge
ISF flag and DMA request on either edge.

enumerator kGPIO_FlagRisingEdge
Flag sets on rising edge.

enumerator kGPIO_FlagFallingEdge
Flag sets on falling edge.

enumerator kGPIO_FlagEitherEdge
Flag sets on either edge.

enumerator kGPIO_InterruptLogicZero
Interrupt when logic zero.

enumerator kGPIO_InterruptRisingEdge
Interrupt on rising edge.

enumerator kGPIO_InterruptFallingEdge
Interrupt on falling edge.

enumerator kGPIO_InterruptEitherEdge
Interrupt on either edge.

enumerator kGPIO_InterruptLogicOne
Interrupt when logic one.

enumerator kGPIO_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kGPIO_ActiveLowTriggerOutputEnable

Enable active low-trigger output.

enum _gpio_interrupt_selection

Configures the selection of interrupt/DMA request/trigger output.

Values:

enumerator kGPIO_InterruptOutput0

Interrupt/DMA request/trigger output 0.

enumerator kGPIO_InterruptOutput1

Interrupt/DMA request/trigger output 1.

enum gpio_pin_interrupt_control_t

GPIO pin and interrupt control.

Values:

enumerator kGPIO_PinControlNonSecure

Pin Control Non-Secure.

enumerator kGPIO_InterruptControlNonSecure

Interrupt Control Non-Secure.

enumerator kGPIO_PinControlNonPrivilege

Pin Control Non-Privilege.

enumerator kGPIO_InterruptControlNonPrivilege

Interrupt Control Non-Privilege.

typedef enum _gpio_pin_direction gpio_pin_direction_t

GPIO direction definition.

typedef enum _gpio_checker_attribute gpio_checker_attribute_t

GPIO checker attribute.

typedef struct _gpio_pin_config gpio_pin_config_t

The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

typedef enum _gpio_interrupt_config gpio_interrupt_config_t

Configures the interrupt generation condition.

typedef enum _gpio_interrupt_selection gpio_interrupt_selection_t

Configures the selection of interrupt/DMA request/trigger output.

typedef struct _gpio_version_info gpio_version_info_t

GPIO version information.

GPIO_FIT_REG(value)

struct _gpio_pin_config

#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Public Members

gpio_pin_direction_t pinDirection

GPIO direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

struct __gpio_version_info

#include <fsl_gpio.h> GPIO version information.

Public Members

uint16_t feature

Feature Specification Number.

uint8_t minor

Minor Version Number.

uint8_t major

Major Version Number.

2.29 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PortDenit(GPIO_Type *base)

Denitalizes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalInput,
    0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalOutput,
    0,
}
```

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

```
void GPIO_GetVersionInfo(GPIO_Type *base, gpio_version_info_t *info)
```

Get GPIO version information.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- info – GPIO version information

```
static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, gpio_pin_interrupt_control_t mask)
```

lock or unlock secure privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

```
static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Enable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Disable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
```

Enable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
```

Disable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Enable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputEnable(GPIO_Type *base, uint32_t mask)
Enable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputDisable(GPIO_Type *base, uint32_t mask)
Disable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)
Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
 - 0: corresponding pin output low-logic level.
 - 1: corresponding pin output high-logic level.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)
Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)
```

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)
```

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)
```

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

```
static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin,  
                                             gpio_interrupt_config_t config)
```

Configures the gpio pin interrupt/DMA request.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- config – GPIO pin interrupt configuration.
 - kGPIO_InterruptStatusFlagDisabled: Interrupt/DMA request disabled.
 - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
 - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
 - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
 - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
 - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
 - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
 - kGPIO_InterruptLogicZero : Interrupt when logic zero.
 - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
 - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
 - kGPIO_InterruptEitherEdge : Interrupt on either edge.
 - kGPIO_InterruptLogicOne : Interrupt when logic one.

- kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
- kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

```
static inline void GPIO_SetPinInterruptChannel(GPIO_Type *base, uint32_t pin,  
                                             gpio_interrupt_selection_t selection)
```

Configures the gpio pin interrupt/DMA request/trigger output channel selection.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- selection – GPIO pin interrupt output selection.
 - kGPIO_InterruptOutput0: Interrupt/DMA request/trigger output 0.
 - kGPIO_InterruptOutput1 : Interrupt/DMA request/trigger output 1.

```
uint32_t GPIO_GpioGetInterruptFlags(GPIO_Type *base)
```

Read the GPIO interrupt status flags.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)

Returns

The current GPIO's interrupt status flag. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

```
uint32_t GPIO_GpioGetInterruptChannelFlags(GPIO_Type *base, uint32_t channel)
```

Read the GPIO interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

Returns

The current GPIO's interrupt status flag based on the selected interrupt channel. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

```
uint8_t GPIO_PinGetInterruptFlag(GPIO_Type *base, uint32_t pin)
```

Read individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on)
- pin – GPIO specific pin number.

Returns

The current selected pin's interrupt status flag.

```
void GPIO_GpioClearInterruptFlags(GPIO_Type *base, uint32_t mask)
```

Clears GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

void GPIO_GpioClearInterruptChannelFlags(GPIO_Type *base, uint32_t mask, uint32_t channel)
Clears GPIO pin interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro
- channel – ‘0’ means selete interrupt channel 0, ‘1’ means selete interrupt channel 1.

void GPIO_PinClearInterruptFlag(GPIO_Type *base, uint32_t pin)
Clear GPIO individual pin’s interrupt status flag.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO specific pin number.

static inline void GPIO_SetMultipleInterruptPinsConfig(GPIO_Type *base, uint32_t mask, *gpio_interrupt_config_t* config)

Sets the GPIO interrupt configuration in PCR register for multiple pins.

Parameters

- base – GPIO peripheral base pointer.
- mask – GPIO pin number macro.
- config – GPIO pin interrupt configuration.
 - kGPIO_InterruptStatusFlagDisabled: Interrupt disabled.
 - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
 - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
 - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
 - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
 - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
 - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
 - kGPIO_InterruptLogicZero : Interrupt when logic zero.
 - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
 - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
 - kGPIO_InterruptEitherEdge : Interrupt on either edge.
 - kGPIO_InterruptLogicOne : Interrupt when logic one.
 - kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
 - kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

void GPIO_CheckAttributeBytes(GPIO_Type *base, *gpio_checker_attribute_t* attribute)

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If

the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

2.30 I3C: I3C Driver

FSL_I3C_DRIVER_VERSION

I3C driver version.

I3C status return codes.

Values:

enumerator kStatus_I3C_Busy

The master is already performing a transfer.

enumerator kStatus_I3C_Idle

The slave driver is idle.

enumerator kStatus_I3C_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus_I3C_WriteAbort

The slave device sent a NAK in response to a write.

enumerator kStatus_I3C_Term

The master terminates slave read.

enumerator kStatus_I3C_HdrParityError

Parity error from DDR read.

enumerator kStatus_I3C_CrcError

CRC error from DDR read.

enumerator kStatus_I3C_ReadFifoError

Read from M/SRDATA register when FIFO empty.

enumerator kStatus_I3C_WriteFifoError

Write to M/SWDATA register when FIFO full.

enumerator kStatus_I3C_MsgError

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus_I3C_InvalidReq

Invalid use of request.

enumerator kStatus_I3C_Timeout

The module has stalled too long in a frame.

enumerator kStatus_I3C_SlaveCountExceed

The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

enumerator kStatus_I3C_IBIWon

The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus_I3C_OverrunError
Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus_I3C_UnderrunError
Slave internal to-bus buffer/FIFO underrun

enumerator kStatus_I3C_UnderrunNak
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus_I3C_InvalidStart
Slave invalid start flag

enumerator kStatus_I3C_SdrParityError
SDR parity error

enumerator kStatus_I3C_S0S1Error
S0 or S1 error

enum _i3c_hdr_mode
I3C HDR modes.

Values:

enumerator kI3C_HDRModeNone

enumerator kI3C_HDRModeDDR

enumerator kI3C_HDRModeTSP

enumerator kI3C_HDRModeTSL

typedef enum _i3c_hdr_mode i3c_hdr_mode_t
I3C HDR modes.

typedef struct _i3c_device_info i3c_device_info_t
I3C device information.

I3C_RETRY_TIMES
Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C_MAX_DEVCNT

I3C_IBI_BUFF_SIZE

struct _i3c_device_info
#include <fsl_i3c.h> I3C device information.

Public Members

uint8_t dynamicAddr
Device dynamic address.

uint8_t staticAddr
Static address.

uint8_t dcr
Device characteristics register information.

uint8_t bcr
Bus characteristics register information.

`uint16_t` vendorID
Device vendor ID(manufacture ID).

`uint32_t` partNumber
Device part number info

`uint16_t` maxReadLength
Maximum read length.

`uint16_t` maxWriteLength
Maximum write length.

`uint8_t` hdrMode
Support hdr mode, could be OR logic in `i3c_hdr_mode`.

2.31 I3C Common Driver

`typedef struct i3c_config i3c_config_t`

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

`uint32_t` `I3C_GetInstance(I3C_Type *base)`
Get which instance current I3C is used.

Parameters

- `base` – The I3C peripheral base address.

`void` `I3C_GetDefaultConfig(i3c_config_t *config)`

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster           = kI3C_MasterCapable;
config->disableTimeout         = false;
config->hKeep                   = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop     = true;
config->enableOpenDrainHigh     = true;
config->baudRate_Hz.i2cBaud     = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress    = 0x0AU;
config->slowClock_Hz           = 1000000U;
config->enableSlave             = true;
config->vendorID                = 0x11BU;
config->enableRandomPart        = false;
config->partNumber              = 0;
config->dcr                     = 0;
config->bcr = 0;
config->hdrMode                 = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest           = false;
config->ignoreS0S1Error        = false;
config->offline                 = false;
config->matchSlaveStartStop    = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with `I3C_Init()`.

Parameters

- `config` – **[out]** User provided configuration structure for default values. Refer to `i3c_config_t`.

```
void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)
```

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- `base` – The I3C peripheral base address.
- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
struct _i3c_config
```

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

```
i3c_master_enable_t enableMaster
```

Enable master mode.

```
bool disableTimeout
```

Whether to disable timeout to prevent the ERRWARN.

```
i3c_master_hkeep_t hKeep
```

High keeper mode setting.

```
bool enableOpenDrainStop
```

Whether to emit open-drain speed STOP.

```
bool enableOpenDrainHigh
```

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

```
i3c_baudrate_hz_t baudRate_Hz
```

Desired baud rate settings.

```
uint8_t masterDynamicAddress
```

Main master dynamic address configuration.

```
uint32_t maxWriteLength
```

Maximum write length.

```
uint32_t maxReadLength
```

Maximum read length.

<code>bool enableSlave</code>	Whether to enable slave.
<code>uint8_t staticAddr</code>	Static address.
<code>uint16_t vendorID</code>	Device vendor ID(manufacture ID).
<code>uint32_t partNumber</code>	Device part number info
<code>uint8_t dcr</code>	Device characteristics register information.
<code>uint8_t bcr</code>	Bus characteristics register information.
<code>uint8_t hdrMode</code>	Support hdr mode, could be OR logic in enumeration: <code>i3c_hdr_mode_t</code> .
<code>bool nakAllRequest</code>	Whether to reply NAK to all requests except broadcast CCC.
<code>bool ignoreS0S1Error</code>	Whether to ignore S0/S1 error in SDR mode.
<code>bool offline</code>	Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.
<code>bool matchSlaveStartStop</code>	Whether to assert start/stop status only the time slave is addressed.

2.32 I3C Master Driver

`void I3C_MasterGetDefaultConfig(i3c_master_config_t *masterConfig)`

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster      = kI3C_MasterOn;
masterConfig->disableTimeout    = false;
masterConfig->hKeep             = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop = true;
masterConfig->enableOpenDrainHigh = true;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busType           = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I3C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_master_config_t`.

```
void I3C_MasterInit(I3C_Type *base, const i3c_master_config_t *masterConfig, uint32_t
    sourceClock_Hz)
```

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The I3C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I3C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void I3C_MasterDeinit(I3C_Type *base)
```

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)
```

```
status_t I3C_CheckForBusyBus(I3C_Type *base)
```

```
static inline void I3C_MasterEnable(I3C_Type *base, i3c_master_enable_t enable)
```

Set I3C module master mode.

Parameters

- `base` – The I3C peripheral base address.
- `enable` – Enable master mode.

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t
    slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- `base` – The I3C peripheral base address.

`static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)`

Enable/Disable Slave.

Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

`static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)`

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_master_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C master status flag state.

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`
- `kI3C_MasterSlave2MasterFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_master_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_master_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master error status flag state.

See also:

`_i3c_master_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_master_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
i3c_master_state_t I3C_MasterGetState(I3C_Type *base)
```

Gets the I3C master state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C master state.

```
static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)
```

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

```
static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

```
i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)
```

Gets the I3C slave state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

```
status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
static inline void I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Enables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Disables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_MasterGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C master interrupt requests.

Parameters

- `base` – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C master interrupt requests.

Parameters

- `base` – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx,
                                       uint32_t width)
```

Enables or disables I3C master DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

```
static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

```
static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t
                                       width)
```

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.

- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.
- `width` – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_MasterSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,  
                                           i3c_rx_trigger_level_t rxLvl, bool flushTx, bool  
                                           flushRx)
```

Sets the watermarks for I3C master FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C master FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,  
                                           i3c_rx_trigger_level_t rxLvl, bool flushTx, bool  
                                           flushRx)
```

Sets the watermarks for I3C slave FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txCount* – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- *rxCount* – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
void I3C_MasterSetBaudRate(I3C_Type *base, const i3c_baudrate_hz_t *baudRate_Hz, uint32_t sourceClock_Hz)
```

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

- *base* – The I3C peripheral base address.
- *baudRate_Hz* – Pointer to structure of requested bus frequency in Hertz.
- *sourceClock_Hz* – I3C functional clock frequency in Hertz.

```
static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- *base* – The I3C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t I3C_MasterStartWithRxSize(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir, uint8_t rxSize)
```

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *a* address parameter. Note that this function

does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

`status_t` I3C_MasterStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

`status_t` I3C_MasterRepeatedStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the `rxSize = 2`, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the `rxSize` configuration.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
static inline status_t I3C_MasterRepeatedStart(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
```

Performs a polling send transfer on the I2C/I3C bus.

Sends up to `txSize` number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_I3C_Nak`.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.

- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`status_t I3C_MasterReceive(I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)`

Performs a polling receive transfer on the I2C/I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`status_t I3C_MasterStop(I3C_Type *base)`

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The I3C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

```
void I3C_MasterEmitRequest(I3C_Type *base, i3c_bus_request_t masterReq)
```

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- masterReq – I3C master request of type `i3c_bus_request_t`

```
static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, i3c_ibi_response_t ibiResponse)
```

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- ibiResponse – I3C master emit IBI response of type `i3c_ibi_response_t`

```
void I3C_MasterRegisterIBI(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)
```

I3C master register IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to ibi rule description of type `i3c_register_ibi_addr_t`

```
void I3C_MasterGetIBIRules(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)
```

I3C master get IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to store the read out ibi rule description.

```
i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)
```

I3C master get IBI Type.

Parameters

- base – The I3C peripheral base address.

Return values

`i3c_ibi_type_t` – Type of `i3c_ibi_type_t`.

```
static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)
```

I3C master get IBI Address.

Parameters

- base – The I3C peripheral base address.

Return values

The – 8-bit IBI address.

```
status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t
count, i3c_master_daa_baudrate_t
*daaBaudRate)
```

Performs a DAA in the i3c bus with specified temporary baud rate.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list.

- `daaBaudRate` – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
static inline status_t I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t count)
```

Performs a DAA in the i3c bus.

Parameters

- `base` – The I3C peripheral base address.
- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)
```

Get device information list after DAA process is done.

Parameters

- `base` – The I3C peripheral base address.
- `count` – **[out]** The pointer to store the available device count.

Returns

Pointer to the `i3c_device_info_t` array.

```
void I3C_MasterClearDeviceCount(I3C_Type *base)
```

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterTransferBlocking(I3C_Type *base, i3c_master_transfer_t *transfer)
```

Performs a master polling transfer on the I2C/I3C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The I3C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_I3C_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_IBIWon` – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`void I3C_MasterTransferCreateHandle(I3C_Type *base, i3c_master_handle_t *handle, const i3c_master_transfer_callback_t *callback, void *userData)`

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_MasterTransferNonBlocking(I3C_Type *base, i3c_master_handle_t *handle, i3c_master_transfer_t *transfer)`

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

`status_t I3C_MasterTransferGetCount(I3C_Type *base, i3c_master_handle_t *handle, size_t *count)`

Returns number of bytes transferred so far.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`void I3C_MasterTransferAbort(I3C_Type *base, i3c_master_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

Return values

- `kStatus_Success` – A transaction was successfully aborted.
- `kStatus_I3C_Idle` – There is not a non-blocking transaction currently in progress.

`void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to the I3C master driver handle.

`enum _i3c_master_flags`

I3C master peripheral flags.

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`
- `kI3C_MasterSlave2MasterFlag`

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator `kI3C_MasterBetweenFlag`
Between messages/DAAs flag

enumerator `kI3C_MasterNackDetectFlag`
NACK detected flag

enumerator `kI3C_MasterSlaveStartFlag`
Slave request start flag

enumerator `kI3C_MasterControlDoneFlag`
Master request complete flag

enumerator `kI3C_MasterCompleteFlag`
Transfer complete flag

enumerator `kI3C_MasterRxReadyFlag`
Rx data ready in Rx buffer flag

enumerator `kI3C_MasterTxReadyFlag`
Tx buffer ready for Tx data flag

enumerator `kI3C_MasterArbitrationWonFlag`
Header address won arbitration flag

enumerator kI3C_MasterErrorFlag

Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag

Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum _i3c_master_error_flags

I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag

Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag

Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag

Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag

CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag

Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag

Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag

Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag

The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags

All error flags

enum _i3c_master_state

I3C working master state.

Values:

enumerator kI3C_MasterStateIdle

Bus stopped.

enumerator kI3C_MasterStateSlvReq

Bus stopped but slave holding SDA low.

enumerator kI3C_MasterStateMsgSdr

In SDR Message mode from using MWMSG_SDR.

enumerator kI3C_MasterStateNormAct

In normal active SDR mode.

enumerator kI3C_MasterStateDdr

In DDR Message mode.

enumerator kI3C_MasterStateDaa

In ENTDAAs mode.

enumerator kI3C_MasterStateIbiAck

Waiting on IBI ACK/NACK decision.

enumerator kI3C_MasterStateIbiRcv

Receiving IBI.

enum _i3c_master_enable

I3C master enable configuration.

Values:

enumerator kI3C_MasterOff

Master off.

enumerator kI3C_MasterOn

Master on.

enumerator kI3C_MasterCapable

Master capable.

enum _i3c_master_hkeep

I3C high keeper configuration.

Values:

enumerator kI3C_MasterHighKeeperNone

Use PUR to hold SCL high.

enumerator kI3C_MasterHighKeeperWiredIn

Use pin_HK controls.

enumerator kI3C_MasterPassiveSDA

Hi-Z for Bus Free and hold SDA.

enumerator kI3C_MasterPassiveSDASCL

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum _i3c_bus_request

Emits the requested operation when doing in pieces vs. by message.

Values:

enumerator kI3C_RequestNone

No request.

enumerator kI3C_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA

Process DAA.

enumerator kI3C_RequestForceExit

Request to force exit.

enumerator kI3C_RequestAutoIbi

Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type

Bus type with EmitStartAddr.

Values:

enumerator kI3C_TypeI3CSdr

SDR mode of I3C.

enumerator kI3C_TypeI2C

Standard i2c protocol.

enumerator kI3C_TypeI3CDdr

HDR-DDR mode of I3C.

enum _i3c_ibi_response

IBI response.

Values:

enumerator kI3C_IbiRespAck

ACK with no mandatory byte.

enumerator kI3C_IbiRespNack

NACK.

enumerator kI3C_IbiRespAckMandatory

ACK with mandatory byte.

enumerator kI3C_IbiRespManual

Reserved.

enum _i3c_ibi_type

IBI type.

Values:

enumerator kI3C_IbiNormal

In-band interrupt.

enumerator kI3C_IbiHotJoin

slave hot join.

enumerator kI3C_IbiMasterRequest

slave master ship request.

enum _i3c_ibi_state

IBI state.

Values:

enumerator kI3C_IbiReady

In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed

In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending

In-band interrupt Ack/Nack pending for decision.

enum `_i3c_direction`

Direction of master and slave transfers.

Values:

enumerator `kI3C_Write`

Master transmit.

enumerator `kI3C_Read`

Master receive.

enum `_i3c_tx_trigger_level`

Watermark of TX int/dma trigger level.

Values:

enumerator `kI3C_TxTriggerOnEmpty`

Trigger on empty.

enumerator `kI3C_TxTriggerUntilOneQuarterOrLess`

Trigger on 1/4 full or less.

enumerator `kI3C_TxTriggerUntilOneHalfOrLess`

Trigger on 1/2 full or less.

enumerator `kI3C_TxTriggerUntilOneLessThanFull`

Trigger on 1 less than full or less.

enum `_i3c_rx_trigger_level`

Watermark of RX int/dma trigger level.

Values:

enumerator `kI3C_RxTriggerOnNotEmpty`

Trigger on not empty.

enumerator `kI3C_RxTriggerUntilOneQuarterOrMore`

Trigger on 1/4 full or more.

enumerator `kI3C_RxTriggerUntilOneHalfOrMore`

Trigger on 1/2 full or more.

enumerator `kI3C_RxTriggerUntilThreeQuarterOrMore`

Trigger on 3/4 full or more.

enum `_i3c_rx_term_ops`

I3C master read termination operations.

Values:

enumerator `kI3C_RxTermDisable`

Master doesn't terminate read, used for CCC transfer.

enumerator `kI3C_RxAutoTerm`

Master auto terminate read after receiving specified bytes(<=255).

enumerator `kI3C_RxTermLastByte`

Master terminates read at any time after START, no length limitation.

enum `_i3c_start_scl_delay`

I3C start SCL delay options.

Values:

enumerator `kI3C_NoDelay`

No delay.

enumerator `kI3C_IncreaseSclHalfPeriod`

Increases SCL clock period by 1/2.

enumerator `kI3C_IncreaseSclOnePeriod`

Increases SCL clock period by 1.

enumerator `kI3C_IncreaseSclOneAndHalfPeriod`

Increases SCL clock period by 1 1/2

enum `_i3c_master_transfer_flags`

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_i3c_master_transfer::flags` field.

Values:

enumerator `kI3C_TransferDefaultFlag`

Transfer starts with a start signal, stops with a stop signal.

enumerator `kI3C_TransferNoStartFlag`

Don't send a start condition, address, and sub address

enumerator `kI3C_TransferRepeatedStartFlag`

Send a repeated start condition

enumerator `kI3C_TransferNoStopFlag`

Don't send a stop condition.

enumerator `kI3C_TransferWordsFlag`

Transfer in words, else transfer in bytes.

enumerator `kI3C_TransferDisableRxTermFlag`

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator `kI3C_TransferRxAutoTermFlag`

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in `I3C_MasterReceive`.

enumerator `kI3C_TransferStartWithBroadcastAddr`

Start transfer with 0x7E, then read/write data with device address.

typedef enum `_i3c_master_state` `i3c_master_state_t`

I3C working master state.

typedef enum `_i3c_master_enable` `i3c_master_enable_t`

I3C master enable configuration.

typedef enum `_i3c_master_hkeep` `i3c_master_hkeep_t`

I3C high keeper configuration.

typedef enum `_i3c_bus_request` `i3c_bus_request_t`

Emits the requested operation when doing in pieces vs. by message.

typedef enum `_i3c_bus_type` `i3c_bus_type_t`

Bus type with `EmitStartAddr`.

```

typedef enum _i3c_ibi_response i3c_ibi_response_t
    IBI response.

typedef enum _i3c_ibi_type i3c_ibi_type_t
    IBI type.

typedef enum _i3c_ibi_state i3c_ibi_state_t
    IBI state.

typedef enum _i3c_direction i3c_direction_t
    Direction of master and slave transfers.

typedef enum _i3c_tx_trigger_level i3c_tx_trigger_level_t
    Watermark of TX int/dma trigger level.

typedef enum _i3c_rx_trigger_level i3c_rx_trigger_level_t
    Watermark of RX int/dma trigger level.

typedef enum _i3c_rx_term_ops i3c_rx_term_ops_t
    I3C master read termination operations.

typedef enum _i3c_start_scl_delay i3c_start_scl_delay_t
    I3C start SCL delay options.

typedef struct _i3c_register_ibi_addr i3c_register_ibi_addr_t
    Structure with setting master IBI rules and slave registry.

typedef struct _i3c_baudrate i3c_baudrate_hz_t
    Structure with I3C baudrate settings.

typedef struct _i3c_master_daa_baudrate i3c_master_daa_baudrate_t
    I3C DAA baud rate configuration.

typedef struct _i3c_master_config i3c_master_config_t
    Structure with settings to initialize the I3C master module.

    This structure holds configuration settings for the I3C peripheral. To initialize this structure
    to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer
    to your configuration structure instance.

    The configuration structure can be made constant so it resides in flash.

typedef struct _i3c_master_transfer i3c_master_transfer_t

typedef struct _i3c_master_handle i3c_master_handle_t

typedef struct _i3c_master_transfer_callback i3c_master_transfer_callback_t
    i3c master callback functions.

typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
    Typedef for master interrupt handler.

struct _i3c_register_ibi_addr
    #include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.

```

Public Members

```

uint8_t address[5]
    Address array for registry.

bool i3cFastStart
    Allow the START header to run as push-pull speed if all dynamic addresses take MSB
    0.

```

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct _i3c_baudrate

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

uint32_t i2cBaud

Desired I2C baud rate in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct _i3c_master_daa_baudrate

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

uint32_t sourceClock_Hz

FCLK, function clock in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct _i3c_master_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

```
struct _i3c_master_transfer_callback
#include <fsl_i3c.h> i3c master callback functions.
```

Public Members

```
void (*slave2Master)(I3C_Type *base, void *userData)
    Transfer complete callback
```

```
void (*ibiCallback)(I3C_Type *base, i3c_master_handle_t *handle, i3c_ibi_type_t ibiType,
i3c_ibi_state_t ibiState)
    IBI event callback
```

```
void (*transferComplete)(I3C_Type *base, i3c_master_handle_t *handle, status_t
completionStatus, void *userData)
    Transfer complete callback
```

```
struct _i3c_master_transfer
#include <fsl_i3c.h> Non-blocking transfer descriptor structure.
```

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Public Members

```
uint32_t flags
```

Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options. Set to 0 or `kI3C_TransferDefaultFlag` for normal transfers.

```
uint8_t slaveAddress
```

The 7-bit slave address.

```
i3c_direction_t direction
```

Either `kI3C_Read` or `kI3C_Write`.

```
uint32_t subaddress
```

Sub address. Transferred MSB first.

```
size_t subaddressSize
```

Length of sub address to send in bytes. Maximum size is 4 bytes.

```
void *data
```

Pointer to data to transfer.

```
size_t dataSize
```

Number of bytes to transfer.

```
i3c_bus_type_t busType
```

bus type.

```
i3c_ibi_response_t ibiResponse
```

ibi response during transfer.

```
struct _i3c_master_handle
#include <fsl_i3c.h> Driver handle for master non-blocking APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

- `uint8_t` state
Transfer state machine current state.
- `uint32_t` remainingBytes
Remaining byte count in current state.
- `i3c_rx_term_ops_t` rxTermOps
Read termination operation.
- `i3c_master_transfer_t` transfer
Copy of the current transfer info.
- `uint8_t` ibiAddress
Slave address which request IBI.
- `uint8_t *` ibiBuff
Pointer to IBI buffer to keep ibi bytes.
- `size_t` ibiPayloadSize
IBI payload size.
- `i3c_ibi_type_t` ibiType
IBI type.
- `i3c_master_transfer_callback_t` callback
Callback functions pointer.
- `void *` userData
Application data passed to callback.

2.33 I3C Slave Driver

`void` I3C_SlaveGetDefaultConfig(`i3c_slave_config_t *`slaveConfig)

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableSlave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- slaveConfig – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

`void` I3C_SlaveInit(`I3C_Type *`base, `const i3c_slave_config_t *`slaveConfig, `uint32_t` slowClock_Hz)

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.

- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- `base` – The I3C peripheral base address.

`static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)`

Enable/Disable Slave.

Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

`static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)`

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.

- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

`static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)`

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

`i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)`

Gets the I3C slave state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

`status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)`

`static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`

- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)
```

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

Parameters

- `base` – The I3C peripheral base address.
- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.
- `width` – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

Parameters

- `base` – The I3C peripheral base address.
- `width` – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`void I3C_SlaveTransferCreateHandle(I3C_Type *base, i3c_slave_handle_t *handle, i3c_slave_transfer_callback_t callback, void *userData)`

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i3c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kI3C_SlaveTransmitEvent* and *kI3C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kI3C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.
- *eventMask* – Bit mask formed by OR'ing together *i3c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kI3C_SlaveAllEvents* to enable all events.

Return values

- *kStatus_Success* – Slave transfers were successfully started.
- *kStatus_I3C_Busy* – Slave transfers have already been started on this handle.

status_t I3C_SlaveTransferGetCount(I3C_Type *base, *i3c_slave_handle_t* *handle, *size_t* *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to *i2c_slave_handle_t* structure.
- *count* – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- *kStatus_Success* –
- *kStatus_NoTransferInProgress* –

void I3C_SlaveTransferAbort(I3C_Type *base, *i3c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.

Return values

- *kStatus_Success* –
- *kStatus_I3C_Idle* –

void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

enum `_i3c_slave_flags`

I3C slave peripheral flags.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator `kI3C_SlaveNotStopFlag`

Slave status not stop flag

enumerator `kI3C_SlaveMessageFlag`

Slave status message, indicating slave is listening to the bus traffic or responding

enumerator `kI3C_SlaveRequiredReadFlag`

Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator `kI3C_SlaveRequiredWriteFlag`

Slave status request write, master is doing SDR write to slave, except slave in ENTDAAMode

enumerator kI3C_SlaveBusDAAFlag
I3C bus is in ENTDAAs mode

enumerator kI3C_SlaveBusHDRModeFlag
I3C bus is in HDR mode

enumerator kI3C_SlaveBusStartFlag
Start/Re-start event is seen since the bus was last cleared

enumerator kI3C_SlaveMatchedFlag
Slave address(dynamic/static) matched since last cleared

enumerator kI3C_SlaveBusStopFlag
Stop event is seen since the bus was last cleared

enumerator kI3C_SlaveRxReadyFlag
Rx data ready in rx buffer flag

enumerator kI3C_SlaveTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_SlaveDynamicAddrChangedFlag
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C_SlaveReceivedCCCFlag
Slave received Common command code

enumerator kI3C_SlaveErrorFlag
Error occurred flag

enumerator kI3C_SlaveHDRCommandMatchFlag
High data rate command match

enumerator kI3C_SlaveCCCHandledFlag
Slave received Common command code is handled by I3C module

enumerator kI3C_SlaveEventSentFlag
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C_SlaveIbiDisableFlag
Slave in band interrupt is disabled.

enumerator kI3C_SlaveMasterRequestDisabledFlag
Slave master request is disabled.

enumerator kI3C_SlaveHotJoinDisabledFlag
Slave Hot-Join is disabled.

enumerator kI3C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C_SlaveAllIrqFlags

enum _i3c_slave_error_flags
I3C slave error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveErrorOverrunFlag
Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C_SlaveErrorUnderrunFlag
Slave internal to-bus buffer/FIFO underrun

enumerator kI3C_SlaveErrorUnderrunNakFlag
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C_SlaveErrorTermFlag
Terminate error from master

enumerator kI3C_SlaveErrorInvalidStartFlag
Slave invalid start flag

enumerator kI3C_SlaveErrorSdrParityFlag
SDR parity error

enumerator kI3C_SlaveErrorHdrParityFlag
HDR parity error

enumerator kI3C_SlaveErrorHdrCRCFlag
HDR-DDR CRC error

enumerator kI3C_SlaveErrorS0S1Flag
S0 or S1 error

enumerator kI3C_SlaveErrorOverreadFlag
Over-read error

enumerator kI3C_SlaveErrorOverwriteFlag
Over-write error

enum _i3c_slave_event

I3C slave.event.

Values:

enumerator kI3C_SlaveEventNormal
Normal mode.

enumerator kI3C_SlaveEventIBI
In band interrupt event.

enumerator kI3C_SlaveEventMasterReq
Master request event.

enumerator kI3C_SlaveEventHotJoinReq
Hot-join event.

enum _i3c_slave_activity_state

I3C slave.activity state.

Values:

enumerator kI3C_SlaveNoLatency
Normal bus operation

enumerator kI3C_SlaveLatency1Ms
1ms of latency.

enumerator kI3C_SlaveLatency100Ms
100ms of latency.

enumerator kI3C_SlaveLatency10S
10s latency.

enum `_i3c_slave_transfer_event`

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator `kI3C_SlaveAddressMatchEvent`

Received the slave address after a start or repeated start.

enumerator `kI3C_SlaveTransmitEvent`

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator `kI3C_SlaveReceiveEvent`

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `kI3C_SlaveRequiredTransmitEvent`

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `kI3C_SlaveStartEvent`

A start/repeated start was detected.

enumerator `kI3C_SlaveHDRCommandMatchEvent`

Slave Match HDR Command.

enumerator `kI3C_SlaveCompletionEvent`

A stop was detected, completing the transfer.

enumerator `kI3C_SlaveRequestSentEvent`

Slave request event sent.

enumerator `kI3C_SlaveReceivedCCCEvent`

Slave received CCC event, need to handle by application.

enumerator `kI3C_SlaveAllEvents`

Bit mask of all available events.

typedef enum `_i3c_slave_event` `i3c_slave_event_t`

I3C slave.event.

typedef enum `_i3c_slave_activity_state` `i3c_slave_activity_state_t`

I3C slave.activity state.

typedef struct `_i3c_slave_config` `i3c_slave_config_t`

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _i3c_slave_transfer_event i3c_slave_transfer_event_t
```

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

```
typedef struct _i3c_slave_transfer i3c_slave_transfer_t
```

I3C slave transfer structure.

```
typedef struct _i3c_slave_handle i3c_slave_handle_t
```

```
typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `I3C_SlaveSetCallback()` function after you have created a handle.

Param base

Base address for the I3C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
```

Typedef for slave interrupt handler.

```
struct _i3c_slave_config
```

#include <fsl_i3c.h> Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint8_t hdrMode

Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

uint32_t maxWriteLength

Maximum write length.

uint32_t maxReadLength

Maximum read length.

struct _i3c_slave_transfer

#include <fsl_i3c.h> I3C slave transfer structure.

Public Members

uint32_t event

Reason the callback is being invoked.

uint8_t *txData

Transfer buffer

size_t txDataSize

Transfer size

uint8_t *rxData

Transfer buffer

size_t rxDataSize

Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for I3C_SlaveCompletionEvent.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct _i3c_slave_handle

#include <fsl_i3c.h> I3C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

i3c_slave_transfer_t transfer
I3C slave transfer copy.

bool isBusy
Whether transfer is busy.

bool wasTransmit
Whether the last transfer was a transmit.

uint32_t eventMask
Mask of enabled events.

uint32_t transferredCount
Count of bytes transferred.

i3c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

size_t txFifoSize
Tx Fifo size

2.34 IMU: Inter CPU Messaging Unit

status_t IMU_Init(imu_link_t link)

Initializes the IMU module.

This function sets IMU to initialized state, including:

- Flush the send FIFO.
- Unlock the send FIFO.
- Set the water mark to (IMU_MAX_MSG_FIFO_WATER_MARK)
- Flush the receive FIFO.

If IMU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout if flushing the receive FIFO takes too long.

Parameters

- link – IMU link.

Return values

- kStatus_Success – The IMU was initialized successfully.
- kStatus_InvalidArgument – Invalid link parameter.
- kStatus_Timeout – Timeout occurred while flushing the receive FIFO.

Returns

status_t

```
void IMU_Deinit(imu_link_t link)
```

De-initializes the IMU module.

Parameters

- link – IMU link.

```
static inline void IMU_WriteMsg(imu_link_t link, uint32_t msg)
```

Write one message to TX FIFO.

This function writes message to the TX FIFO, user need to make sure there is empty space in the TX FIFO, and TX FIFO not locked before calling this function.

Parameters

- link – IMU link.
- msg – The message to send.

```
static inline uint32_t IMU_ReadMsg(imu_link_t link)
```

Read one message from RX FIFO.

User need to make sure there is available message in the RX FIFO.

Parameters

- link – IMU link.

Returns

The message.

```
int32_t IMU_SendMsgsBlocking(imu_link_t link, const uint32_t *msgs, int32_t msgCount, bool lockSendFifo)
```

Blocking to send messages.

This function blocks until all messages have been filled to TX FIFO.

- If the TX FIFO is locked, this function returns `IMU_ERR_TX_FIFO_LOCKED`.
- If TX FIFO not locked, this function waits the available empty slot in TX FIFO, and fills the message to TX FIFO.
- To lock TX FIFO after filling all messages, set `lockSendFifo` to true.

If `IMU_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `IMU_ERR_TIMEOUT` if waiting for FIFO space takes too long.

Parameters

- link – IMU link.
- msgs – The messages to send.
- msgCount – Message count, one message is a 32-bit word.
- lockSendFifo – If set to true, the TX FIFO is locked after all messages filled to TX FIFO.

Returns

If TX FIFO is locked, this function returns `IMU_ERR_TX_FIFO_LOCKED`. If a timeout occurs while waiting for FIFO space, it returns `IMU_ERR_TIMEOUT`. Otherwise, this function returns the actual message count sent out, which equals `msgCount` because this function is blocking until all messages have been filled into TX FIFO or a timeout occurs.

```
int32_t IMU_TrySendMsgs(imu_link_t link, const uint32_t *msgs, int32_t msgCount, bool
                      lockSendFifo)
```

Try to send messages.

This function is similar with `IMU_SendMsgsBlocking`, the difference is, this function tries to send as many as possible, if there is not enough empty slot in TX FIFO, this function fills messages to available empty slots and returns how many messages have been filled.

- If the TX FIFO is locked, this function returns `IMU_ERR_TX_FIFO_LOCKED`.
- If TX FIFO not locked, this function fills messages to TX FIFO empty slot, and returns how many messages have been filled.
- If `lockSendFifo` is set to true, TX FIFO is locked after all messages have been filled to TX FIFO. In other word, TX FIFO is locked if the function return value equals `msgCount`, when `lockSendFifo` set to true.

Parameters

- `link` – IMU link.
- `msgs` – The messages to send.
- `msgCount` – Message count, one message is a 32-bit word.
- `lockSendFifo` – If set to true, the TX FIFO is locked after all messages filled to TX FIFO.

Returns

If TX FIFO is locked, this function returns `IMU_ERR_TX_FIFO_LOCKED`, otherwise, this function returns the actual message count sent out.

```
int32_t IMU_TryReceiveMsgs(imu_link_t link, uint32_t *msgs, int32_t desiredMsgCount, bool
                          *needAckLock)
```

Try to receive messages.

This function tries to read messages from RX FIFO. It reads the messages already exists in RX FIFO and returns the actual read count.

- If the RX FIFO has enough messages, this function reads the messages and returns.
- If the RX FIFO does not have enough messages, this function the messages in RX FIFO and returns the actual read count.
- During message reading, if RX FIFO is empty and locked, in this case peer CPU will not send message until current CPU send lock ack message. Then this function returns the message count actually received, and sets `needAckLock` to true to inform upper layer.

Parameters

- `link` – IMU link.
- `msgs` – The buffer to read messages.
- `desiredMsgCount` – Desired read count, one message is a 32-bit word.
- `needAckLock` – Upper layer should always check this value. When this is set to true by this function, upper layer should send lock ack message to peer CPU.

Returns

Count of messages actually received.

```
int32_t IMU_ReceiveMsgsBlocking(imu_link_t link, uint32_t *msgs, int32_t desiredMsgCount,
                               bool *needAckLock)
```

Blocking to receive messages.

This function blocks until all desired messages have been received or the RX FIFO is locked.

- If the RX FIFO has enough messages, this function reads the messages and returns.
- If the RX FIFO does not have enough messages, this function waits for the new messages.
- During message reading, if RX FIFO is empty and locked, in this case peer CPU will not send message until current CPU send lock ack message. Then this function returns the message count actually received, and sets `needAckLock` to true to inform upper layer.

Parameters

- `link` – IMU link.
- `msgs` – The buffer to read messages.
- `desiredMsgCount` – Desired read count, one message is a 32-bit word.
- `needAckLock` – Upper layer should always check this value. When this is set to true by this function, upper layer should send lock ack message to peer CPU.

Returns

Count of messages actually received.

```
int32_t IMU_SendMsgPtrBlocking(imu_link_t link, uint32_t msgPtr, bool lockSendFifo)
```

Blocking to send messages pointer.

Compared with `IMU_SendMsgsBlocking`, this function fills message pointer to TX FIFO, but not the message content.

This function blocks until the message pointer is filled to TX FIFO.

- If the TX FIFO is locked, this function returns `IMU_ERR_TX_FIFO_LOCKED`.
- If TX FIFO not locked, this function waits the available empty slot in TX FIFO, and fills the message pointer to TX FIFO.
- To lock TX FIFO after filling the message pointer, set `lockSendFifo` to true.

If `IMU_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `IMU_ERR_TIMEOUT` if waiting for FIFO space takes too long.

Parameters

- `link` – IMU link.
- `msgPtr` – The buffer pointer to message to send.
- `lockSendFifo` – If set to true, the TX FIFO is locked after message pointer filled to TX FIFO.

Returns

If TX FIFO is locked, this function returns `IMU_ERR_TX_FIFO_LOCKED`. If a timeout occurs while waiting for FIFO space, it returns `IMU_ERR_TIMEOUT`. Otherwise, this function returns 0 to indicate success.

```
static inline void IMU_LockSendFifo(imu_link_t link, bool lock)
```

Lock or unlock the TX FIFO.

Parameters

- link – IMU link.
- lock – Use true to lock the FIFO, use false to unlock.

```
void IMU_FlushSendFifo(imu_link_t link)
```

Flush the send FIFO.

Flush all messages in send FIFO.

Parameters

- link – IMU link.

```
static inline void IMU_SetSendFifoWaterMark(imu_link_t link, uint8_t waterMark)
```

Set send FIFO water mark.

The water mark must be less than `IMU_MAX_MSG_FIFO_WATER_MARK`, i.e. $0 < \text{waterMark} \leq \text{IMU_MAX_MSG_FIFO_WATER_MARK}$.

Parameters

- link – IMU link.
- waterMark – Send FIFO water mark.

```
static inline uint32_t IMU_GetReceivedMsgCount(imu_link_t link)
```

Get the message count in receive FIFO.

Parameters

- link – IMU link.

Returns

The message count in receive FIFO.

```
static inline uint32_t IMU_GetSendFifoEmptySpace(imu_link_t link)
```

Get the empty slot in send FIFO.

Parameters

- link – IMU link.

Returns

The empty slot count in send FIFO.

```
uint32_t IMU_GetStatusFlags(imu_link_t link)
```

Gets the IMU status flags.

Parameters

- link – IMU link.

Returns

Bit mask of the IMU status flags, see `_imu_status_flags`.

```
static inline void IMU_ClearPendingInterrupts(imu_link_t link, uint32_t mask)
```

Clear the IMU IRQ.

Parameters

- link – IMU link.
- mask – Bit mask of the interrupts to clear, see `_imu_interrupts`.

FSL_IMU_DRIVER_VERSION

IMU driver version.

enum _imu_status_flags

IMU status flags. .

Values:

enumerator kIMU_TxFifoEmpty

enumerator kIMU_TxFifoFull

enumerator kIMU_TxFifoAlmostFull

enumerator kIMU_TxFifoLocked

enumerator kIMU_RxFifoEmpty

enumerator kIMU_RxFifoFull

enumerator kIMU_RxFifoAlmostFull

enumerator kIMU_RxFifoLocked

enum _imu_interrupts

IMU interrupt. .

Values:

enumerator kIMU_RxMsgReadyInterrupt

enumerator kIMU_TxFifoSpaceAvailableInterrupt

IMU_MSG_FIFO_STATUS_MSG_FIFO_LOCKED_MASK

IMU_MSG_FIFO_STATUS_MSG_FIFO_ALMOST_FULL_MASK

IMU_MSG_FIFO_STATUS_MSG_FIFO_FULL_MASK

IMU_MSG_FIFO_STATUS_MSG_FIFO_EMPTY_MASK

IMU_MSG_FIFO_STATUS_MSG_COUNT_MASK

IMU_MSG_FIFO_STATUS_MSG_COUNT_SHIFT

IMU_MSG_FIFO_STATUS_WR_PTR_MASK

IMU_MSG_FIFO_STATUS_WR_PTR_SHIFT

IMU_MSG_FIFO_STATUS_RD_PTR_MASK

IMU_MSG_FIFO_STATUS_RD_PTR_SHIFT

IMU_MSG_FIFO_CNTL_MSG_RDY_INT_CLR_MASK

IMU_MSG_FIFO_CNTL_SP_AV_INT_CLR_MASK

IMU_MSG_FIFO_CNTL_FIFO_FLUSH_MASK

IMU_MSG_FIFO_CNTL_WAIT_FOR_ACK_MASK

IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK_MASK

IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK_SHIFT

IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK(x)

IMU_WR_MSG(link, msg)

IMU_RD_MSG(link)

IMU_RX_FIFO_LOCKED(link)

IMU_TX_FIFO_LOCKED(link)

IMU_TX_FIFO_ALMOST_FULL(link)

IMU_RX_FIFO_EMPTY(link)

Get Rx FIFO empty status.

IMU_LOCK_TX_FIFO(link)

IMU_UNLOCK_TX_FIFO(link)

IMU_RX_FIFO_MSG_COUNT(link)

IMU_TX_FIFO_MSG_COUNT(link)

IMU_RX_FIFO_MSG_COUNT_FROM_STATUS(rxFifoStatus)

IMU_RX_FIFO_LOCKED_FROM_STATUS(rxFifoStatus)

IMU_TX_FIFO_STATUS(link)

IMU_RX_FIFO_STATUS(link)

IMU_TX_FIFO_CNTL(link)

IMU_ERR_TX_FIFO_LOCKED

IMU driver returned error value.

IMU_ERR_TIMEOUT

IMU driver returned error value timeout.

IMU_MSG_FIFO_MAX_COUNT

Maximum message numbers in FIFO.

IMU_MAX_MSG_FIFO_WATER_MARK

Maximum message FIFO water mark.

IMU_FIFO_SW_WRAPAROUND(ptr)

IMU_WR_PTR(link)

IMU_RD_PTR(link)

IMU_BUSY_POLL_COUNT

Maximum polling iterations for IMU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the IMU driver code before timing out and returning an error.

It applies to all waiting loops in IMU driver.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if sensors don't respond or if communication interfaces fail.

struct IMU_Type

#include <fsl_imu.h> IMU register structure.

2.35 Common Driver

FSL_COMMON_DRIVER_VERSION

common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE

No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART

Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART

Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI

Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC

Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM

Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART

Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART

Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART

Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO

Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI

Debug console based on QSCI.

MIN(*a*, *b*)

Computes the minimum of *a* and *b*.

MAX(*a*, *b*)

Computes the maximum of *a* and *b*.

UINT16_MAX

Max value of uint16_t type.

UINT32_MAX

Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(*addr*, *val*)

Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(*addr*, *val*)

Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(*addr*, *bits*)

Set the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(*addr*, *bits*)

Clear the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specified by *clearBits* and set the bits specified by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

AT_NONCACHEABLE_SECTION(var)

Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)

Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPi

Group number for LPSPi status codes.

enumerator `kStatusGroup_FLEXIO_SPI`
Group number for FLEXIO SPI status codes.

enumerator `kStatusGroup_DSPI`
Group number for DSPI status codes.

enumerator `kStatusGroup_FLEXIO_UART`
Group number for FLEXIO UART status codes.

enumerator `kStatusGroup_FLEXIO_I2C`
Group number for FLEXIO I2C status codes.

enumerator `kStatusGroup_LPI2C`
Group number for LPI2C status codes.

enumerator `kStatusGroup_UART`
Group number for UART status codes.

enumerator `kStatusGroup_I2C`
Group number for UART status codes.

enumerator `kStatusGroup_LPSCI`
Group number for LPSCI status codes.

enumerator `kStatusGroup_LPUART`
Group number for LPUART status codes.

enumerator `kStatusGroup_SPI`
Group number for SPI status code.

enumerator `kStatusGroup_XRDC`
Group number for XRDC status code.

enumerator `kStatusGroup_SEMA42`
Group number for SEMA42 status code.

enumerator `kStatusGroup_SDHC`
Group number for SDHC status code

enumerator `kStatusGroup_SDMMC`
Group number for SDMMC status code

enumerator `kStatusGroup_SAI`
Group number for SAI status code

enumerator `kStatusGroup_MCG`
Group number for MCG status codes.

enumerator `kStatusGroup_SCG`
Group number for SCG status codes.

enumerator `kStatusGroup_SDSPI`
Group number for SDSPI status codes.

enumerator `kStatusGroup_FLEXIO_I2S`
Group number for FLEXIO I2S status codes

enumerator `kStatusGroup_FLEXIO_MCULCD`
Group number for FLEXIO LCD status codes

enumerator `kStatusGroup_FLASHIAP`
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

- enumerator kStatusGroup_DMIC
Group number for DMIC status codes.
- enumerator kStatusGroup_SDIF
Group number for SDIF status codes.
- enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.
- enumerator kStatusGroup_OTP
Group number for OTP status codes.
- enumerator kStatusGroup_MCAN
Group number for MCAN status codes.
- enumerator kStatusGroup_CAAM
Group number for CAAM status codes.
- enumerator kStatusGroup_ECSPi
Group number for ECSPi status codes.
- enumerator kStatusGroup_USDHC
Group number for USDHC status codes.
- enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.
- enumerator kStatusGroup_DCP
Group number for DCP status codes.
- enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.
- enumerator kStatusGroup_ESAI
Group number for ESAI status codes.
- enumerator kStatusGroup_FLEXSPi
Group number for FLEXSPi status codes.
- enumerator kStatusGroup_MMDC
Group number for MMDC status codes.
- enumerator kStatusGroup_PDM
Group number for MIC status codes.
- enumerator kStatusGroup_SDMA
Group number for SDMA status codes.
- enumerator kStatusGroup_ICs
Group number for ICs status codes.
- enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.
- enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.
- enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes
- enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

- enumerator `kStatusGroup_HAL_FLASH`
Group number for HAL FLASH status codes.
- enumerator `kStatusGroup_HAL_PWM`
Group number for HAL PWM status codes.
- enumerator `kStatusGroup_HAL_RNG`
Group number for HAL RNG status codes.
- enumerator `kStatusGroup_HAL_I2S`
Group number for HAL I2S status codes.
- enumerator `kStatusGroup_HAL_ADC_SENSOR`
Group number for HAL ADC SENSOR status codes.
- enumerator `kStatusGroup_TIMERMANAGER`
Group number for TiMER MANAGER status codes.
- enumerator `kStatusGroup_SERIALMANAGER`
Group number for SERIAL MANAGER status codes.
- enumerator `kStatusGroup_LED`
Group number for LED status codes.
- enumerator `kStatusGroup_BUTTON`
Group number for BUTTON status codes.
- enumerator `kStatusGroup_EXTERN_EEPROM`
Group number for EXTERN EEPROM status codes.
- enumerator `kStatusGroup_SHELL`
Group number for SHELL status codes.
- enumerator `kStatusGroup_MEM_MANAGER`
Group number for MEM MANAGER status codes.
- enumerator `kStatusGroup_LIST`
Group number for List status codes.
- enumerator `kStatusGroup_OSA`
Group number for OSA status codes.
- enumerator `kStatusGroup_COMMON_TASK`
Group number for Common task status codes.
- enumerator `kStatusGroup_MSG`
Group number for messaging status codes.
- enumerator `kStatusGroup_SDK_OCOTP`
Group number for OCOTP status codes.
- enumerator `kStatusGroup_SDK_FLEXSPINOR`
Group number for FLEXSPINOR status codes.
- enumerator `kStatusGroup_CODEC`
Group number for codec status codes.
- enumerator `kStatusGroup_ASRC`
Group number for codec status ASRC.
- enumerator `kStatusGroup_OTFAD`
Group number for codec status codes.

- enumerator kStatusGroup_SDIO SLV
Group number for SDIO SLV status codes.
- enumerator kStatusGroup_MECC
Group number for MECC status codes.
- enumerator kStatusGroup_ENET_QOS
Group number for ENET_QOS status codes.
- enumerator kStatusGroup_LOG
Group number for LOG status codes.
- enumerator kStatusGroup_I3CBUS
Group number for I3CBUS status codes.
- enumerator kStatusGroup_QSCI
Group number for QSCI status codes.
- enumerator kStatusGroup_ELE MU
Group number for ELE MU status codes.
- enumerator kStatusGroup_QUEUED SPI
Group number for QUEUED SPI status codes.
- enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.
- enumerator kStatusGroup_IPED
Group number for IPED status codes.
- enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.
- enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.
- enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.
- enumerator kStatusGroup_CLIF
Group number for CLIF status codes.
- enumerator kStatusGroup_BMA
Group number for BMA status codes.
- enumerator kStatusGroup_NETC
Group number for NETC status codes.
- enumerator kStatusGroup_ELE
Group number for ELE status codes.
- enumerator kStatusGroup_GLIKEY
Group number for GLIKEY status codes.
- enumerator kStatusGroup_AON_POWER
Group number for AON_POWER status codes.
- enumerator kStatusGroup_AON_COMMON
Group number for AON_COMMON status codes.
- enumerator kStatusGroup_ENDAT3
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
Generic status for Success.

enumerator kStatus_Fail
Generic status for Fail.

enumerator kStatus_ReadOnly
Generic status for read only failure.

enumerator kStatus_OutOfRange
Generic status for out of range access.

enumerator kStatus_InvalidArgument
Generic status for invalid argument check.

enumerator kStatus_Timeout
Generic status for timeout.

enumerator kStatus_NoTransferInProgress
Generic status for no transfer in progress.

enumerator kStatus_Busy
Generic status for module is busy.

enumerator kStatus_NoData
Generic status for no data is found for the operation.

typedef int32_t status_t
Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values

The – allocated memory.

```
void SDK_Free(void *ptr)
```

Free memory.

Parameters

- ptr – The memory to be release.

```
void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)
```

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- delayTime_us – Delay time in unit of microsecond.
- coreClock_Hz – Core clock frequency with Hz.

```
static inline status_t EnableIRQ(IRQn_Type interrupt)
```

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt enabled successfully
- kStatus_Fail – Failed to enable the interrupt

```
static inline status_t DisableIRQ(IRQn_Type interrupt)
```

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt disabled successfully
- kStatus_Fail – Failed to disable the interrupt

```
static inline status_t EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)
```

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

```
static inline status_t IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)
```

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

```
static inline status_t IRQ_ClearPendingIRQ(IRQn_Type interrupt)
```

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

```
static inline uint32_t DisableGlobalIRQ(void)
```

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

Returns

Current primask value.

```
static inline void EnableGlobalIRQ(uint32_t primask)
```

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management

mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

```
static inline bool _SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t
newValue)
```

```
static inline uint32_t _SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)
```

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

Unused	Major Version	Minor Version	Bug Fix
31 25 24	17 16	9 8	0

ARRAY_SIZE(x)

Computes the number of elements in an array.

UINT64_H(X)

Macro to get upper 32 bits of a 64-bit value

UINT64_L(X)

Macro to get lower 32 bits of a 64-bit value

SUPPRESS_FALL_THROUGH_WARNING()

For switch case code block, if case section ends without “break;” statement, there will be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, “SUPPRESS_FALL_THROUGH_WARNING();” need to be added at the end of each case section which misses “break;”statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.36 LIN: Local Interconnect Network Driver

2.37 LIN Driver

`uint32_t LIN_CalcMaxHeaderTimeoutCnt(uint32_t baudRate)`

Calculates maximal header time length.

$T_{header_Maximum} = 1.4 * T_{header_Nominal}$, $T_{header_Nominal} = 34 * T_{bit}$, (13 nominal bits of break; 1 nominal bit of break delimiter; 10 bits for SYNC and 10 bits of PID) The function is not include time for conveying break and break delimiter $T_{IME_OUT_UNIT}$ is in micro second

Parameters

- `baudRate` – baudrate

`uint32_t LIN_CalcMaxResTimeoutCnt(uint32_t baudRate, uint8_t size)`

Calculates maximal header time length.

$T_{response_Maximum} = 1.4 * T_{response_Nominal}$, $T_{response_Nominal} = 10 * (N_{data} + 1) * T_{bit}$

Parameters

- `baudRate` – Baudrate
- `size` – Frame size

`lin_status_t LIN_SetResponse(uint8_t instance, uint8_t *response_buff, uint8_t response_length, uint8_t max_frame_res_timeout)`

Forwards a response to a lower level.

Parameters

- `instance` – LPUART instance
- `response_buff` – response message
- `response_length` – length of response
- `max_frame_res_timeout` – maximal timeout duration for message

Returns

An error code or `lin_status_t`

`lin_status_t LIN_RxResponse(uint8_t instance, uint8_t *response_buff, uint8_t response_length, uint8_t max_frame_res_timeout)`

Forwards a response to a higher level.

Parameters

- `instance` – LPUART instance
- `response_buff` – response message
- `response_length` – length of response
- `max_frame_res_timeout` – maximal timeout duration for message

Returns

An error code or `lin_status_t`

`lin_status_t LIN_IgnoreResponse(uint8_t instance)`

Put a node into idle state.

Parameters

- `instance` – LPUART instance

Returns

An error code or `lin_status_t`

void LIN_GetSlaveDefaultConfig(*lin_user_config_t* *linUserConfig)

Initializes linUserConfig variable for a slave node.

Parameters

- linUserConfig – Pointer to LIN user config structure

void LIN_GetMasterDefaultConfig(*lin_user_config_t* *linUserConfig)

Initializes linUserConfig variable for a master node.

Parameters

- linUserConfig – Pointer to LIN user config structure

void LIN_CalculateBaudrate(uint32_t instance, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *osr, uint16_t *sbr)

Calculates baudrate registers values for given baudrate.

Parameters

- instance – LPUART instance
- instance – baudRate_Bps LPUART baudrate
- instance – srcClock_Hz LPUART clock frequency
- instance – osr LPUART baudrate OSR value, return value
- instance – sbr LPUART baudrate SBR value, return value

void LIN_SetBaudrate(uint32_t instance, uint32_t osr, uint16_t sbr)

Set baudrate registers values.

Parameters

- instance – LPUART instance
- instance – osr LPUART baudrate OSR value
- instance – sbr LPUART baudrate SBR value

lin_status_t LIN_Init(uint32_t instance, *lin_user_config_t* *linUserConfig, *lin_state_t* *linCurrentState, uint32_t clockSource)

Initializes an instance LIN Hardware Interface for LIN Network.

The caller provides memory for the driver state structures during initialization. The user must select the LIN Hardware Interface clock source in the application to initialize the LIN Hardware Interface.

Parameters

- instance – LPUART instance
- linUserConfig – user configuration structure of type *lin_user_config_t*
- linCurrentState – pointer to the LIN Hardware Interface driver state structure

Returns

An error code or *lin_status_t*

lin_status_t LIN_Deinit(uint32_t instance)

Shuts down the LIN Hardware Interface by disabling interrupts and transmitter/receiver.

Parameters

- instance – LPUART instance

Returns

An error code or *lin_status_t*

lin_callback_t LIN_InstallCallback(uint32_t instance, *lin_callback_t* function)

Installs callback function that is used for LIN_DRV_IRQHandler.

Note: After a callback is installed, it bypasses part of the LIN Hardware Interface IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

Parameters

- instance – LPUART instance.
- function – the LIN receive callback function.

Returns

Former LIN callback function pointer.

lin_status_t LIN_SendFrameDataBlocking(uint32_t instance, const uint8_t *txBuff, uint8_t txSize, uint32_t timeoutMSec)

Sends Frame data out through the LIN Hardware Interface using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete.

Parameters

- instance – LPUART instance
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send
- timeoutMSec – timeout value in milli seconds

Returns

An error code or *lin_status_t* LIN_BUS_BUSY if the bus is currently busy, transmission cannot be started.

lin_status_t LIN_SendFrameData(uint32_t instance, const uint8_t *txBuff, uint8_t txSize)

Sends frame data out through the LIN Hardware Interface using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data.

Note: If users use LIN_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN_SetTimeoutCounter(instance, timeoutValue). The timeout value should be big enough to complete the transmission. Timeout in real time is (timeoutValue) * (time period that LIN_TimeoutService is called). For example, if LIN_TimeoutService is called in an timer interrupt with period of 500 micro seconds, then timeout in real time is timeoutValue * 500 micro seconds.

Parameters

- instance – LPUART instance
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send

Returns

An error code or *lin_status_t* LIN_BUS_BUSY if the bus is currently busy, transmission cannot be started. LIN_SUCCESS if the transmission was completed.

lin_status_t LIN_GetTransmitStatus(uint32_t instance, uint8_t *bytesRemaining)

Get status of an on-going non-blocking transmission. While sending frame data using non-blocking method, users can use this function to get status of that transmission. The bytesRemaining shows number of bytes that still needed to transmit.

Parameters

- instance – LPUART instance
- bytesRemaining – Number of bytes still needed to transmit

Returns

lin_status_t LIN_TX_BUSY if the transmission is still in progress.
 LIN_TIMEOUT if timeout occurred and transmission was not completed.
 LIN_SUCCESS if the transmission was successful.

lin_status_t LIN_ReceiveFrameDataBlocking(uint32_t instance, uint8_t *rxBuff, uint8_t rxSize, uint32_t timeoutMSec)

Receives frame data through the LIN Hardware Interface using blocking method. This function will check the checksum byte. If the checksum is correct, it will receive the frame data. Blocking means that the function does not return until the reception is complete.

Parameters

- instance – LPUART instance
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive
- timeoutMSec – timeout value in milli seconds

Returns

An error code or *lin_status_t*

lin_status_t LIN_ReceiveFrameData(uint32_t instance, uint8_t *rxBuff, uint8_t rxSize)

Receives frame data through the LIN Hardware Interface using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete.

Note: If users use LIN_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN_SetTimeoutCounter(instance, timeoutValue). The timeout value should be big enough to complete the reception. Timeout in real time is (timeoutValue) * (time period that LIN_TimeoutService is called). For example, if LIN_TimeoutService is called in a timer interrupt with period of 500 micro seconds, then timeout in real time is timeoutValue * 500 micro seconds.

Parameters

- instance – LPUART instance
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive

Returns

An error code or *lin_status_t*

lin_status_t LIN_AbortTransferData(uint32_t instance)

Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

- instance – LPUART instance

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_GetReceiveStatus(uint32_t instance, uint8_t *bytesRemaining)

Get status of an on-going non-blocking reception While receiving frame data using non-blocking method, users can use this function to get status of that receiving. This function return the current event ID, LIN_RX_BUSY while receiving and return LIN_SUCCESS, or timeout (LIN_TIMEOUT) when the reception is complete. The bytesRemaining shows number of bytes that still needed to receive.

Parameters

- instance – LPUART instance
- bytesRemaining – Number of bytes still needed to receive

Returns

`lin_status_t` LIN_RX_BUSY, LIN_TIMEOUT or LIN_SUCCESS

`lin_status_t` LIN_GoToSleepMode(uint32_t instance)

Puts current LIN node to sleep mode This function changes current node state to LIN_NODE_STATE_SLEEP_MODE.

Parameters

- instance – LPUART instance

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_GotoIdleState(uint32_t instance)

Puts current LIN node to Idle state This function changes current node state to LIN_NODE_STATE_IDLE.

Parameters

- instance – LPUART instance

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_SendWakeupSignal(uint32_t instance)

Sends a wakeup signal through the LIN Hardware Interface.

Parameters

- instance – LPUART instance

Returns

An error code or `lin_status_t`

`lin_node_state_t` LIN_GetCurrentNodeState(uint32_t instance)

Get the current LIN node state.

Parameters

- instance – LPUART instance

Returns

current LIN node state

void LIN_TimeoutService(uint32_t instance)

Callback function for Timer Interrupt Handler Users may use (optional, not required) LIN_TimeoutService to check if timeout has occurred during non-blocking frame data transmission and reception. User may initialize a timer (for example FTM) in Output Compare

Mode with period of 500 micro seconds (recommended). In timer IRQ handler, call this function.

Parameters

- instance – LPUART instance

Returns

void

void LIN_SetTimeoutCounter(uint32_t instance, uint32_t timeoutValue)

Set Value for Timeout Counter that is used in LIN_TimeoutService.

Parameters

- instance – LPUART instance
- timeoutValue – Timeout Value to be set

Returns

void

lin_status_t LIN_MasterSendHeader(uint32_t instance, uint8_t id)

Sends frame header out through the LIN Hardware Interface using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity.

Parameters

- instance – LPUART instance
- id – Frame Identifier

Returns

An error code or lin_status_t

lin_status_t LIN_EnableIRQ(uint32_t instance)

Enables LIN hardware interrupts.

Parameters

- instance – LPUART instance

Returns

An error code or lin_status_t

lin_status_t LIN_DisableIRQ(uint32_t instance)

Disables LIN hardware interrupts.

Parameters

- instance – LPUART instance

Returns

An error code or lin_status_t

void LIN_IRQHandler(uint8_t instance)

Interrupt handler for LIN Hardware Interface.

Parameters

- instance – LPUART instance

Returns

void

lin_status_t LIN_AutoBaudCapture(uint32_t instance)

This function capture bits time to detect break char, calculate baudrate from sync bits and enable transceiver if autobaud successful. This function should only be used in Slave. The timer should be in mode input capture of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

- instance – LPUART instance

Returns

lin_status_t

uint8_t LIN_ProcessParity(uint8_t PID, uint8_t typeAction)

Makes or checks parity bits. If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID. This is not a public API as it is called by other API functions.

Parameters

- PID – PID byte in case of checking parity bits or ID byte in case of making parity bits.
- typeAction – 1 for Checking parity bits, 0 for making parity bits

Returns

0xFF if parity bits are incorrect, ID in case of checking parity bits and they are correct. Function returns PID in case of making parity bits.

uint8_t LIN_MakeChecksumByte(const uint8_t *buffer, uint8_t sizeBuffer, uint8_t PID)

Makes the checksum byte for a frame.

Parameters

- buffer – Pointer to Tx buffer
- sizeBuffer – Number of bytes that are contained in the buffer.
- PID – Protected Identifier byte.

Returns

the checksum byte.

FSL_LIN_DRIVER_VERSION

LIN driver version.

enum lin_frame_type

List of LIN frame types.

Values:

enumerator LIN_FRM_UNCD
unconditional frame

enumerator LIN_FRM_EVNT
event triggered frame

enumerator LIN_FRM_SPRDC
sporadic frame

enumerator LIN_FRM_DIAG
diagnostic frame

enum lin_frame_response

List of LIN frame response types.

Values:

enumerator LIN_RES_NOTHING
node is neither publisher nor subscriber

enumerator LIN_RES_PUB
node is publisher

enumerator LIN_RES_SUB
node is subscriber

enum lin_diagnostic_mode
Diagnostic mode.

Values:

enumerator DIAG_NONE
None

enumerator DIAG_INTERLEAVE_MODE
Interleave mode

enumerator DIAG_ONLY_MODE
Diagnostic only mode

enum lin_event_id_t
Defines types for an enumerating event related to an Identifier.

Values:

enumerator LIN_NO_EVENT
No event yet

enumerator LIN_WAKEUP_SIGNAL
Received a wakeup signal

enumerator LIN_BAUDRATE_ADJUSTED
Indicate that baudrate was adjusted to Master's baudrate

enumerator LIN_RECV_BREAK_FIELD_OK
Indicate that correct Break Field was received

enumerator LIN_SYNC_OK
Sync byte is correct

enumerator LIN_SYNC_ERROR
Sync byte is incorrect

enumerator LIN_PID_OK
PID correct

enumerator LIN_PID_ERROR
PID incorrect

enumerator LIN_FRAME_ERROR
Framing Error

enumerator LIN_READBACK_ERROR
Readback data is incorrect

enumerator LIN_CHECKSUM_ERROR
Checksum byte is incorrect

enumerator LIN_TX_COMPLETED
Sending data completed

enumerator LIN_RX_COMPLETED
Receiving data completed

enumerator LIN_NO_DATA_TIMEOUT

No data timeout

enumerator LIN_BUS_ACTIVITY_TIMEOUT

Bus activity timeout

enumerator LIN_TIMEOUT_ERROR

Indicate that timeout has occurred

enumerator LIN_LAST_RESPONSE_SHORT_ERROR

Indicate that the last frame was too short

enum lin_status_t

Defines Error codes of the LIN driver.

Values:

enumerator LIN_IFC_NOT_SUPPORT

This interface is not supported

enumerator LIN_INITIALIZED

LIN Hardware has been initialized

enumerator LIN_SUCCESS

Successfully done

enumerator LIN_ERROR

Error

enumerator LIN_TX_BUSY

Transmitter is busy

enumerator LIN_RX_BUSY

Receiver is busy

enumerator LIN_BUS_BUSY

Bus is busy

enumerator LIN_NO_TRANSFER_IN_PROGRESS

No data transfer is in progress

enumerator LIN_TIMEOUT

Timeout

enumerator LIN_LPUART_STAT_CLOCK_GATED_OFF

LPUART is gated from clock manager

enum lin_node_state_t

Define type for an enumerating LIN Node state.

Values:

enumerator LIN_NODE_STATE_UNINIT

Uninitialized state

enumerator LIN_NODE_STATE_SLEEP_MODE

Sleep mode state

enumerator LIN_NODE_STATE_IDLE

Idle state

enumerator LIN_NODE_STATE_SEND_BREAK_FIELD

Send break field state

enumerator LIN_NODE_STATE_RECV_SYNC
Receive the synchronization byte state

enumerator LIN_NODE_STATE_SEND_PID
Send PID state

enumerator LIN_NODE_STATE_RECV_PID
Receive PID state

enumerator LIN_NODE_STATE_RECV_DATA
Receive data state

enumerator LIN_NODE_STATE_RECV_DATA_COMPLETED
Receive data completed state

enumerator LIN_NODE_STATE_SEND_DATA
Send data state

enumerator LIN_NODE_STATE_SEND_DATA_COMPLETED
Send data completed state

enum lin_protocol_handle
List of protocols.
Values:

enumerator LIN_PROTOCOL_21
LIN protocol version 2.1

enumerator LIN_PROTOCOL_J2602
J2602 protocol

enum lin_supported_baudrates_t
List of supported baudrates for autobaud feature.
Values:

enumerator kLIN_BAUD_2400

enumerator kLIN_BAUD_4800

enumerator kLIN_BAUD_9600

enumerator kLIN_BAUD_14400

enumerator kLIN_BAUD_19200

typedef void (*lin_timer_get_time_interval_t)(uint32_t *nanoSeconds)
Callback function to get time interval in nano seconds.

typedef void (*lin_callback_t)(uint32_t instance, void *linState)
LIN Driver callback function type.

uint8_t hardware_instance
interface instance number

uint32_t baudRate
baudrate of LIN Hardware Interface to configure

bool nodeFunction
Node function as Master or Slave

bool autobaudEnable
Enable Autobaud feature

lin_timer_get_time_interval_t timerGetTimeIntervalCallback
Callback function to get time interval in nano seconds

const uint8_t *txBuff
The buffer of data being sent.

uint8_t *rxBuff
The buffer of received data.

uint8_t frame_index

uint8_t cntByte
To count number of bytes already transmitted or received.

volatile uint8_t txSize
The remaining number of bytes to be received.

volatile uint8_t rxSize
The remaining number of bytes to be received.

uint8_t checksum
Checksum byte.

volatile bool isTxBusy
True if the LIN interface is transmitting frame data.

volatile bool isRxBusy
True if the LIN interface is receiving frame data.

volatile bool isBusBusy
True if there are data, frame headers being transferred on bus

volatile bool isTxBlocking
True if transmit is blocking transaction.

volatile bool isRxBlocking
True if receive is blocking transaction.

lin_callback_t Callback
Callback function to invoke after receiving a byte or transmitting a byte.

uint8_t currentId
Current ID

uint8_t currentPid
Current PID

volatile *lin_event_id_t* currentEventId
Current ID Event

volatile *lin_node_state_t* currentNodeState
Current Node state

volatile uint32_t timeoutCounter
Value of the timeout counter

volatile bool timeoutCounterFlag
Timeout counter flag

volatile bool baudrateEvalEnable
Baudrate Evaluation Process Enable

`volatile uint8_t` `fallingEdgeInterruptCount`
Falling Edge count of a sync byte

`uint32_t` `linSourceClockFreq`
Frequency of the source clock for LIN

`volatile bool` `txCompleted`
Used to wait for LIN interface ISR to complete transmission.

`volatile bool` `rxCompleted`
Used to wait for LIN interface ISR to complete reception

`uint32_t` `baudRate`

`uint32_t` `osrValue`

`uint16_t` `sbrValue`

`lin_frame_type` `frm_type`
Frame information (unconditional or event triggered..)

`uint8_t` `frm_len`
Length of the frame

`lin_frame_response` `frm_response`
Action response when received PID

`uint8_t` `frm_offset`
Frame byte offset in frame buffer

`uint8_t` `flag_offset`
Flag byte offset in flag buffer

`uint8_t` `flag_size`
Flag size in flag buffer

`uint32_t` `delay`
Frame delay

`const uint8_t *``frame_data_ptr`
List of Signal to which the frame is associated and its offset

`lin_protocol_handle` `protocol_version`
Protocol version

`lin_protocol_handle` `language_version`
Language version

`uint8_t` `number_of_configurable_frames`
Number of frame except diagnostic frames

`uint8_t` `frame_start`
Start index of frame list

`const lin_frame_struct *``frame_tbl_ptr`
Frame list except diagnostic frames

`const uint16_t *``list_identifiers_ROM_ptr`
Configuration in ROM

`uint8_t *``list_identifiers_RAM_ptr`
Configuration in RAM

`uint16_t max_idle_timeout_cnt`
Max Idle timeout counter

`uint16_t max_message_length`
Max message length

`uint16_t supplier_id`
Supplier ID

`uint16_t function_id`
Function ID

`uint8_t variant`
Variant value

`uint8_t serial_0`
Serial 0

`uint8_t serial_1`
Serial 1

`uint8_t serial_2`
Serial 2

`uint8_t serial_3`
Serial 3

`uint16_t baud_rate`
Adjusted baud rate

`uint8_t *response_buffer_ptr`
Response buffer

`uint8_t response_length`
Response length

`uint8_t successful_transfer`
Sets when frame is transferred successfully

`uint8_t error_in_response`
Sets when frame received/transmitter by the node contains an error in the response field

`uint8_t timeout_in_response`
Timeout response

`bool go_to_sleep_flg`
Go to sleep flag

`uint8_t current_id`
Current PID

`uint8_t num_of_processed_frame`
Number of processed frames

`uint8_t num_of_successfull_frame`
Number of processed frames

`uint8_t next_transmit_tick`
Used to count the next transmit tick

`bool save_config_flg`
Set when save configuration request has been received

lin_diagnostic_mode diagnostic_mode
Diagnostic mode

uint16_t frame_timeout_cnt
Frame timeout counter

uint16_t idle_timeout_cnt
Idle timeout counter, node will go to sleep when count down to 0

bool transmit_error_resp_sig_flg
Flag indicates that the error response signal is going to be sent

bool event_trigger_collision_flg
Flag indicates collision on bus

uint8_t *configured_NAD_ptr
NAD value used in configuration command

uint8_t initial_NAD
Initial NAD

lin_product_id product_id
Product ID

lin_serial_number serial_number
Serial number

uint8_t *resp_err_frm_id_ptr
Pointer to the list of index of frames that carries response error signal

uint8_t num_frame_have_esignal
The count of frames that carry response error signal

uint8_t response_error_byte_offset
Byte offset of response error signal

uint8_t response_error_bit_offset
Bit offset of response error signal

uint8_t num_of_fault_state_signal
Number of Fault state signal

uint16_t P2_min
P2 min

uint16_t ST_min
ST min

uint16_t N_As_timeout
N_As_timeout

uint16_t N_Cr_timeout
N_Cr_timeout

uint8_t active_schedule_id
Active schedule table id

uint8_t previous_schedule_id
Previous schedule table id

uint8_t *schedule_start_entry_ptr
Start entry of each schedule table

bool event_trigger_collision_flg

uint8_t data_buffer[8]
Master data buffer

uint8_t frm_offset

uint8_t frm_size

LPUART_Type *const g_linLpuartBase[1]
Table of base addresses for LPUART instances.

const IRQn_Type g_linLpuartRxTxIrqId[1]
Table to save LPUART IRQ enumeration numbers defined in the CMSIS header file.

lin_baudrate_values_t g_linConfigBaudrates[5U]
Table to save LIN user config buadrate values.

lin_state_t *g_linStatePtr[1]
Pointers to LPUART bases for each instance.
Table to save LPUART state structure pointers

lin_user_config_t *g_linUserconfigPtr[1]
Table to save LIN user config structure pointers.

LIN_SLAVE

LIN_MASTER

MAKE_PARITY

CHECK_PARITY

LIN_TIME_OUT_UNIT_US

LIN_MAKE_UNCONDITIONAL_FRAME

LIN_UPDATE_UNCONDITIONAL_FRAME

LIN_NUM_OF_SUPP_BAUDRATES

struct lin_user_config_t
#include <fsl_lin.h> LIN hardware configuration structure.

struct lin_state_t
#include <fsl_lin.h> Runtime state of the LIN driver.

Note that the caller provides memory for the driver state structures during initialization because the driver does not statically allocate memory.

struct lin_baudrate_values_t
#include <fsl_lin.h> Structure of baudrate properties.

struct lin_frame_struct
#include <fsl_lin.h> Informations of frame.

struct lin_protocol_user_config_t
#include <fsl_lin.h> Protocol configuration structure.

struct lin_product_id
#include <fsl_lin.h> Product id structure.

struct lin_serial_number
#include <fsl_lin.h> Serial number.

```
struct lin_protocol_state_t
    #include <fsl_lin.h> Protocol state structure.
struct lin_node_attribute
    #include <fsl_lin.h> Attributes of LIN node.
struct lin_master_data_t
    #include <fsl_lin.h> LIN master data structure.
```

2.38 LIN LPUART Driver

FSL_LIN_LPUART_DRIVER_VERSION

LIN LPUART driver version.

enum _lin_lpuart_stop_bit_count

Values:

enumerator kLPUART_OneStopBit

One stop bit

enumerator kLPUART_TwoStopBit

Two stop bits

enum _lin_lpuart_flags

Values:

enumerator kLPUART_TxDataRegEmptyFlag

Transmit data register empty flag, sets when transmit buffer is empty

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start bit

enumerator kLPUART_DataMatch1Flag

The next character to be read from LPUART_DATA matches MA1

enumerator kLPUART_DataMatch2Flag

The next character to be read from LPUART_DATA matches MA2

enumerator kLPUART_NoiseErrorInRxDataRegFlag

NOISY bit, sets if noise detected in current data word

enumerator kLPUART_ParityErrorInRxDataRegFlag

PARITY bit, sets if noise detected in current data word

enumerator kLPUART_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty

enumerator kLPUART_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty

enumerator kLPUART_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred

enumerator kLPUART_RxFifoUnderflowFlag

RXUF bit, sets if receive buffer underflow occurred

enum _lin_lpuart_interrupt_enable

Values:

enumerator kLPUART_LinBreakInterruptEnable

LIN break detect.

enumerator kLPUART_RxActiveEdgeInterruptEnable

Receive Active Edge.

enumerator kLPUART_TxDataRegEmptyInterruptEnable

Transmit data register empty.

enumerator kLPUART_TransmissionCompleteInterruptEnable

Transmission complete.

enumerator kLPUART_RxDataRegFullInterruptEnable

Receiver data register full.

enumerator kLPUART_IdleLineInterruptEnable

Idle line.

enumerator kLPUART_RxOverrunInterruptEnable

Receiver Overrun.

enumerator kLPUART_NoiseErrorInterruptEnable

Noise error flag.

enumerator kLPUART_FramingErrorInterruptEnable

Framing error flag.

enumerator kLPUART_ParityErrorInterruptEnable

Parity error flag.

enumerator kLPUART_TxFifoOverflowInterruptEnable

Transmit FIFO Overflow.

```

    enumerator kLPUART_RxFifoUnderflowInterruptEnable
        Receive FIFO Underflow.
enum _lin_lpuart_status
    Values:
    enumerator kStatus_LPUART_TxBusy
        TX busy
    enumerator kStatus_LPUART_RxBusy
        RX busy
    enumerator kStatus_LPUART_TxIdle
        LPUART transmitter is idle.
    enumerator kStatus_LPUART_RxIdle
        LPUART receiver is idle.
    enumerator kStatus_LPUART_TxWatermarkTooLarge
        TX FIFO watermark too large
    enumerator kStatus_LPUART_RxWatermarkTooLarge
        RX FIFO watermark too large
    enumerator kStatus_LPUART_FlagCannotClearManually
        Some flag can't manually clear
    enumerator kStatus_LPUART_Error
        Error happens on LPUART.
    enumerator kStatus_LPUART_RxRingBufferOverrun
        LPUART RX software ring buffer overrun.
    enumerator kStatus_LPUART_RxHardwareOverrun
        LPUART RX receiver overrun.
    enumerator kStatus_LPUART_NoiseError
        LPUART noise error.
    enumerator kStatus_LPUART_FramingError
        LPUART framing error.
    enumerator kStatus_LPUART_ParityError
        LPUART parity error.
enum lin_lpuart_bit_count_per_char_t
    Values:
    enumerator LPUART_8_BITS_PER_CHAR
        8-bit data characters
    enumerator LPUART_9_BITS_PER_CHAR
        9-bit data characters
    enumerator LPUART_10_BITS_PER_CHAR
        10-bit data characters
typedef enum _lin_lpuart_stop_bit_count lin_lpuart_stop_bit_count_t
static inline bool LIN_LPUART_GetRxDataPolarity(const LPUART_Type *base)
static inline void LIN_LPUART_SetRxDataPolarity(LPUART_Type *base, bool polarity)

```

```
static inline void LIN_LPUART_WriteByte(LPUART_Type *base, uint8_t data)
```

```
static inline void LIN_LPUART_ReadByte(const LPUART_Type *base, uint8_t *readData)
```

```
status_t LIN_LPUART_CalculateBaudRate(LPUART_Type *base, uint32_t baudRate_Bps,  
                                     uint32_t srcClock_Hz, uint32_t *osr, uint16_t *sbr)
```

Calculates the best osr and sbr value for configured baudrate.

Parameters

- base – LPUART peripheral base address
- baudRate_Bps – user configuration structure of type `lin_user_config_t`
- srcClock_Hz – pointer to the LIN_LPUART driver state structure
- osr – pointer to osr value
- sbr – pointer to sbr value

Returns

An error code or `lin_status_t`

```
void LIN_LPUART_SetBaudRate(LPUART_Type *base, uint32_t *osr, uint16_t *sbr)
```

Configure baudrate according to osr and sbr value.

Parameters

- base – LPUART peripheral base address
- osr – pointer to osr value
- sbr – pointer to sbr value

```
lin_status_t LIN_LPUART_Init(LPUART_Type *base, lin_user_config_t *linUserConfig,  
                             lin_state_t *linCurrentState, uint32_t linSourceClockFreq)
```

Initializes an LIN_LPUART instance for LIN Network.

The caller provides memory for the driver state structures during initialization. The user must select the LIN_LPUART clock source in the application to initialize the LIN_LPUART. This function initializes a LPUART instance for operation. This function will initialize the run-time state structure to keep track of the on-going transfers, initialize the module to user defined settings and default settings, set break field length to be 13 bit times minimum, enable the break detect interrupt, Rx complete interrupt, frame error detect interrupt, and enable the LPUART module transmitter and receiver

Parameters

- base – LPUART peripheral base address
- linUserConfig – user configuration structure of type `lin_user_config_t`
- linCurrentState – pointer to the LIN_LPUART driver state structure

Returns

An error code or `lin_status_t`

```
lin_status_t LIN_LPUART_Deinit(LPUART_Type *base)
```

Shuts down the LIN_LPUART by disabling interrupts and transmitter/receiver.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

lin_status_t LIN_LPUART_SendFrameDataBlocking(LPUART_Type *base, const uint8_t *txBuff, uint8_t txSize, uint32_t timeoutMSec)

Sends Frame data out through the LIN_LPUART module using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send
- timeoutMSec – timeout value in milli seconds

Returns

An error code or *lin_status_t*

lin_status_t LIN_LPUART_SendFrameData(LPUART_Type *base, const uint8_t *txBuff, uint8_t txSize)

Sends frame data out through the LIN_LPUART module using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send

Returns

An error code or *lin_status_t*

lin_status_t LIN_LPUART_GetTransmitStatus(LPUART_Type *base, uint8_t *bytesRemaining)

Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission. This function return LIN_TX_BUSY while sending, or LIN_TIMEOUT if timeout has occurred, or return LIN_SUCCESS when the transmission is complete. The bytesRemaining shows number of bytes that still needed to transmit.

Parameters

- base – LPUART peripheral base address
- bytesRemaining – Number of bytes still needed to transmit

Returns

lin_status_t LIN_TX_BUSY, LIN_SUCCESS or LIN_TIMEOUT

lin_status_t LIN_LPUART_RecvFrmDataBlocking(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize, uint32_t timeoutMSec)

Receives frame data through the LIN_LPUART module using blocking method. This function will check the checksum byte. If the checksum is correct, it will receive the frame data. Blocking means that the function does not return until the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive

- timeoutMSec – timeout value in milli seconds

Returns

An error code or `lin_status_t`

`lin_status_t` `LIN_LPUART_RecvFrmData(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize)`

Receives frame data through the LIN_LPUART module using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive

Returns

An error code or `lin_status_t`

`lin_status_t` `LIN_LPUART_AbortTransferData(LPUART_Type *base)`

Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` `LIN_LPUART_GetReceiveStatus(LPUART_Type *base, uint8_t *bytesRemaining)`

Get status of an on-going non-blocking reception While receiving frame data using non-blocking method, users can use this function to get status of that receiving. This function return the current event ID, `LIN_RX_BUSY` while receiving and return `LIN_SUCCESS`, or timeout (`LIN_TIMEOUT`) when the reception is complete. The `bytesRemaining` shows number of bytes that still needed to receive.

Parameters

- base – LPUART peripheral base address
- bytesRemaining – Number of bytes still needed to receive

Returns

`lin_status_t` `LIN_RX_BUSY`, `LIN_TIMEOUT` or `LIN_SUCCESS`

`lin_status_t` `LIN_LPUART_GoToSleepMode(LPUART_Type *base)`

This function puts current node to sleep mode This function changes current node state to `LIN_NODE_STATE_SLEEP_MODE`.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` `LIN_LPUART_GotoIdleState(LPUART_Type *base)`

Puts current LIN node to Idle state This function changes current node state to `LIN_NODE_STATE_IDLE`.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_SendWakeupSignal(LPUART_Type *base)

Sends a wakeup signal through the LIN_LPUART interface.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_MasterSendHeader(LPUART_Type *base, uint8_t id)

Sends frame header out through the LIN_LPUART module using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity.

Parameters

- base – LPUART peripheral base address
- id – Frame Identifier

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_EnableIRQ(LPUART_Type *base)

Enables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_DisableIRQ(LPUART_Type *base)

Disables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns

An error code or `lin_status_t`

`lin_status_t` LIN_LPUART_AutoBaudCapture(uint32_t instance)

This function capture bits time to detect break char, calculate baudrate from sync bits and enable transceiver if autobaud successful. This function should only be used in Slave. The timer should be in mode input capture of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

- instance – LPUART instance

Returns

`lin_status_t`

`void` LIN_LPUART_IRQHandler(LPUART_Type *base)

LIN_LPUART RX TX interrupt handler.

Parameters

- base – LPUART peripheral base address

Returns

`void`

LIN_LPUART_TRANSMISSION_COMPLETE_TIMEOUT

Max loops to wait for LPUART transmission complete.

When de-initializing the LIN LPUART module, the program shall wait for the previous transmission to complete. This parameter defines how many loops to check completion before return error. If defined as 0, driver will wait forever until completion.

AUTOBAUD_BAUDRATE_TOLERANCE

BIT_RATE_TOLERANCE_UNSYNC

BIT_DURATION_MAX_19200

BIT_DURATION_MIN_19200

BIT_DURATION_MAX_14400

BIT_DURATION_MIN_14400

BIT_DURATION_MAX_9600

BIT_DURATION_MIN_9600

BIT_DURATION_MAX_4800

BIT_DURATION_MIN_4800

BIT_DURATION_MAX_2400

BIT_DURATION_MIN_2400

TWO_BIT_DURATION_MAX_19200

TWO_BIT_DURATION_MIN_19200

TWO_BIT_DURATION_MAX_14400

TWO_BIT_DURATION_MIN_14400

TWO_BIT_DURATION_MAX_9600

TWO_BIT_DURATION_MIN_9600

TWO_BIT_DURATION_MAX_4800

TWO_BIT_DURATION_MIN_4800

TWO_BIT_DURATION_MAX_2400

TWO_BIT_DURATION_MIN_2400

AUTOBAUD_BREAK_TIME_MIN

2.39 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum _lpadc_status_flags

Define hardware flags of the module.

Values:

enumerator kLPADC_ResultFIFO0OverflowFlag

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC_ResultFIFO0ReadyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC_TriggerExceptionFlag

Indicates that a trigger exception event has occurred.

enumerator kLPADC_TriggerCompletionFlag

Indicates that a trigger completion event has occurred.

enumerator kLPADC_CalibrationReadyFlag

Indicates that the calibration process is done.

enumerator kLPADC_ActiveFlag

Indicates that the ADC is in active state.

enumerator kLPADC_ResultFIFOOverflowFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowFlag as instead.

enumerator kLPADC_ResultFIFOReadyFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0ReadyFlag as instead.

enum _lpadc_interrupt_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowInterruptEnable as instead.

enumerator kLPADC_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_FIFO0WatermarkInterruptEnable as instead.

enumerator kLPADC_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kLPADC_Trigger2CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 2 completion.

enumerator kLPADC_Trigger3CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 3 completion.

enumerator kLPADC_Trigger4CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 4 completion.

enumerator kLPADC_Trigger5CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 5 completion.

enumerator kLPADC_Trigger6CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 6 completion.

enumerator kLPADC_Trigger7CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 7 completion.

enumerator kLPADC_Trigger8CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 8 completion.

enumerator kLPADC_Trigger9CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 9 completion.

enumerator kLPADC_Trigger10CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 10 completion.

enumerator kLPADC_Trigger11CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 11 completion.

enumerator kLPADC_Trigger12CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 12 completion.

enumerator kLPADC_Trigger13CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 13 completion.

enumerator kLPADC_Trigger14CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 14 completion.

enumerator kLPADC_Trigger15CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 15 completion.

enum _lpadc_trigger_status_flags

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_Trigger0InterruptedFlag
Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC_Trigger1InterruptedFlag
Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC_Trigger2InterruptedFlag
Trigger 2 is interrupted by a high priority exception.

enumerator kLPADC_Trigger3InterruptedFlag
Trigger 3 is interrupted by a high priority exception.

enumerator kLPADC_Trigger4InterruptedFlag
Trigger 4 is interrupted by a high priority exception.

enumerator kLPADC_Trigger5InterruptedFlag
Trigger 5 is interrupted by a high priority exception.

enumerator kLPADC_Trigger6InterruptedFlag
Trigger 6 is interrupted by a high priority exception.

enumerator kLPADC_Trigger7InterruptedFlag

Trigger 7 is interrupted by a high priority exception.

enumerator kLPADC_Trigger8InterruptedFlag

Trigger 8 is interrupted by a high priority exception.

enumerator kLPADC_Trigger9InterruptedFlag

Trigger 9 is interrupted by a high priority exception.

enumerator kLPADC_Trigger10InterruptedFlag

Trigger 10 is interrupted by a high priority exception.

enumerator kLPADC_Trigger11InterruptedFlag

Trigger 11 is interrupted by a high priority exception.

enumerator kLPADC_Trigger12InterruptedFlag

Trigger 12 is interrupted by a high priority exception.

enumerator kLPADC_Trigger13InterruptedFlag

Trigger 13 is interrupted by a high priority exception.

enumerator kLPADC_Trigger14InterruptedFlag

Trigger 14 is interrupted by a high priority exception.

enumerator kLPADC_Trigger15InterruptedFlag

Trigger 15 is interrupted by a high priority exception.

enumerator kLPADC_Trigger0CompletedFlag

Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC_Trigger1CompletedFlag

Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kLPADC_Trigger2CompletedFlag

Trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kLPADC_Trigger3CompletedFlag

Trigger 3 is completed and trigger 3 has enabled completion interrupts.

enumerator kLPADC_Trigger4CompletedFlag

Trigger 4 is completed and trigger 4 has enabled completion interrupts.

enumerator kLPADC_Trigger5CompletedFlag

Trigger 5 is completed and trigger 5 has enabled completion interrupts.

enumerator kLPADC_Trigger6CompletedFlag

Trigger 6 is completed and trigger 6 has enabled completion interrupts.

enumerator kLPADC_Trigger7CompletedFlag

Trigger 7 is completed and trigger 7 has enabled completion interrupts.

enumerator kLPADC_Trigger8CompletedFlag

Trigger 8 is completed and trigger 8 has enabled completion interrupts.

enumerator kLPADC_Trigger9CompletedFlag

Trigger 9 is completed and trigger 9 has enabled completion interrupts.

enumerator kLPADC_Trigger10CompletedFlag

Trigger 10 is completed and trigger 10 has enabled completion interrupts.

enumerator kLPADC_Trigger11CompletedFlag

Trigger 11 is completed and trigger 11 has enabled completion interrupts.

enumerator kLPADC_Trigger12CompletedFlag

Trigger 12 is completed and trigger 12 has enabled completion interrupts.

enumerator kLPADC_Trigger13CompletedFlag

Trigger 13 is completed and trigger 13 has enabled completion interrupts.

enumerator kLPADC_Trigger14CompletedFlag

Trigger 14 is completed and trigger 14 has enabled completion interrupts.

enumerator kLPADC_Trigger15CompletedFlag

Trigger 15 is completed and trigger 15 has enabled completion interrupts.

enum _lpadc_sample_scale_mode

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Values:

enumerator kLPADC_SamplePartScale

Use divided input voltage signal. (For scale select, please refer to the reference manual).

enumerator kLPADC_SampleFullScale

Full scale (Factor of 1).

enum _lpadc_sample_channel_mode

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Values:

enumerator kLPADC_SampleChannelSingleEndSideA

Single-end mode, only A-side channel is converted.

enumerator kLPADC_SampleChannelSingleEndSideB

Single-end mode, only B-side channel is converted.

enumerator kLPADC_SampleChannelDiffBothSideAB

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDiffBothSideBA

Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC_SampleChannelDiffBothSide

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDualSingleEndBothSide

Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum _lpadc_hardware_average_mode

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Values:

enumerator kLPADC_HardwareAverageCount1

Single conversion.

enumerator kLPADC_HardwareAverageCount2

2 conversions averaged.

enumerator kLPADC_HardwareAverageCount4

4 conversions averaged.

enumerator kLPADC_HardwareAverageCount8

8 conversions averaged.

enumerator kLPADC_HardwareAverageCount16

16 conversions averaged.

enumerator kLPADC_HardwareAverageCount32

32 conversions averaged.

enumerator kLPADC_HardwareAverageCount64

64 conversions averaged.

enumerator kLPADC_HardwareAverageCount128

128 conversions averaged.

enum _lpadc_sample_time_mode

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Values:

enumerator kLPADC_SampleTimeADCK3

3 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK5

5 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK7

7 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK11

11 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK19

19 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK35

35 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK67

69 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK131

131 ADCK cycles total sample time.

enum `_lpadc_hardware_compare_mode`

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Values:

enumerator `kLPADC_HardwareCompareDisabled`

Compare disabled.

enumerator `kLPADC_HardwareCompareStoreOnTrue`

Compare enabled. Store on true.

enumerator `kLPADC_HardwareCompareRepeatUntilTrue`

Compare enabled. Repeat channel acquisition until true.

enum `_lpadc_conversion_resolution_mode`

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to `lpadc_sample_channel_mode_t`

Values:

enumerator `kLPADC_ConversionResolutionStandard`

Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator `kLPADC_ConversionResolutionHigh`

High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum `_lpadc_conversion_average_mode`

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of `CAL_AVGS` field in CTRL register.

Values:

enumerator `kLPADC_ConversionAverage1`

Single conversion.

enumerator `kLPADC_ConversionAverage2`

2 conversions averaged.

enumerator `kLPADC_ConversionAverage4`

4 conversions averaged.

enumerator `kLPADC_ConversionAverage8`

8 conversions averaged.

enumerator `kLPADC_ConversionAverage16`

16 conversions averaged.

enumerator `kLPADC_ConversionAverage32`

32 conversions averaged.

enumerator kLPADC_ConversionAverage64
64 conversions averaged.

enumerator kLPADC_ConversionAverage128
128 conversions averaged.

enum _lpadc_reference_voltage_mode
Define enumeration of reference voltage source.
For detail information, need to check the SoC's specification.

Values:

enumerator kLPADC_ReferenceVoltageAlt1
Option 1 setting.

enumerator kLPADC_ReferenceVoltageAlt2
Option 2 setting.

enumerator kLPADC_ReferenceVoltageAlt3
Option 3 setting.

enum _lpadc_power_level_mode
Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Values:

enumerator kLPADC_PowerLevelAlt1
Lowest power setting.

enumerator kLPADC_PowerLevelAlt2
Next lowest power setting.

enumerator kLPADC_PowerLevelAlt3

...

enumerator kLPADC_PowerLevelAlt4
Highest power setting.

enum _lpadc_offset_calibration_mode
Define enumeration of offset calibration mode.

Values:

enumerator kLPADC_OffsetCalibration12bitMode
12 bit offset calibration mode.

enumerator kLPADC_OffsetCalibration16bitMode
16 bit offset calibration mode.

enum _lpadc_trigger_priority_policy
Define enumeration of trigger priority policy.
This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

Values:

enumerator `kLPADC_ConvPreemptImmediatelyNotAutoResumed`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator `kLPADC_ConvPreemptSoftlyNotAutoResumed`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator `kLPADC_ConvPreemptImmediatelyAutoRestarted`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator `kLPADC_ConvPreemptSoftlyAutoRestarted`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator `kLPADC_ConvPreemptImmediatelyAutoResumed`

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator `kLPADC_ConvPreemptSoftlyAutoResumed`

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator `kLPADC_TriggerPriorityPreemptImmediately`

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator `kLPADC_TriggerPriorityPreemptSoftly`

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator `kLPADC_TriggerPriorityExceptionDisabled`

High priority trigger exception disabled.

typedef enum *lpadc_sample_scale_mode* `lpadc_sample_scale_mode_t`

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

`typedef enum _lpadc_sample_channel_mode lpadc_sample_channel_mode_t`

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

`typedef enum _lpadc_hardware_average_mode lpadc_hardware_average_mode_t`

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

`typedef enum _lpadc_sample_time_mode lpadc_sample_time_mode_t`

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

`typedef enum _lpadc_hardware_compare_mode lpadc_hardware_compare_mode_t`

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

`typedef enum _lpadc_conversion_resolution_mode lpadc_conversion_resolution_mode_t`

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to `lpadc_sample_channel_mode_t`

`typedef enum _lpadc_conversion_average_mode lpadc_conversion_average_mode_t`

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

`typedef enum _lpadc_reference_voltage_mode lpadc_reference_voltage_source_t`

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

`typedef enum _lpadc_power_level_mode lpadc_power_level_mode_t`

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

`typedef enum _lpadc_offset_calibration_mode lpadc_offset_calibration_mode_t`

Define enumeration of offset calibration mode.

typedef enum *lpadc_trigger_priority_policy* lpadc_trigger_priority_policy_t

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

typedef struct *lpadc_calibration_value* lpadc_calibration_value_t

A structure of calibration value.

LPADC_CONVERSION_COMPLETE_TIMEOUT

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

LPADC_CALIBRATION_READY_TIMEOUT

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

LPADC_GAIN_CAL_READY_TIMEOUT

Max loops to wait for LPADC gain calibration GAIN_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN_CAL ready. This parameter defines how many loops to check the gain calibration GAIN_CAL ready. If defined as 0, driver will wait forever until ready.

ADC_OFSTRIM_OFSTRIM_MAX

ADC_OFSTRIM_OFSTRIM_SIGN

LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

void LPADC_Init(ADC_Type *base, const *lpadc_config_t* *config)

Initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See “*lpadc_config_t*”.

void LPADC_GetDefaultConfig(*lpadc_config_t* *config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```

config->enableInDozeMode      = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay          = 0x80;
config->referenceVoltageSource = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode        = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy  = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause       = false;
config->convPauseDelay        = 0U;
config->FIFOWatermark         = 0U;

```

Parameters

- config – Pointer to configuration structure.

```
void LPADC_Deinit(ADC_Type *base)
```

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

```
static inline void LPADC_Enable(ADC_Type *base, bool enable)
```

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

```
static inline void LPADC_DoResetFIFO(ADC_Type *base)
```

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

```
static inline void LPADC_DoResetConfig(ADC_Type *base)
```

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

```
static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)
```

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns

status flags' mask. See to `_lpadc_status_flags`.

```
static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)
```

Clear status flags.

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to `_lpadc_status_flags`.

static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Parameters

- base – LPADC peripheral base address.

Returns

The OR'ed value of `_lpadc_trigger_status_flags`.

static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of `_lpadc_trigger_status_flags`.

static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns

The count of result kept in conversion FIFO.

bool LPADC_GetConvResult(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns

Status whether FIFO entry is valid.

```
void LPADC_GetConvResultBlocking(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

```
void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const
                                lpadc_conv_trigger_config_t *config)
```

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to `lpadc_conv_trigger_config_t`.

```
void LPADC_GetDefaultConvTriggerConfig(lpadc_conv_trigger_config_t *config)
```

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId    = 0U;
config->delayPower         = 0U;
config->priority           = 0U;
config->channelAFIFOSelect = 0U;
config->channelBFIFOSelect = 0U;
config->enableHardwareTrigger = false;
```

Parameters

- config – Pointer to configuration structure.

```
static inline void LPADC_DoSoftwareTrigger(ADC_Type *base, uint32_t triggerIdMask)
```

Do software trigger to conversion command.

Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

```
void LPADC_SetConvCommandConfig(ADC_Type *base, uint32_t commandId, const
                                 lpadc_conv_command_config_t *config)
```

Configure conversion command.

Note: The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

Parameters

- base – LPADC peripheral base address.

- `commandId` – ID for command in command buffer. Typically, the available value range is 1 - 15.
- `config` – Pointer to configuration structure. See to `lpadc_conv_command_config_t`.

`void LPADC_GetDefaultConvCommandConfig(lpadc_conv_command_config_t *config)`

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode      = kLPADC_SampleFullScale;
config->channelBScaleMode    = kLPADC_SampleFullScale;
config->sampleChannelMode    = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber        = 0U;
config->channelBNumber       = 0U;
config->chainedNextCommandNumber = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount            = 0U;
config->hardwareAverageMode   = kLPADC_HardwareAverageCount1;
config->sampleTimeMode       = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode   = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh = 0U;
config->hardwareCompareValueLow  = 0U;
config->conversionResolutionMode = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger      = false;
config->enableChannelB        = false;
```

Parameters

- `config` – Pointer to configuration structure.

`void LPADC_EnableCalibration(ADC_Type *base, bool enable)`

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6-bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

Parameters

- `base` – LPADC peripheral base address.
- `enable` – switcher to the calibration function.

`static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)`

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

Parameters

- `base` – LPADC peripheral base address.
- `value` – Setting offset value.

`status_t LPADC_DoAutoCalibration(ADC_Type *base)`

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional

API including: -LPADC_EnableCalibration(...) -LPADC_LPADC_SetOffsetValue(...) -LPADC_SetConvCommandConfig(...) -LPADC_SetConvTriggerConfig(...)

Parameters

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

```
static inline void LPADC_SetOffsetValue(ADC_Type *base, int16_t value)
```

Set trim value for offset.

Note: For 16-bit conversions, each increment is 1/2 LSB resulting in a programmable offset range of -256 LSB to 255.5 LSB; For 12-bit conversions, each increment is 1/32 LSB resulting in a programmable offset range of -16 LSB to 15.96875 LSB.

Parameters

- base – LPADC peripheral base address.
- value – Offset trim value, is a 10-bit signed value between -512 and 511.

```
static inline void LPADC_GetOffsetValue(ADC_Type *base, int16_t *pValue)
```

Get trim value of offset.

Parameters

- base – LPADC peripheral base address.
- pValue – Pointer to the variable in type of int16_t to store offset value.

```
static inline void LPADC_EnableOffsetCalibration(ADC_Type *base, bool enable)
```

Enable the offset calibration function.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

```
static inline void LPADC_SetOffsetCalibrationMode(ADC_Type *base,
                                                  lpadc_offset_calibration_mode_t mode)
```

Set offset calibration mode.

Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode. see to lpadc_offset_calibration_mode_t.

```
status_t LPADC_DoOffsetCalibration(ADC_Type *base)
```

Do offset calibration.

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_PrepareAutoCalibration(ADC_Type *base)

Prepare auto calibration, LPADC_FinishAutoCalibration has to be called before using the LPADC. LPADC_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

- base – LPADC peripheral base address.

status_t LPADC_FinishAutoCalibration(ADC_Type *base)

Finish auto calibration start with LPADC_PrepareAutoCalibration.

Note: This feature is used for LPADC with CTRL[CALOFSMODE].

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_GetCalibrationValue(ADC_Type *base, lpadc_calibration_value_t *ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

Note: Please note the ADC will be disabled temporary.

Note: This function should be used after finish calibration.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure, this memory block should be always powered on even in low power modes.

status_t LPADC_SetCalibrationValue(ADC_Type *base, const lpadc_calibration_value_t *ptrCalibrationValue)

Set calibration value into ADC calibration registers.

Note: Please note the ADC will be disabled temporary.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure which contains ADC's calibration value.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.9.3.

struct lpadc_config_t

#include <fsl_lpadc.h> LPADC global configuration.

This structure would used to keep the settings for initialization.

Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay * 4) ADCK cycles must result in a longer delay than the analog startup time.

lpadc_reference_voltage_source_t referenceVoltageSource

Selects the voltage reference high used for conversions.

lpadc_power_level_mode_t powerLevelMode

Power Configuration Selection.

lpadc_trigger_priority_policy_t triggerPriorityPolicy

Control how higher priority triggers are handled, see to *lpadc_trigger_priority_policy_t*.

bool enableConvPause

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in “Compare Until True” configuration.

uint32_t convPauseDelay

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (convPauseDelay*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

uint32_t FIFOWatermark

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

struct lpadc_conv_command_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion command.

Public Members

lpadc_sample_scale_mode_t sampleScaleMode

Sample scale mode.

lpadc_sample_scale_mode_t channelBScaleMode

Alternate channel B Scale mode.

lpadc_sample_channel_mode_t sampleChannelMode

Channel sample mode.

uint32_t channelNumber

Channel number, select the channel or channel pair.

uint32_t channelBNumber

Alternate Channel B number, select the channel.

uint32_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the “loopCount” field selects the number of times the selected channel is converted consecutively; when enabled, the “loopCount” field defines how many consecutive channels are converted as part of the command execution.

uint32_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

lpadc_hardware_average_mode_t hardwareAverageMode

Hardware average selection.

lpadc_sample_time_mode_t sampleTimeMode

Sample time selection.

lpadc_hardware_compare_mode_t hardwareCompareMode

Hardware compare selection.

uint32_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

lpadc_conversion_resolution_mode_t conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

```
struct lpadc_conv_trigger_config_t
    #include <fsl_lpadc.h> Define structure to keep the configuration for conversion trigger.
```

Public Members

`uint32_t` targetCommandId
Select the command from command buffer to execute upon detect of the associated trigger event.

`uint32_t` delayPower
Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is $2^{\text{delayPower}}$ ADCK cycles. The available value range is 4-bit.

`uint32_t` priority
Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

`bool` enableHardwareTrigger
Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. The software trigger is always available.

```
struct lpadc_conv_result_t
    #include <fsl_lpadc.h> Define the structure to keep the conversion result.
```

Public Members

`uint32_t` commandIdSource
Indicate the command buffer being executed that generated this result.

`uint32_t` loopCountIndex
Indicate the loop count value during command execution that generated this result.

`uint32_t` triggerIdSource
Indicate the trigger source that initiated a conversion and generated this result.

`uint16_t` convValue
Data result.

```
struct _lpadc_calibration_value
    #include <fsl_lpadc.h> A structure of calibration value.
```

2.40 LPCMP: Low Power Analog Comparator Driver

```
void LPCMP_Init(LPCMP_Type *base, const lpcmp_config_t *config)
```

Initialize the LPCMP.

This function initializes the LPCMP module. The operations included are:

- Enabling the clock for LPCMP module.
- Configuring the comparator.
- Enabling the LPCMP module. Note: For some devices, multiple LPCMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the LPCMPs. Check the chip reference manual for the clock assignment of the LPCMP.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp_config_t” structure.

void LPCMP_Deinit(LPCMP_Type *base)

De-initializes the LPCMP module.

This function de-initializes the LPCMP module. The operations included are:

- Disabling the LPCMP module.
- Disabling the clock for LPCMP module.

This function disables the clock for the LPCMP. Note: For some devices, multiple LPCMP instance shares the same clock gate. In this case, before disabling the clock for the LPCMP, ensure that all the LPCMP instances are not used.

Parameters

- base – LPCMP peripheral base address.

void LPCMP_GetDefaultConfig(*lpcmp_config_t* *config)

Gets an available pre-defined settings for the comparator’s configuration.

This function initializes the comparator configuration structure to these default values:

```
config->enableStopMode      = false;
config->enableOutputPin     = false;
config->enableCmpToDacLink  = false;
config->useUnfilteredOutput = false;
config->enableInvertOutput  = false;
config->hysteresisMode       = kLPCMP_HysteresisLevel0;
config->powerMode            = kLPCMP_LowSpeedPowerMode;
config->functionalSourceClock = kLPCMP_FunctionalClockSource0;
config->plusInputSrc         = kLPCMP_PlusInputSrcMux;
config->minusInputSrc        = kLPCMP_MinusInputSrcMux;
```

Parameters

- config – Pointer to “lpcmp_config_t” structure.

static inline void LPCMP_Enable(LPCMP_Type *base, bool enable)

Enable/Disable LPCMP module.

Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable the module, and “false” means disable the module.

void LPCMP_SetInputChannels(LPCMP_Type *base, uint32_t positiveChannel, uint32_t negativeChannel)

Select the input channels for LPCMP. This function determines which input is selected for the negative and positive mux.

Parameters

- base – LPCMP peripheral base address.
- positiveChannel – Positive side input channel number. Available range is 0-7.
- negativeChannel – Negative side input channel number. Available range is 0-7.

static inline void LPCMP_EnableDMA(LPCMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events. Normally, the LPCMP generates a CPU interrupt if there is a rising/falling event. When DMA support is enabled and the rising/falling interrupt is enabled, the rising/falling event forces a DMA transfer request rather than a CPU interrupt instead.

Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable DMA support, and “false” means disable DMA support.

void LPCMP_SetFilterConfig(LPCMP_Type *base, const *lpcmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “*lpcmp_filter_config_t*” structure.

void LPCMP_SetDACConfig(LPCMP_Type *base, const *lpcmp_dac_config_t* *config)

Configure the internal DAC module.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “*lpcmp_dac_config_t*” structure. If config is “NULL”, disable internal DAC.

static inline void LPCMP_EnableInterrupts(LPCMP_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See “*_lpcmp_interrupt_enable*”.

static inline void LPCMP_DisableInterrupts(LPCMP_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See “*_lpcmp_interrupt_enable*”.

static inline uint32_t LPCMP_GetStatusFlags(LPCMP_Type *base)

Get the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.

Returns

Mask value for the asserted flags. See “*_lpcmp_status_flags*”.

static inline void LPCMP_ClearStatusFlags(LPCMP_Type *base, uint32_t mask)

Clear the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for the flags. See “*_lpcmp_status_flags*”.

static inline void LPCMP_EnableWindowMode(LPCMP_Type *base, bool enable)

Enable/Disable window mode. When any windowed mode is active, COUTA is clocked by the bus clock whenever WINDOW = 1. The last latched value is held when WINDOW = 0. The optionally inverted comparator output COUT_RAW is sampled on every bus clock when WINDOW=1 to generate COUTA.

Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable window mode, and “false” means disable window mode.

void LPCMP_SetWindowControl(LPCMP_Type *base, const *lpcmp_window_control_config_t* *config)

Configure the window control, users can use this API to implement operations on the window, such as inverting the window signal, setting the window closing event (only valid in windowing mode), and setting the COUTA signal after the window is closed (only valid in windowing mode).

Parameters

- base – LPCMP peripheral base address.
- config – Pointer “lpcmp_window_control_config_t” structure.

FSL_LPCMP_DRIVER_VERSION

LPCMP driver version 2.3.2.

enum *_lpcmp_status_flags*

LPCMP status flags mask.

Values:

enumerator kLPCMP_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kLPCMP_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kLPCMP_OutputAssertEventFlag

Return the current value of the analog comparator output. The flag does not support W1C.

enum *_lpcmp_interrupt_enable*

LPCMP interrupt enable/disable mask.

Values:

enumerator kLPCMP_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kLPCMP_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enum *_lpcmp_hysteresis_mode*

LPCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

Values:

enumerator kLPCMP_HysteresisLevel0

The hard block output has level 0 hysteresis internally.

enumerator kLPCMP_HysteresisLevel1

The hard block output has level 1 hysteresis internally.

enumerator kLPCMP_HysteresisLevel2

The hard block output has level 2 hysteresis internally.

enumerator kLPCMP_HysteresisLevel3

The hard block output has level 3 hysteresis internally.

enum _lpcmp_power_mode

LPCMP nano mode.

Values:

enumerator kLPCMP_LowSpeedPowerMode

Low speed comparison mode is selected.

enumerator kLPCMP_HighSpeedPowerMode

High speed comparison mode is selected.

enumerator kLPCMP_NanoPowerMode

Nano power comparator is enabled.

enum _lpcmp_dac_reference_voltage_source

Internal DAC reference voltage source.

Values:

enumerator kLPCMP_VrefSourceVin1

vrefh_int is selected as resistor ladder network supply reference Vin.

enumerator kLPCMP_VrefSourceVin2

vrefh_ext is selected as resistor ladder network supply reference Vin.

enum _lpcmp_couta_signal

Set the COUTA signal value when the window is closed.

Values:

enumerator kLPCMP_COUTASignalNoSet

NO set the COUTA signal value when the window is closed.

enumerator kLPCMP_COUTASignalLow

Set COUTA signal low(0) when the window is closed.

enumerator kLPCMP_COUTASignalHigh

Set COUTA signal high(1) when the window is closed.

enum _lpcmp_close_window_event

Set COUT event, which can close the active window in window mode.

Values:

enumerator kLPCMP_CloseWindowEventNoSet

No Set COUT event, which can close the active window in window mode.

enumerator kLPCMP_CloseWindowEventRisingEdge

Set rising edge COUT signal as COUT event.

enumerator kLPCMP_CloseWindowEventFallingEdge

Set falling edge COUT signal as COUT event.

enumerator kLPCMP_CloseWindowEventBothEdge

Set both rising and falling edge COUT signal as COUT event.

typedef enum *_lpcmp_hysteresis_mode* lpcmp_hysteresis_mode_t
 LPCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

typedef enum *_lpcmp_power_mode* lpcmp_power_mode_t
 LPCMP nano mode.

typedef enum *_lpcmp_dac_reference_voltage_source* lpcmp_dac_reference_voltage_source_t
 Internal DAC reference voltage source.

typedef enum *_lpcmp_couta_signal* lpcmp_couta_signal_t
 Set the COUTA signal value when the window is closed.

typedef enum *_lpcmp_close_window_event* lpcmp_close_window_event_t
 Set COUT event, which can close the active window in window mode.

typedef struct *_lpcmp_filter_config* lpcmp_filter_config_t
 Configure the filter.

typedef struct *_lpcmp_dac_config* lpcmp_dac_config_t
 configure the internal DAC.

typedef struct *_lpcmp_config* lpcmp_config_t
 Configures the comparator.

typedef struct *_lpcmp_window_control_config* lpcmp_window_control_config_t
 Configure the window mode control.

LPCMP_CCR1_COUTA_CFG_MASK

LPCMP_CCR1_COUTA_CFG_SHIFT

LPCMP_CCR1_COUTA_CFG(x)

LPCMP_CCR1_EVT_SEL_CFG_MASK

LPCMP_CCR1_EVT_SEL_CFG_SHIFT

LPCMP_CCR1_EVT_SEL_CFG(x)

struct *_lpcmp_filter_config*
#include <fsl_lpcmp.h> Configure the filter.

Public Members

bool enableSample
 Decide whether to use the external SAMPLE as a sampling clock input.

uint8_t filterSampleCount
 Filter Sample Count. Available range is 1-7; 0 disables the filter.

uint8_t filterSamplePeriod
 Filter Sample Period. The divider to the bus clock. Available range is 0-255. The sampling clock must be at least 4 times slower than the system clock to the comparator. So if enableSample is “false”, filterSamplePeriod should be set greater than 4.

struct *_lpcmp_dac_config*
#include <fsl_lpcmp.h> configure the internal DAC.

Public Members

`bool enableLowPowerMode`

Decide whether to enable DAC low power mode.

`lpcmp_dac_reference_voltage_source_t referenceVoltageSource`

Internal DAC supply voltage reference source.

`uint8_t DACValue`

Value for the DAC Output Voltage. Different devices has different available range, for specific values, please refer to the reference manual.

`struct _lpcmp_config`

`#include <fsl_lpcmp.h>` Configures the comparator.

Public Members

`bool enableOutputPin`

Decide whether to enable the comparator is available in selected pin.

`bool useUnfilteredOutput`

Decide whether to use unfiltered output.

`bool enableInvertOutput`

Decide whether to inverts the comparator output.

`lpcmp_hysteresis_mode_t hysteresisMode`

LPCMP hysteresis mode.

`lpcmp_power_mode_t powerMode`

LPCMP power mode.

`struct _lpcmp_window_control_config`

`#include <fsl_lpcmp.h>` Configure the window mode control.

Public Members

`bool enableInvertWindowSignal`

True: enable invert window signal, False: disable invert window signal.

`lpcmp_couta_signal_t COUTASignal`

Decide whether to define the COUTA signal value when the window is closed.

`lpcmp_close_window_event_t closeWindowEvent`

Decide whether to select COUT event signal edge defines a COUT event to close window.

2.41 LPI2C: Low Power Inter-Integrated Circuit Driver

`void LPI2C_DriverIRQHandler(uint32_t instance)`

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

Parameters

- `instance` – LPI2C instance.

FSL_LPI2C_DRIVER_VERSION

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy

The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle

The slave driver is idle.

enumerator kStatus_LPI2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail

DMA request failed.

enumerator kStatus_LPI2C_Timeout

Timeout polling status flags.

IRQn_Type const kLpi2cIrqs[]

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

lpi2c_master_isr_t s_lpi2cMasterIsr

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

void *s_lpi2cMasterHandle[]

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

uint32_t LPI2C_GetInstance(LPI2C_Type *base)

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- base – The LPI2C peripheral base address.

Returns

LPI2C instance number starting from 0.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.42 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```

masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig         = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns = 0;
masterConfig->pinLowTimeout_ns  = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable = false;
masterConfig->hostRequest.source  = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;

```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *lpi2c_master_config_t*.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

`status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)`

Convert provided flags to status code, and clear any errors if present.

Parameters

- `base` – The LPI2C peripheral base address.
- `status` – Current status flags value that will be checked.

Return values

- `kStatus_Success` –
- `kStatus_LPI2C_PinLowTimeout` –
- `kStatus_LPI2C_ArbitrationLost` –
- `kStatus_LPI2C_Nak` –
- `kStatus_LPI2C_FifoError` –

`status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)`

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `kStatus_Success` –
- `kStatus_LPI2C_Busy` –

`static inline void LPI2C_MasterReset(LPI2C_Type *base)`

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

`static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)`

Enables or disables the LPI2C module as master.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as master.

`static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)`

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_master_flags`

Parameters

- `base` – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

[_lpi2c_master_flags](#).

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of [_lpi2c_master_flags](#) enumerators OR'd together. You may pass the result of a previous call to [LPI2C_MasterGetStatusFlags\(\)](#).

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except [kLPI2C_MasterBusyFlag](#) and [kLPI2C_MasterBusBusyFlag](#) can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See [_lpi2c_master_flags](#) for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except [kLPI2C_MasterBusyFlag](#) and [kLPI2C_MasterBusBusyFlag](#) can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See [_lpi2c_master_flags](#) for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)
```

Enables or disables LPI2C master DMA requests.

Parameters

- `base` – The LPI2C peripheral base address.
- `enableTx` – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- `enableRx` – Enable flag for receive DMA request. Pass true for enable, false for disable.

```
static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)
```

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

```
static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)
```

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

```
static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)
```

Sets the watermarks for LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txWords` – Transmit FIFO watermark value in words. The `kLPI2C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO is equal or less than `txWords`. Writing a value equal or greater than the FIFO size is truncated.
- `rxWords` – Receive FIFO watermark value in words. The `kLPI2C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO is greater than `rxWords`. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t
                             baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

```
static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)
```

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the `address` parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

```
static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address,
                                                  lpi2c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

`status_t` LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to `txSize` number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or over run.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t` LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.

- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterStop(LPI2C_Type *base)`

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, lpi2c_master_transfer_t *transfer)`

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The LPI2C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, lpi2c_master_handle_t *handle, lpi2c_master_transfer_callback_t callback, void *userData)`

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *size_t* *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags

Errors to check for.

enum _lpi2c_direction

Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write

Master transmit.

enumerator kLPI2C_Read

Master receive.

enum _lpi2c_master_pin_config

LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source

LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Values:

enumerator kLPI2C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag
Don't send a stop condition.

```
typedef enum _lpi2c_direction lpi2c_direction_t
```

Direction of master and slave transfers.

```
typedef enum _lpi2c_master_pin_config lpi2c_master_pin_config_t
```

LPI2C pin configuration.

```
typedef enum _lpi2c_host_request_source lpi2c_host_request_source_t
```

LPI2C master host request selection.

```
typedef enum _lpi2c_host_request_polarity lpi2c_host_request_polarity_t
```

LPI2C master host request pin polarity configuration.

```
typedef struct _lpi2c_master_config lpi2c_master_config_t
```

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_data_match_config_mode lpi2c_data_match_config_mode_t
```

LPI2C master data match configuration modes.

```
typedef struct _lpi2c_match_config lpi2c_data_match_config_t
```

LPI2C master data match configuration structure.

```
typedef struct _lpi2c_master_transfer lpi2c_master_transfer_t
```

LPI2C master descriptor of the transfer.

```
typedef struct _lpi2c_master_handle lpi2c_master_handle_t
```

LPI2C master handle of the transfer.

```
typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)
```

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to `LPI2C_MasterTransferCreateHandle()`.

Param base

The LPI2C peripheral base address.

Param handle

Pointer to the LPI2C master driver handle.

Param completionStatus

Either `kStatus_Success` or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members**bool** enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest

Host request options.

struct *_lpi2c_match_config**#include <fsl_lpi2c.h>* LPI2C master data match configuration structure.**Public Members***lpi2c_data_match_config_mode_t* matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32_t match0

Match value 0.

uint32_t match1

Match value 1.

struct *_lpi2c_master_transfer**#include <fsl_lpi2c.h>* Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the `LPI2C_MasterTransferNonBlocking()` API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration `_lpi2c_master_transfer_flags` for available options. Set to 0 or `kLPI2C_TransferDefaultFlag` for normal transfers.

uint16_t slaveAddress

The 7-bit slave address.

lpi2c_direction_t direction

Either `kLPI2C_Read` or `kLPI2C_Write`.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

struct `_lpi2c_master_handle`

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint16_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: `Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]`

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

struct `hostRequest`

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.43 LPI2C Master DMA Driver

```
void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
    edma_handle_t *rxDmaHandle, edma_handle_t
    *txDmaHandle, lpi2c_master_edma_transfer_callback_t
    callback, void *userData)
```

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – **[out]** Pointer to the LPI2C master driver handle.
- *rxDmaHandle* – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
    lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

Return values

- *kStatus_Success* – The transaction was started successfully.
- *kStatus_LPI2C_Busy* – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                           *handle, size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

```
status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                        *handle)
```

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

```
typedef struct _lpi2c_master_edma_handle lpi2c_master_edma_handle_t
LPI2C master EDMA handle of the transfer.
```

```
typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)
```

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Param base

The LPI2C peripheral base address.

Param handle

Handle associated with the completed transfer.

Param completionStatus

Either kStatus_Success or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_master_edma_handle
    #include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

```
LPI2C_Type *base
    LPI2C base pointer.

bool isBusy
    Transfer state machine current state.

uint8_t nbytes
    eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[20]
    LPI2C command sequence. When all 10 command words are used:
    Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] +
    receive&Size[4 words]

lpi2c_master_transfer_t transfer
    Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback
    Callback function pointer.

void *userData
    Application data passed to callback.

edma_handle_t *rx
    Handle for receive DMA channel.

edma_handle_t *tx
    Handle for transmit DMA channel.

edma_tcd_t tcds[3]
    Software TCD. Three are allocated to provide enough room to align to 32-bytes.
```

2.44 LPI2C Slave Driver

```
void LPI2C_SlaveGetDefaultConfig(lpi2c_slave_config_t *slaveConfig)
    Provides a default configuration for the LPI2C slave peripheral.
```

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave      = true;
slaveConfig->address0         = 0U;
slaveConfig->address1         = 0U;
slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable = true;
slaveConfig->filterEnable     = true;
slaveConfig->enableGeneralCall = false;
slaveConfig->sclStall.enableAck = false;
slaveConfig->sclStall.enableTx  = true;
slaveConfig->sclStall.enableRx  = true;
```

(continues on next page)

(continued from previous page)

```

slaveConfig->sclStall.enableAddress = true;
slaveConfig->ignoreAck             = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns = 0;
slaveConfig->sclGlitchFilterWidth_ns = 0;
slaveConfig->dataValidDelay_ns      = 0;
slaveConfig->clockHoldTime_ns       = 0;

```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with `LPI2C_SlaveInit()`. Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure that is set to default values. Refer to `lpi2c_slave_config_t`.

```
void LPI2C_SlaveInit(LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t
                    sourceClock_Hz)
```

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `LPI2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

```
void LPI2C_SlaveDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveReset(LPI2C_Type *base)
```

Performs a software reset of the LPI2C slave peripheral.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)
```

Enables or disables the LPI2C module as slave.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as slave.

```
static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)
```

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

[_lpi2c_slave_flags](#)

Parameters

- `base` – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)
```

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- `kLPI2C_SlaveRepeatedStartDetectFlag`
- `kLPI2C_SlaveStopDetectFlag`
- `kLPI2C_SlaveBitErrFlag`
- `kLPI2C_SlaveFifoErrFlag`

Attempts to clear other flags has no effect.

See also:

[_lpi2c_slave_flags](#).

Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_SlaveGetStatusFlags()`.

```
static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Enables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Disables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)
```

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)
```

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

```
static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

```
static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)
```

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.
- ackOrNack – Pass true for an ACK or false for a NAK.

```
static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)
```

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL, false will disable ACKSTALL.

```
static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)
```

Returns the slave address sent by the I2C master.

This function should only be called if the `kLPI2C_SlaveAddressValidFlag` is asserted.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

```
status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)
```

Performs a polling send transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `actualTxSize` – **[out]**

Returns

Error or success status returned by API.

```
status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)
```

Performs a polling receive transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `actualRxSize` – **[out]**

Returns

Error or success status returned by API.

```
void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, lpi2c_slave_handle_t *handle,  
                                     lpi2c_slave_transfer_callback_t callback, void *userData)
```

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_SlaveTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – **[out]** Pointer to the LPI2C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

```
status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, lpi2c_slave_handle_t *handle,
                                       uint32_t eventMask)
```

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kLPI2C_SlaveTransmitEvent* and *kLPI2C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to *lpi2c_slave_handle_t* structure which stores the transfer state.
- *eventMask* – Bit mask formed by OR'ing together *lpi2c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kLPI2C_SlaveAllEvents* to enable all events.

Return values

- *kStatus_Success* – Slave transfers were successfully started.
- *kStatus_LPI2C_Busy* – Slave transfers have already been started on this handle.

```
status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, lpi2c_slave_handle_t *handle, size_t
                                     *count)
```

Gets the slave transfer status during a non-blocking transfer.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to *i2c_slave_handle_t* structure.
- *count* – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- *kStatus_Success* –
- *kStatus_NoTransferInProgress* –

```
void LPI2C_SlaveTransferAbort(LPI2C_Type *base, lpi2c_slave_handle_t *handle)
```

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- *base* – The LPI2C peripheral base address.

- `handle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, lpi2c_slave_handle_t *handle)`

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`enum _lpi2c_slave_flags`

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- `kLPI2C_SlaveRepeatedStartDetectFlag`
- `kLPI2C_SlaveStopDetectFlag`
- `kLPI2C_SlaveBitErrFlag`
- `kLPI2C_SlaveFifoErrFlag`

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator `kLPI2C_SlaveTxReadyFlag`

Transmit data flag

enumerator `kLPI2C_SlaveRxReadyFlag`

Receive data flag

enumerator `kLPI2C_SlaveAddressValidFlag`

Address valid flag

enumerator `kLPI2C_SlaveTransmitAckFlag`

Transmit ACK flag

enumerator `kLPI2C_SlaveRepeatedStartDetectFlag`

Repeated start detect flag

enumerator `kLPI2C_SlaveStopDetectFlag`

Stop detect flag

enumerator `kLPI2C_SlaveBitErrFlag`

Bit error flag

enumerator `kLPI2C_SlaveFifoErrFlag`

FIFO error flag

enumerator `kLPI2C_SlaveAddressMatch0Flag`

Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag

Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag

General call flag

enumerator kLPI2C_SlaveBusyFlag

Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag

Bus busy flag

enumerator kLPI2C_SlaveClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags

Errors to check for.

enum _lpi2c_slave_address_match

LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0

Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `LPI2C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

LPI2C slave handle structure.

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `LPI2C_SlaveSetCallback()` function after you have created a handle.

Param base

Base address for the LPI2C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

struct *_lpi2c_slave_config*

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the `LPI2C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Enable slave mode.

uint8_t address0

Slave's 7-bit address.

uint8_t address1

Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode

Address matching options.

bool filterDozeEnable

Enable digital glitch filter in doze mode.

bool filterEnable

Enable digital glitch filter.

bool enableGeneralCall

Enable general call address matching.

struct *_lpi2c_slave_config* sclStall

SCL stall enable options.

bool ignoreAck

Continue transfers after a NACK is detected.

bool enableReceivedAddressRead

Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns

Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns

Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns

Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns

Width in nanoseconds of the clock hold time.

struct *_lpi2c_slave_transfer*

#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event

Reason the callback is being invoked.

uint8_t receivedAddress

Matching address send by master.

uint8_t *data

Transfer buffer

size_t dataSize

Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct *_lpi2c_slave_handle*

#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer

LPI2C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck

Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

2.45 LPIT: Low-Power Interrupt Timer

enum `_lpit_chnl`

List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

Values:

enumerator `kLPIT_Chnl_0`
LPIT channel number 0

enumerator `kLPIT_Chnl_1`
LPIT channel number 1

enumerator `kLPIT_Chnl_2`
LPIT channel number 2

enumerator `kLPIT_Chnl_3`
LPIT channel number 3

enum `_lpit_timer_modes`

Mode options available for the LPIT timer.

Values:

enumerator `kLPIT_PeriodicCounter`
Use the all 32-bits, counter loads and decrements to zero

enumerator `kLPIT_DualPeriodicCounter`
Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement

enumerator `kLPIT_TriggerAccumulator`
Counter loads on first trigger and decrements on each trigger

enumerator `kLPIT_InputCapture`
Counter loads with `0xFFFFFFFF`, decrements to zero. It stores the inverse of the current value when a input trigger is detected

enum `_lpit_trigger_select`

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Values:

enumerator `kLPIT_Trigger_TimerChn0`
Channel 0 is selected as a trigger source

enumerator `kLPIT_Trigger_TimerChn1`
Channel 1 is selected as a trigger source

enumerator `kLPIT_Trigger_TimerChn2`
Channel 2 is selected as a trigger source

enumerator `kLPIT_Trigger_TimerChn3`
Channel 3 is selected as a trigger source

enumerator `kLPIT_Trigger_TimerChn4`
Channel 4 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn5

Channel 5 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn6

Channel 6 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn7

Channel 7 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn8

Channel 8 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn9

Channel 9 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn10

Channel 10 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn11

Channel 11 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn12

Channel 12 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn13

Channel 13 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn14

Channel 14 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn15

Channel 15 is selected as a trigger source

enum _lpit_trigger_source

Trigger source options available.

Values:

enumerator kLPIT_TriggerSource_External

Use external trigger input

enumerator kLPIT_TriggerSource_Internal

Use internal trigger

enum _lpit_interrupt_enable

List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerInterruptEnable

Channel 0 Timer interrupt

enumerator kLPIT_Channel1TimerInterruptEnable

Channel 1 Timer interrupt

enumerator kLPIT_Channel2TimerInterruptEnable

Channel 2 Timer interrupt

enumerator kLPIT_Channel3TimerInterruptEnable

Channel 3 Timer interrupt

enum `_lpit_status_flags`

List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator `kLPIT_Channel0TimerFlag`
Channel 0 Timer interrupt flag

enumerator `kLPIT_Channel1TimerFlag`
Channel 1 Timer interrupt flag

enumerator `kLPIT_Channel2TimerFlag`
Channel 2 Timer interrupt flag

enumerator `kLPIT_Channel3TimerFlag`
Channel 3 Timer interrupt flag

typedef enum `_lpit_chnl` `lpit_chnl_t`

List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

typedef enum `_lpit_timer_modes` `lpit_timer_modes_t`

Mode options available for the LPIT timer.

typedef enum `_lpit_trigger_select` `lpit_trigger_select_t`

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

typedef enum `_lpit_trigger_source` `lpit_trigger_source_t`

Trigger source options available.

typedef enum `_lpit_interrupt_enable` `lpit_interrupt_enable_t`

List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

typedef enum `_lpit_status_flags` `lpit_status_flags_t`

List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

typedef struct `_lpit_chnl_params` `lpit_chnl_params_t`

Structure to configure the channel timer.

typedef struct `_lpit_config` `lpit_config_t`

LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the `LPIT_GetDefaultConfig()` function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

FSL_LPIT_DRIVER_VERSION

Version 2.1.2

LPIT_RESET_STATE_DELAY

Delay used in LPIT_Reset.

The macro value should be larger than $4 * \text{core clock} / \text{LPIT peripheral clock}$.

void LPIT_Init(LPIT_Type *base, const *lpit_config_t* *config)

Ungates the LPIT clock and configures the peripheral for a basic operation.

This function issues a software reset to reset all channels and registers except the Module Control register.

Note: This API should be called at the beginning of the application using the LPIT driver.

Parameters

- base – LPIT peripheral base address.
- config – Pointer to the user configuration structure.

void LPIT_Deinit(LPIT_Type *base)

Disables the module and gates the LPIT clock.

Parameters

- base – LPIT peripheral base address.

void LPIT_GetDefaultConfig(*lpit_config_t* *config)

Fills in the LPIT configuration structure with default settings.

The default values are:

```
config->enableRunInDebug = false;
config->enableRunInDoze = false;
```

Parameters

- config – Pointer to the user configuration structure.

status_t LPIT_SetupChannel(LPIT_Type *base, *lpit_chnl_t* channel, const *lpit_chnl_params_t* *chnlSetup)

Sets up an LPIT channel based on the user's preference.

This function sets up the operation mode to one of the options available in the enumeration *lpit_timer_modes_t*. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

- base – LPIT peripheral base address.
- channel – Channel that is being configured.
- chnlSetup – Configuration parameters.

static inline void LPIT_EnableInterrupts(LPIT_Type *base, uint32_t mask)

Enables the selected PIT interrupts.

Parameters

- base – LPIT peripheral base address.

- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `lpit_interrupt_enable_t`

```
static inline void LPIT_DisableInterrupts(LPIT_Type *base, uint32_t mask)
```

Disables the selected PIT interrupts.

Parameters

- `base` – LPIT peripheral base address.
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `lpit_interrupt_enable_t`

```
static inline uint32_t LPIT_GetEnabledInterrupts(LPIT_Type *base)
```

Gets the enabled LPIT interrupts.

Parameters

- `base` – LPIT peripheral base address.

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `lpit_interrupt_enable_t`

```
static inline uint32_t LPIT_GetStatusFlags(LPIT_Type *base)
```

Gets the LPIT status flags.

Parameters

- `base` – LPIT peripheral base address.

Returns

The status flags. This is the logical OR of members of the enumeration `lpit_status_flags_t`

```
static inline void LPIT_ClearStatusFlags(LPIT_Type *base, uint32_t mask)
```

Clears the LPIT status flags.

Parameters

- `base` – LPIT peripheral base address.
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `lpit_status_flags_t`

```
static inline void LPIT_SetTimerPeriod(LPIT_Type *base, lpit_chnl_t channel, uint32_t ticks)
```

Sets the timer period in units of count.

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: User can call the utility macros provided in `fsl_common.h` to convert to ticks.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.
- `ticks` – Timer period in units of ticks.

```
static inline void LPIT_SetTimerValue(LPIT_Type *base, lpit_chnl_t channel, uint32_t ticks)
```

Sets the timer period in units of count.

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will

then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

Note: Set TVAL register to 0 or 1 is invalid in compare mode.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.
- `ticks` – Timer period in units of ticks.

```
static inline uint32_t LPIT_GetCurrentTimerCount(LPIT_Type *base, lpit_chnl_t channel)
```

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in `fsl_common.h` to convert ticks to microseconds or milliseconds.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

Returns

Current timer counting value in ticks.

```
static inline void LPIT_StartTimer(LPIT_Type *base, lpit_chnl_t channel)
```

Starts the timer counting.

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

```
static inline void LPIT_StopTimer(LPIT_Type *base, lpit_chnl_t channel)
```

Stops the timer counting.

Parameters

- `base` – LPIT peripheral base address.
- `channel` – Timer channel number.

```
static void LPIT_ResetStateDelay(void)
```

Short wait for LPIT state reset.

After clear or set `LPIT_EN`, there should be delay longer than 4 LPIT functional clock.

Parameters

- `count` – Delay count.

```
static inline void LPIT_Reset(LPIT_Type *base)
```

Performs a software reset on the LPIT module.

This resets all channels and registers except the Module Control Register.

Parameters

- base – LPIT peripheral base address.

struct `_lpit_chnl_params`

#include <fsl_lpit.h> Structure to configure the channel timer.

Public Members

bool chainChannel

true: Timer chained to previous timer; false: Timer not chained

lpit_timer_modes_t timerMode

Timers mode of operation.

lpit_trigger_select_t triggerSelect

Trigger selection for the timer

lpit_trigger_source_t triggerSource

Decides if we use external or internal trigger.

bool enableReloadOnTrigger

true: Timer reloads when a trigger is detected; false: No effect

bool enableStopOnTimeout

true: Timer will stop after timeout; false: does not stop after timeout

bool enableStartOnTrigger

true: Timer starts when a trigger is detected; false: decrement immediately

struct `_lpit_config`

#include <fsl_lpit.h> LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the `LPIT_GetDefaultConfig()` function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

bool enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

bool enableRunInDoze

true: Timers run in doze mode; false: Timers stop in doze mode

2.46 LPSPI: Low Power Serial Peripheral Interface

2.47 LPSPI Peripheral driver

void `LPSPI_MasterInit(LPSPI_Type *base, const lpspi_master_config_t *masterConfig, uint32_t srcClock_Hz)`

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure `lpspi_master_config_t`.

- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the *lpspi_master_config_t* structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to *lpspi_master_config_t* structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure *lpspi_slave_config_t*.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the *lpspi_slave_config_t* structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to *lpspi_slave_config_t* structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

```
static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)
```

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

```
static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)
```

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

```
static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)
```

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

```
static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)
```

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

```
static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)
```

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

```
static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)
```

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the receive FIFO.

```
static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)
```

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type `_lpspi_flags`.

```
static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)
```

```
static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)
```

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

```
static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)
```

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

```
static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)
```

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

```
static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

```
static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

```
bool LPSPI_CheckTransferArgument(LPSPI_Type *base, lpspi_transfer_t *transfer, bool isEdma)
```

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

```
static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, lpspi_master_slave_mode_t mode)
```

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

```
static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, lpspi_which_pcs_t select)
```

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

```
static inline void LPSPI_SetPCSContinuous(LPSPI_Type *base, bool IsContinuous)
```

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinuous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

```
static inline bool LPSPI_IsMaster(LPSPI_Type *base)
```

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

```
static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)
```

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

```
static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)
```

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)
```

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFG1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum `_lpspi_pcs_polarity`.

```
static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)
```

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame

size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

```
uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)
```

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

```
void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, lpspi_delay_type_t whichDelay)
```

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type *lpspi_delay_type_t*.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.

- `scaler` – The 8-bit delay value 0x00 to 0xFF (255).
- `whichDelay` – The desired delay to configure, must be of type `lpspi_delay_type_t`.

```
uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec,
                                   lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)
```

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the `delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler)`.

Parameters

- `base` – LPSPI peripheral address.
- `delayTimeInNanoSec` – The desired delay value in nano-seconds.
- `whichDelay` – The desired delay to configuration, which must be of type `lpspi_delay_type_t`.
- `srcClock_Hz` – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

```
static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)
```

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- `base` – LPSPI peripheral address.
- `data` – The data word to be sent.

```
static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)
```

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- `base` – LPSPI peripheral address.

Returns

The data read from the data buffer.

```
void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)
```

Set up the dummy data.

Parameters

- `base` – LPSPI peripheral address.
- `dummyData` – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

```
void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, lpspi_master_handle_t *handle,  
                                     lpspi_master_transfer_callback_t callback, void  
                                     *userData)
```

Initializes the LPSPI master handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- `base` – LPSPI peripheral address.
- `handle` – LPSPI handle pointer to `lpspi_master_handle_t`.
- `callback` – DSPI callback.
- `userData` – callback function parameter.

```
status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, lpspi_transfer_t *transfer)
```

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- `base` – LPSPI peripheral address.
- `transfer` – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

```
status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, lpspi_master_handle_t *handle,  
                                         lpspi_transfer_t *transfer)
```

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- `base` – LPSPI peripheral address.
- `handle` – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, `size_t` *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of `status_t`.

`void` LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

`void` LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

`void` LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, `void` *userData)

Initializes the LPSPI slave handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to `lpspi_slave_handle_t`.
- callback – DSPI callback.
- userData – callback function parameter.

`status_t` LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of `status_t`.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns

true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag

Data Match flag

enumerator kLPSPI_ModuleBusyFlag

Module Busy flag

enumerator kLPSPI_AllStatusFlag

Used for clearing all w1c status flags

enum _lpspi_interrupt_enable

LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable

Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable

Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable

Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable

Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable

Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable

Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable

Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable

Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable

All above interrupts enable.

enum _lpspi_dma_enable

LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable

Receive data DMA enable

enum _lpspi_master_slave_mode

LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master

LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave

LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0

PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity
LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity
LPSPI clock polarity configuration.

Values:

enumerator kLPSPI_ClockPolarityActiveHigh
CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow
CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase
LPSPI clock phase configuration.

Values:

enumerator kLPSPI_ClockPhaseFirstEdge
CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge
CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum `_lpspi_shift_direction`

LPSPI data shifter direction options.

Values:

enumerator `kLPSPI_MsbFirst`

Data transfers start with most significant bit.

enumerator `kLPSPI_LsbFirst`

Data transfers start with least significant bit.

enum `_lpspi_host_request_select`

LPSPI Host Request select configuration.

Values:

enumerator `kLPSPI_HostReqExtPin`

Host Request is an ext pin.

enumerator `kLPSPI_HostReqInternalTrigger`

Host Request is an internal trigger.

enum `_lpspi_match_config`

LPSPI Match configuration options.

Values:

enumerator `kLPSI_MatchDisabled`

LPSPI Match Disabled.

enumerator `kLPSI_1stWordEqualsM0orM1`

LPSPI Match Enabled.

enumerator `kLPSI_AnyWordEqualsM0orM1`

LPSPI Match Enabled.

enumerator `kLPSI_1stWordEqualsM0and2ndWordEqualsM1`

LPSPI Match Enabled.

enumerator `kLPSI_AnyWordEqualsM0andNxtWordEqualsM1`

LPSPI Match Enabled.

enumerator `kLPSI_1stWordAndM1EqualsM0andM1`

LPSPI Match Enabled.

enumerator `kLPSI_AnyWordAndM1EqualsM0andM1`

LPSPI Match Enabled.

enum `_lpspi_pin_config`

LPSPI pin (SDO and SDI) configuration.

Values:

enumerator `kLPSPI_SdiInSdoOut`

LPSPI SDI input, SDO output.

enumerator `kLPSPI_SdiInSdiOut`

LPSPI SDI input, SDI output.

enumerator `kLPSPI_SdoInSdoOut`

LPSPI SDO input, SDO output.

enumerator `kLPSPI_SdoInSdiOut`

LPSPI SDO input, SDI output.

enum `_lpspi_data_out_config`

LPSPI data output configuration.

Values:

enumerator `kLpspiDataOutRetained`

Data out retains last value when chip select is de-asserted

enumerator `kLpspiDataOutTristate`

Data out is tristated when chip select is de-asserted

enum `_lpspi_transfer_width`

LPSPI transfer width configuration.

Values:

enumerator `kLPSPISingleBitXfer`

1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator `kLPSPITwoBitXfer`

2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator `kLPSPIFourBitXfer`

4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum `_lpspi_delay_type`

LPSPI delay type selection.

Values:

enumerator `kLPSPIPcsToSck`

PCS-to-SCK delay.

enumerator `kLPSPILastSckToPcs`

Last SCK edge to PCS delay.

enumerator `kLPSPIBetweenTransfer`

Delay between transfers.

enum `_lpspi_transfer_config_flag_for_master`

Use this enumeration for LPSPi master transfer configFlags.

Values:

enumerator `kLPSPIMasterPcs0`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS0 signal

enumerator `kLPSPIMasterPcs1`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS1 signal

enumerator `kLPSPIMasterPcs2`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS2 signal

enumerator `kLPSPIMasterPcs3`

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS3 signal

enumerator `kLPSPIMasterPcsContinuous`

Is PCS signal continuous

enumerator `kLPSPIMasterByteSwap`

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).

- i. If you set `bitPerFrame = 8` , no matter the `kLPSPIMasterByteSwap` you flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

enum `_lpspi_transfer_config_flag_for_slave`

Use this enumeration for LPSPI slave transfer configFlags.

Values:

enumerator `kLPSPI_SlavePcs0`

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS0 signal

enumerator `kLPSPI_SlavePcs1`

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS1 signal

enumerator `kLPSPI_SlavePcs2`

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS2 signal

enumerator `kLPSPI_SlavePcs3`

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS3 signal

enumerator `kLPSPI_SlaveByteSwap`

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).

- i. If you set bitPerFrame = 8 , no matter the `kLPSPI_SlaveByteSwap` flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the `kLPSPI_SlaveByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPI_SlaveByteSwap` flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the `kLPSPI_SlaveByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPI_SlaveByteSwap` flag.

enum `_lpspi_transfer_state`

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator `kLPSPI_Idle`

Nothing in the transmitter/receiver.

enumerator `kLPSPI_Busy`

Transfer queue is not finished.

enumerator `kLPSPI_Error`

Transfer error.

typedef enum `_lpspi_master_slave_mode` `lpspi_master_slave_mode_t`

LPSPI master or slave mode configuration.

typedef enum `_lpspi_which_pcs_config` `lpspi_which_pcs_t`

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum `_lpspi_pcs_polarity_config` `lpspi_pcs_polarity_config_t`

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

```
typedef enum _lpspi_clock_polarity lpspi_clock_polarity_t
    LPSPI clock polarity configuration.
typedef enum _lpspi_clock_phase lpspi_clock_phase_t
    LPSPI clock phase configuration.
typedef enum _lpspi_shift_direction lpspi_shift_direction_t
    LPSPI data shifter direction options.
typedef enum _lpspi_host_request_select lpspi_host_request_select_t
    LPSPI Host Request select configuration.
typedef enum _lpspi_match_config lpspi_match_config_t
    LPSPI Match configuration options.
typedef enum _lpspi_pin_config lpspi_pin_config_t
    LPSPI pin (SDO and SDI) configuration.
typedef enum _lpspi_data_out_config lpspi_data_out_config_t
    LPSPI data output configuration.
typedef enum _lpspi_transfer_width lpspi_transfer_width_t
    LPSPI transfer width configuration.
typedef enum _lpspi_delay_type lpspi_delay_type_t
    LPSPI delay type selection.
typedef struct _lpspi_master_config lpspi_master_config_t
    LPSPI master configuration structure.
typedef struct _lpspi_slave_config lpspi_slave_config_t
    LPSPI slave configuration structure.
typedef struct _lpspi_master_handle lpspi_master_handle_t
    Forward declaration of the _lpspi_master_handle typedefs.
typedef struct _lpspi_slave_handle lpspi_slave_handle_t
    Forward declaration of the _lpspi_slave_handle typedefs.
typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t
*handle, status_t status, void *userData)
    Master completion callback function pointer type.
    Param base
        LPSPI peripheral address.
    Param handle
        Pointer to the handle for the LPSPI master.
    Param status
        Success or error code describing whether the transfer is completed.
    Param userData
        Arbitrary pointer-dataSized value passed from the application.
typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle,
status_t status, void *userData)
    Slave completion callback function pointer type.
    Param base
        LPSPI peripheral address.
    Param handle
        Pointer to the handle for the LPSPI slave.
```

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef struct lpspi_transfer lpspi_transfer_t
```

LPSPI master/slave transfer structure.

```
volatile uint8_t g_lpspiDummyData[]
```

Global variable for dummy data value setting.

```
LPSPI_DUMMY_DATA
```

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
LPSPI_MASTER_PCS_SHIFT
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_PCS_MASK
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_SHIFT
```

LPSPI slave PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_MASK
```

LPSPI slave PCS shift macro , internal used.

```
struct lpspi_master_config
```

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

```
uint32_t baudRate
```

Baud Rate for LPSPI.

```
uint32_t bitsPerFrame
```

Bits per frame, minimum 8, maximum 4096.

```
lpspi_clock_polarity_t cpol
```

Clock polarity.

```
lpspi_clock_phase_t cpha
```

Clock phase.

```
lpspi_shift_direction_t direction
```

MSB or LSB data shift direction.

```
uint32_t pcsToSckDelayInNanoSec
```

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

```
uint32_t lastSckToPcsDelayInNanoSec
```

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi_slave_config

#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct __lpspi_transfer

#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize

Transfer bytes.

uint32_t configFlags

Transfer transfer configuration flags. Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

struct `_lpspi_master_handle`

#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame

Number of bytes in each frame

volatile uint16_t frameSize

Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes

Read RDR register remaining times.

uint32_t totalByteCount
 Number of transfer bytes

uint32_t txBuffIfNull
 Used if the txData is NULL.

volatile uint8_t state
 LPSPI transfer state , `_lpspi_transfer_state`.

lpspi_master_transfer_callback_t callback
 Completion callback.

void *userData
 Callback user data.

struct `_lpspi_slave_handle`
#include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
 A flag that whether should byte swap.

volatile uint8_t fifoSize
 FIFO dataSize.

volatile uint8_t rxWatermark
 Rx watermark.

volatile uint8_t bytesEachWrite
 Bytes for each write TDR.

volatile uint8_t bytesEachRead
 Bytes for each read RDR.

const uint8_t *volatile txData
 Send buffer.

uint8_t *volatile rxData
 Receive buffer.

volatile size_t txRemainingByteCount
 Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
 Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
 Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
 Read RDR register remaining times.

uint32_t totalByteCount
 Number of transfer bytes

volatile uint8_t state
 LPSPI transfer state , `_lpspi_transfer_state`.

volatile uint32_t errorCount
 Error count for slave transfer.

lpspi_slave_transfer_callback_t callback

Completion callback.

void *userData

Callback user data.

2.48 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION

LPSPI EDMA driver version.

DMA_MAX_TRANSFER_COUNT

DMA max transfer size.

typedef struct *lpspi_master_edma_handle* lpspi_master_edma_handle_t

Forward declaration of the *lpspi_master_edma_handle* typedefs.

typedef struct *lpspi_slave_edma_handle* lpspi_slave_edma_handle_t

Forward declaration of the *lpspi_slave_edma_handle* typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base

LPSPI peripheral base address.

Param handle

Pointer to the handle for the LPSPI master.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base

LPSPI peripheral base address.

Param handle

Pointer to the handle for the LPSPI slave.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for `edmaRxRegToRxDataHandle`.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – LPSPI handle pointer to `lpspi_master_edma_handle_t`.
- `callback` – LPSPI callback.
- `userData` – callback function parameter.
- `edmaRxRegToRxDataHandle` – `edmaRxRegToRxDataHandle` pointer to `edma_handle_t`.
- `edmaTxDataToTxRegHandle` – `edmaTxDataToTxRegHandle` pointer to `edma_handle_t`.

`status_t` LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `transfer` – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, `uint32_t` configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `configFlags` – transfer configuration flags. `_lpspi_transfer_config_flag_for_master`.

Return values

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.

Returns

Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.
- transfer – pointer to *lpspi_transfer_t* structure, config field is not used.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.
- kStatus_InvalidArgument – The transfer structure is invalid.

Returns

Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *size_t* *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns

status of *status_t*.

```
void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t
                                         *handle, lpspi_slave_edma_transfer_callback_t
                                         callback, void *userData, edma_handle_t
                                         *edmaRxRegToRxDataHandle, edma_handle_t
                                         *edmaTxDataToTxRegHandle)
```

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for *edmaRxRegToRxDataHandle* and Tx DMAMUX source for *edmaTxDataToTxRegHandle*. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for *edmaRxRegToRxDataHandle*.

Parameters

- *base* – LPSPI peripheral base address.
- *handle* – LPSPI handle pointer to *lpspi_slave_edma_handle_t*.
- *callback* – LPSPI callback.
- *userData* – callback function parameter.
- *edmaRxRegToRxDataHandle* – *edmaRxRegToRxDataHandle* pointer to *edma_handle_t*.
- *edmaTxDataToTxRegHandle* – *edmaTxDataToTxRegHandle* pointer to *edma_handle_t*.

```
status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle,
                                  lpspi_transfer_t *transfer)
```

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of *bytesPerFrame* if *bytesPerFrame* is less than or equal to 4. For *bytesPerFrame* greater than 4: The transfer data size should be equal to *bytesPerFrame* if the *bytesPerFrame* is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of *bytesPerFrame*.

Parameters

- *base* – LPSPI peripheral base address.
- *handle* – pointer to *lpspi_slave_edma_handle_t* structure which stores the transfer state.
- *transfer* – pointer to *lpspi_transfer_t* structure.

Returns

status of *status_t*.

```
void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)
LPSPI slave aborts a transfer which is using eDMA.
```

This function aborts a transfer which is using eDMA.

Parameters

- *base* – LPSPI peripheral base address.

- `handle` – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.

`status_t` LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, `size_t` *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the eDMA transaction.

Returns

status of `status_t`.

`struct _lpspi_master_edma_handle`

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

`volatile bool` `isPcsContinuous`

Is PCS continuous in transfer.

`volatile bool` `isByteSwap`

A flag that whether should byte swap.

`volatile uint8_t` `fifoSize`

FIFO dataSize.

`volatile uint8_t` `rxWatermark`

Rx watermark.

`volatile uint8_t` `bytesEachWrite`

Bytes for each write TDR.

`volatile uint8_t` `bytesEachRead`

Bytes for each read RDR.

`volatile uint8_t` `bytesLastRead`

Bytes for last read RDR.

`volatile bool` `isThereExtraRxBytes`

Is there extra RX byte.

`const uint8_t *volatile` `txData`

Send buffer.

`uint8_t *volatile` `rxData`

Receive buffer.

`volatile size_t` `txRemainingByteCount`

Number of bytes remaining to send.

`volatile size_t` `rxRemainingByteCount`

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

edma_tcd_t *lastTimeTCD
Pointer to the lastTime TCD

bool isMultiDMATransmit
Is there multi DMA transmit

volatile uint8_t dmaTransmitTime
DMA Transfer times.

uint32_t lastTimeDataBytes
DMA transmit last Time data Bytes

uint32_t dataBytesEveryTime
Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

edma_transfer_config_t transferConfigRx
Config of DMA rx channel.

edma_transfer_config_t transferConfigTx
Config of DMA tx channel.

uint32_t txBuffIfNull
Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
Used if there is not rxData for DMA purpose.

uint32_t transmitCommand
Used to write TCR for DMA purpose.

volatile uint8_t state
LPSPI transfer state , *_lpspi_transfer_state*.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
SoftwareTCD, internal used

struct *_lpspi_slave_edma_handle*
#include <fsl_lpspi_edma.h> LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

volatile uint8_t bytesLastRead

Bytes for last read RDR.

volatile bool isThereExtraRxBytes

Is there extra RX byte.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes

Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes

Read RDR register remaining times.

uint32_t totalByteCount

Number of transfer bytes

uint32_t txBuffIfNull

Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull

Used if there is not rxData for DMA purpose.

volatile uint8_t state

LPSPi transfer state.

uint32_t errorCount

Error count for slave transfer.

lpspi_slave_edma_transfer_callback_t callback

Completion callback.

```
void *userData
    Callback user data.
edma_handle_t *edmaRxRegToRxDataHandle
    edma_handle_t handle point used for RxReg to RxData buff
edma_handle_t *edmaTxDataToTxRegHandle
    edma_handle_t handle point used for TxData to TxReg
edma_tcd_t lpspiSoftwareTCD[2]
    SoftwareTCD, internal used
```

2.49 LPTMR: Low-Power Timer

```
void LPTMR_Init(LPTMR_Type *base, const lptmr_config_t *config)
    Ungates the LPTMR clock and configures the peripheral for a basic operation.
```

Note: This API should be called at the beginning of the application using the LPTMR driver.

Parameters

- base – LPTMR peripheral base address
- config – A pointer to the LPTMR configuration structure.

```
void LPTMR_Deinit(LPTMR_Type *base)
    Gates the LPTMR clock.
```

Parameters

- base – LPTMR peripheral base address

```
void LPTMR_GetDefaultConfig(lptmr_config_t *config)
    Fills in the LPTMR configuration structure with default settings.
```

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

Parameters

- config – A pointer to the LPTMR configuration structure.

```
static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)
    Enables the selected LPTMR interrupts.
```

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `lptmr_interrupt_enable_t`

static inline void LPTMR_DisableInterrupts(LPTMR_Type *base, uint32_t mask)

Disables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration `lptmr_interrupt_enable_t`.

static inline uint32_t LPTMR_GetEnabledInterrupts(LPTMR_Type *base)

Gets the enabled LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `lptmr_interrupt_enable_t`

static inline uint32_t LPTMR_GetStatusFlags(LPTMR_Type *base)

Gets the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `lptmr_status_flags_t`

static inline void LPTMR_ClearStatusFlags(LPTMR_Type *base, uint32_t mask)

Clears the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `lptmr_status_flags_t`.

static inline void LPTMR_SetTimerPeriod(LPTMR_Type *base, uint32_t ticks)

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note:

- a. The TCF flag is set with the CNR equals the count provided here and then increments.
 - b. Call the utility macros provided in the `fsl_common.h` to convert to ticks.
-

Parameters

- base – LPTMR peripheral base address
- ticks – A timer period in units of ticks

static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

- base – LPTMR peripheral base address

Returns

The current counter value in ticks

```
static inline void LPTMR_StartTimer(LPTMR_Type *base)
```

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

- base – LPTMR peripheral base address

```
static inline void LPTMR_StopTimer(LPTMR_Type *base)
```

Stops the timer.

This function stops the timer and resets the timer's counter register.

Parameters

- base – LPTMR peripheral base address

```
FSL_LPTMR_DRIVER_VERSION
```

Driver Version

```
enum _lptmr_pin_select
```

LPTMR pin selection used in pulse counter mode.

Values:

```
enumerator kLPTMR_PinSelectInput_0
```

Pulse counter input 0 is selected

```
enumerator kLPTMR_PinSelectInput_1
```

Pulse counter input 1 is selected

```
enumerator kLPTMR_PinSelectInput_2
```

Pulse counter input 2 is selected

```
enumerator kLPTMR_PinSelectInput_3
```

Pulse counter input 3 is selected

```
enum _lptmr_pin_polarity
```

LPTMR pin polarity used in pulse counter mode.

Values:

```
enumerator kLPTMR_PinPolarityActiveHigh
```

Pulse Counter input source is active-high

```
enumerator kLPTMR_PinPolarityActiveLow
```

Pulse Counter input source is active-low

```
enum _lptmr_timer_mode
```

LPTMR timer mode selection.

Values:

```
enumerator kLPTMR_TimerModeTimeCounter
```

Time Counter mode

enumerator kLPTMR_TimerModePulseCounter
Pulse Counter mode

enum _lptmr_prescaler_glitch_value
LPTMR prescaler/glitch filter values.

Values:

enumerator kLPTMR_Prescale_Glitch_0
Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR_Prescale_Glitch_1
Prescaler divide 4, glitch filter 2

enumerator kLPTMR_Prescale_Glitch_2
Prescaler divide 8, glitch filter 4

enumerator kLPTMR_Prescale_Glitch_3
Prescaler divide 16, glitch filter 8

enumerator kLPTMR_Prescale_Glitch_4
Prescaler divide 32, glitch filter 16

enumerator kLPTMR_Prescale_Glitch_5
Prescaler divide 64, glitch filter 32

enumerator kLPTMR_Prescale_Glitch_6
Prescaler divide 128, glitch filter 64

enumerator kLPTMR_Prescale_Glitch_7
Prescaler divide 256, glitch filter 128

enumerator kLPTMR_Prescale_Glitch_8
Prescaler divide 512, glitch filter 256

enumerator kLPTMR_Prescale_Glitch_9
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR_Prescale_Glitch_10
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR_Prescale_Glitch_11
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR_Prescale_Glitch_12
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR_Prescale_Glitch_13
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR_Prescale_Glitch_14
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR_Prescale_Glitch_15
Prescaler divide 65536, glitch filter 32768

enum _lptmr_prescaler_clock_select
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

Values:

enum `_lptmr_interrupt_enable`

List of the LPTMR interrupts.

Values:

enumerator `kLPTMR_TimerInterruptEnable`

Timer interrupt enable

enum `_lptmr_status_flags`

List of the LPTMR status flags.

Values:

enumerator `kLPTMR_TimerCompareFlag`

Timer compare flag

typedef enum `_lptmr_pin_select` `lptmr_pin_select_t`

LPTMR pin selection used in pulse counter mode.

typedef enum `_lptmr_pin_polarity` `lptmr_pin_polarity_t`

LPTMR pin polarity used in pulse counter mode.

typedef enum `_lptmr_timer_mode` `lptmr_timer_mode_t`

LPTMR timer mode selection.

typedef enum `_lptmr_prescaler_glitch_value` `lptmr_prescaler_glitch_value_t`

LPTMR prescaler/glitch filter values.

typedef enum `_lptmr_prescaler_clock_select` `lptmr_prescaler_clock_select_t`

LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

typedef enum `_lptmr_interrupt_enable` `lptmr_interrupt_enable_t`

List of the LPTMR interrupts.

typedef enum `_lptmr_status_flags` `lptmr_status_flags_t`

List of the LPTMR status flags.

typedef struct `_lptmr_config` `lptmr_config_t`

LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the `LPTMR_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

static inline void `LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)`

Enable or disable timer DMA request.

Parameters

- `base` – base LPTMR peripheral base address
- `enable` – Switcher of timer DMA feature. “true” means to enable, “false” means to disable.

struct `_lptmr_config`

`#include <fsl_lptmr.h>` LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the `LPTMR_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode

Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler

True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource

LPTMR clock source

lptmr_prescaler_glitch_value_t value

Prescaler or glitch filter value

2.50 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.51 LPUART Driver

```
static inline void LPUART_SoftwareReset(LPUART_Type *base)
```

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

```
status_t LPUART_Init(LPUART_Type *base, const lpuart_config_t *config, uint32_t srcClock_Hz)
```

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;  
lpuartConfig.baudRate_Bps = 115200U;  
lpuartConfig.parityMode = kLPUART_ParityDisabled;  
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;  
lpuartConfig.isMsb = false;  
lpuartConfig.stopBitCount = kLPUART_OneStopBit;  
lpuartConfig.txFifoWatermark = 0;
```

(continues on next page)

(continued from previous page)

```
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

```
status_t LPUART_Deinit(LPUART_Type *base)
```

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

Return values

- kStatus_Success – Deinit is success.
- kStatus_LPUART_Timeout – Timeout during deinit.

```
void LPUART_GetDefaultConfig(lpuart_config_t *config)
```

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

```
status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t
srcClock_Hz)
```

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, false to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in *lpuart_handle_t*.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *_lpuart_flags*. To check for a specific status, compare the return value with enumerators in the *_lpuart_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the *_lpuart_flags*.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: *kLPUART_TxDataRegEmptyFlag*, *kLPUART_TransmissionCompleteFlag*, *kLPUART_RxDataRegFullFlag*, *kLPUART_RxActiveFlag*, *kLPUART_NoiseErrorFlag*, *kLPUART_ParityErrorFlag*, *kLPUART_TxFifoEmptyFlag*, *kLPUART_RxFifoEmptyFlag* Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the *_lpuart_status_flag_t* to do the OR operation and get the mask.

Return values

- *kStatus_LPUART_FlagCannotClearManually* – The flag can't be cleared by this function but it is cleared automatically by hardware.
- *kStatus_Success* – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the *_lpuart_interrupt_enable*. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of `_lpuart_interrupt_enable`.

```
void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)
```

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpuart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of `_lpuart_interrupt_enable`.

```
uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)
```

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_lpuart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in `_lpuart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in `_lpuart_interrupt_enable`.

```
static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)
```

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

```
static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
uint32_t LPUART_GetInstance(LPUART_Type *base)
```

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

```
static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)
```

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

```
static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)
```

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.

- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

```
void LPUART_TransferCreateHandle(LPUART_Type *base, lpuart_handle_t *handle,  
                                lpuart_transfer_callback_t callback, void *userData)
```

Initializes the LPUART handle.

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the “background” receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the `LPUART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing `NULL` as `ringBuffer`.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `callback` – Callback function.
- `userData` – User data.

```
status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, lpuart_handle_t *handle,  
                                        lpuart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the `kStatus_LPUART_TxIdle` as status parameter.

Note: The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART transfer structure, see `lpuart_transfer_t`.

Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_LPUART_TxBusy` – Previous transmission still not finished, data not all written to the TX register.
- `kStatus_InvalidArgument` – Invalid argument.

```
void LPUART_TransferStartRingBuffer(LPUART_Type *base, lpuart_handle_t *handle, uint8_t  
                                    *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `ringBuffer` – Start address of ring buffer for background receiving. Pass `NULL` to disable the ring buffer.
- `ringBufferSize` – size of the ring buffer.

`void LPUART_TransferStopRingBuffer(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, lpuart_handle_t *handle)`

Get the length of received data in RX ring buffer.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

`void LPUART_TransferAbortSend(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the `remainBtyes` to find out how many bytes are not sent out.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`status_t LPUART_TransferGetSendCount(LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- `base` – LPUART peripheral base address.

- `handle` – LPUART handle pointer.
- `count` – Send bytes count.

Return values

- `kStatus_NoTransferInProgress` – No send in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`status_t` LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, `size_t` *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter `kStatus_UART_RxIdle`. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to `xfer->data`, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from `xfer->data[5]`. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `xfer` – LPUART transfer structure, see `uart_transfer_t`.
- `receivedBytes` – Bytes received from the ring buffer directly.

Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_LPUART_RxBusy` – Previous receive request is not finished.
- `kStatus_InvalidArgument` – Invalid argument.

`void` LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.

`status_t` LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, `uint32_t` *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- `base` – LPUART peripheral base address.

- `handle` – LPUART handle pointer.
- `count` – Receive bytes count.

Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)`
LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- `base` – LPUART peripheral base address.
- `irqHandle` – LPUART handle pointer.

`void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)`
LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- `base` – LPUART peripheral base address.
- `irqHandle` – LPUART handle pointer.

`void LPUART_DriverIRQHandler(uint32_t instance)`
LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- `instance` – LPUART instance.

`FSL_LPUART_DRIVER_VERSION`
LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator `kStatus_LPUART_TxBusy`
TX busy

enumerator `kStatus_LPUART_RxBusy`
RX busy

enumerator `kStatus_LPUART_TxIdle`
LPUART transmitter is idle.

enumerator `kStatus_LPUART_RxIdle`
LPUART receiver is idle.

enumerator `kStatus_LPUART_TxWatermarkTooLarge`
TX FIFO watermark too large

enumerator `kStatus_LPUART_RxWatermarkTooLarge`
RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually

Some flag can't manually clear

enumerator kStatus_LPUART_Error

Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun

LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun

LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError

LPUART noise error.

enumerator kStatus_LPUART_FramingError

LPUART framing error.

enumerator kStatus_LPUART_ParityError

LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport

Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected

IDLE flag.

enumerator kStatus_LPUART_Timeout

LPUART times out.

enum _lpuart_parity_mode

LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled

Parity disabled

enumerator kLPUART_ParityEven

Parity enabled, type even, bit setting: PE | PT = 10

enumerator kLPUART_ParityOdd

Parity enabled, type odd, bit setting: PE | PT = 11

enum _lpuart_data_bits

LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits

Eight data bit

enumerator kLPUART_SevenDataBits

Seven data bit

enum _lpuart_stop_bit_count

LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit

One stop bit

enumerator kLPUART_TwoStopBit

Two stop bits

enum _lpuart_transmit_cts_source

LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin

CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult

CTS resource is the match result.

enum _lpuart_transmit_cts_config

LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart

CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle

CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select

LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit

Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit

Start counting after a stop bit.

enum _lpuart_idle_config

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1

the number of idle characters.

enumerator kLPUART_IdleCharacter2

the number of idle characters.

enumerator kLPUART_IdleCharacter4

the number of idle characters.

enumerator kLPUART_IdleCharacter8

the number of idle characters.

enumerator kLPUART_IdleCharacter16

the number of idle characters.

enumerator kLPUART_IdleCharacter32

the number of idle characters.

enumerator kLPUART_IdleCharacter64

the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable

LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags

LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator `kLPUART_TransmissionCompleteFlag`
 Transmission complete flag, sets when transmission activity complete. bit 22

enumerator `kLPUART_RxDataRegFullFlag`
 Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator `kLPUART_IdleLineFlag`
 Idle line detect flag, sets when idle line detected. bit 20

enumerator `kLPUART_RxOverrunFlag`
 Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator `kLPUART_NoiseErrorFlag`
 Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator `kLPUART_FramingErrorFlag`
 Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator `kLPUART_ParityErrorFlag`
 If parity enabled, sets upon parity error detection. bit 16

enumerator `kLPUART_LinBreakFlag`
 LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator `kLPUART_RxActiveEdgeFlag`
 Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator `kLPUART_RxActiveFlag`
 Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator `kLPUART_DataMatch1Flag`
 The next character to be read from `LPUART_DATA` matches MA1. bit 15

enumerator `kLPUART_DataMatch2Flag`
 The next character to be read from `LPUART_DATA` matches MA2. bit 14

enumerator `kLPUART_TxFifoEmptyFlag`
 TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator `kLPUART_RxFifoEmptyFlag`
 RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator `kLPUART_TxFifoOverflowFlag`
 TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator `kLPUART_RxFifoUnderflowFlag`
 RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator `kLPUART_AllClearFlags`

enumerator `kLPUART_AllFlags`

typedef enum `_lpuart_parity_mode` `lpuart_parity_mode_t`
 LPUART parity mode.

typedef enum `_lpuart_data_bits` `lpuart_data_bits_t`
 LPUART data bits count.

typedef enum `_lpuart_stop_bit_count` `lpuart_stop_bit_count_t`
 LPUART stop bit count.

```
typedef enum _lpuart_transmit_cts_source lpuart_transmit_cts_source_t
    LPUART transmit CTS source.

typedef enum _lpuart_transmit_cts_config lpuart_transmit_cts_config_t
    LPUART transmit CTS configure.

typedef enum _lpuart_idle_type_select lpuart_idle_type_select_t
    LPUART idle flag type defines when the receiver starts counting.

typedef enum _lpuart_idle_config lpuart_idle_config_t
    LPUART idle detected configuration. This structure defines the number of idle characters
    that must be received before the IDLE flag is set.

typedef struct _lpuart_config lpuart_config_t
    LPUART configuration structure.

typedef struct _lpuart_transfer lpuart_transfer_t
    LPUART transfer structure.

typedef struct _lpuart_handle lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle,
status_t status, void *userData)
    LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
    Retry times for waiting flag.

struct _lpuart_config
    #include <fsl_lpuart.h> LPUART configuration structure.
```

Public Members

```
uint32_t baudRate_Bps
    LPUART baud rate

lpuart_parity_mode_t parityMode
    Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
    Data bits count, eight (default), seven

bool isMsb
    Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
    Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
    TX FIFO watermark

uint8_t rxFifoWatermark
    RX FIFO watermark
```

bool enableRxRTS
RX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

bool enableTx
Enable TX

bool enableRx
Enable RX

struct *_lpuart_transfer*
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct *_lpuart_handle*
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

volatile size_t txDataSize
Size of the remaining data to send.

size_t txDataSizeAll
Size of the data to send out.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t rxDataSizeAll
Size of the data to receive.

size_t rxRingBufferSize
Size of the ring buffer.

volatile uint16_t rxRingBufferHead
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
Callback function.

void *userData
LPUART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state.

bool isSevenDataBits
Seven data bits flag.

bool is16bitData
16bit data bits flag, only used for 9bit or 10bit data

union __unnamed58__

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

uint16_t *rxData16
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

const uint16_t *txData16
The buffer of data to be sent.

union __unnamed60__

Public Members

const uint8_t *volatile txData
Address of remaining data to send.

const uint16_t *volatile txData16
Address of remaining data to send.

union __unnamed62__

Public Members

uint8_t *volatile rxData
Address of remaining data to receive.

uint16_t *volatile rxData16
Address of remaining data to receive.

union __unnamed64__

Public Members

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16

Start address of the receiver ring buffer.

2.52 LPUART eDMA Driver

```
void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,
                                     lpuart_edma_transfer_callback_t callback, void
                                     *userData, edma_handle_t *txEdmaHandle,
                                     edma_handle_t *rxEdmaHandle)
```

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

```
status_t LPUART_SendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,
                        lpuart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

```
status_t LPUART_ReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,
                            lpuart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.
- xfer – LPUART eDMA transfer structure, see `lpuart_transfer_t`.

Return values

- `kStatus_Success` – if succeed, others fail.
- `kStatus_LPUART_RxBusy` – Previous transfer ongoing.
- `kStatus_InvalidArgument` – Invalid argument.

`void LPUART_TransferAbortSendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`
Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`
Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- `kStatus_NoTransferInProgress` – No send in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

`status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`void LPUART_TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)`
LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- `base` – LPUART peripheral base address.
- `lpuartEdmaHandle` – LPUART handle pointer.

`FSL_LPUART_EDMA_DRIVER_VERSION`

LPUART EDMA driver version.

`typedef struct _lpuart_edma_handle lpuart_edma_handle_t`

`typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`

LPUART transfer callback function.

`struct _lpuart_edma_handle`

`#include <fsl_lpuart_edma.h>` LPUART eDMA handle.

Public Members

`lpuart_edma_transfer_callback_t` callback

Callback function.

`void *userData`

LPUART callback function parameter.

`size_t rxDataSizeAll`

Size of the data to receive.

`size_t txDataSizeAll`

Size of the data to send out.

`edma_handle_t *txEdmaHandle`

The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`

The eDMA RX channel used.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

`volatile uint8_t txState`

TX transfer state.

`volatile uint8_t rxState`

RX transfer state

2.53 LTC: LP Trusted Cryptography

FSL_LTC_DRIVER_VERSION

LTC driver version. Version 2.0.17.

Current version: 2.0.17

Change log:

- Version 2.0.1
 - fixed warning during g++ compilation
- Version 2.0.2
 - fixed [KPSDK-10932][LTC][SHA] LTC_HASH() blocks indefinitely when message size exceeds 4080 bytes
- Version 2.0.3
 - fixed LTC_PKHA_CompareBigNum() in case an integer argument is an array of all zeroes
- Version 2.0.4
 - constant LTC_PKHA_CompareBigNum() processing time
- Version 2.0.5
 - Fix MISRA issues
- Version 2.0.6
 - fixed [KPSDK-23603][LTC] AES Decrypt in ECB and CBC modes fail when ciphertext size > 0xff0 bytes
- Version 2.0.7
 - Fix MISRA-2012 issues
- Version 2.0.8
 - Fix Coverity issues
- Version 2.0.9
 - Fix sign-compare warning in ltc_set_context and in ltc_get_context
- Version 2.0.10
 - Fix MISRA-2012 issues
- Version 2.0.11
 - Fix MISRA-2012 issues
- Version 2.0.12
 - Fix AES Decrypt in CBC modes fail when used kLTC_DecryptKey.
- Version 2.0.13
 - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Init function.
- Version 2.0.14
 - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Deinit function.
- Version 2.0.15
 - Fix MISRA-2012 issues

- Version 2.0.16
 - Fix uninitialized GCC warning in LTC_AES_GenerateDecryptKey()
- Version 2.0.17
 - Fix CMAC for payloads over one block, and if BRIC is present on the device, remove XCBC and “decrypt key” functionality

void LTC_Init(LTC_Type *base)

Initializes the LTC driver. This function initializes the LTC driver.

Parameters

- base – LTC peripheral base address

void LTC_Deinit(LTC_Type *base)

Deinitializes the LTC driver. This function deinitializes the LTC driver.

Parameters

- base – LTC peripheral base address

void LTC_SetDpaMaskSeed(LTC_Type *base, uint32_t mask)

Sets the DPA Mask Seed register.

The DPA Mask Seed register reseeds the mask that provides resistance against DPA (differential power analysis) attacks on AES or DES keys.

Differential Power Analysis Mask (DPA) resistance uses a randomly changing mask that introduces “noise” into the power consumed by the AES or DES. This reduces the signal-to-noise ratio that differential power analysis attacks use to “guess” bits of the key. This randomly changing mask should be seeded at POR, and continues to provide DPA resistance from that point on. However, to provide even more DPA protection it is recommended that the DPA mask be reseeded after every 50,000 blocks have been processed. At that time, software can opt to write a new seed (preferably obtained from an RNG) into the DPA Mask Seed register (DPAMS), or software can opt to provide the new seed earlier or later, or not at all. DPA resistance continues even if the DPA mask is never reseeded.

Parameters

- base – LTC peripheral base address
- mask – The DPA mask seed.

2.54 LTC AES driver

enum _ltc_aes_key_t

Type of AES key for ECB and CBC decrypt operations.

Values:

enumerator kLTC_EncryptKey

Input key is an encrypt key

typedef enum _ltc_aes_key_t ltc_aes_key_t

Type of AES key for ECB and CBC decrypt operations.

status_t LTC_AES_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *key, uint32_t keySize)

Encrypts AES using the ECB block mode.

Encrypts AES using the ECB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

```
status_t LTC_AES_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,
                             uint32_t size, const uint8_t *key, uint32_t keySize, ltc_aes_key_t
                             keyType)
```

Decrypts AES using ECB block mode.

Decrypts AES using ECB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key.
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- keyType – Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns

Status from decrypt operation

```
status_t LTC_AES_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,
                              uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t
                              keySize)
```

Encrypts AES using CBC block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

```
status_t LTC_AES_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,
                              uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t
                              keySize, ltc_aes_key_t keyType)
```

Decrypts AES using CBC block mode.

Parameters

- `base` – LTC peripheral base address
- `ciphertext` – Input cipher text to decrypt
- `plaintext` – **[out]** Output plain text
- `size` – Size of input and output data in bytes. Must be multiple of 16 bytes.
- `iv` – Input initial vector to combine with the first input block.
- `key` – Input key to use for decryption
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `keyType` – Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

Returns

Status from decrypt operation

```
status_t LTC_AES_CryptCtr(LTC_Type *base, const uint8_t *input, uint8_t *output, uint32_t
    size, uint8_t counter[16U], const uint8_t *key, uint32_t keySize,
    uint8_t counterlast[16U], uint32_t *szLeft)
```

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- `base` – LTC peripheral base address
- `input` – Input data for CTR block mode
- `output` – **[out]** Output data for CTR block mode
- `size` – Size of input and output data in bytes
- `counter` – **[inout]** Input counter (updates on return)
- `key` – Input key to use for forward AES cipher
- `keySize` – Size of the input key, in bytes. Must be 16, 24, or 32.
- `counterlast` – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- `szLeft` – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

```
status_t LTC_AES_EncryptTagGcm(LTC_Type *base, const uint8_t *plaintext, uint8_t
    *ciphertext, uint32_t size, const uint8_t *iv, uint32_t ivSize,
    const uint8_t *aad, uint32_t aadSize, const uint8_t *key,
    uint32_t keySize, uint8_t *tag, uint32_t tagSize)
```

Encrypts AES and tags using GCM block mode.

Encrypts AES and optionally tags using GCM block mode. If `plaintext` is NULL, only the GHASH is calculated and output in the ‘tag’ field.

Parameters

- `base` – LTC peripheral base address
- `plaintext` – Input plain text to encrypt

- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Output hash tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from encrypt operation

```
status_t LTC_AES_DecryptTagGcm(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, const uint8_t *tag, uint32_t tagSize)
```

Decrypts AES and authenticates using GCM block mode.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Input hash tag to compare. Set to NULL to skip tag processing.
- tagSize – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from decrypt operation

```
status_t LTC_AES_EncryptTagCcm(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, uint8_t *tag, uint32_t tagSize)
```

Encrypts AES and tags using CCM block mode.

Encrypts AES and optionally tags using CCM block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes. Zero means authentication only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- aad – Input additional authentication data. Can be NULL if aadSize is zero.
- aadSize – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Generated output tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from encrypt operation

```
status_t LTC_AES_DecryptTagCcm(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, const uint8_t *tag, uint32_t tagSize)
```

Decrypts AES and authenticates using CCM block mode.

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes. Zero means authentication only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- aad – Input additional authentication data. Can be NULL if aadSize is zero.
- aadSize – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Received tag. Set to NULL to skip tag processing.
- tagSize – Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from decrypt operation

LTC_AES_BLOCK_SIZE

AES block size in bytes

LTC_AES_IV_SIZE

AES Input Vector size in bytes

LTC_KEY_REGISTER_READABLE

LTC_AES_DecryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR decrypt is mapped to the AES CTR generic operation

LTC_AES_EncryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR encrypt is mapped to the AES CTR generic operation

2.55 LTC DES driver

status_t LTC_DES_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key[8])

Encrypts DES using ECB block mode.

Encrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key[8])

Decrypts DES using ECB block mode.

Decrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using CBC block mode.

Encrypts DES using CBC block mode.

Parameters

- `base` – LTC peripheral base address
- `plaintext` – Input plaintext to encrypt
- `ciphertext` – **[out]** Output ciphertext
- `size` – Size of input and output data in bytes
- `iv` – Input initial vector to combine with the first plaintext block. The `iv` does not need to be secret, but it must be unpredictable.
- `key` – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
uint32_t size, const uint8_t iv[8], const uint8_t key[8])
```

Decrypts DES using CBC block mode.

Decrypts DES using CBC block mode.

Parameters

- `base` – LTC peripheral base address
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input data in bytes
- `iv` – Input initial vector to combine with the first plaintext block. The `iv` does not need to be secret, but it must be unpredictable.
- `key` – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,  
uint32_t size, const uint8_t iv[8], const uint8_t key[8])
```

Encrypts DES using CFB block mode.

Encrypts DES using CFB block mode.

Parameters

- `base` – LTC peripheral base address
- `plaintext` – Input plaintext to encrypt
- `size` – Size of input data in bytes
- `iv` – Input initial block.
- `key` – Input key to use for encryption
- `ciphertext` – **[out]** Output ciphertext

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
uint32_t size, const uint8_t iv[8], const uint8_t key[8])
```

Decrypts DES using CFB block mode.

Decrypts DES using CFB block mode.

Parameters

- `base` – LTC peripheral base address

- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using OFB block mode.

Encrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using OFB block mode.

Decrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using ECB block mode with two keys.

Encrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt

- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using ECB block mode with two keys.

Decrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using CBC block mode with two keys.

Encrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using CBC block mode with two keys.

Decrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address

- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES2_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,
                             uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const
                             uint8_t key2[8])
```

Encrypts triple DES using CFB block mode with two keys.

Encrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES2_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,
                              uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const
                              uint8_t key2[8])
```

Decrypts triple DES using CFB block mode with two keys.

Decrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES2_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,
                              uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const
                              uint8_t key2[8])
```

Encrypts triple DES using OFB block mode with two keys.

Encrypts triple DES using OFB block mode with two keys.

Parameters

- `base` – LTC peripheral base address
- `plaintext` – Input plaintext to encrypt
- `ciphertext` – **[out]** Output ciphertext
- `size` – Size of input and output data in bytes
- `iv` – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- `key1` – First input key for key bundle
- `key2` – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES2_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
                             uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const  
                             uint8_t key2[8])
```

Decrypts triple DES using OFB block mode with two keys.

Decrypts triple DES using OFB block mode with two keys.

Parameters

- `base` – LTC peripheral base address
- `ciphertext` – Input ciphertext to decrypt
- `plaintext` – **[out]** Output plaintext
- `size` – Size of input and output data in bytes
- `iv` – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- `key1` – First input key for key bundle
- `key2` – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,  
                              uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const  
                              uint8_t key3[8])
```

Encrypts triple DES using ECB block mode with three keys.

Encrypts triple DES using ECB block mode with three keys.

Parameters

- `base` – LTC peripheral base address
- `plaintext` – Input plaintext to encrypt
- `ciphertext` – **[out]** Output ciphertext
- `size` – Size of input and output data in bytes. Must be multiple of 8 bytes.
- `key1` – First input key for key bundle
- `key2` – Second input key for key bundle

- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
                             uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const  
                             uint8_t key3[8])
```

Decrypts triple DES using ECB block mode with three keys.

Decrypts triple DES using ECB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,  
                              uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const  
                              uint8_t key2[8], const uint8_t key3[8])
```

Encrypts triple DES using CBC block mode with three keys.

Encrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
                              uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const  
                              uint8_t key2[8], const uint8_t key3[8])
```

Decrypts triple DES using CBC block mode with three keys.

Decrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt

- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,  
                             uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const  
                             uint8_t key2[8], const uint8_t key3[8])
```

Encrypts triple DES using CFB block mode with three keys.

Encrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
                              uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const  
                              uint8_t key2[8], const uint8_t key3[8])
```

Decrypts triple DES using CFB block mode with three keys.

Decrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext,  
                             uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const  
                             uint8_t key2[8], const uint8_t key3[8])
```

Encrypts triple DES using OFB block mode with three keys.

Encrypts triple DES using OFB block mode with three keys.

Parameters

- *base* – LTC peripheral base address
- *plaintext* – Input plaintext to encrypt
- *ciphertext* – **[out]** Output ciphertext
- *size* – Size of input and output data in bytes
- *iv* – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- *key1* – First input key for key bundle
- *key2* – Second input key for key bundle
- *key3* – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

```
status_t LTC_DES3_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext,  
                              uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const  
                              uint8_t key2[8], const uint8_t key3[8])
```

Decrypts triple DES using OFB block mode with three keys.

Decrypts triple DES using OFB block mode with three keys.

Parameters

- *base* – LTC peripheral base address
- *ciphertext* – Input ciphertext to decrypt
- *plaintext* – **[out]** Output plaintext
- *size* – Size of input and output data in bytes
- *iv* – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- *key1* – First input key for key bundle
- *key2* – Second input key for key bundle
- *key3* – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

LTC_DES_KEY_SIZE

LTC DES key size - 64 bits.

LTC_DES_IV_SIZE

LTC DES IV size - 8 bytes.

2.56 LTC HASH driver

enum `_ltc_hash_algo_t`

Supported cryptographic block cipher functions for HASH creation

Values:

enumerator `kLTC_Cmac`

CMAC (AES engine)

enumerator `kLTC_Sha1`

SHA_1 (MDHA engine)

enumerator `kLTC_Sha224`

SHA_224 (MDHA engine)

enumerator `kLTC_Sha256`

SHA_256 (MDHA engine)

typedef enum `_ltc_hash_algo_t` `ltc_hash_algo_t`

Supported cryptographic block cipher functions for HASH creation

typedef struct `_ltc_hash_ctx_t` `ltc_hash_ctx_t`

Storage type used to save hash context.

`status_t` `LTC_HASH_Init(LTC_Type *base, ltc_hash_ctx_t *ctx, ltc_hash_algo_t algo, const uint8_t *key, uint32_t keySize)`

Initialize HASH context.

This function initialize the HASH. Key shall be supplied if the underlying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

Parameters

- `base` – LTC peripheral base address
- `ctx` – **[out]** Output hash context
- `algo` – Underlying algorithm to use for hash computation.
- `key` – Input key (NULL if underlying algorithm is SHA)
- `keySize` – Size of input key in bytes

Returns

Status of initialization

`status_t` `LTC_HASH_Update(ltc_hash_ctx_t *ctx, const uint8_t *input, uint32_t inputSize)`

Add data to current HASH.

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

Parameters

- `ctx` – **[inout]** HASH context
- `input` – Input data
- `inputSize` – Size of input data in bytes

Returns

Status of the hash update operation

`status_t LTC_HASH_Finish(ltc_hash_ctx_t *ctx, uint8_t *output, uint32_t *outputSize)`

Finalize hashing.

Outputs the final hash and erases the context.

Parameters

- `ctx` – **[inout]** Input hash context
- `output` – **[out]** Output hash data
- `outputSize` – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the hash finish operation

`status_t LTC_HASH(LTC_Type *base, ltc_hash_algo_t algo, const uint8_t *input, uint32_t inputSize, const uint8_t *key, uint32_t keySize, uint8_t *output, uint32_t *outputSize)`

Create HASH on given data.

Perform the full keyed HASH in one function call.

Parameters

- `base` – LTC peripheral base address
- `algo` – Block cipher algorithm to use for CMAC creation
- `input` – Input data
- `inputSize` – Size of input data in bytes
- `key` – Input key
- `keySize` – Size of input key in bytes
- `output` – **[out]** Output hash data
- `outputSize` – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

`LTC_HASH_CTX_SIZE`

LTC HASH Context size.

`struct _ltc_hash_ctx_t`

`#include <fsl_ltc.h>` Storage type used to save hash context.

2.57 LTC PKHA driver

`enum _ltc_pkha_timing_t`

Use of timing equalized version of a PKHA function.

Values:

enumerator `kLTC_PKHA_NoTimingEqualized`

Normal version of a PKHA operation

enumerator `kLTC_PKHA_TimingEqualized`

Timing-equalized version of a PKHA operation

```
enum _ltc_pkha_f2m_t
```

Integer vs binary polynomial arithmetic selection.

Values:

```
enumerator kLTC_PKHA_IntegerArith
```

Use integer arithmetic

```
enumerator kLTC_PKHA_F2mArith
```

Use binary polynomial arithmetic

```
enum _ltc_pkha_montgomery_form_t
```

Montgomery or normal PKHA input format.

Values:

```
enumerator kLTC_PKHA_NormalValue
```

PKHA number is normal integer

```
enumerator kLTC_PKHA_MontgomeryFormat
```

PKHA number is in montgomery format

```
typedef struct _ltc_pkha_ecc_point_t ltc_pkha_ecc_point_t
```

PKHA ECC point structure

```
typedef enum _ltc_pkha_timing_t ltc_pkha_timing_t
```

Use of timing equalized version of a PKHA function.

```
typedef enum _ltc_pkha_f2m_t ltc_pkha_f2m_t
```

Integer vs binary polynomial arithmetic selection.

```
typedef enum _ltc_pkha_montgomery_form_t ltc_pkha_montgomery_form_t
```

Montgomery or normal PKHA input format.

```
int LTC_PKHA_CompareBigNum(const uint8_t *a, size_t sizeA, const uint8_t *b, size_t sizeB)
```

Compare two PKHA big numbers.

Compare two PKHA big numbers. Return 1 for $a > b$, -1 for $a < b$ and 0 if they are same. PKHA big number is lsbyte first. Thus the comparison starts at msbyte which is the last member of tested arrays.

Parameters

- a – First integer represented as an array of bytes, lsbyte first.
- sizeA – Size in bytes of the first integer.
- b – Second integer represented as an array of bytes, lsbyte first.
- sizeB – Size in bytes of the second integer.

Returns

1 if $a > b$.

Returns

-1 if $a < b$.

Returns

0 if $a = b$.

```
status_t LTC_PKHA_NormalToMontgomery(LTC_Type *base, const uint8_t *N, uint16_t sizeN,
                                     uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t
                                     *sizeB, uint8_t *R2, uint16_t *sizeR2,
                                     ltc_pkha_timing_t equalTime, ltc_pkha_f2m_t
                                     arithType)
```

Converts from integer to Montgomery format.

This function computes $R2 \bmod N$ and optionally converts A or B into Montgomery format of A or B.

Parameters

- base – LTC peripheral base address
- N – modulus
- sizeN – size of N in bytes
- A – **[inout]** The first input in non-Montgomery format. Output Montgomery format of the first input.
- sizeA – **[inout]** pointer to size variable. On input it holds size of input A in bytes. On output it holds size of Montgomery format of A in bytes.
- B – **[inout]** Second input in non-Montgomery format. Output Montgomery format of the second input.
- sizeB – **[inout]** pointer to size variable. On input it holds size of input B in bytes. On output it holds size of Montgomery format of B in bytes.
- R2 – **[out]** Output Montgomery factor $R2 \bmod N$.
- sizeR2 – **[out]** pointer to size variable. On output it holds size of Montgomery factor $R2 \bmod N$ in bytes.
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t LTC_PKHA_MontgomeryToNormal(LTC_Type *base, const uint8_t *N, uint16_t sizeN,  
                                     uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t  
                                     *sizeB, ltc_pkha_timing_t equalTime, ltc_pkha_f2m_t  
                                     arithType)
```

Converts from Montgomery format to int.

This function converts Montgomery format of A or B into int A or B.

Parameters

- base – LTC peripheral base address
- N – modulus.
- sizeN – size of N modulus in bytes.
- A – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeA – **[inout]** pointer to size variable. On input it holds size of the input A in bytes. On output it holds size of non-Montgomery A in bytes.
- B – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeB – **[inout]** pointer to size variable. On input it holds size of the input B in bytes. On output it holds size of non-Montgomery B in bytes.
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t LTC_PKHA_ModAdd(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t
                        *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t
                        *result, uint16_t *resultSize, ltc_pkha_f2m_t arithType)
```

Performs modular addition - $(A + B) \bmod N$.

This function performs modular addition of $(A + B) \bmod N$, with either integer or binary polynomial (F2m) inputs. In the F2m form, this function is equivalent to a bitwise XOR and it is functionally the same as subtraction.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus. For F2m operation this can be NULL, as N is ignored during F2m polynomial addition.
- sizeN – Size of N in bytes. This must be given for both integer and F2m polynomial additions.
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t LTC_PKHA_ModSub1(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t
                        *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t
                        *result, uint16_t *resultSize)
```

Performs modular subtraction - $(A - B) \bmod N$.

This function performs modular subtraction of $(A - B) \bmod N$ with integer inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes

Returns

Operation status.

```
status_t LTC_PKHA_ModSub2(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t
                        *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t
                        *result, uint16_t *resultSize)
```

Performs modular subtraction - $(B - A) \bmod N$.

This function performs modular subtraction of $(B - A) \bmod N$, with integer inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes

Returns

Operation status.

```
status_t LTC_PKHA_ModMul(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t
                        *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t
                        *result, uint16_t *resultSize, ltc_pkha_f2m_t arithType,
                        ltc_pkha_montgomery_form_t montIn,
                        ltc_pkha_montgomery_form_t montOut, ltc_pkha_timing_t
                        equalTime)
```

Performs modular multiplication - $(A \times B) \bmod N$.

This function performs modular multiplication with either integer or binary polynomial (F2m) inputs. It can optionally specify whether inputs and/or outputs will be in Montgomery form or not.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus.
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)
- montIn – Format of inputs
- montOut – Format of output
- equalTime – Run the function time equalized or no timing equalization. This argument is ignored for F2m modular multiplication.

Returns

Operation status.

```
status_t LTC_PKHA_ModExp(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t
    *N, uint16_t sizeN, const uint8_t *E, uint16_t sizeE, uint8_t
    *result, uint16_t *resultSize, ltc_pkha_f2m_t arithType,
    ltc_pkha_montgomery_form_t montIn, ltc_pkha_timing_t
    equalTime)
```

Performs modular exponentiation - $(A^E) \bmod N$.

This function performs modular exponentiation with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- E – exponent
- sizeE – Size of E in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- montIn – Format of A input (normal or Montgomery)
- arithType – Type of arithmetic to perform (integer or F2m)
- equalTime – Run the function time equalized or no timing equalization.

Returns

Operation status.

```
status_t LTC_PKHA_ModRed(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t
    *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize,
    ltc_pkha_f2m_t arithType)
```

Performs modular reduction - $(A) \bmod N$.

This function performs modular reduction with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

status_t LTC_PKHA_ModInv(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, ltc_pkha_f2m_t arithType)

Performs modular inversion - $(A^{-1}) \bmod N$.

This function performs modular inversion with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

status_t LTC_PKHA_ModR2(LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, ltc_pkha_f2m_t arithType)

Computes integer Montgomery factor $R^2 \bmod N$.

This function computes a constant to assist in converting operands into the Montgomery residue system representation.

Parameters

- base – LTC peripheral base address
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

status_t LTC_PKHA_GCD(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, ltc_pkha_f2m_t arithType)

Calculates the greatest common divisor - GCD (A, N).

This function calculates the greatest common divisor of two inputs with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first value (must be smaller than or equal to N)
- sizeA – Size of A in bytes
- N – second value (must be non-zero)
- sizeN – Size of N in bytes

- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

```
status_t LTC_PKHA_PrimalityTest(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const
                                uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN,
                                bool *res)
```

Executes Miller-Rabin primality test.

This function calculates whether or not a candidate prime number is likely to be a prime.

Parameters

- base – LTC peripheral base address
- A – initial random seed
- sizeA – Size of A in bytes
- B – number of trial runs
- sizeB – Size of B in bytes
- N – candidate prime integer
- sizeN – Size of N in bytes
- res – **[out]** True if the value is likely prime or false otherwise

Returns

Operation status.

```
status_t LTC_PKHA_ECC_PointAdd(LTC_Type *base, const ltc_pkha_ecc_point_t *A, const
                                ltc_pkha_ecc_point_t *B, const uint8_t *N, const uint8_t
                                *R2modN, const uint8_t *aCurveParam, const uint8_t
                                *bCurveParam, uint8_t size, ltc_pkha_f2m_t arithType,
                                ltc_pkha_ecc_point_t *result)
```

Adds elliptic curve points - A + B.

This function performs ECC point addition over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – LTC peripheral base address
- A – Left-hand point
- B – Right-hand point
- N – Prime modulus of the field
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from LTC_PKHA_ModR2() function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns

Operation status.

```
status_t LTC_PKHA_ECC_PointDouble(LTC_Type *base, const ltc_pkha_ecc_point_t *B, const
uint8_t *N, const uint8_t *aCurveParam, const uint8_t
*bCurveParam, uint8_t size, ltc_pkha_f2m_t arithType,
ltc_pkha_ecc_point_t *result)
```

Doubles elliptic curve points - $B + B$.

This function performs ECC point doubling over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – LTC peripheral base address
- B – Point to double
- N – Prime modulus of the field
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns

Operation status.

```
status_t LTC_PKHA_ECC_PointMul(LTC_Type *base, const ltc_pkha_ecc_point_t *A, const
uint8_t *E, uint8_t sizeE, const uint8_t *N, const uint8_t
*R2modN, const uint8_t *aCurveParam, const uint8_t
*bCurveParam, uint8_t size, ltc_pkha_timing_t equalTime,
ltc_pkha_f2m_t arithType, ltc_pkha_ecc_point_t *result,
bool *infinity)
```

Multiplies an elliptic curve point by a scalar - $E \times (A_0, A_1)$.

This function performs ECC point multiplication to multiply an ECC point by a scalar integer multiplier over a prime field (Fp) or a binary field (F2m).

Parameters

- base – LTC peripheral base address
- A – Point as multiplicand
- E – Scalar multiple
- sizeE – The size of E, in bytes
- N – Modulus, a prime number for the Fp field or Irreducible polynomial for F2m field.
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from LTC_PKHA_ModR2() function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (C parameter for operation over F2m).
- size – Size in bytes of curve points and parameters
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

- *infinity* – **[out]** Output true if the result is point of infinity, and false otherwise. Writing of this output will be ignored if the argument is NULL.

Returns

Operation status.

```
struct _ltc_pkha_ecc_point_t
#include <fsl_ltc.h> PKHA ECC point structure
```

Public Members

```
uint8_t *X
    X coordinate (affine)
uint8_t *Y
    Y coordinate (affine)
```

2.58 LTC Blocking APIs

2.59 MCM: Miscellaneous Control Module

```
FSL_MCM_DRIVER_VERSION
MCM driver version.
```

Enum *_mcm_interrupt_flag*. Interrupt status flag mask. .

Values:

```
enumerator kMCM_CacheWriteBuffer
    Cache Write Buffer Error Enable.
enumerator kMCM_ParityError
    Cache Parity Error Enable.
enumerator kMCM_FPUInvalidOperation
    FPU Invalid Operation Interrupt Enable.
enumerator kMCM_FPUDivideByZero
    FPU Divide-by-zero Interrupt Enable.
enumerator kMCM_FPUOverflow
    FPU Overflow Interrupt Enable.
enumerator kMCM_FPUUnderflow
    FPU Underflow Interrupt Enable.
enumerator kMCM_FPUInexact
    FPU Inexact Interrupt Enable.
enumerator kMCM_FPUInputDenormalInterrupt
    FPU Input Denormal Interrupt Enable.
```

```
typedef union _mcm_buffer_fault_attribute mcm_buffer_fault_attribute_t
    The union of buffer fault attribute.
```

```
typedef union _mcm_lmem_fault_attribute mcm_lmem_fault_attribute_t
    The union of LMEM fault attribute.
```

static inline void MCM_EnableCrossbarRoundRobin(MCM_Type *base, bool enable)
Enables/Disables crossbar round robin.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
 - **true** Enable crossbar round robin.
 - **false** disable crossbar round robin.

static inline void MCM_EnableInterruptStatus(MCM_Type *base, uint32_t mask)
Enables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(`mcm_interrupt_flag`).

static inline void MCM_DisableInterruptStatus(MCM_Type *base, uint32_t mask)
Disables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(`mcm_interrupt_flag`).

static inline uint16_t MCM_GetInterruptStatus(MCM_Type *base)
Gets the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline void MCM_ClearCacheWriteBufferErroStatus(MCM_Type *base)
Clears the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultAddress(MCM_Type *base)
Gets buffer fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetBufferFaultAttribute(MCM_Type *base, *mcm_buffer_fault_attribute_t*
*bufferfault)

Gets buffer fault attributes.

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultData(MCM_Type *base)
Gets buffer fault data.

Parameters

- base – MCM peripheral base address.

static inline void MCM_LimitCodeCachePeripheralWriteBuffering(MCM_Type *base, bool enable)
Limit code cache peripheral write buffering.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
 - **true** Enable limit code cache peripheral write buffering.
 - **false** disable limit code cache peripheral write buffering.

static inline void MCM_BypassFixedCodeCacheMap(MCM_Type *base, bool enable)
Bypass fixed code cache map.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
 - **true** Enable bypass fixed code cache map.
 - **false** disable bypass fixed code cache map.

static inline void MCM_EnableCodeBusCache(MCM_Type *base, bool enable)
Enables/Disables code bus cache.

Parameters

- base – MCM peripheral base address.
- enable – Used to disable/enable code bus cache.
 - **true** Enable code bus cache.
 - **false** disable code bus cache.

static inline void MCM_ForceCodeCacheToNoAllocation(MCM_Type *base, bool enable)
Force code cache to no allocation.

Parameters

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
 - **true** Force code cache to no allocation.
 - **false** Force code cache to allocation.

static inline void MCM_EnableCodeCacheWriteBuffer(MCM_Type *base, bool enable)
Enables/Disables code cache write buffer.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
 - **true** Enable code cache write buffer.
 - **false** Disable code cache write buffer.

static inline void MCM_ClearCodeBusCache(MCM_Type *base)
Clear code bus cache.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnablePcParityFaultReport(MCM_Type *base, bool enable)
Enables/Disables PC Parity Fault Report.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
 - **true** Enable PC Parity Fault Report.
 - **false** disable PC Parity Fault Report.

static inline void MCM_EnablePcParity(MCM_Type *base, bool enable)
Enables/Disables PC Parity.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
 - **true** Enable PC Parity.
 - **false** disable PC Parity.

static inline void MCM_LockConfigState(MCM_Type *base)
Lock the configuration state.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnableCacheParityReporting(MCM_Type *base, bool enable)
Enables/Disables cache parity reporting.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
 - **true** Enable cache parity reporting.
 - **false** disable cache parity reporting.

static inline uint32_t MCM_GetLmemFaultAddress(MCM_Type *base)
Gets LMEM fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetLmemFaultAttribute(MCM_Type *base, *mcm_lmem_fault_attribute_t*
*lmemFault)

Get LMEM fault attributes.

Parameters

- base – MCM peripheral base address.

static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)
Gets LMEM fault data.

Parameters

- base – MCM peripheral base address.

MCM_LMFATR_TYPE_MASK

MCM_LMFATR_MODE_MASK

MCM_LMFATR_BUFF_MASK

MCM_LMFATR_CACH_MASK

MCM_ISCR_STAT_MASK

FSL_COMPONENT_ID

union `_mcm_buffer_fault_attribute`

#include <fsl_mcm.h> The union of buffer fault attribute.

Public Members

uint32_t attribute

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

struct `_mcm_buffer_fault_attribute._mcm_buffer_fault_attribut` attribute_memory

struct `_mcm_buffer_fault_attribut`

#include <fsl_mcm.h>

Public Members

uint32_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32_t busErrorSize

Indicates the size of the cache write buffer access.

uint32_t busErrorAccess

Indicates the type of system bus access.

uint32_t busErrorMasterID

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32_t busErrorOverrun

Indicates if another cache write buffer bus error is detected.

union `_mcm_lmem_fault_attribute`

#include <fsl_mcm.h> The union of LMEM fault attribute.

Public Members

uint32_t attribute

Indicates the attributes of the LMEM fault detected.

struct `_mcm_lmem_fault_attribute._mcm_lmem_fault_attribut` attribute_memory

struct `_mcm_lmem_fault_attribut`

#include <fsl_mcm.h>

Public Members

uint32_t parityFaultProtectionSignal

Indicates the features of parity fault protection signal.

uint32_t parityFaultMasterSize

Indicates the parity fault master size.

uint32_t parityFaultWrite

Indicates the parity fault is caused by read or write.

uint32_t backdoorAccess

Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32_t parityFaultSyndrome

Indicates the parity fault syndrome.

uint32_t overrun

Indicates the number of faultss.

2.60 MSCM: Miscellaneous System Control

FSL_MSCM_DRIVER_VERSION

MSCM driver version 2.0.0.

```
typedef struct _mscm_uid mscm_uid_t
```

```
static inline void MSCM_GetUID(MSCM_Type *base, mscm_uid_t *uid)
```

Get MSCM UID.

Parameters

- base – MSCM peripheral base address.
- uid – Pointer to an uid struct.

```
static inline void MSCM_SetSecureIrqParameter(MSCM_Type *base, const uint32_t parameter)
```

Set MSCM Secure Irq.

Parameters

- base – MSCM peripheral base address.
- parameter – Value to be write to SECURE_IRQ.

```
static inline uint32_t MSCM_GetSecureIrq(MSCM_Type *base)
```

Get MSCM Secure Irq.

Parameters

- base – MSCM peripheral base address.

Returns

MSCM Secure Irq.

FSL_COMPONENT_ID

```
struct _mscm_uid
```

```
#include <fsl_mscm.h>
```

2.61 MU: Messaging Unit

`uint32_t MU_GetInstance(MU_Type *base)`

Get the MU instance index.

Parameters

- `base` – MU peripheral base address.

Returns

MU instance index.

`void MU_Init(MU_Type *base)`

Initializes the MU module.

This function enables the MU clock only.

Parameters

- `base` – MU peripheral base address.

`void MU_Deinit(MU_Type *base)`

De-initializes the MU module.

This function disables the MU clock only.

Parameters

- `base` – MU peripheral base address.

`static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)`

Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } Wait for TX0 register empty.
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL); Write message to the TX0 register.
```

Parameters

- `base` – MU peripheral base address.
- `regIndex` – TX register index, see `mu_msg_reg_index_t`.
- `msg` – Message to send.

`status_t MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)`

Blocks to send a message.

This function waits until the TX register is empty and sends the message. If `MU1_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and returns `kStatus_Timeout`.

Parameters

- `base` – MU peripheral base address.
- `regIndex` – MU message register, see `mu_msg_reg_index_t`.
- `msg` – Message to send.

Return values

- `kStatus_Success` – Message sent successfully.
- `kStatus_Timeout` – Timeout occurred while waiting for TX register to be empty.

Returns

status_t

`static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)`

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
    } Wait for the RX0 register full.
msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0); Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see `mu_msg_reg_index_t`.

Returns

The received message.

`status_t MU_ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)`

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If `MU_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `kStatus_Timeout`.

This function provides the same blocking behavior as `MU_ReceiveMsg()` but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

Note: Both `MU_ReceiveMsg()` and `MU_ReceiveMsgTimeout()` are blocking functions. The difference is that this function includes timeout protection while `MU_ReceiveMsg()` waits indefinitely.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see `mu_msg_reg_index_t`.
- readValue – Pointer to store the received message.

Return values

- `kStatus_Success` – Message received successfully.
- `kStatus_InvalidArgument` – Invalid readValue pointer.
- `kStatus_Timeout` – Timeout occurred while waiting for RX register to be full.

Returns

status_t

`uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)`

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

Note: Both `MU_ReceiveMsg()` and `MU_ReceiveMsgTimeout()` are blocking functions. The difference is that `MU_ReceiveMsgTimeout()` includes timeout protection while this function waits indefinitely.

Warning: This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use `MU_ReceiveMsgTimeout()` instead.

Parameters

- `base` – MU peripheral base address.
- `regIndex` – RX register index, see `mu_msg_reg_index_t`.

Returns

The received message.

```
static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)
```

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag `kMU_FlagsUpdatingFlag` asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag `kMU_FlagsUpdatingFlag` is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag `kMU_FlagsUpdatingFlag` is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
    } Wait for previous MU flags updating.
}

MU_SetFlagsNonBlocking(base, 0U); Set the mU flags.
```

Parameters

- `base` – MU peripheral base address.
- `flags` – The 3-bit MU flags to set.

```
status_t MU_SetFlags(MU_Type *base, uint32_t flags)
```

brief Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag `kMU_FlagsUpdatingFlag` asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag `kMU_FlagsUpdatingFlag` is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag `kMU_FlagsUpdatingFlag` cleared and sets the 3-bit MU flags.

If `MU1_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations and return `kStatus_Timeout`.

return `status_t` `retval` `kStatus_Success` Flags were set successfully. `retval` `kStatus_Timeout` Timeout occurred while waiting for flags to update.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

```
static inline uint32_t MU_GetFlags(MU_Type *base)
```

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

```
uint32_t MU_GetStatusFlags(MU_Type *base)
```

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See `_mu_status_flags`.

```
uint32_t flags;
flags = MU_GetStatusFlags(base); Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

If there are more than 4 general purpose interrupts, use `MU_GetGeneralPurposeStatusFlags`.

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU status flags, see `_mu_status_flags`.

```
static inline uint32_t MU_GetInterruptsPending(MU_Type *base)
```

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. `kMU_Tx0EmptyFlag` `kMU_Tx1EmptyFlag` `kMU_Tx2EmptyFlag` `kMU_Tx3EmptyFlag` `kMU_Rx0FullFlag` `kMU_Rx1FullFlag` `kMU_Rx2FullFlag` `kMU_Rx3FullFlag` `kMU_GenInt0Flag` `kMU_GenInt1Flag` `kMU_GenInt2Flag` `kMU_GenInt3Flag`

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

```
static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t flags)
```

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See `_mu_status_flags`.

```
Clear general interrupt 0 and general interrupt 1 pending flags.
MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);
```

If there are more than 4 general purpose interrupts, use `MU_ClearGeneralPurposeStatusFlags`.

Parameters

- `base` – MU peripheral base address.
- `flags` – Bit mask of the MU status flags. See `_mu_status_flags`. Only the following flags can be cleared by software, other flags are cleared by hardware:
 - `kMU_GenInt0Flag`
 - `kMU_GenInt1Flag`
 - `kMU_GenInt2Flag`
 - `kMU_GenInt3Flag`
 - `kMU_MuResetInterruptFlag`
 - `#kMU_OtherSideEnterRunInterruptFlag`
 - `#kMU_OtherSideEnterHaltInterruptFlag`
 - `#kMU_OtherSideEnterWaitInterruptFlag`
 - `#kMU_OtherSideEnterStopInterruptFlag`
 - `#kMU_OtherSideEnterPowerDownInterruptFlag`
 - `#kMU_ResetAssertInterruptFlag`
 - `#kMU_HardwareResetInterruptFlag`

```
static inline void MU_EnableInterrupts(MU_Type *base, uint32_t interrupts)
```

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See `_mu_interrupt_enable`.

```
Enable general interrupt 0 and TX0 empty interrupt.
MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use `MU_EnableGeneralPurposeInterrupts`.

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

```
static inline void MU_DisableInterrupts(MU_Type *base, uint32_t interrupts)
```

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See `_mu_interrupt_enable`.

```
Disable general interrupt 0 and TX0 empty interrupt.
MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use `MU_DisableGeneralPurposeInterrupts`.

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the MU interrupts. See `_mu_interrupt_enable`.

status_t MU_TriggerInterrupts(MU_Type *base, uint32_t interrupts)

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `_mu_interrupt_trigger`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

If there are more than 4 general purpose interrupts, use `MU_TriggerGeneralPurposeInterrupts`.

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the interrupts to trigger. See `_mu_interrupt_trigger`.

Return values

- `kStatus_Success` – Interrupts have been triggered successfully.
- `kStatus_Fail` – Previous interrupts have not been accepted.

static inline void MU_EnableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Enables the MU general purpose interrupts.

This function enables the MU general purpose interrupts. The interrupts to enable should be passed in as bit mask of `mu_general_purpose_interrupt_t`. The function `MU_EnableInterrupts` only support general interrupt 0~3, this function supports all general interrupts.

For example, to enable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the MU general purpose interrupts, see `mu_general_purpose_interrupt_t`.

static inline void MU_DisableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Disables the MU general purpose interrupts.

This function disables the MU general purpose interrupts. The interrupts to disable should be passed in as bit mask of `mu_general_purpose_interrupt_t`. The function `MU_DisableInterrupts` only support general interrupt 0~3, this function supports all general interrupts.

For example, to disable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the MU general purpose interrupts. see `mu_general_purpose_interrupt_t`.

```
static inline uint32_t MU_GetGeneralPurposeStatusFlags(MU_Type *base)
```

Gets the MU general purpose interrupt status flags.

This function returns the bit mask of the MU general purpose interrupt status flags. `MU_GetStatusFlags` can only get general purpose interrupt status 0~3, this function can get all general purpose interrupts status.

This example shows to check whether general purpose interrupt 0 and 3 happened.

```
uint32_t flags;
flags = MU_GetGeneralPurposeStatusFlags(base);
if (kMU_GeneralPurposeInterrupt0 & flags)
{
}
if (kMU_GeneralPurposeInterrupt3 & flags)
{
}
```

Parameters

- `base` – MU peripheral base address.

Returns

Bit mask of the MU general purpose interrupt status flags.

```
static inline void MU_ClearGeneralPurposeStatusFlags(MU_Type *base, uint32_t flags)
```

Clear the MU general purpose interrupt status flags.

This function clears the specific MU general purpose interrupt status flags. The flags to clear should be passed in as bit mask. `mu_general_purpose_interrupt_t_mu_status_flags`.

Example to clear general purpose interrupt 0 and general interrupt 1 pending flags.

```
MU_ClearGeneralPurposeStatusFlags(base, kMU_GeneralPurposeInterrupt0 | kMU_
↳GeneralPurposeInterrupt1);
```

Parameters

- `base` – MU peripheral base address.
- `flags` – Bit mask of the MU general purpose interrupt status flags. See `mu_general_purpose_interrupt_t`.

```
static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)
```

Return the RX status flags in reverse numerical order.

This function return the RX status flags in reverse order. Note: RFn bits of SR[3-0](mu status register) are mapped in ascending numerical order: RF0 -> SR[0] RF1 -> SR[1] RF2 -> SR[2] RF3 -> SR[3] This function will return these bits in reverse numerical order(RF3->RF1) to comply with `MU_GetRxStatusFlags()` of mu driver. See `MU_GetRxStatusFlags()` from `drivers/mu/fsl_mu.h`

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- `base` – MU peripheral base address.

Returns

MU RX status flags in reverse order

`status_t` MU_TriggerGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Triggers general purpose interrupts to the other core.

This function triggers the specific general purpose interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `mu_general_purpose_interrupt_t`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
status_t status;
status = MU_TriggerGeneralPurposeInterrupts(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt2);

if (kStatus_Success != status)
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

Parameters

- `base` – MU peripheral base address.
- `interrupts` – Bit mask of the interrupts to trigger. See `mu_general_purpose_interrupt_t`.

Return values

- `kStatus_Success` – Interrupts have been triggered successfully.
- `kStatus_Fail` – Previous interrupts have not been accepted.

`void` MU_BootOtherCore(MU_Type *base, mu_core_boot_mode_t mode)

Boots the other core.

This function boots the other core with a boot configuration.

Parameters

- `base` – MU peripheral base address.
- `mode` – The other core boot mode.

`void` MU_HoldOtherCoreReset(MU_Type *base)

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

Parameters

- `base` – MU peripheral base address.

`static inline status_t` MU_ResetBothSides(MU_Type *base)

Resets the MU for both A side and B side.

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

If `MU1_BUSY_POLL_COUNT` is defined and non-zero, the function will timeout after the specified number of polling iterations if waiting for the other side to come out of reset takes too long.

Note: For some platforms, only MU side A could use this function, check reference manual for details.

Parameters

- base – MU peripheral base address.

Return values

- kStatus_Success – The MU was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other side to come out of reset.

Returns

status_t

status_t MU_HardwareResetOtherCore(MU_Type *base, bool waitReset, bool holdReset, mu_core_boot_mode_t bootMode)

Hardware reset the other core.

This function resets the other core, the other core could mask the hardware reset by calling MU_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU_BootOtherCore to control the other core reset workflow.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout if waiting for the other core to enter or exit reset takes too long.

Example 1: Reset the other core, and no hold reset

```
MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later. Here the other core enters reset, and the reset is hold

```
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
```

Current core boot the other core when necessary.

```
MU_BootOtherCore(MU_A, bootMode);
```

Note: The feature waitReset, holdReset, and bootMode might be not supported for some platforms. waitReset is only available for platforms that FSL_FEATURE_MU_NO_CORE_STATUS not defined as 1 and FSL_FEATURE_MU_HAS_RESET_ASSERT_INT not defined as 0. holdReset is only available for platforms that FSL_FEATURE_MU_HAS_RSTH not defined as 0. bootMode is only available for platforms that FSL_FEATURE_MU_HAS_BOOT not defined as 0.

Parameters

- base – MU peripheral base address.
- waitReset – Wait the other core enters reset. Only work when there is CSSR0[RAIP].
 - true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
 - false: Don't wait the reset.
- holdReset – Hold the other core reset or not. Only work when there is CCR0[RSTH].

- true: Hold the other core in reset, this function returns directly when the other core enters reset.
- false: Don't hold the other core in reset, this function waits until the other core out of reset.
- bootMode – Boot mode of the other core, if holdReset is true, this parameter is useless.

Return values

- kStatus_Success – The other core was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other core to enter or exit reset.

Returns

status_t

FSL_MU_DRIVER_VERSION

MU driver version.

enum _mu_status_flags

MU status flags.

Values:

enumerator kMU_Tx0EmptyFlag
TX0 empty.

enumerator kMU_Tx1EmptyFlag
TX1 empty.

enumerator kMU_Tx2EmptyFlag
TX2 empty.

enumerator kMU_Tx3EmptyFlag
TX3 empty.

enumerator kMU_Rx0FullFlag
RX0 full.

enumerator kMU_Rx1FullFlag
RX1 full.

enumerator kMU_Rx2FullFlag
RX2 full.

enumerator kMU_Rx3FullFlag
RX3 full.

enumerator kMU_GenInt0Flag
General purpose interrupt 0 pending.

enumerator kMU_GenInt1Flag
General purpose interrupt 1 pending.

enumerator kMU_GenInt2Flag
General purpose interrupt 2 pending.

enumerator kMU_GenInt3Flag
General purpose interrupt 3 pending.

enumerator kMU_RxFullPendingFlag
Any RX full flag is pending.

enumerator kMU_TxEmptyPendingFlag

Any TX empty flag is pending.

enumerator kMU_GenIntPendingFlag

Any general interrupt flag is pending.

enumerator kMU_EventPendingFlag

MU event pending.

enumerator kMU_FlagsUpdatingFlag

MU flags update is on-going.

enumerator kMU_MuInResetFlag

MU of any side is in reset.

enumerator kMU_MuResetInterruptFlag

The other side initializes MU reset.

enum _mu_interrupt_enable

MU interrupt source to enable.

Values:

enumerator kMU_Tx0EmptyInterruptEnable

TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable

TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable

TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable

TX3 empty.

enumerator kMU_Rx0FullInterruptEnable

RX0 full.

enumerator kMU_Rx1FullInterruptEnable

RX1 full.

enumerator kMU_Rx2FullInterruptEnable

RX2 full.

enumerator kMU_Rx3FullInterruptEnable

RX3 full.

enumerator kMU_GenInt0InterruptEnable

General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable

General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable

General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable

General purpose interrupt 3.

enumerator kMU_MuResetInterruptEnable

The other side initializes MU reset.

enum `_mu_interrupt_trigger`

MU interrupt that could be triggered to the other core.

Values:

enumerator `kMU_GenInt0InterruptTrigger`

General purpose interrupt 0.

enumerator `kMU_GenInt1InterruptTrigger`

General purpose interrupt 1.

enumerator `kMU_GenInt2InterruptTrigger`

General purpose interrupt 2.

enumerator `kMU_GenInt3InterruptTrigger`

General purpose interrupt 3.

enum `_mu_msg_reg_index`

MU message register index.

Values:

enumerator `kMU_MsgReg0`

Message register 0.

enumerator `kMU_MsgReg1`

Message register 1.

enumerator `kMU_MsgReg2`

Message register 2.

enumerator `kMU_MsgReg3`

Message register 3.

enum `_mu_general_purpose_interrupt`

MU general purpose interrupts.

Values:

enumerator `kMU_GeneralPurposeInterrupt0`

General purpose interrupt 0

enumerator `kMU_GeneralPurposeInterrupt1`

General purpose interrupt 1

enumerator `kMU_GeneralPurposeInterrupt2`

General purpose interrupt 2

enumerator `kMU_GeneralPurposeInterrupt3`

General purpose interrupt 3

typedef enum `_mu_msg_reg_index` `mu_msg_reg_index_t`

MU message register index.

typedef enum `_mu_general_purpose_interrupt` `mu_general_purpose_interrupt_t`

MU general purpose interrupts.

`MU_CORE_INTR(intr)`

`MU_MISC_INTR(intr)`

`MU_TX_INTR(intr)`

`MU_RX_INTR(intr)`

MU_GI_INTR(*intr*)
MU_GET_CORE_INTR(*intrs*)
MU_GET_TX_INTR(*intrs*)
MU_GET_RX_INTR(*intrs*)
MU_GET_GI_INTR(*intrs*)
MU_CORE_FLAG(*flag*)
MU_STAT_FLAG(*flag*)
MU_TX_FLAG(*flag*)
MU_RX_FLAG(*flag*)
MU_GI_FLAG(*flag*)
MU_GET_CORE_FLAG(*flags*)
MU_GET_STAT_FLAG(*flags*)
MU_GET_TX_FLAG(*flags*)
MU_GET_RX_FLAG(*flags*)
MU_GET_GI_FLAG(*flags*)
MU1_BUSY_POLL_COUNT

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if a core becomes unresponsive.

2.62 PORT: Port Control and Interrupts

static inline void PORT_GetVersionInfo(PORT_Type *base, *port_version_info_t* *info)

Get PORT version information.

Parameters

- *base* – PORT peripheral base pointer
- *info* – PORT version information

static inline void PORT_SecletPortVoltageRange(PORT_Type *base, *port_voltage_range_t* range)

Get PORT version information.

Note: : PORTA_CONFIG[RANGE] controls the voltage ranges of Port A, B, and C. Read or write PORTB_CONFIG[RANGE] and PORTC_CONFIG[RANGE] does not take effect.

Parameters

- base – PORT peripheral base pointer
- range – port voltage range

```
static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const port_pin_config_t *config)
```

Sets the port PCR register.

This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT PCR register configuration structure.

```
static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const port_pin_config_t *config)
```

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```
Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

```
static inline void PORT_SetMultipleInterruptPinsConfig(PORT_Type *base, uint32_t mask, port_interrupt_t config)
```

Sets the port interrupt configuration in PCR register for multiple pins.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT pin interrupt configuration.

- #kPORT_InterruptOrDMADisabled: Interrupt/DMA request disabled.
- #kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
- #kPORT_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
- #kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
- #kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
- #kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
- #kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
- #kPORT_InterruptLogicZero : Interrupt when logic zero.
- #kPORT_InterruptRisingEdge : Interrupt on rising edge.
- #kPORT_InterruptFallingEdge: Interrupt on falling edge.
- #kPORT_InterruptEitherEdge : Interrupt on either edge.
- #kPORT_InterruptLogicOne : Interrupt when logic one.
- #kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
- #kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

static inline void PORT_SetPinMux(PORT_Type *base, uint32_t pin, port_mux_t mux)
Configures the pin muxing.

Note: : This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- mux – pin muxing slot selection.
 - kPORT_PinDisabledOrAnalog: Pin disabled or work in analog function.
 - kPORT_MuxAsGpio : Set as GPIO.
 - kPORT_MuxAlt2 : chip-specific.
 - kPORT_MuxAlt3 : chip-specific.
 - kPORT_MuxAlt4 : chip-specific.
 - kPORT_MuxAlt5 : chip-specific.
 - kPORT_MuxAlt6 : chip-specific.
 - kPORT_MuxAlt7 : chip-specific.

static inline void PORT_EnablePinsDigitalFilter(PORT_Type *base, uint32_t mask, bool enable)
Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

```
static inline void PORT_SetDigitalFilterConfig(PORT_Type *base, const
                                             port_digital_filter_config_t *config)
```

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

```
static inline void PORT_SetPinDriveStrength(PORT_Type *base, uint32_t pin, uint8_t strength)
```

Configures the port pin drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- strength – PORT pin drive strength
 - kPORT_LowDriveStrength = 0U - Low-drive strength is configured.
 - kPORT_HighDriveStrength = 1U - High-drive strength is configured.

```
static inline void PORT_EnablePinDoubleDriveStrength(PORT_Type *base, uint32_t pin, bool
                                                    enable)
```

Enables the port pin double drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- enable – PORT pin drive strength configuration.

```
static inline void PORT_SetPinPullValue(PORT_Type *base, uint32_t pin, uint8_t value)
```

Configures the port pin pull value.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- value – PORT pin pull value
 - kPORT_LowPullResistor = 0U - Low internal pull resistor value is selected.
 - kPORT_HighPullResistor = 1U - High internal pull resistor value is selected.

```
static inline uint32_t PORT_GetEFTDetectFlags(PORT_Type *base)
```

Get EFT detect flags.

Parameters

- base – PORT peripheral base pointer

Returns

EFT detect flags

static inline void PORT_EnableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
 Enable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_DisableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
 Disable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_ClearAllLowEFTDetectors(PORT_Type *base)
 Clear all low EFT detector.

Note: : Port B and Port C pins share the same EFT detector clear control from PORTC_EDCR register. Any write to the PORTB_EDCR does not take effect.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_ClearAllHighEFTDetectors(PORT_Type *base)
 Clear all high EFT detector.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

FSL_PORT_DRIVER_VERSION

PORT driver version.

enum __port_pull

Internal resistor pull feature selection.

Values:

enumerator kPORT_PullDisable

Internal pull-up/down resistor is disabled.

enumerator kPORT_PullDown

Internal pull-down resistor is enabled.

enumerator kPORT_PullUp

Internal pull-up resistor is enabled.

enum __port_pull_value

Internal resistor pull value selection.

Values:

enumerator kPORT_LowPullResistor

Low internal pull resistor value is selected.

enumerator kPORT_HighPullResistor
High internal pull resistor value is selected.

enum _port_slew_rate
Slew rate selection.

Values:

enumerator kPORT_FastSlewRate
Fast slew rate is configured.

enumerator kPORT_SlowSlewRate
Slow slew rate is configured.

enum _port_open_drain_enable
Open Drain feature enable/disable.

Values:

enumerator kPORT_OpenDrainDisable
Open drain output is disabled.

enumerator kPORT_OpenDrainEnable
Open drain output is enabled.

enum _port_passive_filter_enable
Passive filter feature enable/disable.

Values:

enumerator kPORT_PassiveFilterDisable
Passive input filter is disabled.

enumerator kPORT_PassiveFilterEnable
Passive input filter is enabled.

enum _port_drive_strength
Configures the drive strength.

Values:

enumerator kPORT_LowDriveStrength
Low-drive strength is configured.

enumerator kPORT_HighDriveStrength
High-drive strength is configured.

enum _port_drive_strength1
Configures the drive strength1.

Values:

enumerator kPORT_NormalDriveStrength
Normal drive strength

enumerator kPORT_DoubleDriveStrength
Double drive strength

enum _port_invert_input
Digital input is not inverted or it is inverted.

Values:

enumerator kPORT_InputNormal
Digital input is not inverted

enumerator kPORT_InputInvert
Digital input is inverted

enum __port_lock_register
Unlock/lock the pin control register field[15:0].

Values:

enumerator kPORT_UnlockRegister
Pin Control Register fields [15:0] are not locked.

enumerator kPORT_LockRegister
Pin Control Register fields [15:0] are locked.

enum __port_mux
Pin mux selection.

Values:

enumerator kPORT_PinDisabledOrAnalog
Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT_MuxAsGpio
Corresponding pin is configured as GPIO.

enumerator kPORT_MuxAlt0
Chip-specific

enumerator kPORT_MuxAlt1
Chip-specific

enumerator kPORT_MuxAlt2
Chip-specific

enumerator kPORT_MuxAlt3
Chip-specific

enumerator kPORT_MuxAlt4
Chip-specific

enumerator kPORT_MuxAlt5
Chip-specific

enumerator kPORT_MuxAlt6
Chip-specific

enumerator kPORT_MuxAlt7
Chip-specific

enumerator kPORT_MuxAlt8
Chip-specific

enumerator kPORT_MuxAlt9
Chip-specific

enumerator kPORT_MuxAlt10
Chip-specific

enumerator kPORT_MuxAlt11
Chip-specific

enumerator kPORT_MuxAlt12
Chip-specific

enumerator kPORT_MuxAlt13
Chip-specific

enumerator kPORT_MuxAlt14
Chip-specific

enumerator kPORT_MuxAlt15
Chip-specific

enum _port_digital_filter_clock_source
Digital filter clock source selection.

Values:

enumerator kPORT_BusClock
Digital filters are clocked by the bus clock.

enumerator kPORT_LpoClock
Digital filters are clocked by the 1 kHz LPO clock.

enum _port_voltage_range
PORT voltage range.

Values:

enumerator kPORT_VoltageRange1Dot71V_3Dot6V
Port voltage range is 1.71 V - 3.6 V.

enumerator kPORT_VoltageRange2Dot70V_3Dot6V
Port voltage range is 2.70 V - 3.6 V.

typedef enum _port_mux port_mux_t
Pin mux selection.

typedef enum _port_digital_filter_clock_source port_digital_filter_clock_source_t
Digital filter clock source selection.

typedef struct _port_digital_filter_config port_digital_filter_config_t
PORT digital filter feature configuration definition.

typedef struct _port_pin_config port_pin_config_t
PORT pin configuration structure.

typedef struct _port_version_info port_version_info_t
PORT version information.

typedef enum _port_voltage_range port_voltage_range_t
PORT voltage range.

FSL_COMPONENT_ID

struct _port_digital_filter_config
#include <fsl_port.h> PORT digital filter feature configuration definition.

Public Members

uint32_t digitalFilterWidth
Set digital filter width

port_digital_filter_clock_source_t clockSource
Set digital filter clockSource

struct _port_pin_config
#include <fsl_port.h> PORT pin configuration structure.

Public Members

uint16_t pullSelect
No-pull/pull-down/pull-up select

uint16_t pullValueSelect
Pull value select

uint16_t slewRate
Fast/slow slew rate Configure

uint16_t passiveFilterEnable
Passive filter enable/disable

uint16_t openDrainEnable
Open drain enable/disable

uint16_t driveStrength
Fast/slow drive strength configure

uint16_t driveStrength1
Normal/Double drive strength enable/disable

uint16_t invertInput
Invert Input Configure

uint16_t lockRegister
Lock/unlock the PCR field[15:0]

struct __port__version__info
#include <fsl_port.h> PORT version information.

Public Members

uint16_t feature
Feature Specification Number.

uint8_t minor
Minor Version Number.

uint8_t major
Major Version Number.

2.63 RTC: Real Time Clock

void RTC_Init(RTC_Type *base, const *rtc_config_t* *config)
Ungates the RTC clock and configures the peripheral for basic operation.
This function issues a software reset if the timer invalid flag is set.

Note: This API should be called at the beginning of the application using the RTC driver.

Parameters

- base – RTC peripheral base address
- config – Pointer to the user's RTC configuration structure.

static inline void RTC_Deinit(RTC_Type *base)

Stops the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

void RTC_GetDefaultConfig(*rtc_config_t* *config)

Fills in the RTC config struct with the default settings.

The default values are as follows.

```
config->clockOutput = false;
config->wakeupSelect = false;
config->updateMode = false;
config->supervisorAccess = false;
config->compensationInterval = 0;
config->compensationTime = 0;
```

Parameters

- config – Pointer to the user's RTC configuration structure.

status_t RTC_SetDatetime(RTC_Type *base, const *rtc_datetime_t* *datetime)

Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the RTC
kStatus_InvalidArgument: Error because the datetime format is incorrect

void RTC_GetDatetime(RTC_Type *base, *rtc_datetime_t* *datetime)

Gets the RTC time and stores it in the given time structure.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t RTC_SetAlarm(RTC_Type *base, const *rtc_datetime_t* *alarmTime)

Sets the RTC alarm time.

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm
kStatus_InvalidArgument: Error because the alarm datetime format is incorrect
kStatus_Fail: Error because the alarm time has already passed

void RTC_GetAlarm(RTC_Type *base, rtc_datetime_t *datetime)

Returns the RTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected RTC interrupts.

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `rtc_interrupt_enable_t`

void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `rtc_interrupt_enable_t`

uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled RTC interrupts.

Parameters

- base – RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `rtc_interrupt_enable_t`

uint32_t RTC_GetStatusFlags(RTC_Type *base)

Gets the RTC status flags.

Parameters

- base – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `rtc_status_flags_t`

void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clears the RTC status flags.

Parameters

- base – RTC peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `rtc_status_flags_t`

static inline void RTC_StartTimer(RTC_Type *base)

Starts the RTC time counter.

After calling this function, the timer counter increments once a second provided `SR[TOF]` or `SR[TIF]` are not set.

Parameters

- base – RTC peripheral base address

static inline void RTC_StopTimer(RTC_Type *base)

Stops the RTC time counter.

RTC's seconds register can be written to only when the timer is stopped.

Parameters

- base – RTC peripheral base address

void RTC_GetMonotonicCounter(RTC_Type *base, uint64_t *counter)

Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.

Parameters

- base – RTC peripheral base address
- counter – Pointer to variable where the value is stored.

void RTC_SetMonotonicCounter(RTC_Type *base, uint64_t counter)

Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value. The Monotonic Overflow Flag in RTC_SR is cleared due to the API.

Parameters

- base – RTC peripheral base address
- counter – Counter value

status_t RTC_IncrementMonotonicCounter(RTC_Type *base)

Increments the Monotonic Counter by one.

Increments the Monotonic Counter (registers RTC_MCLR and RTC_MCHR accordingly) by setting the monotonic counter enable (MER[MCE]) and then writing to the RTC_MCLR register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

Parameters

- base – RTC peripheral base address

Returns

kStatus_Success: success
kStatus_Fail: error occurred, either time invalid or monotonic overflow flag was found

FSL_RTC_DRIVER_VERSION

Version 2.4.0

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_TimeInvalidInterruptEnable

Time invalid interrupt.

enumerator kRTC_TimeOverflowInterruptEnable

Time overflow interrupt.

enumerator kRTC_AlarmInterruptEnable

Alarm interrupt.

enumerator kRTC_MonotonicOverflowInterruptEnable

Monotonic Overflow Interrupt Enable

enumerator kRTC_SecondsInterruptEnable

Seconds interrupt.

enumerator kRTC_TestModeInterruptEnable

enumerator kRTC_FlashSecurityInterruptEnable

enumerator kRTC_TamperPinInterruptEnable

enumerator kRTC_SecurityModuleInterruptEnable

enumerator kRTC_LossOfClockInterruptEnable

enum `_rtc_status_flags`

List of RTC flags.

Values:

enumerator kRTC_TimeInvalidFlag

Time invalid flag

enumerator kRTC_TimeOverflowFlag

Time overflow flag

enumerator kRTC_AlarmFlag

Alarm flag

enumerator kRTC_MonotonicOverflowFlag

Monotonic Overflow Flag

enumerator kRTC_TamperInterruptDetectFlag

Tamper interrupt detect flag

enumerator kRTC_TestModeFlag

enumerator kRTC_FlashSecurityFlag

enumerator kRTC_TamperPinFlag

enumerator kRTC_SecurityTamperFlag

enumerator kRTC_LossOfClockTamperFlag

typedef enum `_rtc_interrupt_enable` `rtc_interrupt_enable_t`

List of RTC interrupts.

typedef enum `_rtc_status_flags` `rtc_status_flags_t`

List of RTC flags.

typedef struct `_rtc_datetime` `rtc_datetime_t`

Structure is used to hold the date and time.

typedef struct `_rtc_pin_config` `rtc_pin_config_t`

RTC pin config structure.

typedef struct `_rtc_config` `rtc_config_t`

RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the `RTC_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

static inline uint32_t RTC_GetTamperTimeSeconds(RTC_Type *base)

Get the RTC tamper time seconds.

Parameters

- base – RTC peripheral base address

static inline void RTC_Reset(RTC_Type *base)

Performs a software reset on the RTC module.

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

- base – RTC peripheral base address

static inline void RTC_EnableClockOutput(RTC_Type *base, bool enable)

Enables or disables the RTC 32 kHz clock output.

This function enables or disables the RTC 32 kHz clock output.

Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

struct _rtc_datetime

#include <fsl_rtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _rtc_pin_config

#include <fsl_rtc.h> RTC pin config structure.

Public Members

bool inputLogic

true: Tamper pin input data is logic one. false: Tamper pin input data is logic zero.

bool pinActiveLow

true: Tamper pin is active low. false: Tamper pin is active high.

bool filterEnable

true: Input filter is enabled on the tamper pin. false: Input filter is disabled on the tamper pin.

bool pullSelectNegate

true: Tamper pin pull resistor direction will negate the tamper pin. false: Tamper pin pull resistor direction will assert the tamper pin.

bool pullEnable

true: Pull resistor is enabled on tamper pin. false: Pull resistor is disabled on tamper pin.

struct `_rtc_config`

#include `<fsl_rtc.h>` RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the `RTC_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool clockOutput

true: The 32 kHz clock is not output to other peripherals; false: The 32 kHz clock is output to other peripherals

bool wakeupSelect

true: Wakeup pin outputs the 32 KHz clock; false: Wakeup pin used to wakeup the chip

bool updateMode

true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked

bool supervisorAccess

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported

uint32_t compensationInterval

Compensation interval that is written to the CIR field in RTC TCR Register

uint32_t compensationTime

Compensation time that is written to the TCR field in RTC TCR Register

2.64 SEMA42: Hardware Semaphores Driver

FSL_SEMA42_DRIVER_VERSION

SEMA42 driver version.

SEMA42 status return codes.

Values:

enumerator `kStatus_SEMA42_Busy`

SEMA42 gate has been locked by other processor.

enumerator `kStatus_SEMA42_Reseting`

SEMA42 gate reseting is ongoing.

enum `_sema42_gate_status`

SEMA42 gate lock status.

Values:

enumerator `kSEMA42_Unlocked`

The gate is unlocked.

enumerator `kSEMA42_LockedByProc0`

The gate is locked by processor 0.

enumerator `kSEMA42_LockedByProc1`

The gate is locked by processor 1.

enumerator `kSEMA42_LockedByProc2`

The gate is locked by processor 2.

enumerator `kSEMA42_LockedByProc3`

The gate is locked by processor 3.

enumerator `kSEMA42_LockedByProc4`

The gate is locked by processor 4.

enumerator `kSEMA42_LockedByProc5`

The gate is locked by processor 5.

enumerator `kSEMA42_LockedByProc6`

The gate is locked by processor 6.

enumerator `kSEMA42_LockedByProc7`

The gate is locked by processor 7.

enumerator `kSEMA42_LockedByProc8`

The gate is locked by processor 8.

enumerator `kSEMA42_LockedByProc9`

The gate is locked by processor 9.

enumerator `kSEMA42_LockedByProc10`

The gate is locked by processor 10.

enumerator `kSEMA42_LockedByProc11`

The gate is locked by processor 11.

enumerator `kSEMA42_LockedByProc12`

The gate is locked by processor 12.

enumerator `kSEMA42_LockedByProc13`

The gate is locked by processor 13.

enumerator `kSEMA42_LockedByProc14`

The gate is locked by processor 14.

typedef enum `_sema42_gate_status` `sema42_gate_status_t`

SEMA42 gate lock status.

void `SEMA42_Init(SEMA42_Type *base)`

Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either `SEMA42_ResetGate` or `SEMA42_ResetAllGates` function.

Parameters

- base – SEMA42 peripheral base address.

void SEMA42_Deinit(SEMA42_Type *base)

De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

- base – SEMA42 peripheral base address.

status_t SEMA42_TryLock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_SEMA42_Busy – Sema42 gate has been locked by another processor.

status_t SEMA42_Lock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns

status_t

static inline void SEMA42_Unlock(SEMA42_Type *base, uint8_t gateNum)

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

```
static inline sema42_gate_status_t SEMA42_GetGateStatus(SEMA42_Type *base, uint8_t
                                                    gateNum)
```

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

status Current status.

```
status_t SEMA42_ResetGate(SEMA42_Type *base, uint8_t gateNum)
```

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – SEMA42 gate is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

```
static inline status_t SEMA42_ResetAllGates(SEMA42_Type *base)
```

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.

Return values

- kStatus_Success – SEMA42 is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

```
SEMA42_GATE_NUM_RESET_ALL
```

The number to reset all SEMA42 gates.

```
SEMA42_GATEn(base, n)
```

SEMA42 gate n register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42_GATEn gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U: $0 \wedge 3 = 3$ $1 \wedge 3 = 2$ $2 \wedge 3 = 1$ $3 \wedge 3 = 0$ $4 \wedge 3 = 7$ $5 \wedge 3 = 6$ $6 \wedge 3 = 5$ $7 \wedge 3 = 4$...

```
SEMA42_BUSY_POLL_COUNT
```

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.65 SFA: Signal Frequency Analyser

`void SFA_GetDefaultConfig(sfa_config_t *config)`

Fill the SFA configuration structure with default settings.

The default values are:

```
config->mode = kSFA_FrequencyMeasurement0;
config->cutSelect = kSFA_CUTSelect0;
config->refSelect = kSFA_REFSelect0;
config->prediv = 0U;
config->trigStart = kSFA_TriggerStartSelect0;
config->startPolarity = kSFA_TriggerStartPolarityRiseEdge;
config->trigEnd = kSFA_TriggerEndSelect0;
config->endPolarity = kSFA_TriggerEndPolarityRiseEdge;
config->enableTrigMeasurement = false;
config->enableCUTPin = false;
config->cutTarget = 0xffffU;
config->refTarget = 0xffffffffU;
```

Parameters

- `config` – Pointer to the user configuration structure.

`void SFA_Init(SFA_Type *base)`

Initialize SFA.

Parameters

- `base` – SFA peripheral base address.

`status_t SFA_Deinit(SFA_Type *base)`

Clear counter, disable SFA and gate the SFA clock.

Parameters

- `base` – SFA peripheral base address.

Return values

- `kStatus_Success` – run success.
- `kStatus_Timeout` – timeout occurs.

`static inline void SFA_EnableCUTPin(SFA_Type *base, bool enable)`

Control the connection of the clock under test to an external pin.

Parameters

- `base` – SFA peripheral base address.
- `enable` – true: connect the clock under test and external pin. false: Disconnect the clock under test and external pin.

`static inline uint32_t SFA_GetStatusFlags(SFA_Type *base)`

Get SFA status flags.

Parameters

- `base` – SFA peripheral base address.

void SFA_ClearStatusFlag(SFA_Type *base, uint32_t mask)

Clear the SFA status flags.

Note: To clear kSFA_RefStoppedFlag, kSFA_CUTStoppedFlag, kSFA_MeasurementStartedFlag, and kSFA_ReferenceCounterTimeOutFlag, each counter will also be cleared.

Parameters

- base – SFA peripheral base address.
- mask – SFA status flag mask (see `_sfa_status_flags` for bit definition).

static inline void SFA_EnableInterrupts(SFA_Type *base, uint32_t mask)

Enable the selected SFA interrupt.

Parameters

- base – SFA peripheral base address.
- mask – The interrupt to enable (see `_sfa_interrupts_enable` for definition).

static inline void SFA_DisableInterrupts(SFA_Type *base, uint32_t mask)

Disable the selected SFA interrupt.

Parameters

- base – SFA peripheral base address.
- mask – The interrupt to disable (see `_sfa_interrupts_enable` for definition).

static inline uint8_t SFA_GetMode(SFA_Type *base)

Get SFA measurement mode.

Parameters

- base – SFA peripheral base address.

static inline uint8_t SFA_GetCUTPredivide(SFA_Type *base)

Get CUT predivide value.

Parameters

- base – SFA peripheral base address.

void SFA_InstallCallback(SFA_Type *base, *sfa_callback_t* function)

Install the callback function to be called when IRQ happens or measurement completes.

Parameters

- base – SFA peripheral base address.
- function – the SFA measure completed callback function.

void SFA_SetMeasureConfig(SFA_Type *base, const *sfa_config_t* *config)

Set Measurement options with the passed in configuration structure.

Parameters

- base – SFA peripheral base address.
- config – SFA configuration structure.

status_t SFA_MeasureBlocking(SFA_Type *base)

Start SFA measurement in blocking mode.

Parameters

- base – SFA peripheral base address.

Return values

- kStatus_SFA_MeasurementCompleted – SFA measure completes.
- kStatus_SFA_ReferenceCounterTimeout – reference counter timeout error happens.
- kStatus_SFA_CUTCounterTimeout – CUT counter time out happens.

void SFA_MeasureNonBlocking(SFA_Type *base)

Start measure sequence in NonBlocking mode.

This function performs nonblocking measurement by enabling sfa interrupt (Please enable the FreqGreaterThanMax and FreqLessThanMin interrupts individually as needed). The callback function must be installed before invoking this function.

Note: This function has different functions for different instances.

Parameters

- base – SFA peripheral base address.

void SFA_AbortMeasureSequence(SFA_Type *base)

Abort SFA measurement sequence.

Parameters

- base – SFA peripheral base address.

uint32_t SFA_CalculateFrequencyOrPeriod(SFA_Type *base, uint32_t refFrequency)

Calculate the frequency or period.

Parameters

- base – SFA peripheral base address.
- refFrequency – The reference clock frequency(BUS clock recommended).

static inline uint32_t SFA_GetCUTCounter(SFA_Type *base)

Get current count of the clock under test.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTTargetCount(SFA_Type *base, uint32_t count)

Set the target count for the clock under test.

Parameters

- base – SFA peripheral base address.
- count – target count for CUT.

static inline uint32_t SFA_GetCUTTargetCount(SFA_Type *base)

Get the target count of the clock under test.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTLowLimitClockCount(SFA_Type *base, uint32_t count)

Set CUT low limit clock count.

Parameters

- base – SFA peripheral base address.
- count – low limit count for CUT clock.

static inline uint32_t SFA_GetCUTLowLimitClockCount(SFA_Type *base)

Get CUT low limit clock count.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTHighLimitClockCount(SFA_Type *base, uint32_t count)

Set CUT high limit clock count.

Parameters

- base – SFA peripheral base address.
- count – high limit count for CUT clock.

static inline uint32_t SFA_GetCUTHighLimitClockCount(SFA_Type *base)

Get CUT high limit clock count.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFCounter(SFA_Type *base)

Get current count of the reference clock.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFTargetCount(SFA_Type *base, uint32_t count)

Set the target count for the reference clock.

Parameters

- base – SFA peripheral base address.
- count – target count for reference clock.

static inline uint32_t SFA_GetREFTargetCount(SFA_Type *base)

Get the target count of the reference clock.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFStartCount(SFA_Type *base)

Get saved reference clock counter which is loaded when measurement start.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFEndCount(SFA_Type *base)

Get saved reference clock counter which is loaded when measurement complete.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFLowLimitClockCount(SFA_Type *base, uint32_t count)

Set REF low limit clock count.

Parameters

- base – SFA peripheral base address.

- `count` – low limit count for REF clock.

```
static inline uint32_t SFA_GetREFLowLimitClockCount(SFA_Type *base)
```

Get REF low limit clock count.

Parameters

- `base` – SFA peripheral base address.

```
static inline void SFA_SetREFHighLimitClockCount(SFA_Type *base, uint32_t count)
```

Set REF high limit clock count.

Parameters

- `base` – SFA peripheral base address.
- `count` – high limit count for REF clock.

```
static inline uint32_t SFA_GetREFHighLimitClockCount(SFA_Type *base)
```

Get REF high limit clock count.

Parameters

- `base` – SFA peripheral base address.

FSL_SFA_DRVIER_VERSION

SFA driver version 2.1.3.

SFA_MEASUREMENT_START_TIMEOUT

Max loops to wait for SFA measurement started.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA_CUT_COUNTER_STOP_TIMEOUT

Max loops to wait for SFA CUT counter has stopped.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA_REF_COUNTER_STOP_TIMEOUT

Max loops to wait for SFA REF counter has stopped.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA status return codes.

enumeration `_sfa_status`

Values:

enumerator `kStatus_SFA_MeasurementCompleted`

Measurement completed

enumerator `kStatus_SFA_ReferenceCounterTimeout`

Reference counter timeout

enumerator `kStatus_SFA_CUTCounterTimeout`

CUT counter timeout

enumerator `kStatus_SFA_CUTClockFreqLessThanMinLimit`

CUT clock frequency less than minimum limit

enumerator `kStatus_SFA_CUTClockFreqGreaterThanMaxLimit`

CUT clock frequency greater than maximum limit

enum `_sfa_status_flags`

List of SFA status flags.

The following status register flags can be cleared on any write to REF_CNT.

- `kSFA_RefStoppedFlag`
- `kSFA_CutStoppedFlag`
- `kSFA_MeasurementStartedFlag`
- `kSFA_ReferenceCounterTimeOutFlag`

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator `kSFA_RefStoppedFlag`
Reference counter stopped flag

enumerator `kSFA_CutStoppedFlag`
CUT counter stopped flag

enumerator `kSFA_MeasurementStartedFlag`
Measurement Started flag

enumerator `kSFA_ReferenceCounterTimeOutFlag`
Reference counter time out flag

enumerator `kSFA_InterruptRequestFlag`
SFA interrupt request flag

enumerator `kSFA_FreqGreaterThanMaxInterruptFlag`
FREQ_GT_MAX interrupt flag

enumerator `kSFA_FreqLessThanMinInterruptFlag`
FREQ_LT_MIN interrupt flag

enumerator `kSFA_AllStatusFlags`

enum `_sfa_interrupts_enable`

List of SFA interrupt.

Values:

enumerator `kSFA_InterruptEnable`
SFA interrupt enable

enumerator `kSFA_FreqGreaterThanMaxInterruptEnable`
FREQ_GT_MAX interrupt enable

enumerator `kSFA_FreqLessThanMinInterruptEnable`
FREQ_LT_MIN interrupt enable

enum `_sfa_measurement_mode`

List of SFA measurement mode (Please check the mode configuration according to the manual).

Values:

enumerator `kSFA_FrequencyMeasurement0`
Frequency measurement performed with REF frequency > CUT frequency

enumerator kSFA_FrequencyMeasurement1
Frequency measurement performed with REF frequency < CUT frequency

enumerator kSFA_CUTPeriodMeasurement
CUT period measurement performed

enumerator kSFA_TriggerBasedMeasurement
Trigger based measurement performed

enum _sfa_cut_select

List of CUT which is connected to the CUT counter (Please refer to the manual for configuration).

Values:

enumerator kSFA_CUTSelect0

enumerator kSFA_CUTSelect1

enumerator kSFA_CUTSelect2

enumerator kSFA_CUTSelect3

enumerator kSFA_CUTSelect4

enumerator kSFA_CUTSelect5

enumerator kSFA_CUTSelect6

enumerator kSFA_CUTSelect7

enumerator kSFA_CUTSelect8

enumerator kSFA_CUTSelect9

enumerator kSFA_CUTSelect10

enumerator kSFA_CUTSelect11

enumerator kSFA_CUTSelect12

enumerator kSFA_CUTSelect13

enumerator kSFA_CUTSelect14

enumerator kSFA_CUTSelect15

enum _sfa_ref_select

List of REF which is connected to the REF counter (Please refer to the manual for configuration).

Values:

enumerator kSFA_REFSelect0

enumerator kSFA_REFSelect1

enumerator kSFA_REFSelect2

enum _sfa_trigger_start_select

List of Signal MUX for Trigger Based Measurement Start.

Values:

enumerator kSFA_TriggerStartSelect0

enumerator kSFA_TriggerStartSelect1

enum _sfa_trigger_end_select

List of Signal MUX for Trigger Based Measurement End.

Values:

enumerator kSFA_TriggerEndSelect0

enumerator kSFA_TriggerEndSelect1

enum _sfa_trigger_start_polarity

List of Trigger Start Polarity.

Values:

enumerator kSFA_TriggerStartPolarityRiseEdge

Rising edge will begin the measurement sequence

enumerator kSFA_TriggerStartPolarityFallEdge

Falling edge will begin the measurement sequence

enum _sfa_trigger_end_polarity

List of Trigger End Polarity.

Values:

enumerator kSFA_TriggerEndPolarityRiseEdge

Rising edge will end the measurement sequence

enumerator kSFA_TriggerEndPolarityFallEdge

Falling edge will end the measurement sequence

typedef void (*sfa_callback_t)(status_t status)

sfa measure completion callback function pointer type

This callback can be used in non blocking IRQHandler. Specify the callback you want in the call to SFA_InstallCallback().

Param base

SFA peripheral base address.

Param status

The runtime measurement status. kStatus_SFA_MeasurementCompleted: The measurement completes. kStatus_SFA_ReferenceCounterTimeout: Reference counter timeout happens. kStatus_SFA_CUTCounterTimeout: CUT counter timeout happens.

typedef enum _sfa_measurement_mode sfa_measurement_mode_t

List of SFA measurement mode(Please check the mode configuration according to the manual).

typedef enum _sfa_cut_select sfa_cut_select_t

List of CUT which is connected to the CUT counter (Please refer to the manual for configuration).

typedef enum _sfa_ref_select sfa_ref_select_t

List of REF which is connected to the REF counter (Please refer to the manual for configuration).

typedef enum _sfa_trigger_start_select sfa_trigger_start_select_t

List of Signal MUX for Trigger Based Measurement Start.

```
typedef enum _sfa_trigger_end_select sfa_trigger_end_select_t
```

List of Signal MUX for Trigger Based Measurement End.

```
typedef enum _sfa_trigger_start_polarity sfa_trigger_start_polarity_t
```

List of Trigger Start Polarity.

```
typedef enum _sfa_trigger_end_polarity sfa_trigger_end_polarity_t
```

List of Trigger End Polarity.

```
typedef struct _sfa_init_config sfa_config_t
```

Structure with setting to initialize the SFA module.

This structure holds configuration setting for the SFA peripheral. To initialize this structure to reasonable defaults, call the SFA_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

```
SFA_CUT_CLK_Enable(val)
```

```
struct _sfa_init_config
```

#include <fsl_sfa.h> Structure with setting to initialize the SFA module.

This structure holds configuration setting for the SFA peripheral. To initialize this structure to reasonable defaults, call the SFA_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

Public Members

```
sfa_measurement_mode_t mode
```

measurement mode

```
sfa_cut_select_t cutSelect
```

Select clock connected to the clock under test counter

```
sfa_ref_select_t refSelect
```

Select REF connected the bus clock

```
uint8_t prediv
```

Integer divide of the Input CUT signal

```
sfa_trigger_start_select_t trigStart
```

Select the signal will be used to end a trigger based measurement

```
sfa_trigger_start_polarity_t startPolarity
```

Select the polarity of the start trigger signal

```
sfa_trigger_end_select_t trigEnd
```

Select the signal will be used to commence a trigger based measurement

```
sfa_trigger_end_polarity_t endPolarity
```

Select the polarity of the end trigger signal

```
bool enableTrigMeasurement
```

false: The measurement will start by default with a dummy write to the CUT counter;
true : The measurement will start after receiving a dummy write to the REF_CNT followed by receiving the trigger edge

```
bool enableCUTPin
```

Control the connection of the clock under test to an external pin.

```
uint32_t refTarget
```

Reference counter target counts

uint32_t cutTarget
CUT counter target counts

2.66 SMSCM: Secure Miscellaneous System Control Module

FSL_MSCM_DRIVER_VERSION
SMSCM driver version 2.0.0.

enum _smscm_debug
SMSCM debug enable type.

Values:

enumerator kSMSCM_InvasiveDebug
enumerator kSMSCM_SecureInvasiveDebug
enumerator kSMSCM_NonInvasiveDebug
enumerator kSMSCM_SecureNonInvasiveDebug
enumerator kSMSCM_AltInvasiveDebug
enumerator kSMSCM_AltDebug

enum _smscm_mem
SMSCM On-Chip Memory Descriptor Register.

Values:

enumerator kSMSCM_Mem0
enumerator kSMSCM_Mem2
enumerator kSMSCM_Mem3
enumerator kSMSCM_Mem5

enum _smscm_ecc_ctrl
SMSCM ECC control type of On-Chip Memory.

Values:

enumerator kSMSCM_EccDisable
enumerator kSMSCM_EccEnableOnWrite
enumerator kSMSCM_EccEnableOnRead
enumerator kSMSCM_EccEnableOnWriteAndRead

typedef enum _smscm_mem smscm_mem_t
SMSCM On-Chip Memory Descriptor Register.

typedef enum _smscm_ecc_ctrl smscm_ecc_ctrl_t
SMSCM ECC control type of On-Chip Memory.

typedef struct smscm_ecc_fault_attr smscm_ecc_fault_attr_t
SMSCM attribute.

static inline void SMSCM_EnableDebug(SMSCM_Type *base, uint32_t debugToEnable)
Enable the Debug function. DebugToEnable could be bitwise OR of _smscm_debug.

Parameters

- base – SMSCM peripheral address.
- debugToEnable – debug enable type.

static inline void SMSCM_DisableDebug(SMSCM_Type *base, uint32_t debugToDisable)
Disables the Debug function. DebugToDisable could be bitwise OR of _smscm_debug.

Parameters

- base – SMSCM peripheral address.
- debugToDisable – debug disable type.

static inline void SMSCM_DebugLock(SMSCM_Type *base)
Lock the debug function.

Parameters

- base – SMSCM peripheral base address.

static inline uint32_t SMSCM_GetSecurityCount(SMSCM_Type *base)
Get value in Security Counter Register (SCTR).

Parameters

- base – SMSCM peripheral base address.

Returns

SMSCM SCTR value.

static inline void SMSCM_SetSecurityCount(SMSCM_Type *base, uint32_t val)
Set value in Security Counter Register (SCTR).

Parameters

- base – SMSCM peripheral base address.
- val – SCTR value to set.

static inline void SMSCM_IncreaseSecurityCount(SMSCM_Type *base, uint32_t val)
Write value to be plused in Security Counter Register (SCTR).

The entire contents of the write data word are added to the security counter, and next-state
 $SCTR = \text{current-state } SCTR + \text{DATA32}$.

Parameters

- base – SMSCM peripheral base address.
- val – SCTR value to plus.

static inline void SMSCM_DecreaseSecurityCount(SMSCM_Type *base, uint32_t val)
Write value to be minused in Security Counter Register (SCTR).

The entire contents of the write data word are added to the security counter, and next-state
 $SCTR = \text{current-state } SCTR - \text{DATA32}$.

Parameters

- base – SMSCM peripheral base address.
- val – SCTR value to be minused.

static inline void SMSCM_IncreaseSecurityCountBy1(SMSCM_Type *base)

Increase security counter register by 1.

Parameters

- base – SMSCM peripheral base address.

static inline void SMSCM_DecreaseSecurityCountBy1(SMSCM_Type *base)

Decrease security counter register by 1.

Parameters

- base – SMSCM peripheral base address.

static inline void SMSCM_LockMemControlReg(SMSCM_Type *base, *smscm_mem_t* mem)

Lock the on-chip memory descriptor. This register bit provides a mechanism to “lock” the configuration state defined by OCMDRn[11:0]. Once asserted, attempted writes to the OCM-DRn[11:0] register are ignored until the next reset clears the flag.

Parameters

- base – SMSCM peripheral address.
- mem – Select OCMDRn to enable read-only mode.

static inline void SMSCM_EnableFlashCache(SMSCM_Type *base, bool enable)

Enable or disable the on-chip memory flash cache.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_EnableFlashInstructionCache(SMSCM_Type *base, bool enable)

Enable flash instruction cache.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_EnableFlashDataCache(SMSCM_Type *base, bool enable)

Enable flash data cache.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_ClearFlashCache(SMSCM_Type *base)

Clear the on-chip memory flash cache.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_LockFlashIFR1(SMSCM_Type *base)

Lock IFR1 by flash controller.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_EnableFlashSpeculate(SMSCM_Type *base, bool enable)

SMSCM Flash Speculate enable.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_EnableDataPrefetch(SMSCM_Type *base, bool enable)
 SMSCM Data Prefetch enable.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_EnableFlashDataNonCorrectableBusError(SMSCM_Type *base, bool enable)

Disable non-correctable bus errors on flash data fetches.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_EnableFlashInstructionNonCorrectableBusError(SMSCM_Type *base, bool enable)

Disable non-correctable bus errors on flash instruction fetches.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_SetMemEccControl(SMSCM_Type *base, *smscm_mem_t* mem, *smscm_ecc_ctrl_t* eccCtrl)

Select ecc control type in OCMDRn.

Parameters

- base – SMSCM peripheral address.
- mem – Select OCMDRn.
- eccCtrl – Select ecc control type.

static inline void SMSCM_EnableEccReport(SMSCM_Type *base)
 Enable RAM ECC 1 bit and non-correctable reporting.

Parameters

- base – SMSCM peripheral base address.

static inline bool SMSCM_GetEccValid(SMSCM_Type *base)
 Get the ECC location valid states.

Parameters

- base – SMSCM peripheral base address.

Returns

State of ECC Error Location field.

static inline uint8_t SMSCM_GetEccLocation(SMSCM_Type *base)
 Get the ECC location.

Parameters

- base – SMSCM peripheral base address.

Returns

ECC fault location.

void SMSCM_ClearEccError(SMSCM_Type *base, uint8_t errLocation)
 Clear each 1-bit correctable or non-correctable error.

Parameters

- base – SMSCM peripheral address.
- errLocation – ECC Error Location

```
static inline uint32_t SMSCM_GetEccAddress(SMSCM_Type *base)
```

Get the ECC fault address.

Parameters

- base – SMSCM peripheral base address.

Returns

ECC fault address.

```
void SMSCM_GetEccAttribute(SMSCM_Type *base, smscm_ecc_fault_attr_t *eccAttribute)
```

Get ECC attribute.

Parameters

- base – SMSCM peripheral address.
- eccAttribute – Ecc attribute.

```
static inline uint32_t SMSCM_GetEccFaultDataHigh(SMSCM_Type *base)
```

Get ECC Fault Data High.

This read-only field specifies the upper 32-bit read data word (data[63:32]) from the last captured ECCevent. For ECC events that occur in 32-bit RAMs, this 32-bit field will return 32'h0.

Parameters

- base – SMSCM peripheral base address.

Returns

The higher 32-bit read data word.

```
static inline uint32_t SMSCM_GetEccFaultDataLow(SMSCM_Type *base)
```

Get ECC Fault Data Low.

Parameters

- base – SMSCM peripheral base address.

Returns

The lower 32-bit read data word.

```
struct smscm_ecc_fault_attr
```

```
#include <fsl_smscm.h> SMSCM attribute.
```

2.67 SPC: System Power Control driver

SPC status enumeration.

Values:

enumerator kStatus_SPC_Busy

The SPC instance is busy executing any type of power mode transition.

enumerator kStatus_SPC_DCDCLowDriveStrengthIgnore

DCDC Low drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_DCDCPulseRefreshModeIgnore

DCDC Pulse Refresh Mode drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_SYSLDOOverDriveVoltageFail

SYS LDO regulate to Over drive voltage failed for SYS LDO HVD must be disabled.

enumerator kStatus_SPC_SYSLDOLowDriveStrengthIgnore
SYS LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_CORELDOLowDriveStrengthIgnore
CORE LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_CORELDIVoltageWrong
Core LDO voltage is wrong.

enumerator kStatus_SPC_CORELDIVoltageSetFail
Core LDO voltage set fail.

enumerator kStatus_SPC_BandgapModeWrong
Selected Bandgap Mode wrong.

enum _spc_voltage_detect_flags
Voltage Detect Status Flags.

Values:

enumerator kSPC_IOVDDHighVoltageDetectFlag
IO VDD High-Voltage detect flag.

enumerator kSPC_CoreVDDHighVoltageDetectFlag
Core VDD High-Voltage detect flag.

enumerator kSPC_IOVDDLowVoltageDetectFlag
IO VDD Low-Voltage detect flag.

enumerator kSPC_CoreVDDLowVoltageDetectFlag
Core VDD Low-Voltage detect flag.

enum _spc_power_domains
SPC power domain isolation status.

Values:

enumerator kSPC_MAINPowerDomainRetain
Peripherals and IO pads retain in MAIN Power Domain.

enumerator kSPC_WAKEPowerDomainRetain
Peripherals and IO pads retain in WAKE Power Domain.

enumerator kSPC_2P4GPowerDoaminRetain
Peripherals and IO pads retion in 2.4G Power Domain.

enum _spc_power_domain_id
The enumeration of spc power domain, the connected power domain is chip specific, please refer to chip's RM for details.

Values:

enumerator kSPC_PowerDomain0
Power domain0, the connected power domain is chip specific.

enumerator kSPC_PowerDomain1
Power domain1, the connected power domain is chip specific.

enumerator kSPC_PowerDomain2
Power domain2, the connected power domain is chip specific.

enum _spc_power_domain_low_power_mode
The enumeration of Power domain's low power mode.

Values:

enumerator kSPC_SleepWithSYSClockRunning
Power domain request SLEEP mode with SYS clock running.

enumerator kSPC_SleepWithSysClockOff
Power domain request SLEEP mode with SYS clock off.

enumerator kSPC_DeepSleepSysClockOff
Power domain request DEEP SLEEP mode with SYS clock off.

enumerator kSPC_PowerDownWithSysClockOff
Power domain request POWER DOWN mode with SYS clock off.

enumerator kSPC_DeepPowerDownWithSysClockOff
Power domain request DEEP POWER DOWN mode with SYS clock off.

enum _spc_lowPower_request_pin_polarity
SPC low power request output pin polarity.

Values:

enumerator kSPC_HighTruePolarity
Control the High Polarity of the Low Power Request Pin.

enumerator kSPC_LowTruePolarity
Control the Low Polarity of the Low Power Request Pin.

enum _spc_lowPower_request_output_override
SPC low power request output override.

Values:

enumerator kSPC_LowPowerRequestNotForced
Not Forced.

enumerator kSPC_LowPowerRequestReserved
Reserved.

enumerator kSPC_LowPowerRequestForcedLow
Forced Low (Ignore LowPower request output polarity setting.)

enumerator kSPC_LowPowerRequestForcedHigh
Forced High (Ignore LowPower request output polarity setting.)

enum _spc_bandgap_mode
SPC Bandgap mode enumeration in Active mode or Low Power mode.

Values:

enumerator kSPC_BandgapDisabled
Bandgap disabled.

enumerator kSPC_BandgapEnabledBufferDisabled
Bandgap enabled with Buffer disabled.

enumerator kSPC_BandgapEnabledBufferEnabled
Bandgap enabled with Buffer enabled.

enumerator kSPC_BandgapReserved
Reserved.

enum _spc_dcdc_voltage_level
DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

Values:

enumerator kSPC_DCDC_SafeModeVoltage
DCDC VDD Regulator regulate to Safe-Mode Voltage.

enumerator kSPC_DCDC_NormalVoltage
DCDC VDD Regulator regulate to Normal Voltage.

enumerator kSPC_DCDC_MidVoltage
DCDC VDD Regulator regulate to Mid Voltage.

enumerator kSPC_DCDC_LowUnderVoltage
DCDC VDD Regulator regulate to Low Under Voltage.

enum _spc_dc_dc_drive_strength
DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

Values:

enumerator kSPC_DCDC_PulseRefreshMode
DCDC VDD Regulator Drive Strength set to Pulse Refresh Mode. This enum member is only useful for Low Power Mode config.

enumerator kSPC_DCDC_LowDriveStrength
DCDC VDD regulator Drive Strength set to low.

enumerator kSPC_DCDC_NormalDriveStrength
DCDC VDD regulator Drive Strength set to Normal.

enumerator kSPC_DCDC_Reserved
Reserved.

enum _spc_core_ldo_voltage_level
Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

Values:

enumerator kSPC_CoreLDO_NormalVoltage
Core LDO VDD regulator regulate to Normal Voltage.

enumerator kSPC_CoreLDO_MidDriveVoltage
Core LDO VDD regulator regulate to Mid Drive Voltage.

enumerator kSPC_CoreLDO_UnderDriveVoltage
Core LDO VDD regulator regulate to Under Drive Voltage.

enumerator kSPC_CoreLDO_SafeModeVoltage
Core LDO VDD regulator regulate to Safe-Mode Voltage.

enum _spc_core_ldo_drive_strength
CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

Values:

enumerator kSPC_CoreLDO_LowDriveStrength
Core LDO VDD regulator Drive Strength set to low.

enumerator kSPC_CoreLDO_NormalDriveStrength
Core LDO VDD regulator Drive Strength set to Normal.

enum _spc_low_voltage_level_select
System/IO VDD Low-Voltage Level Select.

Values:

enumerator kSPC_LowVoltageNormalLevel

Trip point set to Normal level.

enumerator kSPC_LowVoltageSafeLevel

Trip point set to Safe level.

enum _spc_sram_operat_voltage

SRAM operate voltage enumeration.

Values:

enumerator kSPC_SRAM_OperatVoltage1P0V

SRAM operate voltage set to 1.0V.

enumerator kSPC_SRAM_OperatVoltage1P1V

SRAM operate voltage set to 1.1V.

enum _spc_hp_request_override_option

The enumeration of high power request override option.

Values:

enumerator kSPC_HpRequestOverrideDisable

Disable high power request override feature.

enumerator kSPC_HpRequestOverride0

Enable high power request override feature and force value as 0.

enumerator kSPC_HpRequestOverride1

Enable high power request override feature and force value as 1.

enum _spc_vdd_core_glitch_ripple_counter_select

Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

Values:

enumerator kSPC_selectBit0Of4bitRippleCounter

Select bit-0 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit1Of4bitRippleCounter

Select bit-1 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit2Of4bitRippleCounter

Select bit-2 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit3Of4bitRippleCounter

Select bit-3 of 4-bit Ripple Counter to detect glitch on VDD Core.

typedef enum _spc_power_domain_id spc_power_domain_id_t

The enumeration of spc power domain, the connected power domain is chip specific, please refer to chip's RM for details.

typedef enum _spc_power_domain_low_power_mode spc_power_domain_low_power_mode_t

The enumeration of Power domain's low power mode.

typedef enum _spc_lowPower_request_pin_polarity spc_lowpower_request_pin_polarity_t

SPC low power request output pin polarity.

typedef enum _spc_lowPower_request_output_override spc_lowpower_request_output_override_t

SPC low power request output override.

typedef enum _spc_bandgap_mode spc_bandgap_mode_t

SPC Bandgap mode enumeration in Active mode or Low Power mode.

```
typedef enum _spc_dcdc_voltage_level spc_dcdc_voltage_level_t
    DCDC regulator voltage level enumeration in Active mode or Low Power Mode.
typedef enum _spc_dcdc_drive_strength spc_dcdc_drive_strength_t
    DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.
typedef enum _spc_core_ldo_voltage_level spc_core_ldo_voltage_level_t
    Core LDO regulator voltage level enumeration in Active mode or Low Power mode.
typedef enum _spc_core_ldo_drive_strength spc_core_ldo_drive_strength_t
    CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.
typedef enum _spc_low_voltage_level_select spc_low_voltage_level_select_t
    System/IO VDD Low-Voltage Level Select.
typedef enum _spc_sram_operat_voltage spc_sram_operat_voltage_t
    SRAM operate voltage enumeration.
typedef struct _spc_lowpower_request_config spc_lowpower_request_config_t
    Low Power Request output pin configuration.
typedef struct _spc_intergrated_power_switch_config spc_intergrated_power_switch_config_t
    Integrated power switch configuration.
```

Note: Legacy structure, will be removed.

```
typedef struct _spc_active_mode_core_ldo_option spc_active_mode_core_ldo_option_t
    Core LDO regulator options in Active mode.
typedef struct _spc_active_mode_dcdc_option spc_active_mode_dcdc_option_t
    DCDC regulator options in Active mode.
typedef struct _spc_lowpower_mode_core_ldo_option spc_lowpower_mode_core_ldo_option_t
    Core LDO regulator options in Low Power mode.
typedef struct _spc_lowpower_mode_dcdc_option spc_lowpower_mode_dcdc_option_t
    DCDC regulator options in Low Power mode.
typedef struct _spc_voltage_detect_option spc_voltage_detect_option_t
    CORE/SYS/IO VDD Voltage Detect options.
typedef struct _spc_core_voltage_detect_config spc_core_voltage_detect_config_t
    Core Voltage Detect configuration.
typedef struct _spc_io_voltage_detect_config spc_io_voltage_detect_config_t
    IO Voltage Detect Configuration.
typedef struct _spc_active_mode_regulators_config spc_active_mode_regulators_config_t
    Active mode configuration.
typedef struct _spc_lowpower_mode_regulators_config spc_lowpower_mode_regulators_config_t
    Low Power Mode configuration.
typedef enum _spc_hp_request_override_option spc_hp_override_request_option_t
    The enumeration of high power request override option.
typedef spc_active_mode_dcdc_option_t spc_hp_mode_dcdc_option_t
typedef spc_active_mode_core_ldo_option_t spc_hp_mode_core_ldo_option_t
typedef spc_active_mode_regulators_config_t spc_hp_mode_regulators_config_t
```

```
typedef enum _spc_vdd_core_glitch_ripple_counter_select
spc_vdd_core_glitch_ripple_counter_select_t
```

Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

```
typedef struct _spc_vdd_core_glitch_detector_config spc_vdd_core_glitch_detector_config_t
The configuration of VDD Core glitch detector.
```

```
SPC_BUSY_TIMEOUT
```

Max loops to wait for SPC to stop being busy.

The BUSY bitfield will be set when the SPC performs any kind of power mode transition in active mode or any chip low power mode. You need to wait until this flag is cleared before changing the power mode configuration registers. This parameter defines how many loops to check completion before return timeout. If defined as 0, the driver will wait until completion.

```
SPC_SRAM_ACK_TIMEOUT
```

Max loops to wait for SPC to stop being busy.

When changing SRAM voltage, need to wait for SRAM voltage update request acknowledgment. This parameter defines how many loops to check completion before return timeout. If defined as 0, the driver will wait until completion.

```
SPC_DCDC_ACK_TIMEOUT
```

Max loops to wait for DCDC burst completed.

When the DCDC burst is requested, it is necessary to wait for the DCDC burst to complete. This parameter defines how many loops to check completion before return timeout. If defined as 0, the driver will wait until it completes.

```
SPC_HP_CNFG_CTRL_OVERRIDE_OPT_MASK
```

```
SPC_HP_CNFG_CTRL_OVERRIDE_OPT_SHIFT
```

```
SPC_HP_CNFG_CTRL_OVERRIDE_OPT(x)
```

```
uint8_t SPC_GetPeriphIOIsolationStatus(SPC_Type *base)
```

Gets Isolation status for each power domains.

This function gets the status which indicates whether certain peripheral and the IO pads are in a latched state as a result of having been in POWERDOWN mode.

Parameters

- base – SPC peripheral base address.

Returns

Current isolation status for each power domains. See `_spc_power_domains` for details.

```
static inline void SPC_ClearPeriphIOIsolationFlag(SPC_Type *base)
```

Clears peripherals and I/O pads isolation flags for each power domains.

This function clears peripherals and I/O pads isolation flags for each power domains. After recovering from the POWERDOWN mode, user must invoke this function to release the I/O pads and certain peripherals to their normal run mode state. Before invoking this function, user must restore chip configuration in particular pin configuration for enabled WUU wakeup pins.

Parameters

- base – SPC peripheral base address.

```
static inline bool SPC_GetBusyStatusFlag(SPC_Type *base)
```

Gets SPC busy status flag.

This function gets SPC busy status flag. When SPC executing any type of power mode transition in ACTIVE mode or any of the SOC low power mode, the SPC busy status flag is set and this function returns true. When changing CORE LDO voltage level and DCDC voltage level in ACTIVE mode, the SPC busy status flag is set and this function return true.

Parameters

- base – SPC peripheral base address.

Returns

Ack busy flag. true - SPC is busy. false - SPC is not busy.

```
static inline bool SPC_CheckLowPowerRequest(SPC_Type *base)
```

Checks system low power request.

Note: Only when all power domains request low power mode entry, the result of this function is true. That means when all power domains request low power mode entry, the SPC regulators will be controlled by LP_CFG register.

Parameters

- base – SPC peripheral base address.

Returns

The system low power request check result.

- **true** All power domains have requested low power mode and SPC has entered a low power state and power mode configuration are based on the LP_CFG configuration register.
- **false** SPC in active mode and ACTIVE_CFG register control system power supply.

```
static inline void SPC_ClearLowPowerRequest(SPC_Type *base)
```

Clears system low power request, set SPC in active mode.

Parameters

- base – SPC peripheral base address.

```
static inline bool SPC_CheckPowerSwitchState(SPC_Type *base)
```

Checks power switch state.

Parameters

- base – SPC peripheral base address.

Returns

The state(ON/OFF) of power switch.

- **true** Indicates the power switch is ON.
- **false** Indicates the power switch is OFF.

```
spc_power_domain_low_power_mode_t SPC_GetPowerDomainLowPowerMode(SPC_Type *base,  
                                                                    spc_power_domain_id_t  
                                                                    powerDomainId)
```

Gets selected power domain's requested low power mode.

Parameters

- base – SPC peripheral base address.

- `powerDomainId` – Power Domain Id, please refer to `spc_power_domain_id_t`.

Returns

The selected power domain's requested low power mode, please refer to `spc_power_domain_low_power_mode_t`.

```
static inline bool SPC__CheckPowerDomainLowPowerRequest(SPC_Type *base,  
                                                       spc_power_domain_id_t  
                                                       powerDomainId)
```

Checks power domain's low power request.

Parameters

- `base` – SPC peripheral base address.
- `powerDomainId` – Power Domain Id, please refer to `spc_power_domain_id_t`.

Returns

The result of power domain's low power request.

- **true** The selected power domain requests low power mode entry.
- **false** The selected power domain does not request low power mode entry.

```
static inline void SPC__ClearPowerDomainLowPowerRequestFlag(SPC_Type *base,  
                                                           spc_power_domain_id_t  
                                                           powerDomainId)
```

Clears selected power domain's low power request flag.

Parameters

- `base` – SPC peripheral base address.
- `powerDomainId` – Power Domain Id, please refer to `spc_power_domain_id_t`.

```
void SPC__SetLowPowerRequestConfig(SPC_Type *base, const spc_lowpower_request_config_t  
                                  *config)
```

Configs Low power request output pin.

This function config the low power request output pin

Parameters

- `base` – SPC peripheral base address.
- `config` – Pointer the `spc_lowpower_request_config_t` structure.

```
static inline void SPC__SoftwareGatePowerSwitch(SPC_Type *base, bool gate)
```

Gates/Un-gates power switch in software mode.

Parameters

- `base` – SPC peripheral base address.
- `gate` – Used to gate/ungate power switch
 - **true** The power switch will be gated.
 - **false** The power switch will be un-gated.

```
static inline void SPC__PowerModeControlPowerSwitch(SPC_Type *base)
```

Gates power switch at low power modes entry, and un-gates power switch at low power mode wakeup.

Parameters

- `base` – SPC peripheral base address.

```
static inline void SPC_SetWakeUpValue(SPC_Type *base, uint32_t data)
```

Set the address of the function/image to be executed if chip wake from power down or deep power down mode.

Note: Data written by this function is used by BootROM to quickly restore ARM Core context, or to switch execution to a defined address in Flash/SRAM on WakeUp.

Note: The first word must be SP, and the second word must be PC.

Note: Please remember to calculate the CRC value of the first 48 bytes of image/function and save the result to REGFILE1->REG[0]. The BootROM will check this CRC value, if authenticated successfully then the image/function will be executed.

Parameters

- `base` – SPC peripheral base address.
- `data` – The address of the function/image to be executed if wakeup from low power mode.

```
static inline uint32_t SPC_GetWakeUpValue(SPC_Type *base)
```

Gets back the WakeUp value.

Parameters

- `base` – SPC peripheral base address.

Returns

The WakeUp value.

```
static inline bool SPC_CheckHPCfgSelected(SPC_Type *base)
```

Check if HP_CFG selected as active configuration register.

Parameters

- `base` – SPC peripheral base address.

Return values

- `false` – ACTIVE_CFG selected as the active configuration register.
- `true` – HP_CFG selected as the active configuration register.

```
static inline void SPC_EnableHighPowerRequest(SPC_Type *base, bool enable)
```

Enable/disable high power request feature.

Parameters

- `base` – SPC peripheral base address.
- `enable` – Used to specify enable/disable the high power request feature:
 - **true** Enable high power request;
 - **false** Disable high power request.

```
static inline void SPC_OverrideHighPowerRequest(SPC_Type *base,  
                                                spc_hp_override_request_option_t opt)
```

Override high power request manually.

Parameters

- `base` – SPC peripheral base address.

- `opt` – Specify the option of high power override request, please refer to `spc_hp_override_request_option_t`.

```
static inline spc_core_ldo_voltage_level_t SPC_GetHighPowerModeCoreLDOVDDVoltageLevel(SPC_Type *base)
```

Get voltage level of CORE LDO in high power mode.

Parameters

- `base` – SPC peripheral base address.

Returns

The voltage level of CORE LDO in high power mode, please refer to `spc_core_ldo_voltage_level_t`.

```
static inline spc_bandgap_mode_t SPC_GetHighPowerModeBandgapMode(SPC_Type *base)
```

Get bandgap mode in high power mode.

Parameters

- `base` – SPC peripheral base address.

Returns

The bandgap mode in high power mode, please refer to `spc_bandgap_mode_t`.

```
static inline uint32_t SPC_GetHighPowerModeVoltageDetectStatus(SPC_Type *base)
```

Get enabled state of all voltage detectors.

Parameters

- `base` – SPC peripheral base address.

Returns

All enabled status of all voltage detectors, 1b1 means the corresponding voltage detector is enabled.

```
status_t SPC_SetHighPowerModeBandgapModeConfig(SPC_Type *base, spc_bandgap_mode_t mode)
```

Set bandgap mode in high power mode.

Parameters

- `base` – SPC peripheral base address.
- `mode` – Specify the bandgap mode in high power mode.

Return values

- `kStatus_SPC_BandgapModeWrong` – The Bandgap can not be disabled in high power mode.
- `kStatus_Success` – Config Bandgap mode in high power mode successful.

```
static inline void SPC_DisableHighPowerModeVddCoreGlitchDetect(SPC_Type *base, bool disable)
```

Disable/enable VDD Core Glitch detect feature in high power mode.

Parameters

- `base` – SPC peripheral base address.
- `disable` – Used to disable/enable VDD Core Glitch detect feature:
 - **true** Disable VDD Core Glitch detect feature;
 - **false** Enable VDD Core Glitch detect feature.

```
status_t SPC_SetHighPowerModeCoreLDORegulatorConfig(SPC_Type *base, const spc_hp_mode_core_ldo_option_t *option)
```

Configure CORE LDO regulator in high power mode.

Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to the CORE LDO regulator configuration, please refer to `spc_hp_mode_core_ldo_option_t`.

Return values

- `kStatus_Success` – Config Core LDO regulator in High power mode successful.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – If any voltage detect enabled, `core_ldo`'s drive strength can not set to low.
- `kStatus_SPC_CORELDIVoltageWrong` – The selected voltage level in high power mode is not allowed.
- `kStatus_Timeout` – Timeout occurs while waiting completion.

```
status_t SPC_SetHighPowerModeDCDCRegulatorConfig(SPC_Type *base, const
                                                spc_hp_mode_dcdc_option_t *option)
```

Configure DCDC regulator in high power mode.

Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to the DCDC regulator configuration, please refer to `spc_hp_mode_dcdc_option_t`.

Return values

- `kStatus_Success` – Config DCDC regulator in Active power mode successful.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_DCDCLowDriveStrengthIgnore` – Set driver strength to Low will be ignored.
- `kStatus_Timeout` – Timeout occurs while waiting completion.

```
status_t SPC_SetHighPowerModeRegulatorsConfig(SPC_Type *base, const
                                                spc_hp_mode_regulators_config_t *config)
```

Set configuration of regulators in high power mode.

Parameters

- `base` – SPC peripheral base address.
- `config` – Pointer to the regulator configuration, please refer to `spc_hp_mode_regulators_config_t`.

Return values

- `kStatus_Success` – Config regulators in High power mode successful.
- `kStatus_SPC_BandgapModeWrong` – The bandgap mode setting in high power mode is wrong.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.

- kStatus_SPC_CORELDOVoltageWrong – The selected voltage level in high mode is not allowed.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Fail to regulator to Over Drive Voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low will be ignored.

```
static inline void SPC_EnableHighPowerModeCoreLowVoltageDetect(SPC_Type *base, bool
                                                                enable)
```

Enable/disable low voltage detect for VDD_CORE in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable low voltage detect feature for VDD_CORE in high power mode:
 - **true** Enable low voltage detect feature for VDD_CORE in high power mode;
 - **false** Disable low voltage detect feature for VDD_CORE in high power mode.

```
static inline void SPC_EnableHighPowerModeCoreHighVoltageDetect(SPC_Type *base, bool
                                                                enable)
```

Enable/disable high voltage detect for VDD_CORE in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable high voltage detect feature for VDD_CORE in high power mode:
 - **true** Enable low voltage detect feature for VDD_CORE in high power mode;
 - **false** Disable low voltage detect feature for VDD_CORE in high power mode.

```
static inline void SPC_EnableHighPowerModeIOLowVoltageDetect(SPC_Type *base, bool enable)
```

Enable/disable low voltage detect for VDD_IO_ABC in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable low voltage detect feature for VDD_IO_ABC in high power mode:
 - **true** Enable low voltage detect feature for VDD_IO_ABC in high power mode;
 - **false** Disable low voltage detect feature for VDD_IO_ABC in high power mode.

```
static inline void SPC_EnableHighPowerModeIOHighVoltageDetect(SPC_Type *base, bool enable)
```

Enable/disable high voltage detect for VDD_IO_ABC in high power mode.

Parameters

- base – SPC peripheral base address.

- `enable` – Used to enable/disable high voltage detect feature for VDD_IO_ABC in high power mode:
 - **true** Enable high voltage detect feature for VDD_IO_ABC in high power mode;
 - **false** Disable high voltage detect feature for VDD_IO_ABC in high power mode.

```
static inline spc_core_ldo_voltage_level_t SPC_GetActiveModeCoreLDOVDDVoltageLevel(SPC_Type *base)
```

Gets CORE LDO VDD Regulator Voltage level.

This function returns the voltage level of CORE LDO Regulator in Active mode.

Parameters

- `base` – SPC peripheral base address.

Returns

Voltage level of CORE LDO in type of `spc_core_ldo_voltage_level_t` enumeration.

```
static inline spc_bandgap_mode_t SPC_GetActiveModeBandgapMode(SPC_Type *base)
```

Gets the Bandgap mode in Active mode.

Parameters

- `base` – SPC peripheral base address.

Returns

Bandgap mode in the type of `spc_bandgap_mode_t` enumeration.

```
static inline uint32_t SPC_GetActiveModeVoltageDetectStatus(SPC_Type *base)
```

Gets all voltage detectors status in Active mode.

Parameters

- `base` – SPC peripheral base address.

Returns

All voltage detectors status in Active mode.

```
void SPC_SetActiveModeIntegratedPowerSwitchConfig(SPC_Type *base, const spc_intergrated_power_switch_config_t *config)
```

Configs Integrated power switch in active mode.

Note: Legacy API and will be removed.

Parameters

- `base` – SPC peripheral base address.
- `config` – Pointer to `spc_intergrated_power_switch_config_t` pointer.

```
status_t SPC_SetActiveModeBandgapModeConfig(SPC_Type *base, spc_bandgap_mode_t mode)
```

Configs Bandgap mode in Active mode.

Note: In active mode, because CORELDO_VDD_DS is reserved and set to Normal, so it is impossible to disable Bandgap in active mode

Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

Return values

- kStatus_SPC_BandgapModeWrong – The Bandgap can not be disabled in active mode.
- kStatus_Success – Config Bandgap mode in Active power mode successful.

```
static inline void SPC_SetActiveModeVoltageTrimDelay(SPC_Type *base, uint16_t delay)
```

Sets the delay when the regulators change voltage level in Active mode.

Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles.

```
status_t SPC_SetActiveModeRegulatorsConfig(SPC_Type *base, const  
                                           spc_active_mode_regulators_config_t *config)
```

Configs regulators in Active mode.

This function provides the method to config all on-chip regulators in active mode.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_active_mode_regulators_config_t structure.

Return values

- kStatus_Success – Config regulators in Active power mode successful.
- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in Active mode is wrong.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOVoltageWrong – The selected voltage level in active mode is not allowed.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Fail to regulator to Over Drive Voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

```
static inline void SPC_DisableActiveModeVddCoreGlitchDetect(SPC_Type *base, bool disable)
```

Disable/Enable VDD Core Glitch Detect in Active mode.

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
 - **true** Disable VDD Core Low Voltage detect;

- **false** Enable VDD Core Low Voltage detect.

```
void SPC_SetLowPowerModeIntegratedPowerSwitchConfig(SPC_Type *base, const
                                                    spc_intergrated_power_switch_config_t
                                                    *config)
```

Configs Integrated power switch in Low Power mode.

Note: Legacy API, will be removed.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_intergrated_power_switch_config_t` pointer.

```
static inline void SPC_EnableLowPowerModeVDDCWellBias(SPC_Type *base, bool enable)
```

Enables/Disables VDDC Well Bias in low power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable the VDDC Well Bias. true - Enable Vddc Well Bias. false - Disable Vddc Well Bias.

```
static inline spc_core_ldo_drive_strength_t SPC_GetLowPowerCoreLDOVDDDriveStrength(SPC_Type
                                                                                    *base)
```

Gets CORE LDO VDD Drive Strength for Low Power modes.

Parameters

- base – SPC peripheral base address.

Returns

The CORE LDO's VDD Drive Strength.

```
static inline spc_core_ldo_voltage_level_t SPC_GetLowPowerCoreLDOVDDVoltageLevel(SPC_Type
                                                                                    *base)
```

Gets the CORE LDO VDD Regulator Voltage Level for Low Power modes.

Parameters

- base – SPC peripheral base address.

Returns

The CORE LDO VDD Regulator's voltage level.

```
static inline spc_bandgap_mode_t SPC_GetLowPowerModeBandgapMode(SPC_Type *base)
```

Gets the Bandgap mode in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns

Bandgap mode in the type of `spc_bandgap_mode_t` enumeration.

```
static inline uint32_t SPC_GetLowPowerModeVoltageDetectStatus(SPC_Type *base)
```

Gets the status of all voltage detectors in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns

The status of all voltage detectors in low power mode.

```
static inline void SPC_EnableLowPowerModeLowPowerIREF(SPC_Type *base, bool enable)
```

Enables/Disables Low Power IREF in low power modes.

This function enables/disables Low Power IREF. Low Power IREF can only get disabled in Deep power down mode. In other low power modes, the Low Power IREF is always enabled.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Low Power IREF. true - Enable Low Power IREF for Low Power modes. false - Disable Low Power IREF for Deep Power Down mode.

```
status_t SPC_SetLowPowerModeBandgapmodeConfig(SPC_Type *base, spc_bandgap_mode_t mode)
```

Configs Bandgap mode in Low Power mode.

This function configs Bandgap mode in Low Power mode. IF user want to disable Bandgap while keeping any of the Regulator in Normal Driver Strength or if any of the High voltage detectors/Low voltage detectors are kept enabled, the Bandgap mode will be set as Bandgap Enabled with Buffer Disabled.

Note: This API shall be invoked following set HVDs/LVDs and regulators' driver strength.

Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

Return values

- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in Low Power mode is wrong.
- kStatus_Success – Config Bandgap mode in Low Power power mode successful.

```
static inline void SPC_EnableLowPowerModeCoreVDDInternalVoltageScaling(SPC_Type *base, bool enable)
```

Enables/Disables CORE VDD IVS(Internal Voltage Scaling) in low power modes.

This function gates CORE VDD IVS. When enabled, the IVS regulator will scale the external input CORE VDD to a lower voltage level to reduce internal leakage. IVS is invalid in Sleep or Deep power down mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IVS. true - enable CORE VDD IVS in Deep Sleep mode or Power Down mode. false - disable CORE VDD IVS in Deep Sleep mode or Power Down mode.

```
static inline void SPC_SetLowPowerWakeUpDelay(SPC_Type *base, uint16_t delay)
```

Sets the delay when exit the low power modes.

Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles that the SPC waits on exit from low power modes.

```
status_t SPC_SetLowPowerModeRegulatorsConfig(SPC_Type *base, const
                                             spc_lowpower_mode_regulators_config_t
                                             *config)
```

Configs regulators in Low Power mode.

This function provides the method to config all on-chip regulators in Low Power mode.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_lowpower_mode_regulators_config_t` structure.

Return values

- `kStatus_Success` – Config regulators in Low power mode successful.
- `kStatus_SPC_BandgapModeWrong` – The bandgap mode setting in Low Power mode is wrong.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOVoltageWrong` – The selected voltage level is wrong.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `#kStatus_SPC_CORELDOVoltageSetFail` – Fail to change Core LDO voltage level.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `kStatus_SPC_DCDCPulseRefreshModeIgnore` – Set driver strength to Pulse Refresh mode will be ignored.
- `kStatus_SPC_DCDCLowDriveStrengthIgnore` – Set driver strength to Low Drive Strength will be ignored.

```
static inline void SPC_DisableLowPowerModeVddCoreGlitchDetect(SPC_Type *base, bool disable)
Disable/Enable VDD Core Glitch Detect in low power mode.
```

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
 - **true** Disable VDD Core Low Voltage detect;
 - **false** Enable VDD Core Low Voltage detect.

```
static inline uint8_t SPC_GetVoltageDetectStatusFlag(SPC_Type *base)
Get Voltage Detect Status Flags.
```

Parameters

- base – SPC peripheral base address.

Returns

Voltage Detect Status Flags. See `_spc_voltage_detect_flags` for details.

```
static inline void SPC_ClearVoltageDetectStatusFlag(SPC_Type *base, uint8_t mask)
```

Clear Voltage Detect Status Flags.

Parameters

- base – SPC peripheral base address.
- mask – The mask of the voltage detect status flags. See `_spc_voltage_detect_flags` for details.

```
void SPC_SetCoreVoltageDetectConfig(SPC_Type *base, const spc_core_voltage_detect_config_t *config)
```

Configs CORE voltage detect options.

This function config CORE voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_core_voltage_detect_config_t` structure.

```
static inline void SPC_LockCoreVoltageDetectResetSetting(SPC_Type *base)
```

Locks Core voltage detect reset setting.

This function locks core voltage detect reset setting. After invoking this function any configuration of Core voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

```
static inline void SPC_UnlockCoreVoltageDetectResetSetting(SPC_Type *base)
```

Unlocks Core voltage detect reset setting.

This function unlocks core voltage detect reset setting. If locks the Core voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

```
status_t SPC_EnableActiveModeCoreHighVoltageDetect(SPC_Type *base, bool enable)
```

Enables/Disables the Core High Voltage Detector in Active mode.

Note: If the CORE_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in active mode. false - Disable Core High voltage detector in active mode.

Return values

`kStatus_Success` – Enable/Disable Core High Voltage Detect successfully.

status_t SPC_EnableActiveModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Active mode.

Note: If the CORE_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core LVD. true - Enable Core Low voltage detector in active mode. false - Disable Core Low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core High Voltage Detector in Low Power mode.

This function enables/disables the Core High Voltage Detector. If enabled the Core High Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in low power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in low power mode. false - Disable Core High voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Low Power mode.

This function enables/disables the Core Low Voltage Detector. If enabled the Core Low Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core Low voltage detector in low power mode. false - Disable Core Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect in low power mode successfully.

```
void SPC_SetIOVDDLowVoltageLevel(SPC_Type *base, spc_low_voltage_level_select_t level)
```

Set IO VDD Low-Voltage level selection.

This function selects the IO VDD Low-voltage level. Changing IO VDD low-voltage level must be done after disabling the IO VDD low voltage reset and interrupt.

Parameters

- base – SPC peripheral base address.
- level – IO VDD Low-voltage level selection.

```
void SPC_SetIOVoltageDetectConfig(SPC_Type *base, const spc_io_voltage_detect_config_t *config)
```

Configs IO voltage detect options.

This function config IO voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_voltage_detect_config_t structure.

```
static inline void SPC_LockIOVoltageDetectResetSetting(SPC_Type *base)
```

Lock IO Voltage detect reset setting.

This function locks IO voltage detect reset setting. After invoking this function any configuration of system voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

```
static inline void SPC_UnlockIOVoltageDetectResetSetting(SPC_Type *base)
```

Unlock IO voltage detect reset setting.

This function unlocks IO voltage detect reset setting. If locks the IO voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

```
status_t SPC_EnableActiveModeIOHighVoltageDetect(SPC_Type *base, bool enable)
```

Enables/Disables the IO High Voltage Detector in Active mode.

Note: If the IO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in active mode. false - Disable IO High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable IO High Voltage Detect successfully.

status_t SPC_EnableActiveModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO Low Voltage Detector in Active mode.

Note: If the IO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in active mode. false - Disable IO Low voltage detector in active mode.

Return values

kStatus_Success – Enable IO Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeIOHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO High Voltage Detector in Low Power mode.

Note: If the IO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in low power mode. false - Disable IO High voltage detector in low power mode.

Return values

kStatus_Success – Enable IO High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO Low Voltage Detector in Low Power mode.

Note: If the IO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in low power mode. false - Disable IO Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable IO Low Voltage Detect in low power mode successfully.

```
void SPC_SetExternalVoltageDomainsConfig(SPC_Type *base, uint8_t lowPowerIsoMask, uint8_t IsoMask)
```

Configs external voltage domains.

This function configs external voltage domains isolation.

Parameters

- base – SPC peripheral base address.
- lowPowerIsoMask – The mask of external domains isolate enable during low power mode. Please read the Reference Manual for the Bitmap.
- IsoMask – The mask of external domains isolate. Please read the Reference Manual for the Bitmap.

```
static inline uint8_t SPC_GetExternalDomainsStatus(SPC_Type *base)
```

Gets External Domains status.

This function configs external voltage domains status.

Parameters

- base – SPC peripheral base address.

Returns

The status of each external domain.

```
static inline void SPC_EnableCoreLDORegulator(SPC_Type *base, bool enable)
```

Enable/Disable Core LDO regulator.

Note: The CORE LDO enable bit is write-once.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CORE LDO Regulator. true - Enable CORE LDO Regulator. false - Disable CORE LDO Regulator.

```
static inline void SPC_PullDownCoreLDORegulator(SPC_Type *base, bool pulldown)
```

Enable/Disable the CORE LDO Regulator pull down in Deep Power Down.

Note: This function only useful when enabled the CORE LDO Regulator.

Parameters

- base – SPC peripheral base address.
- pulldown – Enable/Disable CORE LDO pulldown in Deep Power Down mode. true - CORE LDO Regulator will discharge in Deep Power Down mode. false - CORE LDO Regulator will not discharge in Deep Power Down mode.

```
status_t SPC_SetActiveModeCoreLDORegulatorConfig(SPC_Type *base, const  
spc_active_mode_core_ldo_option_t  
*option)
```

Configs Core LDO VDD Regulator in Active mode.

Note: If any voltage detect feature is enabled in Active mode, then CORE_LDO's drive strength must not set to low.

Note: Core VDD level for the Core LDO low power regulator can only be changed when CORELDO_VDD_DS is normal

Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to the `spc_active_mode_core_ldo_option_t` structure.

Return values

- `kStatus_Success` – Config Core LDO regulator in Active power mode successful.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – If any voltage detect enabled, `core_ldo`'s drive strength can not set to low.
- `kStatus_SPC_CORELDOLowVoltageWrong` – The selected voltage level in active mode is not allowed.
- `kStatus_Timeout` – Timeout occurs while waiting completion.

```
status_t SPC_SetLowPowerModeCoreLDORegulatorConfig(SPC_Type *base, const
                                                    spc_lowpower_mode_core_ldo_option_t
                                                    *option)
```

Configs CORE LDO Regulator in low power mode.

This function configs CORE LDO Regulator in Low Power mode. If CORE LDO VDD Drive Strength is set to Normal, the CORE LDO VDD regulator voltage level in Active mode must be equal to the voltage level in Low power mode. And the Bandgap must be programmed to select bandgap enabled. Core VDD voltage levels for the Core LDO low power regulator can only be changed when the CORE LDO Drive Strength set as Normal.

Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to the `spc_lowpower_mode_core_ldo_option_t` structure.

Return values

- `kStatus_Success` – Config Core LDO regulator in power mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `#kStatus_SPC_CORELDOLowVoltageSetFail` – Fail to change Core LDO voltage level.
- `kStatus_Timeout` – Timeout occurs while waiting completion.

```
static inline void SPC_EnableDCDCRegulator(SPC_Type *base, bool enable)
Enable/Disable DCDC Regulator.
```

Note: The DCDC enable bit is write-once.

Parameters

- `base` – SPC peripheral base address.

- enable – Enable/Disable DCDC Regulator. true - Enable DCDC Regulator. false - Disable DCDC Regulator.

status_t SPC_SetActiveModeDCDCRegulatorConfig(SPC_Type *base, const *spc_active_mode_dcdc_option_t* *option)

Configs DCDC VDD Regulator in Active mode.

This function configs DCDC VDD Regulator in Active mode. If DCDC VDD Drive Strength is set to Normal, the Bandgap mode in Active mode must be programmed to a value that enable the bandgap. If any voltage detectors are kept enabled, configuration to set DCDC VDD drive strength to low will be ignored. When switching DCDC from low drive strength to Normal driver strength, make sure the DCDC high VDD LVL setting to the same level that was set prior to switching the DCDC to low drive strength.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the *spc_active_mode_dcdc_option_t* structure.

Return values

- kStatus_Success – Config DCDC regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetLowPowerModeDCDCRegulatorConfig(SPC_Type *base, const *spc_lowpower_mode_dcdc_option_t* *option)

Configs DCDC VDD Regulator in Low power modes.

This function configs DCDC VDD Regulator in Low Power modes. If DCDC VDD Drive Strength is set to Normal, the Bandgap mode in Low Power mode must be programmed to a value that enables the Bandgap. If any of voltage detectors are kept enabled, configuration to set DCDC VDD Drive Strength to Low or Pulse mode will be ignored. In Deep Power Down mode, DCDC regulator is always turned off.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the *spc_lowpower_mode_dcdc_option_t* structure.

Return values

- kStatus_Success – Config DCDC regulator in low power mode successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_DCDCPulseRefreshModeIgnore – Set driver strength to Pulse Refresh mode will be ignored.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low Drive Strength will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetSRAMOperateVoltage(SPC_Type *base, *spc_sram_operat_voltage_t* voltage)
Set the SRAM operate voltage level.

Parameters

- `base` – SPC peripheral base address.
- `voltage` – Target SRAM operate voltage level, please refer to `spc_sram_operat_voltage_t`.

Return values

- `kStatus_Success` – Successfully configured.
- `kStatus_Timeout` – Timeout occurs while waiting completion.

```
void SPC_ConfigVddCoreGlitchDetector(SPC_Type *base, const
                                     spc_vdd_core_glitch_detector_config_t *config)
```

Configures VDD Core Glitch detector, including ripple counter selection, timeout value and so on.

Parameters

- `base` – SPC peripheral base address.
- `config` – Pointer to the structure in type of `spc_vdd_core_glitch_detector_config_t`.

```
static inline bool SPC_CheckGlitchRippleCounterOutput(SPC_Type *base,
                                                       spc_vdd_core_glitch_ripple_counter_select_t
                                                       rippleCounter)
```

Checks selected 4-bit glitch ripple counter's output.

Parameters

- `base` – SPC peripheral base address.
- `rippleCounter` – The ripple counter to check, please refer to `spc_vdd_core_glitch_ripple_counter_select_t`.

Return values

- `true` – The selected ripple counter output is 1, will generate interrupt or reset based on settings.
- `false` – The selected ripple counter output is 0.

```
static inline void SPC_ClearGlitchRippleCounterOutput(SPC_Type *base,
                                                       spc_vdd_core_glitch_ripple_counter_select_t
                                                       rippleCounter)
```

Clears output of selected glitch ripple counter.

Parameters

- `base` – SPC peripheral base address.
- `rippleCounter` – The ripple counter to check, please refer to `spc_vdd_core_glitch_ripple_counter_select_t`.

```
static inline void SPC_LockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)
```

After invoking this function, writes to `SPC_VDD_CORE_GLITCH_DETECT_SC[RE]` register are ignored.

Parameters

- `base` – SPC peripheral base address.

```
static inline void SPC_UnlockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)
```

After invoking this function, writes to `SPC_VDD_CORE_GLITCH_DETECT_SC[RE]` register are allowed.

Parameters

- `base` – SPC peripheral base address.

```
static inline bool SPC_CheckVddCoreVoltageGlitchResetControlState(SPC_Type *base)
    Checks if SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.
```

Parameters

- base – SPC peripheral base address.

Return values

- true – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.
- false – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is not writable.

```
FSL_SPC_DRIVER_VERSION
    SPC driver version 2.8.0.
```

```
SPC_EVD_CFG_REG_EVDISO_SHIFT
```

```
SPC_EVD_CFG_REG_EVDLPISO_SHIFT
```

```
SPC_EVD_CFG_REG_EVDSTAT_SHIFT
```

```
SPC_EVD_CFG_REG_EVDISO(x)
```

```
SPC_EVD_CFG_REG_EVDLPISO(x)
```

```
SPC_EVD_CFG_REG_EVDSTAT(x)
```

```
struct _spc_lowpower_request_config
    #include <fsl_spc.h> Low Power Request output pin configuration.
```

Public Members

- bool enable
Low Power Request Output enable.
- spc_lowpower_request_pin_polarity_t polarity
Low Power Request Output pin polarity select.
- spc_lowpower_request_output_override_t override
Low Power Request Output Override.

```
struct _spc_intergrated_power_switch_config
    #include <fsl_spc.h> Integrated power switch configuration.
```

Note: Legacy structure, will be removed.

Public Members

- bool wakeup
Assert an output pin to un-gate the integrated power switch.
- bool sleep
Assert an output pin to power gate the intergrated power switch.

```
struct _spc_active_mode_core_ldo_option
    #include <fsl_spc.h> Core LDO regulator options in Active mode.
```

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage
Core LDO Regulator Voltage Level selection in Active mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength
Core LDO Regulator Drive Strength selection in Active mode

struct *_spc_active_mode_dcdc_option*
#include <fsl_spc.h> DCDC regulator options in Active mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage
DCDC Regulator Voltage Level selection in Active mode.

spc_dcdc_drive_strength_t DCDCDriveStrength
DCDC VDD Regulator Drive Strength selection in Active mode.

struct *_spc_lowpower_mode_core_ldo_option*
#include <fsl_spc.h> Core LDO regulator options in Low Power mode.

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage
Core LDO Regulator Voltage Level selection in Low Power mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength
Core LDO Regulator Drive Strength selection in Low Power mode

struct *_spc_lowpower_mode_dcdc_option*
#include <fsl_spc.h> DCDC regulator options in Low Power mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage
DCDC Regulator Voltage Level selection in Low Power mode.

spc_dcdc_drive_strength_t DCDCDriveStrength
DCDC VDD Regulator Drive Strength selection in Low Power mode.

struct *_spc_voltage_detect_option*
#include <fsl_spc.h> CORE/SYS/IO VDD Voltage Detect options.

Public Members

bool HVDIInterruptEnable
CORE/SYS/IO VDD High Voltage Detect interrupt enable.

bool HVDRResetEnable
CORE/SYS/IO VDD High Voltage Detect reset enable.

bool LVDIInterruptEnable
CORE/SYS/IO VDD Low Voltage Detect interrupt enable.

bool LVDResetEnable
CORE/SYS/IO VDD Low Voltage Detect reset enable.

```
struct _spc_core_voltage_detect_config
    #include <fsl_spc.h> Core Voltage Detect configuration.
```

Public Members

```
spc_voltage_detect_option_t option
    Core VDD Voltage Detect option.
```

```
struct _spc_io_voltage_detect_config
    #include <fsl_spc.h> IO Voltage Detect Configuration.
```

Public Members

```
spc_voltage_detect_option_t option
    IO VDD Voltage Detect option.

spc_low_voltage_level_select_t level
    IO VDD Low-voltage level selection.
```

```
struct _spc_active_mode_regulators_config
    #include <fsl_spc.h> Active mode configuration.
```

```
struct _spc_lowpower_mode_regulators_config
    #include <fsl_spc.h> Low Power Mode configuration.
```

```
struct _spc_vdd_core_glitch_detector_config
    #include <fsl_spc.h> The configuration of VDD Core glitch detector.
```

Public Members

```
spc_vdd_core_glitch_ripple_counter_select_t rippleCounterSelect
    Used to set ripple counter.
```

```
uint8_t resetTimeoutValue
    The timeout value used to reset glitch detect/compare logic after an initial glitch is detected.
```

```
bool enableReset
    Used to enable/disable POR/LVD reset that caused by CORE VDD glitch detect error.
```

```
bool enableInterrupt
    Used to enable/disable hardware interrupt if CORE VDD glitch detect error.
```

2.68 SYSPM: System Performance Monitor

```
enum _syspm_monitor
    syspm select control monitor
```

Values:

```
enumerator kSYSPM_Monitor0
    Monitor 0
```

```
enum _syspm_event
    syspm select event
```

Values:

```
enumerator kSYSPM_Event1
    Event 1
enumerator kSYSPM_Event2
    Event 2
enumerator kSYSPM_Event3
    Event 3
enum _syspm_mode
    syspm set count mode
Values:
enumerator kSYSPM_BothMode
    count in both modes
enumerator kSYSPM_UserMode
    count only in user mode
enumerator kSYSPM_PrivilegedMode
    count only in privileged mode
enum _syspm_startstop_control
    syspm start/stop control
Values:
enumerator kSYSPM_Idle
    idle >
enumerator kSYSPM_LocalStop
    local stop
enumerator kSYSPM_LocalStart
    local start
enumerator KSYSPM_EnableTraceControl
    enable global TSTART/TSTOP
enumerator kSYSPM_GlobalStart
    global stop
enumerator kSYSPM_GlobalStop
    global start
typedef enum _syspm_monitor syspm_monitor_t
    syspm select control monitor
typedef enum _syspm_event syspm_event_t
    syspm select event
typedef enum _syspm_mode syspm_mode_t
    syspm set count mode
typedef enum _syspm_startstop_control syspm_startstop_control_t
    syspm start/stop control
void SYSPM_Init(SYSPM_Type *base)
    Initializes the SYSPM.
    This function enables the SYSPM clock.
```

Parameters

- *base* – SYSPM peripheral base address.

void SYSPM_Deinit(SYSPM_Type **base*)

Deinitializes the SYSPM.

This function disables the SYSPM clock.

Parameters

- *base* – SYSPM peripheral base address.

void SYSPM_SelectEvent(SYSPM_Type **base*, *syspm_monitor_t* *monitor*, *syspm_event_t* *event*,
uint8_t *eventCode*)

Select event counters.

Parameters

- *base* – SYSPM peripheral base address.
- *event* – *syspm* select event, see to *syspm_event_t*.
- *eventCode* – select which event to be counted in PMECTR_x, see to table Events.

void SYSPM_ResetEvent(SYSPM_Type **base*, *syspm_monitor_t* *monitor*, *syspm_event_t* *event*)

Reset event counters.

Parameters

- *base* – SYSPM peripheral base address.
- *monitor* – *syspm* control monitor, see to *syspm_monitor_t*.

void SYSPM_ResetInstructionEvent(SYSPM_Type **base*, *syspm_monitor_t* *monitor*)

Reset Instruction Counter.

Parameters

- *base* – SYSPM peripheral base address.
- *monitor* – *syspm* control monitor, see to *syspm_monitor_t*.

void SYSPM_SetCountMode(SYSPM_Type **base*, *syspm_monitor_t* *monitor*, *syspm_mode_t*
mode)

Set count mode.

Parameters

- *base* – SYSPM peripheral base address.
- *monitor* – *syspm* control monitor, see to *syspm_monitor_t*.
- *mode* – *syspm* select counter mode, see to *syspm_mode_t*.

void SYSPM_SetStartStopControl(SYSPM_Type **base*, *syspm_monitor_t* *monitor*,
syspm_startstop_control_t *ssc*)

Set Start/Stop Control.

Parameters

- *base* – SYSPM peripheral base address.
- *monitor* – *syspm* control monitor, see to *syspm_monitor_t*.
- *ssc* – This 3-bit field provides a three-phase mechanism to start/stop the counters. It includes a prioritized scheme with local start > local stop > global start > global stop > conditional TSTART > TSTOP. The global and conditional start/stop affect all configured PM/PSAM module concurrently so counters are “coherent”. see to *syspm_startstop_control_t*

void SYSPM_DisableCounter(SYSPM_Type *base, *syspm_monitor_t* monitor)

Disable Counters if Stopped or Halted.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

uint64_t SYSPM_GetEventCounter(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event)

This is the the 40-bits of eventx counter. The value in this register increments each time the event selected in PMCRx[SELEVTx] occurs.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- event – syspm select event, see to *syspm_event_t*.

Returns

- When the return value is not equal to SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE, the return value represents a 40 bits eventx counter.
- When the return value is equal to SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE, the return value represents timeout occurred.

EVENT_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM event count stable (0 means wait forever)

INSTRUCTION_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM instruction count stable (0 means wait forever)

FSL_SYSPM_DRIVER_VERSION

SYSPM driver version.

SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE

2.69 TPM: Timer PWM Module

uint32_t TPM_GetInstance(TPM_Type *base)

Gets the instance from the base address.

Parameters

- base – TPM peripheral base address

Returns

The TPM instance

void TPM_Init(TPM_Type *base, const *tpm_config_t* *config)

Ungates the TPM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the TPM driver.

Parameters

- base – TPM peripheral base address

- config – Pointer to user's TPM config structure.

void TPM_Deinit(TPM_Type *base)

Stops the counter and gates the TPM clock.

Parameters

- base – TPM peripheral base address

void TPM_GetDefaultConfig(*tpm_config_t* *config)

Fill in the TPM config struct with the default settings.

The default values are:

```
config->prescale = kTPM_Prescale_Divide_1;
config->useGlobalTimeBase = false;
config->syncGlobalTimeBase = false;
config->dozeEnable = false;
config->dbgMode = false;
config->enableReloadOnTrigger = false;
config->enableStopOnOverflow = false;
config->enableStartOnTrigger = false;
#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
config->enablePauseOnTrigger = false;
#endif
config->triggerSelect = kTPM_Trigger_Select_0;
#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
config->triggerSource = kTPM_TriggerSource_External;
config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
#endif
#if defined(FSL_FEATURE_TPM_HAS_POL) && FSL_FEATURE_TPM_HAS_POL
config->chnlPolarity = 0U;
#endif
```

Parameters

- config – Pointer to user's TPM config structure.

tpm_clock_prescale_t TPM_CalculateCounterClkDiv(TPM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

Calculates the counter clock prescaler.

This function calculates the values for SC[PS].

return Calculated clock prescaler value.

Parameters

- base – TPM peripheral base address
- counterPeriod_Hz – The desired frequency in Hz which corresponding to the time when the counter reaches the mod value
- srcClock_Hz – TPM counter clock in Hz

status_t TPM_SetupPwm(TPM_Type *base, const *tpm_chnl_pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *tpm_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

- `base` – TPM peripheral base address
- `chnlParams` – Array of PWM channel parameters to configure the channel(s)
- `numOfChnls` – Number of channels to configure, this should be the size of the array passed in
- `mode` – PWM operation mode, options available in enumeration `tpm_pwm_mode_t`
- `pwmFreq_Hz` – PWM signal frequency in Hz
- `srcClock_Hz` – TPM counter clock in Hz

Returns

`kStatus_Success` PWM setup successful
`kStatus_Error` PWM setup failed
`kStatus_Timeout` PWM setup timeout when write register CnV or MOD

```
status_t TPM_UpdatePwmDutycycle(TPM_Type *base, tpm_chnl_t chnlNumber,
                               tpm_pwm_mode_t currentPwmMode, uint8_t
                               dutyCyclePercent)
```

Update the duty cycle of an active PWM signal.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number. In combined mode, this represents the channel pair number
- `currentPwmMode` – The current PWM mode set during PWM setup
- `dutyCyclePercent` – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns

`kStatus_Success` if the PWM setup was successful, `kStatus_Error` on failure

```
void TPM_UpdateChnlEdgeLevelSelect(TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t level)
```

Update the edge level selection for a channel.

Note: When the TPM has PWM pause level select feature (FSL_FEATURE_TPM_HAS_PAUSE_LEVEL_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use `TPM_DisableChannel` API to close the PWM output.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number
- `level` – The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

```
static inline uint8_t TPM_GetChannelControlBits(TPM_Type *base, tpm_chnl_t chnlNumber)
```

Get the channel control bits value (mode, edge and level bit fields).

This function disable the channel by clear all mode and level control bits.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number

Returns

The contorl bits value. This is the logical OR of members of the enumeration `tpm_chnl_control_bit_mask_t`.

```
static inline status_t TPM_DisableChannel(TPM_Type *base, tpm_chnl_t chnlNumber)
```

Disable the channel.

This function disable the channel by clear all mode and level control bits.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number

Returns

`kStatus_Success` PWM setup successful
`kStatus_Timeout` PWM setup timeout when write register CnSC

```
static inline status_t TPM_EnableChannel(TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t control)
```

Enable the channel according to mode and level configs.

This function enable the channel output according to input mode/level config parameters.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number
- `control` – The contorl bits value. This is the logical OR of members of the enumeration `tpm_chnl_control_bit_mask_t`.

Returns

`kStatus_Success` PWM setup successful
`kStatus_Timeout` PWM setup timeout when write register CnSC

```
void TPM_SetupInputCapture(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_input_capture_edge_t captureMode)
```

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the `captureMode` argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number
- `captureMode` – Specifies which edge to capture

```
status_t TPM_SetupOutputCompare(TPM_Type *base, tpm_chnl_t chnlNumber, tpm_output_compare_mode_t compareMode, uint32_t compareValue)
```

Configures the TPM to generate timed pulses.

When the TPM counter matches the value of `compareVal` argument (this is written into CnV reg), the channel output is changed based on what is specified in the `compareMode` argument.

Parameters

- `base` – TPM peripheral base address
- `chnlNumber` – The channel number

- compareMode – Action to take on the channel output when the compare condition is met
- compareValue – Value to be programmed in the CnV register.

Returns

kStatus_Success PWM setup successful
kStatus_Timeout PWM setup timeout when write register CnV

```
void TPM_SetupDualEdgeCapture(TPM_Type *base, tpm_chnl_t chnlPairNumber, const
                             tpm_dual_edge_capture_param_t *edgeParam, uint32_t
                             filterValue)
```

Configures the dual edge capture mode of the TPM.

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

- base – TPM peripheral base address
- chnlPairNumber – The TPM channel pair number; options are 0, 1, 2, 3
- edgeParam – Sets up the dual edge capture function
- filterValue – Filter value, specify 0 to disable filter.

```
void TPM_SetupQuadDecode(TPM_Type *base, const tpm_phase_params_t *phaseAParams,
                         const tpm_phase_params_t *phaseBParams,
                         tpm_quad_decode_mode_t quadMode)
```

Configures the parameters and activates the quadrature decode mode.

Parameters

- base – TPM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters
- quadMode – Selects encoding mode used in quadrature decoder mode

```
static inline void TPM_SetChannelPolarity(TPM_Type *base, tpm_chnl_t chnlNumber, bool
                                         enable)
```

Set the input and output polarity of each of the channels.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Set the channel polarity to active high; false: Set the channel polarity to active low;

```
static inline void TPM_EnableChannelExtTrigger(TPM_Type *base, tpm_chnl_t chnlNumber, bool
                                              enable)
```

Enable external trigger input to be used by channel.

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

Note: No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

void TPM_EnableInterrupts(TPM_Type *base, uint32_t mask)

Enables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `tpm_interrupt_enable_t`

void TPM_DisableInterrupts(TPM_Type *base, uint32_t mask)

Disables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration `tpm_interrupt_enable_t`

uint32_t TPM_GetEnabledInterrupts(TPM_Type *base)

Gets the enabled TPM interrupts.

Parameters

- base – TPM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `tpm_interrupt_enable_t`

void TPM_RegisterCallBack(TPM_Type *base, *tpm_callback_t* callback)

Register callback.

If channel or overflow interrupt is enabled by the user, then a callback can be registered which will be invoked when the interrupt is triggered.

Parameters

- base – TPM peripheral base address
- callback – Callback function

static inline uint32_t TPM_GetChannelValue(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Gets the TPM channel value.

Note: The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns

The channle CnV regisyer value.

```
static inline uint32_t TPM_GetStatusFlags(TPM_Type *base)
```

Gets the TPM status flags.

Parameters

- *base* – TPM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `tpm_status_flags_t`

```
static inline void TPM_ClearStatusFlags(TPM_Type *base, uint32_t mask)
```

Clears the TPM status flags.

Parameters

- *base* – TPM peripheral base address
- *mask* – The status flags to clear. This is a logical OR of members of the enumeration `tpm_status_flags_t`

```
static inline status_t TPM_SetTimerPeriod(TPM_Type *base, uint32_t ticks)
```

Sets the timer period in units of ticks.

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note:

- This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.
 - Call the utility macros provided in the `fsl_common.h` to convert usec or msec to ticks.
-

Parameters

- *base* – TPM peripheral base address
- *ticks* – A timer period in units of ticks, which should be equal or greater than 1.

Returns

`kStatus_Success` PWM setup successful
`kStatus_Timeout` PWM setup timeout when write register CnSC

```
static inline uint32_t TPM_GetCurrentTimerCount(TPM_Type *base)
```

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

- *base* – TPM peripheral base address

Returns

The current counter value in ticks

```
static inline void TPM_StartTimer(TPM_Type *base, tpm_clock_source_t clockSource)
```

Starts the TPM counter.

Parameters

- base – TPM peripheral base address
- clockSource – TPM clock source; once clock source is set the counter will start running

static inline *status_t* TPM_StopTimer(TPM_Type *base)

Stops the TPM counter.

Parameters

- base – TPM peripheral base address

Returns

kStatus_Success PWM setup successful
kStatus_Timeout PWM setup timeout
when write register CnSC

FSL_TPM_DRIVER_VERSION

TPM driver version 2.4.0.

enum *_tpm_chnl*

List of TPM channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kTPM_Chnl_0
TPM channel number 0

enumerator kTPM_Chnl_1
TPM channel number 1

enumerator kTPM_Chnl_2
TPM channel number 2

enumerator kTPM_Chnl_3
TPM channel number 3

enumerator kTPM_Chnl_4
TPM channel number 4

enumerator kTPM_Chnl_5
TPM channel number 5

enumerator kTPM_Chnl_6
TPM channel number 6

enumerator kTPM_Chnl_7
TPM channel number 7

enum *_tpm_pwm_mode*

TPM PWM operation modes.

Values:

enumerator kTPM_EdgeAlignedPwm
Edge aligned PWM

enumerator kTPM_CenterAlignedPwm
Center aligned PWM

enumerator kTPM_CombinedPwm

Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

enum _tpm_pwm_level_select

TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

Values:

enumerator kTPM_HighTrue

High true pulses

enumerator kTPM_LowTrue

Low true pulses

enum _tpm_pwm_pause_level_select

TPM PWM output when first enabled or paused: set or clear.

Values:

enumerator kTPM_ClearOnPause

Clear Output when counter first enabled or paused.

enumerator kTPM_SetOnPause

Set Output when counter first enabled or paused.

enum _tpm_chnl_control_bit_mask

List of TPM channel modes and level control bit mask.

Values:

enumerator kTPM_ChnlELSnAMask

Channel ELSA bit mask.

enumerator kTPM_ChnlELSnBMask

Channel ELSB bit mask.

enumerator kTPM_ChnlMSAMask

Channel MSA bit mask.

enumerator kTPM_ChnlMSBMask

Channel MSB bit mask.

enum _tpm_trigger_select

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

Values:

enumerator kTPM_Trigger_Select_0

enumerator kTPM_Trigger_Select_1

enumerator kTPM_Trigger_Select_2
enumerator kTPM_Trigger_Select_3
enumerator kTPM_Trigger_Select_4
enumerator kTPM_Trigger_Select_5
enumerator kTPM_Trigger_Select_6
enumerator kTPM_Trigger_Select_7
enumerator kTPM_Trigger_Select_8
enumerator kTPM_Trigger_Select_9
enumerator kTPM_Trigger_Select_10
enumerator kTPM_Trigger_Select_11
enumerator kTPM_Trigger_Select_12
enumerator kTPM_Trigger_Select_13
enumerator kTPM_Trigger_Select_14
enumerator kTPM_Trigger_Select_15

enum _tpm_trigger_source

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

Values:

enumerator kTPM_TriggerSource_External

Use external trigger input

enumerator kTPM_TriggerSource_Internal

Use internal trigger (channel pin input capture)

enum _tpm_ext_trigger_polarity

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

Values:

enumerator kTPM_ExtTrigger_Active_High

External trigger input is active high

enumerator kTPM_ExtTrigger_Active_Low

External trigger input is active low

enum _tpm_output_compare_mode

TPM output compare modes.

Values:

enumerator kTPM_NoOutputSignal

No channel output when counter reaches CnV

enumerator kTPM_ToggleOnMatch
Toggle output

enumerator kTPM_ClearOnMatch
Clear output

enumerator kTPM_SetOnMatch
Set output

enumerator kTPM_HighPulseOutput
Pulse output high

enumerator kTPM_LowPulseOutput
Pulse output low

enum _tpm_input_capture_edge
TPM input capture edge.

Values:

enumerator kTPM_RisingEdge
Capture on rising edge only

enumerator kTPM_FallingEdge
Capture on falling edge only

enumerator kTPM_RiseAndFallEdge
Capture on rising or falling edge

enum _tpm_quad_decode_mode
TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

Values:

enumerator kTPM_QuadPhaseEncode
Phase A and Phase B encoding mode

enumerator kTPM_QuadCountAndDir
Count and direction encoding mode

enum _tpm_phase_polarity
TPM quadrature phase polarities.

Values:

enumerator kTPM_QuadPhaseNormal
Phase input signal is not inverted

enumerator kTPM_QuadPhaseInvert
Phase input signal is inverted

enum _tpm_clock_source
TPM clock source selection.

Values:

enumerator kTPM_SystemClock
System clock

enumerator kTPM_ExternalClock
External TPM_EXTCLK pin clock

enumerator kTPM_ExternalInputTriggerClock
Selected external input trigger clock

enum _tpm_clock_prescale
TPM prescale value selection for the clock source.

Values:

enumerator kTPM_Prescale_Divide_1
Divide by 1

enumerator kTPM_Prescale_Divide_2
Divide by 2

enumerator kTPM_Prescale_Divide_4
Divide by 4

enumerator kTPM_Prescale_Divide_8
Divide by 8

enumerator kTPM_Prescale_Divide_16
Divide by 16

enumerator kTPM_Prescale_Divide_32
Divide by 32

enumerator kTPM_Prescale_Divide_64
Divide by 64

enumerator kTPM_Prescale_Divide_128
Divide by 128

enum _tpm_interrupt_enable
List of TPM interrupts.

Values:

enumerator kTPM_Chnl0InterruptEnable
Channel 0 interrupt.

enumerator kTPM_Chnl1InterruptEnable
Channel 1 interrupt.

enumerator kTPM_Chnl2InterruptEnable
Channel 2 interrupt.

enumerator kTPM_Chnl3InterruptEnable
Channel 3 interrupt.

enumerator kTPM_Chnl4InterruptEnable
Channel 4 interrupt.

enumerator kTPM_Chnl5InterruptEnable
Channel 5 interrupt.

enumerator kTPM_Chnl6InterruptEnable
Channel 6 interrupt.

enumerator kTPM_Chnl7InterruptEnable
Channel 7 interrupt.

enumerator kTPM_TimeOverflowInterruptEnable
Time overflow interrupt.

enum _tpm_status_flags
List of TPM flags.

Values:

enumerator kTPM_Chnl0Flag
Channel 0 flag

enumerator kTPM_Chnl1Flag
Channel 1 flag

enumerator kTPM_Chnl2Flag
Channel 2 flag

enumerator kTPM_Chnl3Flag
Channel 3 flag

enumerator kTPM_Chnl4Flag
Channel 4 flag

enumerator kTPM_Chnl5Flag
Channel 5 flag

enumerator kTPM_Chnl6Flag
Channel 6 flag

enumerator kTPM_Chnl7Flag
Channel 7 flag

enumerator kTPM_TimeOverflowFlag
Time overflow flag

typedef enum _tpm_chnl tpm_chnl_t
List of TPM channels.

Note: Actual number of available channels is SoC dependent

typedef enum _tpm_pwm_mode tpm_pwm_mode_t
TPM PWM operation modes.

typedef enum _tpm_pwm_level_select tpm_pwm_level_select_t
TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

typedef enum _tpm_pwm_pause_level_select tpm_pwm_pause_level_select_t
TPM PWM output when first enabled or paused: set or clear.

typedef enum _tpm_chnl_control_bit_mask tpm_chnl_control_bit_mask_t
List of TPM channel modes and level control bit mask.

typedef struct _tpm_chnl_pwm_signal_param tpm_chnl_pwm_signal_param_t
Options to configure a TPM channel's PWM signal.

typedef enum *_tpm_trigger_select* tpm_trigger_select_t

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

typedef enum *_tpm_trigger_source* tpm_trigger_source_t

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

typedef enum *_tpm_ext_trigger_polarity* tpm_ext_trigger_polarity_t

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

typedef enum *_tpm_output_compare_mode* tpm_output_compare_mode_t

TPM output compare modes.

typedef enum *_tpm_input_capture_edge* tpm_input_capture_edge_t

TPM input capture edge.

typedef struct *_tpm_dual_edge_capture_param* tpm_dual_edge_capture_param_t

TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_quad_decode_mode* tpm_quad_decode_mode_t

TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_phase_polarity* tpm_phase_polarity_t

TPM quadrature phase polarities.

typedef struct *_tpm_phase_param* tpm_phase_params_t

TPM quadrature decode phase parameters.

typedef enum *_tpm_clock_source* tpm_clock_source_t

TPM clock source selection.

typedef enum *_tpm_clock_prescale* tpm_clock_prescale_t

TPM prescale value selection for the clock source.

typedef struct *_tpm_config* tpm_config_t

TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the `TPM_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
typedef enum _tpm_interrupt_enable tpm_interrupt_enable_t
```

List of TPM interrupts.

```
typedef enum _tpm_status_flags tpm_status_flags_t
```

List of TPM flags.

```
typedef void (*tpm_callback_t)(TPM_Type *base)
```

TPM callback function pointer.

Param base

TPM peripheral base address.

```
static inline void TPM_Reset(TPM_Type *base)
```

Performs a software reset on the TPM module.

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

Note: TPM software reset is available on certain SoC's only

Parameters

- base – TPM peripheral base address

```
void TPM_DriverIRQHandler(uint32_t instance)
```

TPM driver IRQ handler common entry.

This function provides the common IRQ request entry for TPM.

Parameters

- instance – TPM instance.

```
TPM_TIMEOUT
```

Max loops to wait for writing register.

When writing MOD CnV CnSC and SC register, driver will wait until register is updated. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

```
TPM_MAX_COUNTER_VALUE(x)
```

Help macro to get the max counter value.

```
struct _tpm_chnl_pwm_signal_param
```

#include <fsl_tpm.h> Options to configure a TPM channel's PWM signal.

Public Members

```
tpm_chnl_t chnlNumber
```

TPM channel to configure. In combined mode (available in some SoC's), this represents the channel pair number

```
tpm_pwm_pause_level_select_t pauseLevel
```

PWM output level when counter first enabled or paused

```
tpm_pwm_level_select_t level
```

PWM output active level select

```
uint8_t dutyCyclePercent
```

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

uint8_t firstEdgeDelayPercent

Used only in combined PWM mode to generate asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greater than 100.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

tpm_pwm_pause_level_select_t secPauseLevel

Used only in combined PWM mode. Define the second channel output level when counter first enabled or paused

uint8_t deadTimeValue[2]

The dead time value for channel n and n+1 in combined complementary PWM mode. Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue * 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

struct _tpm_dual_edge_capture_param

#include <fsl_tpm.h> TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

Public Members

bool enableSwap

true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored

tpm_input_capture_edge_t currChanEdgeMode

Input capture edge select for channel n

tpm_input_capture_edge_t nextChanEdgeMode

Input capture edge select for channel n+1

struct _tpm_phase_param

#include <fsl_tpm.h> TPM quadrature decode phase parameters.

Public Members

uint32_t phaseFilterVal

Filter value, filter is disabled when the value is zero

tpm_phase_polarity_t phasePolarity

Phase polarity

struct _tpm_config

#include <fsl_tpm.h> TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the TPM_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

tpm_clock_prescale_t prescale

Select TPM clock prescale value

bool useGlobalTimeBase

true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase

bool syncGlobalTimeBase

true: The TPM counter is synchronized to the global time base; false: disabled

tpm_trigger_select_t triggerSelect

Input trigger to use for controlling the counter operation

tpm_trigger_source_t triggerSource

Decides if we use external or internal trigger.

tpm_ext_trigger_polarity_t extTriggerPolarity

when using external trigger source, need selects the polarity of it.

bool enableDoze

true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode

bool enableDebugMode

true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode

bool enableReloadOnTrigger

true: TPM counter is reloaded on trigger; false: TPM counter not reloaded

bool enableStopOnOverflow

true: TPM counter stops after overflow; false: TPM counter continues running after overflow

bool enableStartOnTrigger

true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately

bool enablePauseOnTrigger

true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

uint8_t chnlPolarity

Defines the input/output polarity of the channels in POL register

2.70 TRDC: Trusted Resource Domain Controller

void TRDC_Init(TRDC_Type *base)

Initializes the TRDC module.

This function enables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

void TRDC_Deinit(TRDC_Type *base)

De-initializes the TRDC module.

This function disables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

static inline uint8_t TRDC_GetCurrentMasterDomainId(TRDC_Type *base)

Gets the domain ID of the current bus master.

Parameters

- base – TRDC peripheral base address.

Returns

Domain ID of current bus master.

void TRDC_GetHardwareConfig(TRDC_Type *base, *trdc_hardware_config_t* *config)

Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.

static inline void TRDC_SetDacGlobalValid(TRDC_Type *base)

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline void TRDC_LockMasterDomainAssignment(TRDC_Type *base, uint8_t master)

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.

static inline void TRDC_SetMasterDomainAssignmentValid(TRDC_Type *base, uint8_t master, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.
- valid – True to set valid, false to set invalid.

```
void TRDC_GetDefaultProcessorDomainAssignment(trdc_processor_domain_assignment_t
                                             *domainAssignment)
```

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->domainIdSelect = kTRDC_DidMda;
assignment->lock          = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_GetDefaultNonProcessorDomainAssignment(trdc_non_processor_domain_assignment_t
                                                  *domainAssignment)
```

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->privilegeAttr  = kTRDC_ForceUser;
assignment->secureAttr    = kTRDC_ForceSecure;
assignment->bypassDomainId = 0U;
assignment->lock          = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

```
void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, const
                                       trdc_processor_domain_assignment_t
                                       *domainAssignment)
```

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;

TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the assignment structure.

```
static inline void TRDC_EnableProcessorDomainAssignment(TRDC_Type *base, bool enable)
```

Enables the processor bus master domain assignment.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

```
void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const
                                         trdc_non_processor_domain_assignment_t
                                         *domainAssignment)
```

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to `trdc_master_t` in processor header file.
- domainAssignment – Pointer to the assignment structure.

```
void TRDC_GetDefaultIDAUConfig(trdc_idau_config_t *idauConfiguration)
```

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```
config->lockSecureVTOR   = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU    = false;
config->lockNonsecureMPU = false;
config->lockSAU          = false;
```

Parameters

- idauConfiguration – Pointer to the configuration structure.

```
void TRDC_SetIDAU(TRDC_Type *base, const trdc_idau_config_t *idauConfiguration)
```

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
idauConfiguration.lockNonsecureMPU = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- idauConfiguration – Pointer to the configuration structure.

```
static inline void TRDC_EnableFlashLogicalWindow(TRDC_Type *base, bool enable)
```

Enables/disables the FLW(flash logical window) function.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

```
static inline void TRDC_LockFlashLogicalWindow(TRDC_Type *base)
```

Locks FLW registers. Once locked the registers can not be updated until next reset.

Parameters

- base – TRDC peripheral base address.

```
static inline uint32_t TRDC_GetFlashLogicalWindowPbase(TRDC_Type *base)
```

Gets the FLW physical base address.

Parameters

- base – TRDC peripheral base address.

Returns

Physical address of the FLW function.

```
static inline void TRDC_GetSetFlashLogicalWindowSize(TRDC_Type *base, uint16_t size)
```

Sets the FLW size.

Parameters

- base – TRDC peripheral base address.
- size – Size of the FLW in unit of 32k bytes.

```
void TRDC_GetDefaultFlashLogicalWindowConfig(trdc_flw_config_t *flwConfiguration)
```

Gets the default FLW(Flash Logical Window) configuration.

```
config->blockCount = false;
config->arrayBaseAddr = false;
config->lock = false;
config->enable = false;
```

Parameters

- flwConfiguration – Pointer to the configuration structure.

```
void TRDC_SetFlashLogicalWindow(TRDC_Type *base, const trdc_flw_config_t
                               *flwConfiguration)
```

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xFFFFFFFF;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- flwConfiguration – Pointer to the configuration structure.

`status_t TRDC_GetAndClearFirstDomainError(TRDC_Type *base, trdc_domain_error_t *error)`

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- `base` – TRDC peripheral base address.
- `error` – Pointer to the error information.

Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter `error`. If no access violation is captured, this function returns the `kStatus_NoData`.

`status_t TRDC_GetAndClearFirstSpecificDomainError(TRDC_Type *base, trdc_domain_error_t *error, uint8_t domainId)`

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- `base` – TRDC peripheral base address.
- `error` – Pointer to the error information.
- `domainId` – The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter `error`. If no access violation is captured, this function returns the `kStatus_NoData`.

`static inline void TRDC_SetMrcGlobalValid(TRDC_Type *base)`

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- `base` – TRDC peripheral base address.

`static inline uint8_t TRDC_GetMrcRegionNumber(TRDC_Type *base, uint8_t mrcIdx)`

Gets the TRDC MRC(Memory Region Checkers) region number valid.

Parameters

- `base` – TRDC peripheral base address.
- `mrcIdx` – MRC index.

Returns

the region number of the given MRC instance

`void TRDC_MrcSetMemoryAccessConfig(TRDC_Type *base, const trdc_memory_access_control_config_t *config, uint8_t mrcIdx, uint8_t regIdx)`

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mrcIdx – MRC index.
- regIdx – Register number.

```
void TRDC_MrcEnableDomainNseUpdate(TRDC_Type *base, uint8_t mrcIdx, uint16_t
                                   domainMask, bool enable)
```

Enables the update of the selected domains.

After the domains' update are enabled, their regions' NSE bits can be set or clear.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

```
void TRDC_MrcRegionNseSet(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

```
void TRDC_MrcRegionNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to clear.

```
void TRDC_MrcDomainNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)
```

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains whose NSE bits to clear.

```
void TRDC__MrcSetRegionDescriptorConfig(TRDC_Type *base, const
                                         trdc_mrc_region_descriptor_config_t *config)
```

Sets the configuration for one of the region descriptor per domain per MRC instance.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to region descriptor configuration structure.

```
static inline void TRDC__SetMbcGlobalValid(TRDC_Type *base)
```

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

```
void TRDC__GetMbcHardwareConfig(TRDC_Type *base, trdc_slave_memory_hardware_config_t
                                *config, uint8_t mbcIdx, uint8_t slvIdx)
```

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

```
void TRDC__MbcSetNseUpdateConfig(TRDC_Type *base, const trdc_mbc_nse_update_config_t
                                  *config, uint8_t mbcIdx)
```

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be update by NSE set/clear.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

```
void TRDC__MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)
```

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

```
void TRDC_MbcWordNseClear(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)
```

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

```
void TRDC_MbcNseClearAll(TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slaveMask)
```

Clears all configuration words' NSE bits of the selected domain and memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

```
void TRDC_MbcSetMemoryAccessConfig(TRDC_Type *base, const
    trdc_memory_access_control_config_t *config, uint8_t
    mbcIdx, uint8_t rgdIdx)
```

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

```
void TRDC_MbcSetMemoryBlockConfig(TRDC_Type *base, const
    trdc_mbc_memory_block_config_t *config)
```

Sets the configuration for one of the memory block per domain per MBC instance.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

enum `_trdc_did_sel`

TRDC domain ID select method, the register bit `TRDC_MDA_W0_0_DFMT0[DIDS]`, used for domain hit evaluation.

Values:

enumerator `kTRDC_DidMda`

Use `MDAn[2:0]` as DID.

enumerator `kTRDC_DidInput`

Use the input DID (`DID_in`) as DID.

enumerator `kTRDC_DidMdaAndInput`

Use `MDAn[2]` concatenated with `DID_in[1:0]` as DID.

enumerator `kTRDC_DidReserved`

Reserved.

enum `_trdc_secure_attr`

TRDC secure attribute, the register bit `TRDC_MDA_W0_0_DFMT0[SA]`, used for bus master domain assignment.

Values:

enumerator `kTRDC_ForceSecure`

Force the bus attribute for this master to secure.

enumerator `kTRDC_ForceNonSecure`

Force the bus attribute for this master to non-secure.

enumerator `kTRDC_MasterSecure`

Use the bus master's secure/nonsecure attribute directly.

enumerator `kTRDC_MasterSecure1`

Use the bus master's secure/nonsecure attribute directly.

enum `_trdc_privilege_attr`

TRDC privileged attribute, the register bit `TRDC_MDA_W0_x_DFMT1[PA]`, used for non-processor bus master domain assignment.

Values:

enumerator `kTRDC_ForceUser`

Force the bus attribute for this master to user.

enumerator `kTRDC_ForcePrivilege`

Force the bus attribute for this master to privileged.

enumerator `kTRDC_MasterPrivilege`

Use the bus master's attribute directly.

enumerator `kTRDC_MasterPrivilege1`

Use the bus master's attribute directly.

enum `_trdc_controller`

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call `TRDC_GetHardwareConfig` to get the actual count.

Values:

enumerator `kTRDC_MemBlockController0`

Memory block checker 0.

enumerator kTRDC_MemBlockController1
Memory block checker 1.

enumerator kTRDC_MemBlockController2
Memory block checker 2.

enumerator kTRDC_MemBlockController3
Memory block checker 3.

enumerator kTRDC_MemRegionChecker0
Memory region checker 0.

enumerator kTRDC_MemRegionChecker1
Memory region checker 1.

enumerator kTRDC_MemRegionChecker2
Memory region checker 2.

enumerator kTRDC_MemRegionChecker3
Memory region checker 3.

enumerator kTRDC_MemRegionChecker4
Memory region checker 4.

enumerator kTRDC_MemRegionChecker5
Memory region checker 5.

enumerator kTRDC_MemRegionChecker6
Memory region checker 6.

enum _trdc_error_state

TRDC domain error state	definition	TRDC_MBCn_DERR_W1[EST]	or
TRDC_MRCn_DERR_W1[EST].			

Values:

enumerator kTRDC_ErrorStateNone
No access violation detected.

enumerator kTRDC_ErrorStateNone1
No access violation detected.

enumerator kTRDC_ErrorStateSingle
Single access violation detected.

enumerator kTRDC_ErrorStateMulti
Multiple access violation detected.

enum _trdc_error_attr

TRDC domain error attribute	definition	TRDC_MBCn_DERR_W1[EATR]	or
TRDC_MRCn_DERR_W1[EATR].			

Values:

enumerator kTRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kTRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kTRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorSecurePrivilegeData
Secure privileged mode, data access.

enumerator kTRDC_ErrorNonSecureUserInst
NonSecure user mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecureUserData
NonSecure user mode, data access.

enumerator kTRDC_ErrorNonSecurePrivilegeInst
NonSecure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecurePrivilegeData
NonSecure privileged mode, data access.

enum _trdc_error_type
TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

Values:

enumerator kTRDC_ErrorTypeRead
Error occurs on read reference.

enumerator kTRDC_ErrorTypeWrite
Error occurs on write reference.

enum _trdc_region_descriptor

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

Values:

enumerator kTRDC_RegionDescriptor0
Region descriptor 0.

enumerator kTRDC_RegionDescriptor1
Region descriptor 1.

enumerator kTRDC_RegionDescriptor2
Region descriptor 2.

enumerator kTRDC_RegionDescriptor3
Region descriptor 3.

enumerator kTRDC_RegionDescriptor4
Region descriptor 4.

enumerator kTRDC_RegionDescriptor5
Region descriptor 5.

enumerator kTRDC_RegionDescriptor6
Region descriptor 6.

enumerator kTRDC_RegionDescriptor7
Region descriptor 7.

enumerator kTRDC_RegionDescriptor8
Region descriptor 8.

enumerator kTRDC_RegionDescriptor9
Region descriptor 9.

enumerator kTRDC_RegionDescriptor10
Region descriptor 10.

enumerator kTRDC_RegionDescriptor11

Region descriptor 11.

enumerator kTRDC_RegionDescriptor12

Region descriptor 12.

enumerator kTRDC_RegionDescriptor13

Region descriptor 13.

enumerator kTRDC_RegionDescriptor14

Region descriptor 14.

enumerator kTRDC_RegionDescriptor15

Region descriptor 15.

enum _trdc_MRC_domain

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

Values:

enumerator kTRDC_MrcDomain0

Domain 0.

enumerator kTRDC_MrcDomain1

Domain 1.

enumerator kTRDC_MrcDomain2

Domain 2.

enumerator kTRDC_MrcDomain3

Domain 3.

enumerator kTRDC_MrcDomain4

Domain 4.

enumerator kTRDC_MrcDomain5

Domain 5.

enumerator kTRDC_MrcDomain6

Domain 6.

enumerator kTRDC_MrcDomain7

Domain 7.

enumerator kTRDC_MrcDomain8

Domain 8.

enumerator kTRDC_MrcDomain9

Domain 9.

enumerator kTRDC_MrcDomain10

Domain 10.

enumerator kTRDC_MrcDomain11

Domain 11.

enumerator kTRDC_MrcDomain12

Domain 12.

enumerator kTRDC_MrcDomain13

Domain 13.

enumerator kTRDC_MrcDomain14
Domain 14.

enumerator kTRDC_MrcDomain15
Domain 15.

enum __trdc_MBC_domain

The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

Values:

enumerator kTRDC_MbcDomain0
Domain 0.

enumerator kTRDC_MbcDomain1
Domain 1.

enumerator kTRDC_MbcDomain2
Domain 2.

enumerator kTRDC_MbcDomain3
Domain 3.

enumerator kTRDC_MbcDomain4
Domain 4.

enumerator kTRDC_MbcDomain5
Domain 5.

enumerator kTRDC_MbcDomain6
Domain 6.

enumerator kTRDC_MbcDomain7
Domain 7.

enum __trdc_MBC_memory

The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

Values:

enumerator kTRDC_MbcSlaveMemory0
Memory 0.

enumerator kTRDC_MbcSlaveMemory1
Memory 1.

enumerator kTRDC_MbcSlaveMemory2
Memory 2.

enumerator kTRDC_MbcSlaveMemory3
Memory 3.

enum __trdc_MBC_bit

The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

Values:

enumerator kTRDC_MbcBit0
Bit 0.

enumerator kTRDC_MbcBit1
Bit 1.

enumerator kTRDC_MbcBit2
Bit 2.

enumerator kTRDC_MbcBit3
Bit 3.

enumerator kTRDC_MbcBit4
Bit 4.

enumerator kTRDC_MbcBit5
Bit 5.

enumerator kTRDC_MbcBit6
Bit 6.

enumerator kTRDC_MbcBit7
Bit 7.

enumerator kTRDC_MbcBit8
Bit 8.

enumerator kTRDC_MbcBit9
Bit 9.

enumerator kTRDC_MbcBit10
Bit 10.

enumerator kTRDC_MbcBit11
Bit 11.

enumerator kTRDC_MbcBit12
Bit 12.

enumerator kTRDC_MbcBit13
Bit 13.

enumerator kTRDC_MbcBit14
Bit 14.

enumerator kTRDC_MbcBit15
Bit 15.

enumerator kTRDC_MbcBit16
Bit 16.

enumerator kTRDC_MbcBit17
Bit 17.

enumerator kTRDC_MbcBit18
Bit 18.

enumerator kTRDC_MbcBit19
Bit 19.

enumerator kTRDC_MbcBit20
Bit 20.

enumerator kTRDC_MbcBit21
Bit 21.

enumerator kTRDC_MbcBit22
Bit 22.

enumerator `kTRDC_MbcBit23`

Bit 23.

enumerator `kTRDC_MbcBit24`

Bit 24.

enumerator `kTRDC_MbcBit25`

Bit 25.

enumerator `kTRDC_MbcBit26`

Bit 26.

enumerator `kTRDC_MbcBit27`

Bit 27.

enumerator `kTRDC_MbcBit28`

Bit 28.

enumerator `kTRDC_MbcBit29`

Bit 29.

enumerator `kTRDC_MbcBit30`

Bit 30.

enumerator `kTRDC_MbcBit31`

Bit 31.

typedef struct *_trdc_hardware_config* trdc_hardware_config_t

TRDC hardware configuration.

typedef struct *_trdc_slave_memory_hardware_config* trdc_slave_memory_hardware_config_t

Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *_trdc_did_sel* trdc_did_sel_t

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

typedef enum *_trdc_secure_attr* trdc_secure_attr_t

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

typedef struct *_trdc_processor_domain_assignment* trdc_processor_domain_assignment_t

Domain assignment for the processor bus master.

typedef enum *_trdc_privilege_attr* trdc_privilege_attr_t

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

typedef struct *_trdc_non_processor_domain_assignment*

trdc_non_processor_domain_assignment_t

Domain assignment for the non-processor bus master.

typedef struct *_trdc_idau_config* trdc_idau_config_t

IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct *_trdc_flw_config* trdc_flw_config_t

FLW(Flash Logical Window) configuration.

typedef enum *_trdc_controller* trdc_controller_t

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

```
typedef enum _trdc_error_state trdc_error_state_t
    TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or
    TRDC_MRCn_DERR_W1[EST].
```

```
typedef enum _trdc_error_attr trdc_error_attr_t
    TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or
    TRDC_MRCn_DERR_W1[EATR].
```

```
typedef enum _trdc_error_type trdc_error_type_t
    TRDC domain error access type definition TRDC_DERR_W1_n[ERW].
```

```
typedef struct _trdc_domain_error trdc_domain_error_t
    TRDC domain error definition.
```

```
typedef struct _trdc_memory_access_control_config trdc_memory_access_control_config_t
    Memory access control configuration for MBC/MRC.
```

```
typedef struct _trdc_mrc_region_descriptor_config trdc_mrc_region_descriptor_config_t
    The configuration of each region descriptor per domain per MRC instance.
```

```
typedef struct _trdc_mbc_nse_update_config trdc_mbc_nse_update_config_t
    The configuration of MBC NSE update.
```

```
typedef struct _trdc_mbc_memory_block_config trdc_mbc_memory_block_config_t
    The configuration of each memory block per domain per MBC instance.
```

```
FSL_TRDC_DRIVER_VERSION
```

```
struct _trdc_hardware_config
    #include <fsl_trdc.h> TRDC hardware configuration.
```

Public Members

```
uint8_t masterNumber
    Number of bus masters.
```

```
uint8_t domainNumber
    Number of domains.
```

```
uint8_t mbcNumber
    Number of MBCs.
```

```
uint8_t mrcNumber
    Number of MRCs.
```

```
struct _trdc_slave_memory_hardware_config
    #include <fsl_trdc.h> Hardware configuration of the two slave memories within each
    MBC(memory block checker).
```

Public Members

```
uint32_t blockNum
    Number of blocks.
```

```
uint32_t blockSize
    Block size.
```

```
struct _trdc_processor_domain_assignment
    #include <fsl_trdc.h> Domain assignment for the processor bus master.
```

Public Members

uint32_t domainId

Domain ID.

uint32_t domainIdSelect

Domain ID select method, see trdc_did_sel_t.

uint32_t __pad0__

Reserved.

uint32_t secureAttr

Secure attribute, see trdc_secure_attr_t.

uint32_t __pad1__

Reserved.

uint32_t lock

Lock the register.

uint32_t __pad2__

Reserved.

struct _trdc_non_processor_domain_assignment

#include <fsl_trdc.h> Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId

Domain ID.

uint32_t privilegeAttr

Privileged attribute, see trdc_privilege_attr_t.

uint32_t secureAttr

Secure attribute, see trdc_secure_attr_t.

uint32_t bypassDomainId

Bypass domain ID.

uint32_t __pad0__

Reserved.

uint32_t lock

Lock the register.

uint32_t __pad1__

Reserved.

struct _trdc_idau_config

#include <fsl_trdc.h> IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

Public Members

uint32_t __pad0__

Reserved.

uint32_t lockSecureVTOR

Disable writes to secure VTOR(Vector Table Offset Register).

uint32_t lockNonsecureVTOR

Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32_t lockSecureMPU

Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32_t lockNonsecureMPU

Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32_t lockSAU

Disable writes to SAU(Security Attribution Unit) registers.

uint32_t __pad1__

Reserved.

struct _trdc_flw_config

#include <fsl_trdc.h> FLW(Flash Logical Window) configuration.

Public Members

uint16_t blockCount

Block count of the Flash Logic Window in 32KByte blocks.

uint32_t arrayBaseAddr

Flash array base address of the Flash Logical Window.

bool lock

Disable writes to FLW registers.

bool enable

Enable FLW function.

struct _trdc_domain_error

#include <fsl_trdc.h> TRDC domain error definition.

Public Members

trdc_controller_t controller

Which controller captured access violation.

uint32_t address

Access address that generated access violation.

trdc_error_state_t errorState

Error state.

trdc_error_attr_t errorAttr

Error attribute.

trdc_error_type_t errorType

Error type.

uint8_t errorPort

Error port.

uint8_t domainId

Domain ID.

uint8_t slaveMemoryIdx

The slave memory index. Only apply when violation in MBC.

struct _trdc_memory_access_control_config

#include <fsl_trdc.h> Memory access control configuration for MBC/MRC.

Public Members

uint32_t nonsecureUsrX

Allow nonsecure user execute access.

uint32_t nonsecureUsrW

Allow nonsecure user write access.

uint32_t nonsecureUsrR

Allow nonsecure user read access.

uint32_t __pad0__

Reserved.

uint32_t nonsecurePrivX

Allow nonsecure privilege execute access.

uint32_t nonsecurePrivW

Allow nonsecure privilege write access.

uint32_t nonsecurePrivR

Allow nonsecure privilege read access.

uint32_t __pad1__

Reserved.

uint32_t secureUsrX

Allow secure user execute access.

uint32_t secureUsrW

Allow secure user write access.

uint32_t secureUsrR

Allow secure user read access.

uint32_t __pad2__

Reserved.

uint32_t securePrivX

Allownonsecure privilege execute access.

uint32_t securePrivW

Allownonsecure privilege write access.

uint32_t securePrivR

Allownonsecure privilege read access.

uint32_t __pad3__

Reserved.

uint32_t lock

Lock the configuration until next reset, only apply to access control register 0.

struct _trdc_mrc_region_descriptor_config

#include <fsl_trdc.h> The configuration of each region descriptor per domain per MRC instance.

Public Members

uint8_t memoryAccessControlSelect

Select one of the 8 access control policies for this region, for access control policies see `trdc_memory_access_control_config_t`.

uint32_t startAddr

Physical start address.

bool valid

Lock the register.

bool nseEnable

Enable non-secure accesses and disable secure accesses.

uint32_t endAddr

Physical start address.

uint8_t mrcIdx

The index of the MRC for this configuration to take effect.

uint8_t domainIdx

The index of the domain for this configuration to take effect.

uint8_t regionIdx

The index of the region for this configuration to take effect.

struct `_trdc_mbc_nse_update_config`

#include <fsl_trdc.h> The configuration of MBC NSE update.

Public Members

uint32_t __pad0__

Reserved.

uint32_t wordIdx

MBC configuration word index to be updated.

uint32_t __pad1__

Reserved.

uint32_t memorySelect

Bit mask of the selected memory to be updated. `_trdc_MBC_memory`.

uint32_t __pad2__

Reserved.

uint32_t domainSelect

Bit mask of the selected domain to be updated. `_trdc_MBC_domain`.

uint32_t __pad3__

Reserved.

uint32_t autoIncrement

Whether to increment the word index after current word is updated using this configuration.

struct `_trdc_mbc_memory_block_config`

#include <fsl_trdc.h> The configuration of each memory block per domain per MBC instance.

Public Members

`uint32_t` memoryAccessControlSelect

Select one of the 8 access control policies for this memory block, for access control policies see `trdc_memory_access_control_config_t`.

`uint32_t` nseEnable

Enable non-secure accesses and disable secure accesses.

`uint32_t` mbcIdx

The index of the MBC for this configuration to take effect.

`uint32_t` domainIdx

The index of the domain for this configuration to take effect.

`uint32_t` slaveMemoryIdx

The index of the slave memory for this configuration to take effect.

`uint32_t` memoryBlockIdx

The index of the memory block for this configuration to take effect.

2.71 TRGMUX: Trigger Mux Driver

`static inline void TRGMUX_LockRegister(TRGMUX_Type *base, uint32_t index)`

Sets the flag of the register which is used to mark writeable.

The function sets the flag of the register which is used to mark writeable. Example:

```
TRGMUX_LockRegister(TRGMUX0, kTRGMUX_Trgmux0Dmamux0);
```

Parameters

- `base` – TRGMUX peripheral base address.
- `index` – The index of the TRGMUX register, see the enum `trgmux_device_t` defined in `<SOC>.h`.

`status_t TRGMUX_SetTriggerSource(TRGMUX_Type *base, uint32_t index, trgmux_trigger_input_t input, uint32_t trigger_src)`

Configures the trigger source of the appointed peripheral.

The function configures the trigger source of the appointed peripheral. Example:

```
TRGMUX_SetTriggerSource(TRGMUX0, kTRGMUX_Trgmux0Dmamux0, kTRGMUX_TriggerInput0,  
↪ kTRGMUX_SourcePortPin);
```

Parameters

- `base` – TRGMUX peripheral base address.
- `index` – The index of the TRGMUX register, see the enum `trgmux_device_t` defined in `<SOC>.h`.
- `input` – The MUX select for peripheral trigger input
- `trigger_src` – The trigger inputs for various peripherals. See the enum `trgmux_source_t` defined in `<SOC>.h`.

Return values

- `kStatus_Success` – Configured successfully.

- `kStatus_TRGMUX_Locked` – Configuration failed because the register is locked.

`FSL_TRGMUX_DRIVER_VERSION`

TRGMUX driver version.

TRGMUX configure status.

Values:

enumerator `kStatus_TRGMUX_Locked`
Configure failed for register is locked

enum `_trgmux_trigger_input`

Defines the MUX select for peripheral trigger input.

Values:

enumerator `kTRGMUX_TriggerInput0`
The MUX select for peripheral trigger input 0

enumerator `kTRGMUX_TriggerInput1`
The MUX select for peripheral trigger input 1

enumerator `kTRGMUX_TriggerInput2`
The MUX select for peripheral trigger input 2

enumerator `kTRGMUX_TriggerInput3`
The MUX select for peripheral trigger input 3

typedef enum `_trgmux_trigger_input` `trgmux_trigger_input_t`

Defines the MUX select for peripheral trigger input.

2.72 TSTMR: Timestamp Timer Driver

void `TSTMR_Init(TSTMR_Type *base)`

Init TSTMR.

This function initializes the TSTMR module.

Parameters

- `base` – TSTMR peripheral base address.

void `TSTMR_Deinit(TSTMR_Type *base)`

Deinit TSTMR.

This function deinitializes the TSTMR module.

Parameters

- `base` – TSTMR peripheral base address.

`FSL_TSTMR_DRIVER_VERSION`

Version 2.1.0

static inline uint64_t `TSTMR_ReadTimeStamp(TSTMR_Type *base)`

Reads the time stamp.

This function reads the low and high registers and returns the 56-bit free running counter value. This can be read by software at any time to determine the software ticks. TSTMR registers can be read with 32-bit accesses only. The TSTMR LOW read should occur first, followed by the TSTMR HIGH read.

Parameters

- base – TSTMR peripheral base address.

Returns

The 56-bit time stamp value.

void TSTMR_DelayUs(TSTMR_Type *base, uint64_t delayInUs)

Delays for a specified number of microseconds.

This function repeatedly reads the timestamp register and waits for the user-specified delay value.

Parameters

- base – TSTMR peripheral base address.
- delayInUs – Delay value in microseconds.

2.73 VBAT: Smart Power Switch

The enumeration of VBAT module status.

Values:

enumerator kStatus_VBAT_Fro16kNotEnabled

Internal 16kHz free running oscillator not enabled.

enumerator kStatus_VBAT_BandgapNotEnabled

Bandgap not enabled.

enum _vbat_status_flag

The enumeration of VBAT status flags.

Values:

enumerator kVBAT_StatusFlagPORDetect

VBAT domain has been reset

enumerator kVBAT_StatusFlagWakeupPin

A falling edge is detected on the wakeup pin.

enumerator kVBAT_StatusFlagBandgapTimer0

Bandgap Timer0 period reached.

enumerator kVBAT_StatusFlagBandgapTimer1

Bandgap Timer1 period reached.

enumerator kVBAT_StatusFlagLdoReady

LDO is enabled and ready.

enum _vbat_interrupt_enable

The enumeration of VBAT interrupt enable.

Values:

enumerator kVBAT_InterruptEnablePORDetect

Enable POR detect interrupt.

enumerator kVBAT_InterruptEnableWakeupPin

Enable the interrupt when a falling edge is detected on the wakeup pin.

enumerator kVBAT_InterruptEnableBandgapTimer0
Enable the interrupt if Bandgap Timer0 period reached.

enumerator kVBAT_InterruptEnableBandgapTimer1
Enable the interrupt if Bandgap Timer1 period reached.

enumerator kVBAT_InterruptEnableLdoReady
Enable LDO ready interrupt.

enumerator kVBAT_AllInterruptsEnable
Enable all interrupts.

enum _vbat_wakeup_enable
The enumeration of VBAT wakeup enable.

Values:

enumerator kVBAT_WakeupEnablePORDetect
Enable POR detect wakeup.

enumerator kVBAT_WakeupEnableWakeupPin
Enable wakeup feature when a falling edge is detected on the wakeup pin.

enumerator kVBAT_WakeupEnableBandgapTimer0
Enable wakeup feature when bandgap timer0 period reached.

enumerator kVBAT_WakeupEnableBandgapTimer1
Enable wakeup feature when bandgap timer1 period reached.

enumerator kVBAT_WakeupEnableLdoReady
Enable wakeup when LDO ready.

enumerator kVBAT_AllWakeupsEnable
Enable all wakeup.

enum _vbat_bandgap_timer_id
The enumeration of bandgap timer id, VBAT support two bandgap timers.

Values:

enumerator kVBAT_BandgapTimer0
Bandgap Timer0.

enumerator kVBAT_BandgapTimer1
Bandgap Timer1.

enum _vbat_bandgap_refresh_period
The enumeration of bandgap refresh period.

Values:

enumerator kVBAT_BandgapRefresh7P8125ms
Bandgap refresh every 7.8125ms.

enumerator kVBAT_BandgapRefresh15P625ms
Bandgap refresh every 15.625ms.

enumerator kVBAT_BandgapRefresh31P25ms
Bandgap refresh every 31.25ms.

enumerator kVBAT_BandgapRefresh62P5ms
Bandgap refresh every 62.5ms.

enum `_vbat_bandgap_timer_timeout_period`

The enumeration of bandgap timer timeout period.

Values:

enumerator `kVBAT_BangapTimerTimeout1s`

Bandgap timer timerout every 1s.

enumerator `kVBAT_BangapTimerTimeout500ms`

Bandgap timer timerout every 500ms.

enumerator `kVBAT_BangapTimerTimeout250ms`

Bandgap timer timerout every 250ms.

enumerator `kVBAT_BangapTimerTimeout125ms`

Bandgap timer timerout every 125ms.

enumerator `kVBAT_BangapTimerTimeout62P5ms`

Bandgap timer timerout every 62.5ms.

enumerator `kVBAT_BangapTimerTimeout31P25ms`

Bandgap timer timerout every 31.25ms.

typedef enum `_vbat_bandgap_refresh_period` `vbat_bandgap_refresh_period_t`

The enumeration of bandgap refresh period.

typedef enum `_vbat_bandgap_timer_timeout_period` `vbat_bandgap_timer_timeout_period_t`

The enumeration of bandgap timer timeout period.

typedef struct `_vbat_fro16k_config` `vbat_fro16k_config_t`

The structure of internal 16kHz free running oscillator attributes.

`VBAT_LDO_READY_TIMEOUT`

Max loops to wait for LDO ready.

When configuring the LDO, driver will wait for LDO ready. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

void `VBAT_ConfigFRO16k(VBAT_Type *base, const vbat_fro16k_config_t *config)`

Configure internal 16kHz free running oscillator, including enable FRO16k, gate FRO16k output.

Parameters

- `base` – VBAT peripheral base address.
- `config` – Pointer to `vbat_fro16k_config_t` structure.

static inline void `VBAT_EnableFRO16k(VBAT_Type *base, bool enable)`

Enable/disable internal 16kHz free running oscillator.

Parameters

- `base` – VBAT peripheral base address.
- `enable` – Used to enable/disable 16kHz FRO.
 - **true** Enable internal 16kHz free running oscillator.
 - **false** Disable internal 16kHz free running oscillator.

static inline bool `VBAT_CheckFRO16kEnabled(VBAT_Type *base)`

Check if internal 16kHz free running oscillator is enabled.

Parameters

- `base` – VBAT peripheral base address.

Return values

- `true` – The internal 16kHz Free running oscillator is enabled.
- `false` – The internal 16kHz Free running oscillator is disabled.

`static inline void VBAT_UngateFRO16k(VBAT_Type *base, bool unGateFRO16k)`
Ungate/gate FRO 16kHz output clock to other modules.

Parameters

- `base` – VBAT peripheral base address.
- `unGateFRO16k` – Used to gate/ungate FRO 16kHz output.
 - **true** FRO 16kHz output clock to other modules is enabled.
 - **false** FRO 16kHz output clock to other modules is disabled.

`static inline void VBAT_LockFRO16kSettings(VBAT_Type *base)`

Lock settings of internal 16kHz free running oscillator, please note that if locked 16kHz FRO's settings can not be updated until the next POR.

Note: Please note that the operation to ungate/gate FRO 16kHz output clock can not be locked by this function.

Parameters

- `base` – VBAT peripheral base address.

`status_t VBAT_EnableBandgap(VBAT_Type *base, bool enable)`
Enable/disable Bandgap.

Note: The FRO16K must be enabled before enabling the bandgap.

Note: This setting can be locked by `VBAT_LockLdoRamSettings()` function.

Parameters

- `base` – VBAT peripheral base address.
- `enable` – Used to enable/disable bandgap.
 - **true** Enable the bandgap.
 - **false** Disable the bandgap.

Return values

- `kStatus_Success` – Success to enable/disable the bandgap.
- `kStatus_VBAT_Fro16kNotEnabled` – Fail to enable the bandgap due to FRO16k is not enabled previously.

`static inline bool VBAT_CheckBandgapEnabled(VBAT_Type *base)`
Check if bandgap is enabled.

Parameters

- `base` – VBAT peripheral base address.

Return values

- true – The bandgap is enabled.
- false – The bandgap is disabled.

```
static inline void VBAT_EnableBandgapRefreshMode(VBAT_Type *base, bool
                                                enableRefreshMode)
```

Enable/disable bandgap low power refresh mode.

Note: This setting can be locked by VBAT_LockLdoRamSettings() function.

Parameters

- base – VBAT peripheral base address.
- enableRefreshMode – Used to enable/disable bandgap low power refresh mode.
 - **true** Enable bandgap low power refresh mode.
 - **false** Disable bandgap low power refresh mode.

```
status_t VBAT_EnableBackupSRAMRegulator(VBAT_Type *base, bool enable)
Enable/disable Backup RAM Regulator(RAM_LDO).
```

Note: This setting can be locked by VBAT_LockLdoRamSettings() function.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable RAM_LDO.
 - **true** Enable backup SRAM regulator.
 - **false** Disable backup SRAM regulator.

Return values

- kStatusSuccess – Success to enable/disable backup SRAM regulator.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable backup SRAM regulator due to FRO16k is not enabled previously.
- kStatus_VBAT_BandgapNotEnabled – Fail to enable backup SRAM regulator due to the bandgap is not enabled previously.
- kStatus_Timeout – Timeout occurs while waiting completion.

```
static inline void VBAT_LockLdoRamSettings(VBAT_Type *base)
```

Lock settings of RAM_LDO, please note that if locked then RAM_LDO's settings can not be updated until the next POR.

Parameters

- base – VBAT peripheral base address.

```
status_t VBAT_SwitchSRAMPowerByVBAT(VBAT_Type *base)
```

Switch the SRAM to be powered by VBAT in software mode.

Note: This function can be used to switch the SRAM to the VBAT retention supply at any time, but please note that the SRAM must not be accessed during this time and software must manually invoke VBAT_SwitchSRAMPowerBySocSupply() before accessing the SRAM again.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatusSuccess – Success to Switch SRAM powered by VBAT.
- kStatus_VBAT_Fro16kNotEnabled – Fail to switch SRAM powered by VBAT due to FRO16K not enabled previously.

```
static inline void VBAT_SwitchSRAMPowerBySocSupply(VBAT_Type *base)
```

Switch the RAM to be powered by Soc Supply in software mode.

Parameters

- base – VBAT peripheral base address.

```
static inline void VBAT_EnableSRAMArrayRetained(VBAT_Type *base, bool enable)
```

Enable/disable SRAM array remains powered from Soc power, when LDO_RAM is disabled.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable SRAM array power retained.
 - **true** SRAM array is retained when powered from VDD_CORE.
 - **false** SRAM array is not retained when powered from VDD_CORE.

```
static inline void VBAT_EnableSRAMIsolation(VBAT_Type *base, bool enable)
```

Enable/disable SRAM isolation.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable SRAM violation.
 - **true** SRAM will be isolated.
 - **false** SRAM state follows the SoC power modes.

```
status_t VBAT_EnableBandgapTimer(VBAT_Type *base, bool enable, uint8_t timerIdMask)
```

Enable/disable Bandgap timer.

Note: The bandgap timer is available when the bandgap is enabled and are clocked by the FRO16k.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap timer.
- timerIdMask – The mask of bandgap timer Id, should be the OR'ed value of vbat_bandgap_timer_id_t.

Return values

- kStatus_Success – Success to enable/disable selected bandgap timer.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable/disable selected bandgap timer due to FRO16k not enabled previously.
- kStatus_VBAT_BandgapNotEnabled – Fail to enable/disable selected bandgap timer due to bandgap not enabled previously.

```
void VBAT_SetBandgapTimerTimeoutValue(VBAT_Type *base,  
                                       vbat_bandgap_timer_timeout_period_t  
                                       timeoutPeriod, uint8_t timerIdMask)
```

Set bandgap timer timeout value.

Parameters

- base – VBAT peripheral base address.
- timeoutPeriod – Bandgap timer timeout value, please refer to `vbat_bandgap_timer_timeout_period_t`.
- timerIdMask – The mask of bandgap timer Id, should be the OR'ed value of `vbat_bandgap_timer_id_t`.

```
static inline uint32_t VBAT_GetStatusFlags(VBAT_Type *base)
```

Get VBAT status flags.

Parameters

- base – VBAT peripheral base address.

Returns

The asserted status flags, should be the OR'ed value of `vbat_status_flag_t`.

```
static inline void VBAT_ClearStatusFlags(VBAT_Type *base, uint32_t mask)
```

Clear VBAT status flags.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of status flags to be cleared, should be the OR'ed value of `vbat_status_flag_t` except `kVBAT_StatusFlagLdoReady`.

```
static inline void VBAT_EnableInterrupts(VBAT_Type *base, uint32_t mask)
```

Enable interrupts for the VBAT module, such as POR detect interrupt, Wakeup Pin interrupt and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be enabled, should be the OR'ed value of `vbat_interrupt_enable_t`.

```
static inline void VBAT_DisableInterrupts(VBAT_Type *base, uint32_t mask)
```

Disable interrupts for the VBAT module, such as POR detect interrupt, wakeup pin interrupt and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be disabled, should be the OR'ed value of `vbat_interrupt_enable_t`.

```
static inline void VBAT_EnableWakeup(VBAT_Type *base, uint32_t mask)
```

Enable wakeup for the VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of enumerators in `vbat_wakeup_enable_t`.

```
static inline void VBAT_DisableWakeup(VBAT_Type *base, uint32_t mask)
```

Disable wakeup for VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of enumerators in `vbat_wakeup_enable_t`.

```
static inline void VBAT_LockInterruptWakeupSettings(VBAT_Type *base)
```

Lock VBAT interrupt and wakeup settings, please note that if locked the interrupt and wakeup settings can not be updated until the next POR.

Parameters

- base – VBAT peripheral base address.

```
FSL_VBAT_DRIVER_VERSION
```

VBAT driver version 2.1.1.

```
struct _vbat_fro16k_config
```

#include <fsl_vbat.h> The structure of internal 16kHz free running oscillator attributes.

Public Members

```
bool enableFRO16k
```

Enable/disable internal 16kHz free running oscillator.

```
bool enableFRO16kOutput
```

Enable/disable FRO 16k output clock to other modules.

2.74 VREF: Voltage Reference Driver

```
void VREF_Init(VREF_Type *base, const vref_config_t *config)
```

Enables the clock gate and configures the VREF module according to the configuration structure.

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up `vref_config_t` parameters and how to call the `VREF_Init` function by passing in these parameters.

```
vref_config_t vrefConfig;
VREF_GetDefaultConfig(VREF, &vrefConfig);
vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
VREF_Init(VREF, &vrefConfig);
```

Parameters

- base – VREF peripheral address.
- config – Pointer to the configuration structure.

```
void VREF_Deinit(VREF_Type *base)
```

Stops and disables the clock for the VREF module.

This function should be called to shut down the module. This is an example.

```
vref_config_t vrefUserConfig;
VREF_GetDefaultConfig(VREF, &vrefUserConfig);
VREF_Init(VREF, &vrefUserConfig);
...
VREF_Deinit(VREF);
```

Parameters

- base – VREF peripheral address.

void VREF_GetDefaultConfig(*vref_config_t* *config)

Initializes the VREF configuration structure.

This function initializes the VREF configuration structure to default values. This is an example.

```
config->bufferMode = kVREF_ModeHighPowerBuffer;
config->enableInternalVoltageRegulator = true;
config->enableChopOscillator = true;
config->enableHCBandgap = true;
config->enableCurvatureCompensation = true;
config->enableLowPowerBuff = true;
```

Parameters

- config – Pointer to the initialization structure.

void VREF_SetVrefTrimVal(VREF_Type *base, uint8_t trimValue)

Sets a TRIM value for the accurate 1.0V bandgap output.

This function sets a TRIM value for the reference voltage. It will trim the accurate 1.0V bandgap by 0.5mV each step.

Parameters

- base – VREF peripheral address.
- trimValue – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

void VREF_SetTrim21Val(VREF_Type *base, uint8_t trim21Value)

Sets a TRIM value for the accurate buffered VREF output.

This function sets a TRIM value for the reference voltage. If buffer mode be set to other values (Buf21 enabled), it will trim the VREF_OUT by 0.1V each step from 1.0V to 2.1V.

Note: When Buf21 is enabled, the value of UTRIM[TRIM2V1] should be ranged from 0b0000 to 0b1011 in order to trim the output voltage from 1.0V to 2.1V, other values will make the VREF_OUT to default value, 1.0V.

Parameters

- base – VREF peripheral address.
- trim21Value – Value of the trim register to set the output reference voltage (maximum 0xF (4-bit)).

uint8_t VREF_GetVrefTrimVal(VREF_Type *base)

Reads the trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[VREFTRIM] (13:8)

Parameters

- base – VREF peripheral address.

Returns

6-bit value of trim setting.

```
uint8_t VREF_GetTrim21Val(VREF_Type *base)
```

Reads the VREF 2.1V trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[TRIM2V1] (3:0),

Parameters

- base – VREF peripheral address.

Returns

4-bit value of trim setting.

```
FSL_VREF_DRIVER_VERSION
```

Version 2.4.0.

```
enum _vref_buffer_mode
```

VREF buffer modes.

Values:

```
enumerator kVREF_ModeBandgapOnly
```

Bandgap enabled/standby.

```
enumerator kVREF_ModeLowPowerBuffer
```

Low-power buffer mode enabled

```
enumerator kVREF_ModeHighPowerBuffer
```

High-power buffer mode enabled

```
typedef enum _vref_buffer_mode vref_buffer_mode_t
```

VREF buffer modes.

```
typedef struct _vref_config vref_config_t
```

The description structure for the VREF module.

```
struct _vref_config
```

#include <fsl_vref.h> The description structure for the VREF module.

Public Members

```
vref_buffer_mode_t bufferMode
```

Buffer mode selection

```
bool enableInternalVoltageRegulator
```

Provide additional supply noise rejection.

```
bool enableChopOscillator
```

Enable Chop oscillator.

```
bool enableHCBandgap
```

Enable High-Accurate bandgap.

```
bool enableCurvatureCompensation
```

Enable second order curvature compensation.

```
bool enableLowPowerBuff
```

Provides bias current for other peripherals.

2.75 WDOG32: 32-bit Watchdog Timer

`void WDOG32_GetDefaultConfig(wdog32_config_t *config)`

Initializes the WDOG32 configuration structure.

This function initializes the WDOG32 configuration structure to default values. The default values are:

```
wdog32Config->enableWdog32 = true;
wdog32Config->clockSource = kWDOG32_ClockSource1;
wdog32Config->prescaler = kWDOG32_ClockPrescalerDivide1;
wdog32Config->workMode.enableWait = true;
wdog32Config->workMode.enableStop = false;
wdog32Config->workMode.enableDebug = false;
wdog32Config->testMode = kWDOG32_TestModeDisabled;
wdog32Config->enableUpdate = true;
wdog32Config->enableInterrupt = false;
wdog32Config->enableWindowMode = false;
wdog32Config->windowValue = 0U;
wdog32Config->timeoutValue = 0xFFFFU;
```

See also:

`wdog32_config_t`

Parameters

- `config` – Pointer to the WDOG32 configuration structure.

`AT_QUICKACCESS_SECTION_CODE (status_t WDOG32_Init(WDOG_Type *base, const wdog32_config_t *config))`

Initializes the WDOG32 module.

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, `enableUpdate` must be set to true in the configuration.

Example:

```
wdog32_config_t config;
WDOG32_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
WDOG32_Init(wdog_base,&config);
```

Note: If there is errata ERR010536 (FSL_FEATURE_WDOG_HAS_ERRATA_010536 defined as 1), then after calling this function, user need delay at least 4 LPO clock cycles before accessing other WDOG32 registers.

Parameters

- `base` – WDOG32 peripheral base address.
- `config` – The configuration of the WDOG32.

Return values

- `kStatus_Success` – The initialization was successful
- `kStatus_Timeout` – The initialization timed out

`status_t WDOG32_Deinit(WDOG_Type *base)`

De-initializes the WDOG32 module.

This function shuts down the WDOG32. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

- `base` – WDOG32 peripheral base address.

Return values

- `kStatus_Success` – The de-initialization was successful
- `kStatus_Timeout` – The de-initialization timed out

`AT_QUICKACCESS_SECTION_CODE (status_t WDOG32_Unlock(WDOG_Type *base))`

Unlocks the WDOG32 register written.

This function unlocks the WDOG32 register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

- `base` – WDOG32 peripheral base address

Return values

- `kStatus_Success` – The unlock sequence was successful
- `kStatus_Timeout` – The unlock sequence timed out

`AT_QUICKACCESS_SECTION_CODE (void WDOG32_Enable(WDOG_Type *base))`

Enables the WDOG32 module.

Disables the WDOG32 module.

This function writes a value into the WDOG_CS register to enable the WDOG32. The WDOG_CS register is a write-once register. Please check the `enableUpdate` is set to true for calling `WDOG32_Init` to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

This function writes a value into the WDOG_CS register to disable the WDOG32. The WDOG_CS register is a write-once register. Please check the `enableUpdate` is set to true for calling `WDOG32_Init` to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- `base` – WDOG32 peripheral base address.
- `base` – WDOG32 peripheral base address

`AT_QUICKACCESS_SECTION_CODE (void WDOG32_EnableInterrupts(WDOG_Type *base, uint32_t mask))`

Enables the WDOG32 interrupt.

Clears the WDOG32 flag.

Disables the WDOG32 interrupt.

This function writes a value into the WDOG_CS register to enable the WDOG32 interrupt. The WDOG_CS register is a write-once register. Please check the `enableUpdate` is set to true

for calling `WDOG32_Init` to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Example to clear an interrupt flag:

```
WDOG32_ClearStatusFlags(wdog_base, kWDOG32_InterruptFlag);
```

Parameters

- `base` – WDOG32 peripheral base address.
- `mask` – The interrupts to enable. The parameter can be a combination of the following source if defined:
 - `kWDOG32_InterruptEnable`

This function writes a value into the `WDOG_CS` register to disable the WDOG32 interrupt. The `WDOG_CS` register is a write-once register. Please check the `enableUpdate` is set to true for calling `WDOG32_Init` to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

- `base` – WDOG32 peripheral base address.
- `mask` – The interrupts to disabled. The parameter can be a combination of the following source if defined:
 - `kWDOG32_InterruptEnable`

This function clears the WDOG32 status flag.

- `base` – WDOG32 peripheral base address.
- `mask` – The status flags to clear. The parameter can be any combination of the following values:
 - `kWDOG32_InterruptFlag`

```
static inline uint32_t WDOG32_GetStatusFlags(WDOG_Type *base)
```

Gets the WDOG32 all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;  
status = WDOG32_GetStatusFlags(wdog_base) & kWDOG32_RunningFlag;
```

See also:

`_wdog32_status_flags_t`

- true: related status flag has been set.
- false: related status flag is not set.

Parameters

- `base` – WDOG32 peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

`AT_QUICKACCESS_SECTION_CODE` (void WDOG32_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount))

Sets the WDOG32 timeout value.

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. To ensure the reconfiguration fits the timing of WCT, unlock function will be called inline.

Parameters

- base – WDOG32 peripheral base address
- timeoutCount – WDOG32 timeout value, count of WDOG32 clock ticks.

`AT_QUICKACCESS_SECTION_CODE` (void WDOG32_SetWindowValue(WDOG_Type *base, uint16_t windowValue))

Sets the WDOG32 window value.

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. Please check the enableUpdate is set to true for calling WDOG32_Init to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.
- windowValue – WDOG32 window value.

static inline void WDOG32_Refresh(WDOG_Type *base)

Refreshes the WDOG32 timer.

This function feeds the WDOG32. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG32 peripheral base address

static inline uint16_t WDOG32_GetCounterValue(WDOG_Type *base)

Gets the WDOG32 counter value.

This function gets the WDOG32 counter value.

Parameters

- base – WDOG32 peripheral base address.

Returns

Current WDOG32 counter value.

WDOG_FIRST_WORD_OF_UNLOCK

First word of unlock sequence

WDOG_SECOND_WORD_OF_UNLOCK

Second word of unlock sequence

WDOG_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WDOG_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

FSL_WDOG32_DRIVER_VERSION

WDOG32 driver version.

enum `_wdog32_clock_source`

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

Values:

enumerator `kWDOG32_ClockSource0`

Clock source 0

enumerator `kWDOG32_ClockSource1`

Clock source 1

enumerator `kWDOG32_ClockSource2`

Clock source 2

enumerator `kWDOG32_ClockSource3`

Clock source 3

enum `_wdog32_clock_prescaler`

Describes the selection of the clock prescaler.

Values:

enumerator `kWDOG32_ClockPrescalerDivide1`

Divided by 1

enumerator `kWDOG32_ClockPrescalerDivide256`

Divided by 256

enum `_wdog32_test_mode`

Describes WDOG32 test mode.

Values:

enumerator `kWDOG32_TestModeDisabled`

Test Mode disabled

enumerator `kWDOG32_UserModeEnabled`

User Mode enabled

enumerator `kWDOG32_LowByteTest`

Test Mode enabled, only low byte is used

enumerator `kWDOG32_HighByteTest`

Test Mode enabled, only high byte is used

enum `_wdog32_interrupt_enable_t`

WDOG32 interrupt configuration structure.

This structure contains the settings for all of the WDOG32 interrupt configurations.

Values:

enumerator `kWDOG32_InterruptEnable`

Interrupt is generated before forcing a reset

enum `_wdog32_status_flags_t`

WDOG32 status flags.

This structure contains the WDOG32 status flags for use in the WDOG32 functions.

Values:

enumerator `kWDOG32_RunningFlag`

Running flag, set when WDOG32 is enabled

enumerator `kWDOG32_InterruptFlag`

Interrupt flag, set when interrupt occurs

typedef enum `_wdog32_clock_source` `wdog32_clock_source_t`

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

typedef enum `_wdog32_clock_prescaler` `wdog32_clock_prescaler_t`

Describes the selection of the clock prescaler.

typedef struct `_wdog32_work_mode` `wdog32_work_mode_t`

Defines WDOG32 work mode.

typedef enum `_wdog32_test_mode` `wdog32_test_mode_t`

Describes WDOG32 test mode.

typedef struct `_wdog32_config` `wdog32_config_t`

Describes WDOG32 configuration structure.

struct `_wdog32_work_mode`

#include `<fsl_wdog32.h>` Defines WDOG32 work mode.

Public Members

bool `enableWait`

Enables or disables WDOG32 in wait mode

bool `enableStop`

Enables or disables WDOG32 in stop mode

bool `enableDebug`

Enables or disables WDOG32 in debug mode

struct `_wdog32_config`

#include `<fsl_wdog32.h>` Describes WDOG32 configuration structure.

Public Members

bool `enableWdog32`

Enables or disables WDOG32

`wdog32_clock_source_t` `clockSource`

Clock source select

wdog32_clock_prescaler_t prescaler

Clock prescaler value

wdog32_work_mode_t workMode

Configures WDOG32 work mode in debug stop and wait mode

wdog32_test_mode_t testMode

Configures WDOG32 test mode

bool enableUpdate

Update write-once register enable

bool enableInterrupt

Enables or disables WDOG32 interrupt

bool enableWindowMode

Enables or disables WDOG32 window mode

uint16_t windowValue

Window value

uint16_t timeoutValue

Timeout value

2.76 WUU: Wakeup Unit driver

```
void WUU_SetExternalWakeUpPinsConfig(WUU_Type *base, uint8_t pinIndex, const  
                                     wuu_external_wakeup_pin_config_t *config)
```

Enables and Configs External WakeUp Pins.

This function enables/disables the external pin as wakeup input. What's more this function configs pins options, including edge detection wakeup event and operate mode.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin. See Reference Manual for the details.
- config – Pointer to `wuu_external_wakeup_pin_config_t` structure.

```
void WUU_ClearExternalWakeupPinsConfig(WUU_Type *base, uint8_t pinIndex)
```

Disable and clear external wakeup pin settings.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin.

```
static inline uint32_t WUU_GetExternalWakeUpPinsFlag(WUU_Type *base)
```

Gets External Wakeup pin flags.

This function return the external wakeup pin flags.

Parameters

- base – WUU peripheral base address.

Returns

Wakeup flags for all external wakeup pins.

```
static inline void WUU_ClearExternalWakeUpPinsFlag(WUU_Type *base, uint32_t mask)
```

Clears External WakeUp Pin flags.

This function clears external wakeup pins flags based on the mask.

Parameters

- `base` – WUU peripheral base address.
- `mask` – The mask of Wakeup pin index to be cleared.

```
void WUU_SetInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                         wuu_internal_wakeup_module_event_t event)
```

Config Internal modules' event as the wake up sources.

This function configs the internal modules event as the wake up sources.

Parameters

- `base` – WUU peripheral base address.
- `moduleIndex` – The selected internal module. See the Reference Manual for the details.
- `event` – Select interrupt or DMA/Trigger of the internal module as the wake up source.

```
void WUU_ClearInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                           wuu_internal_wakeup_module_event_t event)
```

Disable an on-chip internal modules' event as the wakeup sources.

Parameters

- `base` – WUU peripheral base address.
- `moduleIndex` – The selected internal module. See the Reference Manual for the details.
- `event` – The event(interrupt or DMA/trigger) of the internal module to disable.

```
void WUU_SetPinFilterConfig(WUU_Type *base, uint8_t filterIndex, const  
                            wuu_pin_filter_config_t *config)
```

Configs and Enables Pin filters.

This function configs Pin filter, including pin select, filter operate mode filter wakeup event and filter edge detection.

Parameters

- `base` – WUU peripheral base address.
- `filterIndex` – The index of the pin filter.
- `config` – Pointer to `wuu_pin_filter_config_t` structure.

```
bool WUU_GetPinFilterFlag(WUU_Type *base, uint8_t filterIndex)
```

Gets the pin filter configuration.

This function gets the pin filter flag.

Parameters

- `base` – WUU peripheral base address.
- `filterIndex` – A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

void WUU_ClearPinFilterFlag(WUU_Type *base, uint8_t filterIndex)

Clears the pin filter configuration.

This function clears the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

bool WUU_GetExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0. return True if the specific pin is a wakeup source.

void WUU_ClearExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0.

FSL_WUU_DRIVER_VERSION

Defines WUU driver version 2.4.0.

enum _wuu_external_pin_edge_detection

External WakeUp pin edge detection enumeration.

Values:

enumerator kWUU_ExternalPinDisable

External input Pin disabled as wake up input.

enumerator kWUU_ExternalPinRisingEdge

External input Pin enabled with the rising edge detection.

enumerator kWUU_ExternalPinFallingEdge

External input Pin enabled with the falling edge detection.

enumerator kWUU_ExternalPinAnyEdge

External input Pin enabled with any change detection.

enum _wuu_external_wakeup_pin_event

External input wake up pin event enumeration.

Values:

enumerator kWUU_ExternalPinInterrupt

External input Pin configured as interrupt.

enumerator kWUU_ExternalPinDMARequest

External input Pin configured as DMA request.

enumerator kWUU_ExternalPinTriggerEvent

External input Pin configured as Trigger event.

enum _wuu_external_wakeup_pin_mode

External input wake up pin mode enumeration.

Values:

enumerator kWUU_ExternalPinActiveDSPD
External input Pin is active only during Deep Sleep/Power Down Mode.

enumerator kWUU_ExternalPinActiveAlways
External input Pin is active during all power modes.

enum _wuu_internal_wakeup_module_event
Internal module wake up event enumeration.

Values:

enumerator kWUU_InternalModuleInterrupt
Internal modules' interrupt as a wakeup source.

enumerator kWUU_InternalModuleDMATrigger
Internal modules' DMA/Trigger as a wakeup source.

enum _wuu_filter_edge
Pin filter edge enumeration.

Values:

enumerator kWUU_FilterDisabled
Filter disabled.

enumerator kWUU_FilterPosedgeEnable
Filter posedge detect enabled.

enumerator kWUU_FilterNegedgeEnable
Filter negedge detect enabled.

enumerator kWUU_FilterAnyEdge
Filter any edge detect enabled.

enum _wuu_filter_event
Pin Filter event enumeration.

Values:

enumerator kWUU_FilterInterrupt
Filter output configured as interrupt.

enumerator kWUU_FilterDMARequest
Filter output configured as DMA request.

enumerator kWUU_FilterTriggerEvent
Filter output configured as Trigger event.

enum _wuu_filter_mode
Pin filter mode enumeration.

Values:

enumerator kWUU_FilterActiveDSPD
External input pin filter is active only during Deep Sleep/Power Down Mode.

enumerator kWUU_FilterActiveAlways
External input Pin filter is active during all power modes.

typedef enum _wuu_external_pin_edge_detection wuu_external_pin_edge_detection_t
External WakeUp pin edge detection enumeration.

typedef enum _wuu_external_wakeup_pin_event wuu_external_wakeup_pin_event_t
External input wake up pin event enumeration.

```
typedef enum _wuu_external_wakeup_pin_mode wuu_external_wakeup_pin_mode_t
    External input wake up pin mode enumeration.
typedef enum _wuu_internal_wakeup_module_event wuu_internal_wakeup_module_event_t
    Internal module wake up event enumeration.
typedef enum _wuu_filter_edge wuu_filter_edge_t
    Pin filter edge enumeration.
typedef enum _wuu_filter_event wuu_filter_event_t
    Pin Filter event enumeration.
typedef enum _wuu_filter_mode wuu_filter_mode_t
    Pin filter mode enumeration.
typedef struct _wuu_external_wakeup_pin_config wuu_external_wakeup_pin_config_t
    External WakeUp pin configuration.
typedef struct _wuu_pin_filter_config wuu_pin_filter_config_t
    Pin Filter configuration.
struct _wuu_external_wakeup_pin_config
    #include <fsl_wuu.h> External WakeUp pin configuration.
```

Public Members

```
wuu_external_pin_edge_detection_t edge
    External Input pin edge detection.
wuu_external_wakeup_pin_event_t event
    External Input wakeup Pin event
wuu_external_wakeup_pin_mode_t mode
    External Input wakeup Pin operate mode.
struct _wuu_pin_filter_config
    #include <fsl_wuu.h> Pin Filter configuration.
```

Public Members

```
uint32_t pinIndex
    The index of wakeup pin to be muxxed into filter.
wuu_filter_edge_t edge
    The edge of the pin digital filter.
wuu_filter_event_t event
    The event of the filter output.
wuu_filter_mode_t mode
    The mode of the filter operate.
```

Chapter 3

Middleware

3.1 Motor Control

3.1.1 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_pd_bdm** 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 applications 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_QUEUE_SIZE

```
#define FMSTR_COMM_QUEUE_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_QUEUE_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_QUEUE_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-language 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_UINTn	Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)
FMSTR_TSA_SINTn	Signed integer type of size <i>n</i> bits (n=8,16,32,64)
FMSTR_TSA_FRACn	Fractional number of size <i>n</i> bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(<i>m,n</i>)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(<i>m,n</i>)	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)
FM-STR_TSA_USERTYPE(<i>name</i>)	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 responseDataAddr, FMSTR_SIZE responseDataLen);
```

- 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
<i>FM-STR_ADDR</i> 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.	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.
<i>FM-STR_SIZE</i> It is required that this type is unsigned and at least 16 bits wide integer.	Data type used to hold the memory block size.
<i>FM-STR_BOOL</i> This type is used only in zero/non-zero conditions in the driver code.	Data type used as a general boolean type.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command code.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to create the Application Command data buffer.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command result code.

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.

<i>FM-STR_TSA_TII</i>	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 <i>FM-STR_SIZE</i> .
<i>FM-STR_TSA_TS</i>	Data type used to hold a memory block size, as used in the TSA descriptors. By default, this is defined as <i>FM-STR_SIZE</i> .

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

<i>FM-STR_HPIPE</i>	Pipe handle that identifies the open-pipe object. Generally, this is a pointer to a void type.
<i>FM-STR_PIPE_PC</i>	Integer type required to hold at least 7 bits of data. Generally, this is an unsigned 8-bit or 16-bit type.
<i>FM-STR_PIPE_SI</i>	Integer type required to hold at least 16 bits of data. This is used to store the data buffer sizes.
<i>FM-STR_PPIPEF</i>	Pointer to the pipe handler function. See FM-STR_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.

<i>FMSTR_U8</i>	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.	
<i>FM-STR_U16</i>	Unsigned 16-bit integer.
<i>FM-STR_U32</i>	Unsigned 32-bit integer.
<i>FMSTR_S8</i>	Signed 8-bit integer.
<i>FM-STR_S16</i>	Signed 16-bit integer.
<i>FM-STR_S32</i>	Signed 32-bit integer.
<i>FM-STR_FLOAT</i>	4-byte standard IEEE floating-point type.
<i>FM-STR_FLAGS</i>	Data type forming a union with a structure of flag bit-fields.
<i>FM-STR_SIZES</i>	Data type holding a general size value, at least 8 bits wide.
<i>FM-STR_INDEX</i>	General for-loop index. Must be signed, at least 16 bits wide.
<i>FM-STR_BCHR</i>	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.	
<i>FM-STR_BPTR</i>	A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i>).

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.

3.2 MultiCore

3.2.1 Multicore SDK

Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

Multicore SDK (MCSDK) Release Notes

Overview These are the release notes for the NXP Multicore Software Development Kit (MCSDK) version 25.09.00.

This software package contains components for efficient work with multicore devices as well as for the multiprocessor communication.

What is new

- eRPC [CHANGELOG](#)
- RPMsg-Lite [CHANGELOG](#)
- MCMgr [CHANGELOG](#)
- Supported evaluation boards (multicore examples):
 - LPCXpresso55S69
 - FRDM-K32L3A6
 - MIMXRT1170-EVKB
 - MIMXRT1160-EVK
 - MIMXRT1180-EVK
 - MCX-N5XX-EVK
 - MCX-N9XX-EVK
 - FRDM-MCXN947
 - MIMXRT700-EVK
 - KW47-EVK
 - KW47-LOC
 - FRDM-MCXW72
 - MCX-W72-EVK
 - FRDM-IMXRT1186
- Supported evaluation boards (multiprocessor examples):
 - LPCXpresso55S36
 - FRDM-K22F
 - FRDM-K32L2B
 - MIMXRT685-EVK
 - MIMXRT1170-EVKB
 - MIMXRT1180
 - FRDM-MCXN236
 - FRDM-MCXC242
 - FRDM-MCXC444
 - MCX-N9XX-EVK
 - FRDM-MCXN947
 - MIMXRT700-EVK
 - FRDM-IMXRT1186

Development tools The Multicore SDK (MCSDK) was compiled and tested with development tools referred in: [Development tools](#)

Release contents This table describes the release contents. Not all MCUXpresso SDK packages contain the whole set of these components.

Deliverable	Location
Multicore SDK location <MCSDK_dir>	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call component	<MCSDK_dir>/erpc/
Multicore Manager component	<MCSDK_dir>/mcmgr/
RPMsg-Lite	<MCSDK_dir>/rpmsg_lite/
Multicore demo applications	<MCUXpressoSDK_install_dir>/examples/multicore_examples/
Multiprocessor demo applications	<MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/

Multicore SDK release overview Together, the Multicore SDK (MCSDK) and the MCUXpresso SDK (SDK) form a framework for the development of software for NXP multicore devices. The MCSDK release consists of the following elementary software components for multicore:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR) - included just in SDK for multicore devices
- Remote Processor Messaging - Lite (RPMsg-Lite) - included just in SDK for multicore devices

The MCSDK is also accompanied with documentation and several multicore and multiprocessor demo applications.

Demo applications The multicore demo applications demonstrate the usage of the MCSDK software components on supported multicore development boards.

The following multicore demo applications are located together with other MCUXpresso SDK examples in

the <MCUXpressoSDK_install_dir>/examples/multicore_examples subdirectories.

- erpc_matrix_multiply_mu
- erpc_matrix_multiply_mu_rtos
- erpc_matrix_multiply_rpmsg
- erpc_matrix_multiply_rpmsg_rtos
- erpc_two_way_rpc_rpmsg_rtos
- freertos_message_buffers
- hello_world
- multicore_manager
- rpmsg_lite_pingpong
- rpmsg_lite_pingpong_rtos
- rpmsg_lite_pingpong_dsp
- rpmsg_lite_pingpong_tzm

The eRPC multicore component can be leveraged for inter-processor communication and remote procedure calls between SoCs / development boards.

The following multiprocessor demo applications are located together with other MCUXpresso SDK examples in

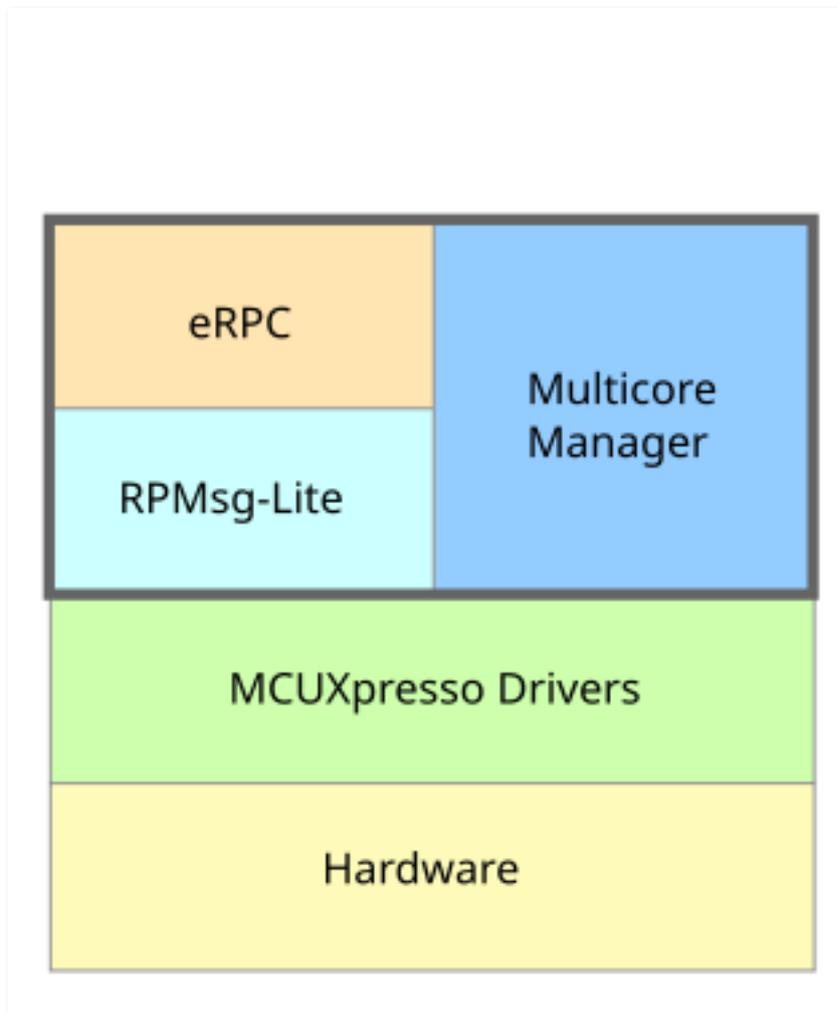
the <MCUXpressoSDK_install_dir>/examples/multiprocessor_examples subdirectories.

- erpc_client_matrix_multiply_spi
- erpc_server_matrix_multiply_spi
- erpc_client_matrix_multiply_uart
- erpc_server_matrix_multiply_uart
- erpc_server_dac_adc
- erpc_remote_control

Getting Started with Multicore SDK (MCSDK)

Overview Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

The following figure highlights the layers and main software components of the MCSDK.

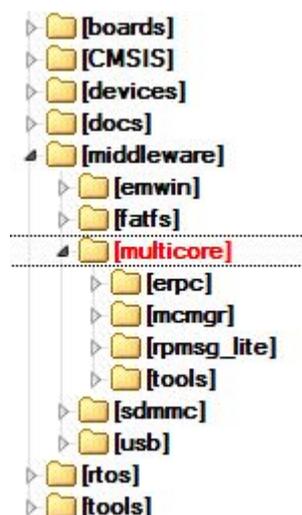


All the MCSDK-related files are located in <MCUXpressoSDK_install_dir>/middleware/multicore folder.

For supported toolchain versions, see the *Multicore SDK v25.09.00 Release Notes* (document MCS-DKRN). For the latest version of this and other MCSDK documents, visit www.nxp.com.

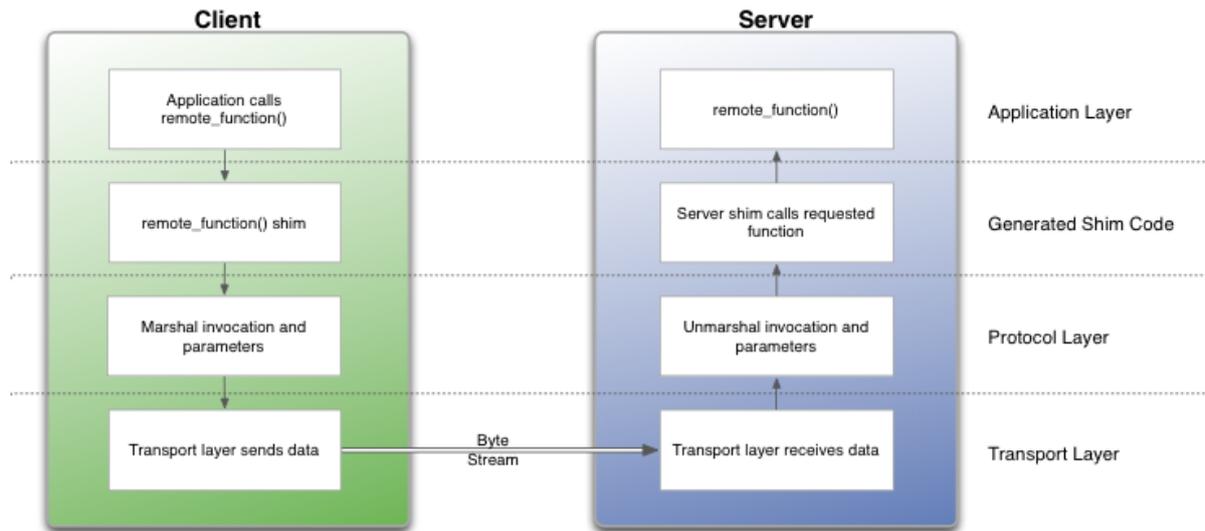
Multicore SDK (MCSDK) components The MCSDK consists of the following software components:

- **Embedded Remote Procedure Call (eRPC):** This component is a combination of a library and code generator tool that implements a transparent function call interface to remote services (running on a different core).
- **Multicore Manager (MCMGR):** This library maintains information about all cores and starts up secondary/auxiliary cores.
- **Remote Processor Messaging - Lite (RPMsg-Lite):** Inter-Processor Communication library.



Embedded Remote Procedure Call (eRPC) The Embedded Remote Procedure Call (eRPC) is the RPC system created by NXP. The RPC is a mechanism used to invoke a software routine on a remote system via a simple local function call.

When a remote function is called by the client, the function's parameters and an identifier for the called routine are marshaled (or serialized) into a stream of bytes. This byte stream is transported to the server through a communications channel (IPC, TPC/IP, UART, and so on). The server unmarshals the parameters, determines which function was invoked, and calls it. If the function returns a value, it is marshaled and sent back to the client.



RPC implementations typically use a combination of a tool (erpcgen) and IDL (interface definition language) file to generate source code to handle the details of marshaling a function's parameters and building the data stream.

Main eRPC features:

- Scalable from BareMetal to Linux OS - configurable memory and threading policies.
- Focus on embedded systems - intrinsic support for C, modular, and lightweight implementation.
- Abstracted transport interface - RPMsg is the primary transport for multicore, UART, or SPI-based solutions can be used for multichip.

The eRPC library is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/erpc folder. For detailed information about the eRPC, see the documentation available in the <MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc folder.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems.

The main MCMGR features:

- Maintains information about all cores in system.
- Secondary/auxiliary cores startup and shutdown.
- Remote core monitoring and event handling.

The MCMGR library is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr folder. For detailed information about the MCMGR library, see the documentation available in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/doc folder.

Remote Processor Messaging Lite (RPMsg-Lite) RPMsg-Lite is a lightweight implementation of the RPMsg protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system. Compared to the legacy OpenAMP implementation, RPMsg-Lite offers a code size reduction, API simplification, and improved modularity.

The main RPMsg protocol features:

- Shared memory interprocessor communication.
- Virtio-based messaging bus.
- Application-defined messages sent between endpoints.

- Portable to different environments/platforms.
- Available in upstream Linux OS.

The RPSMsg-Lite library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpsmsg-lite` folder. For detailed information about the RPSMsg-Lite, see the RPSMsg-Lite User's Guide located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpsmsg_lite/doc` folder.

MCSDK demo applications Multicore and multiprocessor example applications are stored together with other MCUXpresso SDK examples, in the dedicated multicore subfolder.

Location	Folder
Multicore example projects	<code><MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/</code>
Multiprocessor example projects	<code><MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/<application_name>/</code>

See the *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) and *Getting Started with MCUXpresso SDK for XXX Derivatives* documents for more information about the MCUXpresso SDK example folder structure and the location of individual files that form the example application projects. These documents also contain information about building, running, and debugging multicore demo applications in individual supported IDEs. Each example application also contains a readme file that describes the operation of the example and required setup steps.

Inter-Processor Communication (IPC) levels The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Particular ways and levels of IPC are described in this chapter.

IPC using low-level drivers

The NXP multicore SoCs are equipped with peripheral modules dedicated for data exchange between individual cores. They deal with the Mailbox peripheral for LPC parts and the Messaging Unit (MU) peripheral for Kinetis and i.MX parts. The common attribute of both modules is the ability to provide a means of IPC, allowing multiple CPUs to share resources and communicate with each other in a simple manner.

The most lightweight method of IPC uses the MCUXpresso SDK low-level drivers for these peripherals. Using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via the dedicated registers (could be a scalar or a pointer to shared memory) and also to trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific multicore device. The MCUXpresso SDK is accompanied with the RPSMsg-Lite documentation that shows how to use this API in multicore applications.

Messaging mechanism

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to send between multiple endpoints created on each of the CPUs. The RPSMsg-Lite library of the MCSDK provides this ability and serves as the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for messages exchange without the need of a locking mechanism.

The RPSMsg-Lite provides the abstraction layer and can be easily ported to different multicore platforms and environments (Operating Systems). The advantages of such a messaging system are ease of use (there is no need to study behavior of the used underlying hardware) and smooth application code portability between platforms due to unified messaging API.

However, this costs several kB of code and data memory. The MCUXpresso SDK is accompanied by the RPMsg-Lite documentation and several multicore examples. You can also obtain the latest RPMsg-Lite code from the GitHub account github.com/nxp-mcuxpresso/rpmsg-lite.

Remote procedure calls

To facilitate the IPC even more and to allow the remote functions invocation, the remote procedure call mechanism can be implemented. The eRPC of the MCSDK serves for these purposes and allows the ability to invoke a software routine on a remote system via a simple local function call. Utilizing different transport layers, it is possible to communicate between individual cores of multicore SoCs (via RPMsg-Lite) or between separate processors (via SPI, UART, or TCP/IP). The eRPC is mostly applicable to the MPU parts with enough of memory resources like i.MX parts.

The eRPC library allows you to export existing C functions without having to change their prototypes (in most cases). It is accompanied by the code generator tool that generates the shim code for serialization and invocation based on the IDL file with definitions of data types and remote interfaces (API).

If the communicating peer is running as a Linux OS user-space application, the generated code can be either in C/C++ or Python.

Using the eRPC simplifies the access to services implemented on individual cores. This way, the following types of applications running on dedicated cores can be easily interfaced:

- Communication stacks (USB, Thread, Bluetooth Low Energy, Zigbee)
- Sensor aggregation/fusion applications
- Encryption algorithms
- Virtual peripherals

The eRPC is publicly available from the following GitHub account: github.com/EmbeddedRPC/erpc. Also, the MCUXpresso SDK is accompanied by the eRPC code and several multicore and multiprocessor eRPC examples.

The mentioned IPC levels demonstrate the scalability of the Multicore SDK library. Based on application needs, different IPC techniques can be used. It depends on the complexity, required speed, memory resources, system design, and so on. The MCSDK brings users the possibility for quick and easy development of multicore and multiprocessor applications.

Changelog Multicore SDK

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

[25.09.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.14.0
 - eRPC generator (erpcgen) v1.14.0
 - Multicore Manager (MCMgr) v5.0.1
 - RPMsg-Lite v5.2.1

[25.06.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.14.0

- eRPC generator (erpcgen) v1.14.0
- Multicore Manager (MCMgr) v5.0.0
- RMsg-Lite v5.2.0

[25.03.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.7
 - RMsg-Lite v5.1.4

[24.12.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.6
 - RMsg-Lite v5.1.3

[2.16.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.5
 - RMsg-Lite v5.1.2

[2.15.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.12.0
 - eRPC generator (erpcgen) v1.12.0
 - Multicore Manager (MCMgr) v4.1.5
 - RMsg-Lite v5.1.1

[2.14.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.11.0
 - eRPC generator (erpcgen) v1.11.0
 - Multicore Manager (MCMgr) v4.1.4
 - RMsg-Lite v5.1.0

[2.13.0_imxrt1180a0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.10.0
 - eRPC generator (erpcgen) v1.10.0
 - Multicore Manager (MCMgr) v4.1.3
 - RPSmsg-Lite v5.0.0

[2.13.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.10.0
 - eRPC generator (erpcgen) v1.10.0
 - Multicore Manager (MCMgr) v4.1.3
 - RPSmsg-Lite v5.0.0

[2.12.0_imx93]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.1
 - eRPC generator (erpcgen) v1.9.1
 - Multicore Manager (MCMgr) v4.1.2
 - RPSmsg-Lite v4.0.1

[2.12.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.1
 - eRPC generator (erpcgen) v1.9.1
 - Multicore Manager (MCMgr) v4.1.2
 - RPSmsg-Lite v4.0.0

[2.11.1]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.0
 - eRPC generator (erpcgen) v1.9.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSmsg-Lite v3.2.1

[2.11.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.0
 - eRPC generator (erpcgen) v1.9.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.2.0

[2.10.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.8.1
 - eRPC generator (erpcgen) v1.8.1
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.1.2

[2.9.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.8.0
 - eRPC generator (erpcgen) v1.8.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.1.1

[2.8.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.4
 - eRPC generator (erpcgen) v1.7.4
 - Multicore Manager (MCMgr) v4.1.0
 - RPSMsg-Lite v3.1.0

[2.7.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.3
 - eRPC generator (erpcgen) v1.7.3
 - Multicore Manager (MCMgr) v4.1.0
 - RPSMsg-Lite v3.0.0

[2.6.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.2
 - eRPC generator (erpcgen) v1.7.2
 - Multicore Manager (MCMgr) v4.0.3
 - RPSmsg-Lite v2.2.0

[2.5.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.1
 - eRPC generator (erpcgen) v1.7.1
 - Multicore Manager (MCMgr) v4.0.2
 - RPSmsg-Lite v2.0.2

[2.4.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.0
 - eRPC generator (erpcgen) v1.7.0
 - Multicore Manager (MCMgr) v4.0.1
 - RPSmsg-Lite v2.0.1

[2.3.1]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.6.0
 - eRPC generator (erpcgen) v1.6.0
 - Multicore Manager (MCMgr) v4.0.0
 - RPSmsg-Lite v1.2.0

[2.3.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.5.0
 - eRPC generator (erpcgen) v1.5.0
 - Multicore Manager (MCMgr) v3.0.0
 - RPSmsg-Lite v1.2.0

[2.2.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.4.0
 - eRPC generator (erpcgen) v1.4.0
 - Multicore Manager (MCMgr) v2.0.1
 - RPSMsg-Lite v1.1.0

[2.1.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.3.0
 - eRPC generator (erpcgen) v1.3.0

[2.0.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.2.0
 - eRPC generator (erpcgen) v1.2.0
 - Multicore Manager (MCMgr) v2.0.0
 - RPSMsg-Lite v1.0.0

[1.1.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.1.0
 - Multicore Manager (MCMgr) v1.1.0
 - Open-AMP / RPSMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

[1.0.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.0.0
 - Multicore Manager (MCMgr) v1.0.0
 - Open-AMP / RPSMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

Multicore SDK Components

RPSMSG-Lite

MCUXpresso SDK : mcuxsdk-middleware-rpsmsg-lite

Overview This repository is for MCUXpresso SDK RPMSG-Lite middleware delivery and it contains RPMSG-Lite component officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository [mcuxsdk](#) for the complete delivery of MCUXpresso SDK to be able to build and run RPMSG-Lite examples that are based on mcux-sdk-middleware-rpmsg-lite component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [RPMSG-Lite - Documentation](#) to review details on the contents in this sub-repo.

For Further API documentation, please look at [doxygen documentation](#)

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

Contribution We welcome and encourage the community to submit patches directly to the rpmsg-lite project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

RPMSG-Lite This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (<https://github.com/OpenAMP/open-amp>), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For overview please read [RPMSG-Lite VirtIO Overview](#).

For RPMSG-Lite Design Considerations please read [RPMSG-Lite Design Considerations](#).

Motivation to create RPMsg-Lite There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

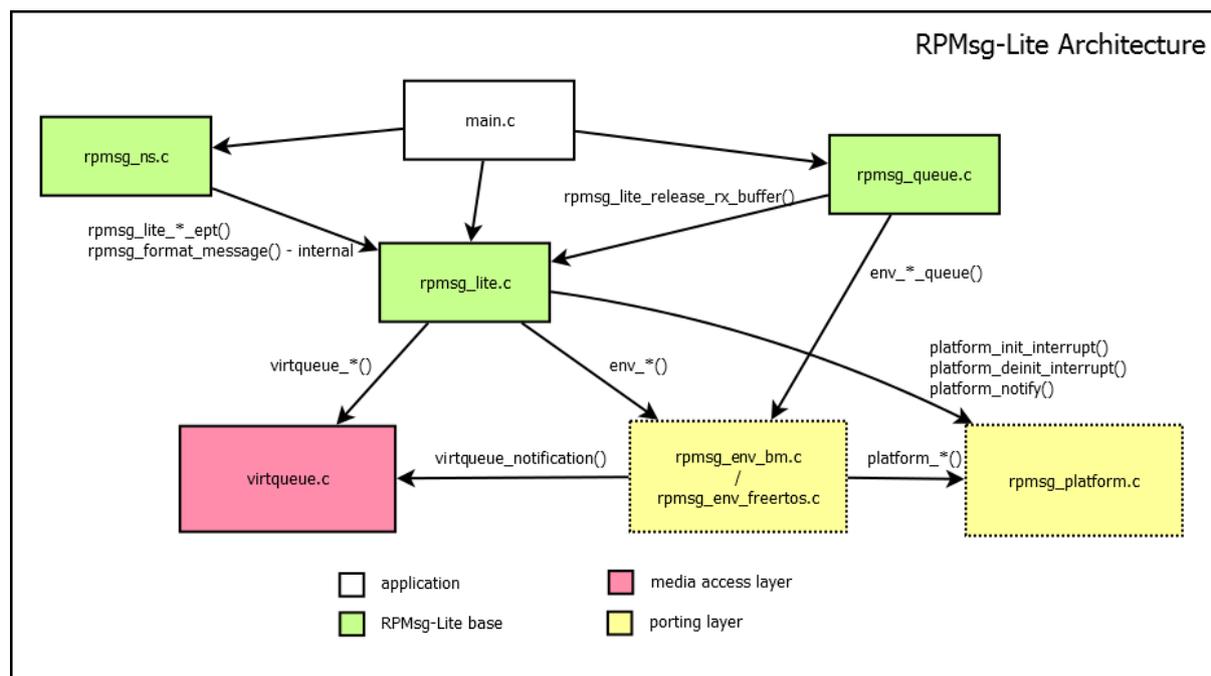
Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

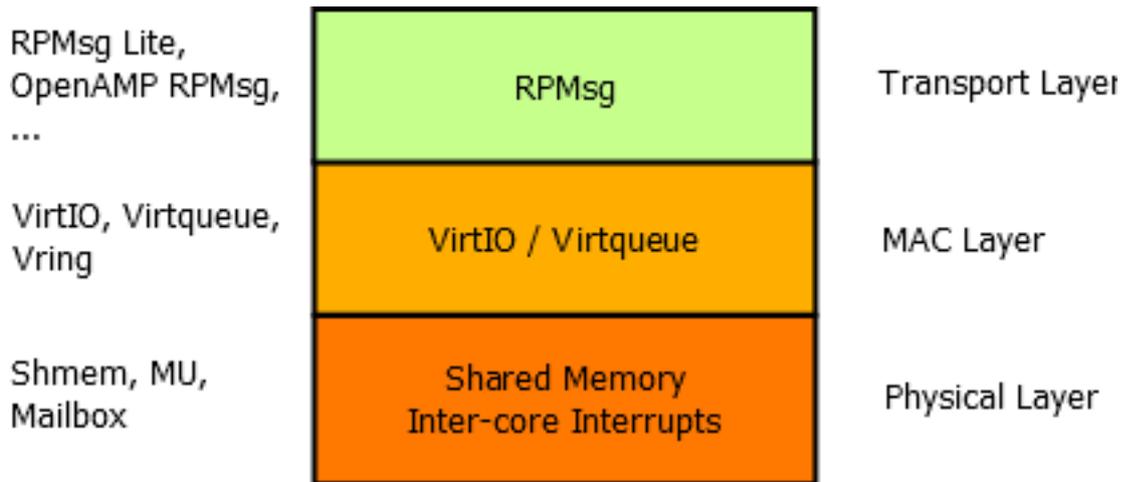
Implementation The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in `rpmsg_lite.c`. Two optional components are used to implement a blocking receive API (in `rpmsg_queue.c`) and dynamic “named” endpoint creation and deletion announcement service (in `rpmsg_ns.c`).

The actual “media access” layer is implemented in `virtqueue.c`, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as `vring` or `virtqueue`.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in `rpmsg_env_bm.c`, and the FreeRTOS environment is implemented in `rpmsg_env_freertos.c` etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in `rpmsg_platform.c` and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional) This subcomponent is optional and requires implementation of the `env_*_queue()` functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional) This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage The application should put the `/rpmmsg_lite/lib/include` directory to the include path and in the application, include either the `rpmmsg_lite.h` header file, or optionally also include the `rpmmsg_queue.h` and/or `rpmmsg_ns.h` files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, `rpmmsg_env_myrtos.c`) and to include it in the project build.

The initialization of the stack is done by calling the `rpmmsg_lite_master_init()` on the master side and the `rpmmsg_lite_remote_init()` on the remote side. This initialization function must be called prior to any RPMMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the `rpmmsg_lite_create_ept()` function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the `RL_USE_STATIC_API` option is set to 1. If not, the stack internally calls `env_alloc()` to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the `rpmmsg_queue_create()` function must be called before calling `rpmmsg_lite_create_ept()`. The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to `rpmmsg_queue_rx_cb()`. Then, it is possible to use `rpmmsg_queue_receive()` function to listen on a queue object for incoming messages. The `rpmmsg_lite_send()` function is used to send messages to the other side.

The RPMMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.

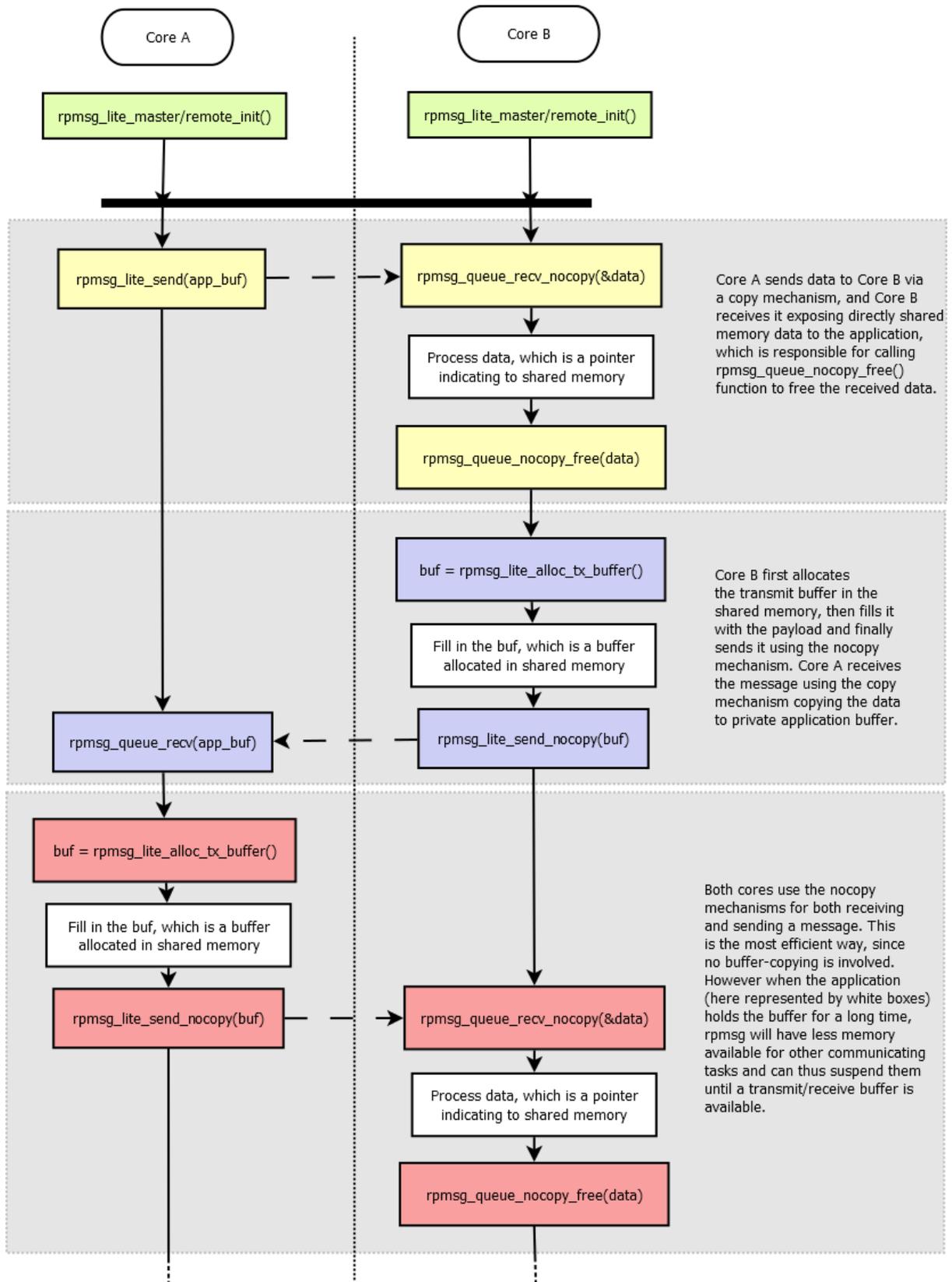
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the `rpmsg_lite_alloc_tx_buffer()` function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the `rpmsg_lite_alloc_tx_buffer()` size output parameter).
- Call the `rpmsg_lite_send_nocopy()` function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the `rpmsg_lite_send_nocopy()` function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the `rpmsg_queue_rcv_nocopy()` function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the `rpmsg_queue_nocopy_free()` function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function `rpmsg_queue_destroy()` is used to destroy a queue, `rpmsg_lite_destroy_ept()` is used to destroy an endpoint and finally, `rpmsg_lite_deinit()` is used to deinitialize the RPMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples RPMsg_Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages. To get the board list with multicore support (RPMsg_Lite included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded,

see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for RPMsg_Lite multicore examples with 'rpmsg_lite_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg_Lite multicore examples is using the [mcuxsdk-manifests](#) Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in [readme section](#). Once done the armgcc rpmsg_lite examples can be found in

mcuxsdk/examples/_<board_name>/multicore_examples

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the RPMsg_Lite examples use the 'rpmsg_lite_' name prefix.

Notes

Environment layers implementation Several environment layers are provided in lib/rpmsg_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg_env_bm.c
- rpmsg_env_freertos.c
- rpmsg_env_xos.c
- rpmsg_env_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

Shared memory configuration It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in linker file is also a good practise, it is recommended to use linker files from MCUXpressoSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

Configuration options The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg_default_config.h header file. This configuration can be customized by the user by including rpmsg_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

Config- uration option	De- fault value	Usage
RL_MS_PE	(1)	Delay in milliseconds used in non-blocking API functions for polling.
RL_BUFFERE	(496)	Size of the buffer payload, it must be equal to (240, 496, 1008, ...) [$2^n - 16$]
RL_BUFFERE	(2)	Number of the buffers, it must be power of two (2, 4, ...)
RL_API_H	(1)	Zero-copy API functions enabled/disabled.
RL_USE_S	(0)	Static API functions (no dynamic allocation) enabled/disabled.
RL_USE_D	(0)	Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.
RL_CLEAF	(0)	Clearing used buffers before returning back to the pool of free buffers enabled/disabled.
RL_USE_M	(0)	When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.
RL_USE_E	(0)	When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).
RL_DEBU	(0)	When enabled buffer pointers passed to <code>rpmsg_lite_send_nocopy()</code> and <code>rpmsg_lite_release_rx_buffer()</code> functions (enabled by <code>RL_API_HAS_ZEROCOPY</code> config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.
RL_ALLO	(0)	When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).
RL_ALLO	(0)	It allows to define custom shared memory configuration and replacing the shared memory related global settings from <code>rpmsg_config.h</code> . This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg-Lite instances use the same shared memory arrangement as defined by common config macros).
RL_ASSER	see rpmsg	Assert implementation.

How to format rpmsg-lite code To format code, use the application developed by Google, named *clang-format*. This tool is part of the *llvm* project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the `.clang-format` file, placed in a root of the rpmsg-lite directory where Python script `run_clang_format.py` can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

References

[1] M. Novak, M. Cingel, **Lockless Shared Memory Based Multicore Communication Protocol** Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

Changelog RPMMSG-Lite All notable changes to this project will be documented in this file. The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

Unreleased

Added

- RT700 porting layer added support to send rpmsg messages between CM33_0 <-> Hifi1 and CM33_1 <-> Hifi4 cores.
- Add new platform macro `RL_PLATFORM_MAX_ISR_COUNT` this will set number of IRQ count per platform. This macro is then used in environment layers to set `isr_table` size where irq handles are registered. It size should match the bit length of `VQ_ID` so all combinations can fit into table.

v5.2.1

Added

- Doc added RPMSG-Lite VirtIO Overview
- Doc added RPMSG-Lite Design Considerations
- Added `frdmimxrt1186` unit testing

Fixed

- Fixed CERT-C INT31-C violation in `platform_notify` function in `rpmsg_platform.c` for `imxrt700_m33`, `imxrt700_hifi4`, `imxrt700_hifi1` platforms

v5.2.0

Added

- Add MCXL20 porting layer and unit testing
- New utility macro `RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE` to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

Changed

- Change `rpmsg_platform.c` to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.
- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

Fixed

- Fixed `env_wait_for_link_up` function to handle timeout in link state checks for baremetal and qnx environment, `RL_BLOCK` mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure `RL_PLATFORM_HIGHEST_LINK_ID` remains within safe range for 16-bit casting in virtqueue ID creation.

- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in `shmem_length - (uint32_t)RL_VRING_OVERHEAD` and `shmem_length - 2U * shmem_config.vring_size` expressions.
- Fixed CERT INT31-C violation in `platform_interrupt_disable()` and similar functions by replacing unsafe cast from `uint32_t` to `int32_t` with a return of 0 constant.
- Fixed unsigned integer underflow in `rpmsg_lite_alloc_tx_buffer()` where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in `rpmsg_lite.c` where `size` parameter was cast from `uint32_t` to `uint16_t` without proper validation.
 - Applied consistent masking approach to both `size` and `flags` parameters: `(uint16_t)(value & 0xFFFFU)`.
 - This fix prevents potential data loss when `size` values exceed 65535.
- Fixed CERT INT31-C violation in `env_memset` functions by explicitly converting `int32_t` values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard `memset()` function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
 - `rpmsg_env_freertos.c`: Added validation before converting `int32_t` to `UBaseType_t`.
 - `rpmsg_env_qnx.c`: Fixed format string and added validation before assigning to `mqstat` fields.
 - `rpmsg_env_threadx.c`: Added validation to prevent integer overflow and negative values.
 - `rpmsg_env_xos.c`: Added range checking before casting to `uint16_t`.
 - `rpmsg_env_zephyr.c`: Added validation before passing values to `k_msgq_init`.
- Fixed a CERT INT31-C compliance issue in `env_get_current_queue_size()` function where an unsigned queue count was cast to a signed `int32_t` without proper validation, which could lead to lost or misinterpreted data if queue size exceeded `INT32_MAX`.
- Fixed CERT INT31-C violation in `rpmsg_platform.c` where `memcmp()` return value (signed int) was compared with unsigned constant without proper type handling.
- Fixed CERT INT31-C violation in `rpmsg_platform.c` where casting from `uint32_t` to `uint16_t` could potentially result in data loss. Changed length variable type from `uint16_t` to `uint32_t` to properly handle memory address differences without truncation.
- Fixed potential integer overflow in `env_sleep_msec()` function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.
- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on `imx943` platform.
- Added `RL_MAX_BUFFER_COUNT` (32768U) and `RL_MAX_VRING_ALIGN` (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in `vring_need_event()`, added cast to `uint16_t` for each operand.

v5.1.4 - 27-Mar-2025

Added

- Add KW43B43 porting layer

Changed

- Doxygen bump to version 1.9.6

v5.1.3 - 13-Jan-2025

Added

- Memory cache management of shared memory. Enable with `#define RL_USE_DCACHE (1)` in `rpmsg_config.h` in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for imx95, imxrt700, mcmxw71x, mcmxw72x, kw47b42 added.

v5.1.2 - 08-Jul-2024

Changed

- Zephyr-related changes.
- Minor Misra corrections.

v5.1.1 - 19-Jan-2024

Added

- Test suite provided.
- Zephyr support added.

Changed

- Minor changes in platform and env. layers, minor test code updates.

v5.1.0 - 02-Aug-2023

Added

- RPMsg-Lite: Added aarch64 support.

Changed

- RPMsg-Lite: Increased the queue size to $(2 * RL_BUFFER_COUNT)$ to cover zero copy cases.
- Code formatting using LLVM16.

Fixed

- Resolved issues in ThreadX env. layer implementation.

v5.0.0 - 19-Jan-2023

Added

- Timeout parameter added to `rpmsg_lite_wait_for_link_up` API function.

Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- `VRING_SIZE` is set based on number of used buffers now (as calculated in `vring_init`) - updated for all platforms that are not communicating to Linux `rpmsg` counterpart.

Fixed

- Fixed wrong `RL_VRING_OVERHEAD` macro comment in `platform.h` files
- Misra corrections.

v4.0.0 - 20-Jun-2022

Added

- Added support for custom shared memory arrangement per the `RPMsg_Lite` instance.
- Introduced new `rpmsg_lite_wait_for_link_up()` API function - this allows to avoid using busy loops in rtos environments, GitHub PR #21.

Changed

- Adjusted `rpmsg_lite_is_link_up()` to return `RL_TRUE/RL_FALSE`.

v3.2.0 - 17-Jan-2022

Added

- Added support for i.MX8 MP multicore platform.

Changed

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR #14.
- Aligned `rpmsg_env_xos.c` and some platform layers to latest static allocation support.

Fixed

- Minor Misra and typo corrections, GitHub PR #19, #20.

v3.1.2 - 16-Jul-2021

Added

- Addressed MISRA 21.6 rule violation in `rpmsg_env.h` (use SDK's `PRINTF` in MCUXpressoSDK examples, otherwise `stdio printf` is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

Fixed

- Fixed incorrect description of the `rpmsg_lite_get_endpoint_from_addr` function.

Changed

- Updated `RL_BUFFER_COUNT` documentation (issue #10).
- Updated `imxrt600_hifi4` platform layer.

v3.1.1 - 15-Jan-2021

Added

- Introduced `RL_ALLOW_CONSUMED_BUFFERS_NOTIFICATION` config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

Changed

- Several MISRA C-2012 violations addressed.

v3.1.0 - 22-Jul-2020

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in `rpmsg_lite_rx_callback()` for QNX env.
- Correction of `rpmsg_lite_instance` structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

Changed

- Clang update to v10.0.0, code re-formatted.

v3.0.0 - 20-Dec-2019

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.
- Code formatted.

v2.2.0 - 20-Mar-2019

Added

- Added configuration macro `RL_DEBUG_CHECK_BUFFERS`.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the `RL_USE_ENVIRONMENT_CONTEXT` configuration macro).
- Data types consolidation.

v1.1.0 - 28-Apr-2017

Added

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremetal and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

Multicore Manager

MCUXpresso SDK : `mcuxsdk-middleware-mcmgr` (Multicore Manager)

Overview This repository is for MCUXpresso SDK Multicore Manager middleware delivery and it contains Multicore Manager component officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository [mcuxsdk](#) for the complete delivery of MCUXpresso SDK to be able to build and run Multicore Manager examples that are based on `mcux-sdk-middleware-mcmgr` component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [Multicore Manager - Documentation](#) to review details on the contents in this sub-repo.

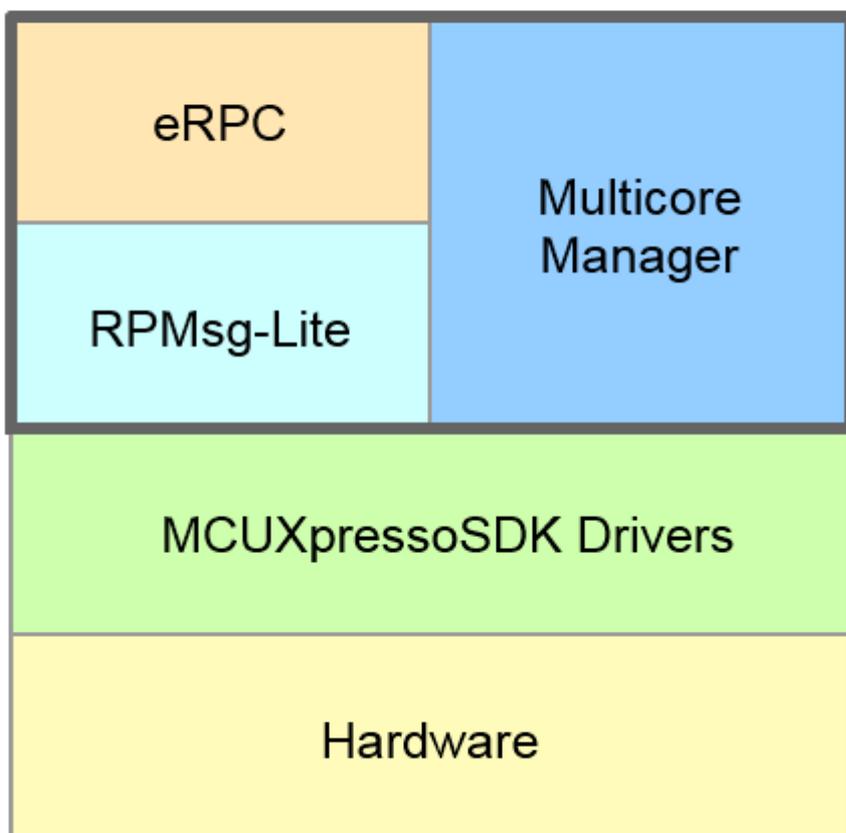
For Further API documentation, please look at [doxygen documentation](#)

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

Contribution We welcome and encourage the community to submit patches directly to the mcmgr project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

Usage of the MCMGR software component The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,
    ↪starting synchronously. */
    MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
    .
    .
    .
    /* Stop secondary core execution. */
    MCMGR_StopCore(kMCMGR_Core1);
}
```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR_StopCore / MCMGR_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR_RegisterEvent() API. Triggering these events is done using the MCMGR_TriggerEvent() API. mcmgr_event_type_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()
↪called on the secondary core side */
void RPSgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPSgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();
```

(continues on next page)

(continued from previous page)

```

/* Initialize MCMGR, install generic event handlers */
MCMGR_Init();

/* Register the application event before starting the secondary core */
MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RMsgRemoteReadyEventHandler, (void*)RMsgRemoteReadyEventData);

/* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rmsg_lite_base address as startup data, starting synchronously. */
MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rmsg_lite_base, kMCMGR_Start_Synchronous);

/* Wait until the secondary core application signals the rmsg remote has been initialized and is ready to communicate. */
while(APP_RMSG_READY_EVENT_DATA != RMsgRemoteReadyEventData[APP_SECONDARY_CORE]) {}
.
.
.
}

```

Code for the secondary side:

```

#include "mcmgr.h"

#define APP_RMSG_READY_EVENT_DATA (1)

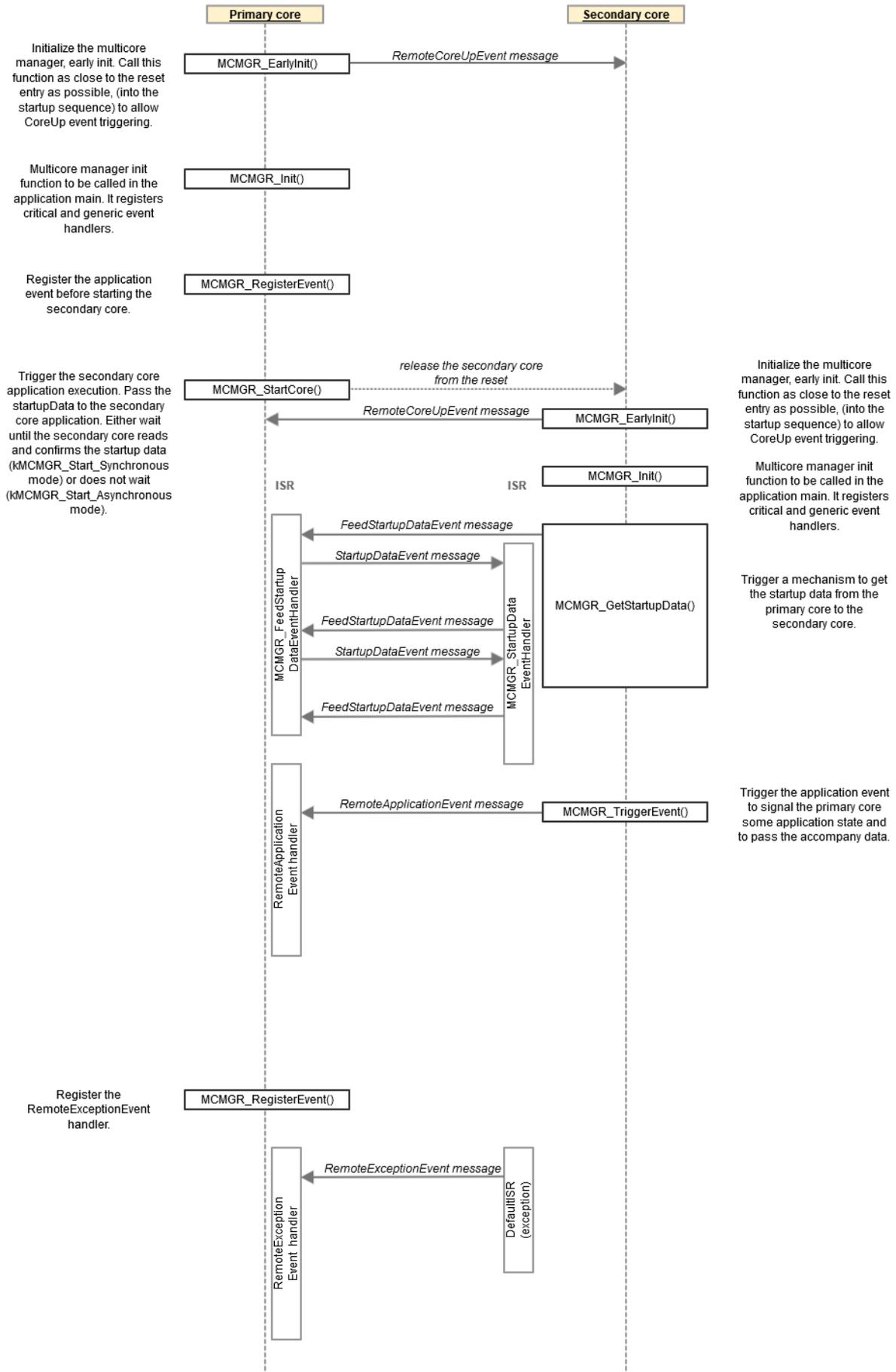
void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
    .
    .
    .

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RMSG_READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RMSG_READY_EVENT_DATA);
    .
    .
    .
}

```

MCMGR Data Exchange Diagram The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.



Changelog Multicore Manager All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

v5.0.1

Added

- Added frdmimxrt1186 unit testing

Changed

- [KW43] Rename core#1 reset control register

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.
- Fix path to mcmgr headers directory in doxyfile

v5.0.0

Added

- Added MCMGR_BUSY_POLL_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more than two cores. Breaking API change by adding parameter `coreNum` specifying core number in functions bellow.
 - `MCMGR_GetStartupData(uint32_t *startupData, mcmgr_core_t coreNum)`
 - `MCMGR_TriggerEvent(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)`
 - `MCMGR_TriggerEventForce(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)`
 - `typedef void (*mcmgr_event_callback_t)(uint16_t data, void *context, mcmgr_core_t coreNum);`

When registering the event with function `MCMGR_RegisterEvent()` user now needs to provide `callbackData` pointer to array of elements per every core in system (see README.md for example). In case of systems with only two cores the `coreNum` in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: [Porting-GuideTo_v5.md](#)

- Updated all porting files to support new MCMGR API.
- Added new platform specific include file `mcmgr_platform.h`. It will contain common platform specific macros that can be then used in `mcmgr` and application. e.g. platform core count `MCMGR_CORECOUNT 4`.
- Move all header files to new `inc` directory.
- Added new platform-specific include files `inc/platform/<platform_name>/mcmgr_platform.h`.

Added

- Add MCXL20 porting layer and unit testing

v4.1.7

Fixed

- `mcmgr_stop_core_internal()` function now returns `kStatus_MCMGR_NotImplemented` status code instead of `kStatus_MCMGR_Success` when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

[v4.1.6]

Added

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New `MCMGR_ProcessDeferredRxIsr()` API added.

[v4.1.5]

Added

- Add notification into `MCMGR_EarlyInit` and `mcmgr_early_init_internal` functions to avoid using uninitialized data in their implementations.

[v4.1.4]

Fixed

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

[v4.1.3]

Added

- Add porting layers for imxrt1180.

Fixed

- `mu_isr()` updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- `mcmgr_mu_internal.c` code adaptation to new supported SoCs.

[v4.1.2]

Fixed

- Update `mcmgr_stop_core_internal()` implementations to set core state to `kMCMGR_ResetCoreState`.

[v4.1.0]

Fixed

- Code adjustments to address MISRA C-2012 Rules

[v4.0.3]

Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

[v4.0.2]

Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

[v4.0.1]

Fixed

- Code formatting, removed unused code

[v4.0.0]

Added

- Add new `MCMGR_TriggerEventForce()` API.

[v3.0.0]

Removed

- Removed `MCMGR_LoadApp()`, `MCMGR_MapAddress()` and `MCMGR_SignalReady()`

Modified

- Modified `MCMGR_GetStartupData()`

Added

- Added MCMGR_EarlyInit(), MCMGR_RegisterEvent() and MCMGR_TriggerEvent()
- Added the ability for remote core monitoring and event handling

[v2.0.1]

Fixed

- Updated to be Misra compliant.

[v2.0.0]

Added

- Support for lpcxpresso54114 board.

[v1.1.0]

Fixed

- Ported to KSDK 2.0.0.

[v1.0.0]

Added

- Initial release.

eRPC

MCUXpresso SDK : mcuxsdk-middleware-erpc

Overview This repository is for MCUXpresso SDK eRPC middleware delivery and it contains eRPC component officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository [mcuxsdk](#) for the complete delivery of MCUXpresso SDK to be able to build and run eRPC examples that are based on mcux-sdk-middleware-erpc component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [eRPC - Documentation](#) to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

Contribution We welcome and encourage the community to submit patches directly to the eRPC project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

eRPC

- *MCUXpresso SDK : mcuxsdk-middleware-erpc*
 - *Overview*
 - *Documentation*
 - *Setup*
 - *Contribution*
- *eRPC*
 - *About*
 - *Releases*
 - * *Edge releases*
 - *Documentation*
 - *Examples*
 - *References*
 - *Directories*
 - *Building and installing*
 - * *Requirements*
 - *Windows*
 - *Mac OS X*
 - * *Building*
 - *CMake and KConfig*
 - *Make*
 - * *Installing for Python*
 - *Known issues and limitations*
 - *Code providing*

About

eRPC (Embedded RPC) is an open source Remote Procedure Call (RPC) system for multichip embedded systems and heterogeneous multicore SoCs.

Unlike other modern RPC systems, such as the excellent [Apache Thrift](#), eRPC distinguishes itself by being designed for tightly coupled systems, using plain C for remote functions, and having a small code size (<5kB). It is not intended for high performance distributed systems over a network.

eRPC does not force upon you any particular API style. It allows you to export existing C functions, without having to change their prototypes. (There are limits, of course.) And although the internal infrastructure is written in C++, most users will be able to use only the simple C setup APIs shown in the examples below.

A code generator tool called `erpcgen` is included. It accepts input IDL files, having an `.erpc` extension, that have definitions of your data types and remote interfaces, and generates the shim code that handles serialization and invocation. `erpcgen` can generate either C/C++ or Python code.

Example `.erpc` file:

```
// Define a data type.
enum LEDName { kRed, kGreen, kBlue }

// An interface is a logical grouping of functions.
interface IO {
    // Simple function declaration with an empty reply.
    set_led(LEDName whichLed, bool onOrOff) -> void
}

```

Client side usage:

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* init eRPC client IO service */
    initIO_client(client_manager);

    // Now we can call the remote function to turn on the green LED.
    set_led(kGreen, true);

    /* deinit objects */
    deinitIO_client();
    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* scope for client service */
    {
        /* init eRPC client IO service */

```

(continues on next page)

(continued from previous page)

```

    IO_client client(client_manager);

    // Now we can call the remote function to turn on the green LED.
    client.set_led(kGreen, true);
}

/* deinit objects */
erpc_client_deinit(client_manager);
erpc_mbf_dynamic_deinit(message_buffer_factory);
erpc_transport_tcp_deinit(transport);
}

```

Server side usage:

```

// Implement the remote function.
void set_led(LEDName whichLed, bool onOrOff) {
    // implementation goes here
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    erpc_service_t service = create_IO_service();

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, service);

    // Run the server.
    erpc_server_run();

    /* deinit objects */
    destroy_IO_service(service);
    erpc_server_deinit(server);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

```

// Implement the remote function.
class IO : public IO_interface
{
    /* eRPC call definition */
    void set_led(LEDName whichLed, bool onOrOff) override {
        // implementation goes here
    }
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    IO IOImpl;
    IO_service io(&IOImpl);

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);

```

(continues on next page)

(continued from previous page)

```

message_buffer_factory = erpc_mbf_dynamic_init();
server = erpc_server_init(transport, message_buffer_factory);

/* add custom service implementation to the server */
erpc_add_service_to_server(server, &io);

/* poll for requests */
erpc_status_t err = server.run();

/* deinit objects */
erpc_server_deinit(server);
erpc_mbf_dynamic_deinit(message_buffer_factory);
erpc_transport_tcp_deinit(transport);
}

```

A number of transports are supported, and new transport classes are easy to write.

Supported transports can be found in *erpc/erpc_c/transport* folder. E.g:

- CMSIS UART
- NXP Kinetis SPI and DSPI
- POSIX and Windows serial port
- TCP/IP (mostly for testing)
- NXP RPMsg-Lite / RPMsg TTY
- SPIdev Linux
- USB CDC
- NXP Messaging Unit

eRPC is available with an unrestrictive BSD 3-clause license. See the [LICENSE file](#) for the full license text.

Releases [eRPC releases](#)

Edge releases Edge releases can be found on [eRPC CircleCI](#) webpage. Choose build of interest, then platform target and choose ARTIFACTS tab. Here you can find binary application from chosen build.

Documentation [Documentation](#) is in the wiki section.

[eRPC Infrastructure documentation](#)

Examples *Example IDL* is available in the *examples/* folder.

Plenty of eRPC multicore and multiprocessor examples can be also found in NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages.

To get the board list with multicore support (eRPC included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for eRPC multicore examples (RPMsg_Lite or Messaging Unit transports used) or

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples for eRPC multiprocessor examples (UART or SPI transports used).

eRPC examples use the 'erpc_' name prefix.

Another way of getting NXP MCUXpressoSDK eRPC multicore and multiprocessor examples is using the [mcux-sdk](#) Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk repo in [readme Overview section](#). Once done the armgcc eRPC examples can be found in

mcuxsdk/examples/<board_name>/multicore_examples or in

mcuxsdk/examples/<board_name>/multiprocessor_examples folders.

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the eRPC examples use the 'erpc_' name prefix.

References This section provides links to interesting erpc-based projects, articles, blogs or guides:

- [erpc \(EmbeddedRPC\) getting started notes](#)
- [ERPC Linux Local Environment Construction and Use](#)
- [The New Wio Terminal eRPC Firmware](#)

Directories *doc* - Documentation.

doxygen - Configuration and support files for running Doxygen over the eRPC C++ infrastructure and erpcgen code.

erpc_c - Holds C/C++ infrastructure for eRPC. This is the code you will include in your application.

erpc_python - Holds Python version of the eRPC infrastructure.

erpcgen - Holds source code for erpcgen and makefiles or project files to build erpcgen on Windows, Linux, and OS X.

erpcsniffer - Holds source code for erpcsniffer application.

examples - Several example IDL files.

mk - Contains common makefiles for building eRPC components.

test - Client/server tests. These tests verify the entire communications path from client to server and back.

utilities - Holds utilities which bring additional benefit to eRPC apps developers.

Building and installing These build instructions apply to host PCs and embedded Linux. For bare metal or RTOS embedded environments, you should copy the *erpc_c* directory into your application sources.

CMake and KConfig build:

It builds a static library of the eRPC C/C++ infrastructure, the *erpcgen* executable, and optionally the unit tests and examples.

CMake is compatible with gcc and clang. On Windows local MingGW downloaded by *script* can be used.

Make build:

It builds a static library of the eRPC C/C++ infrastructure, the *erpcgen* executable, and optionally the unit tests.

The makefiles are compatible with gcc or clang on Linux, OS X, and Cygwin. A Windows build of erpcgen using Visual Studio is also available in the *erpcgen/VisualStudio_v14* directory. There is also an Xcode project file in the *erpcgen* directory, which can be used to build erpcgen for OS X.

Requirements eRPC now support building **erpcgen**, **erpc_lib**, **tests** and **C examples** using CMake.

Requirements when using CMake:

- **CMake** (minimal version 3.20.0)
- Generator - **Make, Ninja, ...**
- **C/C++ compiler - GCC, CLANG, ...**
- **Binson** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

Requirements when using Make:

- **Make**
- **C/C++ compiler - GCC, CLANG, ...**
- **Binson** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

Windows Related steps to build **erpcgen** using **Visual Studio** are described in *erpcgen/VisualStudio_v14/readme_erpcgen.txt*.

To install MinGW, Bison, Flex locally on Windows:

```
./install_dependencies.ps1
* ````

#### Linux

```bash
./install_dependencies.sh
```

Mandatory for case, when build for different architecture is needed

- gcc-multilib, g++-multilib

## Mac OS X

```
./install_dependencies.sh
```

## Building

**CMake and KConfig** eRPC use CMake and KConfig to configurate and build eRPC related targets. KConfig can be edited by *prj.conf* or *menuconfig* when building.

Generate project, config and build. In *erpc/* execute:

```
cmake -B ./build # in erpc/build generate cmake project
cmake --build ./build --target menuconfig # Build menuconfig and configurate erpcgen, erpc_lib, tests and_
↳examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

\*\*CMake will use the system's default compilers and generator

If you want to use Windows and locally installed MinGW, use *CMake preset* :

```
cmake --preset mingw64 # Generate project in ./build using mingw64's make and compilers
cmake --build ./build --target menuconfig # Build menuconfig and configurate erpcgen, erpc_lib, tests and ↵
↪examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

**Make** To build the library and erpcgen, run from the repo root directory:

```
make
```

To install the library, erpcgen, and include files, run:

```
make install
```

You may need to sudo the make install.

By default this will install into `/usr/local`. If you want to install elsewhere, set the `PREFIX` environment variable. Example for installing into `/opt`:

```
make install PREFIX=/opt
```

List of top level Makefile targets:

- `erpc`: build the `liberpc.a` static library
- `erpcgen`: build the `erpcgen` tool
- `erpcsniffer`: build the sniffer tool
- `test`: build the unit tests under the `test` directory
- `all`: build all of the above
- `install`: install `liberpc.a`, `erpcgen`, and include files

eRPC code is validated with respect to the C++ 11 standard.

**Installing for Python** To install the Python infrastructure for eRPC see instructions in the *erpc python readme*.

### Known issues and limitations

- Static allocations controlled by the `ERPC_ALLOCATION_POLICY` config macro are not fully supported yet, i.e. not all `erpc` objects can be allocated statically now. It deals with the ongoing process and the full static allocations support will be added in the future.

**Code providing** Repository on Github contains two main branches: **main** and **develop**. Code is developed on **develop** branch. Release version is created via merging **develop** branch into **main** branch.

---

Copyright 2014-2016 Freescale Semiconductor, Inc.

Copyright 2016-2025 NXP

### eRPC Getting Started

**Overview** This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc` folder.

**Create an eRPC application** This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
  1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
  2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
  3. Add infrastructure files.
  4. Add user code for client and server applications.
  5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

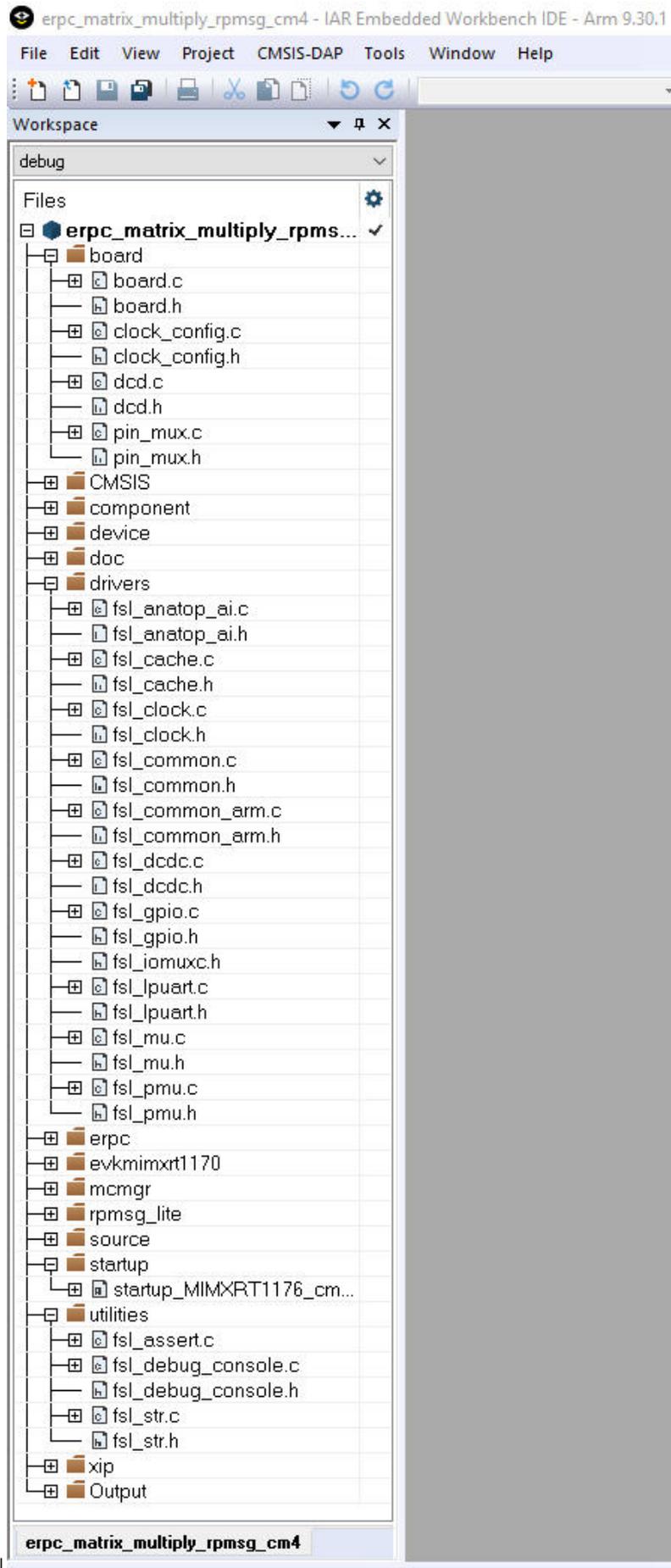
**Multicore server application** The “Matrix multiply” eRPC server project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmc/cm4/iar`

The project files for the eRPC server have the `_cm4` suffix.

**Server project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



|

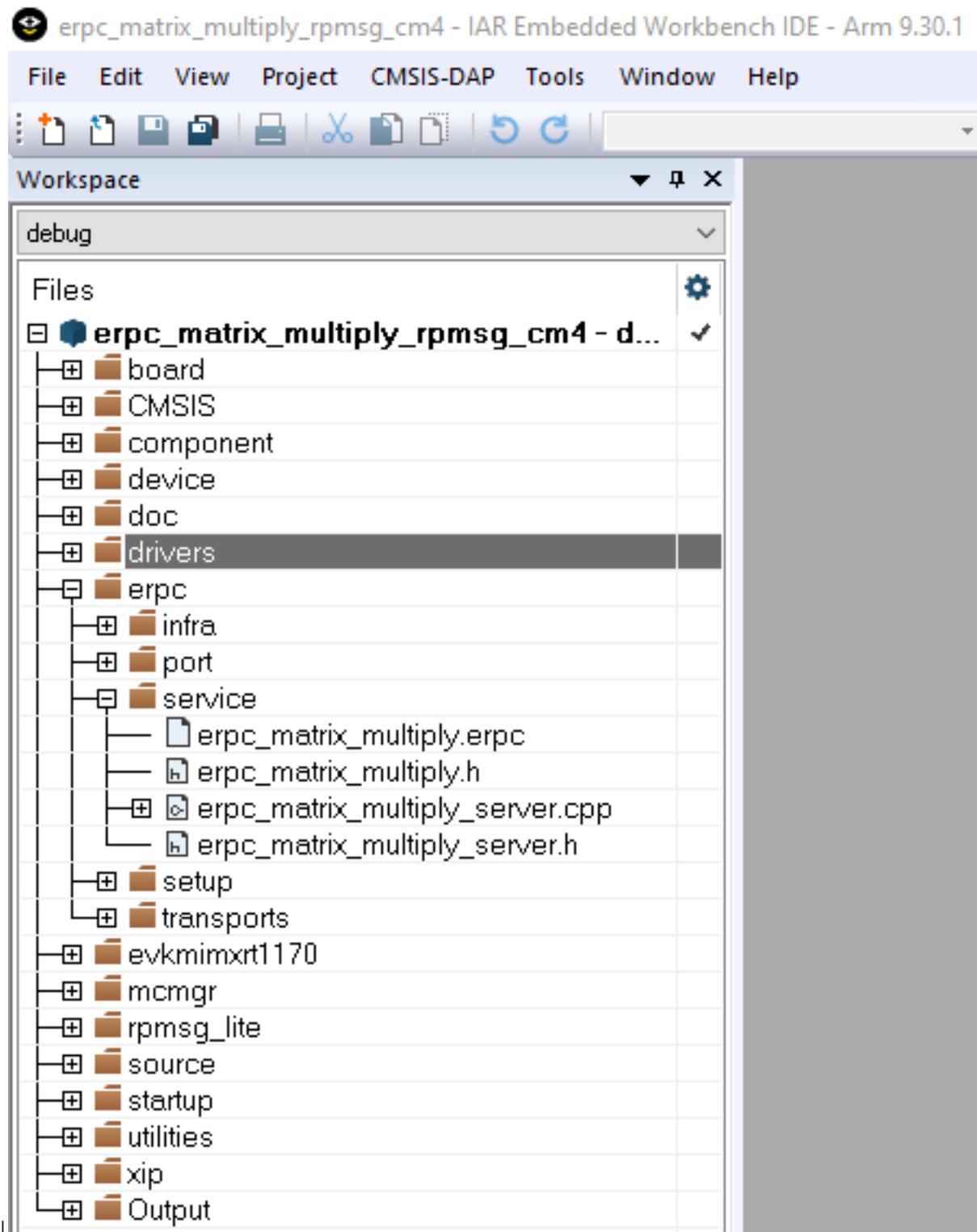
**Parent topic:**Multicore server application

**Server related generated files** The server-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_server.h
- erpc\_\_matrix\_\_multiply\_\_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_common/erpc\_matrix\_multiply/



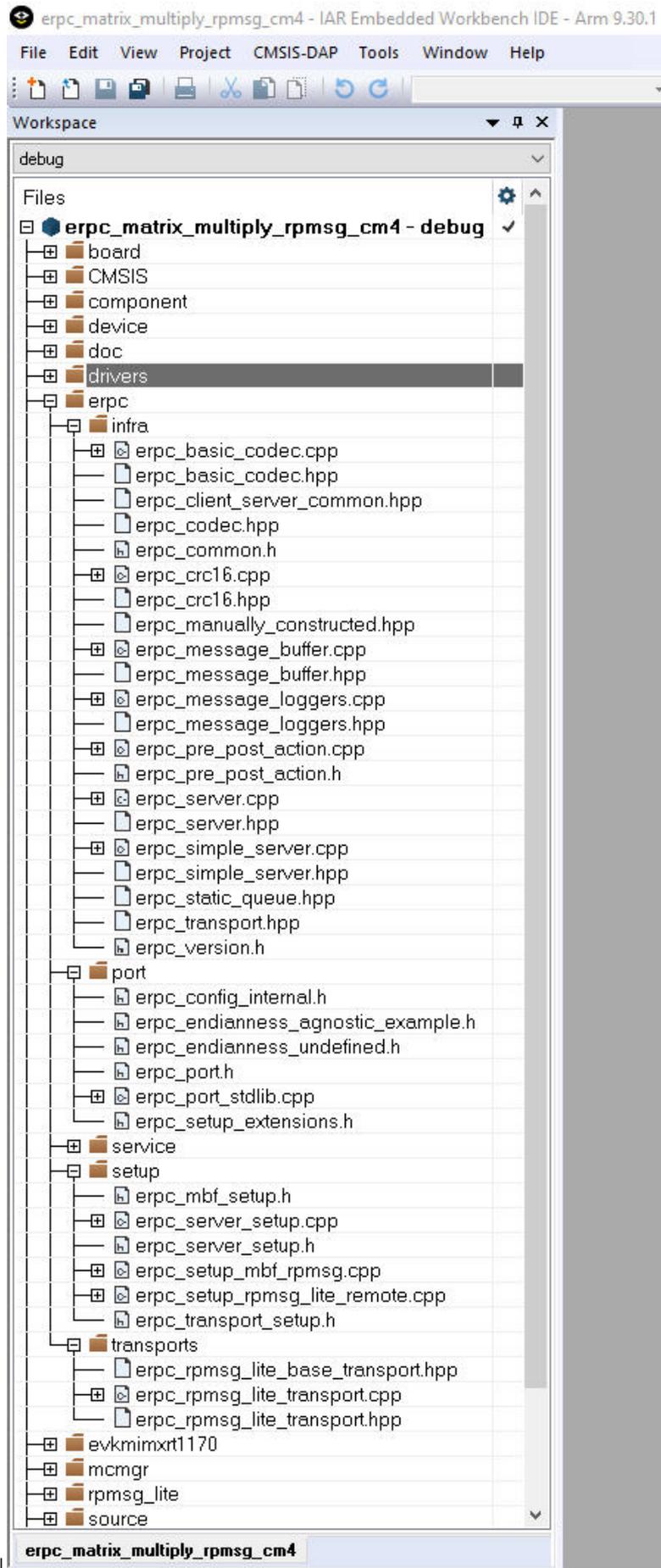
**Parent topic:**Multicore server application

**Server infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
  - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
  - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
  - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
  - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
  - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPLite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.



|

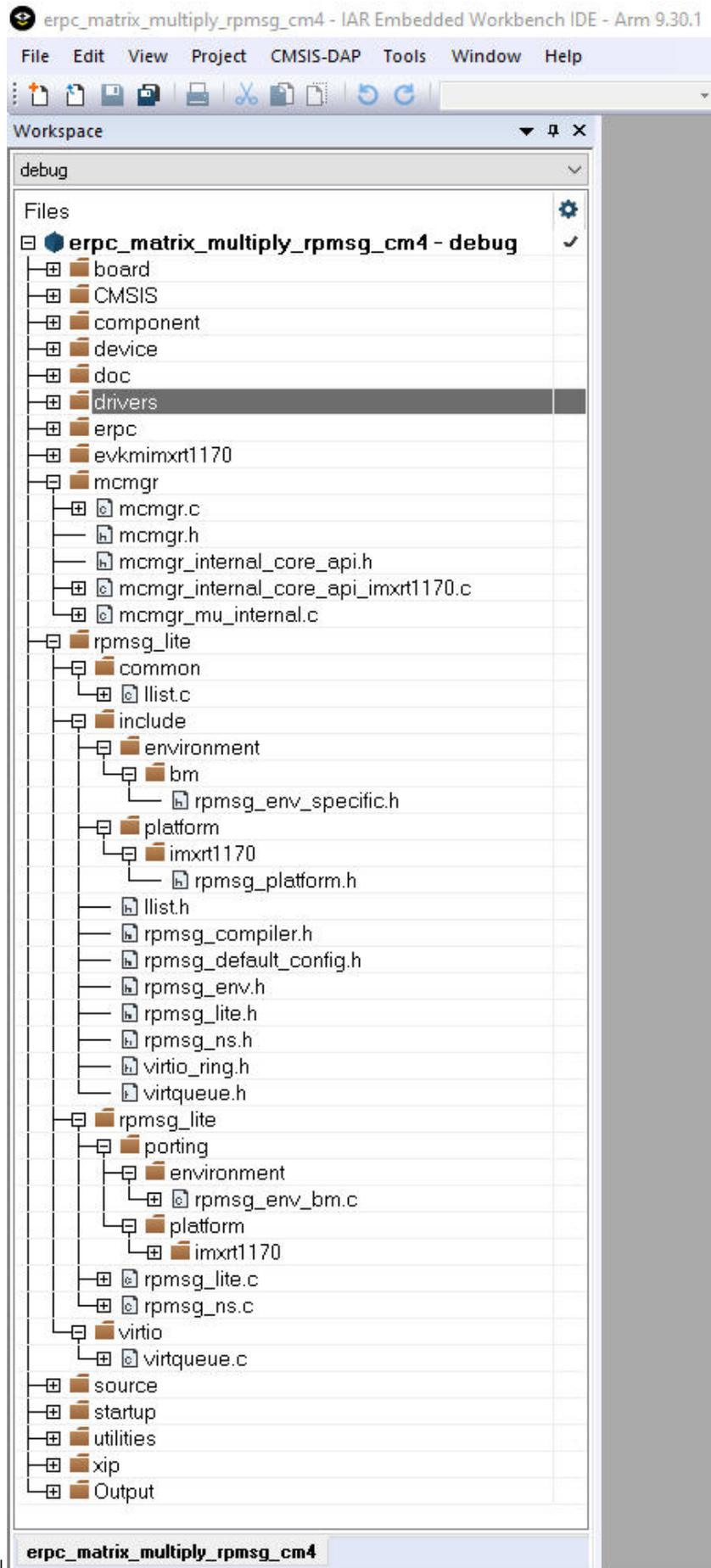
**Parent topic:**Multicore server application

**Server multicore infrastructure files** Because of the RPSMsg-Lite (transport layer), it is also necessary to include RPSMsg-Lite related files, which are in the following folder:

*<MCUXpressoSDK\_install\_dir>/middleware/multicore/rpsmsg\_lite/*

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

*<MCUXpressoSDK\_install\_dir>/middleware/multicore/mcmgr/*



|

**Parent topic:** Multicore server application

**Server user code** The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

The `main_core1.c` file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

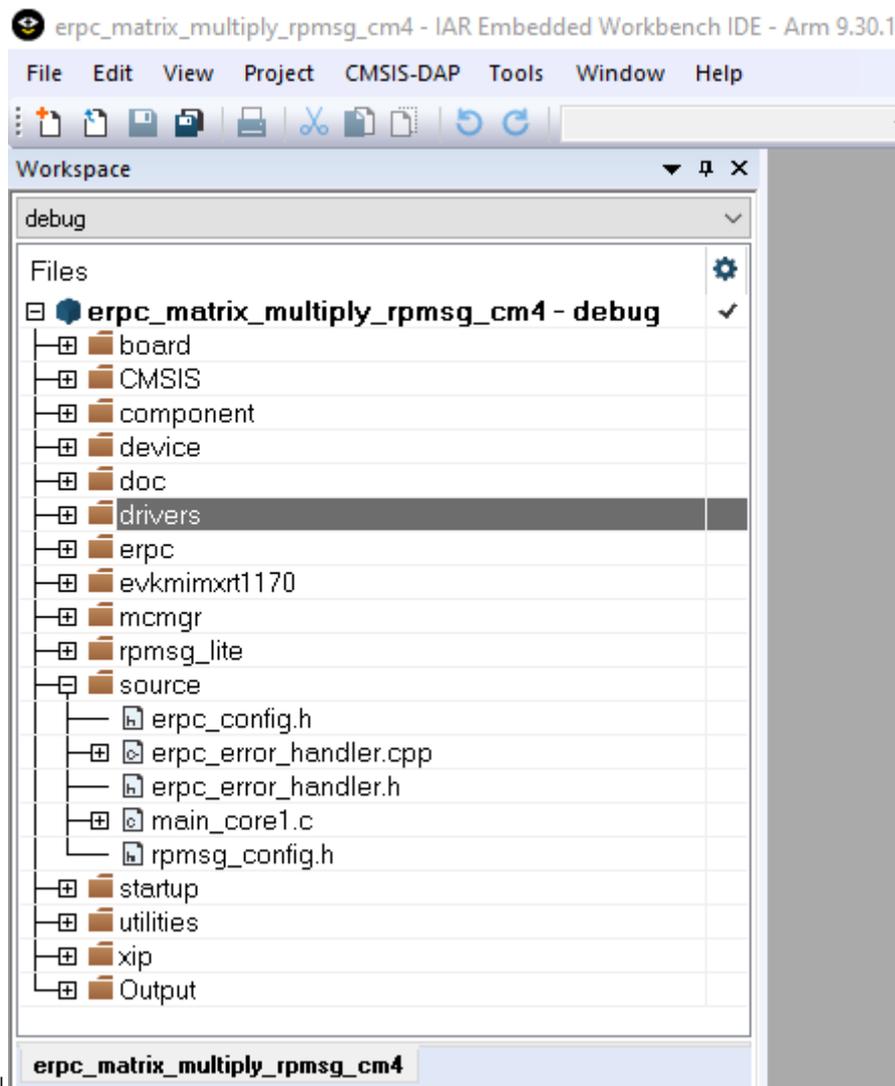
The eRPC-relevant code is captured in the following code snippet:

```

/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
 ...
}
int main()
{
 ...
 /* RPMsg-Lite transport layer initialization */
 erpc_transport_t transport;
 transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
 ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_rpmsg_init(transport);
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server);
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}

```

Except for the application main file, there are configuration files for the RPSMsg-Lite (`rpmsg_config.h`) and eRPC (`erpc_config.h`), located in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/ erpc_matrix_multiply_rpmsg` folder.



**Parent topic:** Multicore server application

**Parent topic:** [Create an eRPC application](#)

**Multicore client application** The “Matrix multiply” eRPC client project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar/`

Project files for the eRPC client have the `_cm7` suffix.

**Client project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



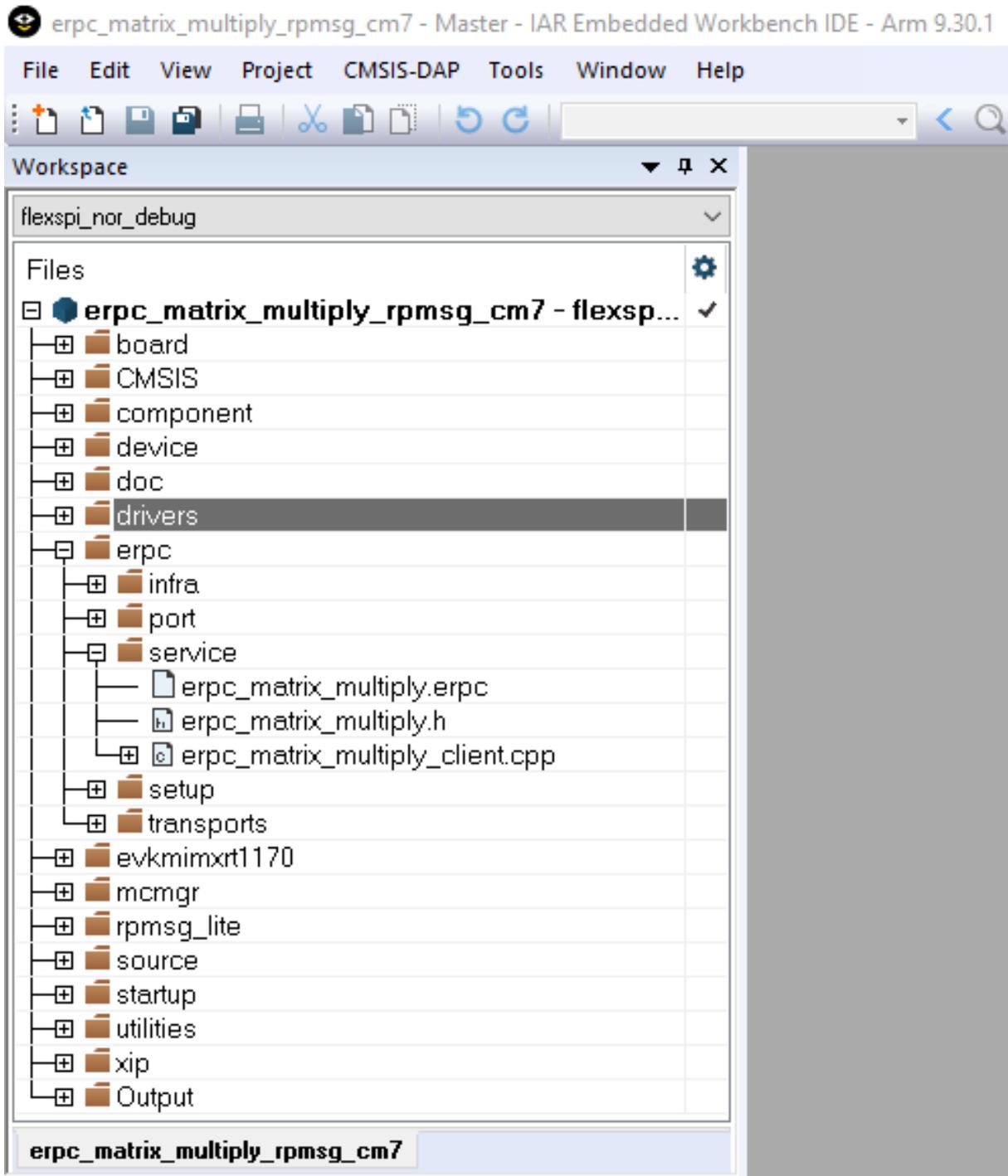
|

**Parent topic:**Multicore client application

**Client-related generated files** The client-related generated files are:

- erpc\_matric\_multiply.h
- erpc\_matrix\_multiply\_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



**Parent topic:**Multicore client application

**Client infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

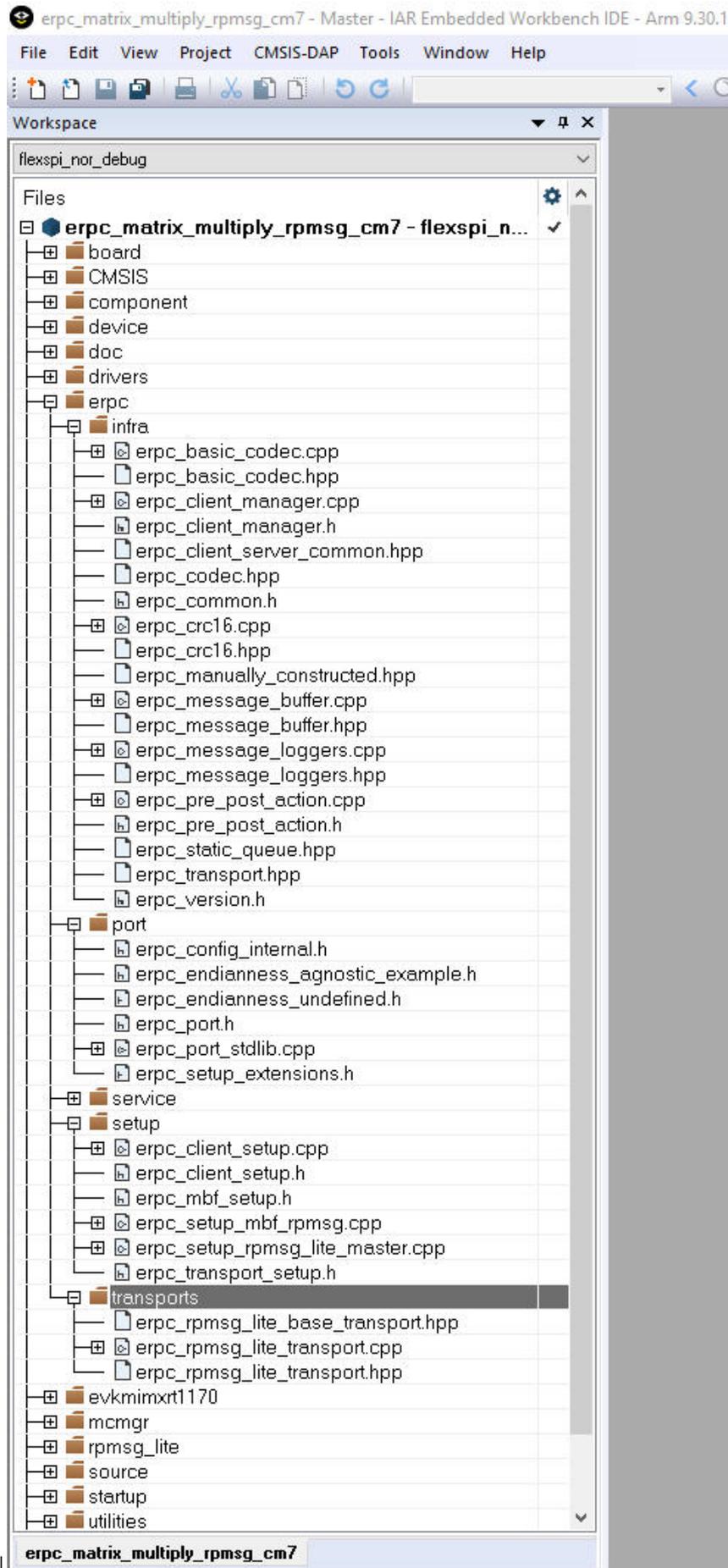
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmg_lite_base_transport.hpp`, `erpc_rpmg_lite_transport.hpp`, and `erpc_rpmg_lite_transport.cpp` files needs to be added into the client project.



|

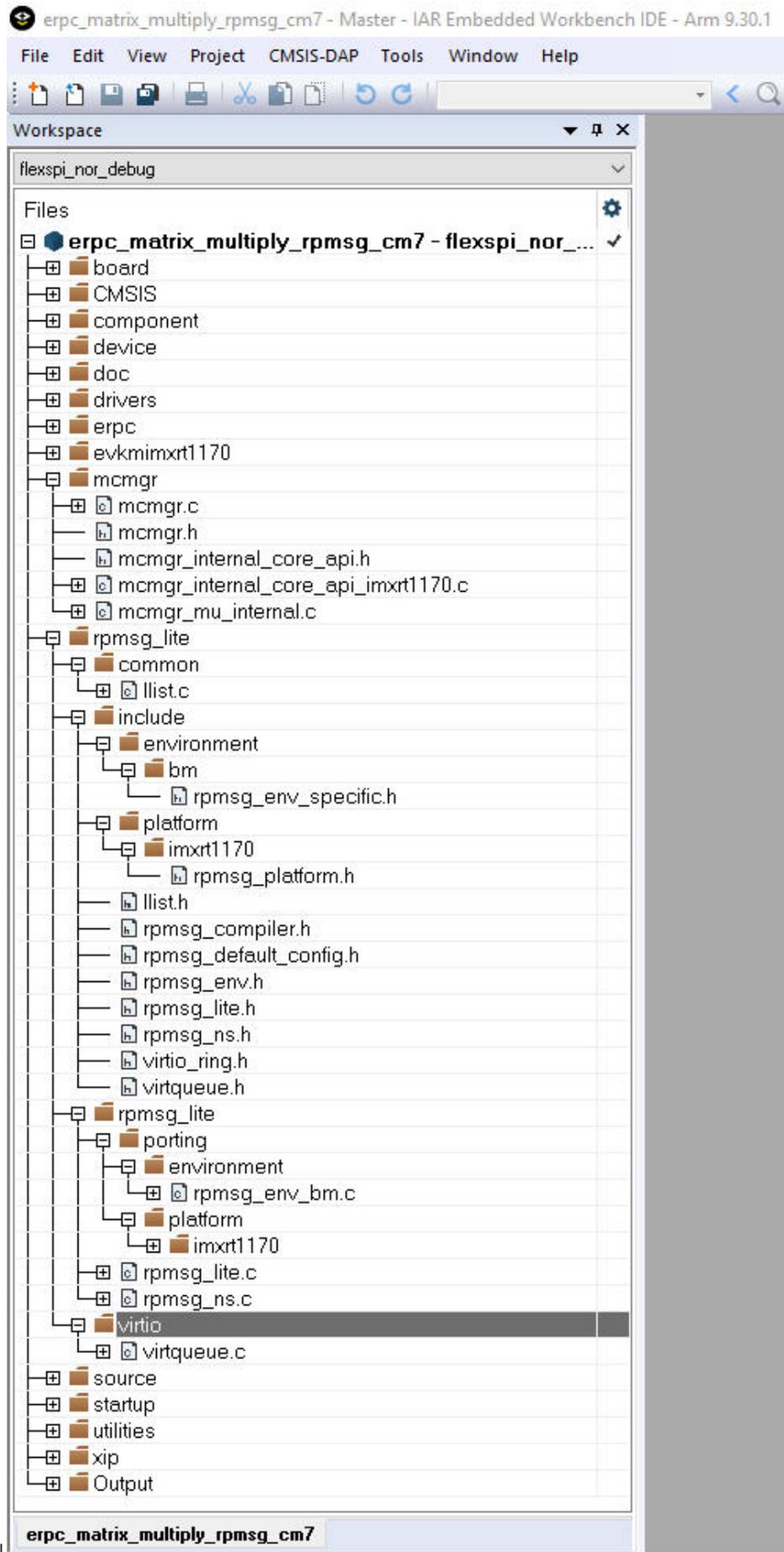
**Parent topic:**Multicore client application

**Client multicore infrastructure files** Because of the RMsg-Lite (transport layer), it is also necessary to include RMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



|  
**Parent topic:**Multicore client application

**Client user code** The client's user code is stored in the main\_core0.c file, located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_example/erpc\_matrix\_multiply\_rpmsg/cm7

The main\_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc\_error\_handler.h and erpc\_error\_handler.cpp files.

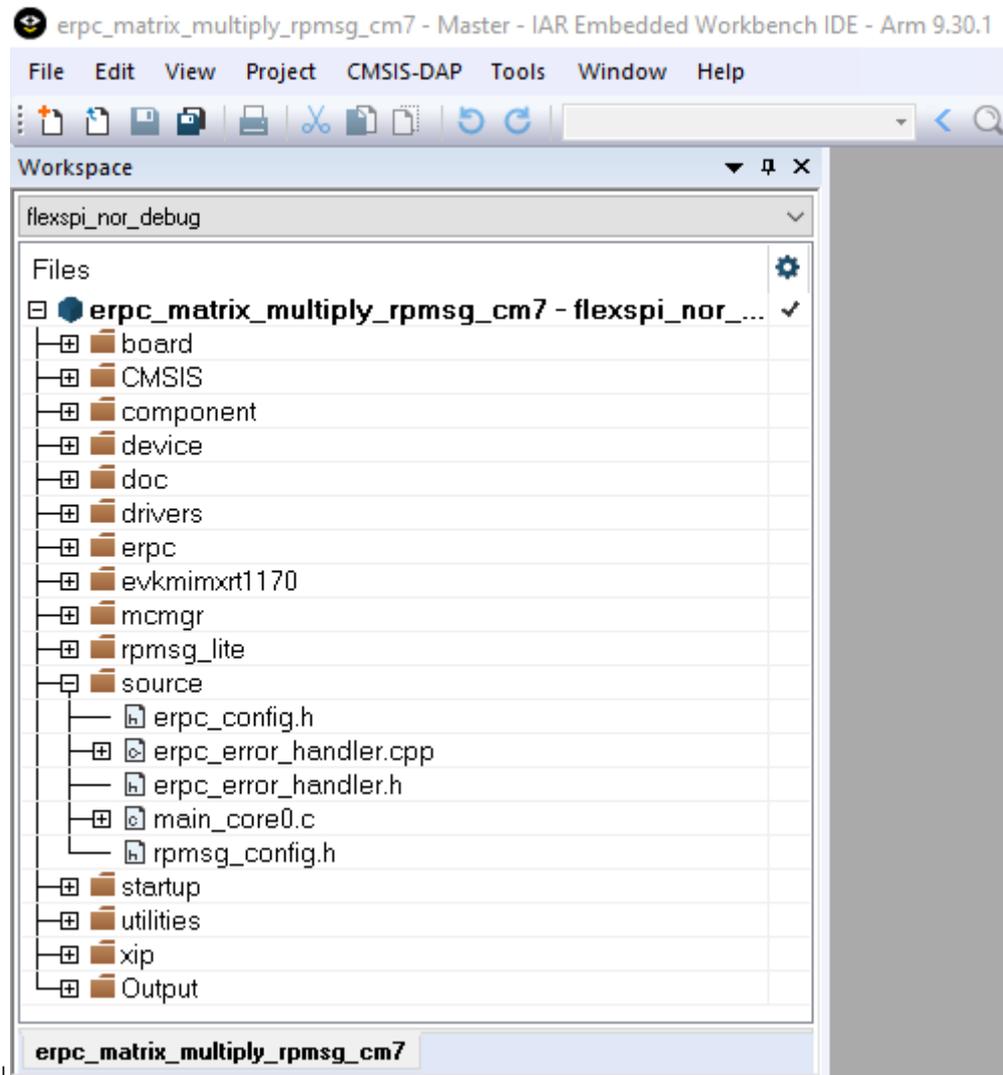
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
/* Invoke the erpcMatrixMultiply function */
erpcMatrixMultiply(matrix1, matrix2, result_matrix);
...
/* Check if some error occurred in eRPC */
if (g_erpc_error_occurred)
{
/* Exit program loop */
break;
}
...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_matrix\_multiply\_rpmsg



Parent topic:Multicore client application

Parent topic:[Create an eRPC application](#)

**Multiprocessor server application** The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPSMsg-Lite). The RPSMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.hpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.cpp

|

**Server user code** The server's user code is stored in the main\_server.c file, located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_server\_matrix\_multiply\_<transport\_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
 ...
}
int main()
{
 ...
 /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
 ↪operations */
 erpc_transport_t transport;
 transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_dynamic_init();
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server)
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}
```

**Parent topic:**Multiprocessor server application

**Multiprocessor client application** The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_client\_matrix\_multiply\_<transport\_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RMPMsg-Lite). The RMPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp
	<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp
	<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp
	<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp
	<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

**Client user code** The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

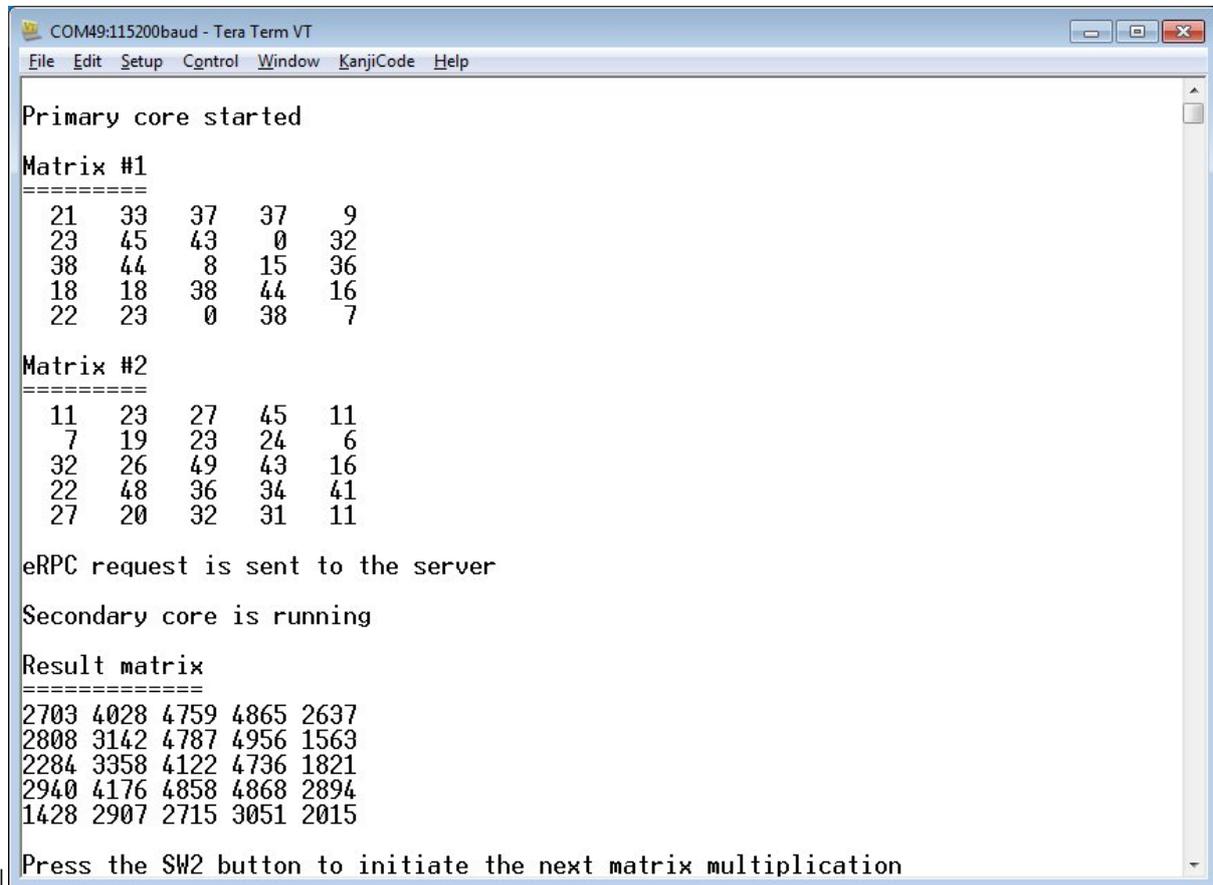
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver_
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
/* Invoke the erpcMatrixMultiply function */
erpcMatrixMultiply(matrix1, matrix2, result_matrix);
...
/* Check if some error occurred in eRPC */
if (g_erpc_error_occurred)
{
/* Exit program loop */
break;
}
...
}
```

**Parent topic:**Multiprocessor client application

**Parent topic:**Multiprocessor server application

Parent topic:[Create an eRPC application](#)

**Running the eRPC application** Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK\_install\_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



```

COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Primary core started
Matrix #1
=====
 21 33 37 37 9
 23 45 43 0 32
 38 44 8 15 36
 18 18 38 44 16
 22 23 0 38 7
Matrix #2
=====
 11 23 27 45 11
 7 19 23 24 6
 32 26 49 43 16
 22 48 36 34 41
 27 20 32 31 11
eRPC request is sent to the server
Secondary core is running
Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015
Press the SW2 button to initiate the next matrix multiplication

```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:[Create an eRPC application](#)

Parent topic:[eRPC example](#)

**eRPC example** This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The “Matrix multiply” application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC communicates with the eRPC server that runs on another SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

| Multicore application source and project files | `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore/`  
 | Multiprocessor application source and project files | `<MCUXpressoSDK_install_dir>/boards/<board_name>/multi`  
`<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr`  
 | |eRPC source files| `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/|` | RMsg-Lite  
 source files | `<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/|`

**Designing the eRPC application** The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has is declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

**Parent topic:** [eRPC example](#)

**Creating the IDL file** The created IDL file is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/`

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc\_matrix\_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix\_size* follows next. The *matrix\_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.
- As shown above, the function’s declaration contains three parameters of *Matrix* type: *matrix1* and *matrix2* are input parameters, while *result\_matrix* is the output parameter. Additionally, the returned data type is declared as *void*.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

**Parent topic:** [eRPC example](#)

**Using the eRPC generator tool** | Windows OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64`  
| Linux OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86`  
`<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86`  
| | Mac OS | `<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac` |

The files for the “Matrix multiply” example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (\*.erpc) is located; then run the following command:

```
erpcgen <IDL_file>.erpc
```

- In the “Matrix multiply” example, the command should look like:

```
erpcgen erpc_matrix_multiply.erpc
```

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- “-?”/”—help” – Shows supported commands.
- “-o <filePath>”/”—output<filePath>” – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen -o <path_to_output>
<path_to_IDL>/<IDL_file_name>.erpc
```

For the “Matrix multiply” example, when the command is executed from the default erpcgen location, it looks like:

```
erpcgen -o
../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service
../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu
```

In both cases, the following four files are generated into the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service` folder:

- erpc\_matrix\_multiply.h
- erpc\_matrix\_multiply\_client.cpp
- erpc\_matrix\_multiply\_server.h
- erpc\_matrix\_multiply\_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/service` folder.

**For Linux OS users:**

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as `./erpcgen...` instead of as `erpcgen ....`

Parent topic: [eRPC example](#)

**Create an eRPC application** This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
  1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
  2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
  3. Add infrastructure files.
  4. Add user code for client and server applications.
  5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

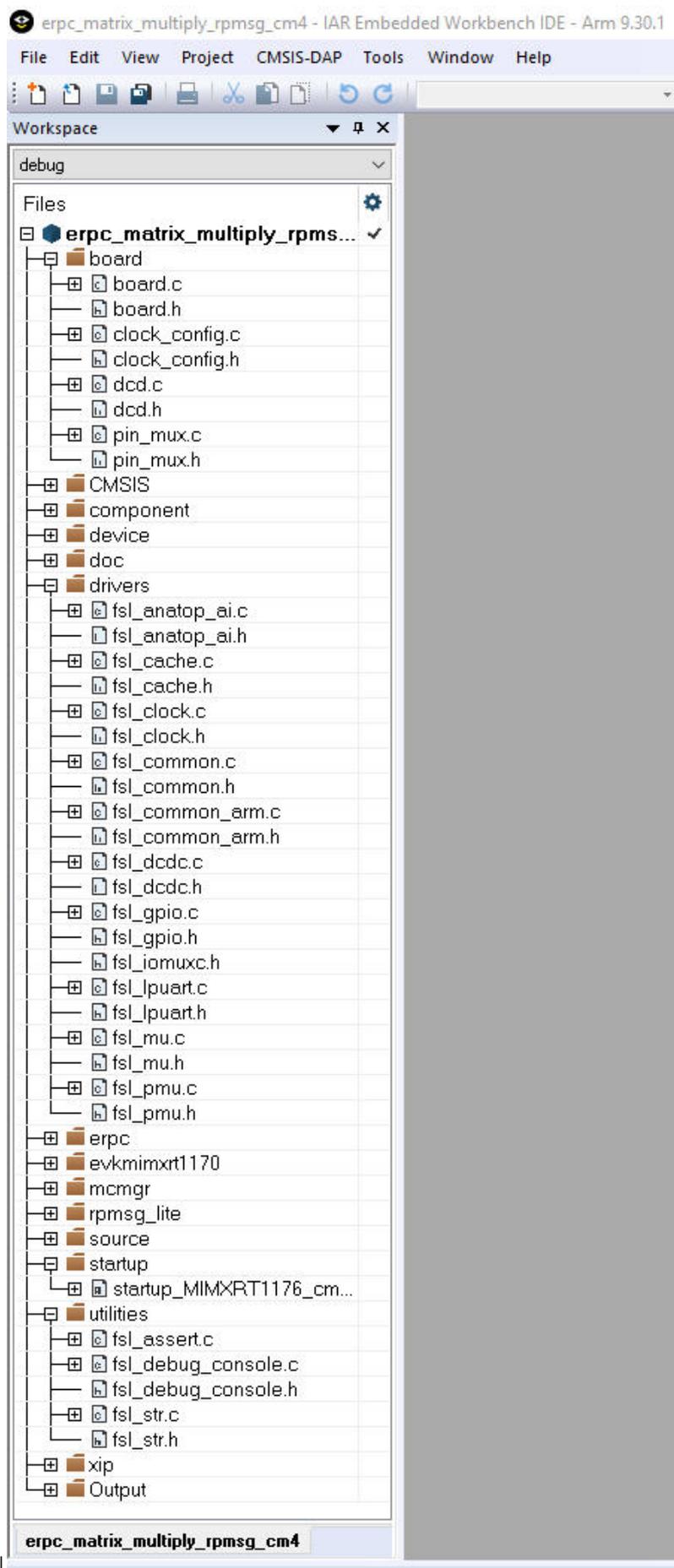
**Multicore server application** The “Matrix multiply” eRPC server project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmcg/cm4/iar/`

The project files for the eRPC server have the `_cm4` suffix.

**Server project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



|

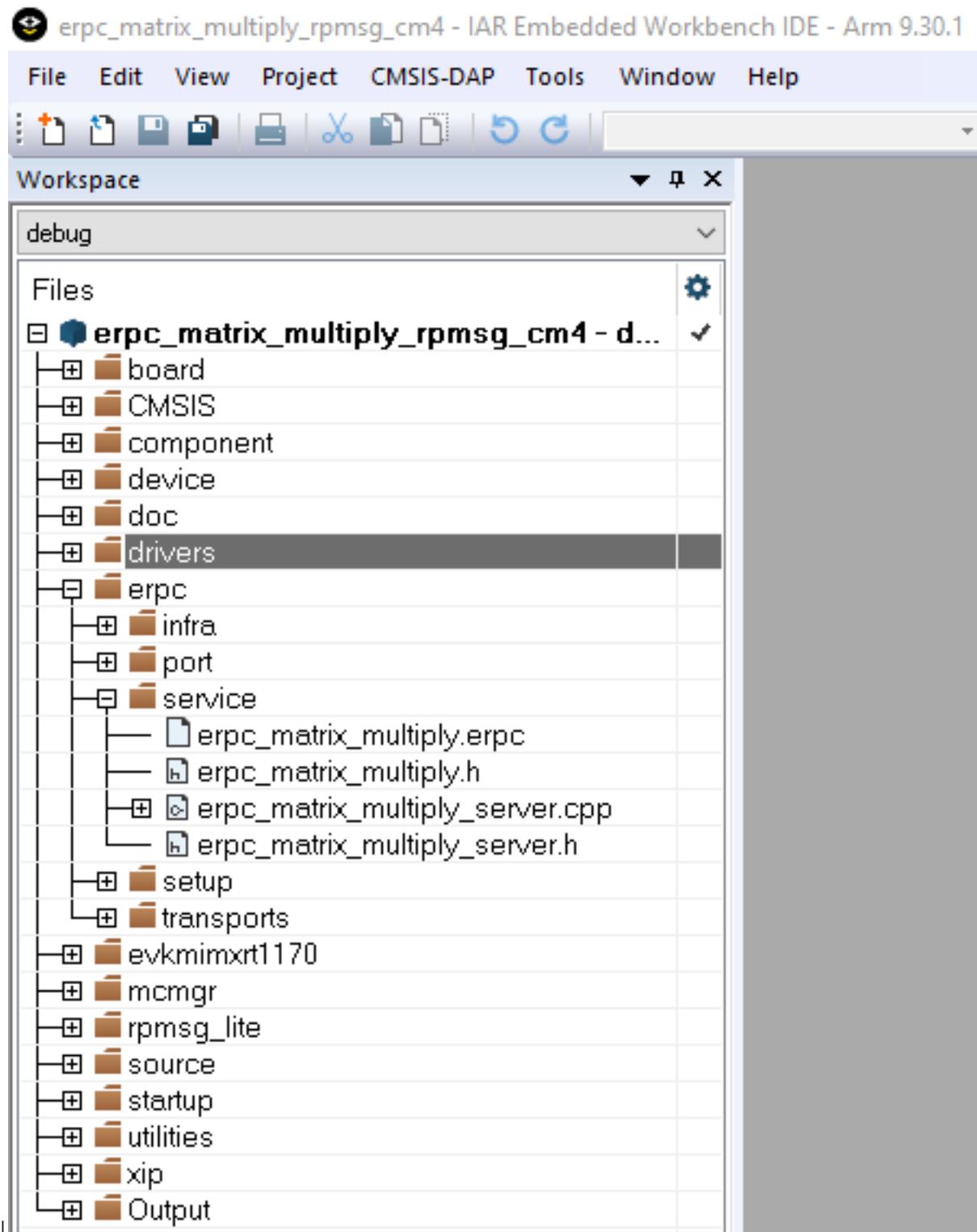
**Parent topic:**Multicore server application

**Server related generated files** The server-related generated files are:

- erpc\_\_matric\_\_multiply.h
- erpc\_\_matrix\_\_multiply\_\_server.h
- erpc\_\_matrix\_\_multiply\_\_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_common/erpc\_matrix\_multiply/



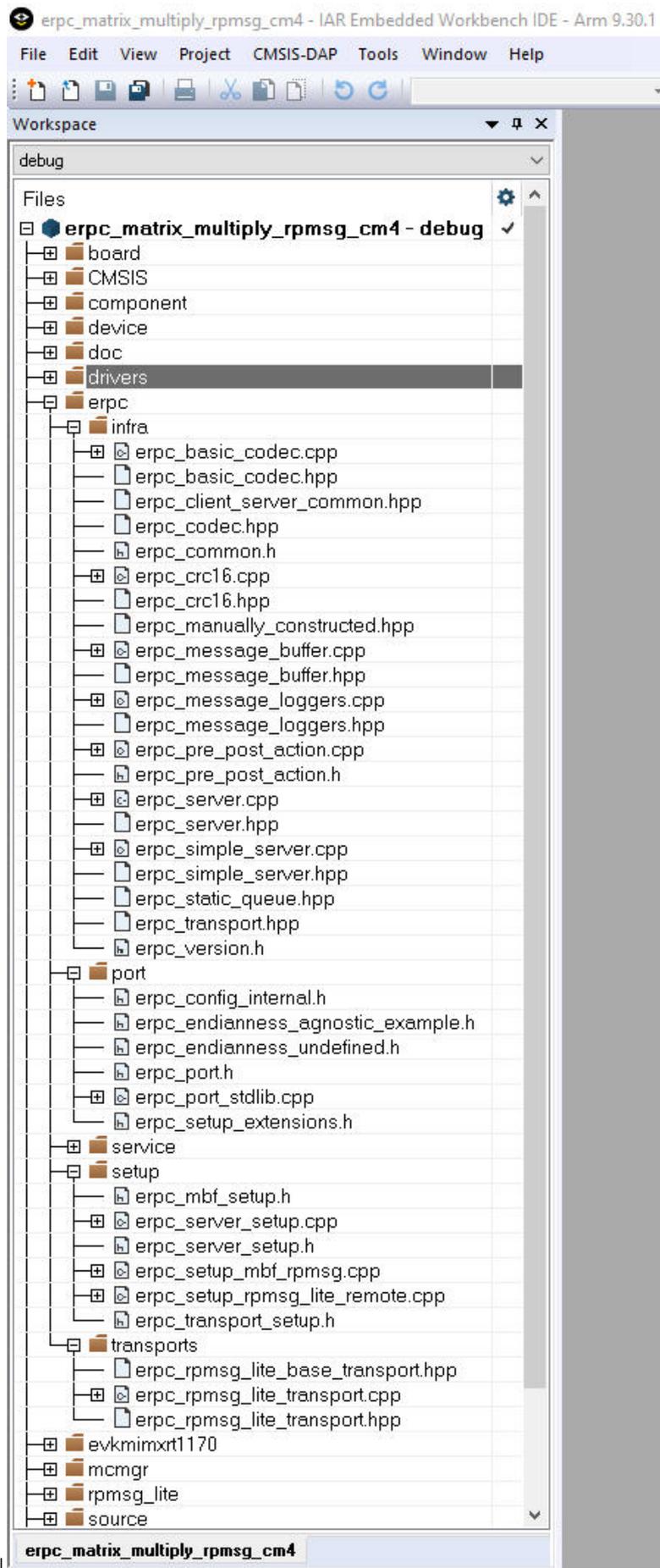
**Parent topic:**Multicore server application

**Server infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
  - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
  - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
  - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
  - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
  - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPLite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.



|

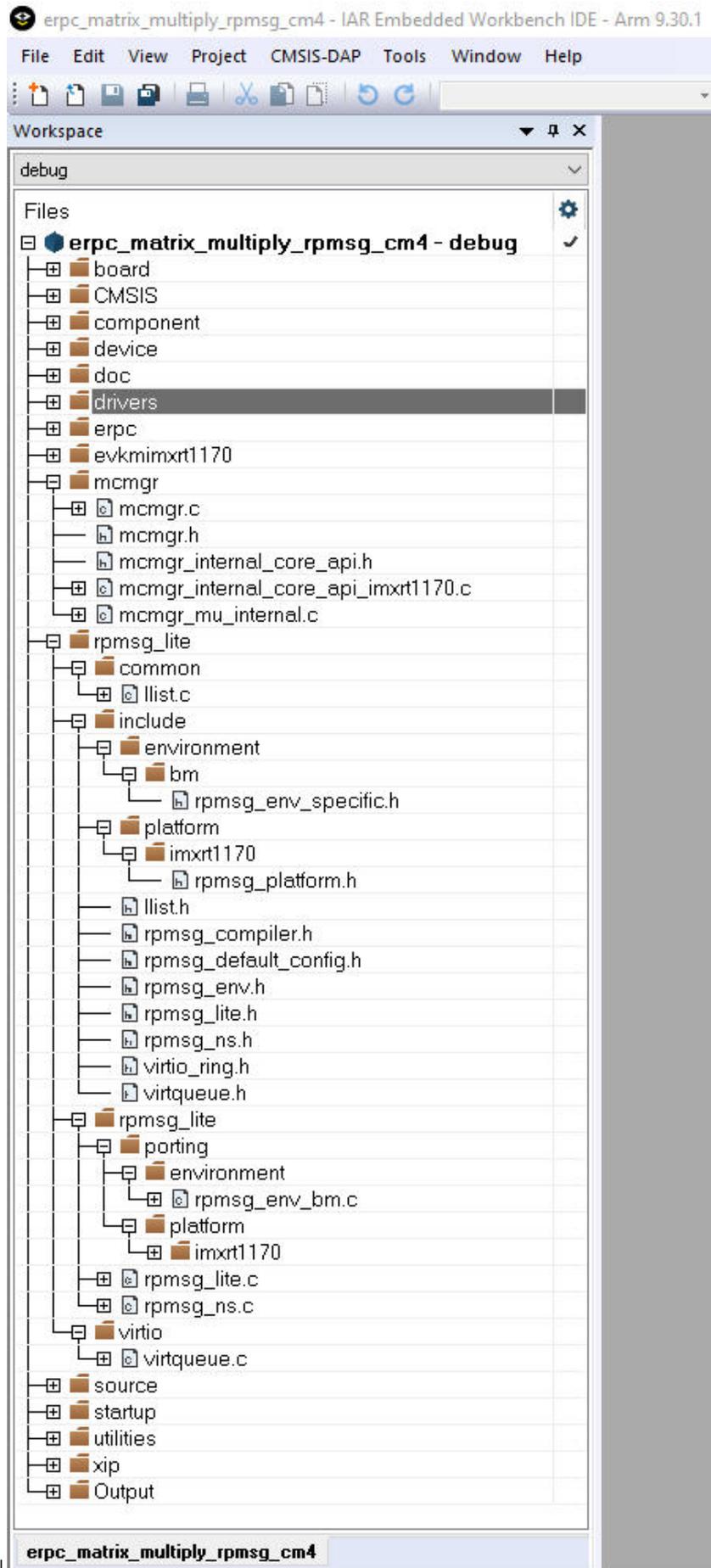
**Parent topic:**Multicore server application

**Server multicore infrastructure files** Because of the RPLite (transport layer), it is also necessary to include RPLite related files, which are in the following folder:

*<MCUXpressoSDK\_install\_dir>/middleware/multicore/rplite/*

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

*<MCUXpressoSDK\_install\_dir>/middleware/multicore/mcmgr/*



|

**Parent topic:**Multicore server application

**Server user code** The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

The `main_core1.c` file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

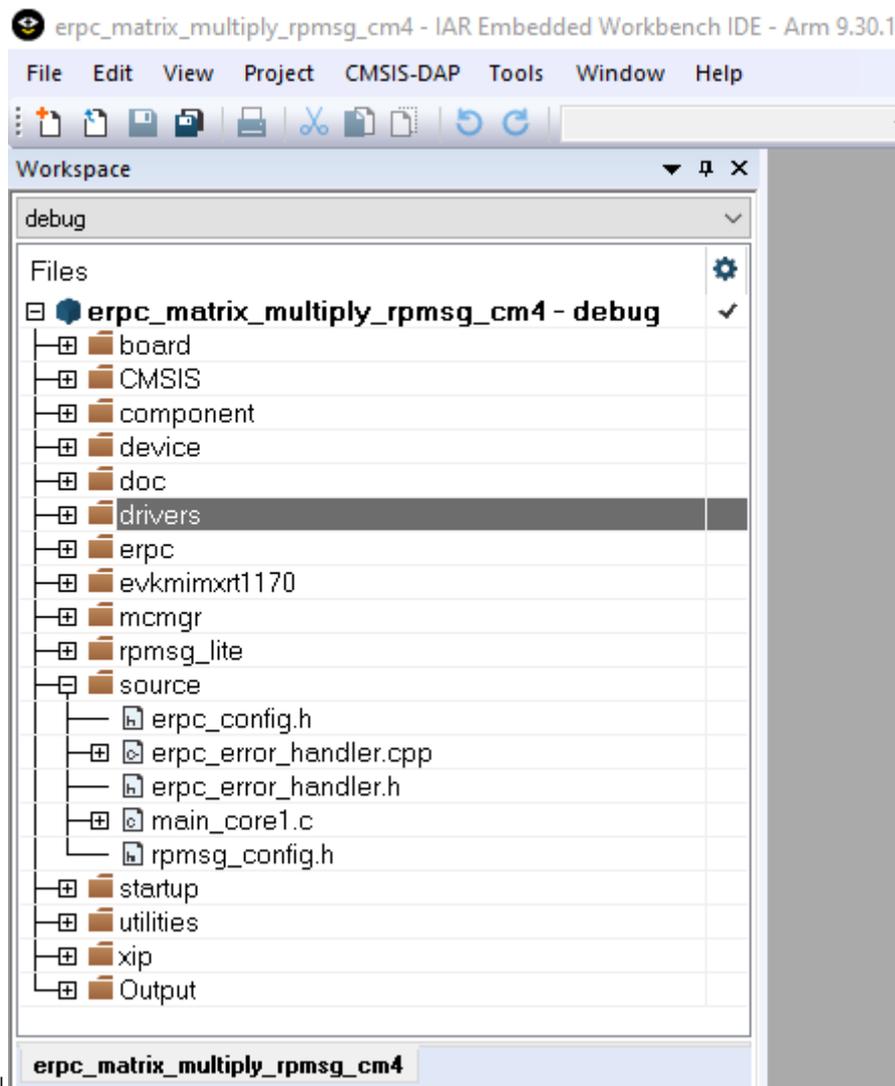
The eRPC-relevant code is captured in the following code snippet:

```

/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
 ...
}
int main()
{
 ...
 /* RPMsg-Lite transport layer initialization */
 erpc_transport_t transport;
 transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
 ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_rpmsg_init(transport);
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server);
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}

```

Except for the application main file, there are configuration files for the RPSMsg-Lite (`rpmsg_config.h`) and eRPC (`erpc_config.h`), located in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/ erpc_matrix_multiply_rpmsg` folder.



**Parent topic:**Multicore server application

**Parent topic:**[Create an eRPC application](#)

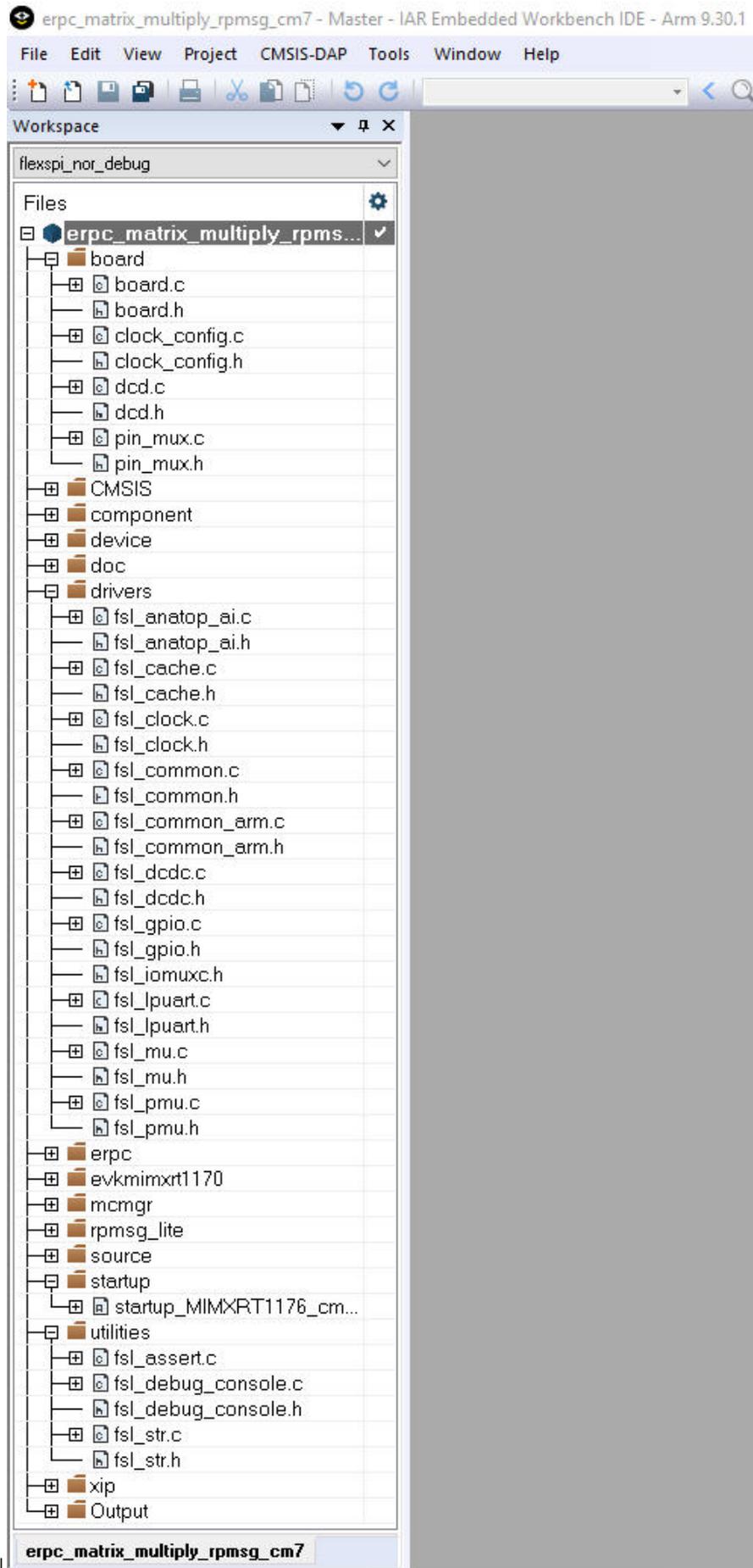
**Multicore client application** The “Matrix multiply” eRPC client project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar`

Project files for the eRPC client have the `_cm7` suffix.

**Client project basic source files** The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



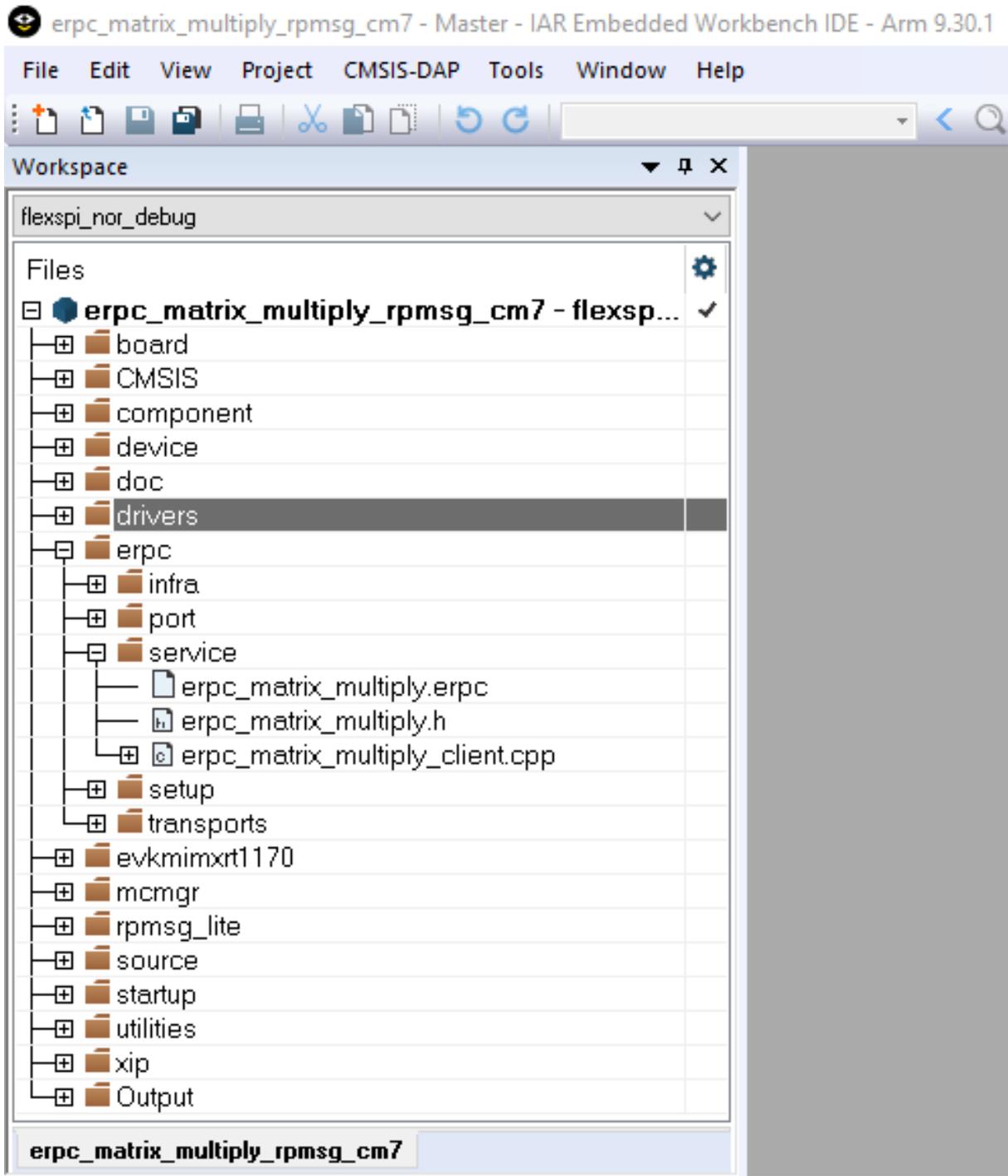
|

**Parent topic:**Multicore client application

**Client-related generated files** The client-related generated files are:

- erpc\_matrix\_multiply.h
- erpc\_matrix\_multiply\_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



**Parent topic:**Multicore client application

**Client infrastructure files** The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc\_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

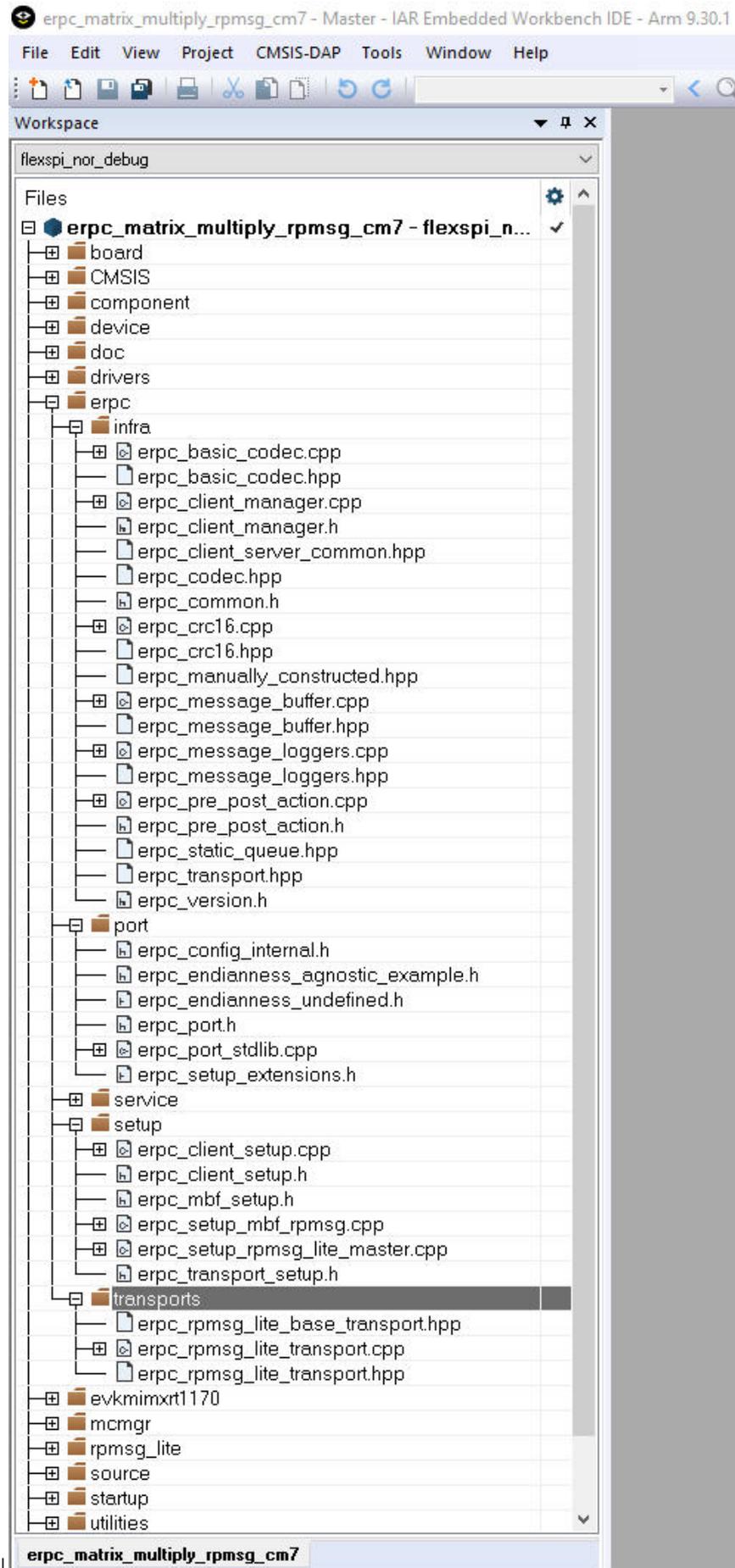
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files needs to be added into the client project.



|

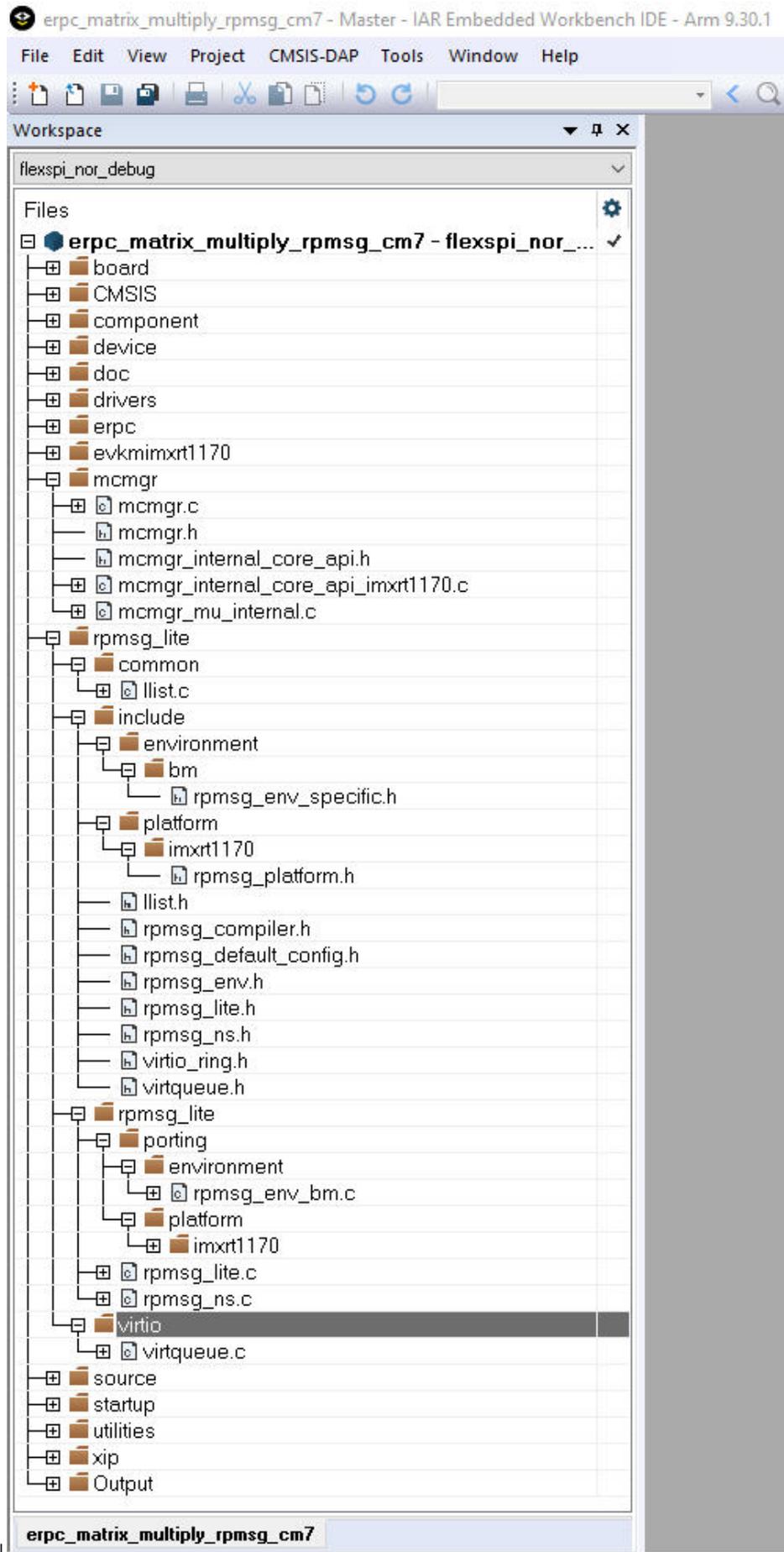
**Parent topic:**Multicore client application

**Client multicore infrastructure files** Because of the RMsg-Lite (transport layer), it is also necessary to include RMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



|  
**Parent topic:**Multicore client application

**Client user code** The client's user code is stored in the main\_core0.c file, located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_example/erpc\_matrix\_multiply\_rpmsg/cm7

The main\_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc\_error\_handler.h and erpc\_error\_handler.cpp files.

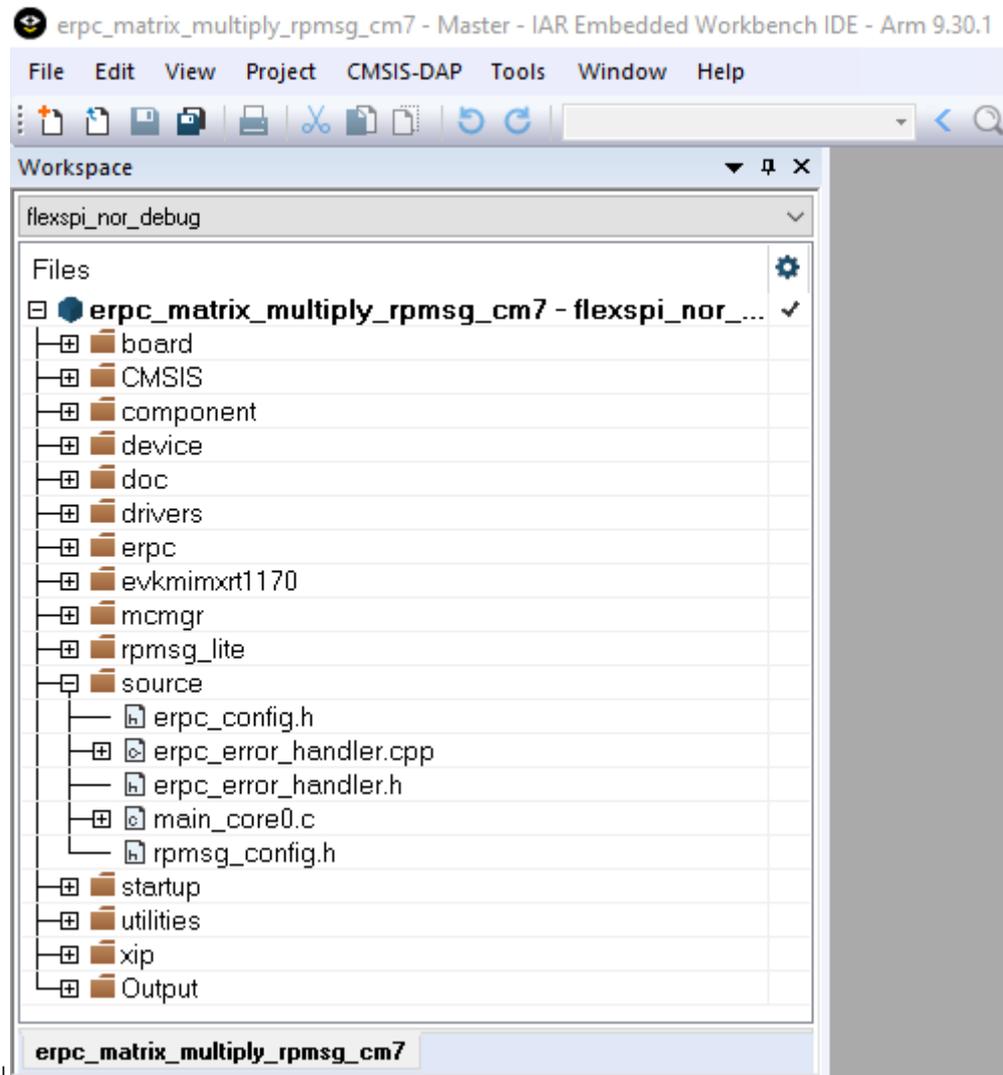
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPSMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
 /* Invoke the erpcMatrixMultiply function */
 erpcMatrixMultiply(matrix1, matrix2, result_matrix);
 ...
 /* Check if some error occurred in eRPC */
 if (g_erpc_error_occurred)
 {
 /* Exit program loop */
 break;
 }
 ...
}
```

Except for the application main file, there are configuration files for the RPSMsg-Lite (rpmsg\_config.h) and eRPC (erpc\_config.h), located in the following folder:

<MCUXpressoSDK\_install\_dir>/boards/evkmimxrt1170/multicore\_examples/erpc\_matrix\_multiply\_rpmsg



Parent topic:Multicore client application

Parent topic:[Create an eRPC application](#)

**Multiprocessor server application** The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPSMsg-Lite). The RPSMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.hpp

<eRPC base directory>/erpc\_c/transport/erpc\_uart\_cmsis\_transport.cpp

|

**Server user code** The server's user code is stored in the main\_server.c file, located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_server\_matrix\_multiply\_<transport\_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
 ...
}
int main()
{
 ...
 /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
 ↪operations */
 erpc_transport_t transport;
 transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
 ...
 /* MessageBufferFactory initialization */
 erpc_mbf_t message_buffer_factory;
 message_buffer_factory = erpc_mbf_dynamic_init();
 ...
 /* eRPC server side initialization */
 erpc_server_t server;
 server = erpc_server_init(transport, message_buffer_factory);
 ...
 /* Adding the service to the server */
 erpc_service_t service = create_MatrixMultiplyService_service();
 erpc_add_service_to_server(server, service);
 ...
 while (1)
 {
 /* Process eRPC requests */
 erpc_status_t status = erpc_server_poll(server)
 /* handle error status */
 if (status != kErpcStatus_Success)
 {
 /* print error description */
 erpc_error_handler(status, 0);
 ...
 }
 ...
 }
}
```

**Parent topic:**Multiprocessor server application

**Multiprocessor client application** The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK\_install\_dir>/boards/<board\_name>/multiprocessor\_examples/erpc\_client\_matrix\_multiply\_<transport\_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RMPMsg-Lite). The RMPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp
	<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp
	<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp
	<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp
	<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

**Client user code** The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver_
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
/* Invoke the erpcMatrixMultiply function */
erpcMatrixMultiply(matrix1, matrix2, result_matrix);
...
/* Check if some error occurred in eRPC */
if (g_erpc_error_occurred)
{
/* Exit program loop */
break;
}
...
}
```

**Parent topic:**Multiprocessor client application

**Parent topic:**Multiprocessor server application

Parent topic:[Create an eRPC application](#)

**Running the eRPC application** Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK\_install\_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:

```

COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Primary core started
Matrix #1
=====
 21 33 37 37 9
 23 45 43 0 32
 38 44 8 15 36
 18 18 38 44 16
 22 23 0 38 7
Matrix #2
=====
 11 23 27 45 11
 7 19 23 24 6
 32 26 49 43 16
 22 48 36 34 41
 27 20 32 31 11
eRPC request is sent to the server
Secondary core is running
Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015
Press the SW2 button to initiate the next matrix multiplication

```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:[Create an eRPC application](#)

Parent topic:[eRPC example](#)

**Other uses for an eRPC implementation** The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the [github.com/EmbeddedRPC](https://github.com/EmbeddedRPC).

**Note about the source code in the document** Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 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.

**Changelog eRPC** All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

## Unreleased

### Added

### Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.

## 1.14.0

### Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

### Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.

- erpc: Fixed Makefile, install the erpc\_simple\_server header, GitHub PR #447.
- erpc\_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

### 1.13.0

#### Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr\_uart, zephyr\_mbox).
- eRPC: Add new Zephyr-related examples.

#### Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

#### Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

### 1.12.0

#### Added

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in Message-Buffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

#### Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

### 1.11.0

#### Fixed

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.
- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU\_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSBSIO.GetNumPorts API use with libusbsio python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcopy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

### 1.10.0

#### Fixed

- eRPC: MU transport layer switched to blocking MU\_SendMsg() API use.

### 1.10.0

#### Added

- eRPC: Add TCP\_NODELAY option to python, GitHub PR #298.

## Fixed

- eRPC: MUPTransport adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.
- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.
- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT\_TRUE with EXPECT\_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg\_lite based transports to changed rpmsg\_lite\_wait\_for\_link\_up() API parameters.
- eRPC, erpcgen: Better distinguish which file can and cannot be linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc\_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest\_enumErrorCode\_allDirection, serialize enums as int32 instead of uint32.

### 1.9.1

## Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.

- eRPC: Remove ERPC\_PRE\_POST\_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcopy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid for serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.
- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg\_lite\_wait\_for\_link\_up() in rpmsg\_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.
- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

## 1.9.0

### Added

- eRPC: Allow used LIBUSB\_SIO device index being specified from the Python command line argument.

### Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run\_clang\_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g\_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc\_setup\_mbf\_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related API, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

## 1.8.1

### Added

- eRPC: New i2c\_slave\_transport transport introduced.

### Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message\_loggers and pre\_post\_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.
- erpcgen: Fix bison file, GitHub PR #156.

### 1.8.0

### Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char\*, GitHub PR #125.

### Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native \_WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix\_multiply.py example: Add -serial and -baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc\_framed\_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '\$' character from generated symbol name in '\_\$union' suffix, GitHub PR #103.

- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

#### 1.7.4

##### Added

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

##### Fixed

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

#### 1.7.3

##### Fixed

- eRPC: Improved the test\_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

#### 1.7.2

##### Added

- eRPC: Add missing doxygen comments for transports.

##### Fixed

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.

### 1.7.1

#### Fixed

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

### 1.7.0

#### Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.
- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

#### Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

### 1.6.0

#### Added

- eRPC: Added @nullable support for scalar types.

#### Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

### 1.5.0

### Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.
- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

### 1.4.0

#### Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

#### Fixed

- eRPC: win\_flex\_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

### [1.3.0]

#### Added

- eRPC: New annotation types introduced (@length, @max\_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

### [1.2.0]

#### Added

- eRPC source directory organization changed.
- Many eRPC improvements.

#### [1.1.0]

##### Added

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

#### [1.0.0]

##### Added

- Initial Release

## 3.3 Wireless

### 3.3.1 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**

#### [1.10.12] - 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 Coded 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

- **Encrypted Advertising Data** support in Extended 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 **nep\_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 ( <i>ble_constants.h</i> )	Description
<b>pBondHeader:</b> points to a <code>bleBondIdentityHeaderBlob_t</code> element	<code>gBleBondIdentityHeaderSize_c</code>	Bonding information which is sufficient to identify a bonded device.
<b>pBondDataDynamic:</b> points to a <code>bleBondDataDynamicBlob_t</code> element	<code>gBleBondDataDynamicSize_c</code>	Bonding information that might change frequently.
<b>pBondDataStatic:</b> points to a <code>bleBondDataStaticBlob_t</code> element	<code>gBleBondDataStaticSize_c</code>	Bonding information that is unlikely to change frequently.
<b>pBondDataLegacy:</b> points to a <code>bleBondDataLegacyBlob_t</code> element	<code>gBleBondDataLegacySize_c</code>	Stores legacy pairing and Connection Signature Resolving Key (CSRK) bond information.
<b>pBondDataDeviceInfo:</b> points to a <code>bleBondDataDeviceInfoBlob_t</code> element	<code>gBleBondDataDeviceInfoSize_c</code>	Additional bonding information that can be accessed using the host stack API.
<b>pBondDataDescriptor:</b> points to a <code>bleBondDataDescriptorBlob_t</code> element	<code>gBleBondDataDescriptorSize_c</code>	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, GATTDDB)

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
);
```

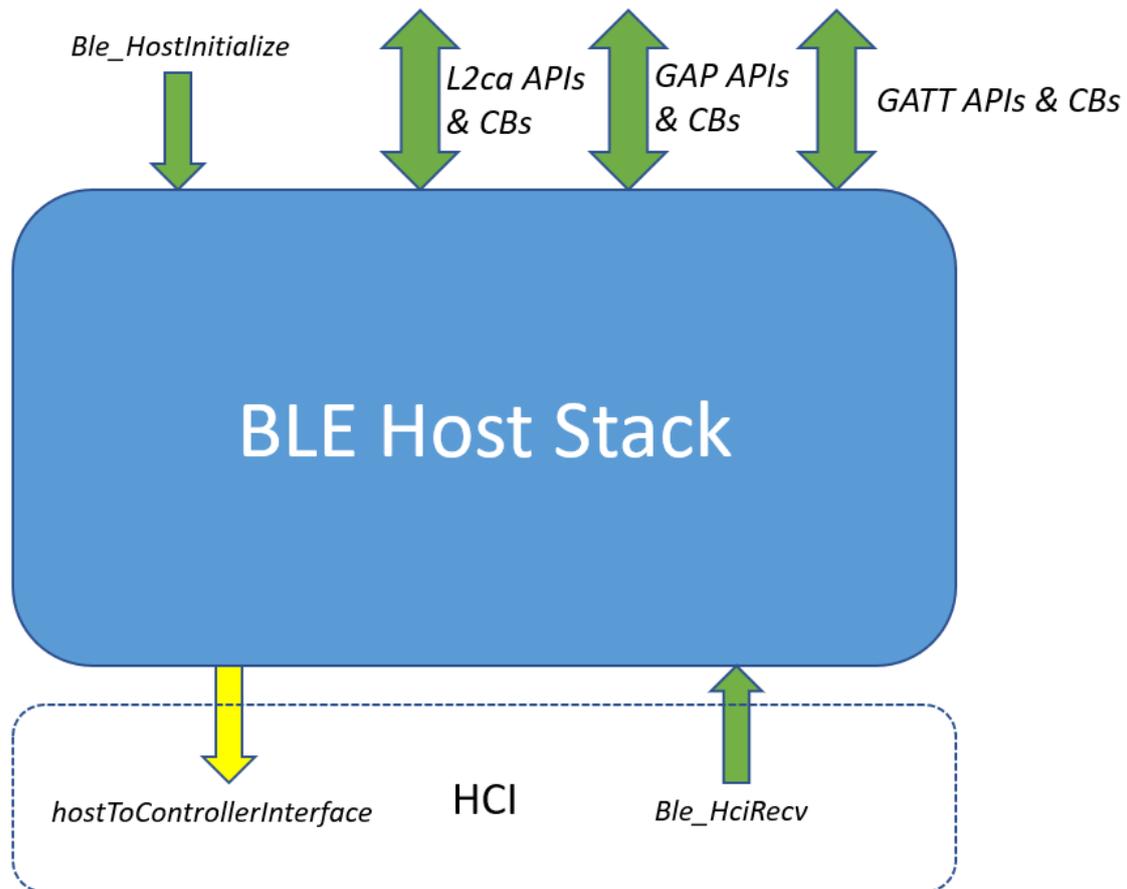
*BluetoothLEHost\_Init* takes as parameter a function to be called at the end of Bluetooth LE Host Stack initialization. In this callback, the application can register its callbacks with the Host layer, allocate timers, start services, and perform similar tasks. The callback should have a prototype as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

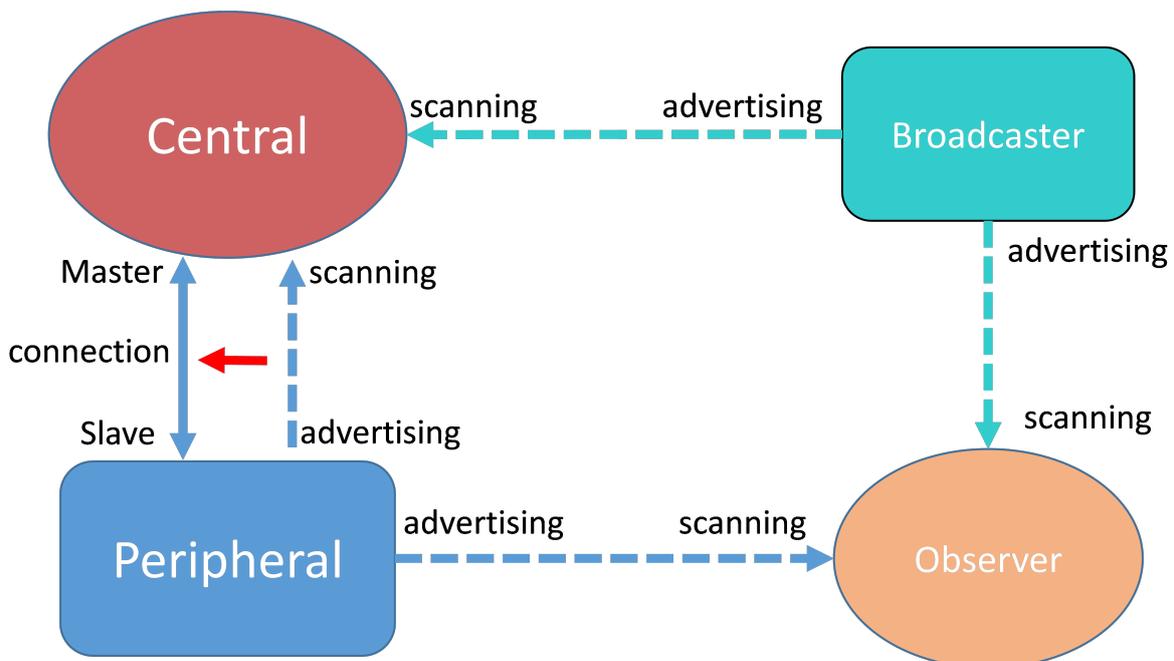
**Parent topic:** *Bluetooth LE Host Stack Initialization and APIs*

**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`).

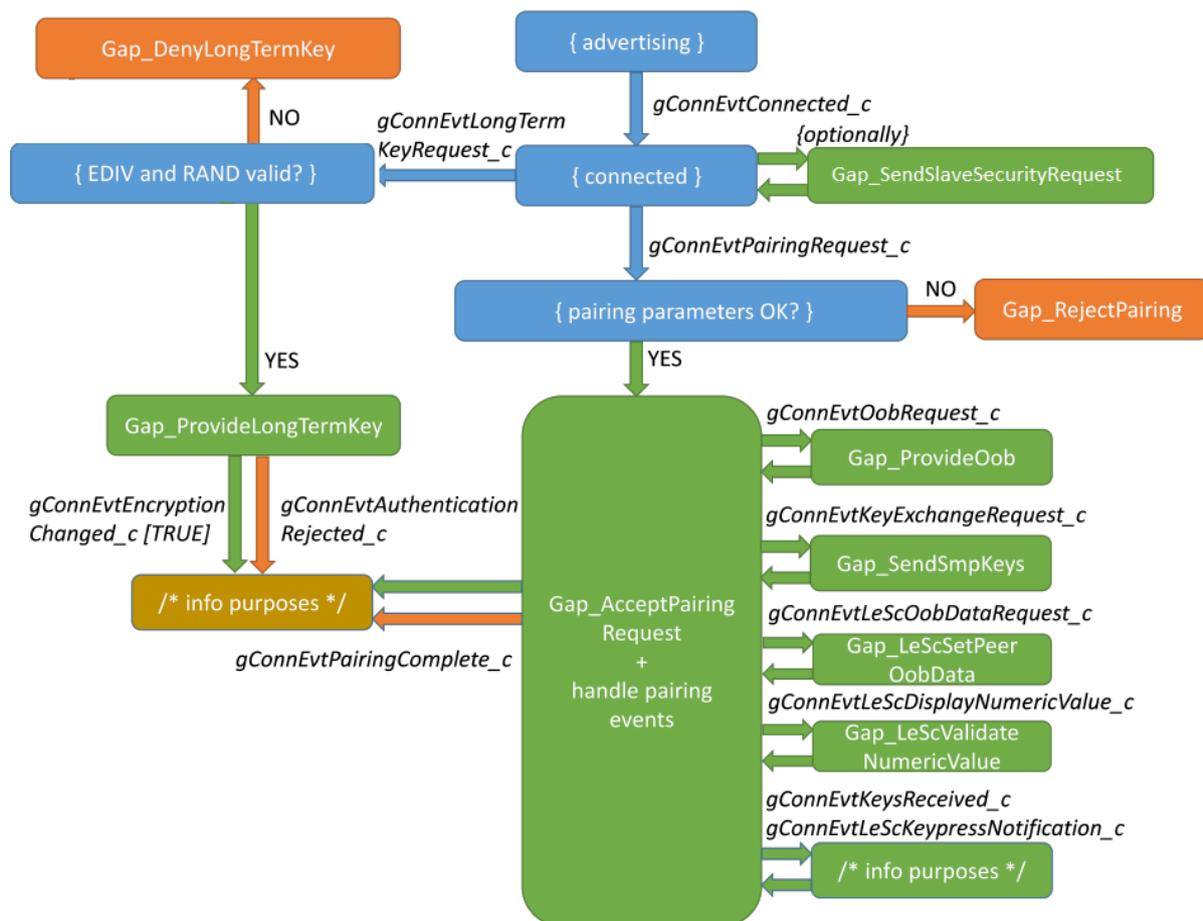
If the Peripheral application never calls `Gap_SetLocalPasskey`, then the `gConnEvtPasskeyRequest_c` event is sent to the application as usual.

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,
 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(&scanningParameters, 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 ;
 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,
 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, \
```

(continues on next page)

(continued from previous page)

```

/* 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;
 FLlib_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 im-

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 ;
 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
<b>Mode 1 (encryption)</b> distributed LTK (EDIV+RAND) or generated LTK	<b>Level 1</b> no security	<b>Level 2</b> unauthenticated encryption	<b>Level 3</b> authenticated encryption	<b>Level 4</b> LE SC authenticated encryption
<b>Mode 2 (data signing)</b> distributed CSRK	—	<b>Level 1</b> unauthenticated data signing	<b>Level 2</b> 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**

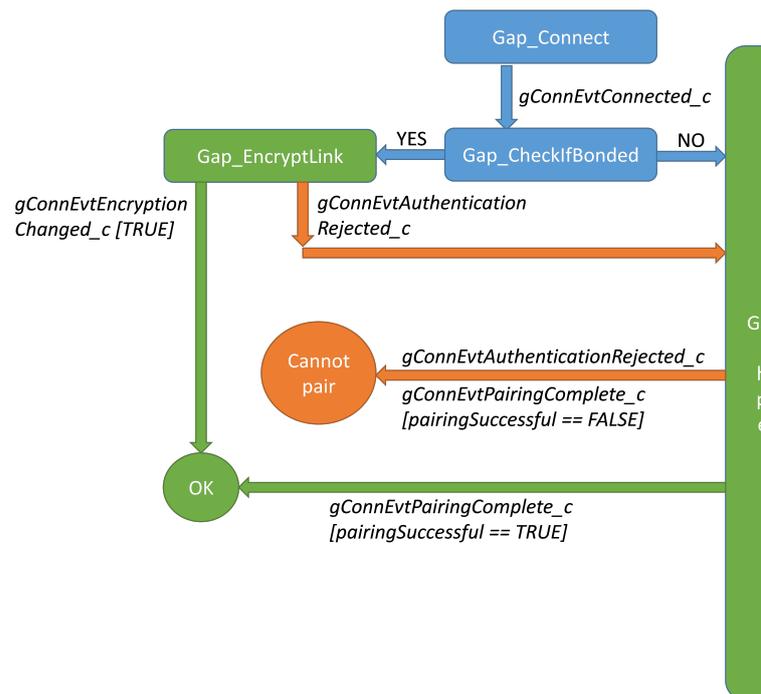
- *gConnEvtPasskeyRequest\_c*: a PIN is required for pairing; the application must respond with the *Gap\_EnterPasskey(deviceId, passkey)*.
- *gConnEvtOobRequest\_c*: if the pairing started with the *oobAvailable* set to *TRUE* by both sides; the application must respond with the *Gap\_ProvideOob(deviceId, 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

```

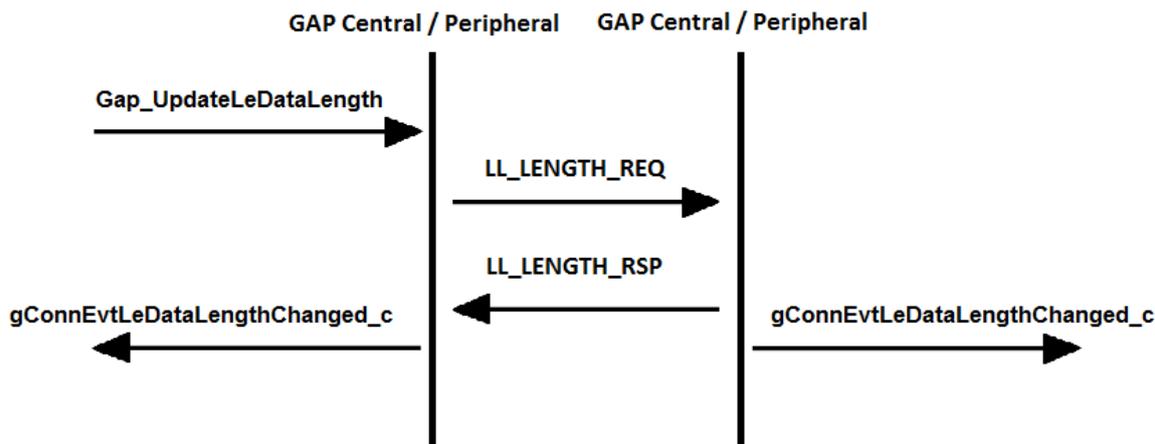
The device can update the data length anytime, while in connection. The function that triggers this mechanism is the following:

```

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 *gGap-ControllerPrivacyTimeout* 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\_GetBondedDevicesIdentityInformationAPI*.

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 is not 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. If aLtk 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-addressType	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
deviceId	deviceId_t	Device id of the peer, valid for connection events
rss	int8_t	RSSI, valid for RX event types
channel	uint8_t	Channel, valid for connection event over or Rx/Tx events
ce_count	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 $\mu$ s 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

- **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**

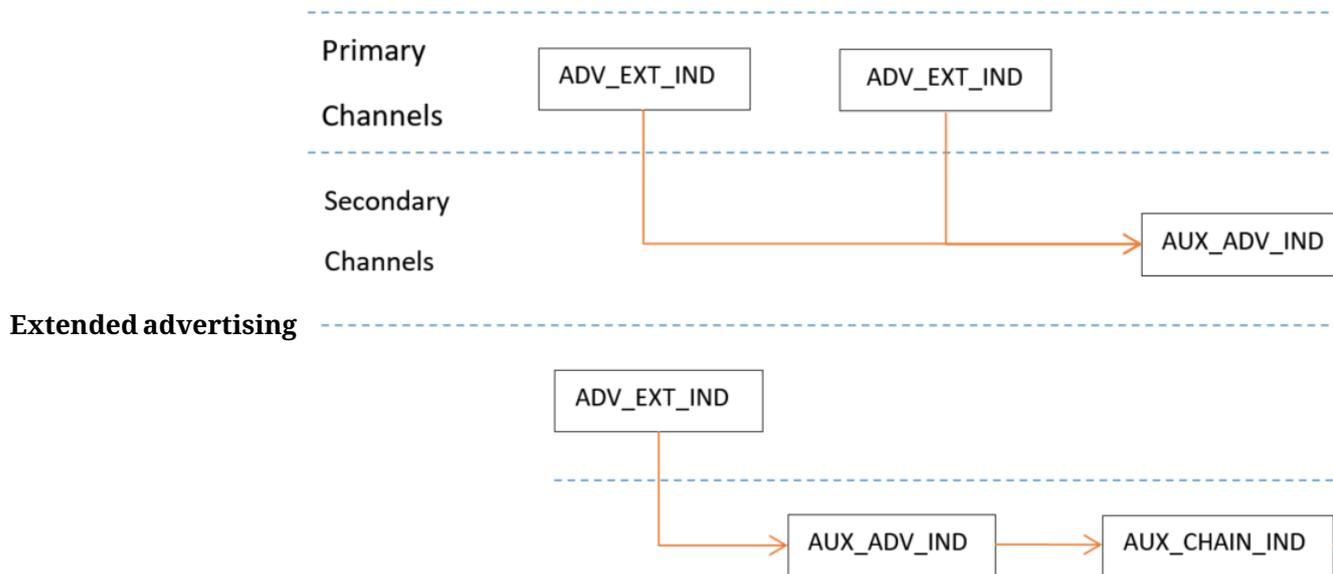
Event Data	Data type	Data Description
event_type	bleNotificationEvent_t	Enhanced notification event type
conn_handle	uint16_t	Connection handle of the peer, valid for connection events
rssi	int8_t	RSSI, valid for RX event types
channel_index	uint8_t	Channel, valid for connection event over or Rx/Tx events
conn_ev_counter	uint16_t	Connection event counter, valid for connection events only
timestamp	uint16_t	Timestamp in 625 $\mu$ s 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 – 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 advertising 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 handle	Value of the Advertising SID subfield in the ADI field of the PDU. Used to identify an advertising set. Possible values are 0x00 or 0x01 since the current implementation supports two advertising sets.
extAdvPropert	BIT0 - Connectable advertising
extAdvPropert	BIT1 - Scannable advertising
extAdvPropert	BIT2 - Directed advertising
extAdvPropert	BIT3 - High Duty Cycle Directed Connectable advertising (□3.75 ms Advertising Interval)
extAdvPropert	BIT4 - Use legacy advertising PDUs
extAdvPropert	BIT5 - Omit advertiser's address from all PDUs ("anonymous advertising")
extAdvPropert	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).
txPower	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
secondaryPHY	PHY for AUX_ADV_IND and periodic advertising: LE 1 M, LE 2 M or LE Coded. Ignored for legacy advertising
secondaryMaxS	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) or pScanResponseData (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.

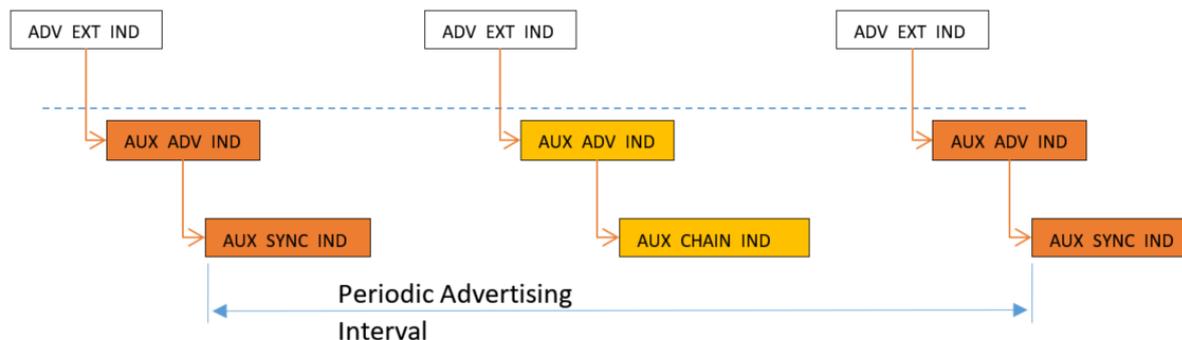
**Parent topic:**Extended advertising

**Parent topic:**[Generic Access Profile \(GAP\) Layer](#)

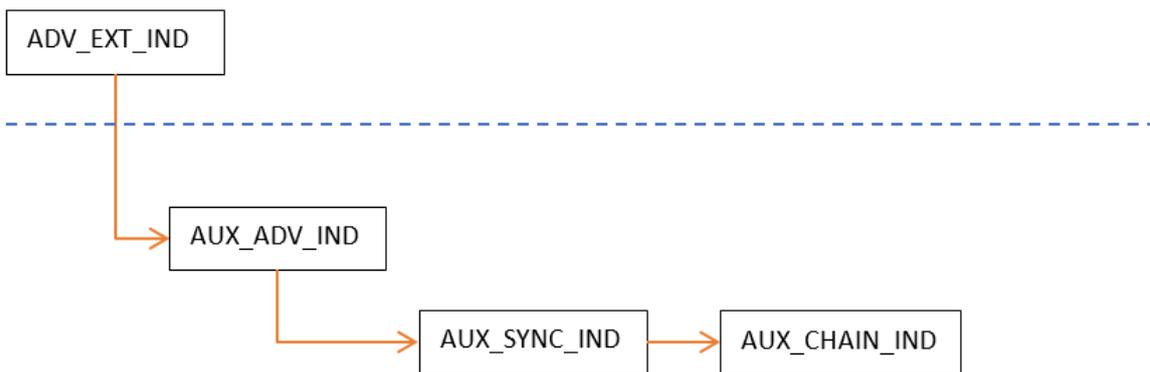
**Periodic Advertising** Periodic channels are used for periodic broadcast between unconnected devices. A periodic channel is represented by a channel map and a set of hopping and timing parameters.

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_cevent` 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_cevent` 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_cevent` 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 of `gEnhancedBearerSuspendedNoLocalCredits_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 Un-enhanced 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 `mpps` 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\_cconnection* 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 identifying 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 Characteristic 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 to 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]);
 deviceIdNameLength -= 1;
 /* Read characteristic value handle */
 PRINT(aValue[1] | (aValue[2] << 8));
 deviceIdNameLength -= 2;
 /* Read value length */
 PRINT(deviceNameLength);
 /* Read data */
 for (uint8_t j = 0; j < deviceIdNameLength; 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.pValue* 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 procedureType,
 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 set 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
↪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 following 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 known 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* if 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 defined 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 `gGattDb-Dynamic_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)*

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:

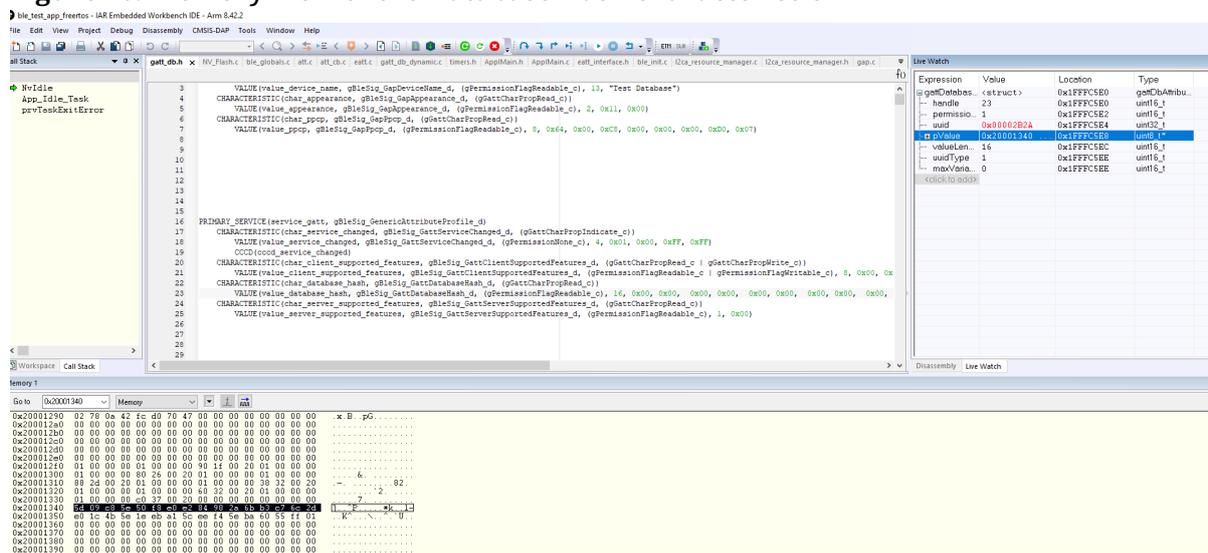
- The client receives and confirms a Service Changed Indication.

- 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 *\*Ble-ServDisc\_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,0x80,0x00,0x00,0x80,0x00,0x10,0x00,0x02,0x00,0x00,0x00,0x00,0x00)
```

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 ;

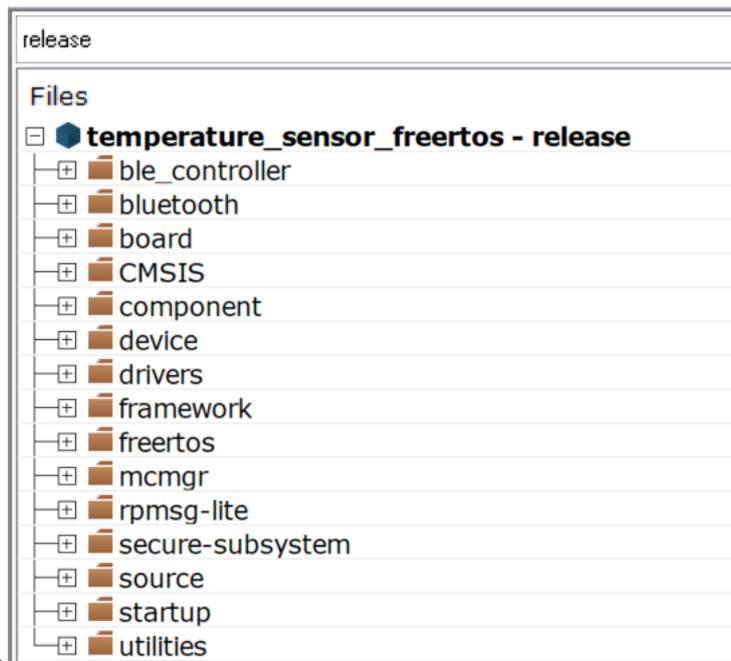
```

**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.

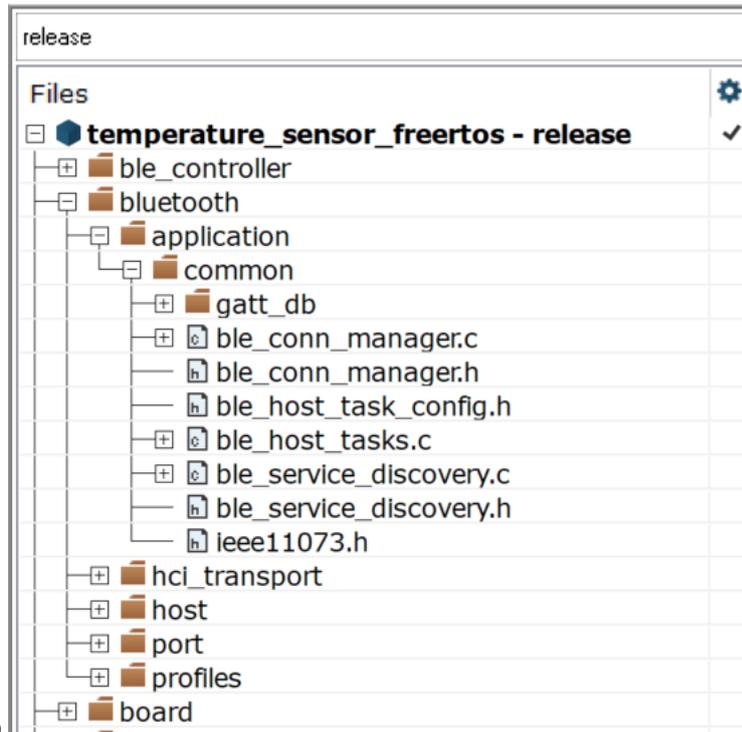
**Folder structure** The Figure shows the application folder structure.



#### 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

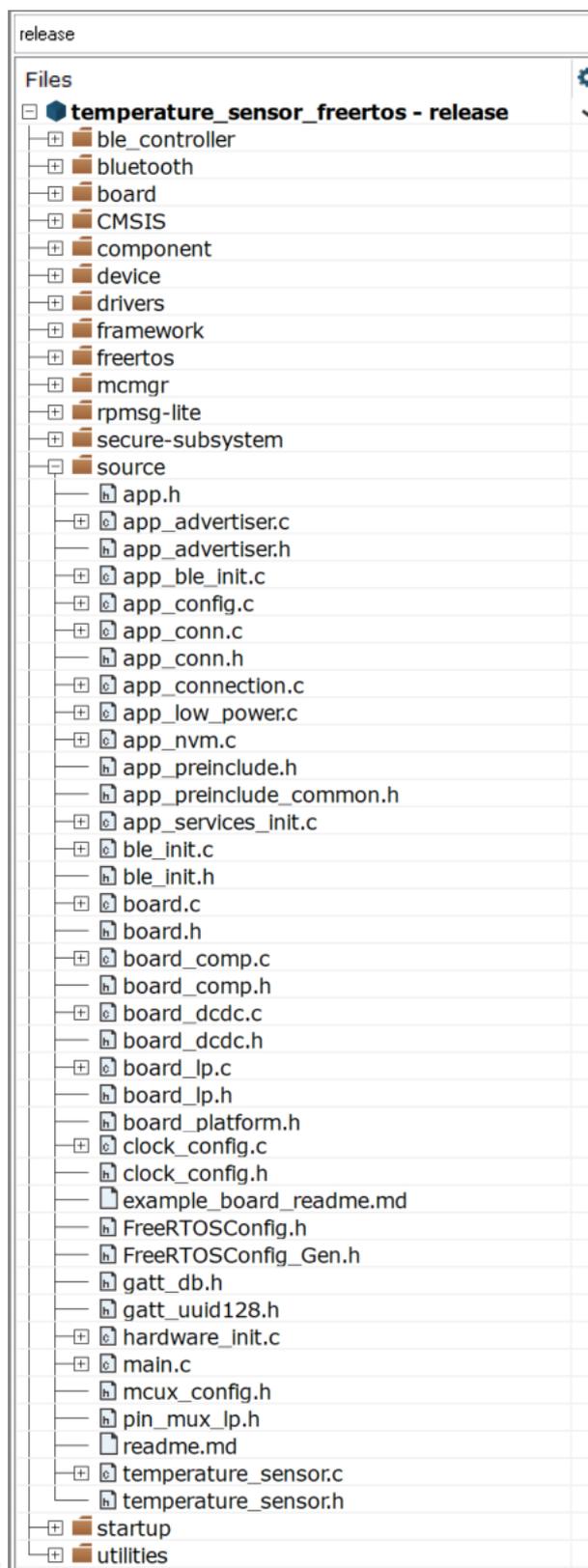


### Common Group

- 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.



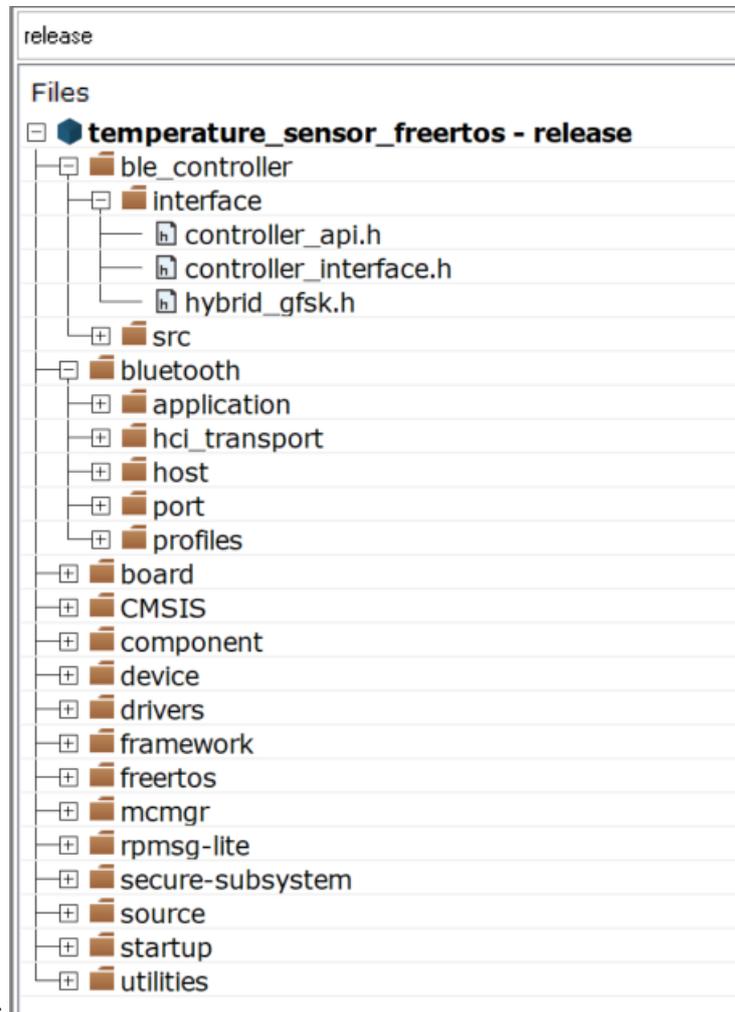
The *ble\_controller* folder/group contains:

- 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.
- *ble\_controller/profiles* contains profile-specific code; it is used by each demo application of stan-

standard 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 *p
↪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 `gattDatabasearray` 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.

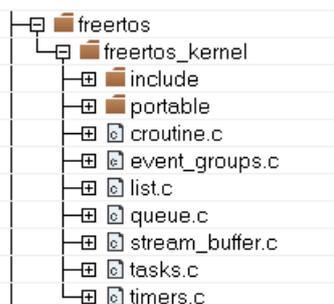
To complete the GATT database initialization, this demo application includes the required `gatt_db.h` and `gatt_uuid128.h` files in its specific application folder, along with other profile-specific configuration and code files.

**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.

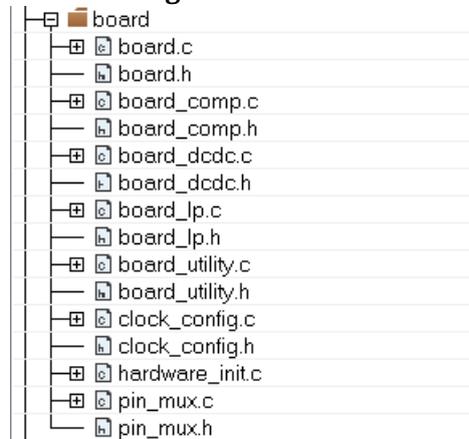
Reusing these files is recommended because they perform all the necessary RTOS-related work. The application developer must only modify the macros from *\*\_config.h* files whenever tasks need a bigger stack size or different priority settings. The new values should be overridden in the *app\_preinclude.h* 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.

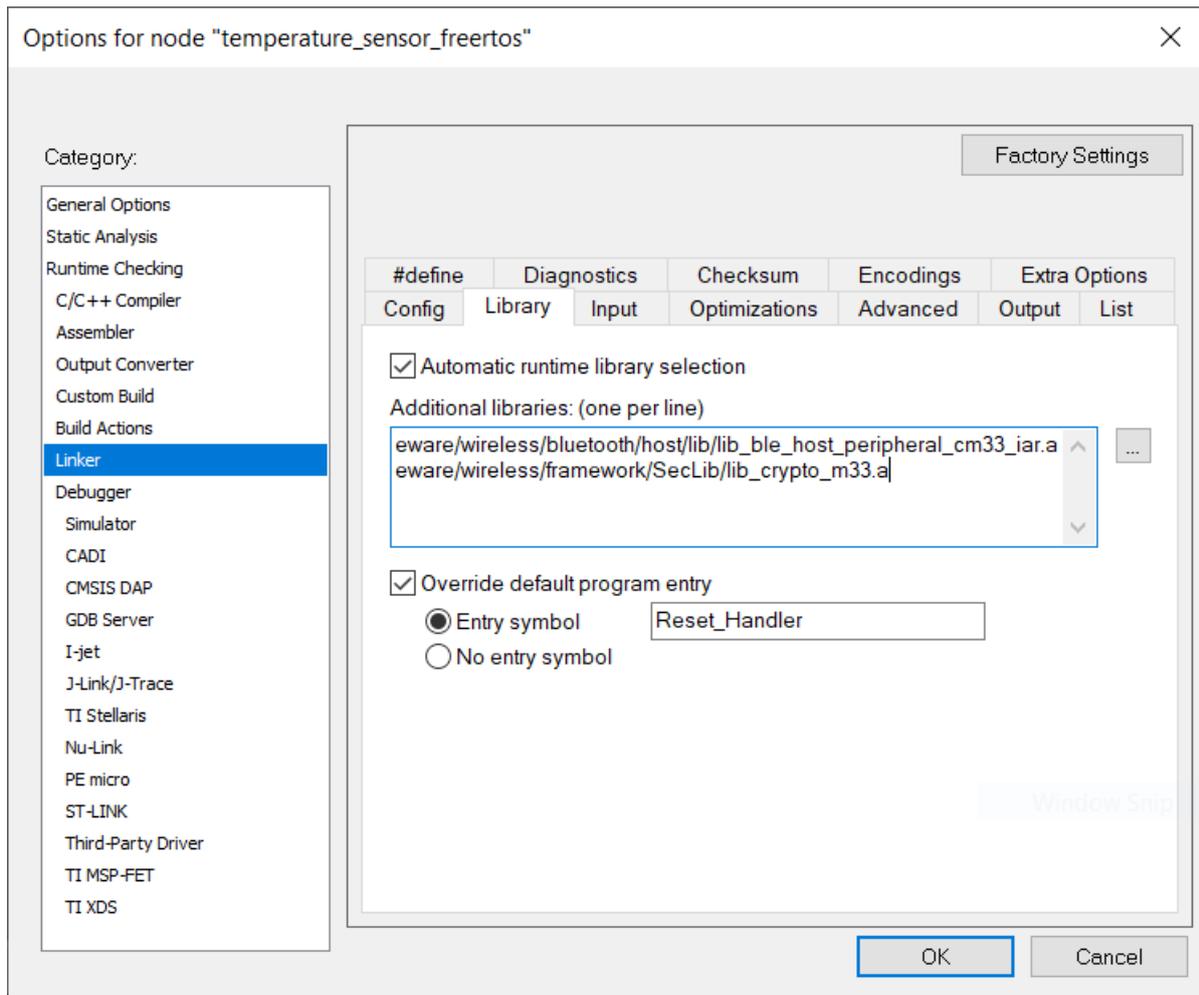
Step-by-step analysis is provided below:

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](#)

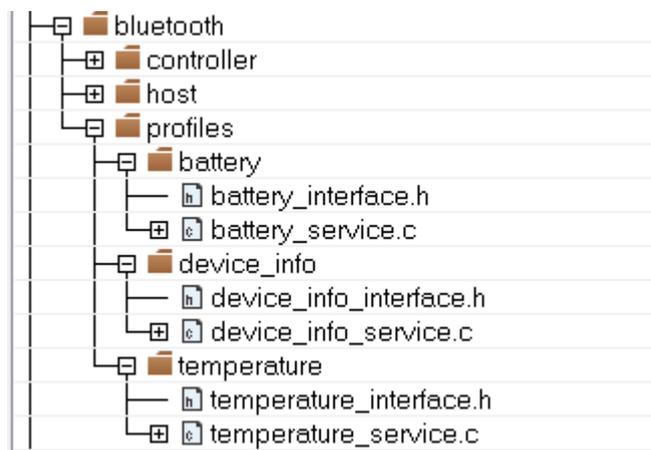
**Bluetooth Low Energy Host Stack configuration** The Bluetooth LE Host Stack libraries are found in the `middleware/wireless/bluetooth/host/lib` folder. The user should add the best matching library for its use case to the linker options of its project.

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();
}
}
}

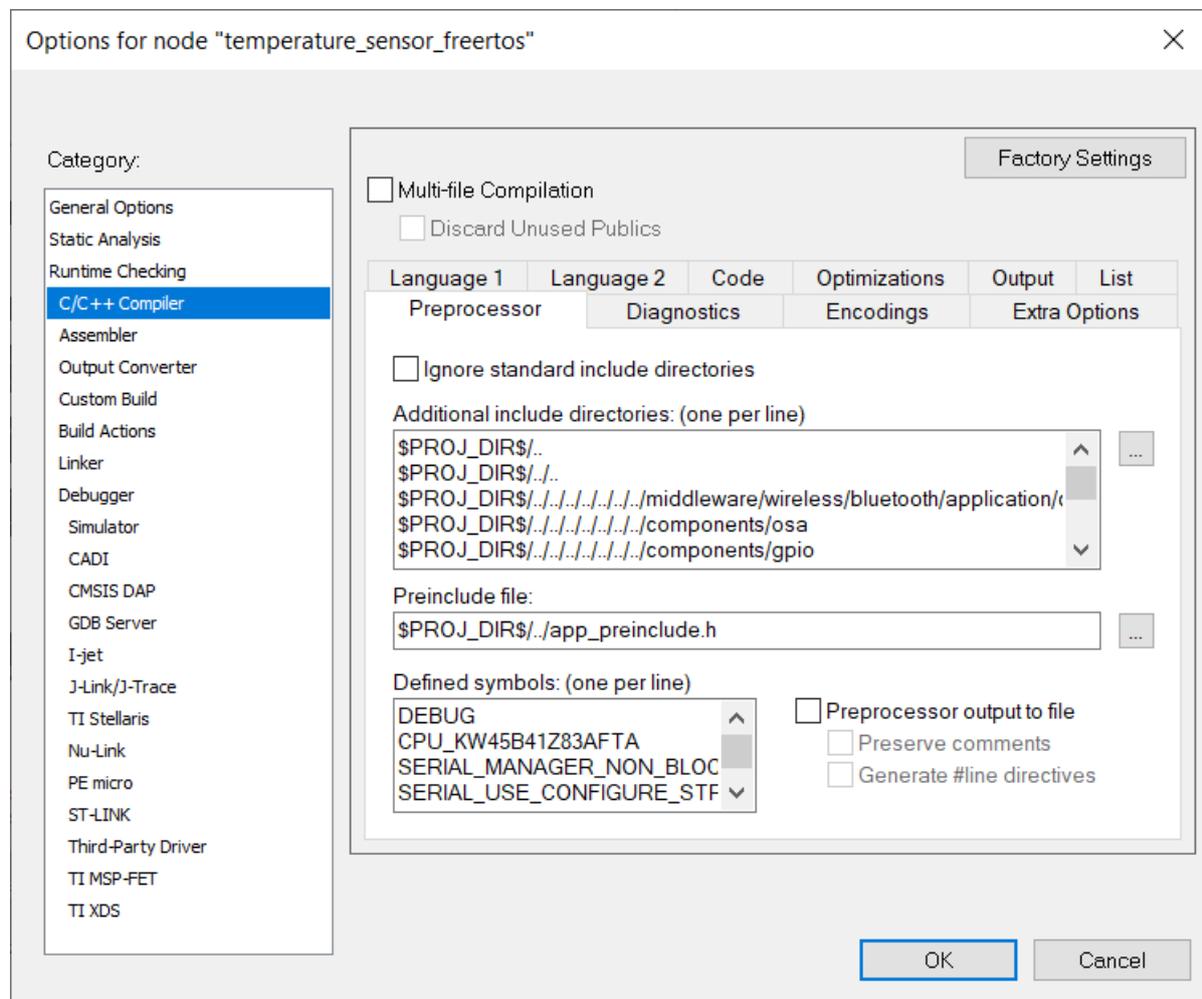
```

- *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 `gAppLowpower_Enabled_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.

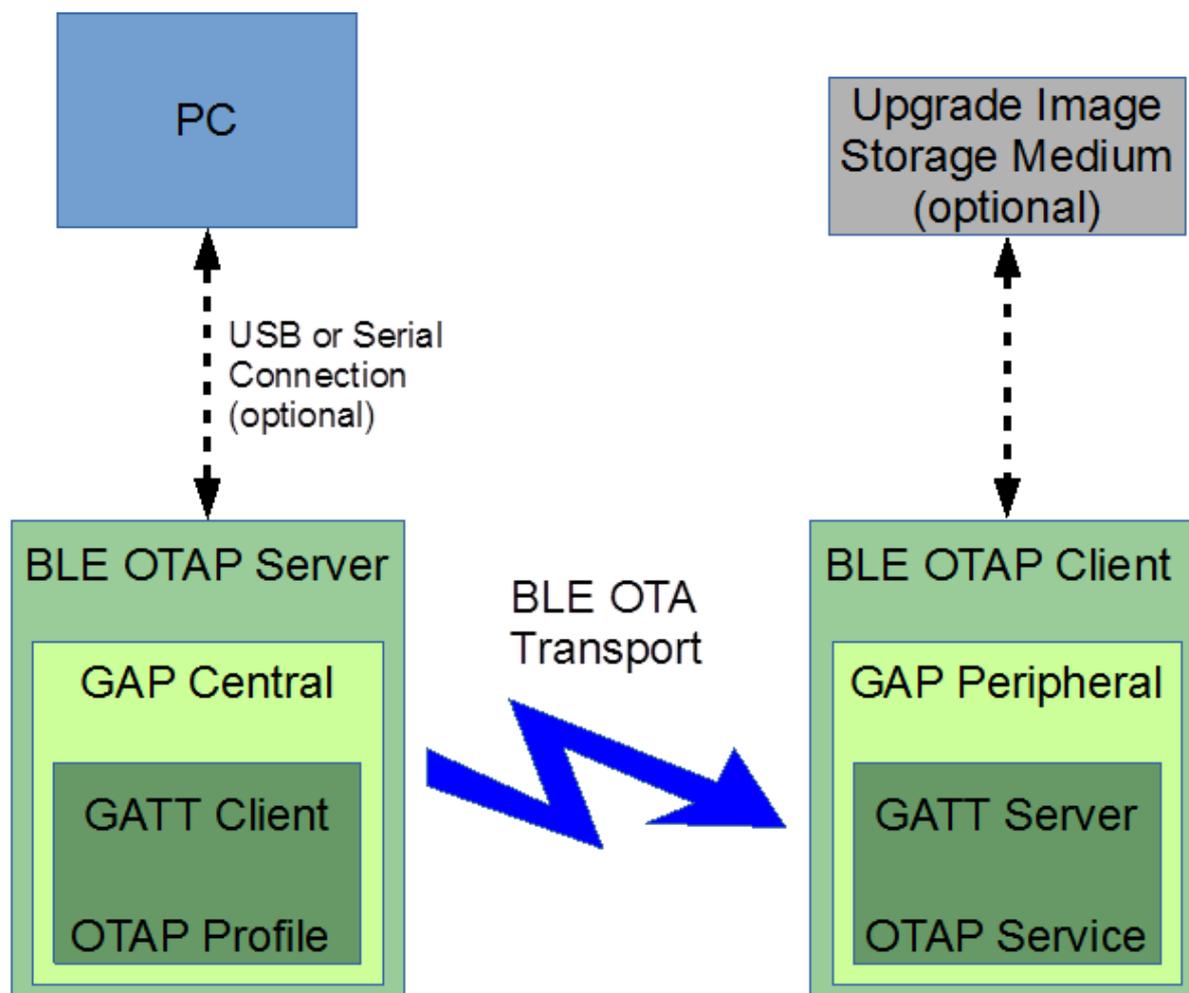
The container for the upgrade image is an image file which has a predefined format which is described in detail. The image file format is independent of the protocol but must contain information specific to the image upgrade infrastructure on an OTAP Client device. Detailed information on how to build 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, (gGattCharPropWriteWithoutResp_
↪ 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	>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	>S	CurrImageId	2	Id of the currently running image. Should be 0x0000.	11
0x02	New Image Info Request	>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	Name	Dir	Parameters	Param Size	Description	Total Size (CmdId+Payload)
0x04	Image Block Request	C->S	StartPositionBlock-SizeChunk-Size-Transfer-MethodL2ca	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 from 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	Param Size	Description	Total Size (CmdId+Payload)
0x05	Image Chunk	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 0xFF) 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 Transfer	Image 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 *ImageVersion* parameter). The best strategy depends on application requirements.

After the OTAP Client has determined that the OTAP Server 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 Upgrade Image Sub-element	Variable	The header contains general information about the image file. 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 <i>Value</i> 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 – 0xffff	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 Identifiers	Sub-element Type 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.

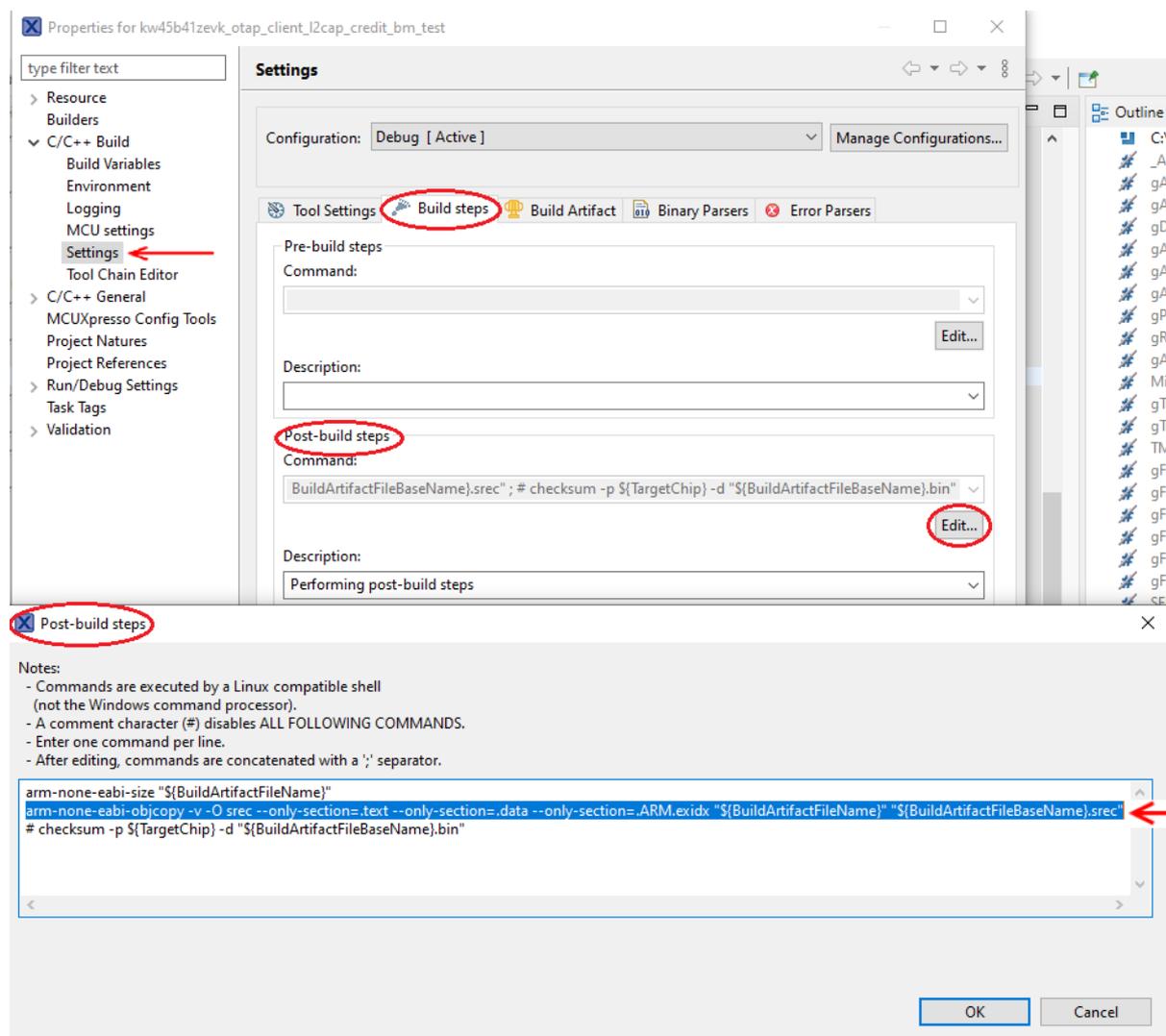
For this, open the target properties and go to the **Output Converter** tab. Activate the **Generate additional output** checkbox and choose the **Motorola** option from the **Output format** drop down menu. From the same pane you can also override the name of the output file. A screenshot of the described configuration is shown in Figure.

![(./images/figure\_19\_new\_srec.png "Enabling Options for Node "otap\_client\_att\_freertos" in IAR Embedded Workbench")

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
"${BuildArtifactFileName}"
"${BuildArtifactFileName}.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
S113001075060000750600007506000075060000F0
S113002075060000750600007506000075060000E0
S113003075060000750600007506000075060000D0
S1130040000000000000000000000000000000000AC
S11300500000000000000000000000000000000009C
.....
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	ASCI- Ihex digits	ASCI- Ihex digits	ASCIIhex digits	ASCI- Ihex 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\\_\(file\\_format\)](http://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* field of the sub-element. Fill all address gaps in the S records with 0xFF.

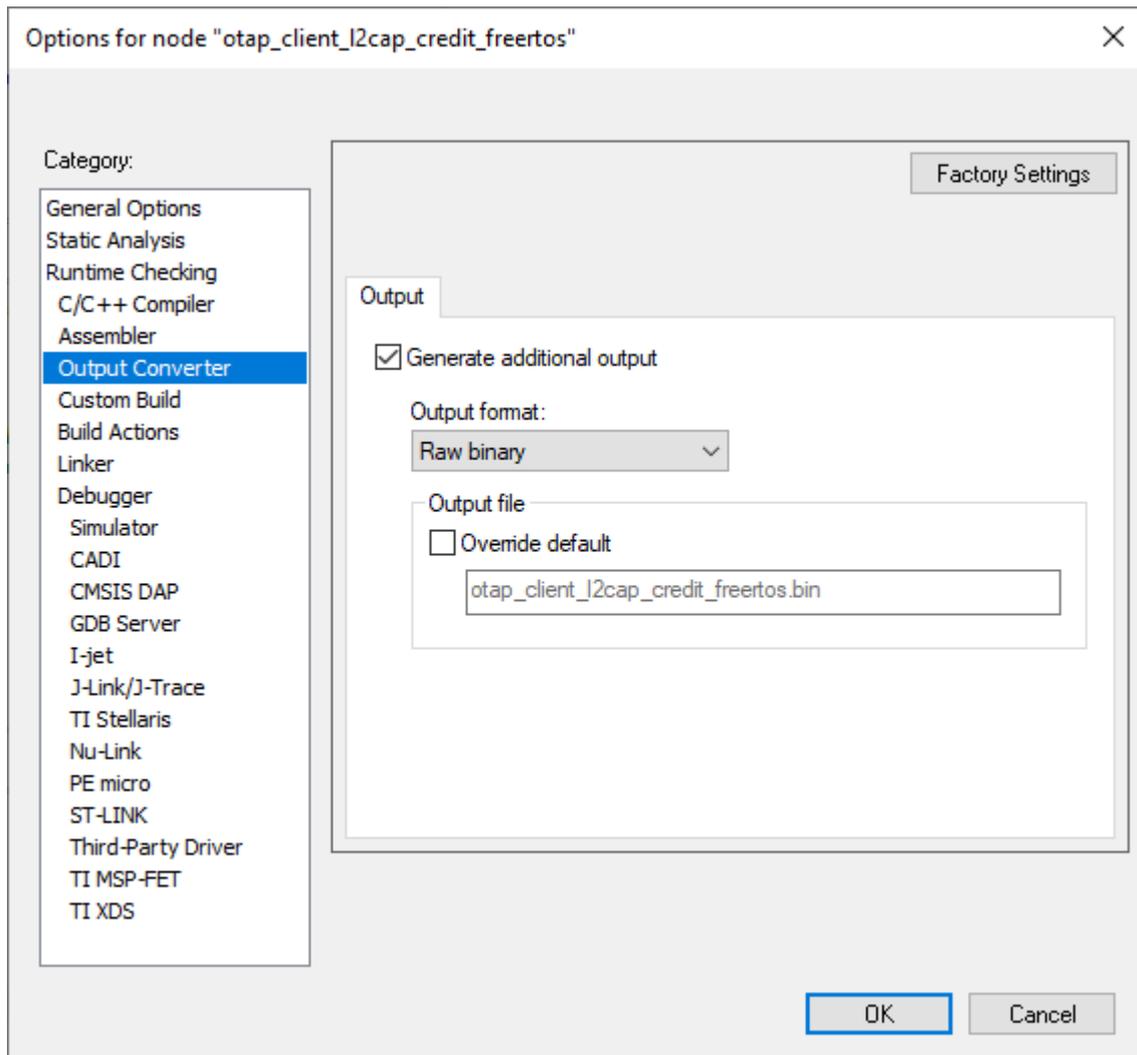
- Fill in the *Length* field of the sub-element with the length of the written *Value* field.
- Create the Sector Bitmap Sub-element
  - A default working setting would be all bytes 0xFF for the *Value* field of this sub-element.
- Create the Image File CRC Sub-element
  - Compute the total image file size as the length of the header + the length of all 3 sub-elements and fill in the appropriate field 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 a 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.

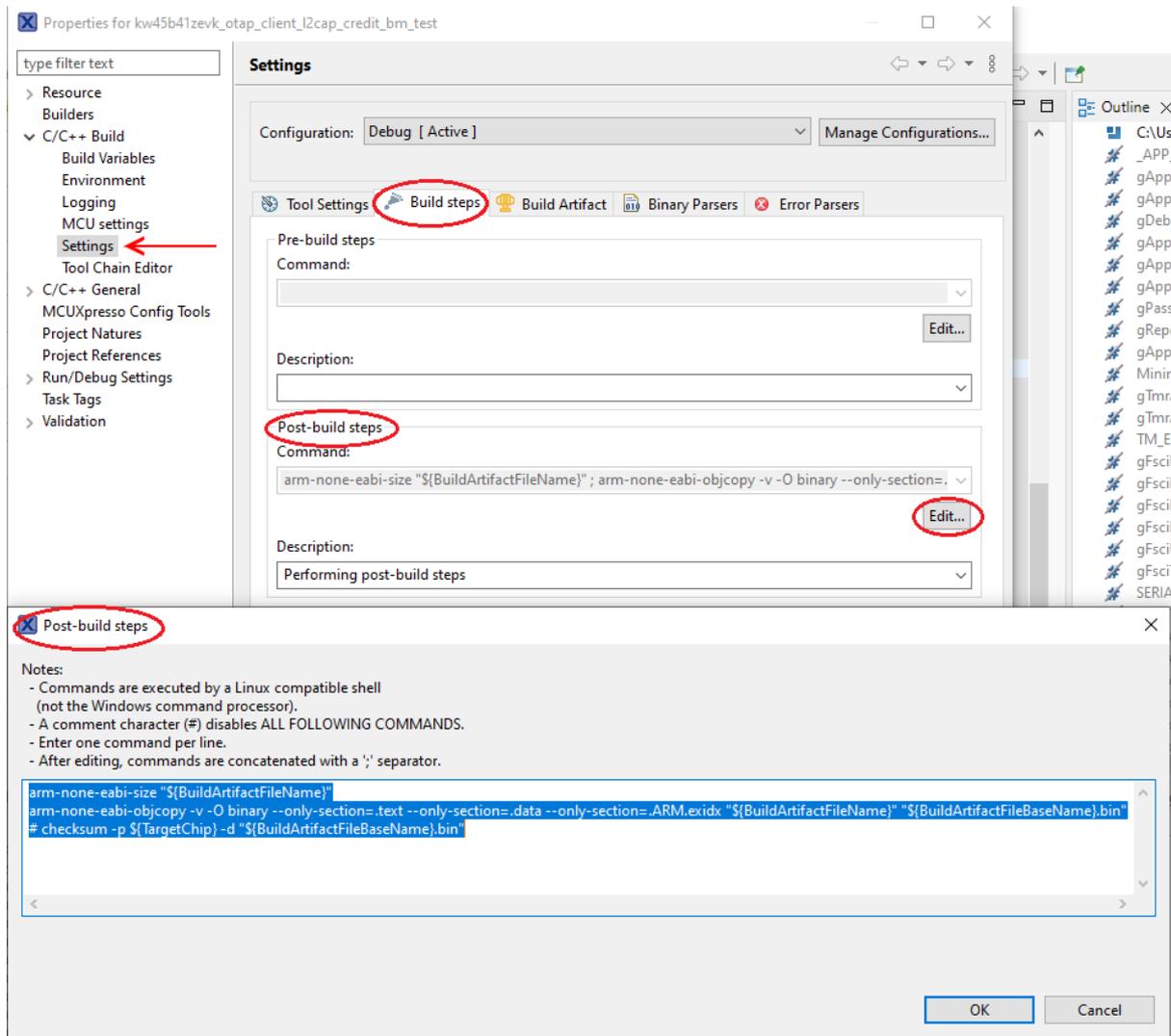
To enable the creation of a BIN file for your embedded application in IAR Embedded Workbench open the target properties and go to the *Output Converter* tab. Activate the “*Generate additional output*” checkbox and choose the *binary* option from the “*Output format*” drop down menu. From the same pane 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
"${BuildArtifactFileName}"
"${BuildArtifactFileName}.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* field of the sub-element.
  - 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 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 field 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 \(OTAP\)](#)

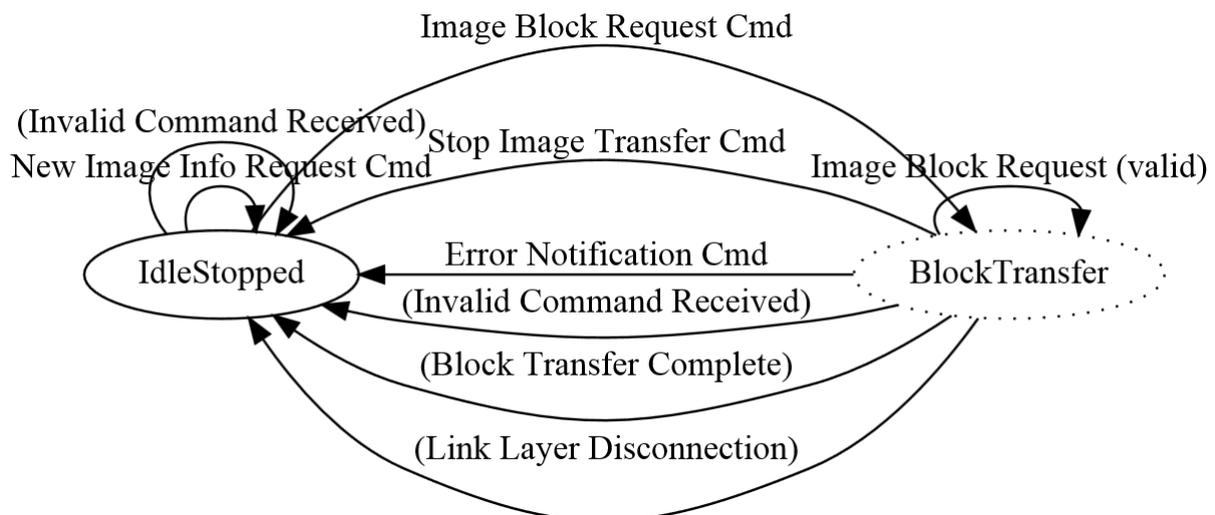
**Bluetooth Low Energy OTAP application integration** The Bluetooth Low Energy OTAP demo applications are standalone applications that only run the OTAP Server and the OTAP Client. In practice, however the OTAP Server and OTAP Client are used alongside with other functions. The OTAP functionality is 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 *gGattProcWriteCharacteristic-Value\_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`:

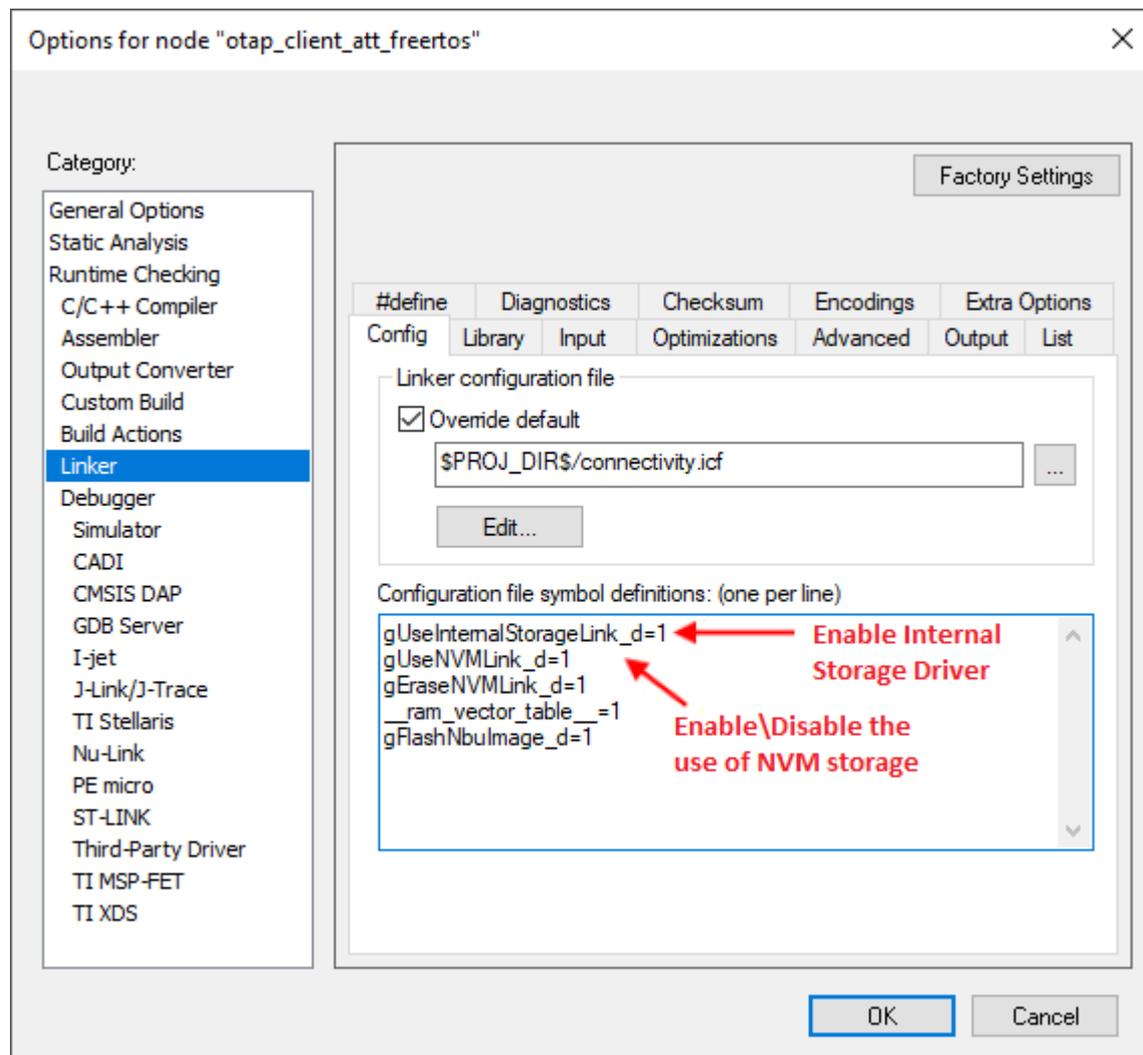
```

/!* 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, the `gUseNVMLink_d` symbol must be set to 1. The `gUseInternalStorageLink_d` symbol must be set to 0 when OTAP external storage is used and to 1 when the internal storage is used. The `gEraseNVMLink_d` must be set to 0.

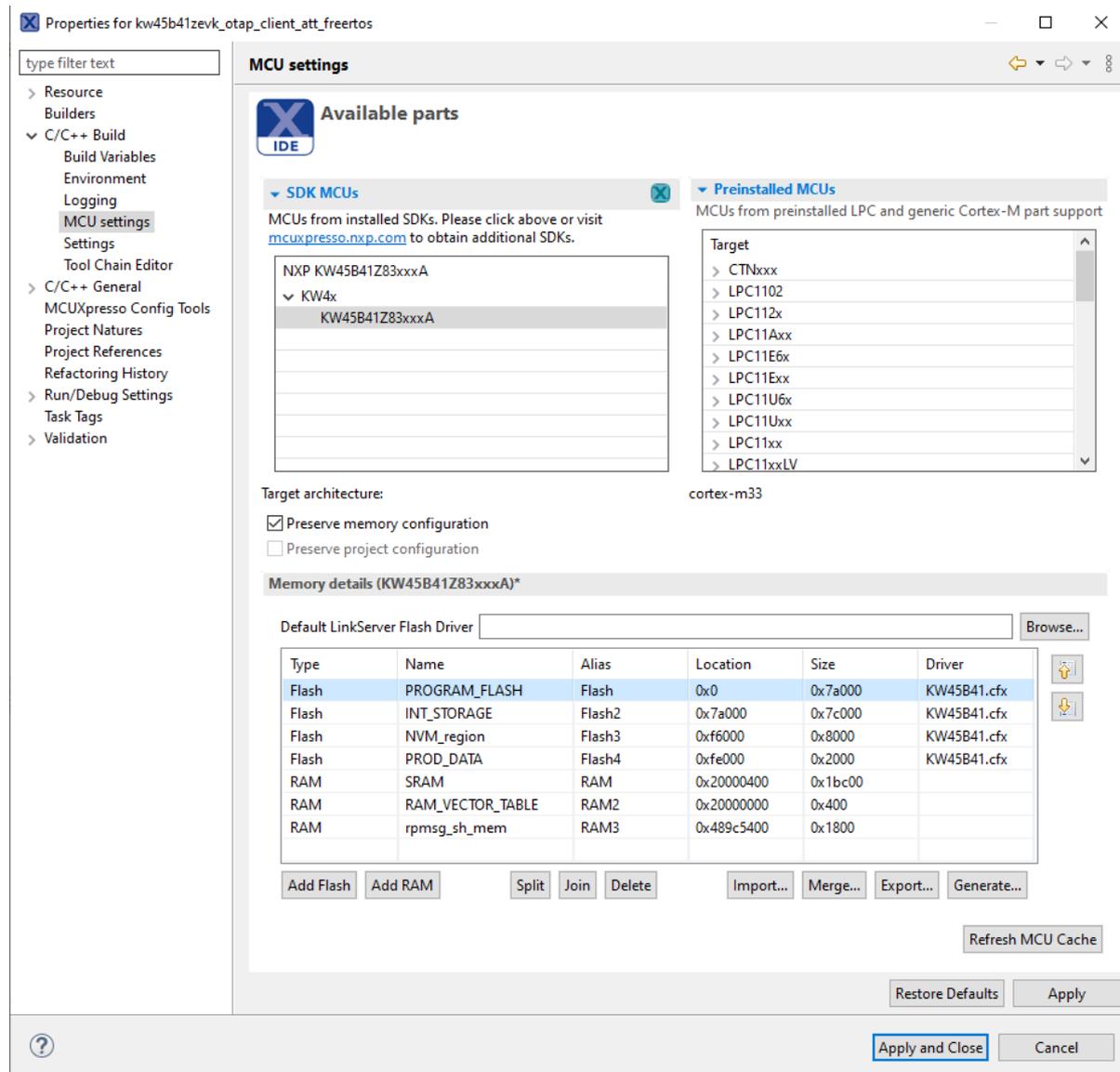
An example linker configuration window for IAR is shown below.



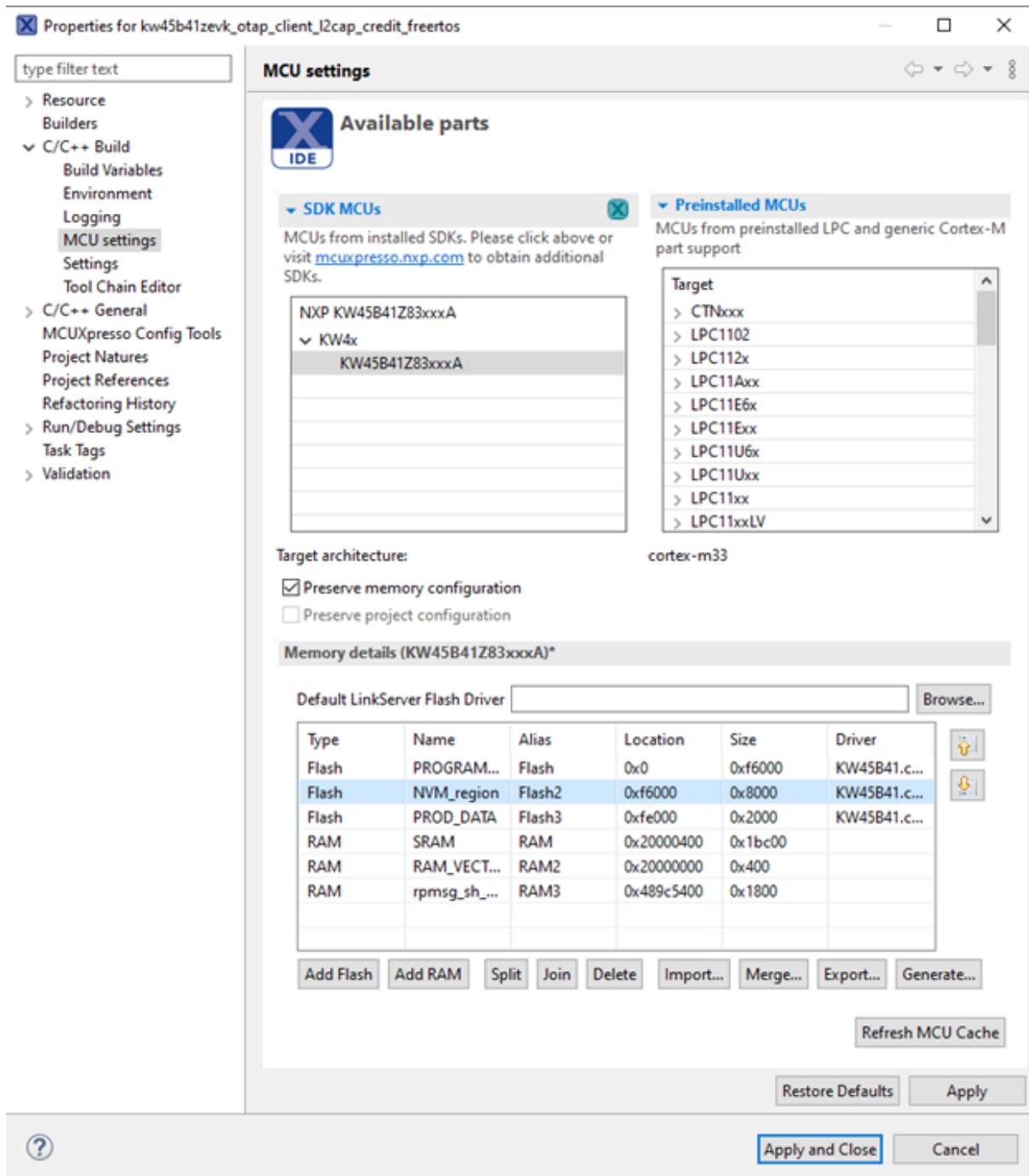
**Note:** The `gEraseNVMLink_d=1` IAR linker flag places some dummy bytes into the NVM region to invalidate the data and force the application to erase the entire NVM region. When generating

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\_FLASH section as shown in the Figure.



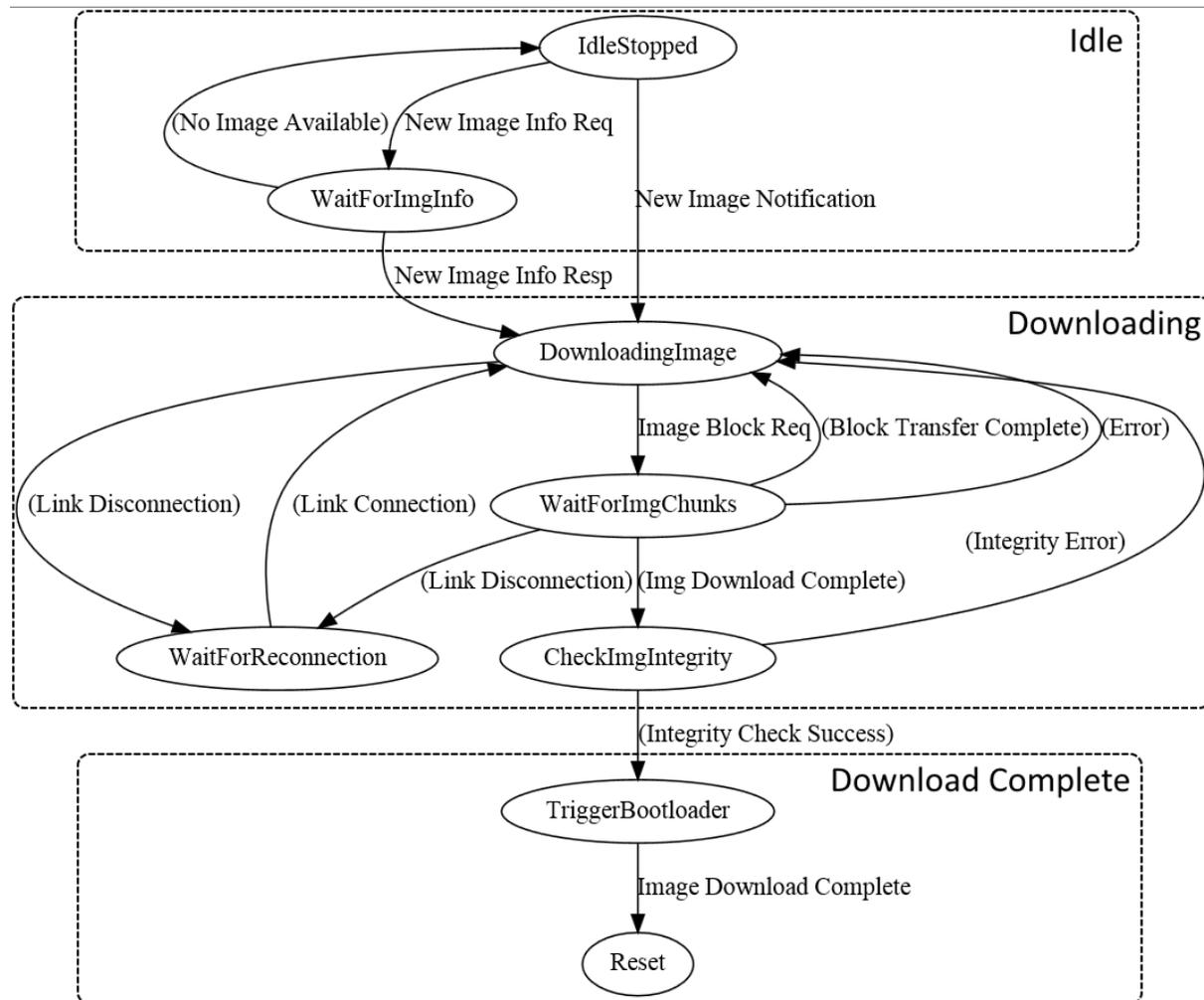
Once the application starts and bidirectional OTAP communication is established via the OTAP Service, then the OTAP Client must determine if the connected OTAP Server has a newer image than the one currently present on the device. This can be done in two ways:

- 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 *ImageVersion* 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 ... */
 if (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 also 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 GATTDDB) registration** For receiving messages from all the Bluetooth Low Energy Host Stack serial interfacing layers (GAP, GATT, and GATTDDB), 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, *GATTDynamic* commands must be used. The *GATTDynamic* 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_AddDeviceInfoService (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, GATTDDB, 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, GATTDDB, 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

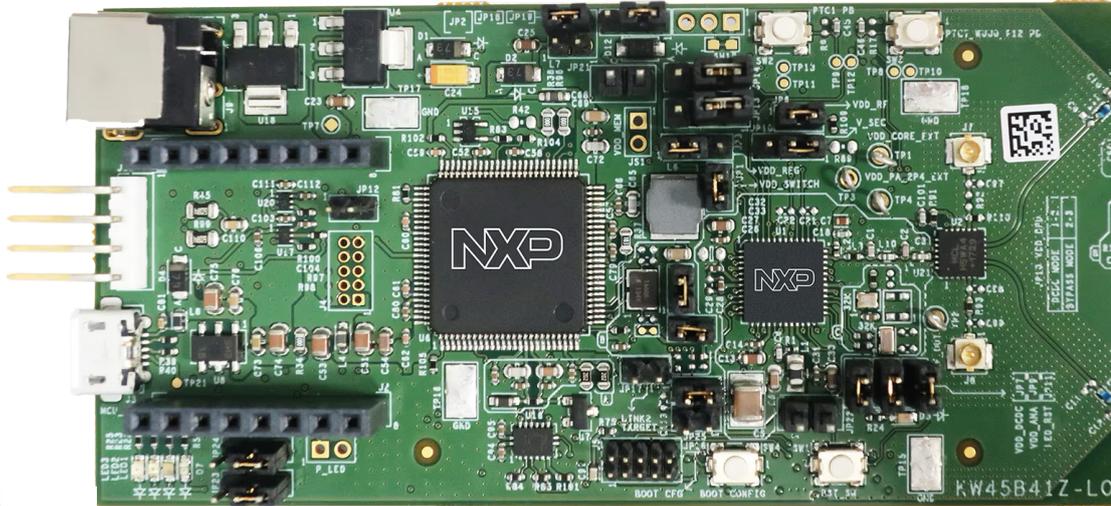
**Introduction** This document briefly describes the process of using NXP Bluetooth Low Energy Software for the KW45, K32W1, MCXW71, KW47, or MCXW72 wireless microcontroller platforms (version 1.1.0). It lists the hardware setup and steps for building and usage of the provided demo applications.

**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.

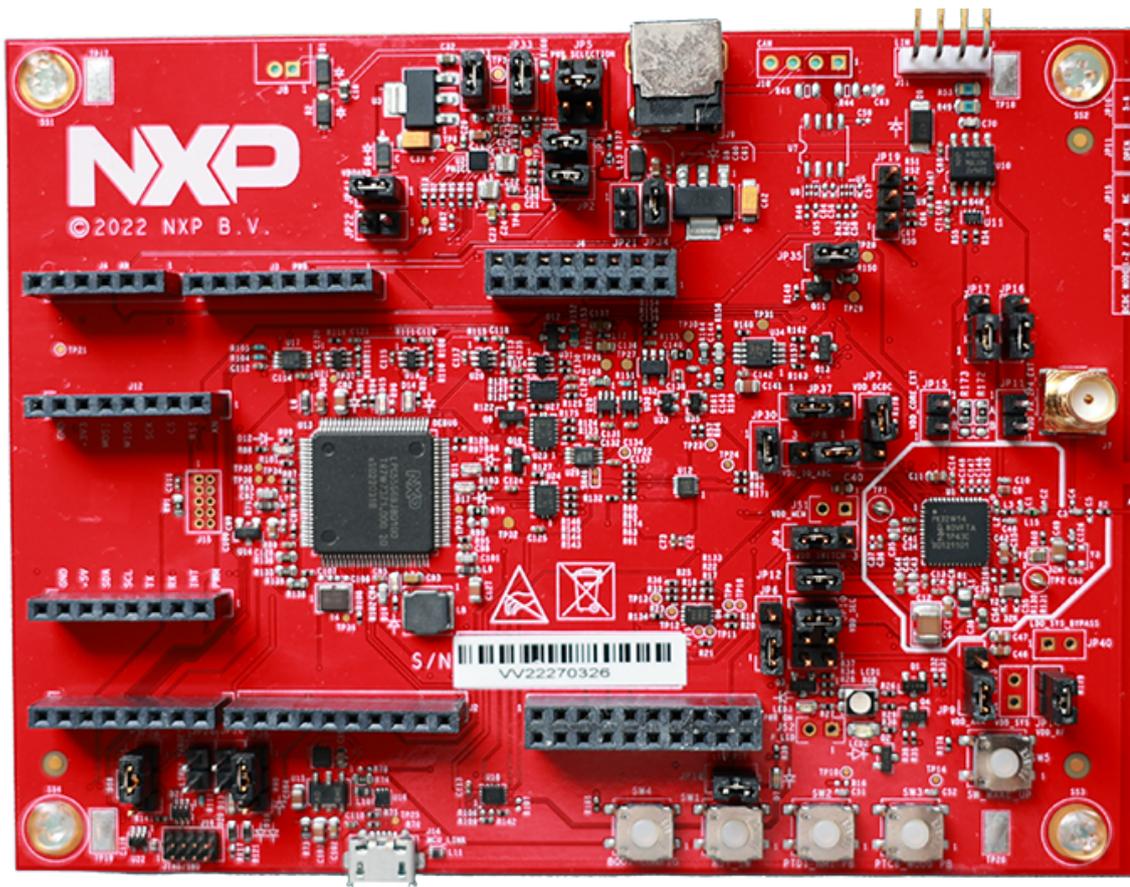
- The default interface selected in the IAR Embedded Workbench for Arm projects included in this release is below:
  - CMSIS-DAP for kw45b41zevk, kw45b41zloc, k32w148evk, mcxw71evk, frdmmcwx71
  - JLink for kw47evk, kw47loc, mcxw72evk, and frdmmcwx72 platforms
- Use jumpers to configure the boards in 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.



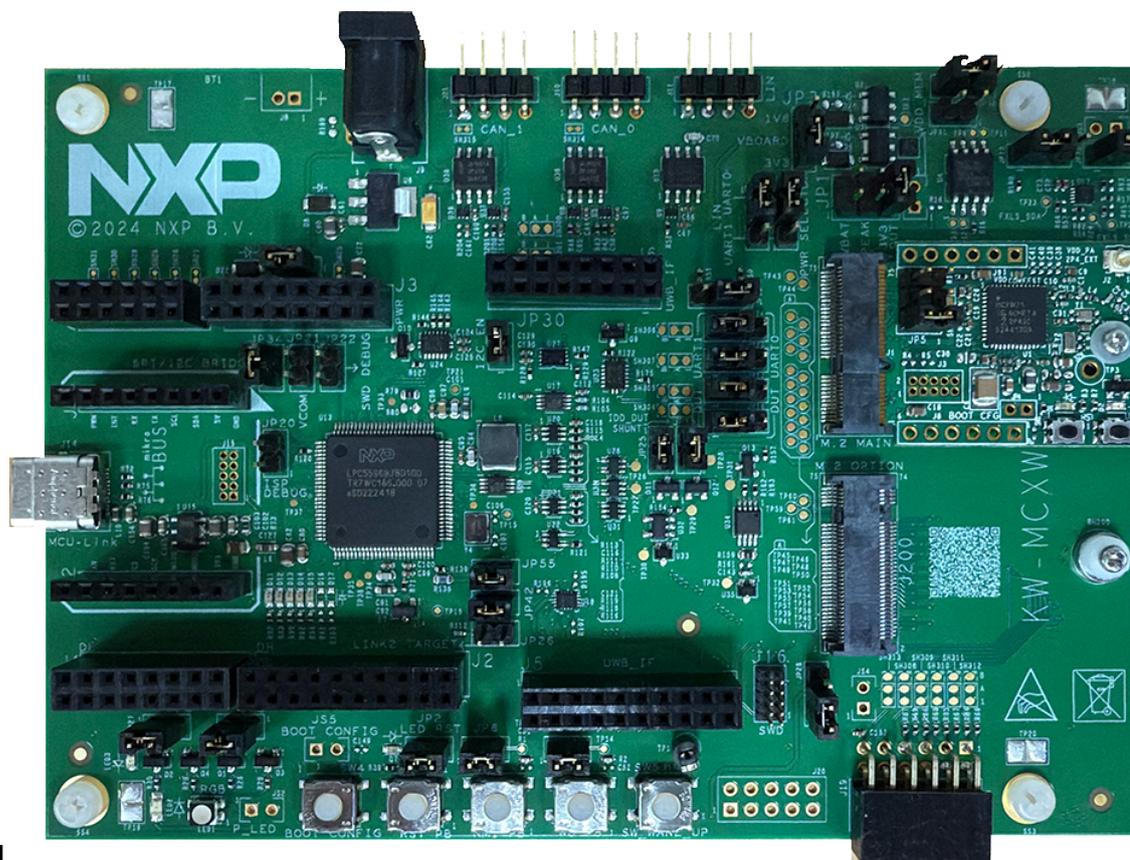
KW45B41Z-EVK board



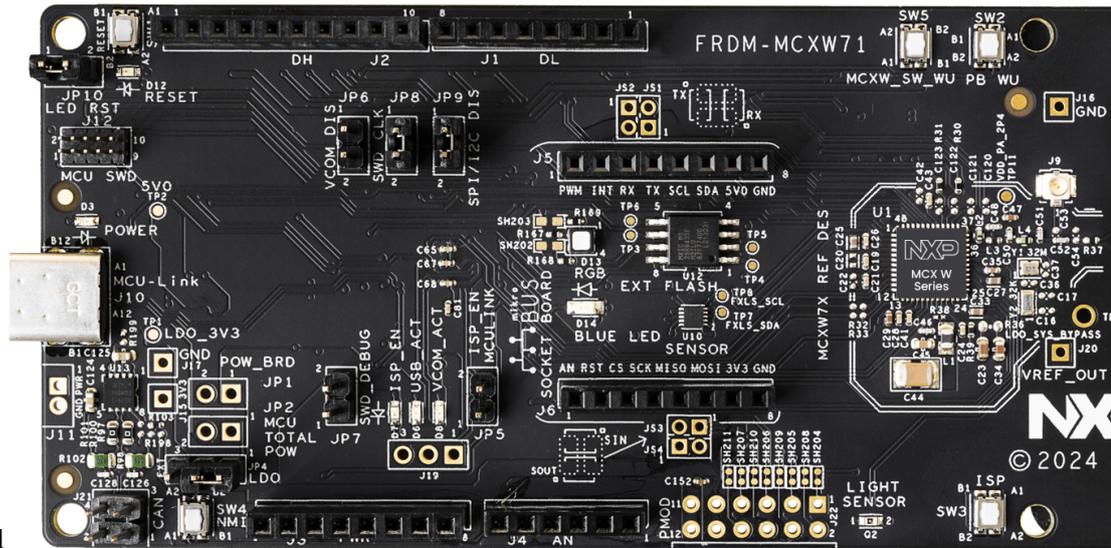
KW45B41Z-LOC board



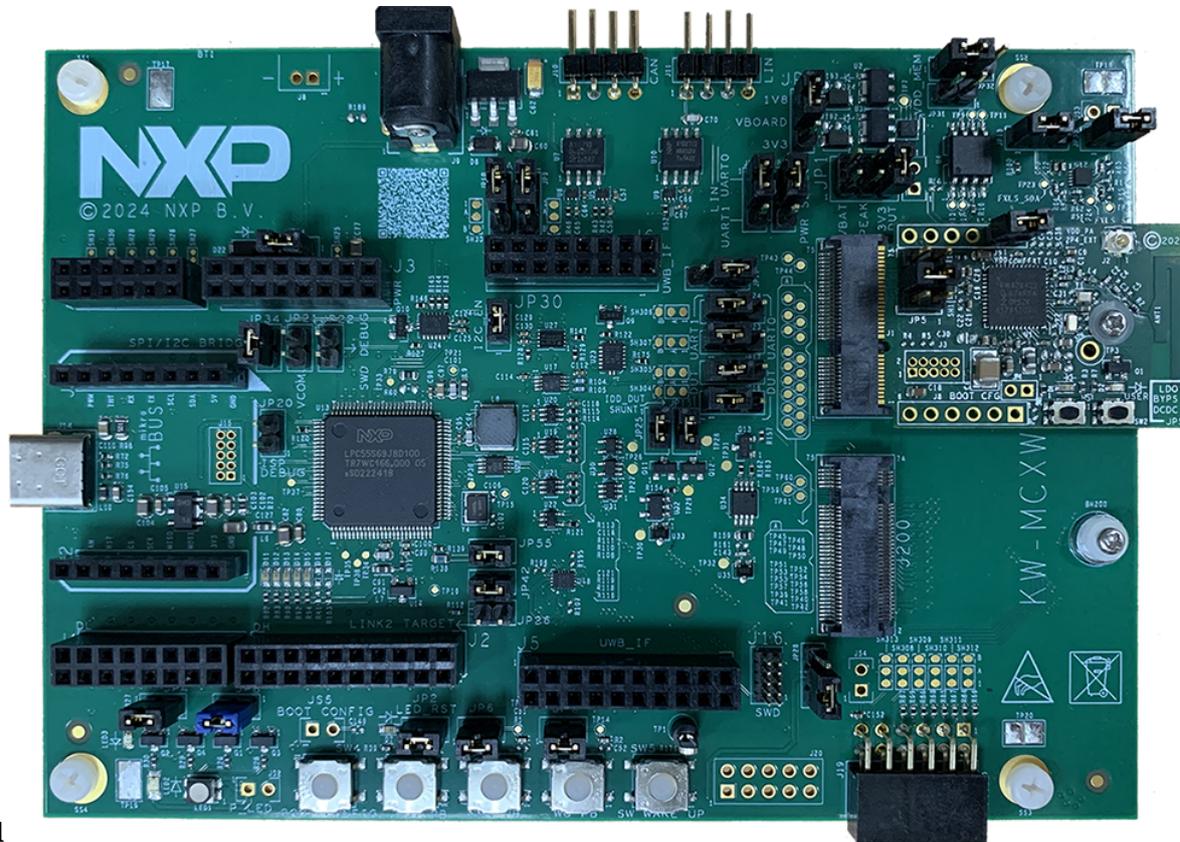
K32W148-EVK board



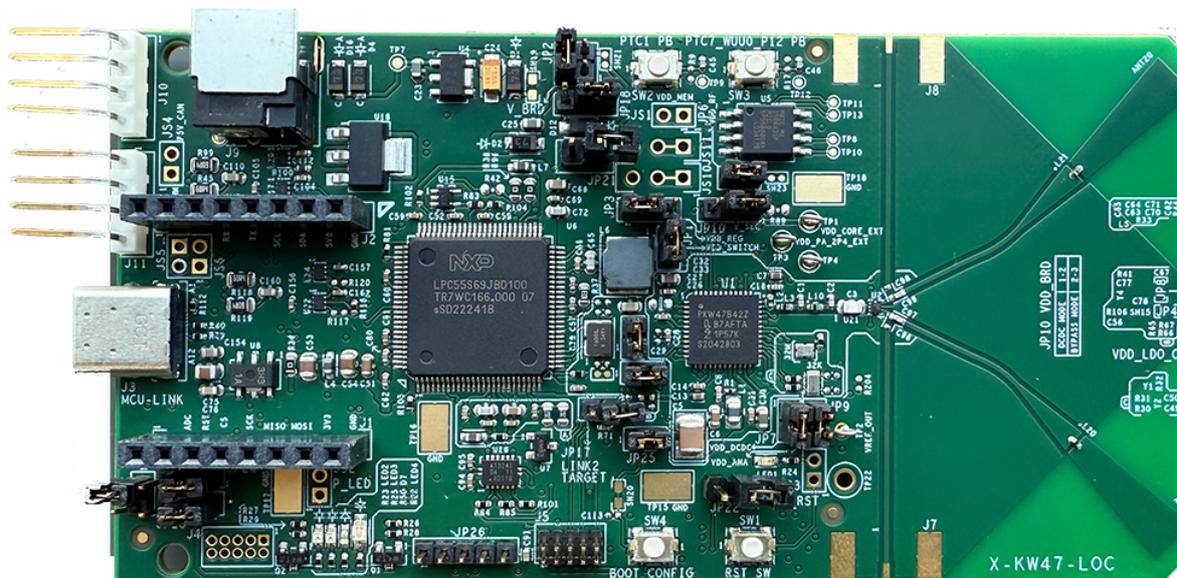
MCX-W71-EVK board



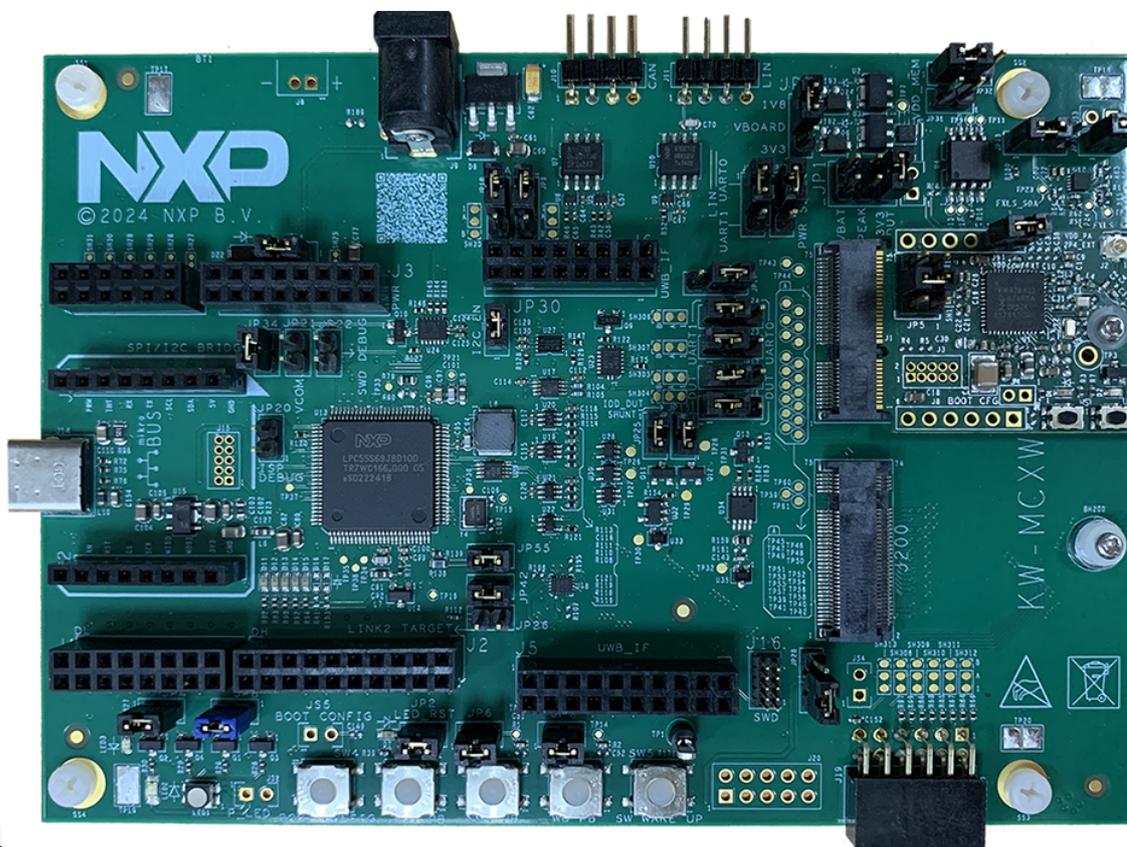
FRDM-MCXW71 board



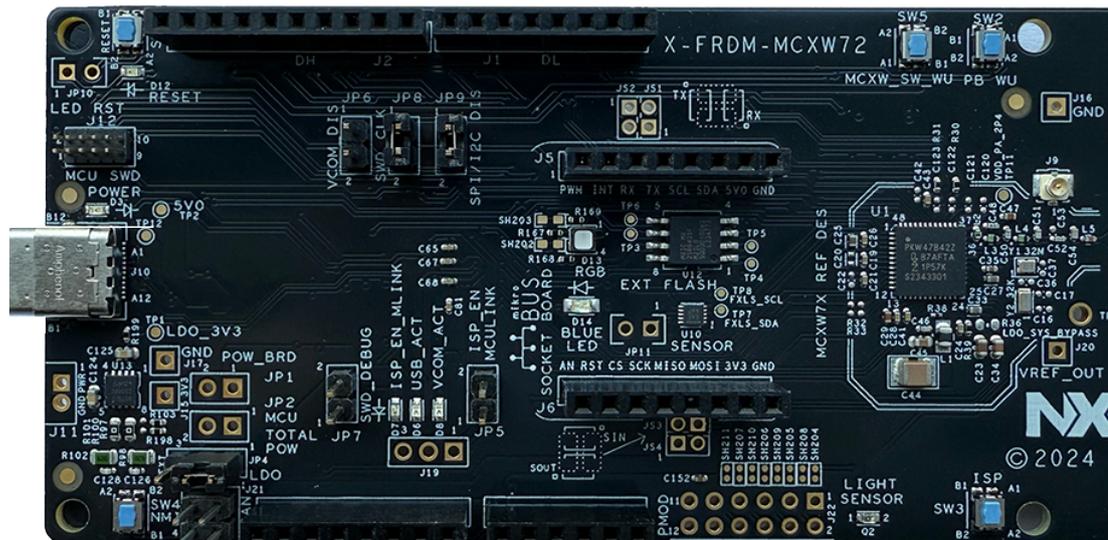
KW47-EVK board



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
<code>-b kw47evk</code>	Specifies the board
<code>--toolchain=iar</code>	Specifies the toolchain (default is <code>armgcc</code> )
<code>-t guiproject</code>	Creates an IDE project
<code>-Dcore_id=cm33_core0</code>	Core ID, if available
<code>--pristine</code>	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>\boards\<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

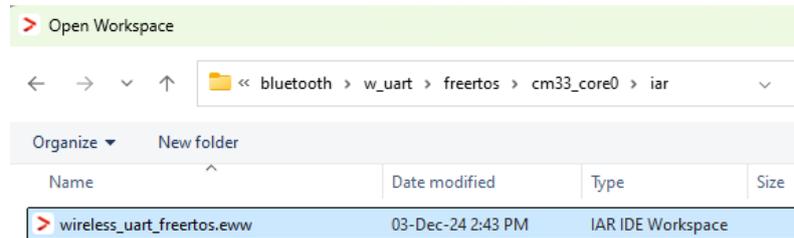
**Resulting location:**

<connectivity\_path>\boards\<kw45b41zevk / kw45b41zloc / k32w148evk / frdm-mcxw71 / kw47evk / kw47loc / mcxw72evk / frdm-mcxw72>\wireless\_examples\bluetooth\w\_uart\freertos\<IDE>

**Building and flashing the BLE software demo applications using IAR Embedded Workbench**

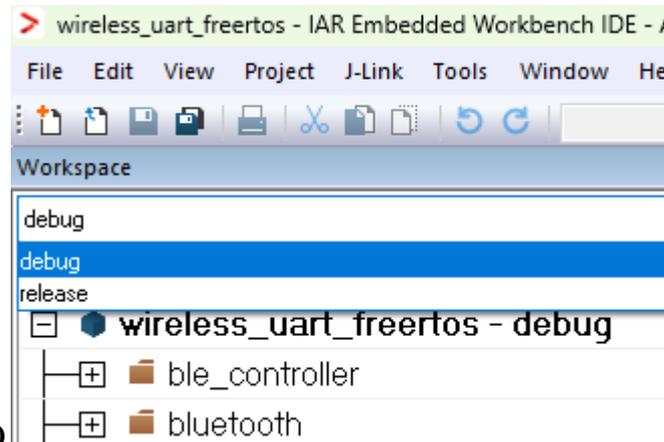
Use the following steps in order to build and flash the BLE software demo applications using the IAR Embedded Workbench:

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.



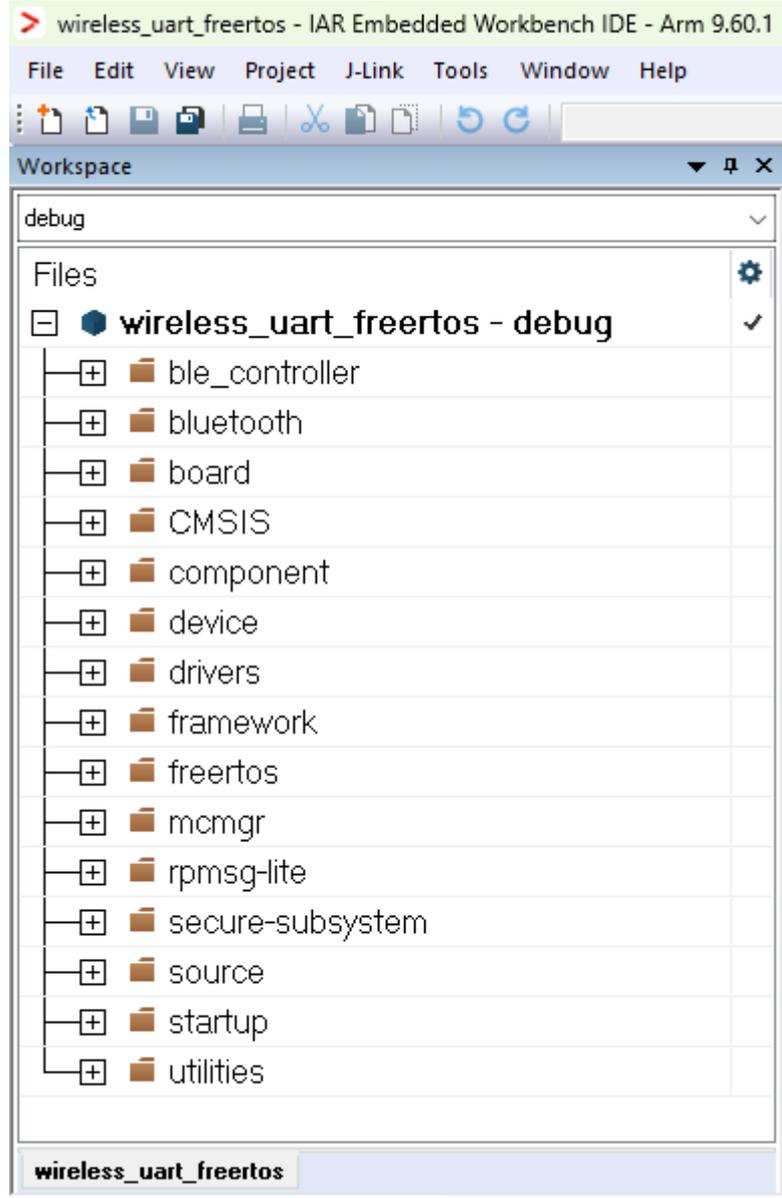
**Wireless UART IAR demo project location**

3. Choose between Debug and Release configurations in the drop-down selector above the project tree in the workspace.

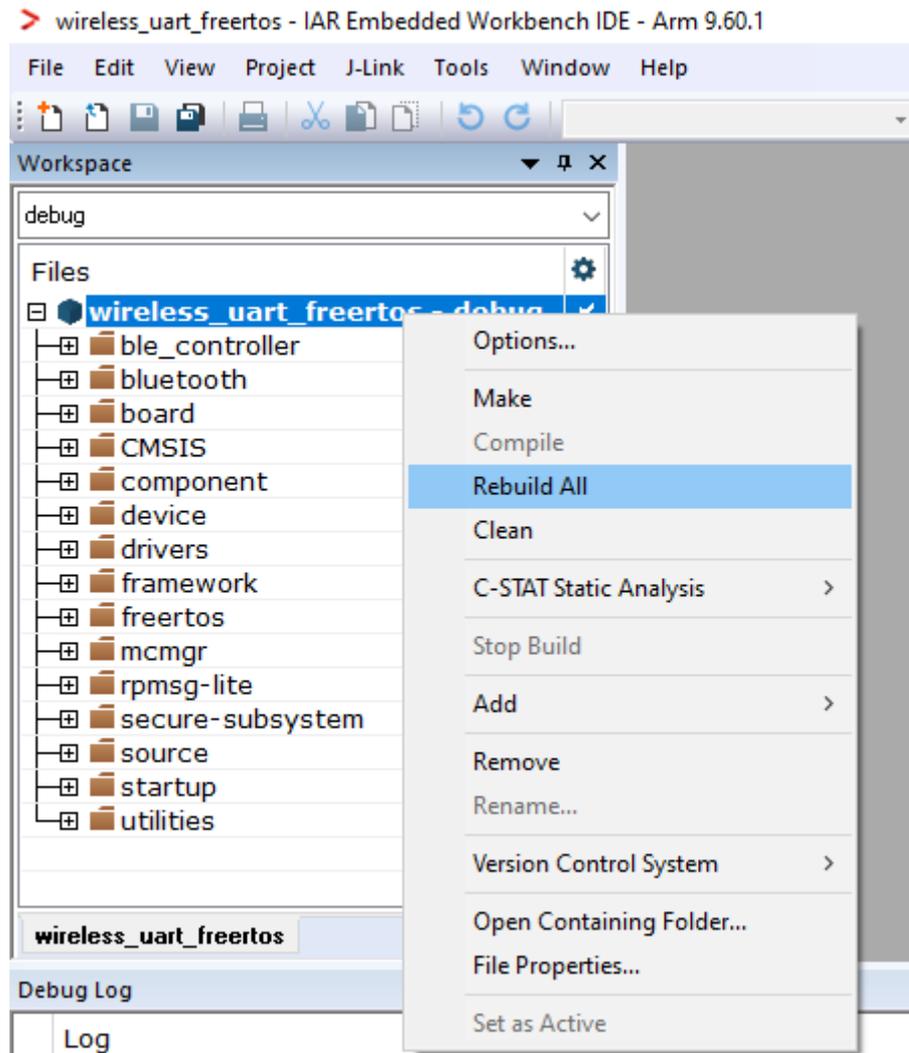


**Select the desired configuration (Debug or Release)**

The figure below shows the Wireless UART - IAR workspace.

**Wireless UART - IAR workspace**

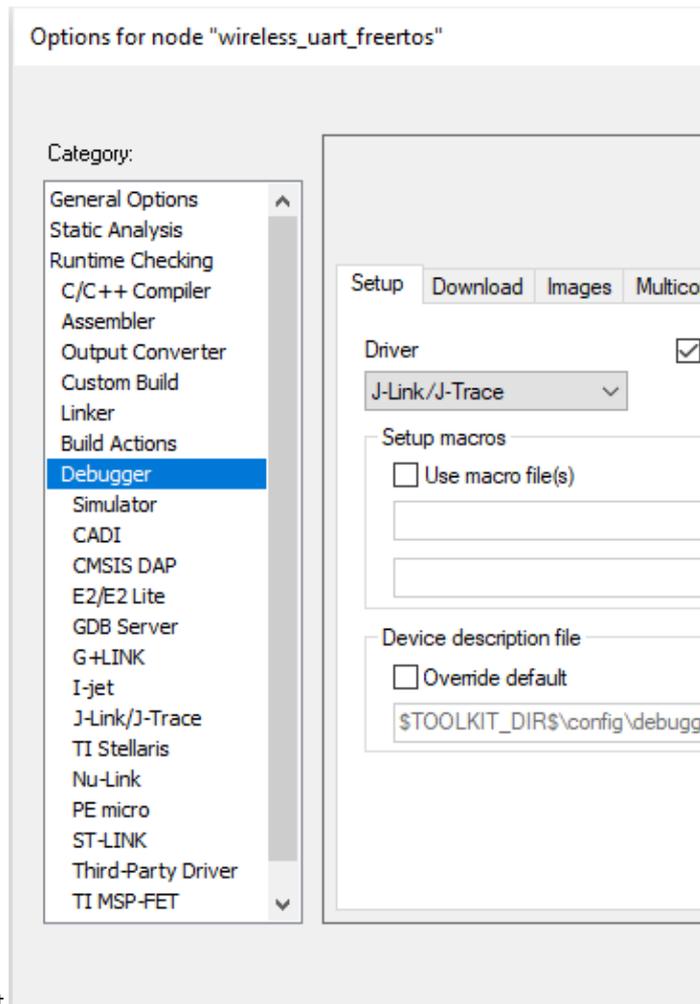
4. Build the Wireless UART project using the options shown in the figure.



### Build Wireless UART application

5. Make the appropriate debugger settings in the project options window, as seen in the next figure.

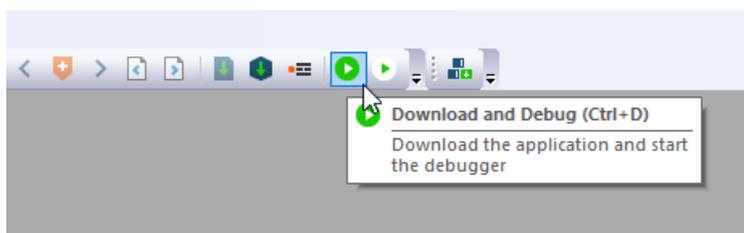
Go to: **Project > Options (Alt+F7) > Debugger > Setup (tab) > Driver > J-Link/J-Trace**



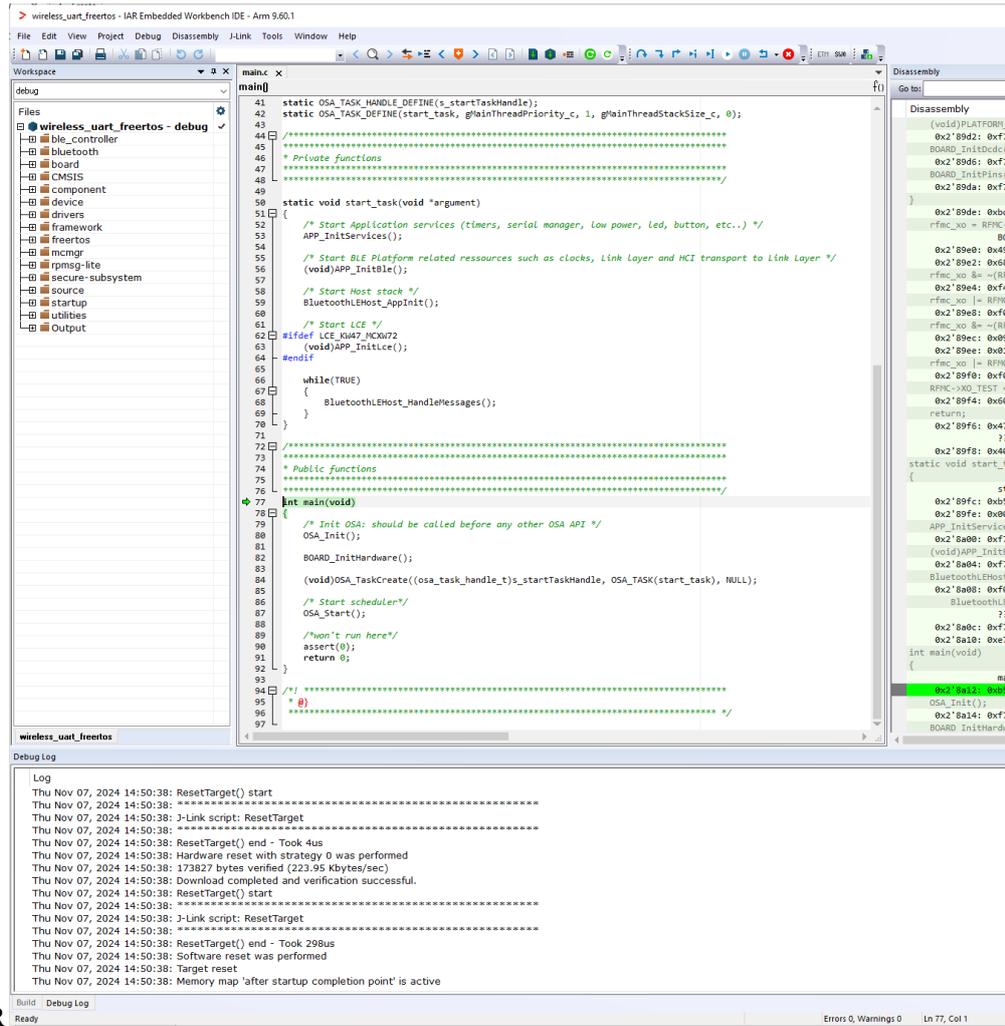
**Debugger Settings for the Wireless UART project**

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.

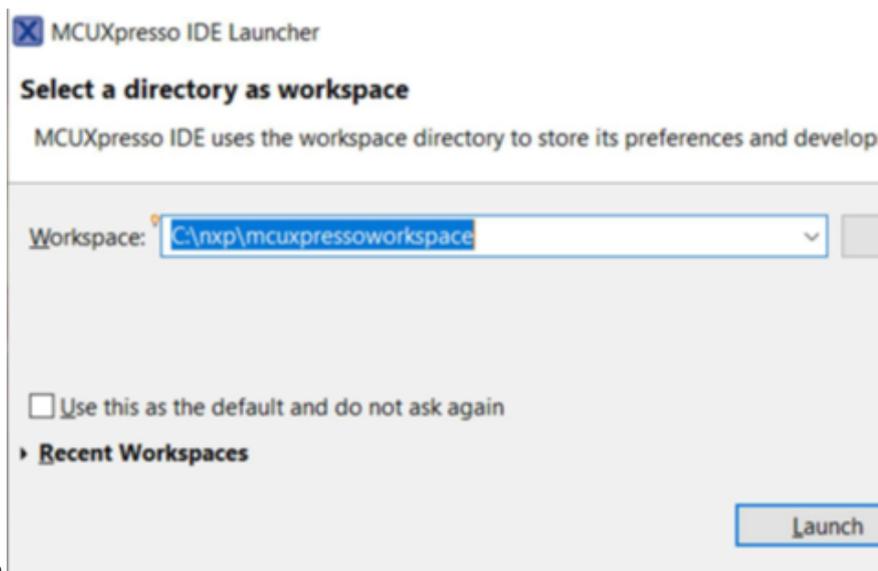


### Running the code on IAR

Parent topic: [Building the binaries](#)

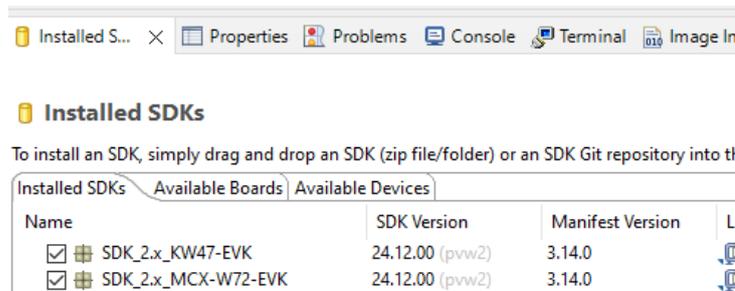
**Building and flashing the BLE Software Demo applications using MCUXpresso IDE** To build and flash the BLE software demo applications using MCUXpresso IDE, follow the steps listed below:

1. Open MCUXpresso IDE and open an existing or new workspace location.



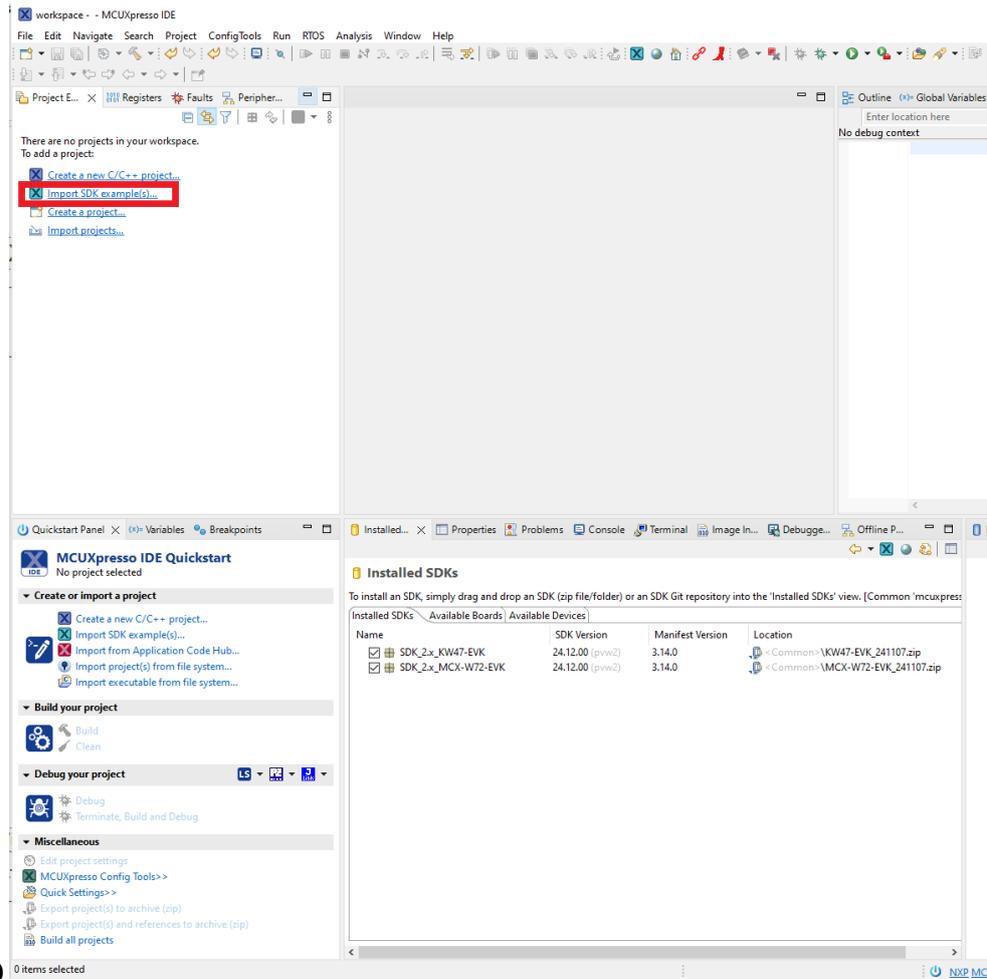
**Select MCUXpresso IDE workspace**

2. Drag and drop the package archive into the **MCUXpresso Installed SDKs** area in the lower right of the main window.



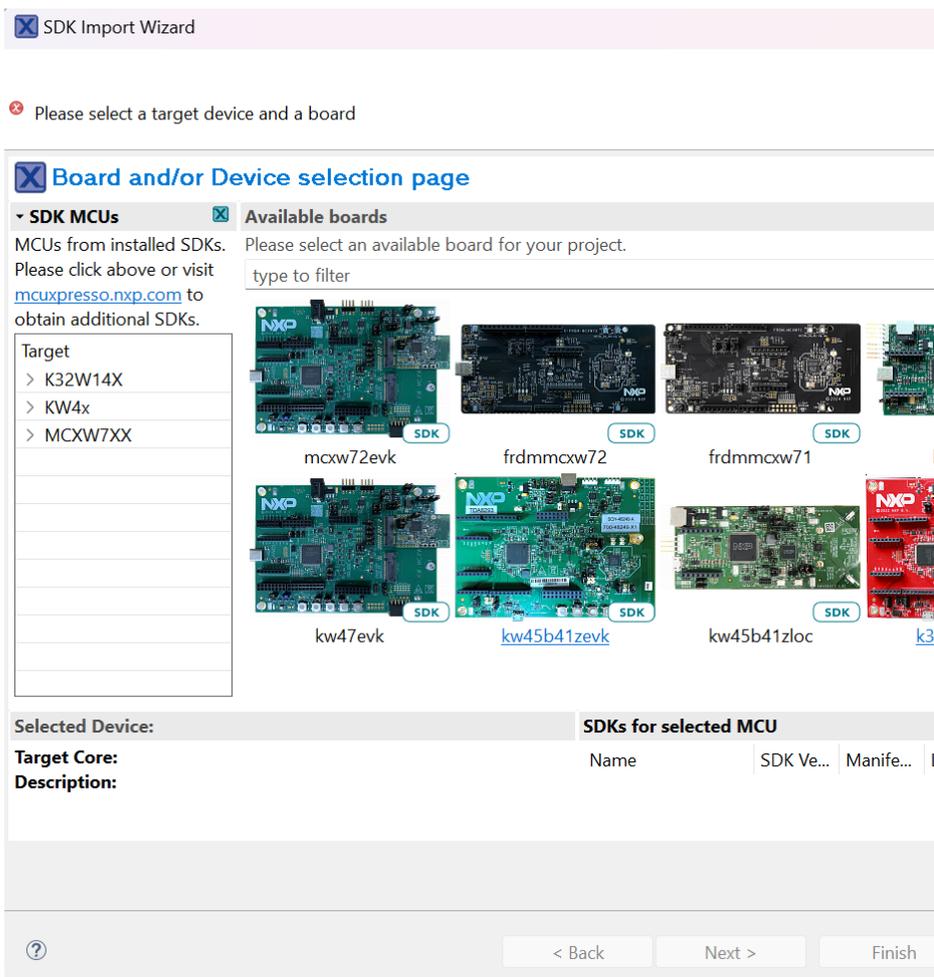
**Installed SDKs in MCUXpresso IDE workspace**

3. After the SDK is loaded successfully, select the **“Import the SDK examples(s)...”** to add examples to your workspace.

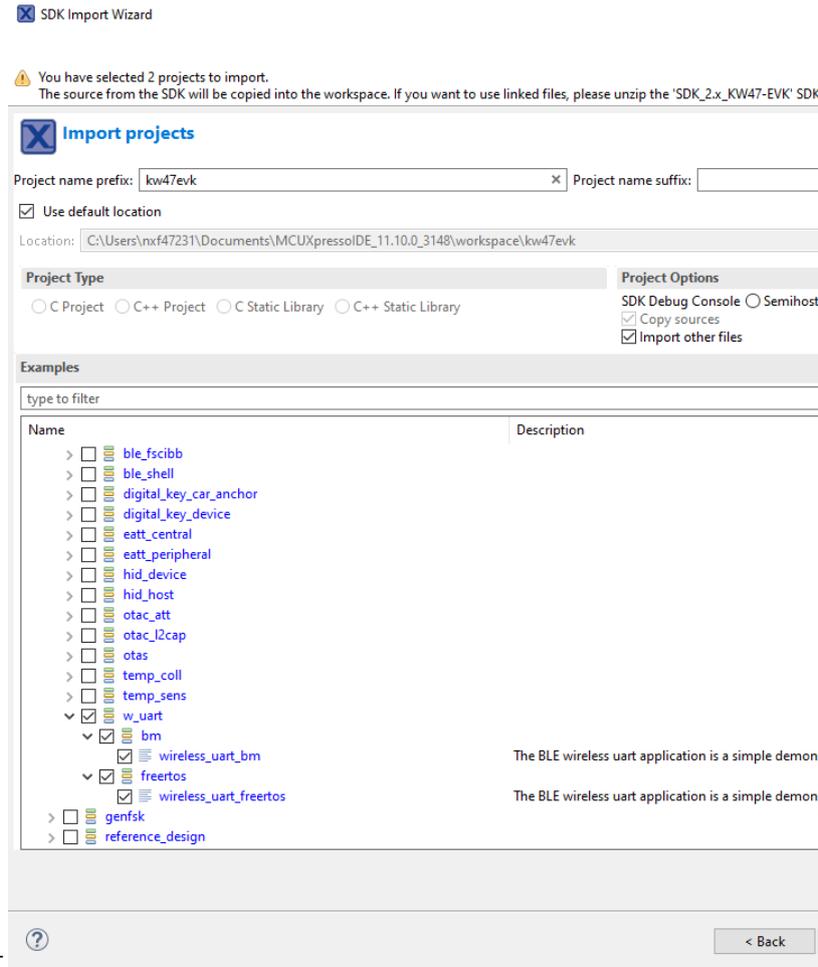


**Importing SDK example(s)**

4. To select the desired example(s), select the *kw45b41zevk / kw45b41zloc / k32w148evk / kw47evk / frdm-mcxw71 / frdm-mcxw72 / mcxw72evk / kw47loc* board and then click the "Next" button:

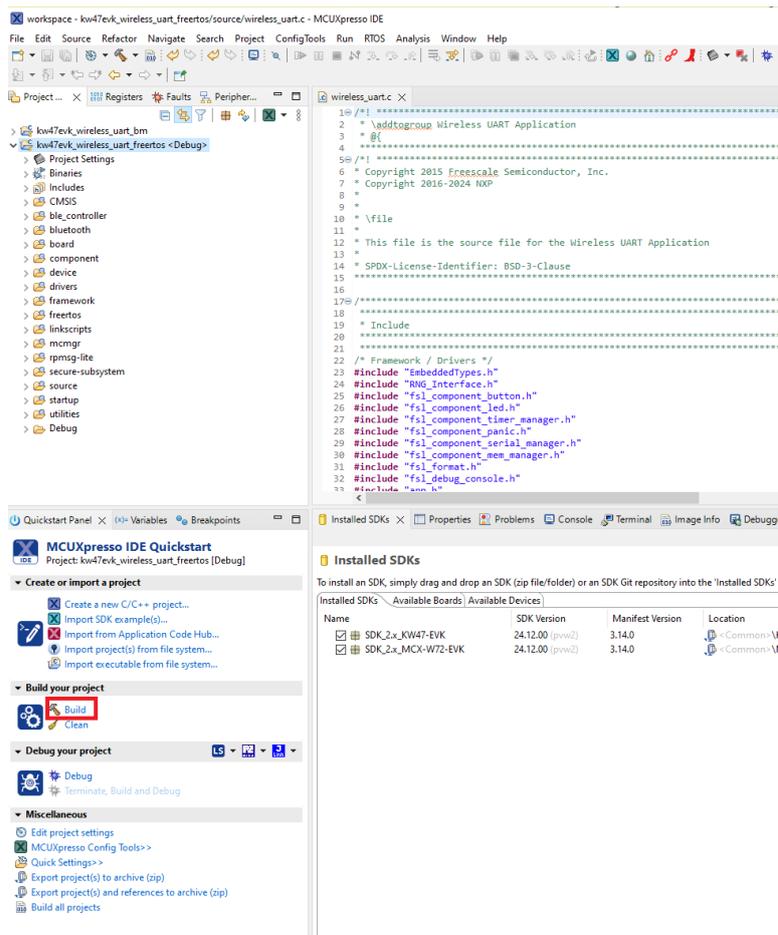


Selecting the KW47-EVK board



**Select Wireless UART FreeRTOS project**

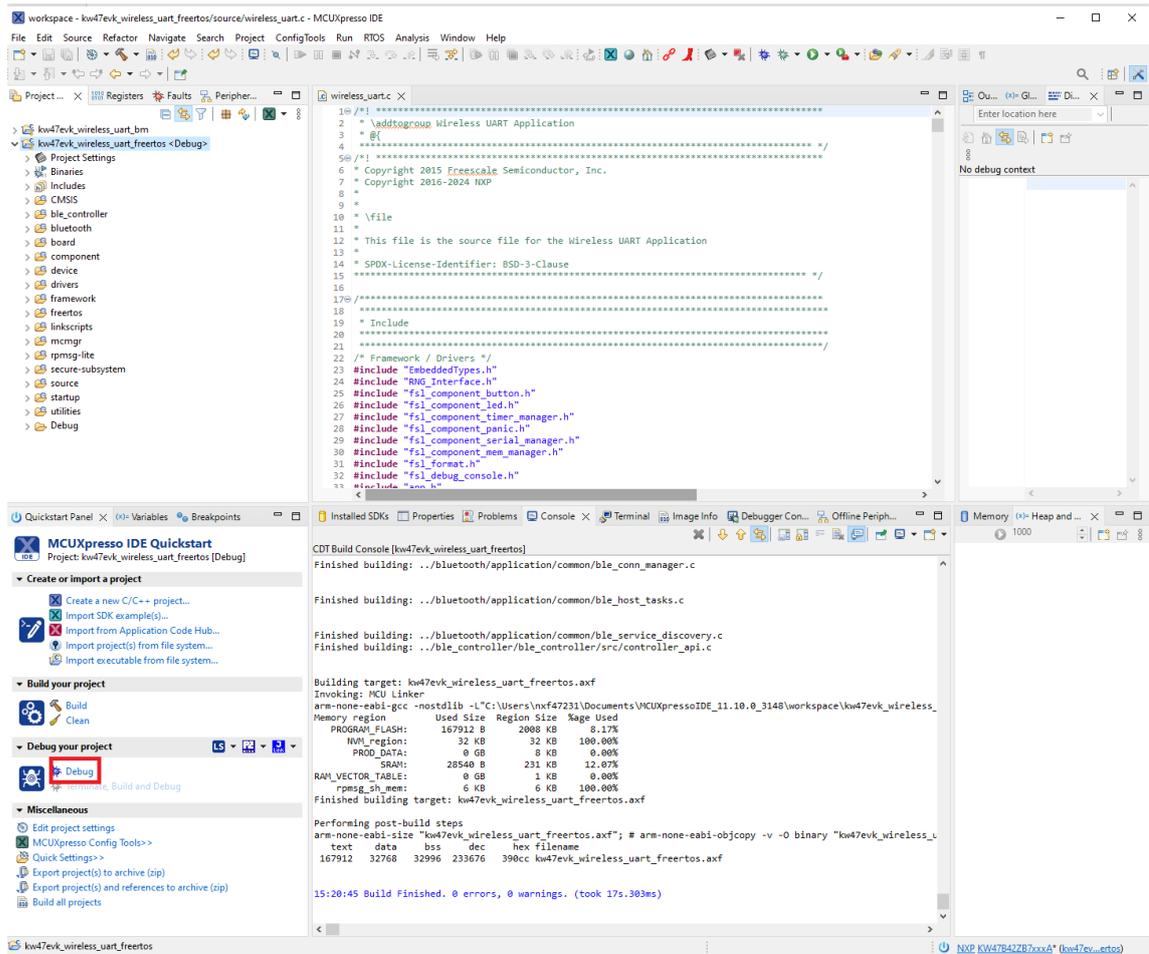
5. Build the wireless\_uart\_freertos project.



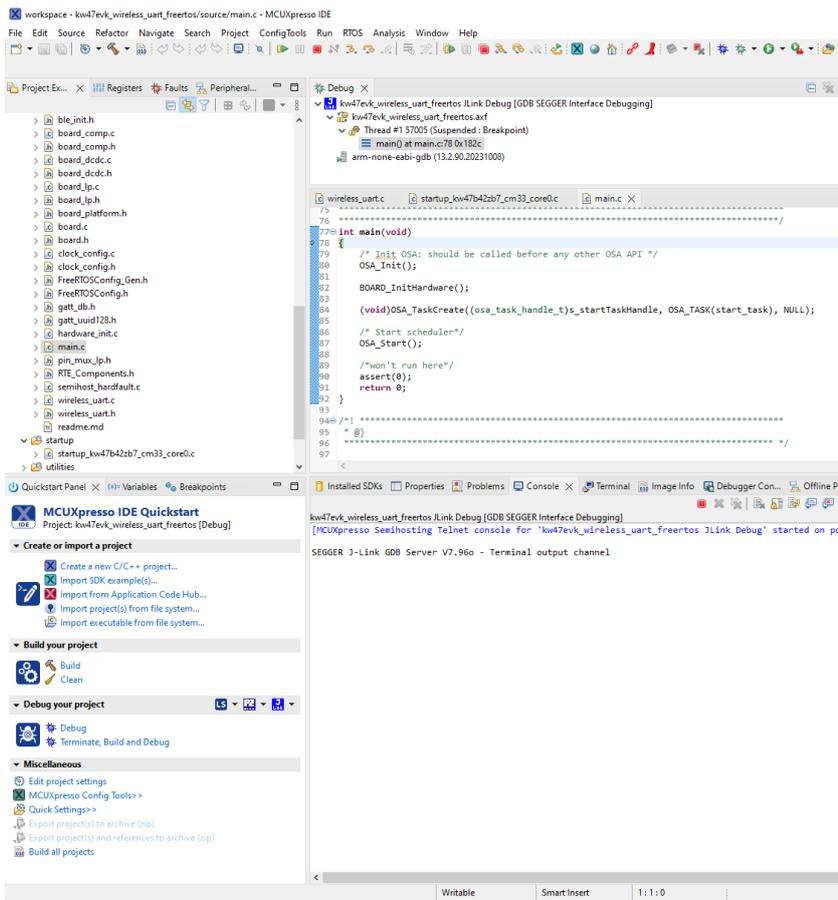
**Build the Wireless UART FreRTOS project**

- 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 FreRTOS project**



7. Pressing the **Run** button makes the board run the application.

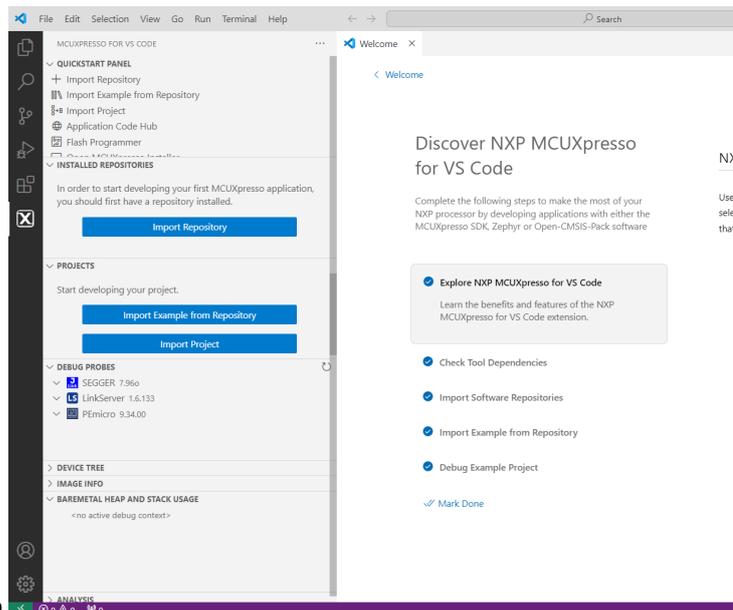


**Running the code on MCUXpresso IDE**

Parent topic: [Building the binaries](#)

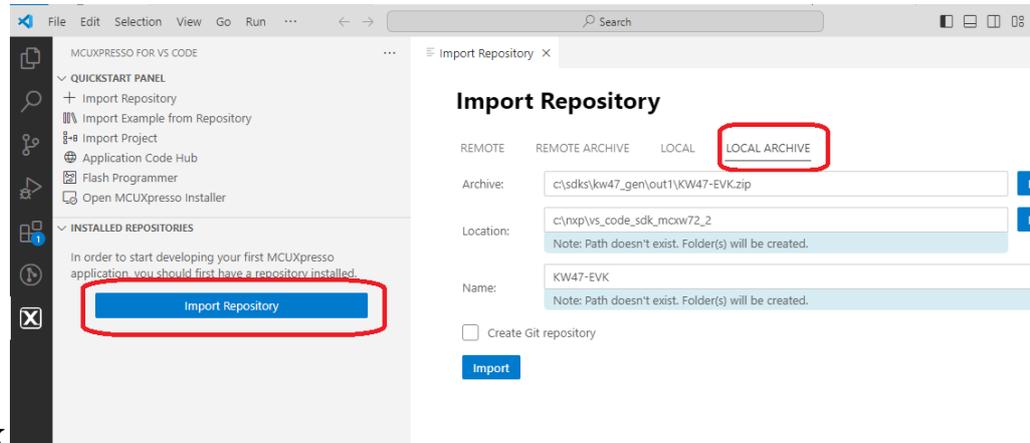
**Building and flashing the BLE software demo applications using Visual Studio Code** To build and flash the BLE software demo applications using Visual Studio Code, follow the steps listed below:

1. Open Visual Studio Code and open the MCUXpresso for Visual Studio Code extension as shown in the figure below.



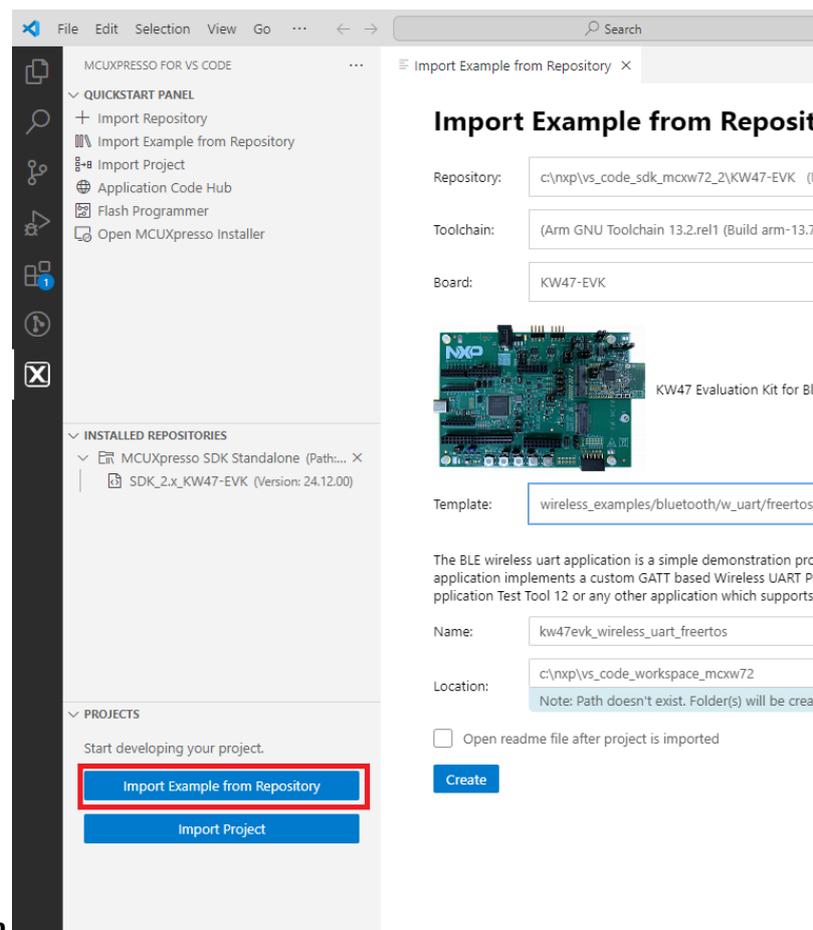
**MCUXpresso for Visual Studio Code extension**

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”**.



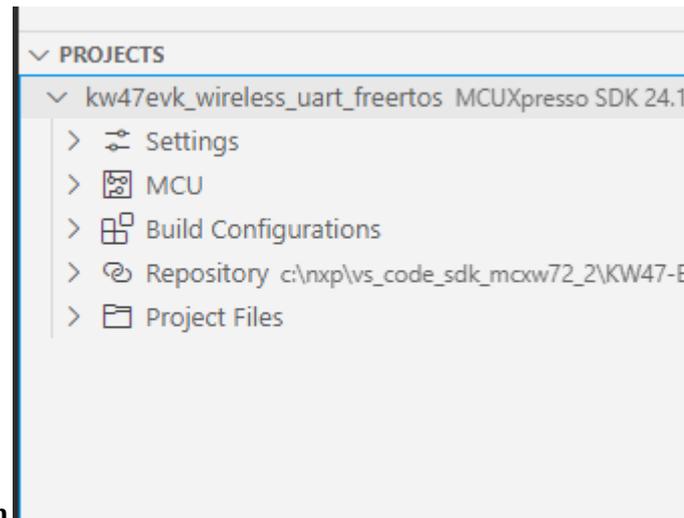
### Steps to import the SDK

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”**.



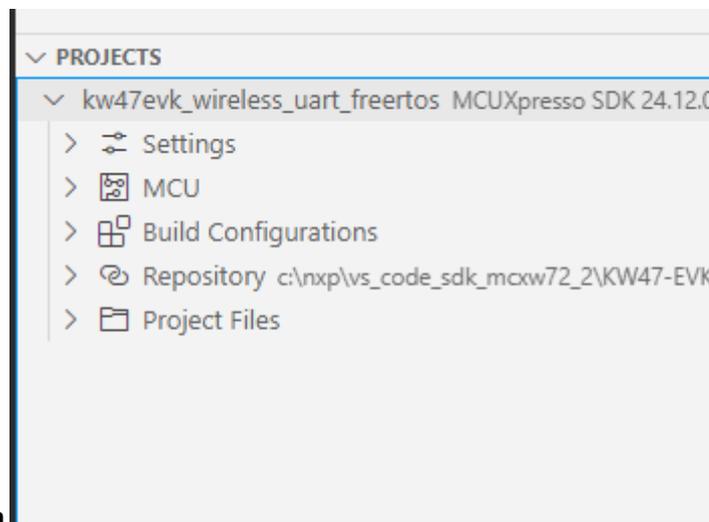
### Steps to import the example application

4. The application now appears in the **“Projects”** tab on the left. Build the application by pressing the **“Build selected”** button.



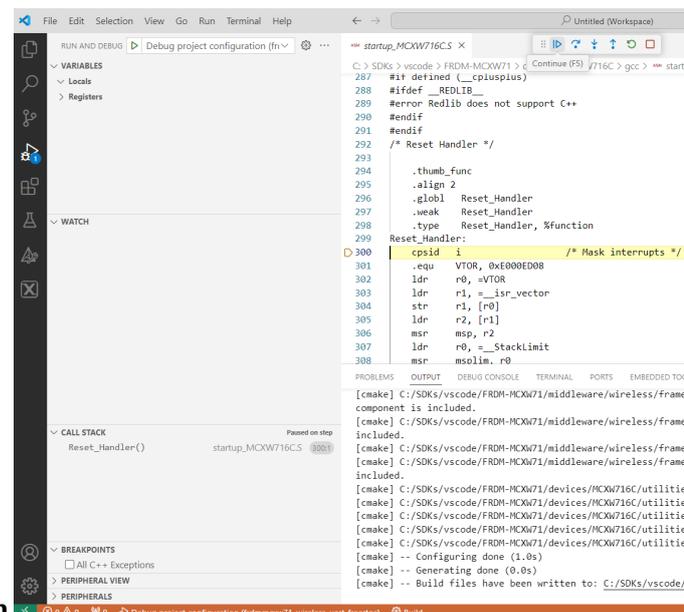
### Building the Wireless UART FreeRTOS application

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

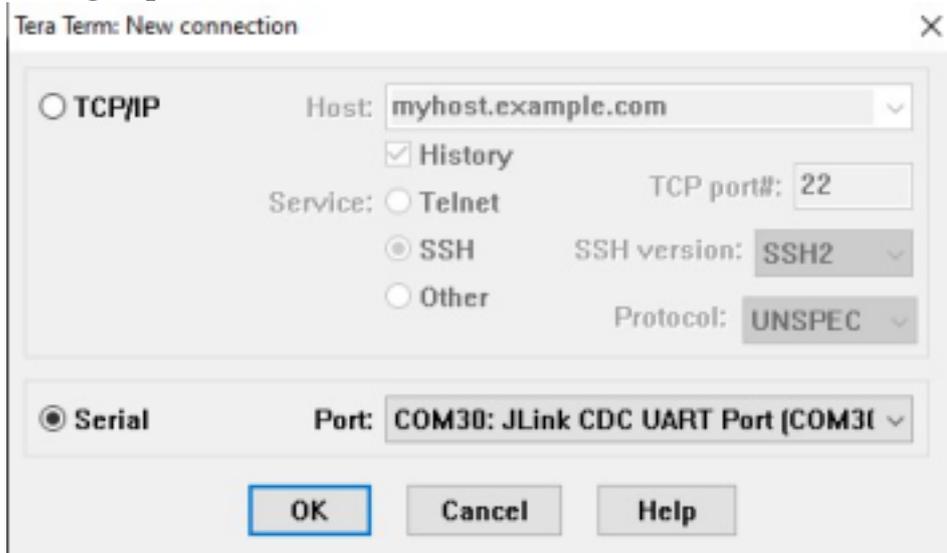
Parent topic: [Building the binaries](#)

**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.

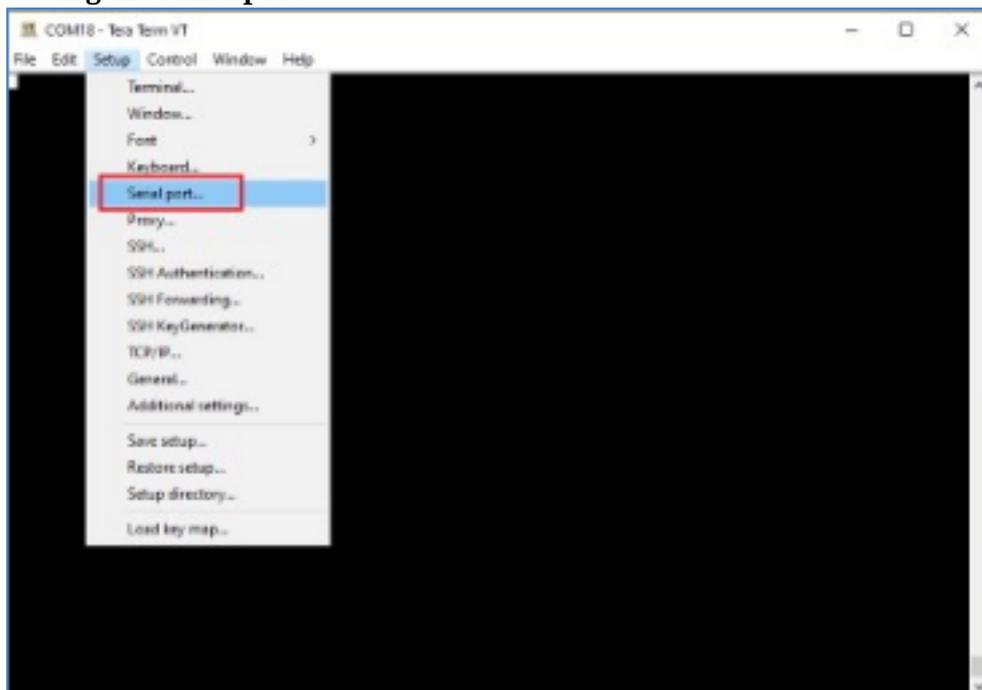
1. Flash the board with the Wireless UART application as previously described.
2. Open Tera Term (or any other Serial Communication software) and choose **Serial**. Then choose the port of your board and click **OK**.

**Setting up a new Tera Term connection for Wireless UART application**



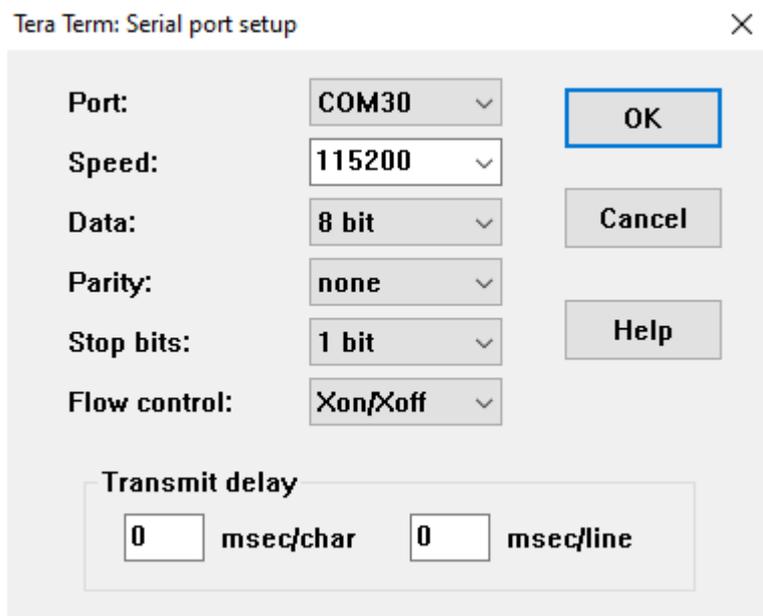
3. Stop running the code on the board. Go to **Setup >Serial Port**.

**Setting the serial port**



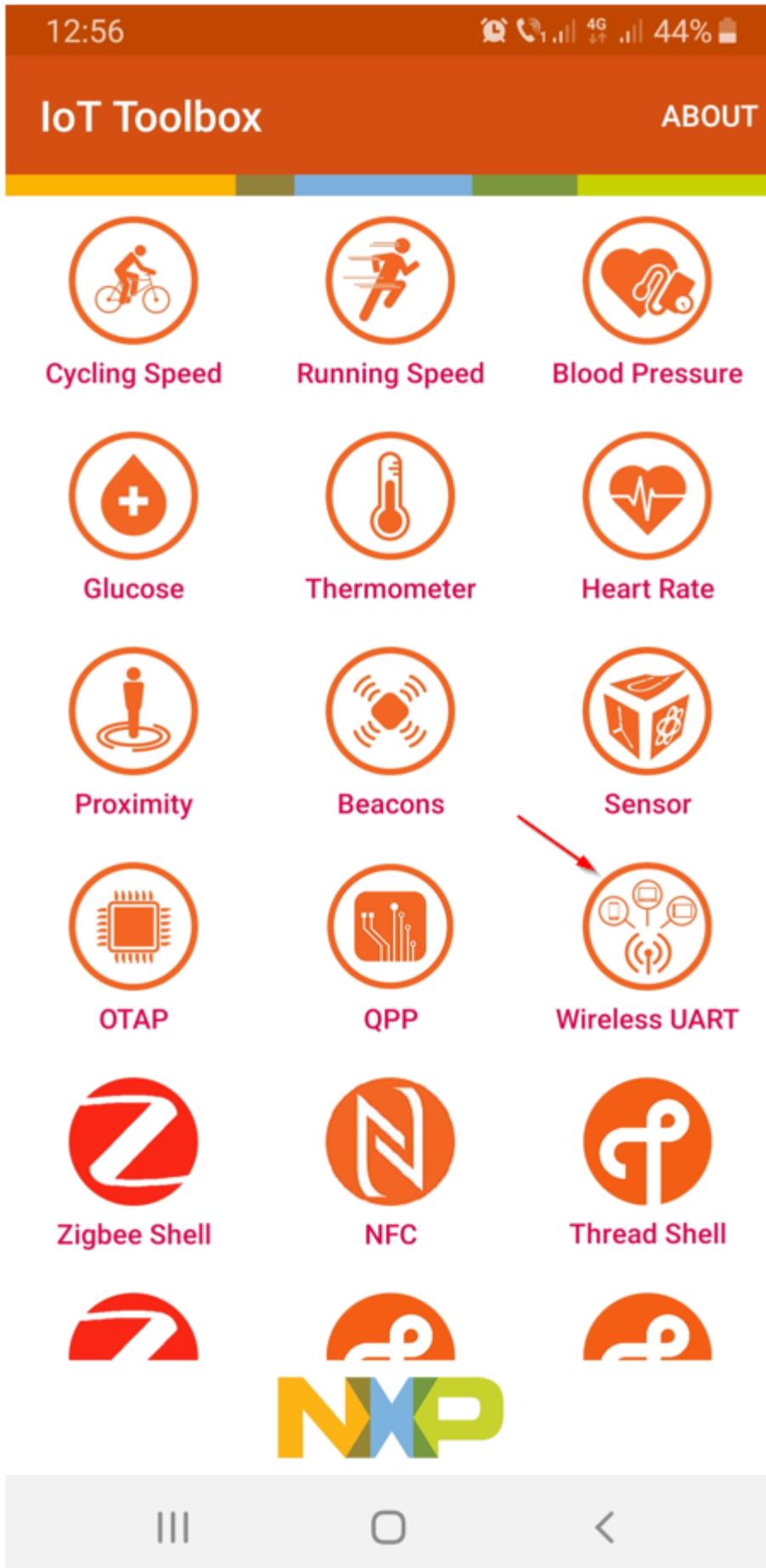
4. Choose the required Speed (baud rate) and Flow Control values. Here, the baud rate should be 115200 and the Flow control mode should be set to **Xon/Xoff**.

**Setting the baud rate**



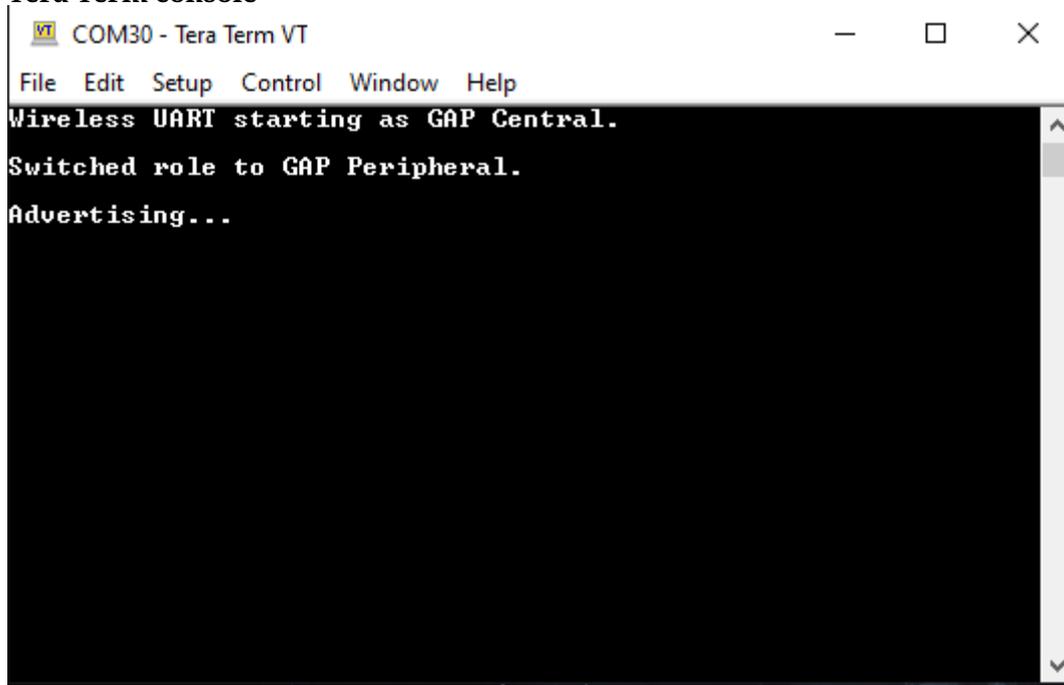
5. Ensure that the Bluetooth device of your phone is enabled.
6. Open the *IoT Toolbox* application and select the **Wireless UART** icon.

**NXP IoT Toolbox application**



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**



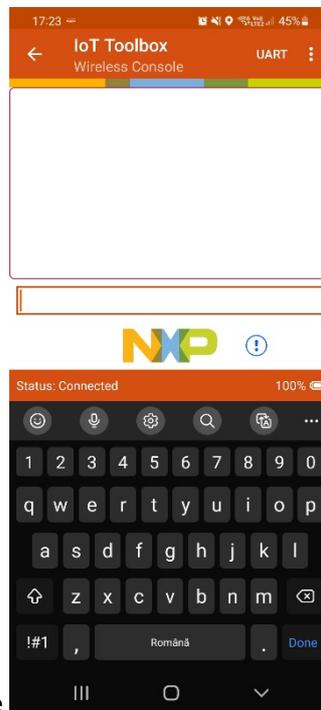
8. The device should become visible for the Wireless UART mobile application.



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.

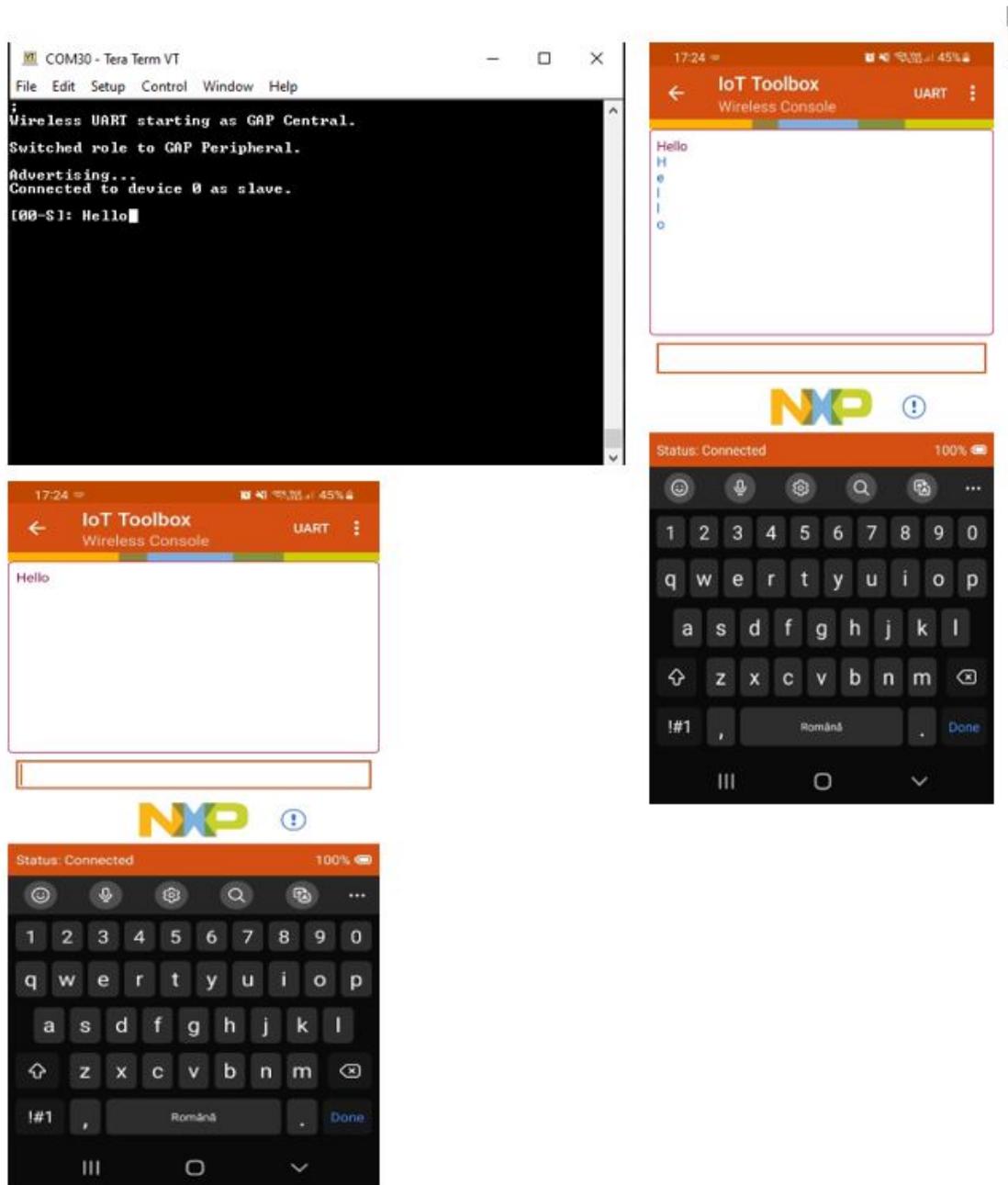


#### Wireless UART working in Connected mode

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:

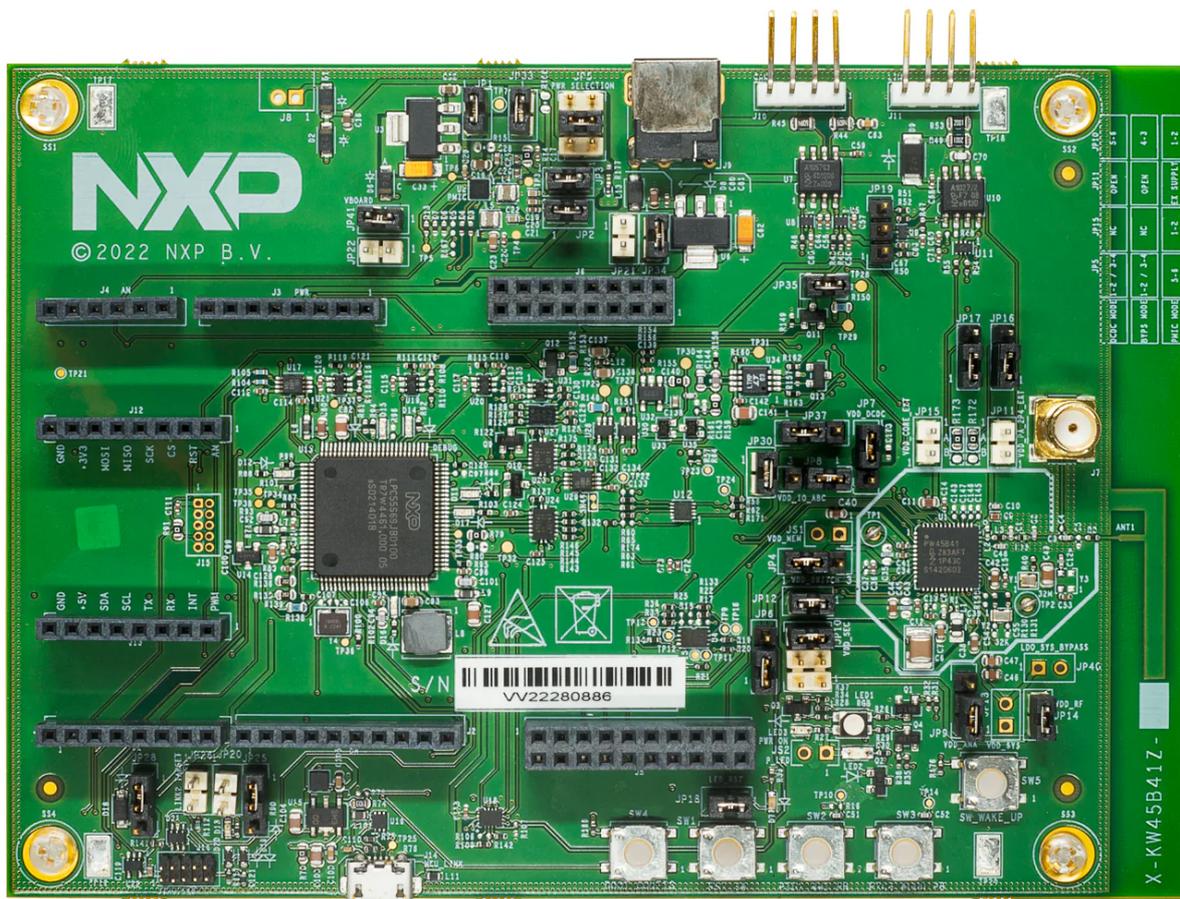
- KW45B41Z-EVK
- KW45B41Z-LOC
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- KW47-LOC
- FRDM-MCXW72
- MCX-W72-EVK
- MCXW72-LOC
- FRDM-MCXW23
- MCXW23-EVK

**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.

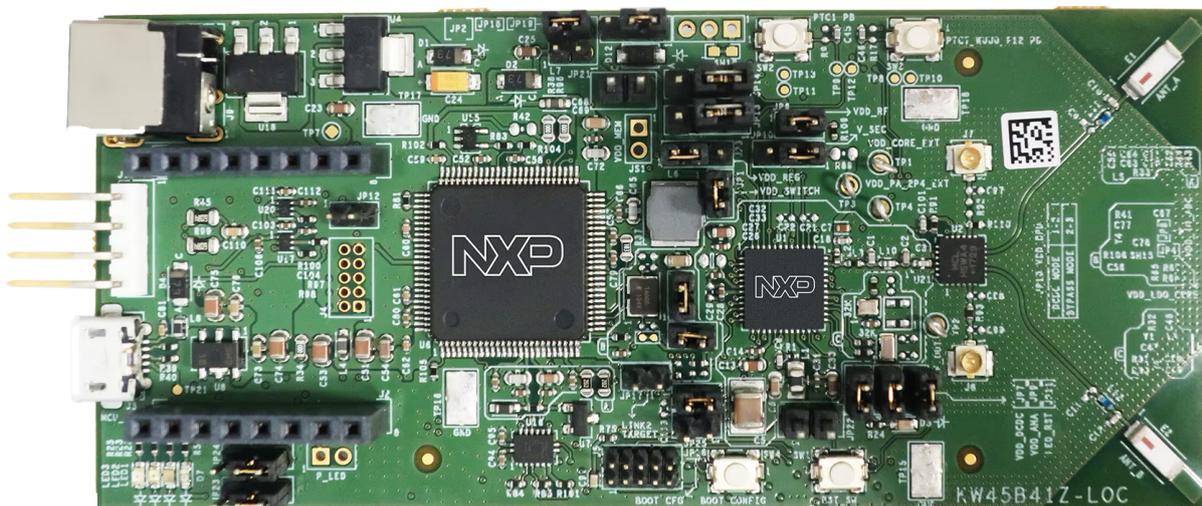
**Parent topic:**[Hardware configurations](#)

**KW45B41Z-EVK platform** The figure below displays the top view of the KW45B41Z-EVK board.

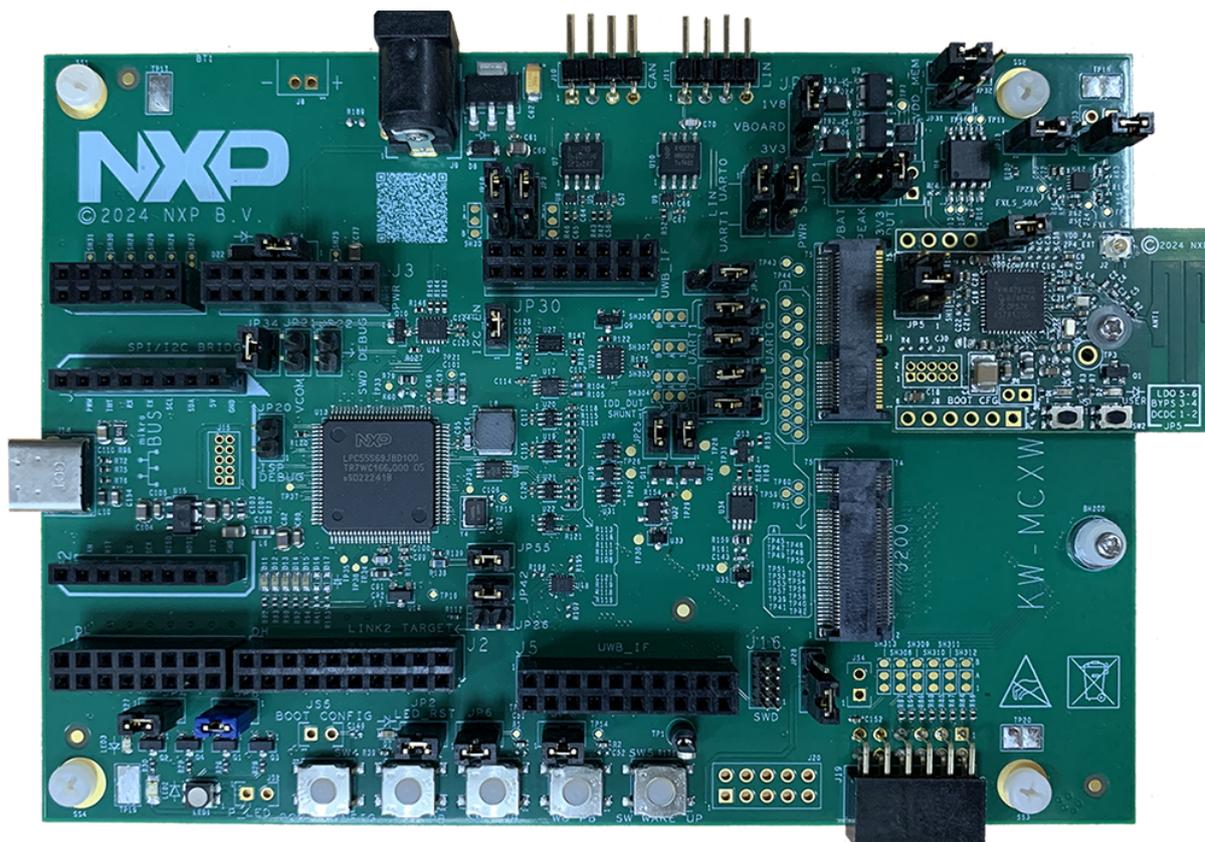


**Parent topic:**[Hardware configurations](#)

**KW45B41Z-LOC platform** The figure below displays the top view of the KW45B41Z-LOC board.

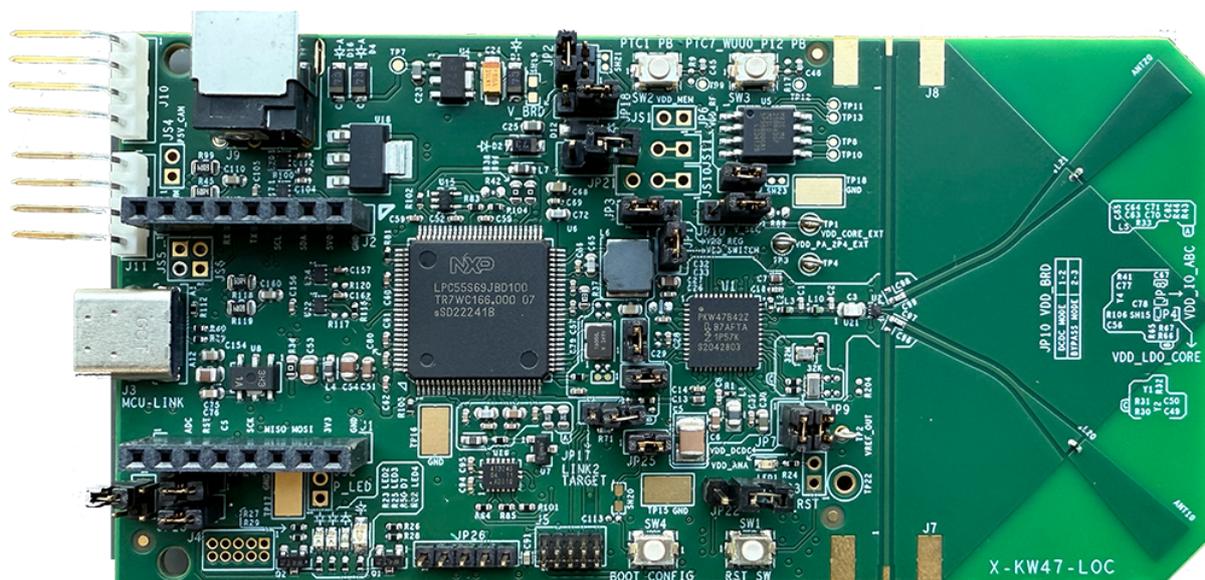






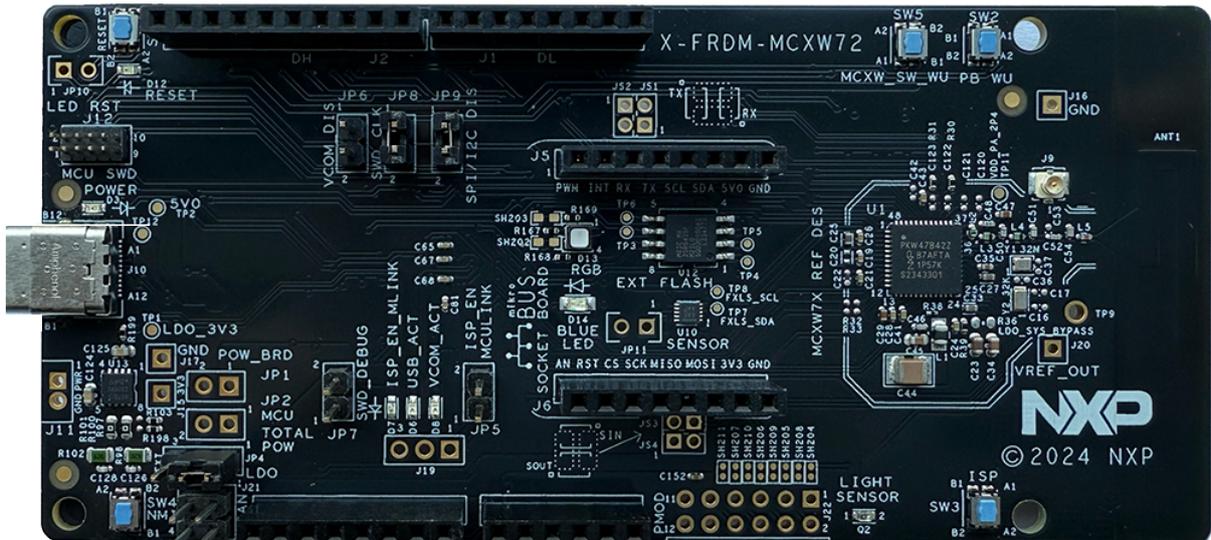
Parent topic:[Hardware configurations](#)

**KW47-LOC platform** The figure below displays the top view of the KW47-LOC board.



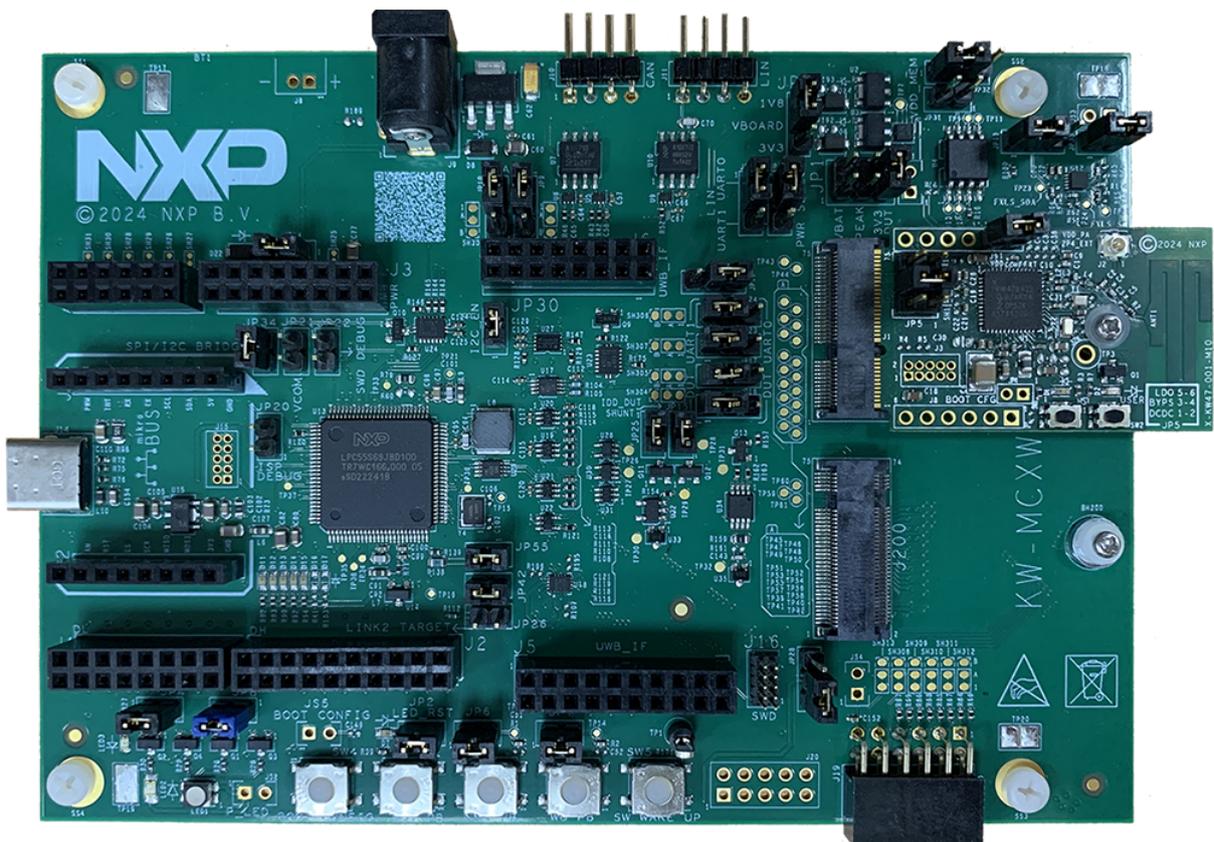
Parent topic:[Hardware configurations](#)

**FRDM-MCXW72 platform** The figure below displays the top view of the FRDM-MCXW72 board.



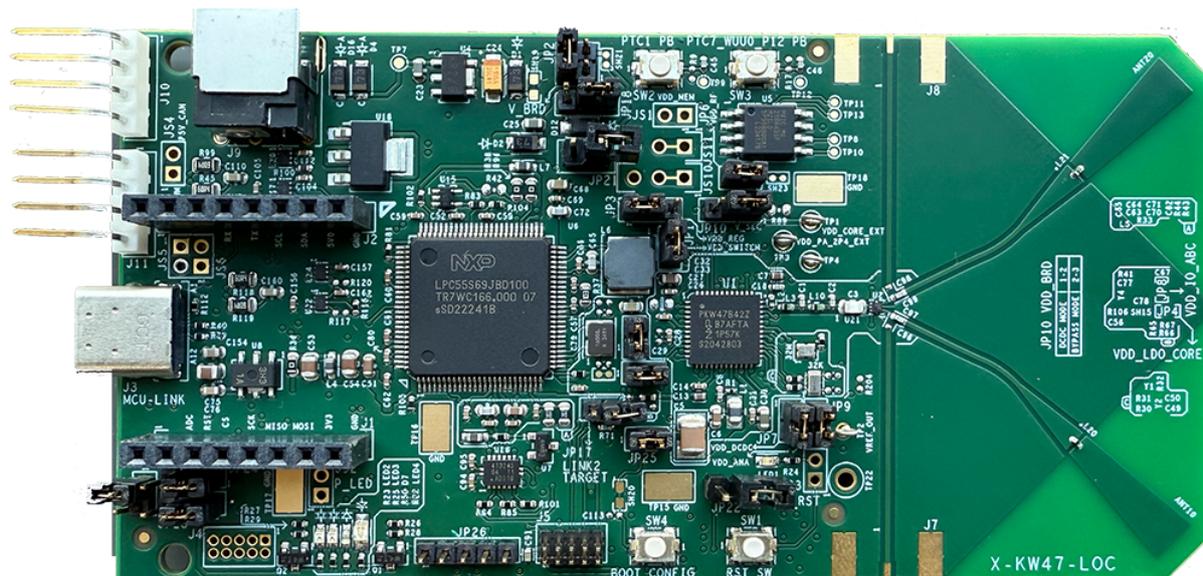
Parent topic:[Hardware configurations](#)

MCX-W72-EVK platform The figure below displays the top view of the MCX-W72-EVK board.



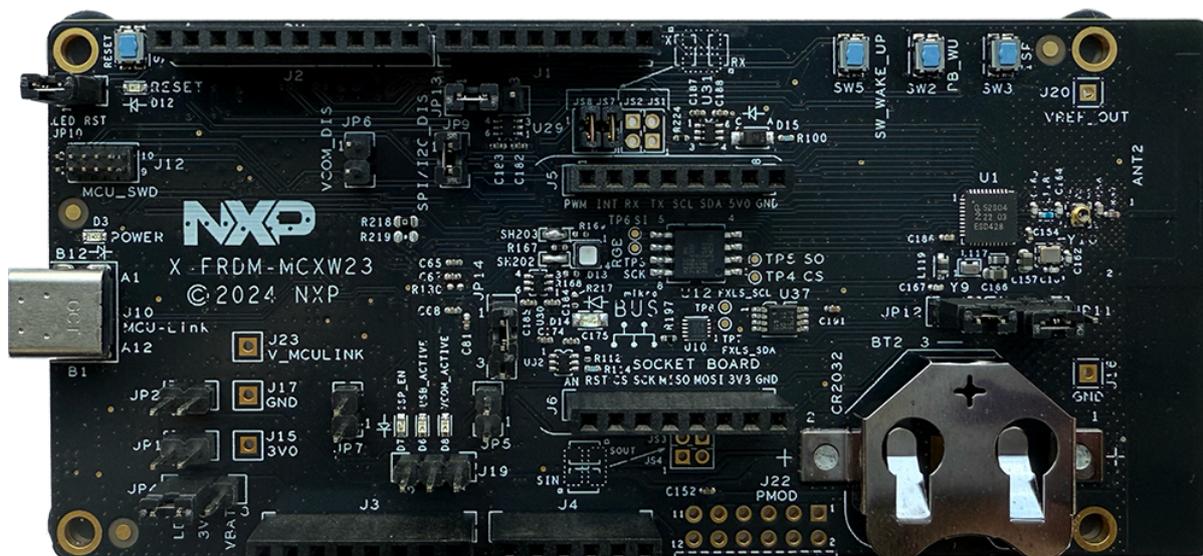
Parent topic:[Hardware configurations](#)

MCXW72-LOC platform The figure below displays the top view of the MCXW72-LOC board.



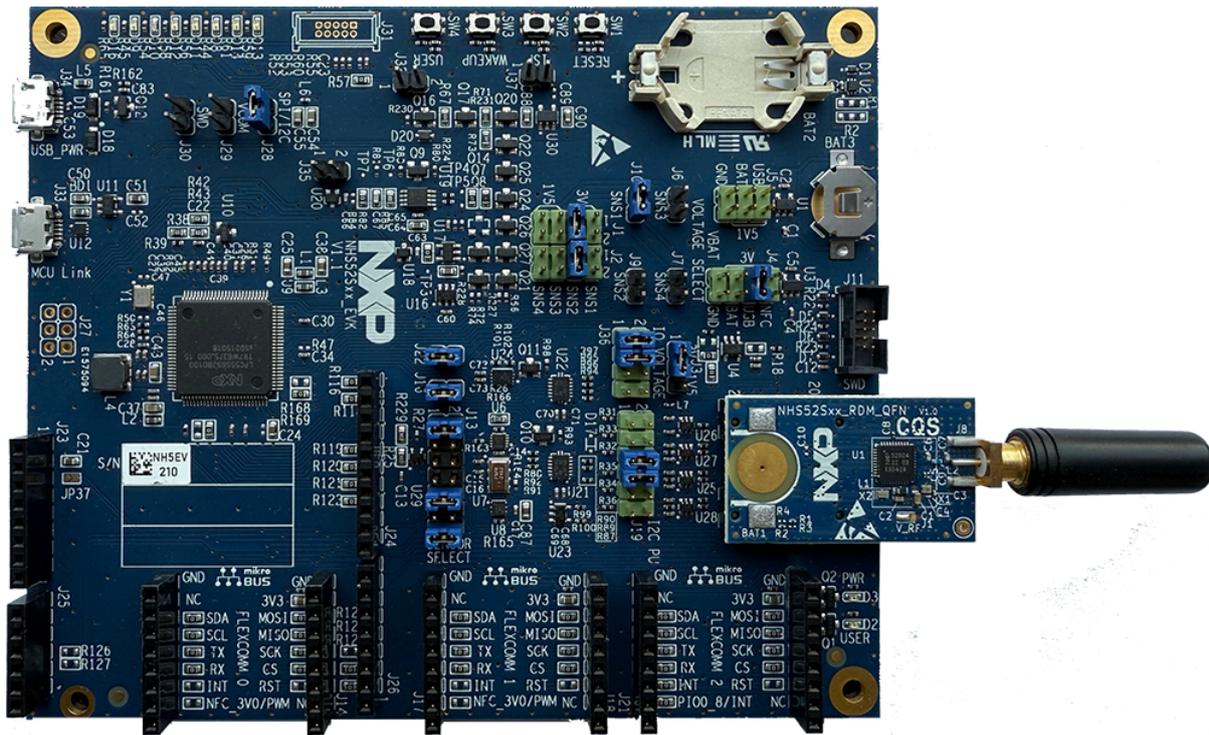
Parent topic:[Hardware configurations](#)

**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



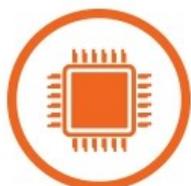
Proximity



Beacons



Sensor



OTAP



QPP



Wireless UART



Zigbee Shell

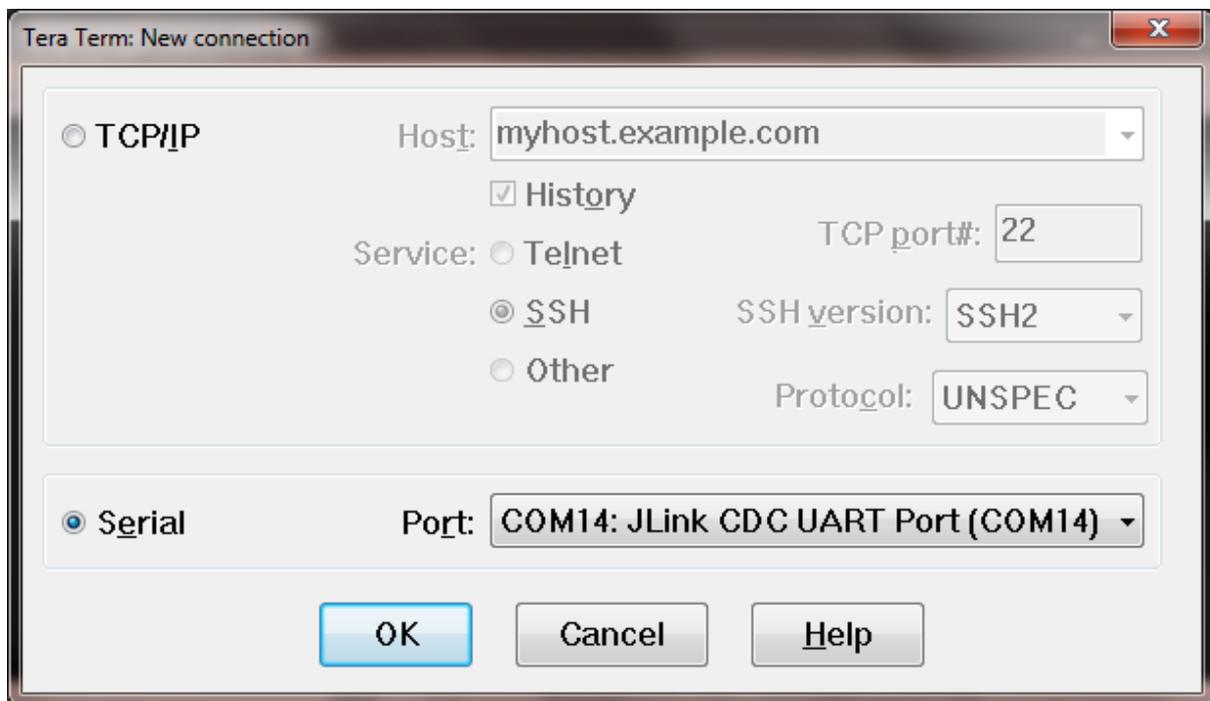


The recommended application is the IoT Toolbox, which can be installed on Apple iOS or Android OS handheld devices that support Bluetooth Low Energy. The application can be found on [Apple Playstore](#) or on [Google Play](#).

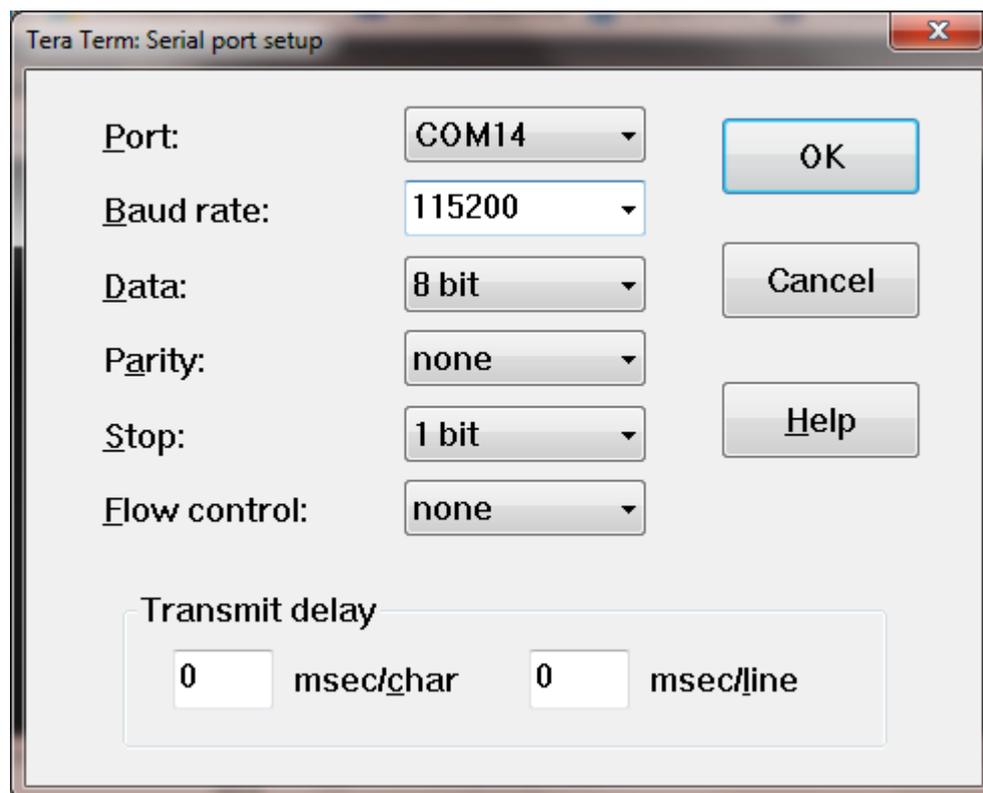
Other demos can be run by using two platforms, one for the peripheral and one for the central role. A few examples of such demos are listed below:

- 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

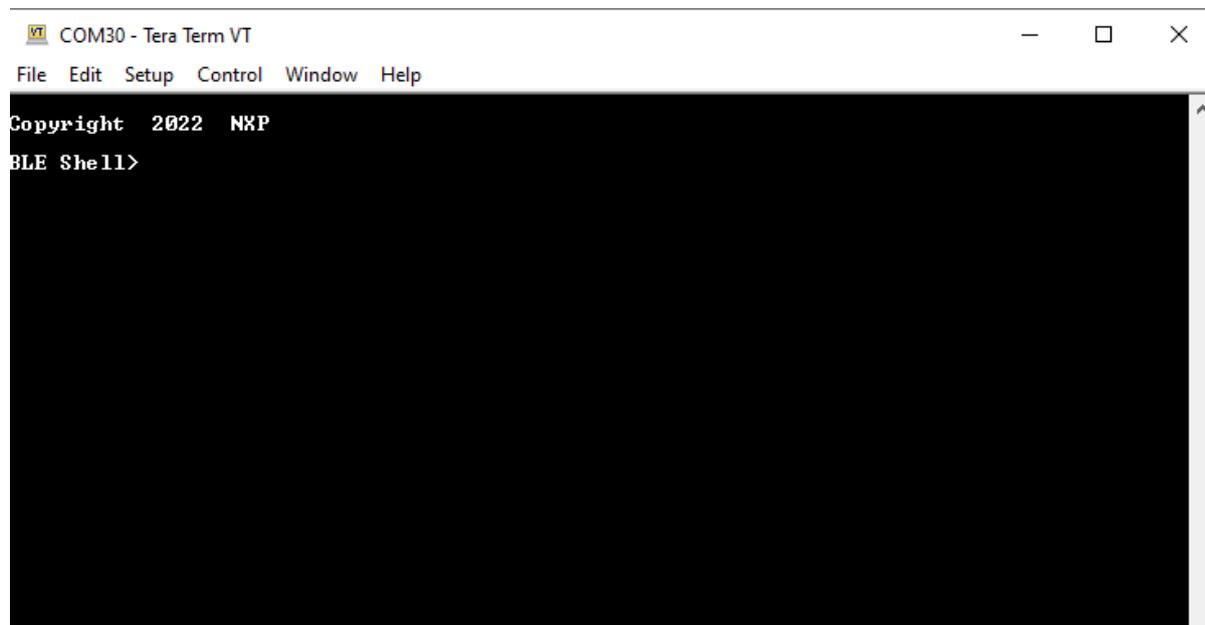
To provide feedback and more interaction, some examples use a shell console via the virtual COM port. To access the device, open a serial port terminal and as shown in the figure below. For this example, Tera Term VT and a KW45B41Z-EVK or a FRDM-MCXW71 board can be used. The communication parameters 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)

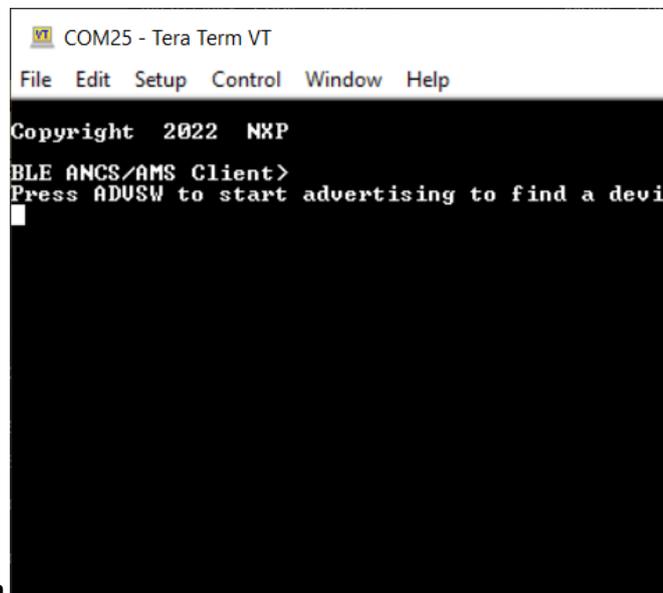
**User interface** After flashing the board, the device is in idle mode (all LEDs flashing). To start advertising, press the **ADVS** button. When in GAP Discoverable Mode, **CONNLED** is flashing. When the ANCS/AMS Server (Gap Central) connects to the ANCS/AMS Client (GAP Peripheral), **CONNLED** turns solid. To disconnect, hold the **ADVS** 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)

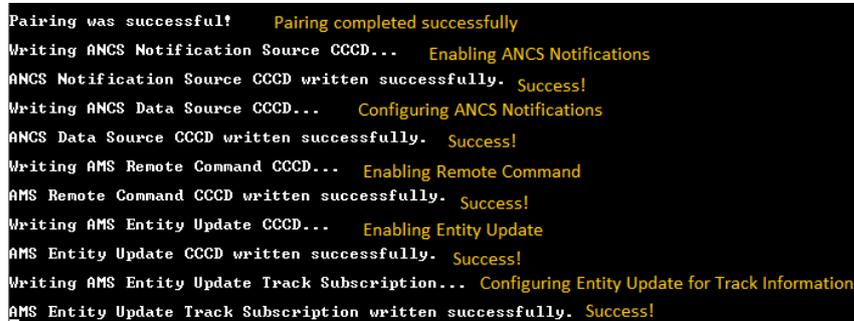
**Usage** The ANCS/AMS Client demo application is designed to work with a peer mobile device that exposes the ANCS and AMS service. Also, a serial terminal application is required for displaying ANCS Notifications information and AMS commands and information.

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.

```

BLE ANCS Client>
Notif_UID Flags Category Application
0x00000000 S_E_N Social Hangouts
0x00000001 S__N Social Hangouts
0x00000002 S__N Social Hangouts
0x00000003 ___N Email Hangouts

BLE ANCS Client>
Notif_UID Flags Category Application
0x00000000 S_E_N Social Hangouts
0x00000001 S__N Social Hangouts
0x00000002 S__N Social Hangouts
0x00000003 ___N Email Mail

BLE ANCS Client>
Notif_UID Flags Category Application
0x00000000 S_E_N Social Hangouts
0x00000001 S__N Social Hangouts
0x00000002 S__N Social Hangouts

BLE ANCS Client>
Notif_UID Flags Category Application
0x00000001 S__N Social Hangouts
0x00000002 S__N Social Hangouts

BLE ANCS Client>
Notif_UID Flags Category Application
0x00000002 S__N Social Hangouts

BLE ANCS Client>
Notif_UID Flags Category Application
0x00000004 ___N Social Hangouts

BLE ANCS Client>
Notif_UID Flags Category Application
0x00000004 ___N Social Hangouts

BLE ANCS Client>

```

**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.

```

COM25 - Tera Term VT
File Edit Setup Control Window Help
Notif_UID Flags Category Application
AMS
Artist :
Album :
Title :
Duration:
Available commands: VolumeUp VolumeDown
BLE ANCS/AMS Client>

```

### AMS no active player

- When media is playing, a player is active and the state must look similar to the figure below.

```

COM3 - Tera Term VT
File Edit Setup Control Window Help
Notif_UID Flags Category Application
0x00000001 N Other Home Assistant
0x00000002 N Social Messages
AMS
Artist : Jain, Ian Asher
Album : Makeba <Ian Asher Remix>
Title : Makeba - Ian Asher Remix
Duration: 125.178
Available commands: Play Pause TogglePlayPause NextTrack PreviousTrack VolumeUp VolumeDown SkipForward SkipBackward
BLE ANCS/AMS Client>

```

### 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

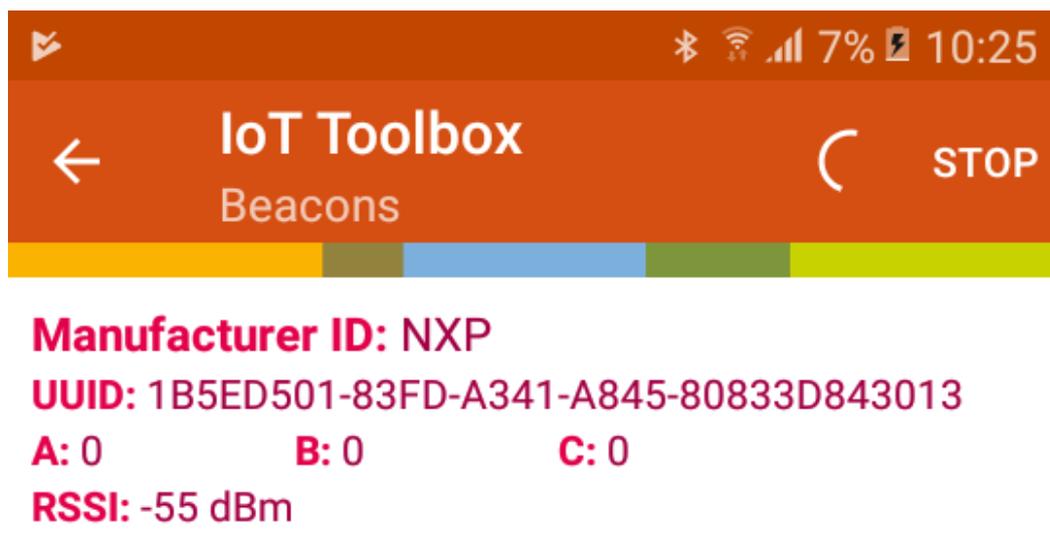
**Parent topic:**Beacon

**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 `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.



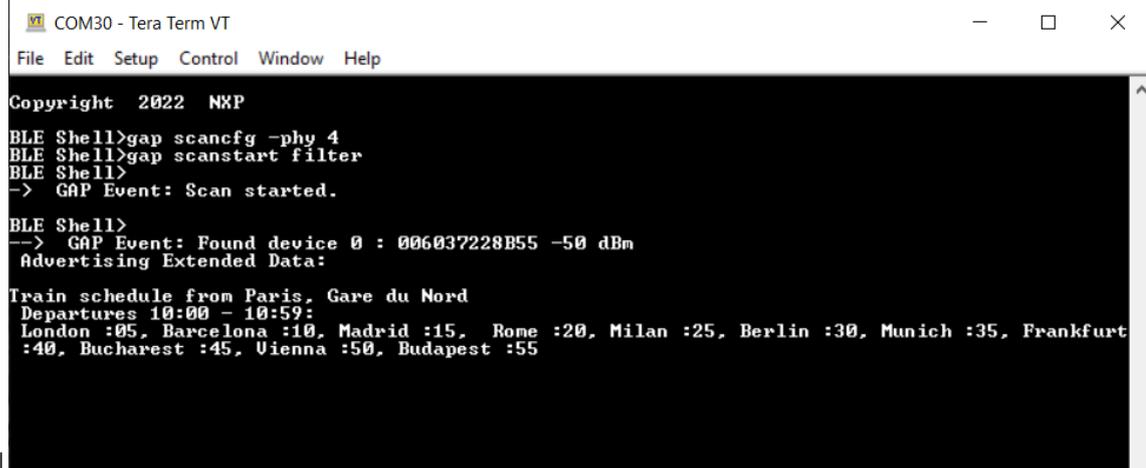
**Parent topic:** Beacon

**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 **ADVS**W press starts legacy advertising, **CONNLED** turns solid.
- The second **ADVS**W press stops legacy advertising and starts extended advertising, **CONNLED** turns off, **EXTADVLED** turns solid.
- The third **ADVS**W press stops extended advertising carrying data and then starts extended advertising without data and periodic advertising, **EXTADVLED** starts flashing.
- The fourth **ADVS**W press stops periodic advertising and extended advertising without data and starts legacy advertising and extended advertising, both **CONNLED** and **EXTADVLED** turn solid.
- The fifth **ADVS**W 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 **ADVS**W 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.



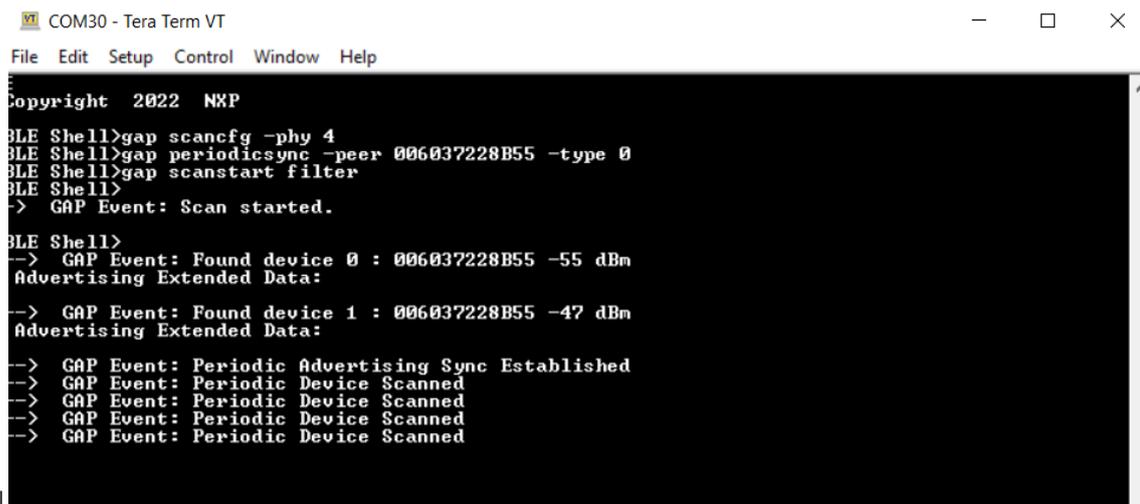
```

COM30 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2022 NXP
BLE Shell>gap scanfg -phy 4
BLE Shell>gap scanstart filter
BLE Shell>
-> GAP Event: Scan started.

BLE Shell>
-> GAP Event: Found device 0 : 006037228B55 -50 dBm
Advertising Extended Data:
Train schedule from Paris, Gare du Nord
Departures 10:00 - 10:59:
London :05, Barcelona :10, Madrid :15, Rome :20, Milan :25, Berlin :30, Munich :35, Frankfurt
:40, Bucharest :45, Vienna :50, Budapest :55

```

5. To start the periodic advertising, press **ADVS**W 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.



```
COM30 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2022 NXP
BLE Shell>gap scancfg -phy 4
BLE Shell>gap periodicsync -peer 006037228B55 -type 0
BLE Shell>gap scanstart filter
BLE Shell>
--> GAP Event: Scan started.
BLE Shell>
--> GAP Event: Found device 0 : 006037228B55 -55 dBm
Advertising Extended Data:
--> GAP Event: Found device 1 : 006037228B55 -47 dBm
Advertising Extended Data:
--> GAP Event: Periodic Advertising Sync Established
--> GAP Event: Periodic Device Scanned
```

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.

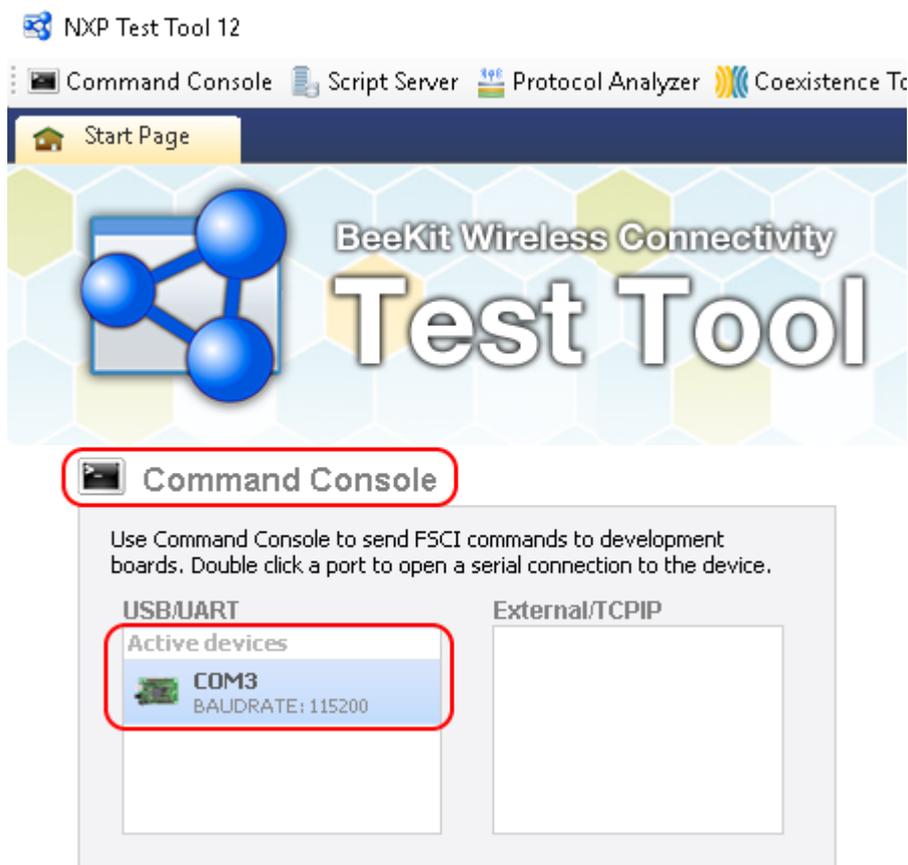


- FRDM-MCXW72
- MCX-W72-EVK

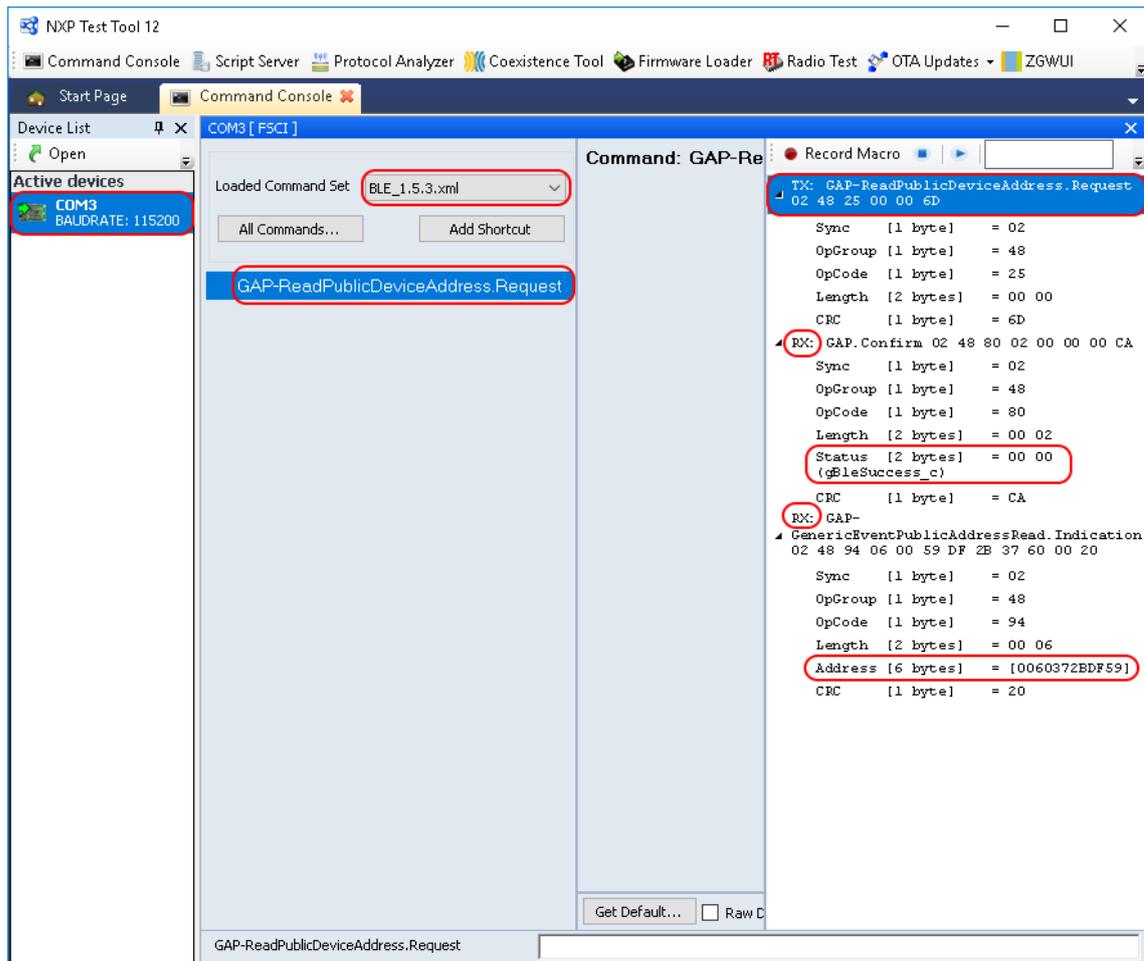
**Parent topic:**Bluetooth LE FSCI Black Box

**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

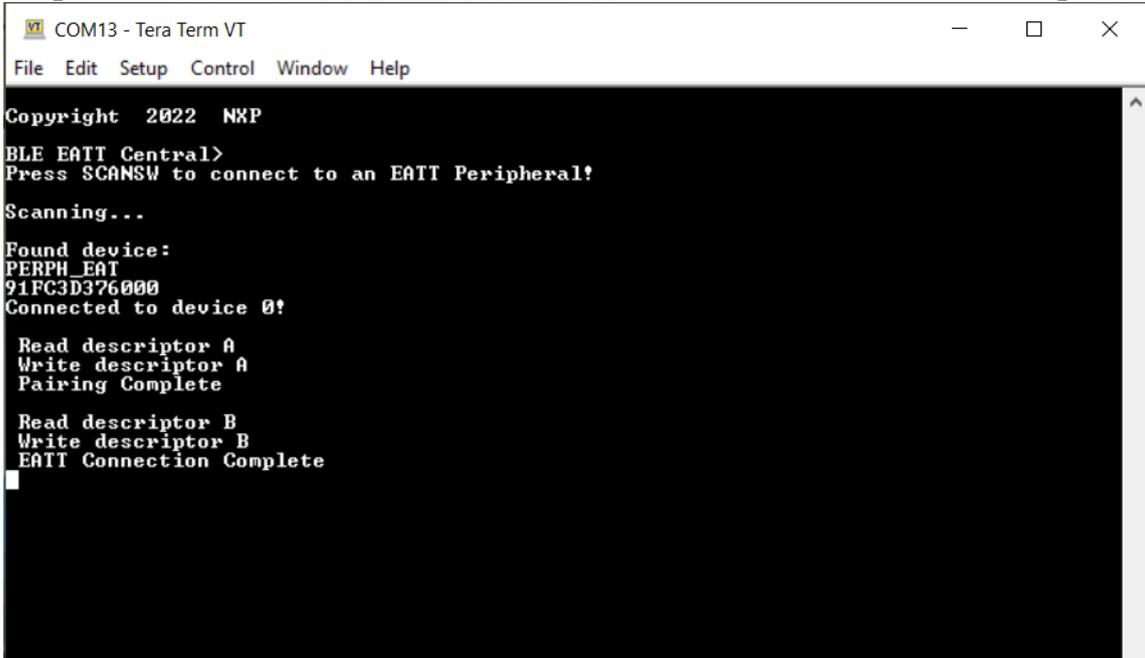
**User interface** After flashing the board, the central is in Idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. After connecting to the peripheral, the **CONNLED** turns solid. The data information together with the bearer it was received on is sent over UART. To disconnect 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



```
COM13 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2022 NXP
BLE EATT Central>
Press SCANSW to connect to an EATT Peripheral!
Scanning...
Found device:
PERPH_EAT
91FC3D376000
Connected to device 0!
Read descriptor A
Write descriptor A
Pairing Complete
Read descriptor B
Write descriptor B
EATT Connection Complete
```

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.

```

COM13 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2022 NXP
BLE EATT Central>
Press SCANSW to connect to an EATT Peripheral!
Scanning...
Found device:
PERPH_EAT
91FC3D376000
Connected to device 0!

Read descriptor A
Write descriptor A
Pairing Complete

Read descriptor B
Write descriptor B
EATT Connection Complete

Received value of Service A: 0000 On bearer: 01
Received value of Service B: 0000 On bearer: 02
Received value of Service A: 0100 On bearer: 01
Received value of Service B: 0100 On bearer: 02
Received value of Service A: 0200 On bearer: 01
Received value of Service B: 0200 On bearer: 02
Received value of Service A: 0300 On bearer: 01
Received value of Service B: 0300 On bearer: 02
Received value of Service A: 0400 On bearer: 01
Received value of Service B: 0400 On bearer: 02
Received value of Service A: 0500 On bearer: 01
Received value of Service B: 0500 On bearer: 02
Received value of Service A: 0600 On bearer: 01
Received value of Service B: 0600 On bearer: 02
Received value of Service A: 0700 On bearer: 01
Received value of Service B: 0700 On bearer: 02
Received value of Service A: 0800 On bearer: 01
Received value of Service B: 0800 On bearer: 02
Received value of Service A: 0900 On bearer: 01
Received value of Service B: 0900 On bearer: 02
Received value of Service A: 0000 On bearer: 01
Received value of Service B: 0000 On bearer: 02

```

### 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 `hctool` in Linux, check the article [“FRDM-KW40Z Bluetooth LE Controller Usage with the Linux hctool”](#).

**Parent topic:**HCI Black Box

**Supported platforms** The following platforms support the HCI Black Box application:

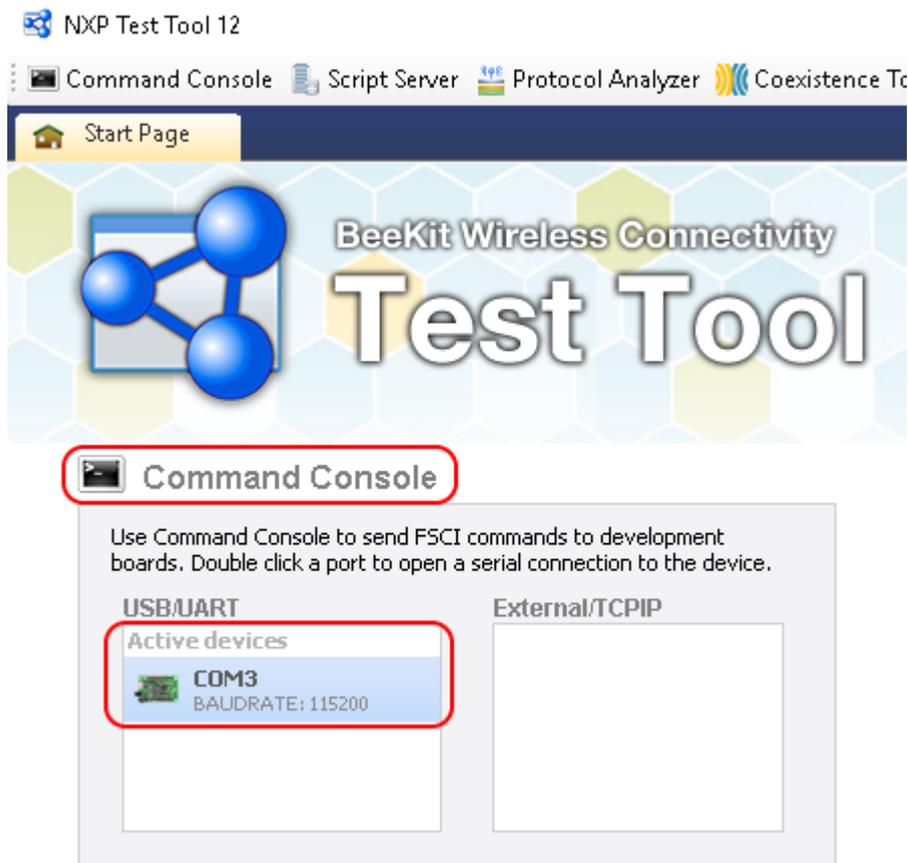
- KW45B41Z-EVK
- FRDM-MCXW71

- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK

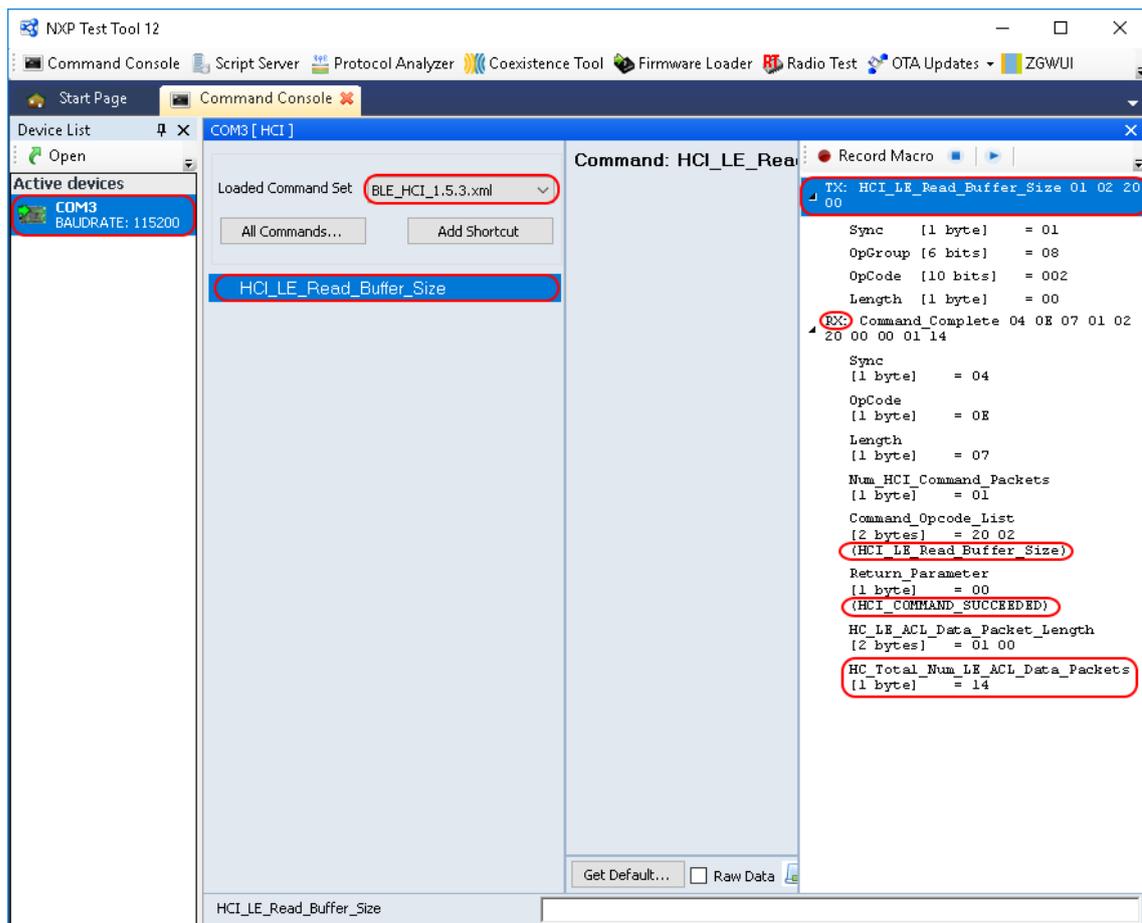
**Parent topic:** HCI Black Box

**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

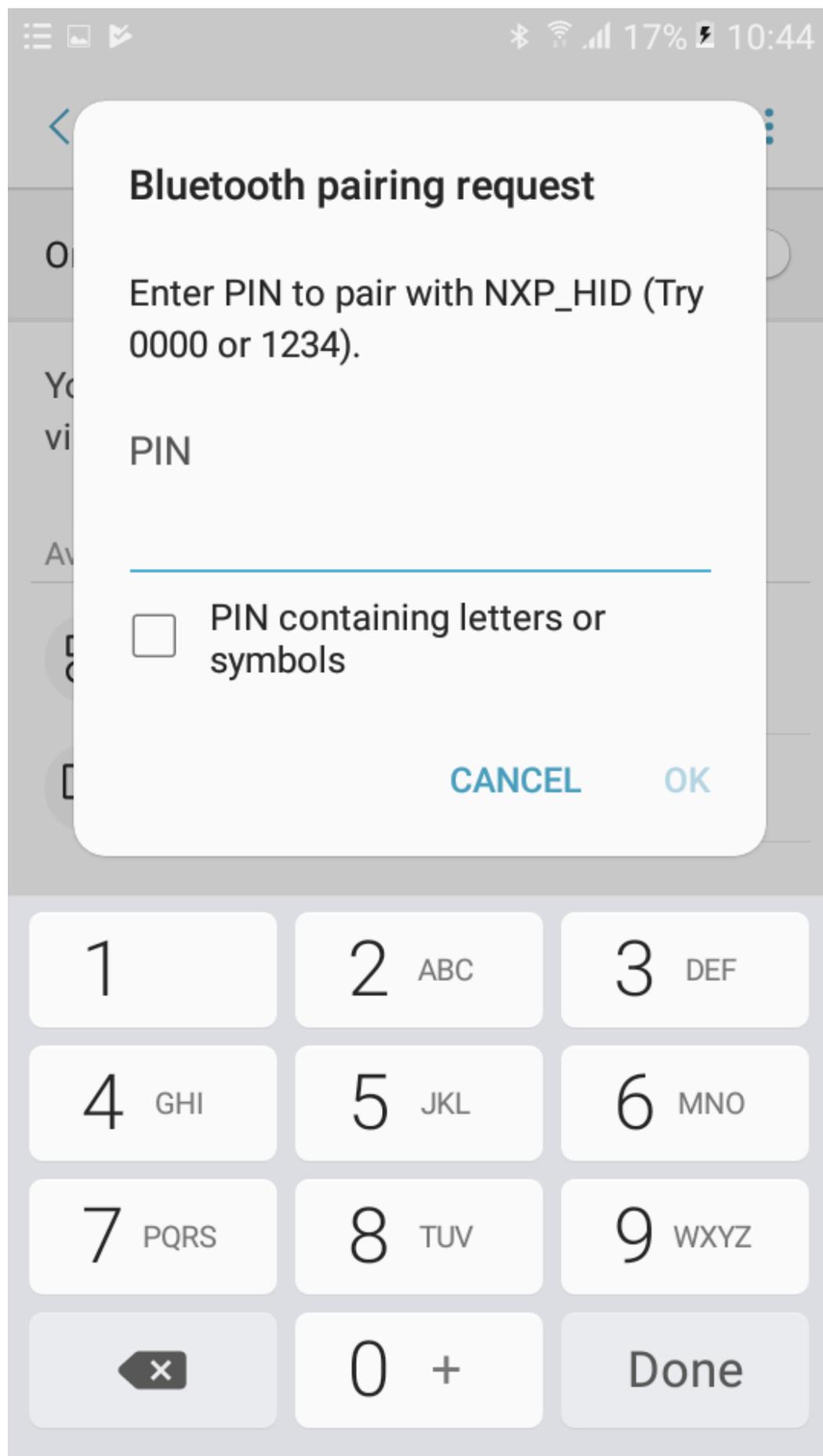
**Parent topic:**HID Device (Mouse)

**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)

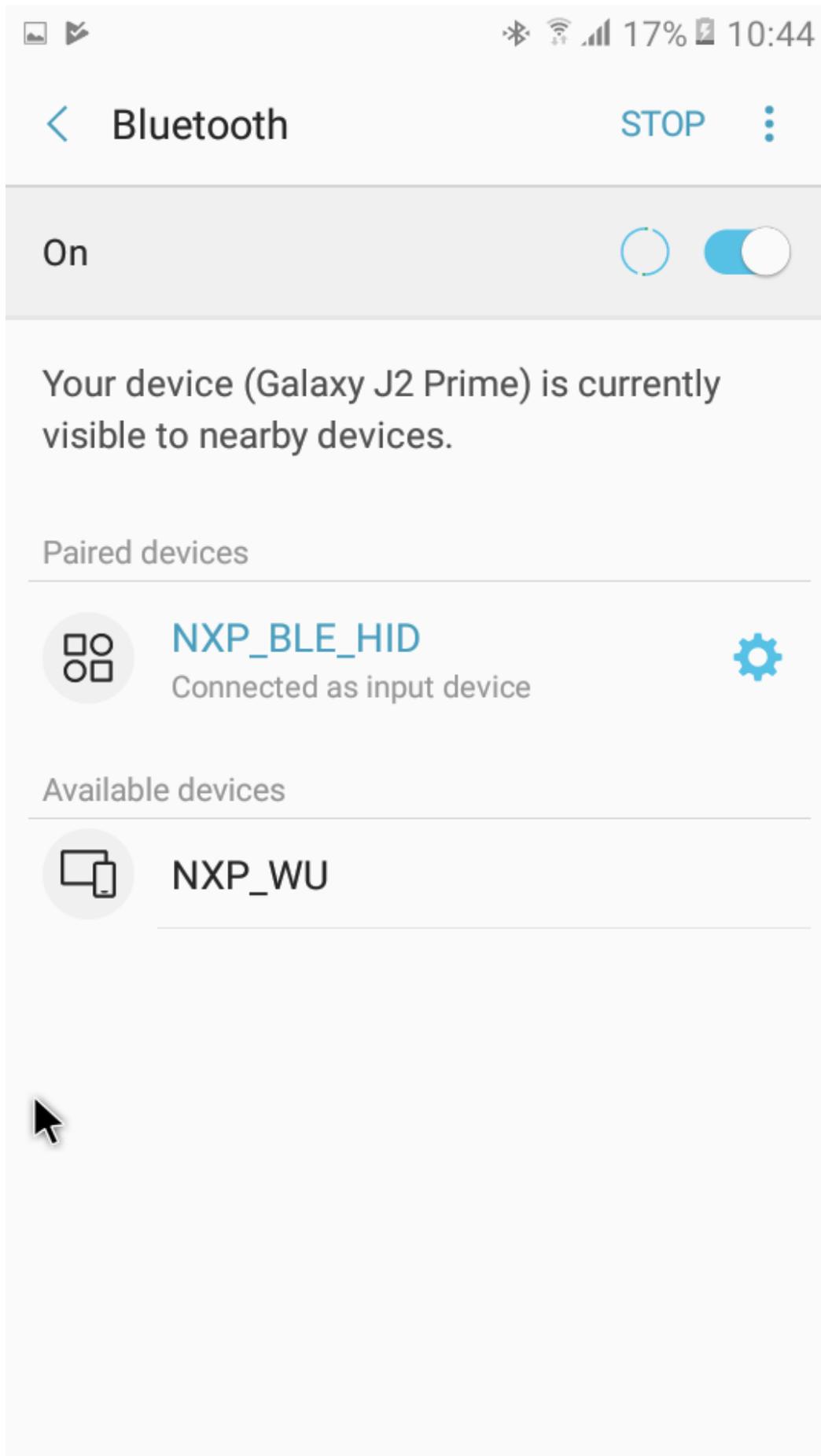
**Usage** The HID mouse can be connected to any Bluetooth Smart Ready products available on the market that supports HID devices or to another supported platform running the HID Host example (setup steps detailed in the HID Host section).

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.



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.



**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.



**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.

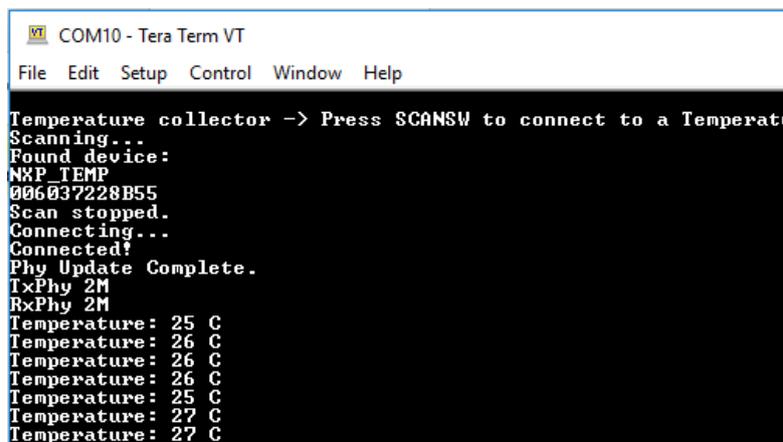
The user can manually change this configuration and enable LED support, else all subsequent LED behavior references are ignored and all LEDs are off. The devices disconnect and enter Deep-sleep only if low power is enabled. When the node is awake and communicating, **CONNLED** is on. To wake up the node, 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.



```
COM10 - Tera Term VT
File Edit Setup Control Window Help
Temperature collector -> Press SCANSW to connect to a Temperat
Scanning...
Found device:
NXP_TEMP
006037228B55
Scan stopped.
Connecting...
Connected?
Phy Update Complete.
TxPhy 2M
RxPhy 2M
Temperature: 25 C
Temperature: 26 C
Temperature: 26 C
Temperature: 26 C
Temperature: 25 C
Temperature: 27 C
Temperature: 27 C
```

#### 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_central`s 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:

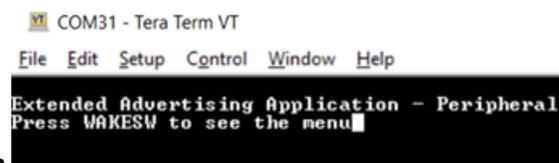
**baud rate 115200, data bits 8, parity none, stop bits 1.**

**Parent topic:**Low-power extended advertising Peripheral and Central

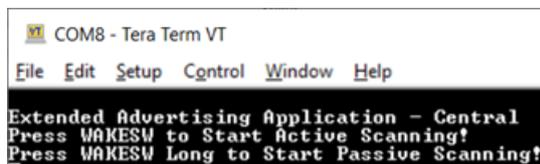
**Usage** The setup requires two supported platforms, one for the `adv_ext_peripheral`, and one for the `adv_ext_central`.

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

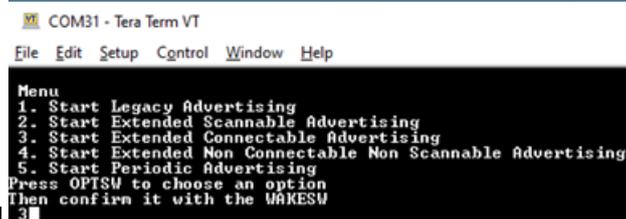


Advertizing external Central start screen



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.

```

COM156 - Tera Term VT
File Edit Setup Control Window Help
Passive Scanning Started
Extended Advertising Found
Adv Properties:
Non Connectable
Non Scannable
Undirected
Adv Data
Extended Advertising
Adv Address CA70364AD91F
Data Set Id = 4
PrimaryPHY = gLePhyCoded_c
SecondaryPHY = gLePhyCoded_c
periodicAdvInterval = 2400
Adv Data
Decrypted data :
EA Non Connectable Non Scannable DataId1 01 EA Non Connectable N
EA Non Connectable Non Scannable DataId1 03 EA Non Connectable N
Decrypted data :
EA Non Connectable Non Scannable DataId1 11 EA Non Connectable N
EA Non Connectable Non Scannable DataId1 13 EA Non Connectable N
Decrypted data :
EA Non Connectable Non Scannable DataId1 21 EA Non Connectable N
EA Non Connectable Non Scannable DataId1 23 EA Non Connectable N
Decrypted data :
EA Non Connectable Non Scannable DataId1 31 EA Non Connectable N
EA Non Connectable Non Scannable DataId1 33 EA Non Connectable N
Gap_PeriodicAdvCreateSync Succeeded
Periodic Adv Sync Established
Periodic Adv Found
syncHandle = 80
Periodic Data
Decrypted data :
EA Periodic Data Id1 01 EA Periodic Data Id1 02 EA Periodic Dat
EA Periodic Data Id1 05 EA Periodic Data Id1 06 EA Periodic Dat
Decrypted data :
EA Periodic Data Id1 11 EA Periodic Data Id1 12 EA Periodic Dat
EA Periodic Data Id1 15 EA Periodic Data Id1 16 EA Periodic Dat
Decrypted data :
EA Periodic Data Id1 21 EA Periodic Data Id1 22 EA Periodic Dat
EA Periodic Data Id1 25 EA Periodic Data Id1 26 EA Periodic Dat
Stop Scanning
Periodic Adv Sync Terminated

```

### 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 Scannable DataId1 01 EA Non Connectable Non Scannable DataId1 02
EA Non Connectable Non Scannable DataId1 03 EA Non Connectable Non Scannable DataId1 04
Gap_PeriodicAdvCreateSync Succeeded
Periodic Adv Sync Established

```

(continues on next page)

(continued from previous page)

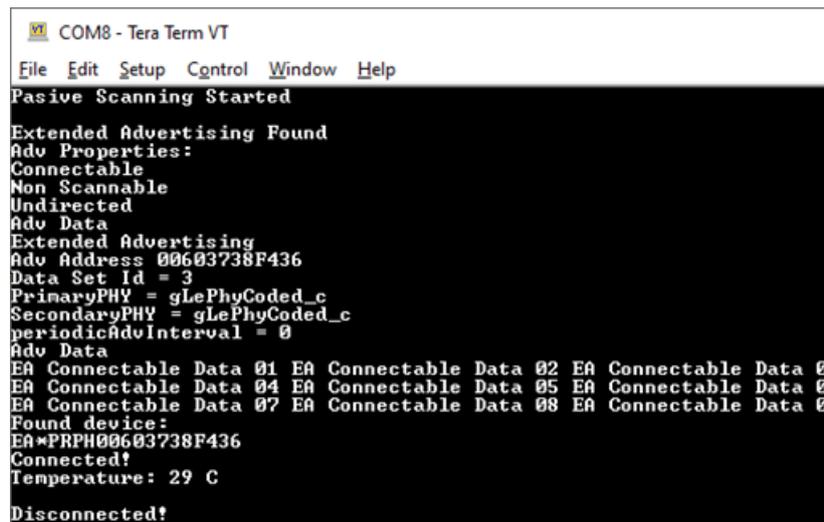
```

Gap_SetPeriodicSyncSubevent Succeeded
PAWR received
Sync Handle: 0050
Event Counter: 0007
Subevent: 0
Subevent data:
PAWR Data0 For Subevent 0
PAWR Data1 For Subevent 0
Gap_SetPeriodicAdvResponseData Succeeded
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.



```

COM8 - Tera Term VT
File Edit Setup Control Window Help
Pasive Scanning Started
Extended Advertising Found
Adv Properties:
Connectable
Non Scannable
Undirected
Adv Data
Extended Advertising
Adv Address 00603738F436
Data Set Id = 3
PrimaryPHY = gLePhyCoded_c
SecondaryPHY = gLePhyCoded_c
periodicAdvInterval = 0
Adv Data
EA Connectable Data 01 EA Connectable Data 02 EA Connectable Data 03
EA Connectable Data 04 EA Connectable Data 05 EA Connectable Data 06
EA Connectable Data 07 EA Connectable Data 08 EA Connectable Data 09
Found device:
EA*PRPH00603738F436
Connected!
Temperature: 29 C
Disconnected!

```

#### 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 Succeeded
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:

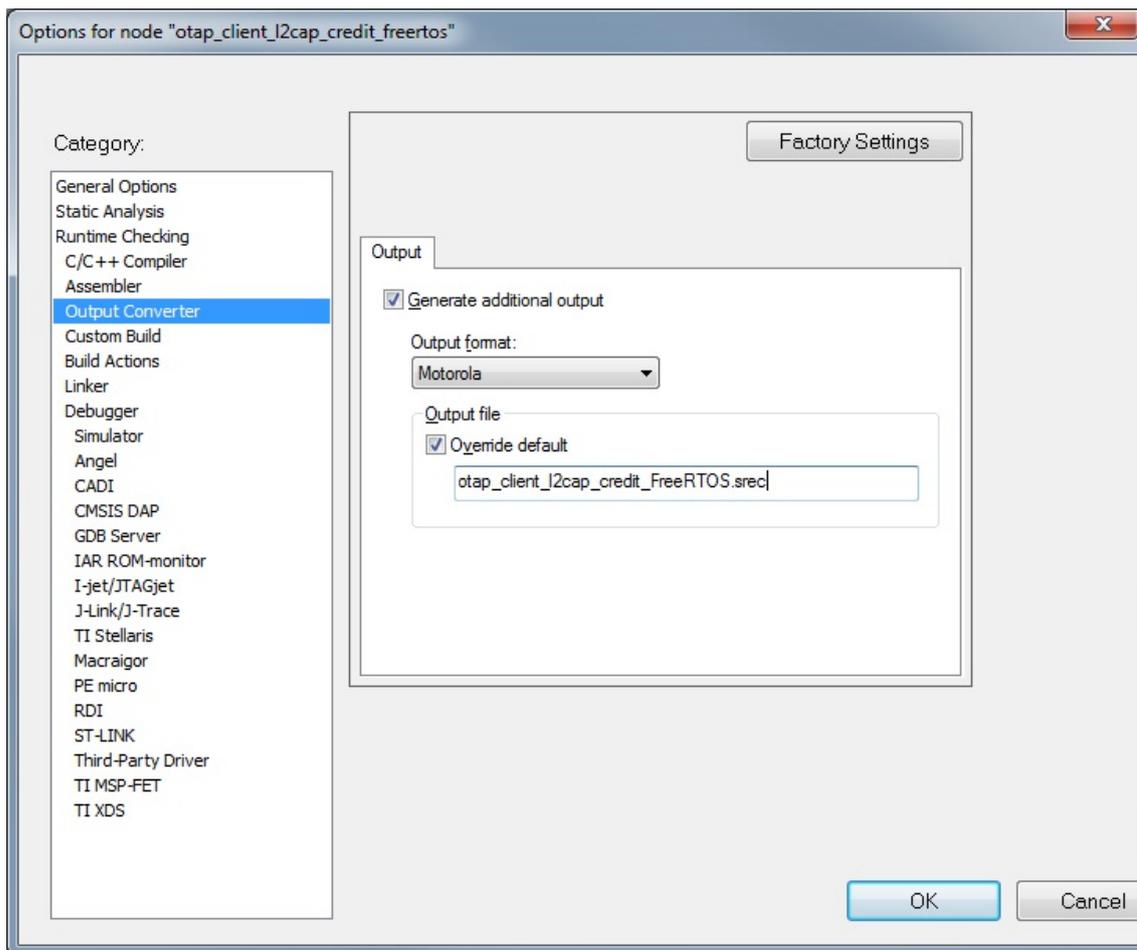
- Over The Air Programming Tool 1.4.0 or newer on [CONNECTIVITY-TOOL-SUITE](#)
- Serial COM port drivers – these are board-specific.

To run the application, follow the steps below:

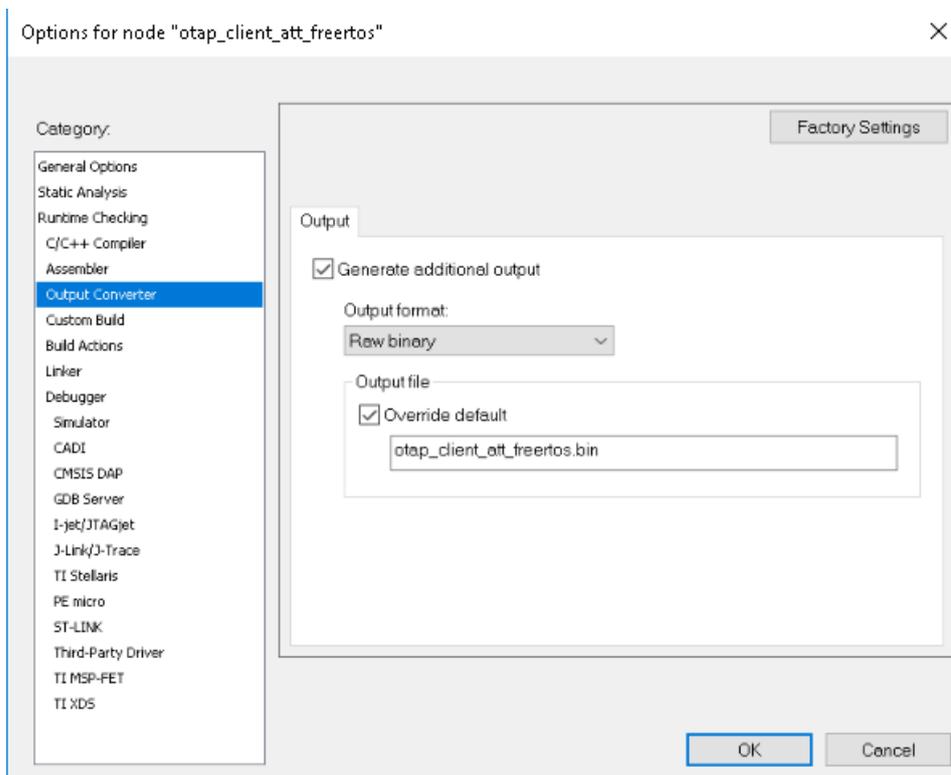
1. Flash the OTAP Server onto a supported platform and the OTAP Client to another supported platform. Make sure the board running the OTAP Server is connected to your PC and your PC has appropriate drivers for the USB to serial device on that board.
2. Create the application to send over the air. The 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.



- To obtain a .bin file from MCUXpresso IDE, go to the **Project properties -> Settings -> Build stepswindow** and press the **Edit** button for the Post-build steps. A **Post-build steps** window shows up. In this window, add the following command:

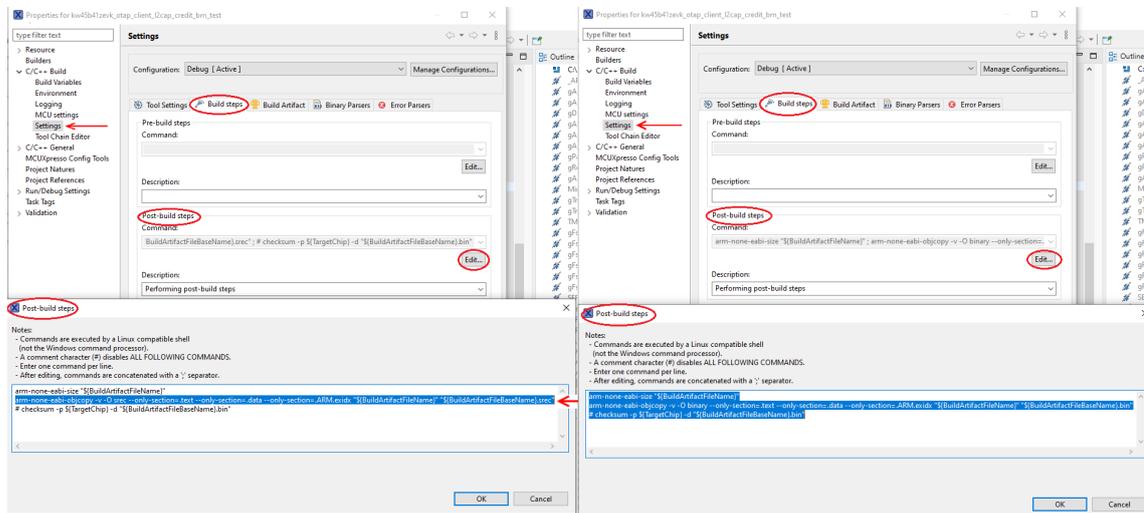
```
arm-none-eabi-objcopy -v -O binary --only-section=.text
--only-section=.data
--only-section=.ARM.exidx "${BuildArtifactFileName}"
"${BuildArtifactFileName}.bin"
```

In case the command already exists, uncomment it by removing the '#' character at the beginning.

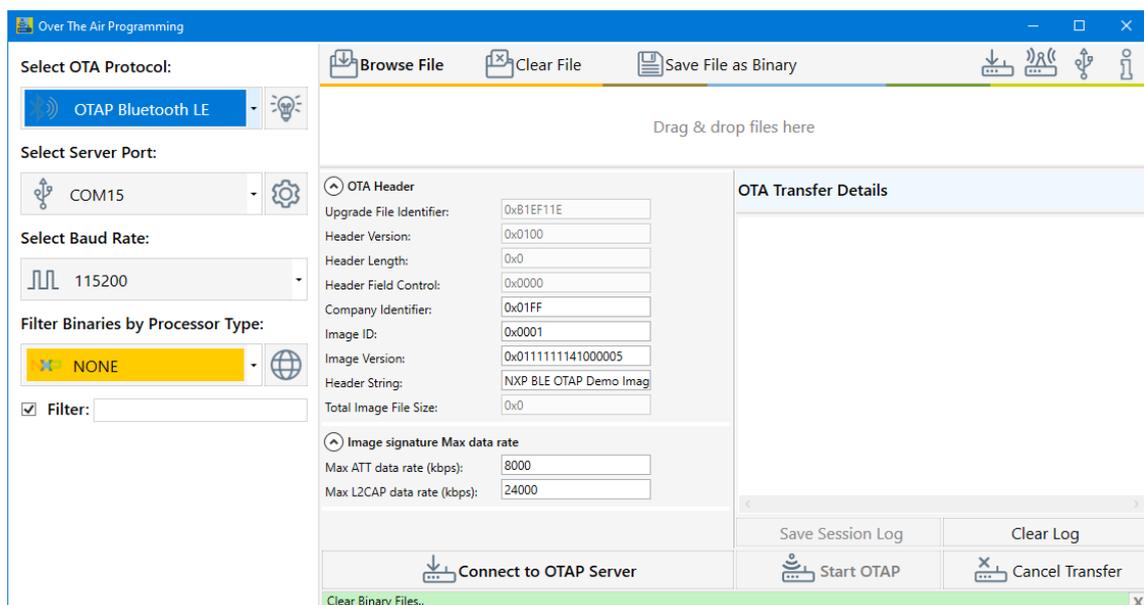
To obtain a .srec (.s19) file, add or uncomment the following post-build command in the same window:

```
arm-none-eabi-objcopy -v -O srec --only-section=.text
--only-section=.data --only-section=.ARM.exidx"
"${BuildArtifactFileName}" "${BuildArtifactFileName}.s19"
```

This window is shown in the figure below.

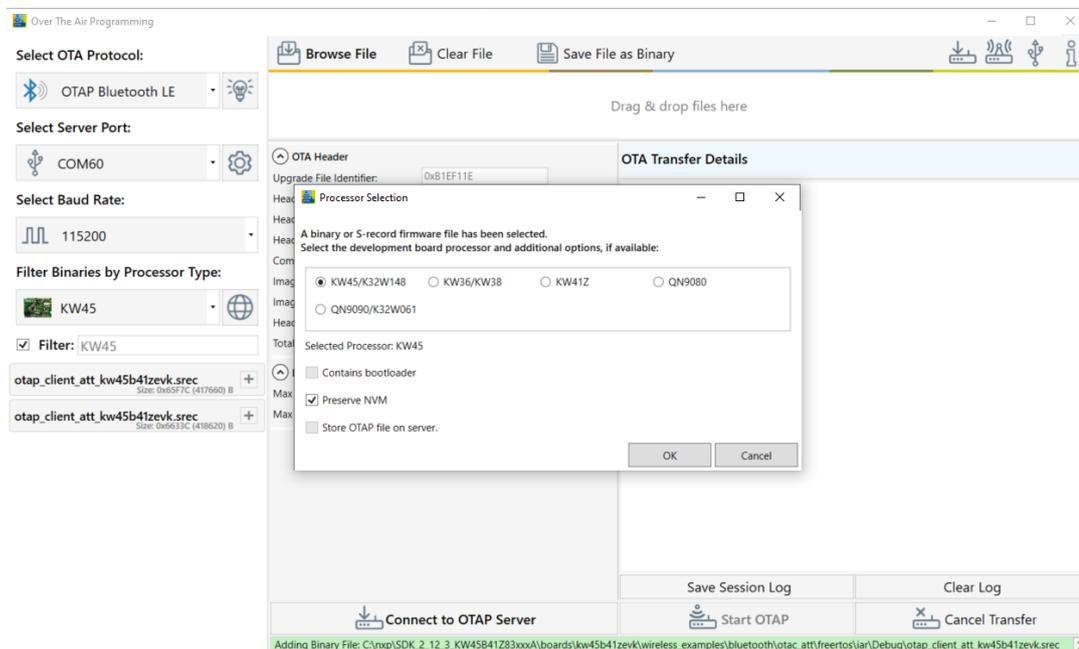


- Start the **Over The Air Programming** application and select **"OTAP Bluetooth LE"** from the **"Select OTA Protocol"** combo box as shown in the figure below.

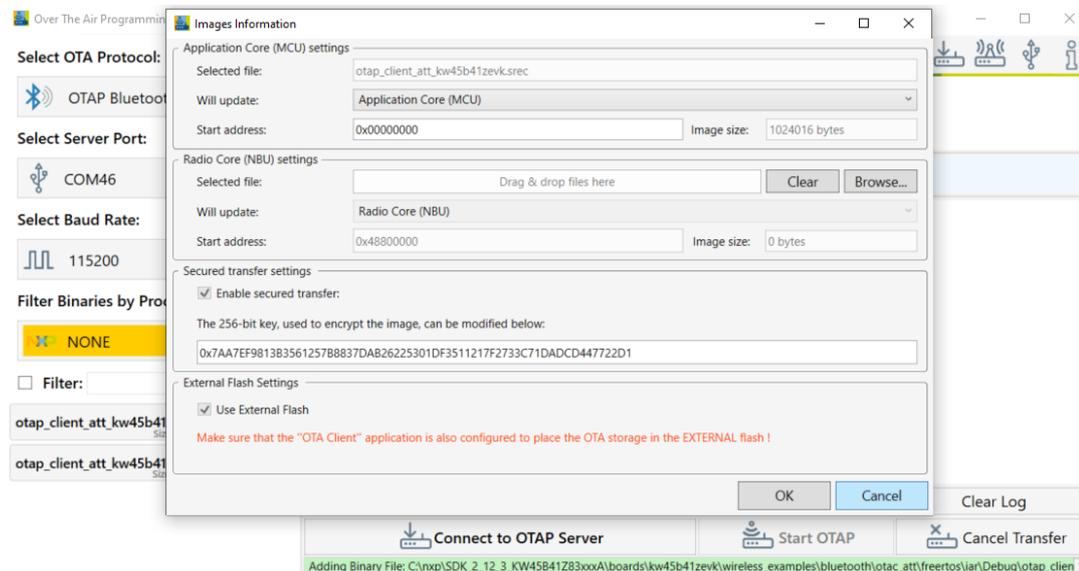


6. Load the image file into the application, then configure the image file header and start the OTAP Server:

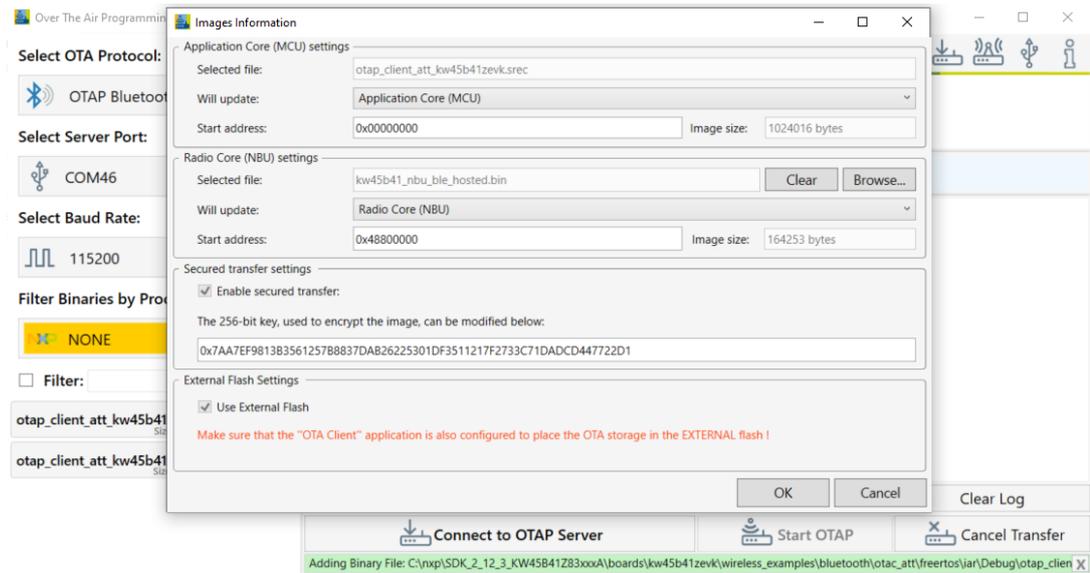
- To select the updated image In the Over the Air Programming tool, select the “**Browse File**” button and then navigate to the .srec or .bin file containing the image to be sent to the OTAP Client. After the .srec or .bin file is chosen, a pop-up window asks to choose the target processor. In this 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).

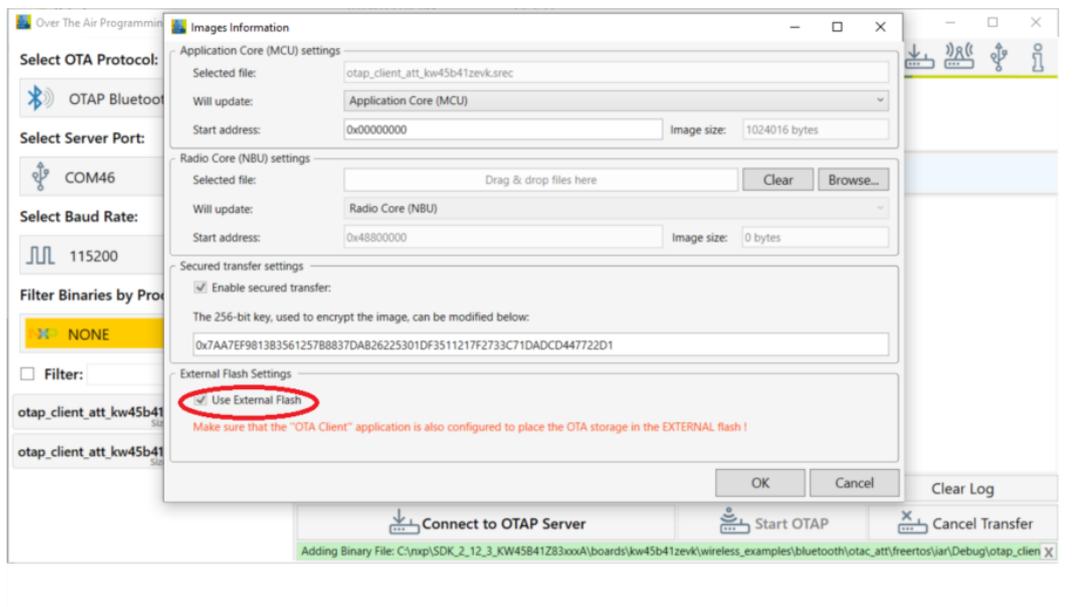


- To update the KW45B/KW32W1 (radio) image, select it by pressing the “**Browse**” button in the M3 group. Then navigate to the .bin file as shown in the figure below.



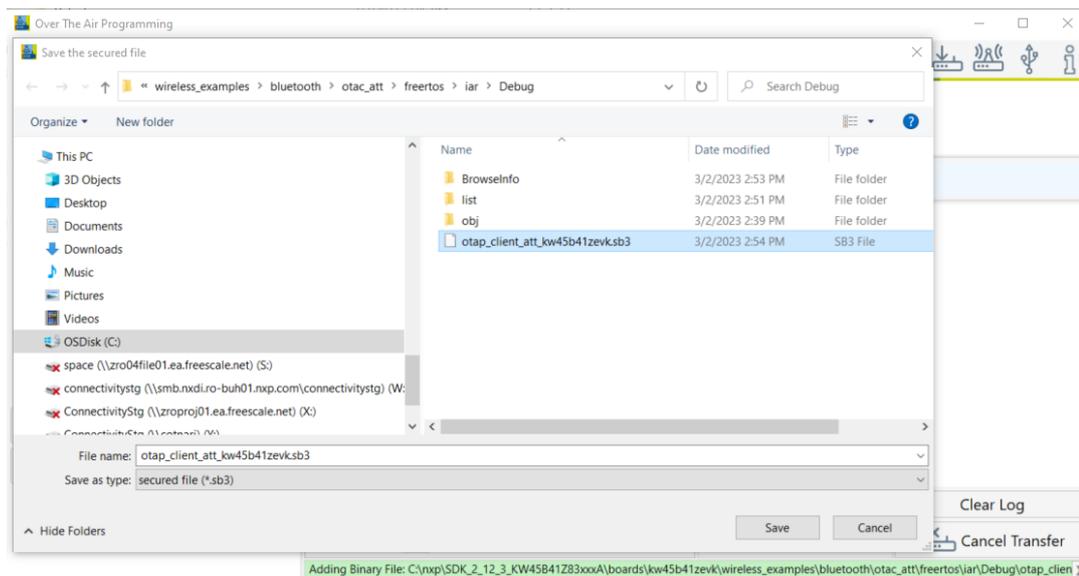
- If the OTA client has configured external memory support, then **“Use External Flash”** checkbox must be checked as in the figure below. If the OTA client has configured internal memory support, the checkbox must be left unchecked.

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.

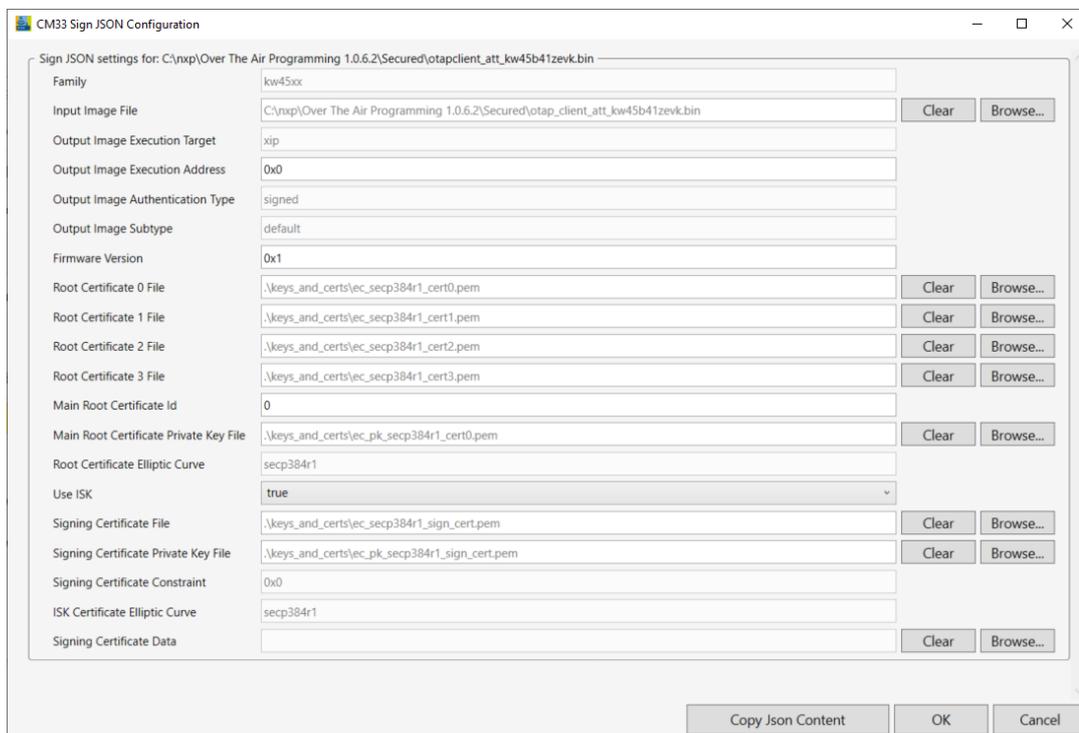
**Note:** The extension of the secured file is \*.sb3. See the figure below.



- 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.

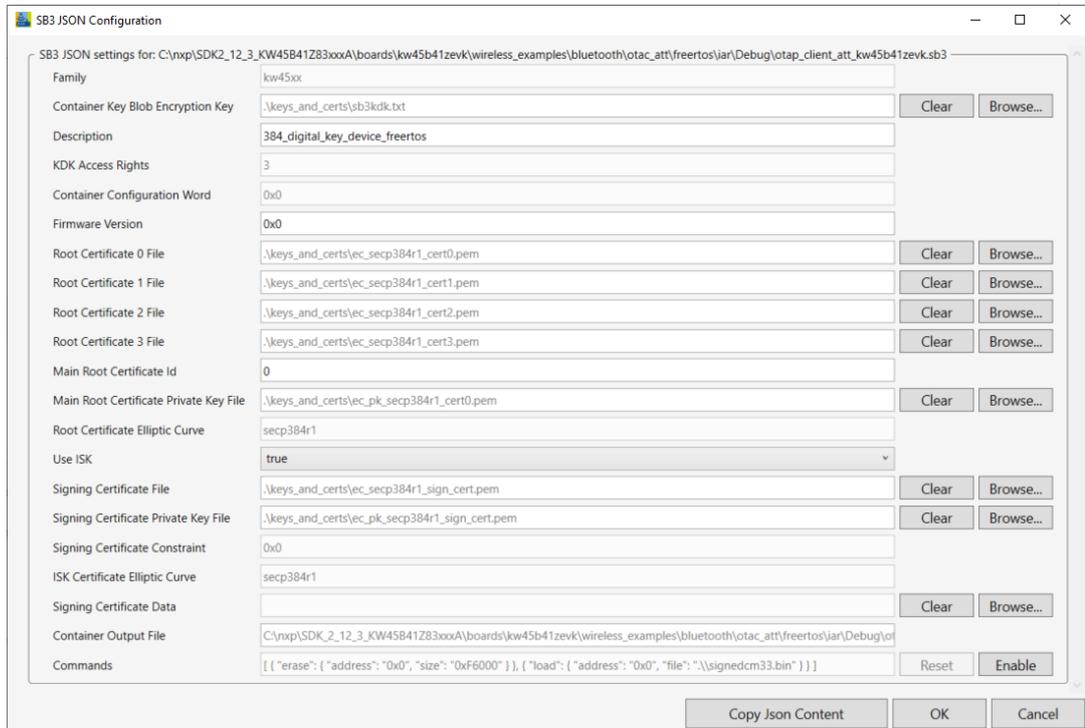
By default, the JSON is configured for the demo applications to run as shown in the figure below.



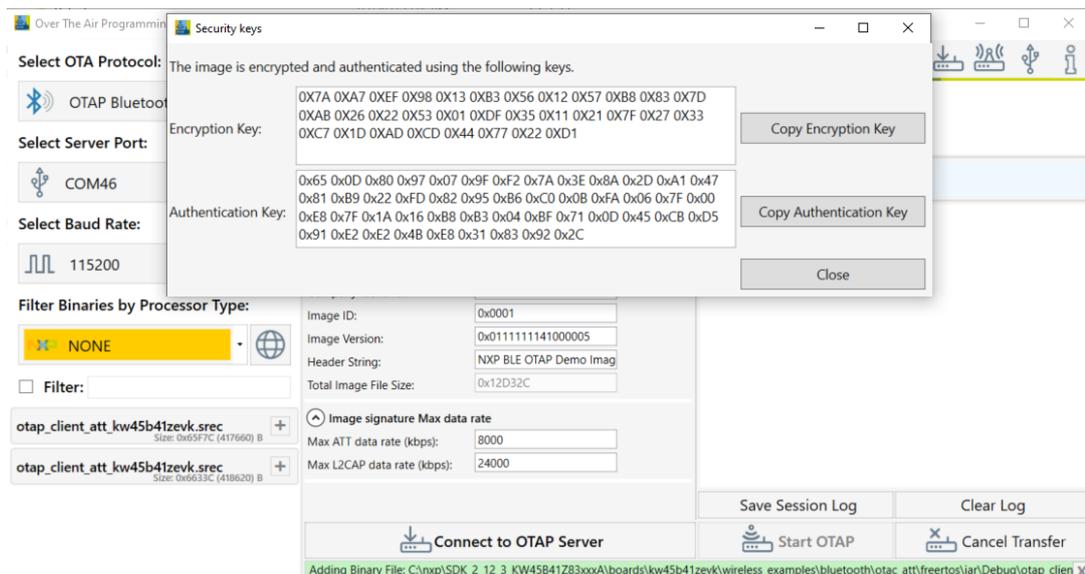
After configuring the JSON file used for signing the MCU file, a new similar window appears. As shown in the Figure, the window is designed for configuring the \*.sb3 con-

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.



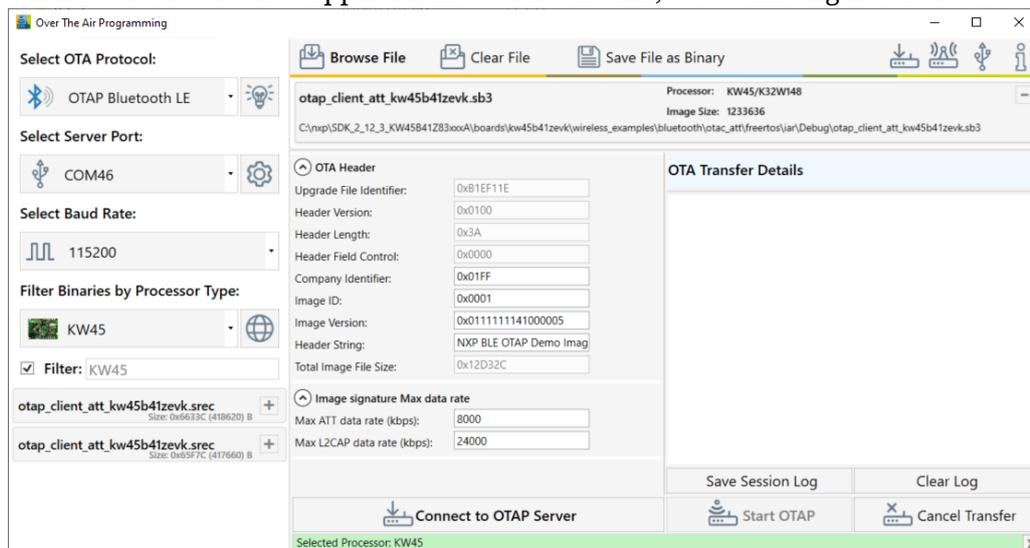
- After the .sb3 file is created, the “**Encryption Key**” and “**Authentication Key**” are presented. For the secured update to be successful, the destination board must have been provisioned with these keys through fuse burning, as described in the accompanying document. Depending on the board type, 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).

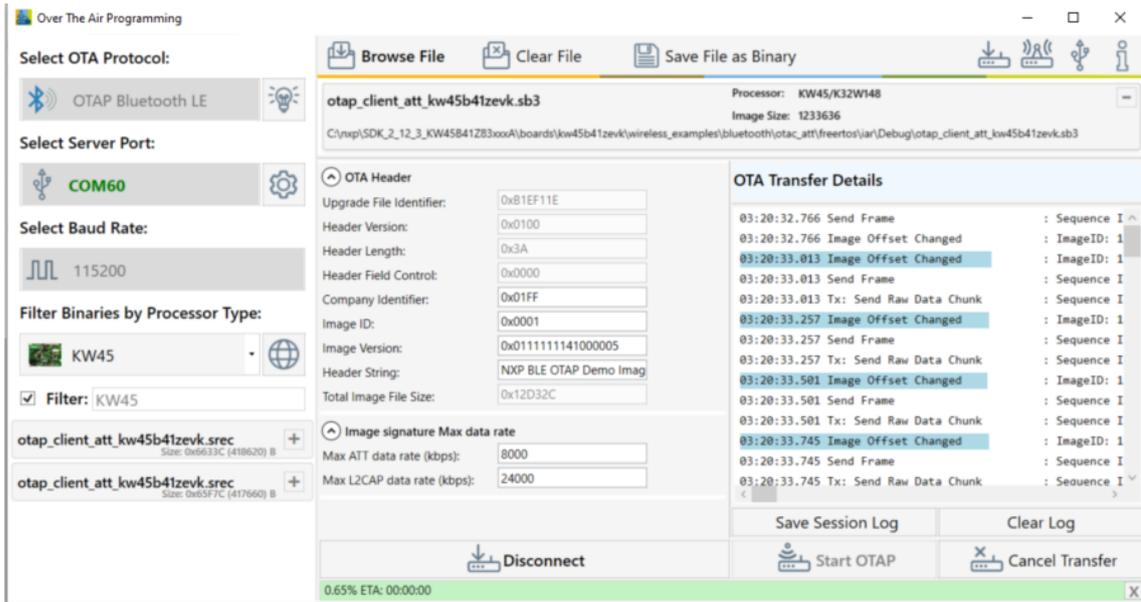
- After the image is loaded, go to the “**Select Server Port**” box, select the correct COM Port for the OTAP Server board. Also select the default baud rate of 115200 and press the “**Connect to OTAP Server**” button. A successful connection is displayed in the Message Log.
  - If the image is loaded 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.



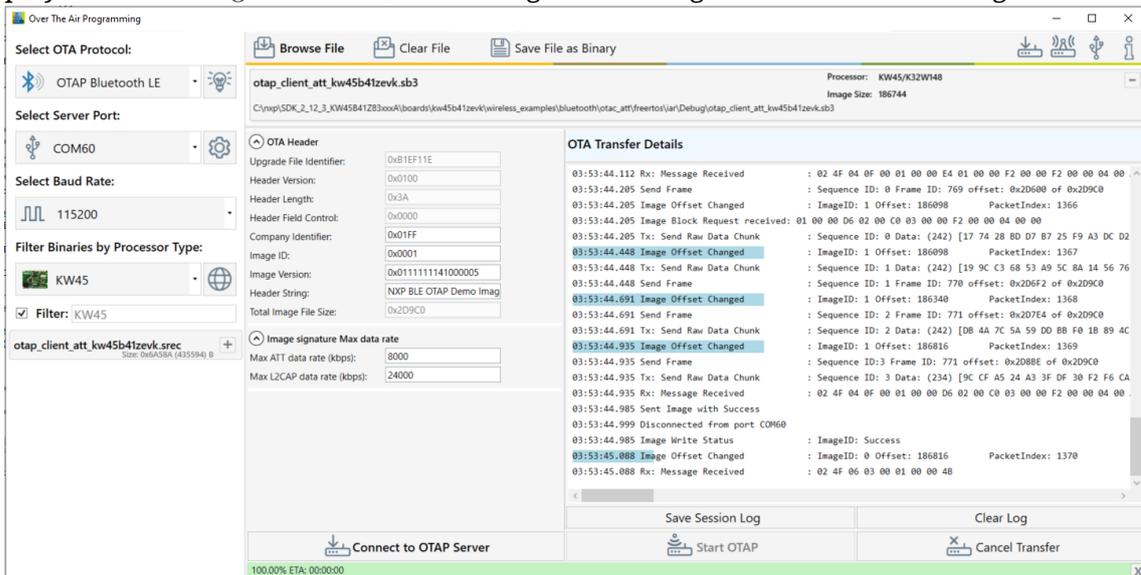
- Before starting the image transfer process, the data rate must be configured for each transfer method (ATT or L2CAP CoC). The image chunks of a block are sent over the serial interface and over-the-air without waiting for confirmation. Data rate can significantly slow down if configuration is not done 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 **ADVS** 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 below:

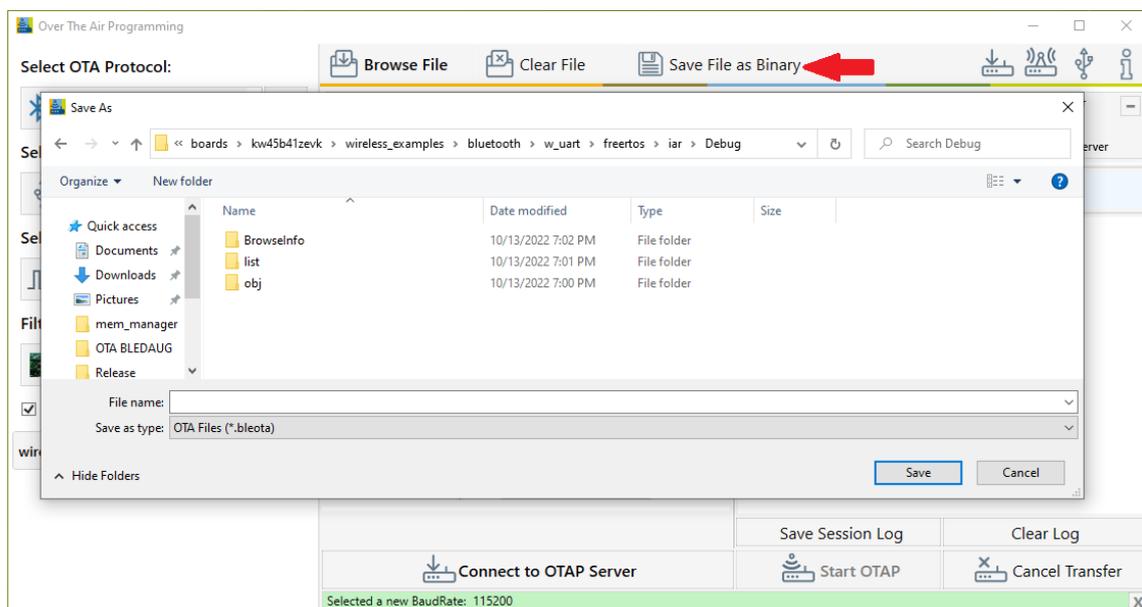
1. Set the `gUseInternalStorageLink_d` define to 0 in the project settings. For GitHub ARMGCC, replace `CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support.no_flash_selected=y` with `CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support.external_flash=y` in the application `prj.conf` file.
2. In `app_preinclude.h`, set the `gAppOtaExternalStorage_c` define to 1.

**Usage with IoT Toolbox** This is the list of requirements.

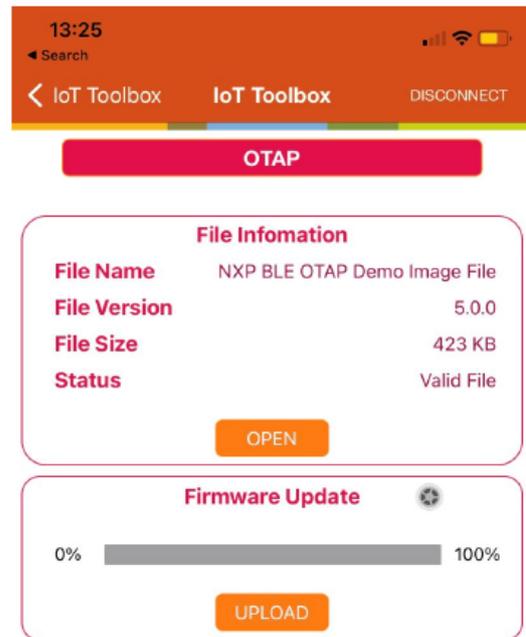
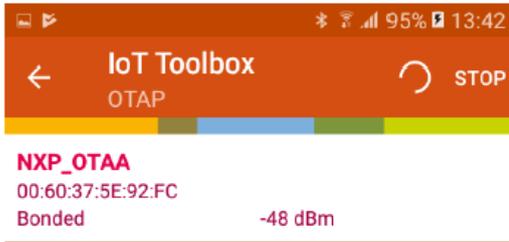
- **Mobile device** running **Android** platform or **iOS** with hardware and software supporting **Bluetooth 4.0 and later**.
- **Kinetis Bluetooth LE Toolbox** application – download from the specific application store for your device.

To run the application, perform the following steps:

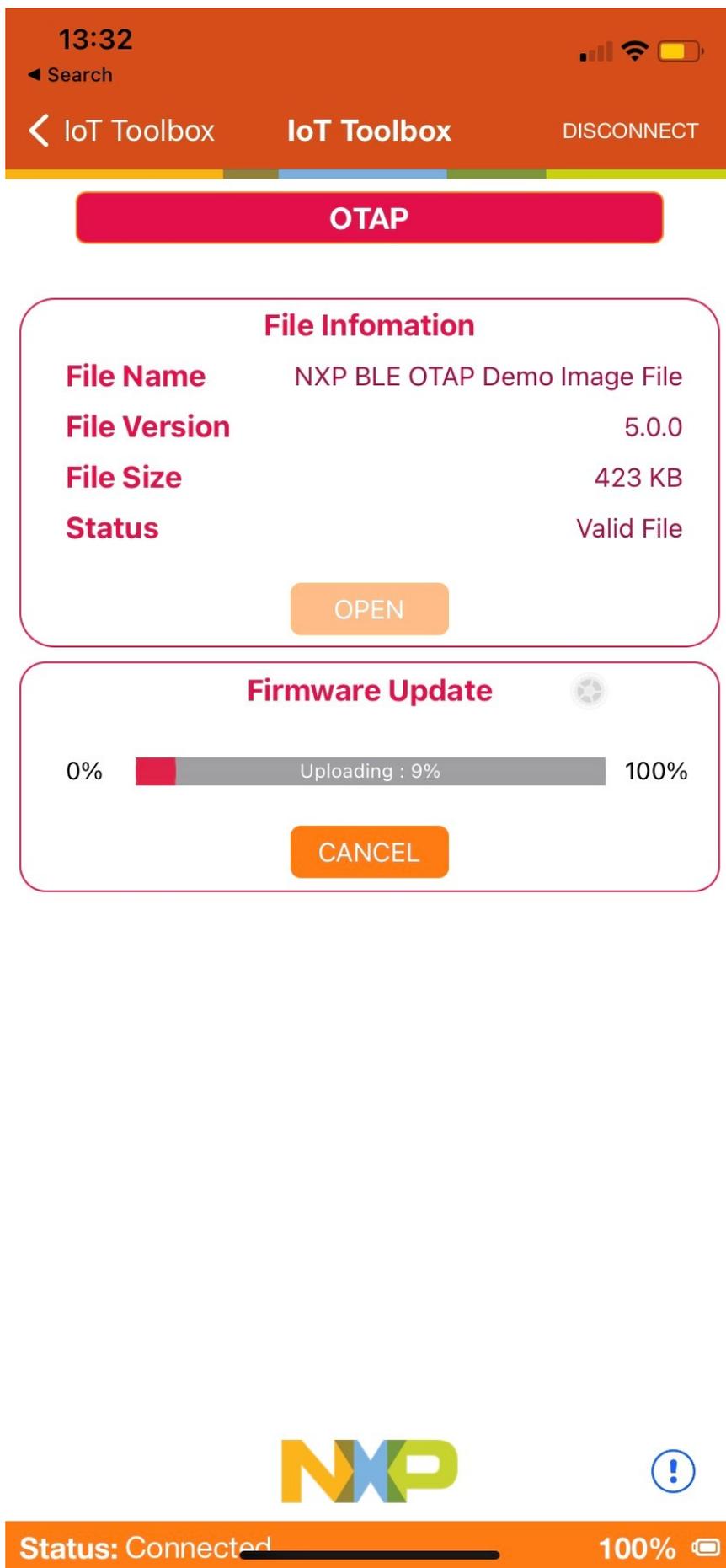
1. Flash the OTAP Client ATT to either the KW45B41Z-EVK or the FRDM-MCXW71 platform. The Kinetis Bluetooth LE Toolbox only supports the ATT OTAP Client.
2. In order to send over the air in .bleota format, create the application. In order to load the image file into the Over the Air Programming application 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 **ADVS** 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.



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

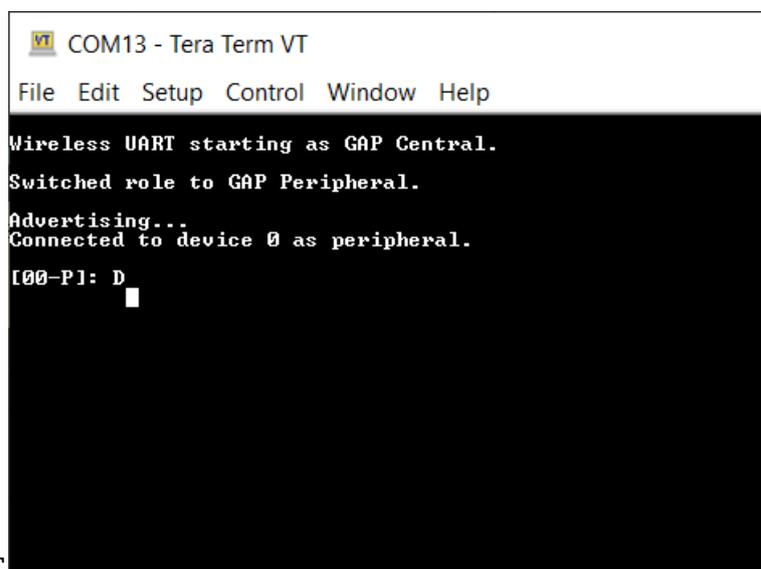
**User interface** After flashing the board, the device is in idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure of GAP General Discoverable Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. 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.

A screenshot of a Tera Term terminal window titled "COM13 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The terminal output shows the following text:

```
Wireless UART starting as GAP Central.
Switched role to GAP Peripheral.
Advertising...
Connected to device 0 as peripheral.
[00-P]: D
█
```

**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 rssiindicator C4603770BCC5--> Reading RSSI on advertising channel:
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
RSSI: -27 dBm
RSSI: -29 dBm
```

In the below example, the RSSI is 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 rssiindicator
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:
rssimonitor test
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 **thruput** keyword:

- **thruput setparam**: configures connection interval, packet count and payload size.
- **thruput 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>).
- **thruput 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>).
- **thruput 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>thruput 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 -arbmasks 00000000ffff
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.

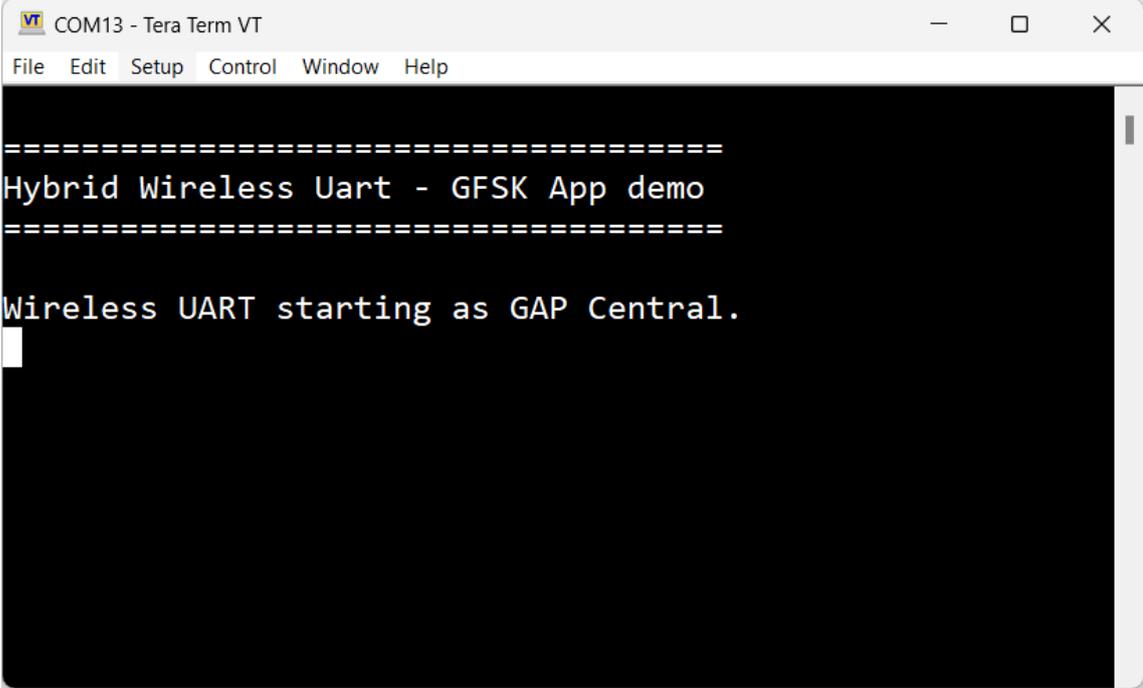
**Parent topic:**Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

**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.

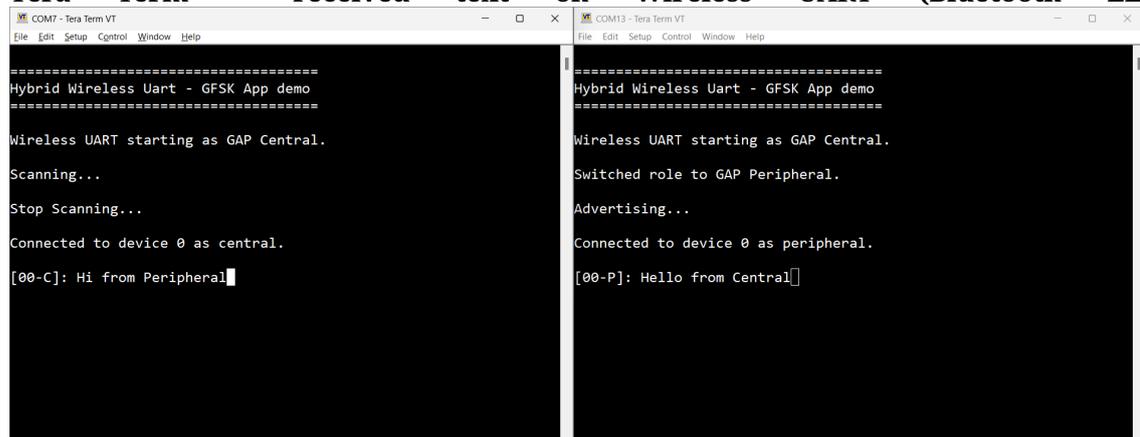
**Tera Term – Hybrid Wireless UART (Bluetooth LE) - Generic FSK start screen**



```
COM13 - Tera Term VT
File Edit Setup Control Window Help
=====
Hybrid Wireless Uart - GFSK App demo
=====
Wireless UART starting as GAP Central.
```

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.

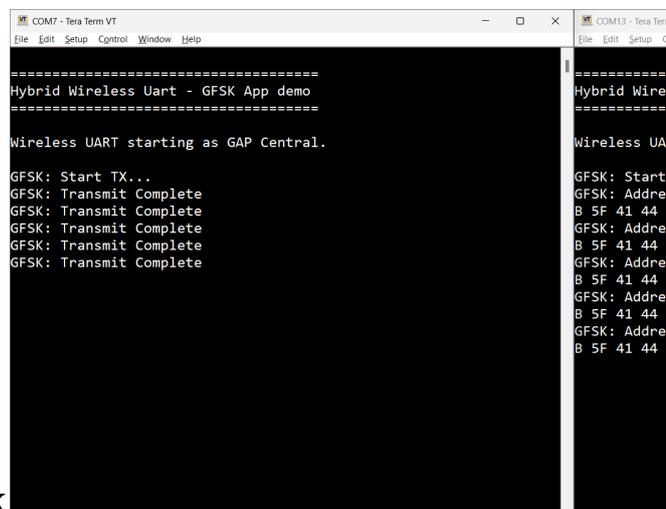
**Tera Term – received text on Wireless UART (Bluetooth LE)**



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.

- 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.

The image shows two side-by-side terminal windows from Tera Term. The left window, titled 'COM7 - Tera Term VT', displays a continuous list of 'GFSK: Transmit Complete' messages. The right window, titled 'COM13 - Tera Term VT', displays a sequence of 'GFSK: Address: 09 09 09 09 09 09' messages, each followed by a hexadecimal address starting with 'B 5F 41 44' and ending with a letter from 'BB' to 'CD'. At the bottom of the left window, the text 'GFSK: Stop TX' is visible, and at the bottom of the right window, 'GFSK: Stop RX' is visible.

**Tera Term – stop Generic FSK activity**

Parent topic: Usage

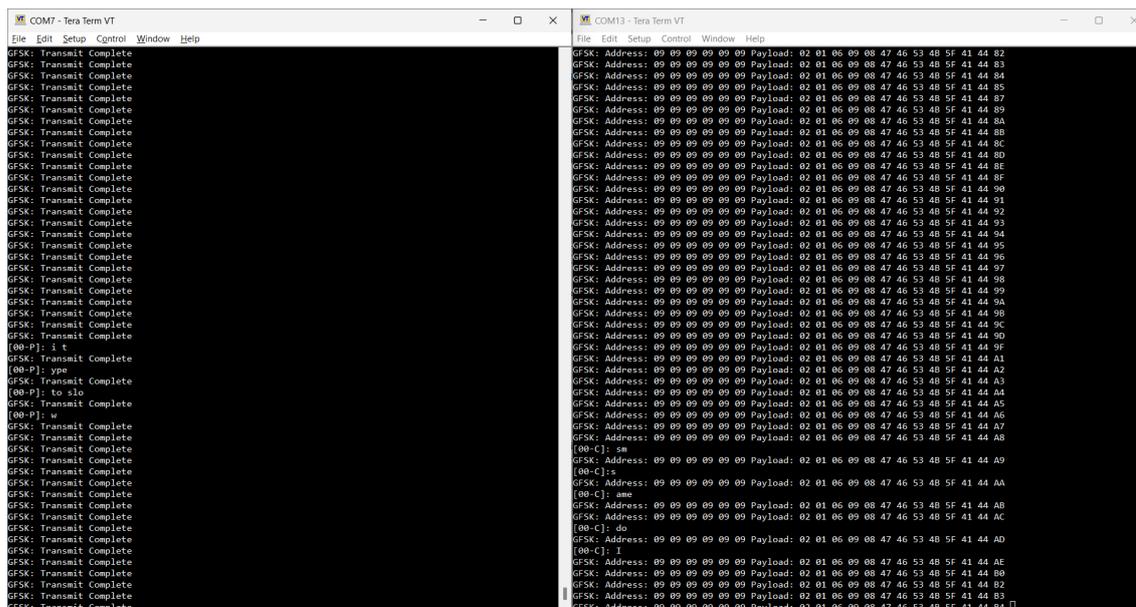
**Case 3 (Dual-mode)** The application is ready to combine both scenarios. It can either:

- 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:

- Establish a Bluetooth LE connection as described in Case 1 (Bluetooth LE).
- Start Generic FSK activity as described in Case 2 (Generic FSK), independent of the Bluetooth LE roles chosen in Case 1.
- 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
gGen-FSK_Netw	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.
gGen-FSK_H0Val	H0 value is used in the header.
gGen-FSK_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.
gGenF-skApp_TxI	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_gfsk.c	Controller common module. Handles initialization of Generic FSK.
hybrid_gfsk.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

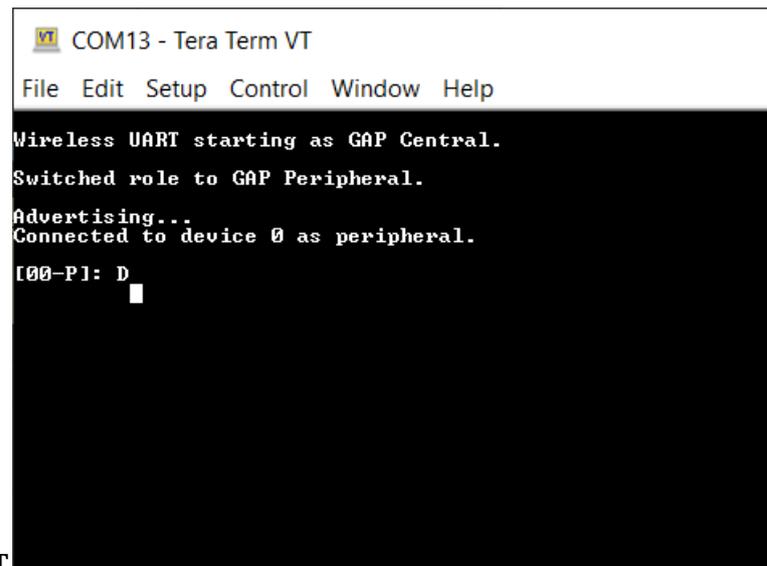
Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

**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.



```
COM13 - Tera Term VT
File Edit Setup Control Window Help
Wireless UART starting as GAP Central.
Switched role to GAP Peripheral.
Advertising...
Connected to device 0 as peripheral.
[00-P]: D
```

**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

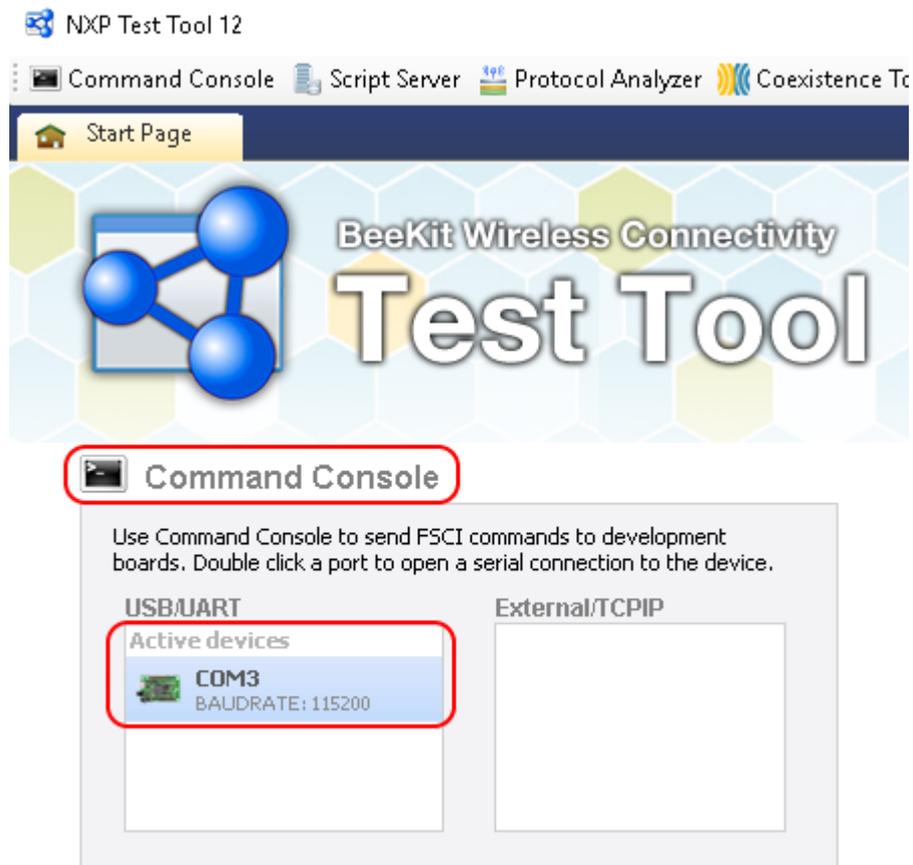
**Supported platforms** The following platforms support the FSCI Bridge application:

- MCX-W72-EVK
- FRDM-MCXW72

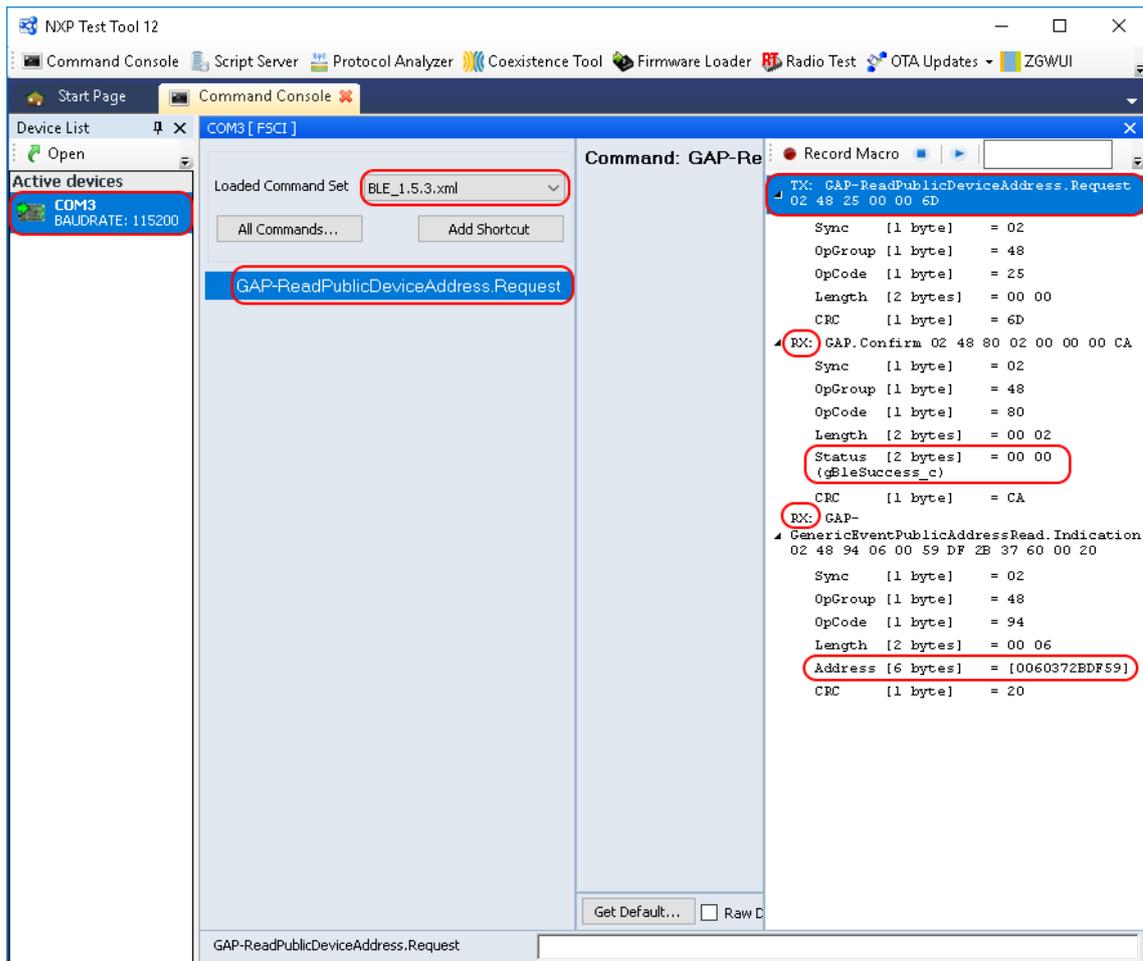
**Parent topic:**FSCI Bridge

**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\_BLEHSFSCIPIRM)
- *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.

- 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.

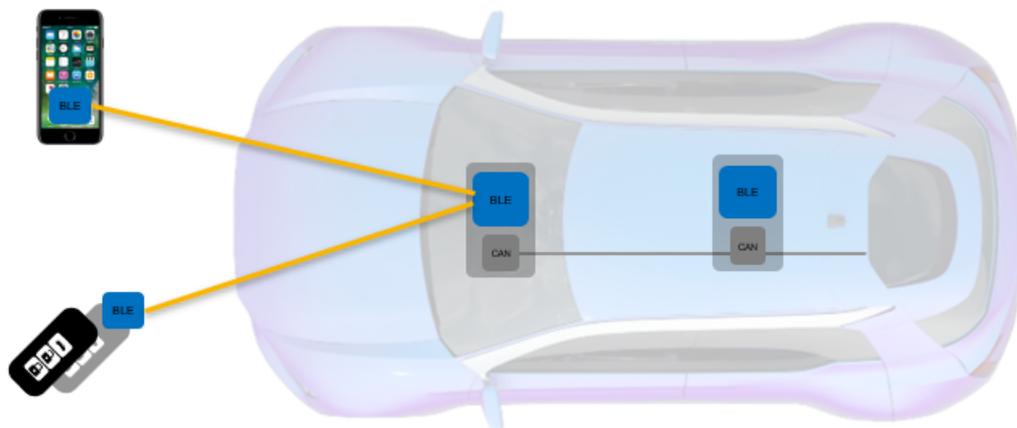
Simultaneously it can also act as a Bluetooth Low Energy central, scanning for non-CCC key fobs. There can be multiple anchors inside a car. The anchors must act as a single device and share information between them (addresses, bonding data, application specific keys, and such data) such that a smartphone 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](#)

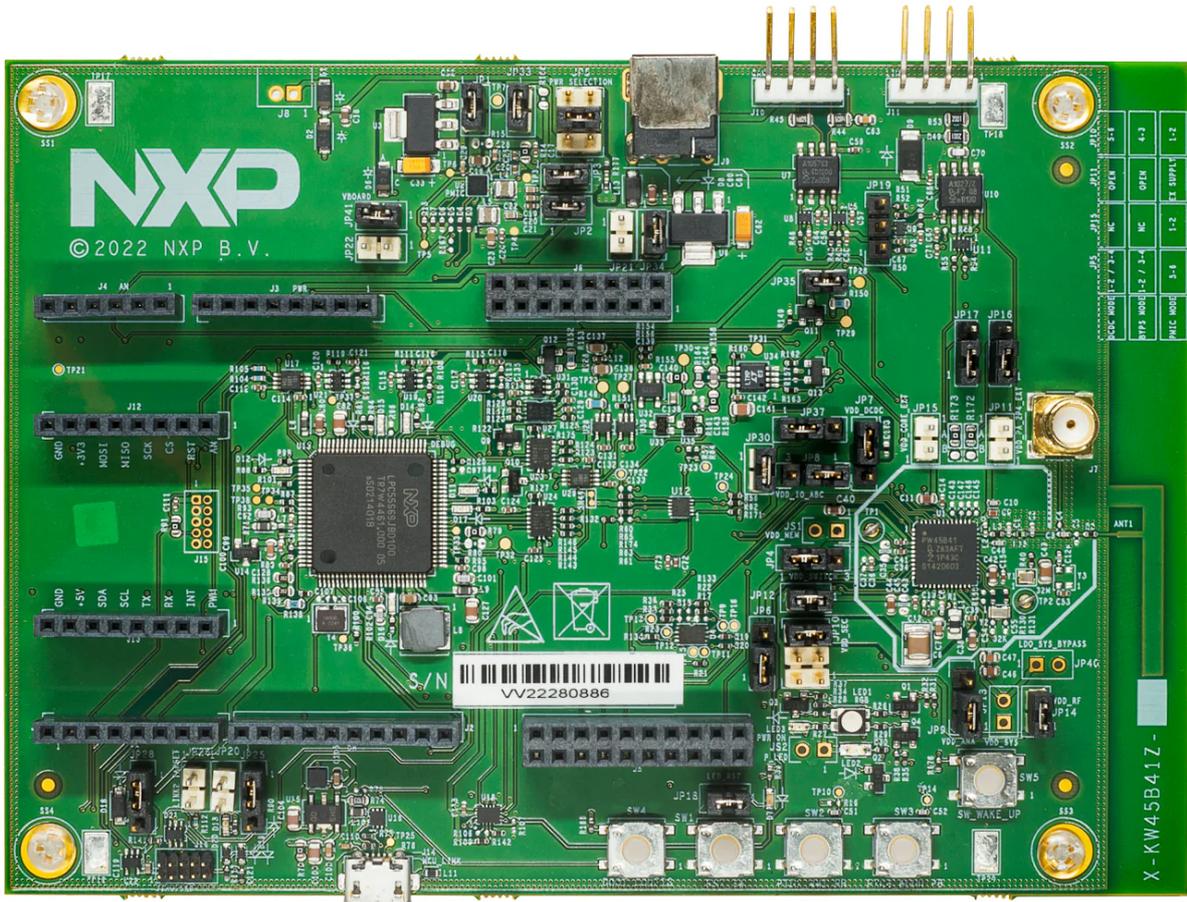
**CCC Digital Key hardware platforms** The CCC Digital Key R3 demo applications support the following platforms that have Bluetooth Low Energy transceiver capabilities:

- FRDM-KW38 (see Figure 1)
- KW45B41Z-EVK (see Figure 2)
- KW47-EVK (see Figure 3 )
- KW47-LOC (see Figure 4 )

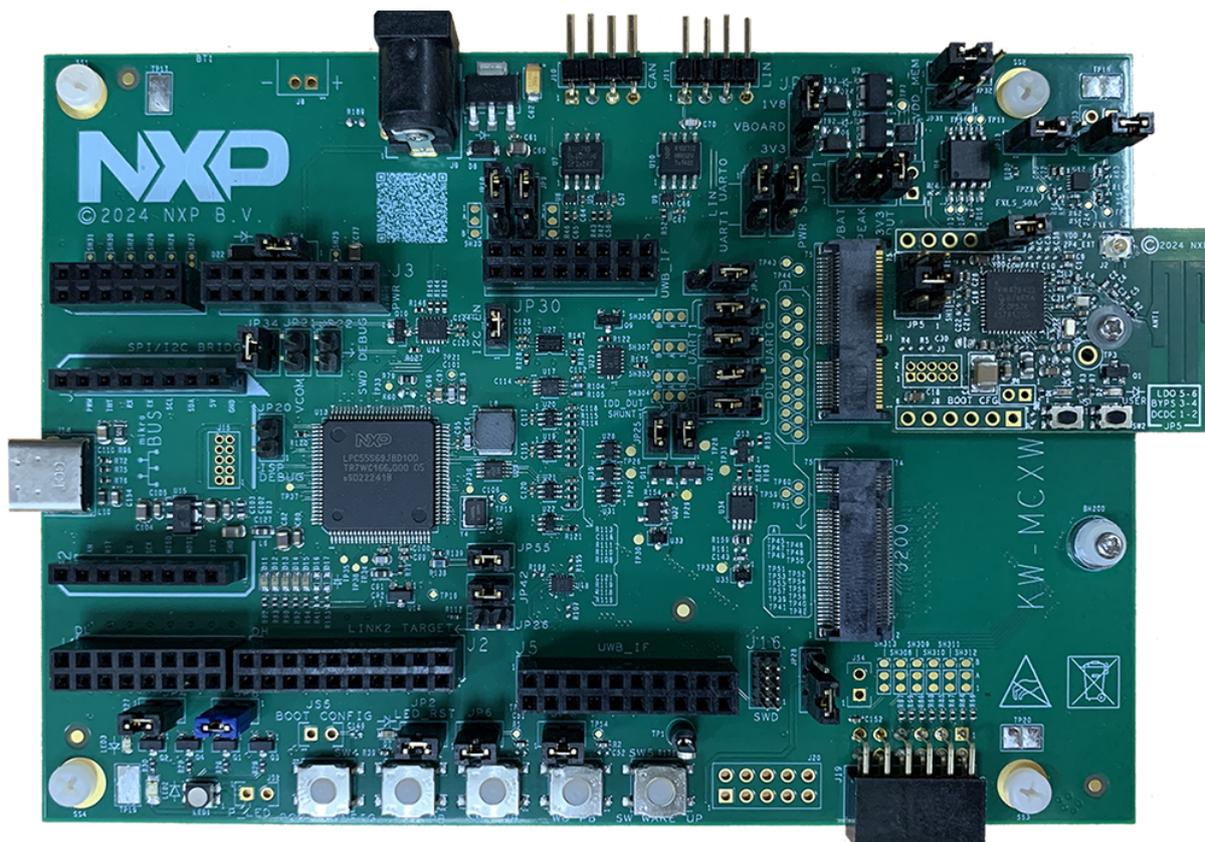
**FRDM-KW38 board**



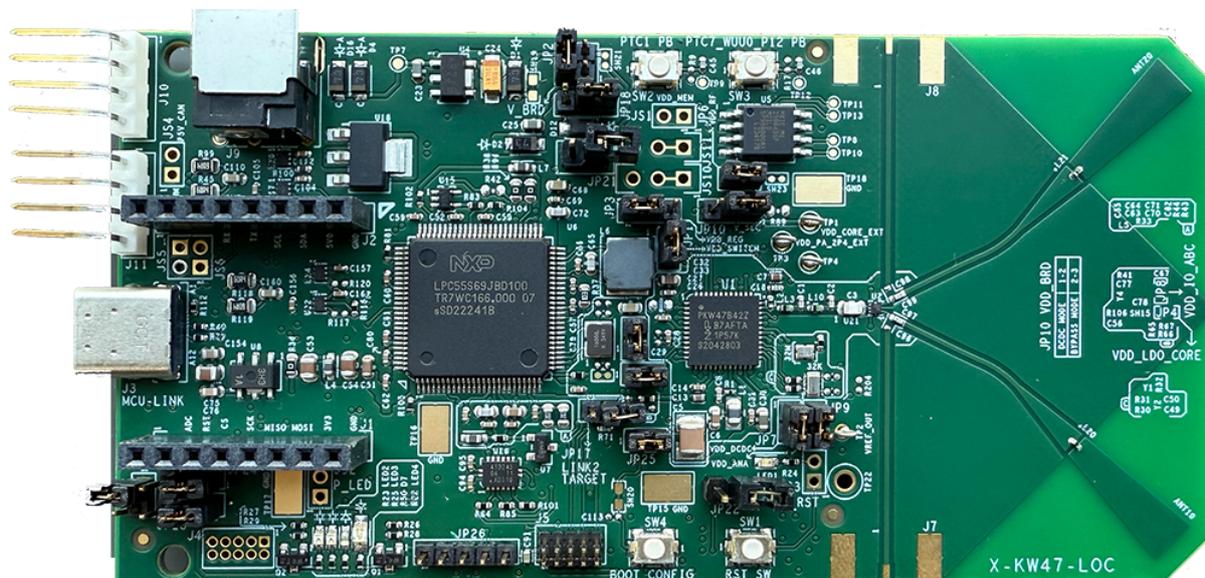
KW45B41Z-EVK board



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.

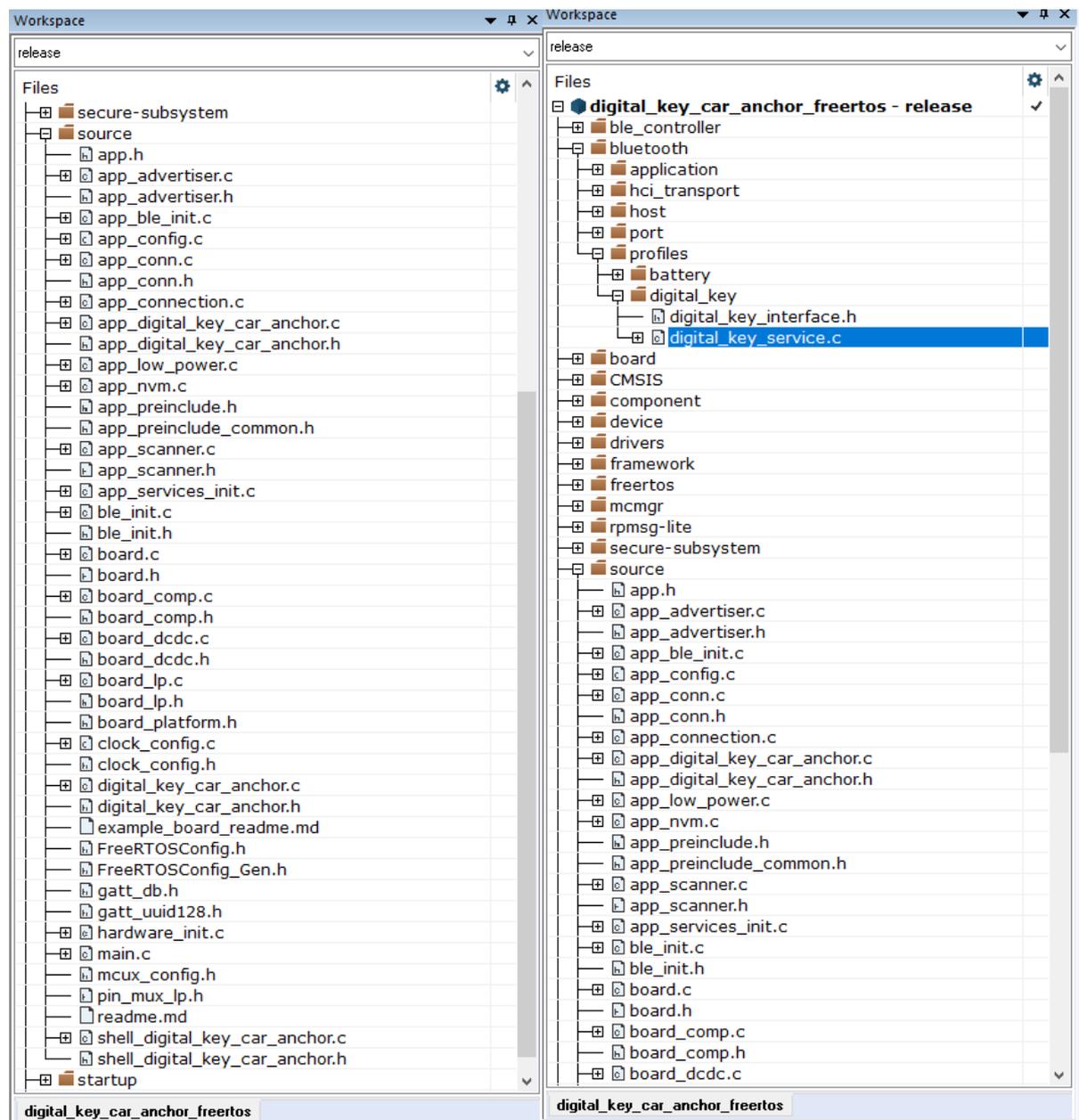
To open, build, and run any example application, see the *Bluetooth Low Energy Quick Start Guide* document of the corresponding board. The CCC Digital Key R3 examples are built and run in the same fashion as Bluetooth Low Energy examples.

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).

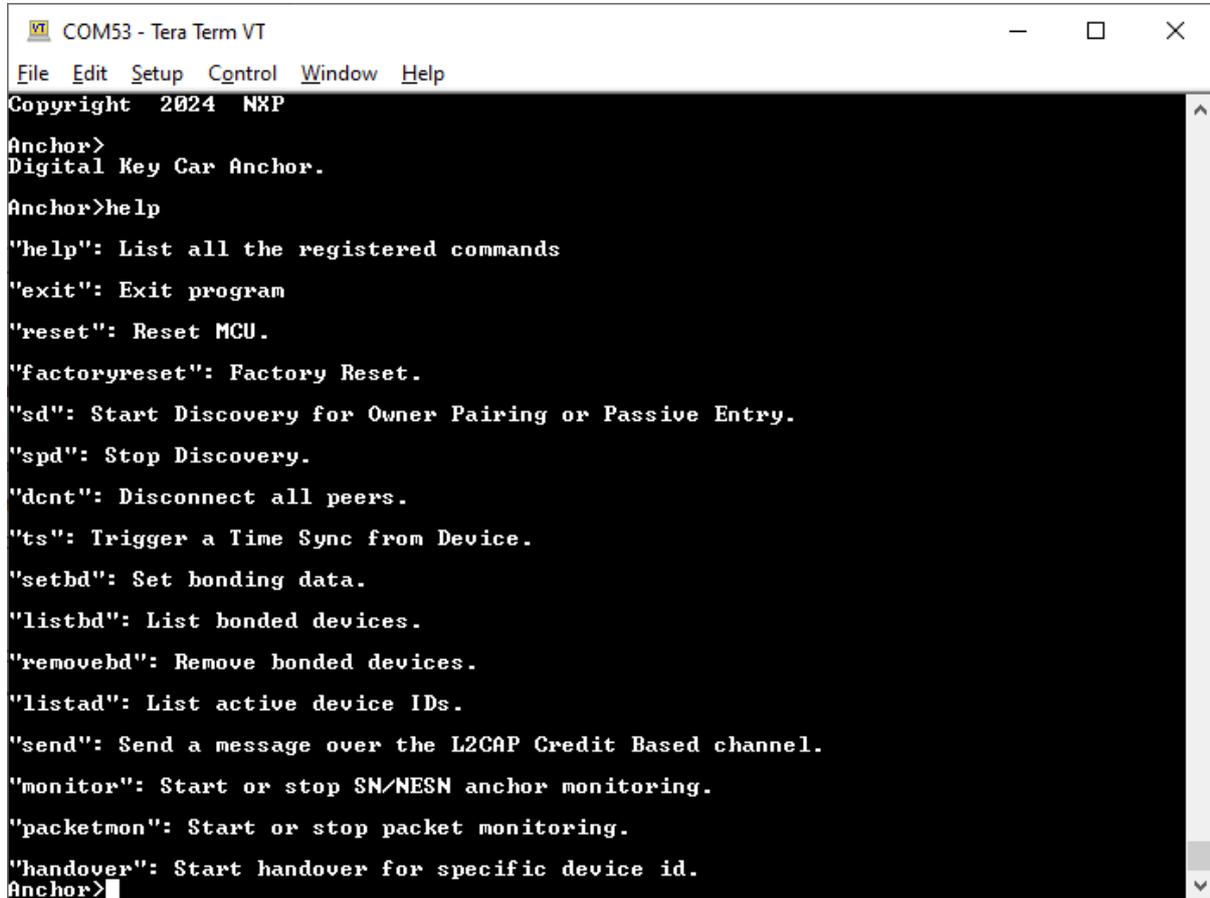
**Running CCC Digital Key scenarios using the Shell Interface** The following CCC Digital Key R3 examples are provisioned by default with a shell command line interface accessible via a Terminal application such as Tera Term:

- 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



```

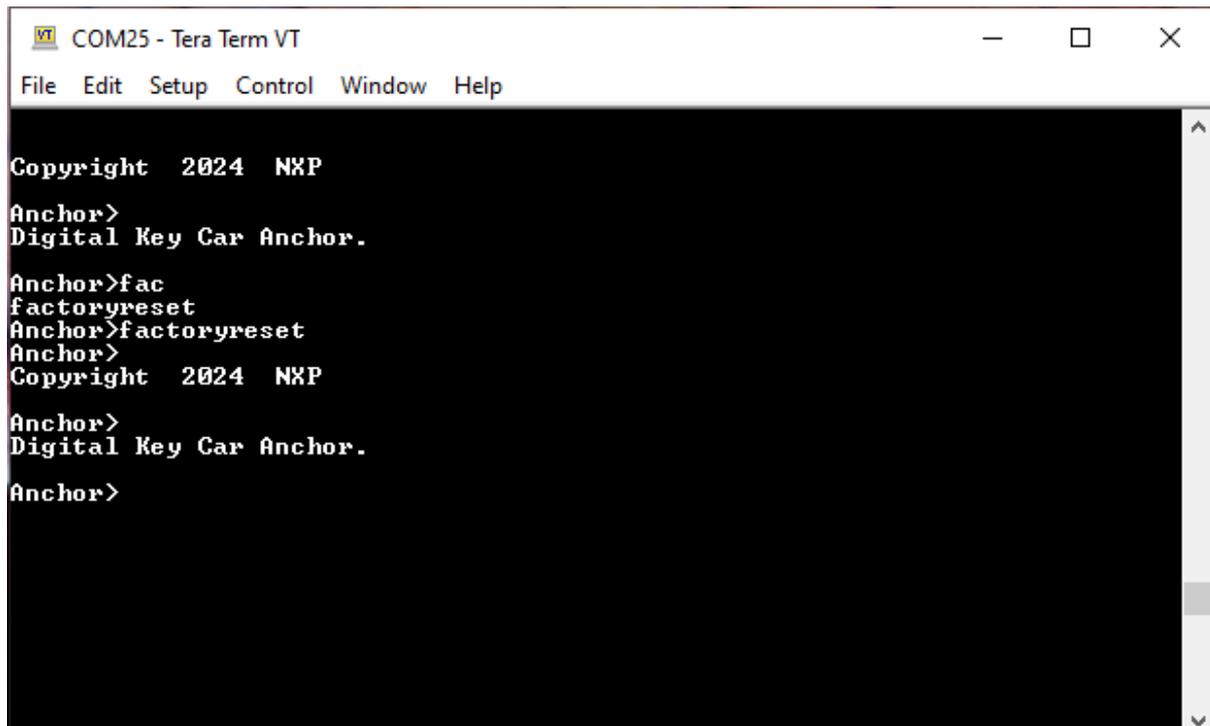
COM53 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>help
"help": List all the registered commands
"exit": Exit program
"reset": Reset MCU.
"factoryreset": Factory Reset.
"sd": Start Discovery for Owner Pairing or Passive Entry.
"spd": Stop Discovery.
"dcnt": Disconnect all peers.
"ts": Trigger a Time Sync from Device.
"setbd": Set bonding data.
"listbd": List bonded devices.
"removebd": Remove bonded devices.
"listad": List active device IDs.
"send": Send a message over the L2CAP Credit Based channel.
"monitor": Start or stop SN/NESN anchor monitoring.
"packetmon": Start or stop packet monitoring.
"handover": Start handover for specific device id.
Anchor>

```

**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.

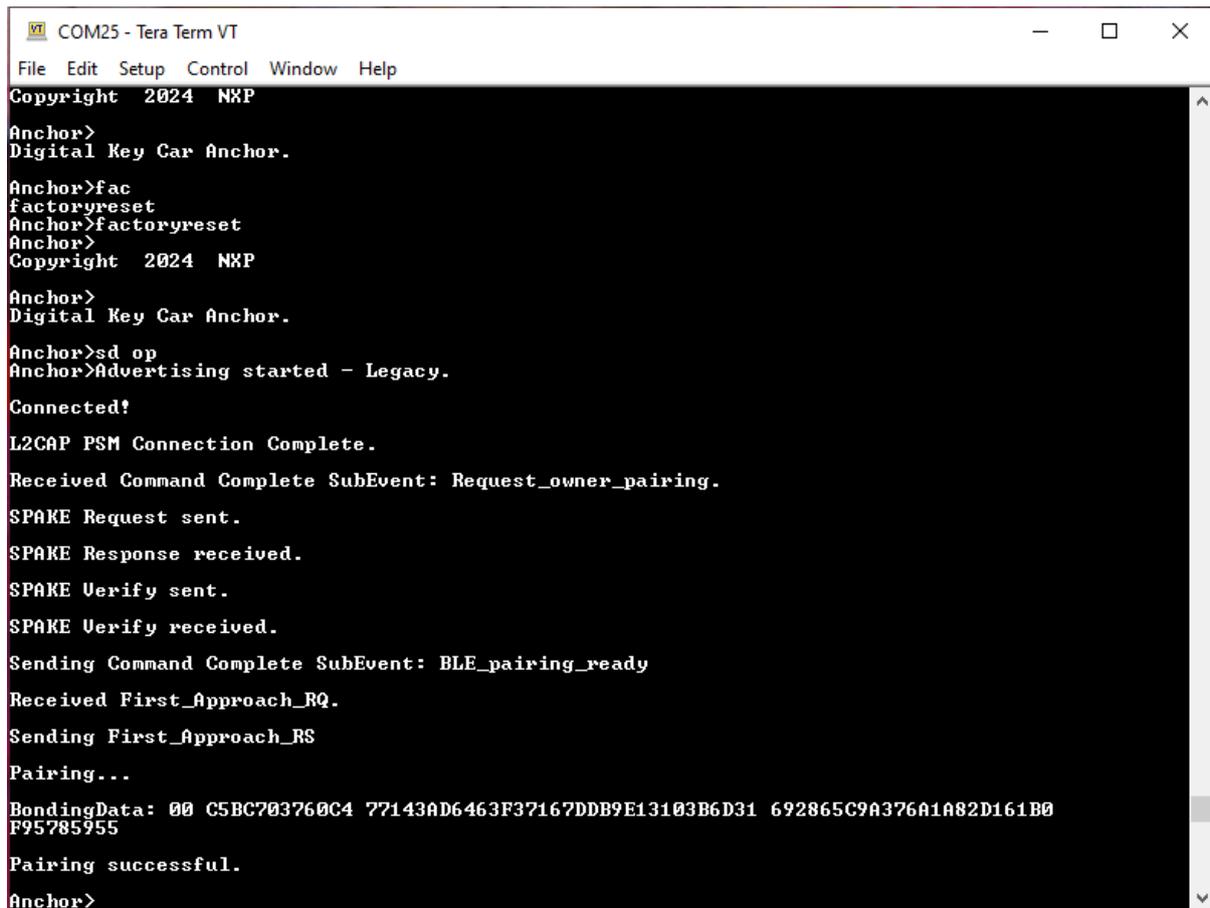
### Factory reset on Car Anchor



```
COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>fac
factoryreset
Anchor>factoryreset
Anchor>
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
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



```
COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>fac
factoryreset
Anchor>factoryreset
Anchor>
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>sd op
Anchor>Advertising started - Legacy.
Connected!
L2CAP PSM Connection Complete.
Received Command Complete SubEvent: Request_owner_pairing.
SPAKE Request sent.
SPAKE Response received.
SPAKE Verify sent.
SPAKE Verify received.
Sending Command Complete SubEvent: BLE_pairing_ready
Received First_Approach_RQ.
Sending First_Approach_RS
Pairing...
BondingData: 00 C5BC703760C4 77143AD6463F37167DDB9E13103B6D31 692865C9A376A1A82D161B0
F95785955
Pairing successful.
Anchor>
```

Figure 3 shows the owner pairing on the Device.

### Owner Pairing on Device

```

COM10 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Device>
Digital Key Device.
Device>
Copyright 2024 NXP
Device>
Digital Key Device.
Device>sd
Device>Scanning...
Legacy ADU: 523F7C6537B8
Scan stopped.
Connecting...
Connected!
L2CAP PSM Connection Complete.
Sending Command Complete SubEvent: Request_owner_pairing
SPAKE Request received.
SPAKE Response sent.
SPAKE Verify received.
SPAKE Verify sent.
Received Command Complete SubEvent: BLE_pairing_ready
Sending First_Approach_RQ
OOB Data:
Address, Confirm, Random: D7B20F9A1479 ACDD8A8F693259FD8367DD10F4C648A8 098948104686F8EBD7266A898
17E489
Received First_Approach_RS.
Pairing...
BondingData: 00 C5BC703760C4 77143AD6463F37167DDB9E13103B6D31 0A2DF465E3BD7B491EB4C09595134673
Pairing successful.
Device>
Time Sync sent with UWB Device Time:000000000A65E72C
Device>

```

Parent topic:[Running CCC Digital Key scenarios using the Shell Interface](#)

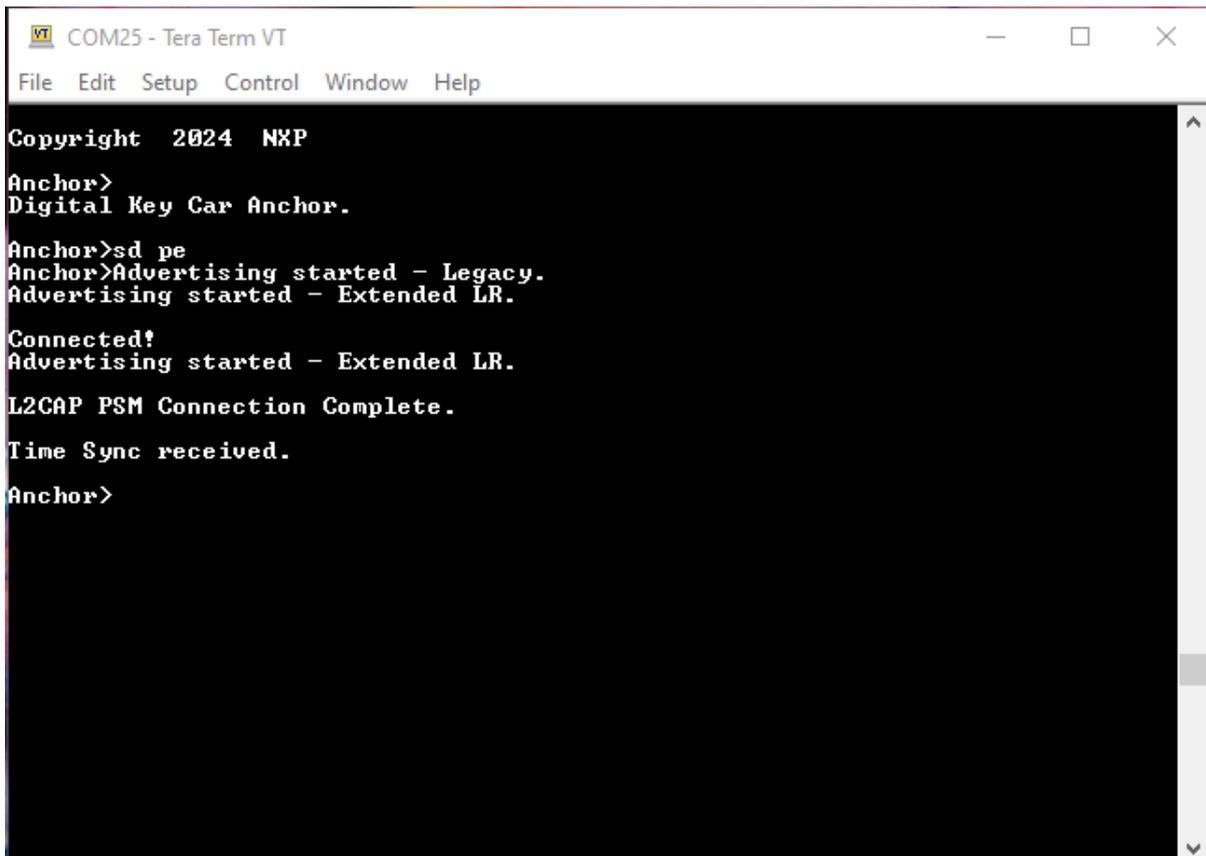
**Passive Entry Scenario** The Passive Entry scenario runs after Owner Pairing between a Car Anchor and a Device completes successfully.

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.

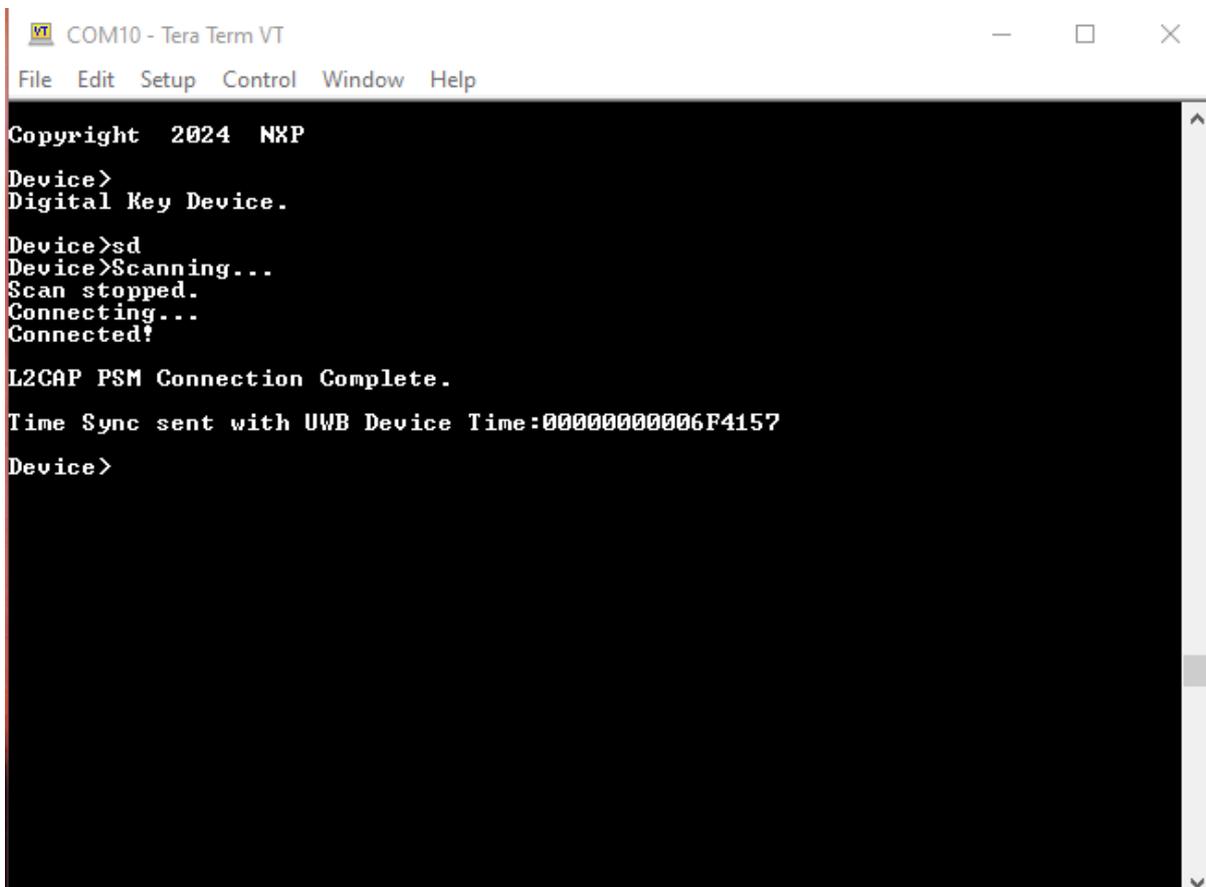
**Note:** The CCC specification requires using coding scheme S=2 for Coded PHY advertising, equivalent to a data rate of 500 Kbits per sec. To experiment with other coding schemes, change the advertising parameters in the `app_config.c` file for the `digital_key_car_anchor` project.

### Passive Entry on Car Anchor



```
COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>sd pe
Anchor>Advertising started - Legacy.
Advertising started - Extended LR.
Connected!
Advertising started - Extended LR.
L2CAP PSM Connection Complete.
Time Sync received.
Anchor>
```

### Passive Entry on Device



```
COM10 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Device>
Digital Key Device.
Device>sd
Device>Scanning...
Scan stopped.
Connecting...
Connected!
L2CAP PSM Connection Complete.
Time Sync sent with UWB Device Time:00000000006F4157
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.

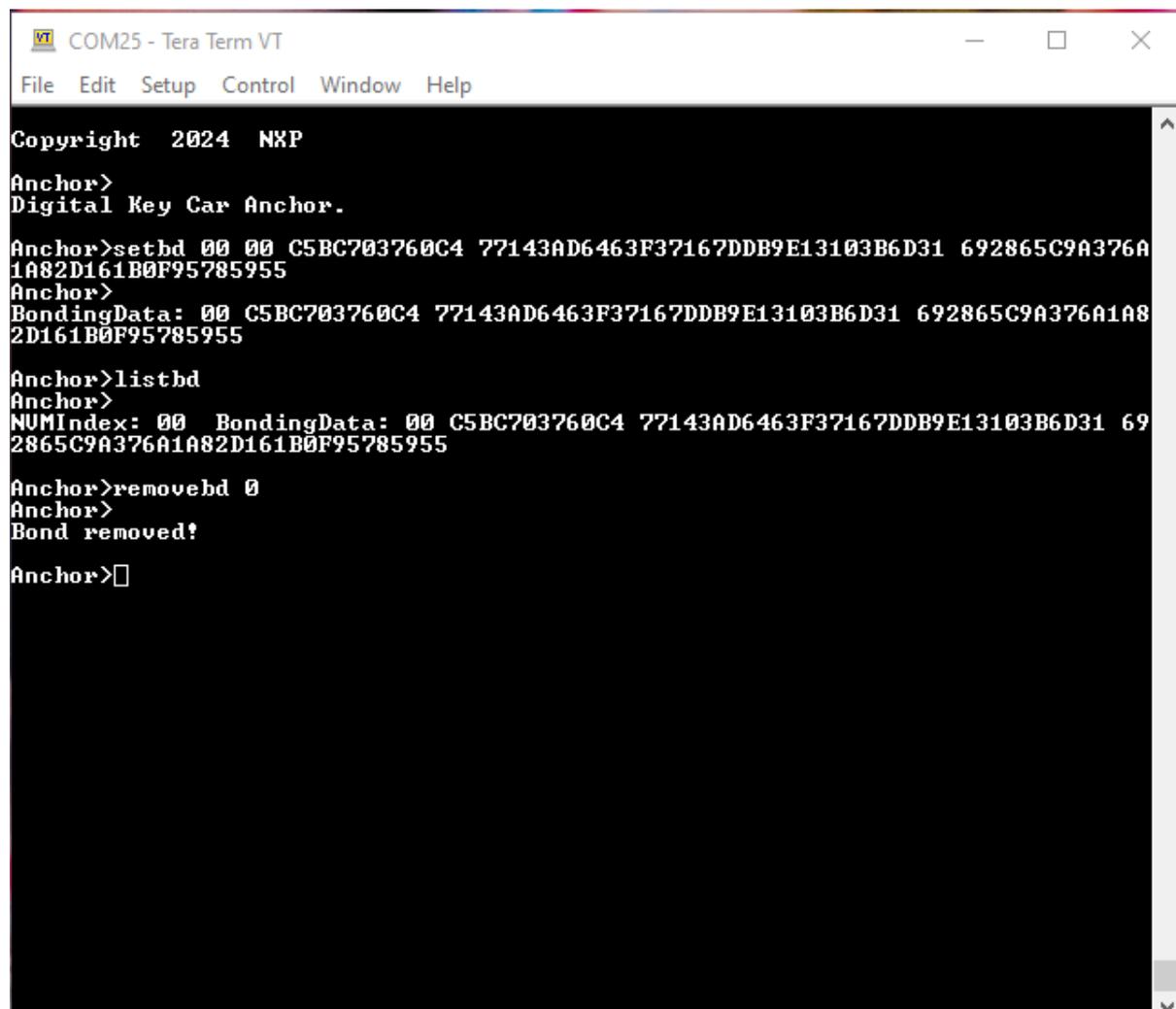
During owner pairing, the bonding data is displayed in the shell as seen in the Owner Pairing section. By passing this data to the “setbd” command on another Car Anchor, that anchor is able to connect with the original Device and perform the Passive Entry flow.

**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



```
COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>setbd 00 00 C5BC703760C4 77143AD6463F37167DDB9E13103B6D31 692865C9A376A
1A82D161B0F95785955
Anchor>
BondingData: 00 C5BC703760C4 77143AD6463F37167DDB9E13103B6D31 692865C9A376A1A8
2D161B0F95785955
Anchor>listbd
Anchor>
NUMIndex: 00 BondingData: 00 C5BC703760C4 77143AD6463F37167DDB9E13103B6D31 69
2865C9A376A1A82D161B0F95785955
Anchor>removebd 0
Anchor>
Bond removed!
Anchor>
```

**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](#)

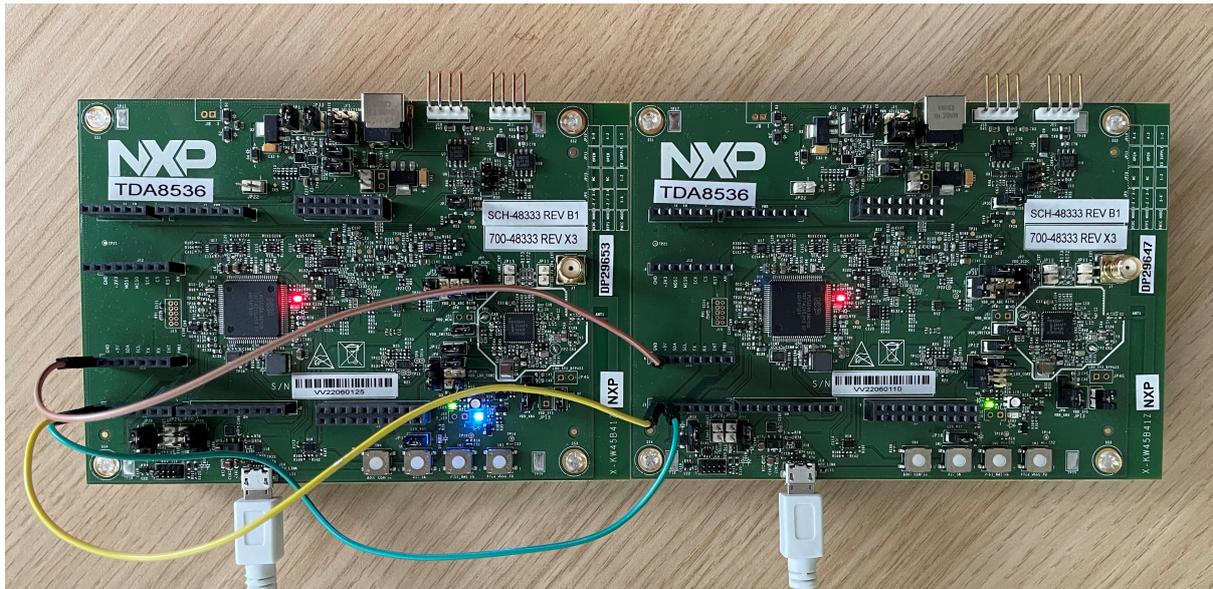
**Running the Connection Handover scenario** Connection Handover is an NXP proprietary feature that enables a Bluetooth Low Energy connection to be seamlessly transferred from one peripheral to another while the central device remains unaware. It is best exemplified by (but not limited to) the CCC Digital 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.

### Pairing and connection of First Car Anchor with Device and handover

```

COM53 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>sd op
Anchor>Advertising started - Legacy.
Connected!
L2CAP PSM Connection Complete.
Received Command Complete SubEvent: Request_owner_pairing.
SPAKE Request sent.
SPAKE Response received.
SPAKE Verify sent.
SPAKE Verify received.
Sending Command Complete SubEvent: BLE_pairing_ready
Received First_Approach_RQ.
Sending First_Approach_RS
Pairing...
BondingData: 00 D92718376000 08ECC51B79A5849D3BEAB9B071977DE9 692865C9A376A1A82D161B0
F95785955
Pairing successful.
Anchor>
Time Sync received.
Anchor>send
Anchor>listad
Anchor>
DevId AddrType Address
00 00 484E0020907E
Anchor>handover 0
Anchor>
Handover started.
Handover complete, disconnected.
Anchor>

```

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

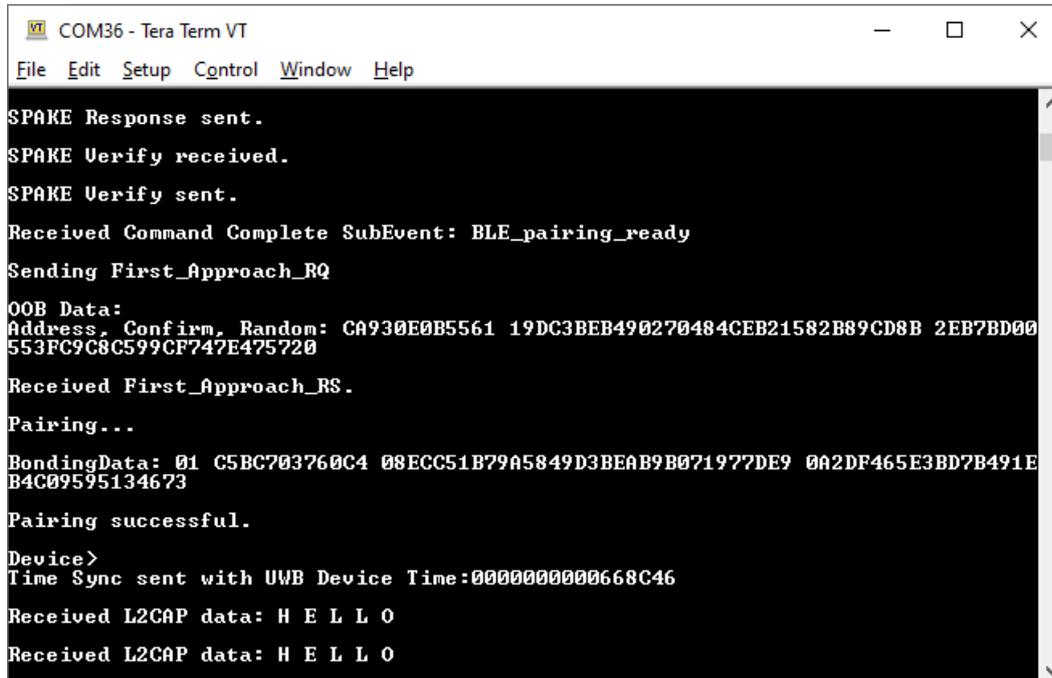
```

COM56 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>
BondingData: 00 D92718376000 08ECC51B79A5849D3BEAB9B071977DE9 692865C9A376A1A82D
161B0F95785955
Anchor>
Handover started.
Anchor>
Handover complete, connected.
Anchor>send
Anchor>

```

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

A screenshot of a Tera Term VT window titled 'COM36 - Tera Term VT'. The window displays a log of BLE pairing events. The text is as follows:

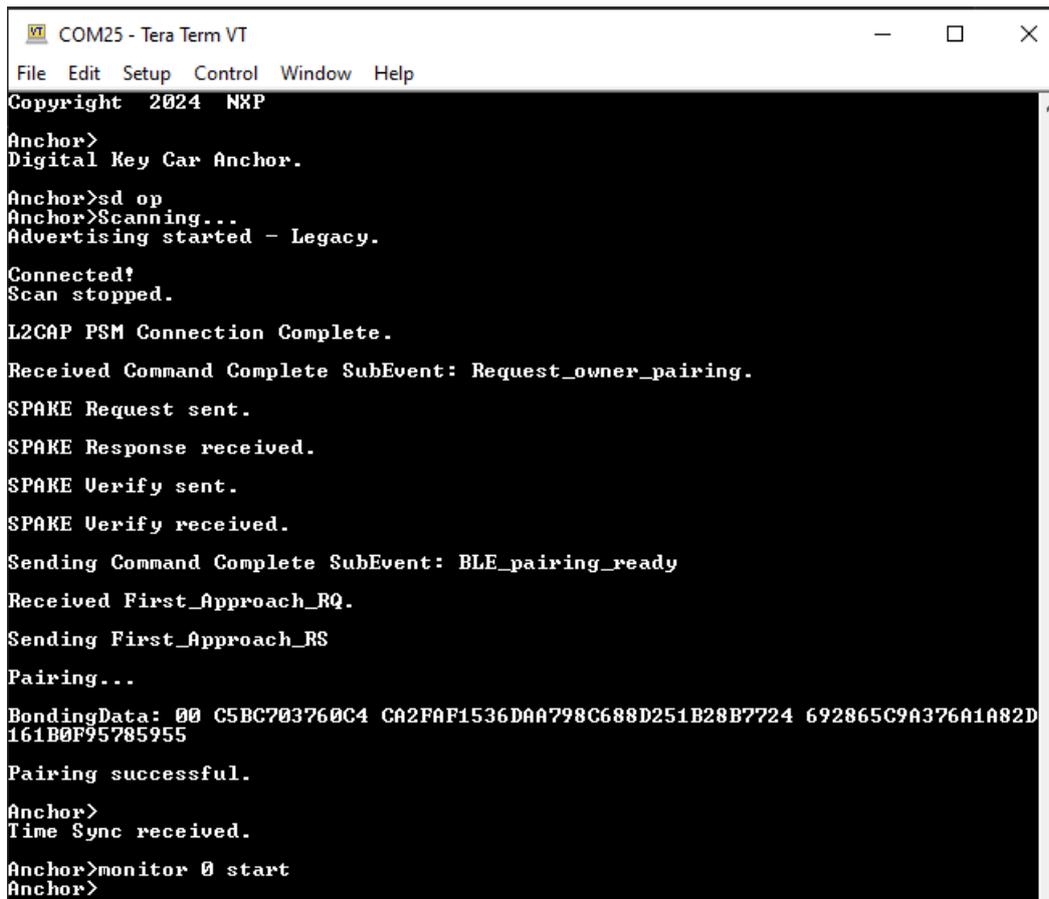
```
SPAKE Response sent.
SPAKE Verify received.
SPAKE Verify sent.
Received Command Complete SubEvent: BLE_pairing_ready
Sending First_Approach_RQ
OOB Data:
Address, Confirm, Random: CA930E0B5561 19DC3BEB490270484CEB21582B89CD8B 2EB7BD00
553FC9C8C599CF747E475720
Received First_Approach_RS.
Pairing...
BondingData: 01 C5BC703760C4 08ECC51B79A5849D3BEAB9B071977DE9 0A2DF465E3BD7B491E
B4C09595134673
Pairing successful.
Device>
Time Sync sent with UWB Device Time:0000000000668C46
Received L2CAP data: H E L L O
Received L2CAP data: H E L L O
```

**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**



```

COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.

Anchor>sd op
Anchor>Scanning...
Advertising started - Legacy.

Connected!
Scan stopped.

L2CAP PSM Connection Complete.

Received Command Complete SubEvent: Request_owner_pairing.

SPAKE Request sent.

SPAKE Response received.

SPAKE Verify sent.

SPAKE Verify received.

Sending Command Complete SubEvent: BLE_pairing_ready

Received First_Approach_RQ.

Sending First_Approach_RS

Pairing...

BondingData: 00 C5BC703760C4 CA2FAF1536DAA798C688D251B28B7724 692865C9A376A1A82D
161B0F95785955

Pairing successful.

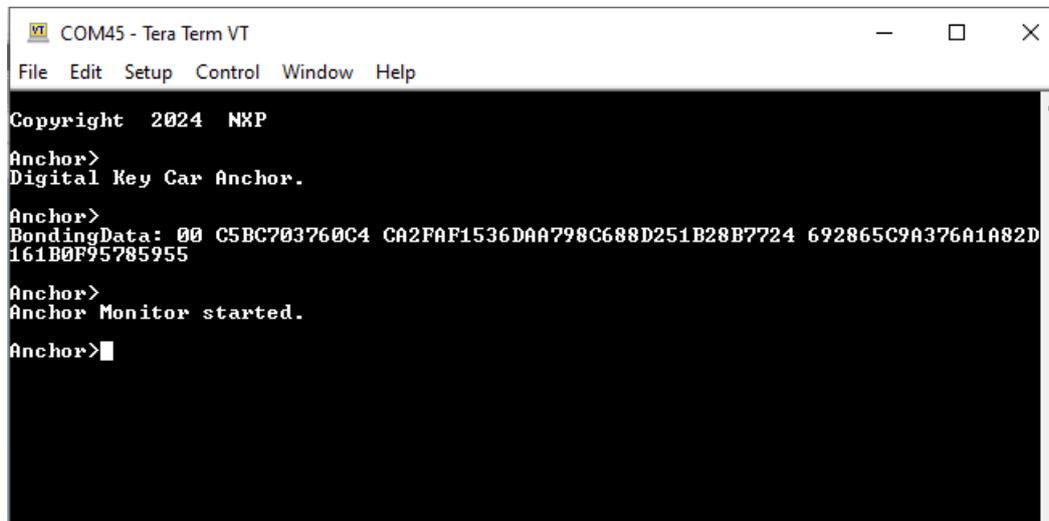
Anchor>
Time Sync received.

Anchor>monitor 0 start
Anchor>

```

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**



```

COM45 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.

Anchor>
BondingData: 00 C5BC703760C4 CA2FAF1536DAA798C688D251B28B7724 692865C9A376A1A82D
161B0F95785955

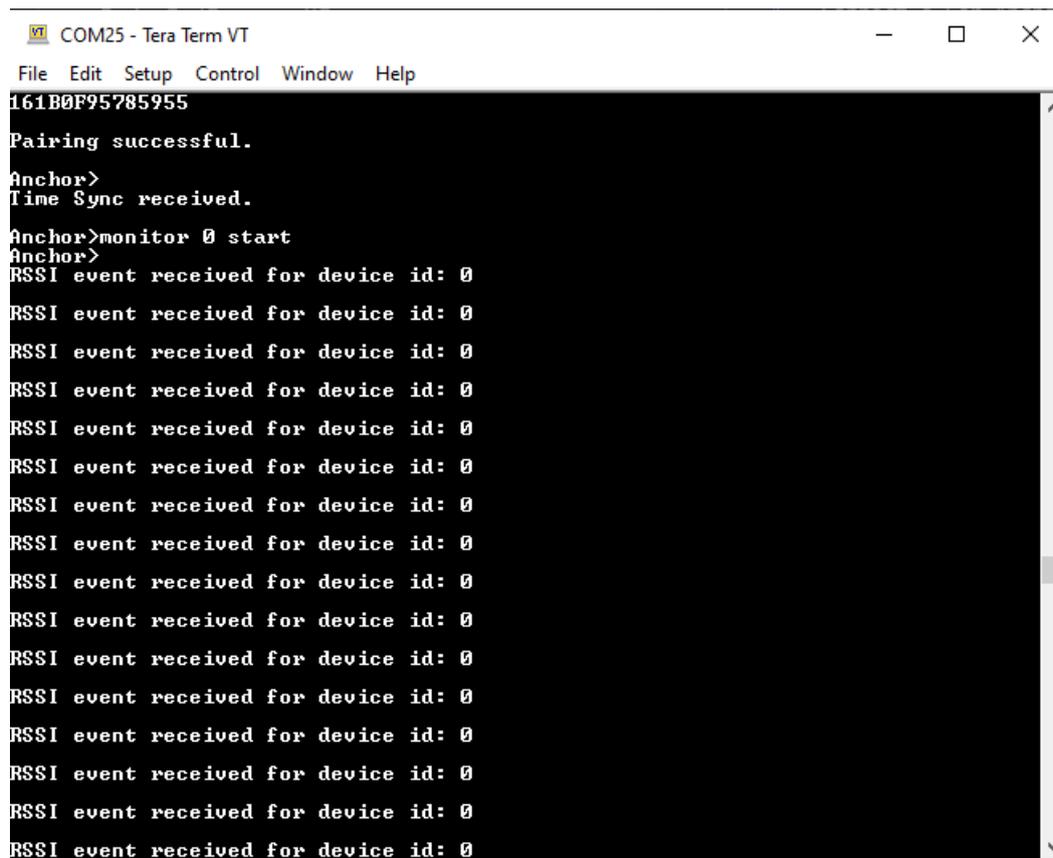
Anchor>
Anchor Monitor started.

Anchor>

```

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.**



```
COM25 - Tera Term VT
File Edit Setup Control Window Help
161B0F95785955
Pairing successful.
Anchor>
Time Sync received.
Anchor>monitor 0 start
Anchor>
RSSI event received for device id: 0
```

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**

```

COM25 - Tera Term VT
File Edit Setup Control Window Help
Anchor>
Digital Key Car Anchor.
Anchor>sd op
Anchor>Scanning...
Advertising started - Legacy.
Connected!
Scan stopped.
L2CAP PSM Connection Complete.
Received Command Complete SubEvent: Request_owner_pairing.
SPAKE Request sent.
SPAKE Response received.
SPAKE Verify sent.
SPAKE Verify received.
Sending Command Complete SubEvent: BLE_pairing_ready
Received First_Approach_RQ.
Sending First_Approach_RS
Pairing...
BondingData: 00 C5BC703760C4 AEC40ADCFCBF9D55650B8003820911F 692865C9A376A1A82D
161B0F95785955
Pairing successful.
Anchor>
Time Sync received.
Anchor>packetmon 0 start
Anchor>

```

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**

```

COM45 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>
BondingData: 00 C5BC703760C4 AEC40ADCFCBF9D55650B8003820911F 692865C9A376A1A82D
161B0F95785955
Anchor>
Anchor Monitor started.
Anchor>

```

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**

```

COM25 - Tera Term VT
File Edit Setup Control Window Help
Anchor>
Packet monitor event received for device id 0, from central with RSSI: -35
Packet monitor event received for device id 0, from central with RSSI: -37
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -62
Packet monitor event received for device id 0, from central with RSSI: -36
Packet monitor event received for device id 0, from central with RSSI: -36
Packet monitor event received for device id 0, from central with RSSI: -38
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -62
Packet monitor event received for device id 0, from peripheral with RSSI: -64
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from central with RSSI: -36
Packet monitor event received for device id 0, from peripheral with RSSI: -62
Packet monitor event received for device id 0, from central with RSSI: -35
Packet monitor event received for device id 0, from central with RSSI: -36
Packet monitor event received for device id 0, from central with RSSI: -36
Packet monitor event received for device id 0, from central with RSSI: -37
Packet monitor event received for device id 0, from central with RSSI: -36
Packet monitor event received for device id 0, from peripheral with RSSI: -62
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from central with RSSI: -35

```

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 the 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:**

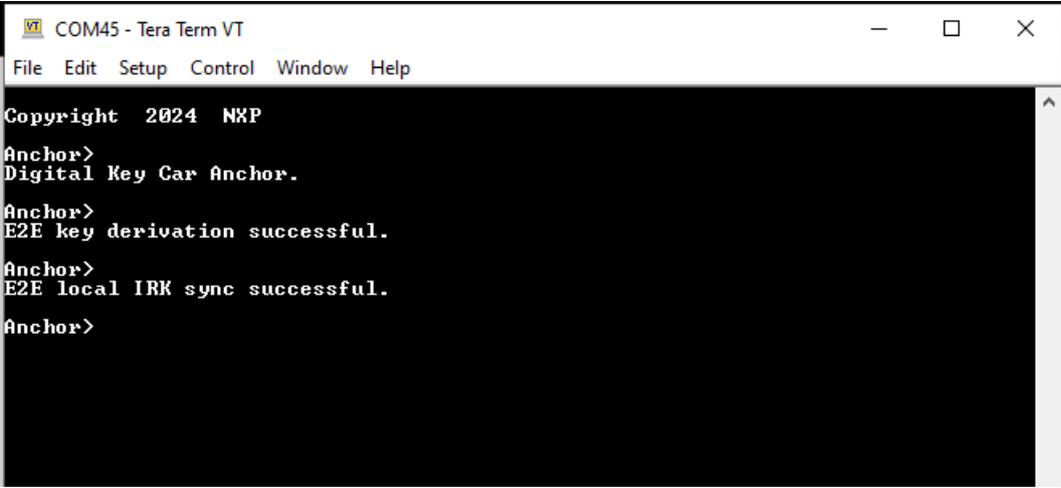
- Three boards (two act as Car Anchors, one as a Device). The demo is currently limited to a maximum of two Car Anchors.
- The two Car Anchors must be connected via a serial interface as described in the section above.

- 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

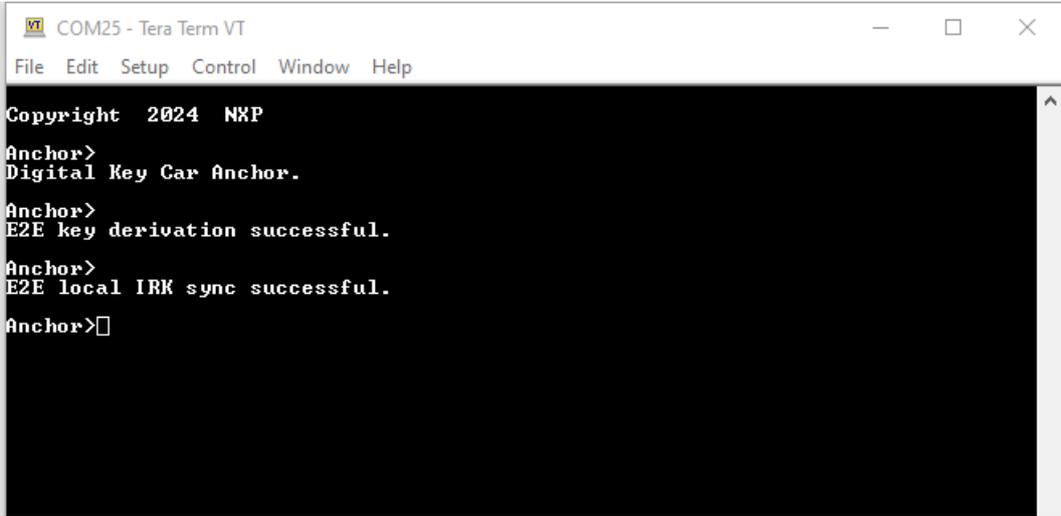


```

COM45 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>
E2E key derivation successful.
Anchor>
E2E local IRK sync successful.
Anchor>

```

#### Car Anchor B starts up and syncs with Car Anchor B



```

COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>
E2E key derivation successful.
Anchor>
E2E local IRK sync successful.
Anchor>

```

- 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

```

COM45 - Tera Term VT
File Edit Setup Control Window Help
Anchor>Scanning...
Advertising started - Legacy.
Connected!
Scan stopped.
L2CAP PSM Connection Complete.
Received Command Complete SubEvent: Request_owner_pairing.
SPAKE Request sent.
SPAKE Response received.
SPAKE Verify sent.
SPAKE Verify received.
Sending Command Complete SubEvent: BLE_pairing_ready
Received First_Approach_RQ.
Sending First_Approach_RS
Pairing...
BondingData: 00 C5BC703760C4 5F2092BB51AF5AB530F016E48B74454ED651780D81B35EB573F
4FE8C11F0BC124CE4C77A056CE888 692865C9A376A1A82D161B0F95785955
Pairing successful.
Anchor>
Time Sync received.
Anchor>

```

**Car Anchor B has received the Bonding Data via the serial interface**

```

COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>
E2E key derivation successful.
Anchor>
E2E local IRK sync successful.
Anchor>
Received peer SKD: CDDCE9C2D0A8A06EA65CF8EE0A564CC6
BondingData: 00 C5BC703760C4 3714B3CC2C01CFABC2370B583A93E57F9B7B24878E42A9F6E68
A3F9B609456DAABDF4756EC8E17D4 692865C9A376A1A82D161B0F95785955
Anchor>

```

- Disconnect Car Anchor A by sending the “dcont” 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**

```

COM25 - Tera Term VT
File Edit Setup Control Window Help
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>
E2E key derivation successful.
Anchor>
E2E local IRK sync successful.
Anchor>
Received peer SKD: CDDCE9C2D0A8A06EA65CF8EE0A564CC6
BondingData: 00 C5BC703760C4 3714B3CC2C01CFABC2370B583A93E57F9B7B24878E42A9F6E68
A3F9B609456DAABDF4756EC8E17D4 692865C9A376A1A82D161B0F95785955
Anchor>sd pe
Anchor>Scanning...
Advertising started - Legacy.
Advertising started - Extended LR.
Connected!
Advertising started - Extended LR.
Scan stopped.
L2CAP PSM Connection Complete.
Time Sync received.
Anchor>

```

**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:

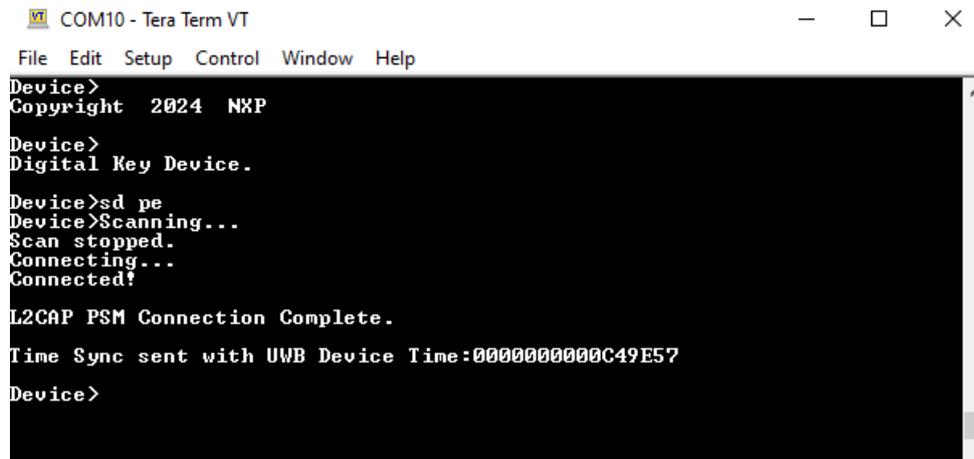
- In `app_preinclude.h` file, set `gBLE60_DdecisionBasedAdvertisingFilteringSupport_d` to `TRUE`.

**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.

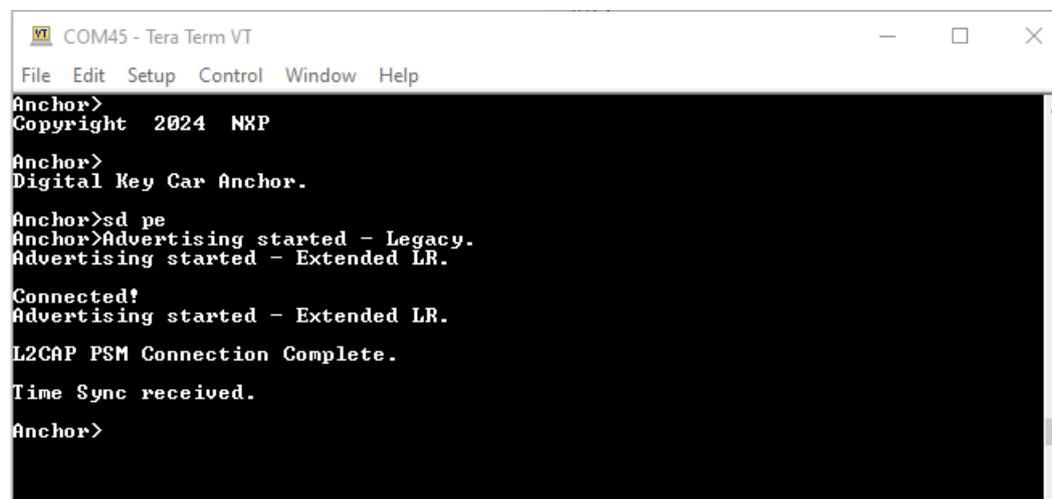
1. To run the Passive Entry with DBAF demo, reset the recently paired boards using either the **RST** switch or the “reset” command in the shell. Do NOT perform a factory reset, as this removes the bonding information from non-volatile memory.
2. After reset, run the “sd pe” command on the Car Anchor and “sd pe” command on the Device as shown in Figure 1 and Figure 2.

**Passive Entry with DBAF on Device**



```
COM10 - Tera Term VT
File Edit Setup Control Window Help
Device>
Copyright 2024 NXP
Device>
Digital Key Device.
Device>sd pe
Device>Scanning...
Scan stopped.
Connecting...
Connected!
L2CAP PSM Connection Complete.
Time Sync sent with UWB Device Time:0000000000C49E57
Device>
```

### Passive Entry with DBAF on Car Anchor



```
COM45 - Tera Term VT
File Edit Setup Control Window Help
Anchor>
Copyright 2024 NXP
Anchor>
Digital Key Car Anchor.
Anchor>sd pe
Anchor>Advertising started - Legacy.
Advertising started - Extended LR.
Connected!
Advertising started - Extended LR.
L2CAP PSM Connection Complete.
Time Sync received.
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.

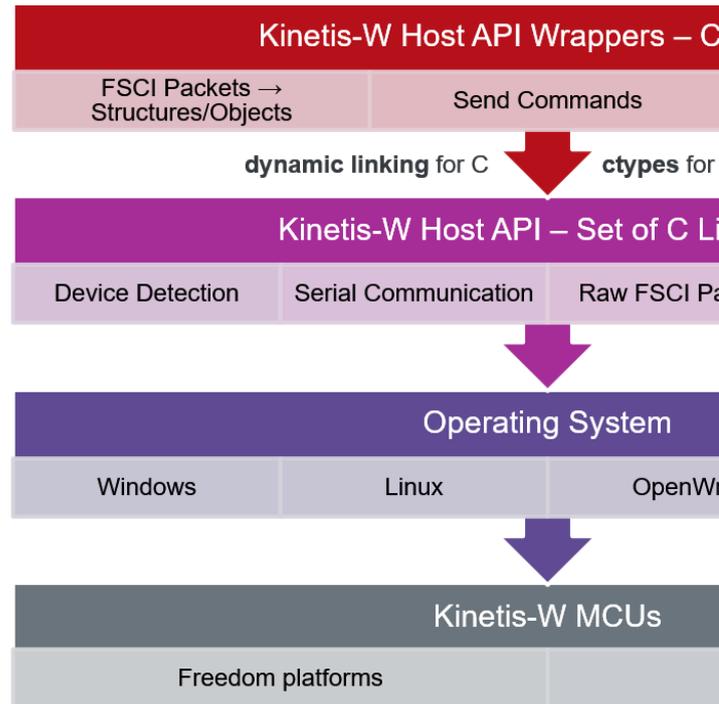
The user can compile the *black box* image of the ble\_fscibb software application using IAR Embedded Workbench for Arm (EWARM) or MCUXpresso IDE. For information on how to build the application see additional documentation provided in the package.

**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	
h sdk	
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.
h sdkError.h	Macros for error reporting.
h sdkLogger.h	Logger implementation for debugging.
h sdkOSCommon.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.
h sdk.conf	Configuration file to control FSCI-ACKs.
sys	Implementation of the system/OS portable base module.
EventManager.c	
h sdkEvent.c	
h sdkFile.c	
h sdkLock.c	
h sdkLogger.c	
h sdkSemaphore.c	
h sdkThread.c	
MessageQueue.c	
RawFrame.c	
utils.c	
ConnectivityLibrary.sln	Microsoft Visual Studio solution file.
HSDK.rc	
Makefile	
README.md	
resource.h	
h sdk-c	Described in Host API C Bindings
h sdk-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 shared libraries (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 `h sdk-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 architecture (not Windows architecture) copy the appropriate HSDK.dll from `h sdk-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 (`vc_redist_x86.exe` or `vc_redist_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 Compilers](#).

## Using local built library

1. Depending on your Python environment architecture (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 Compilers](#).

Optionally, copy the `h sdk\res\h sdk.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

```

h sdk-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 than Host SDK, the user needs to provide the implementation for

```
void FSCI_transmitPayload (
 void *arg, /* Optional argument passed to the function */
 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
h sdk
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

```

**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 `h SDK-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 Bluetooth 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. `GATTDynamicAddCharacteristicDeclarationAndValue` 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 = GATTDynamicAddCharacteristicDeclarationAndValue(
 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.
 ↪None_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
<ParmName>	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\_HandoverGetCsLIContext

```
bleResult_t Gap_HandoverGetCsLIContext(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.
- `rsiPacket`: 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(bleResult_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.

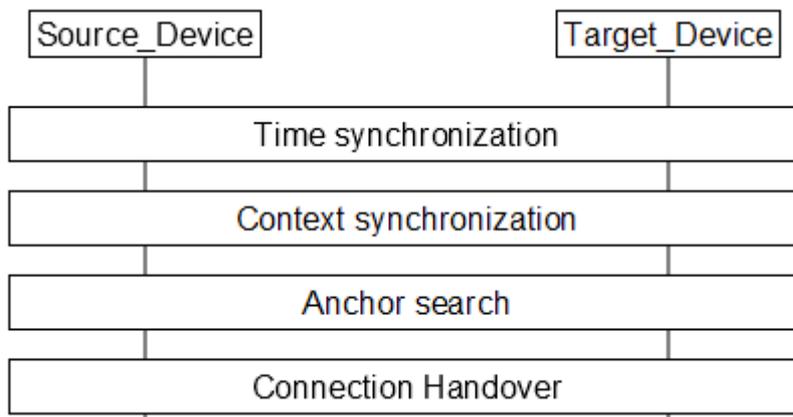
**Parameters:** None

**Returns:** bleResult\_t - Result of the operation.

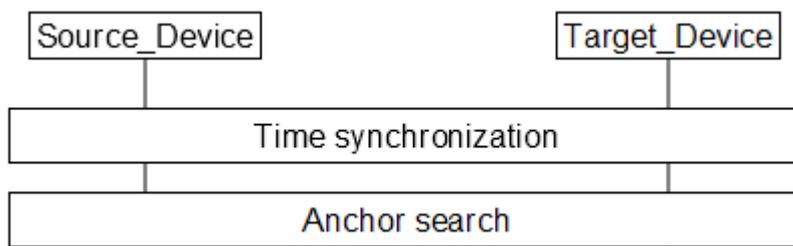
**Procedures** This section describes the available Connection Handover procedures. A procedure is executed between a device in a connection, referred as the Source device, and another device involved in the Connection Handover or Anchor/Packet Monitoring process, referred to as the Target device.

**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.

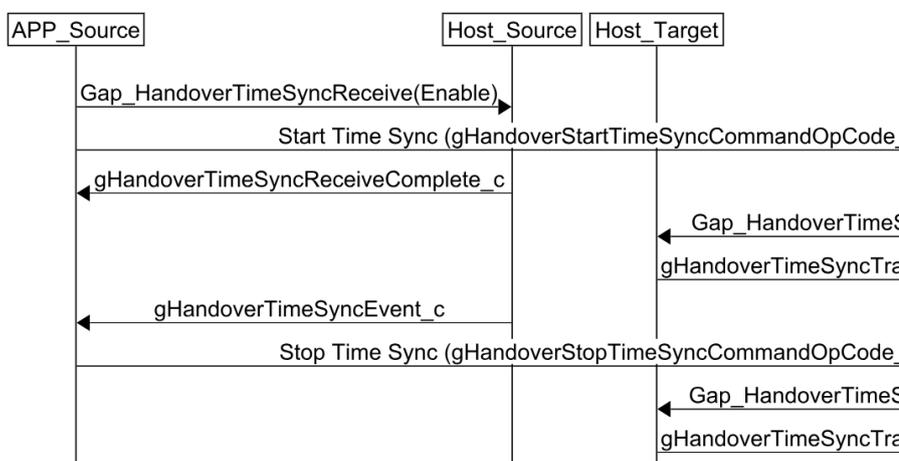
**Time synchronization** The Time Synchronization procedure enables precise timing alignment between the Source and Target devices for the Connection Handover and Anchor/Packet monitoring processes. The timing information obtained is further required for the Anchor Search procedure.

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;'`



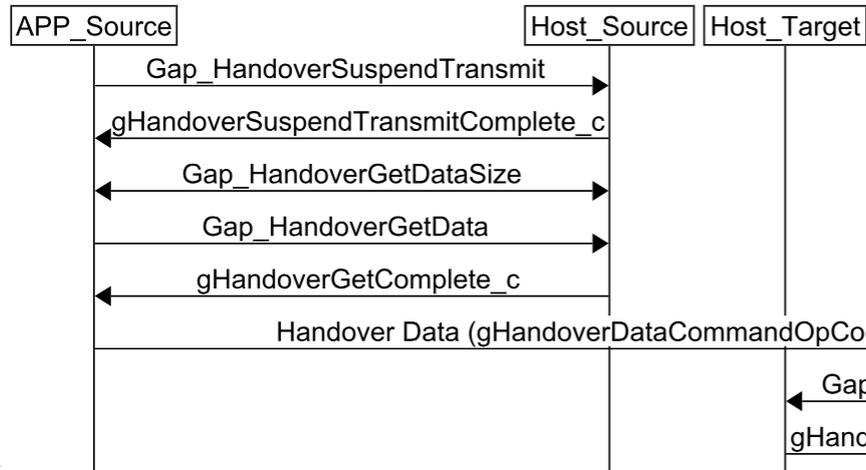
Time Synchronization signaling chart

**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 provide 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.

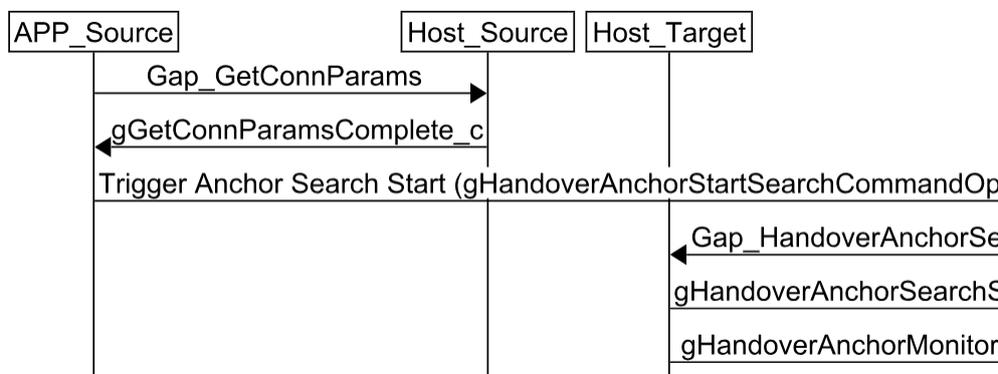


Context Synchronization signaling chart

**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

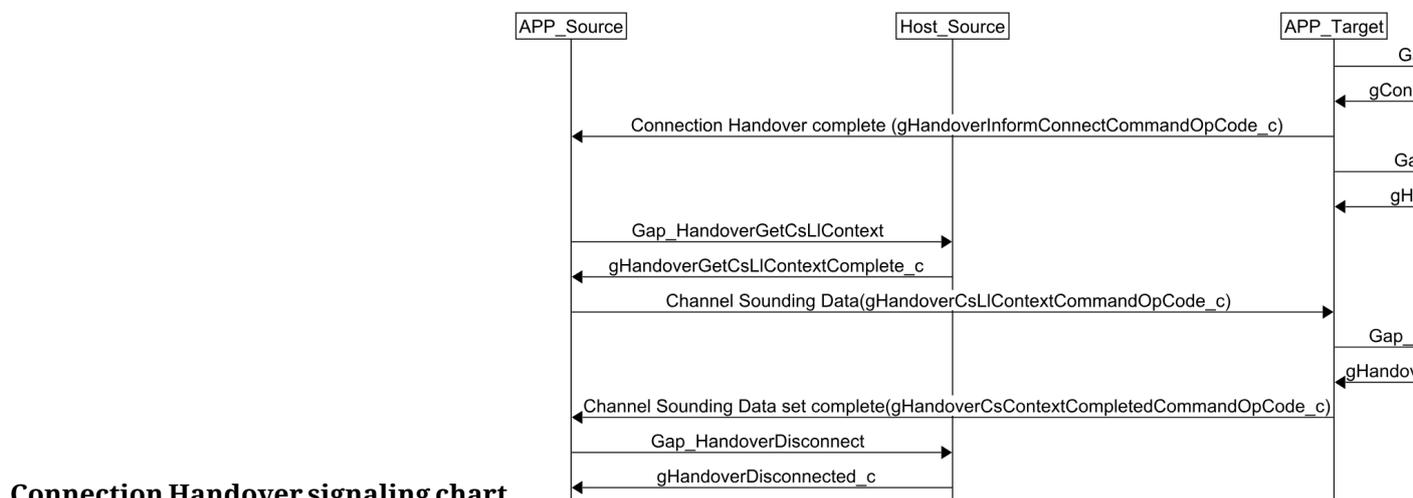
**Connection Handover** The Connection Handover procedure may be performed after a successful Anchor Search procedure and consists in the Target device taking over the connection from the Source device.

**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.

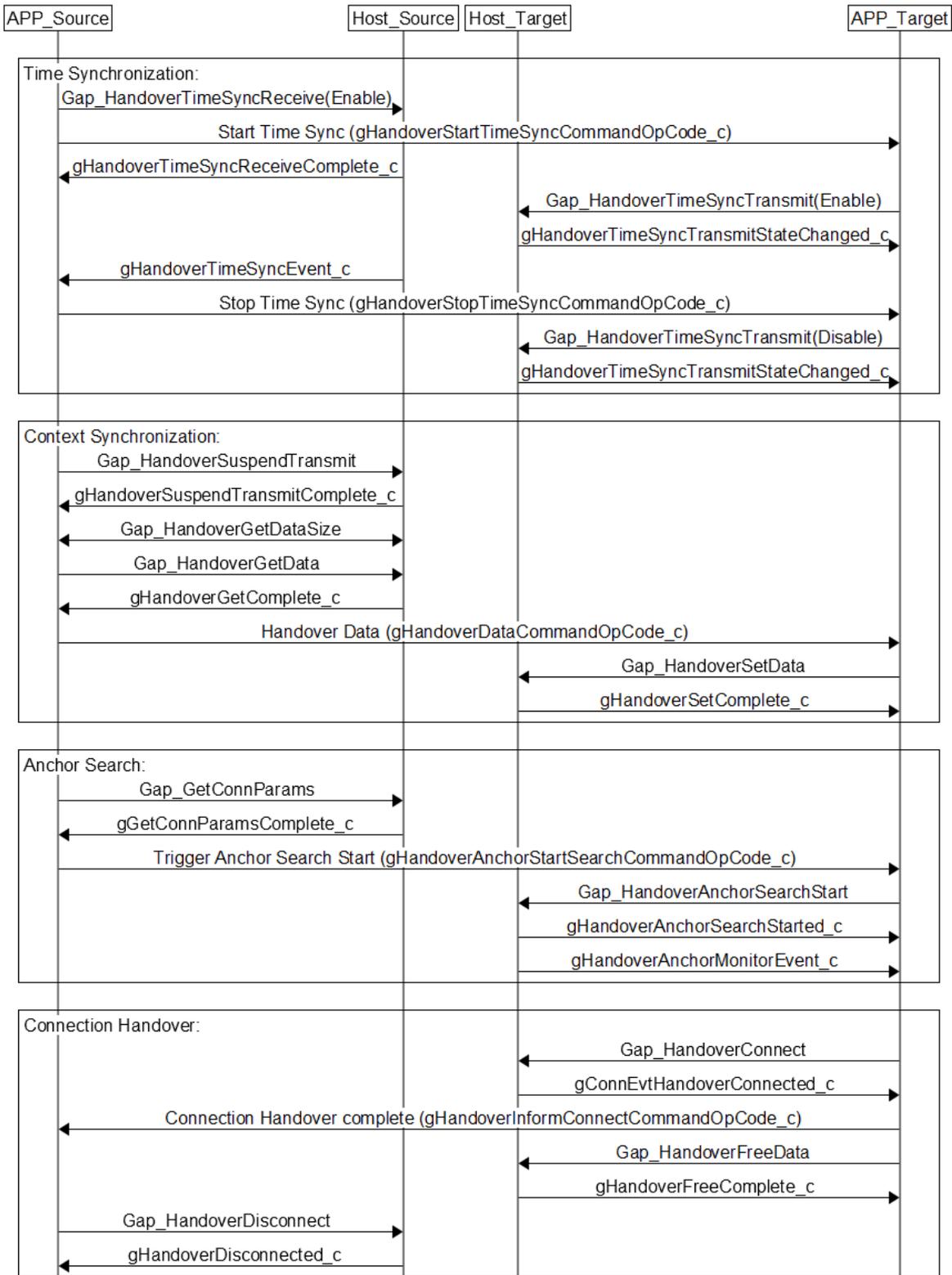
Additional steps for Channel Sounding applications:

1. After step 3 above, retrieve the Channel Sounding context on the Source device by calling `Gap_HandoverGetCsLIContext()` and wait for the `gHandoverGetCsLIContextComplete_c` event.
2. Send the Channel Sounding context to the Target device.
3. Set the context in the Target device by calling `Gap_HandoverSetCsLIContext()` and wait for the `gHandoverSetCsLIContextComplete_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.

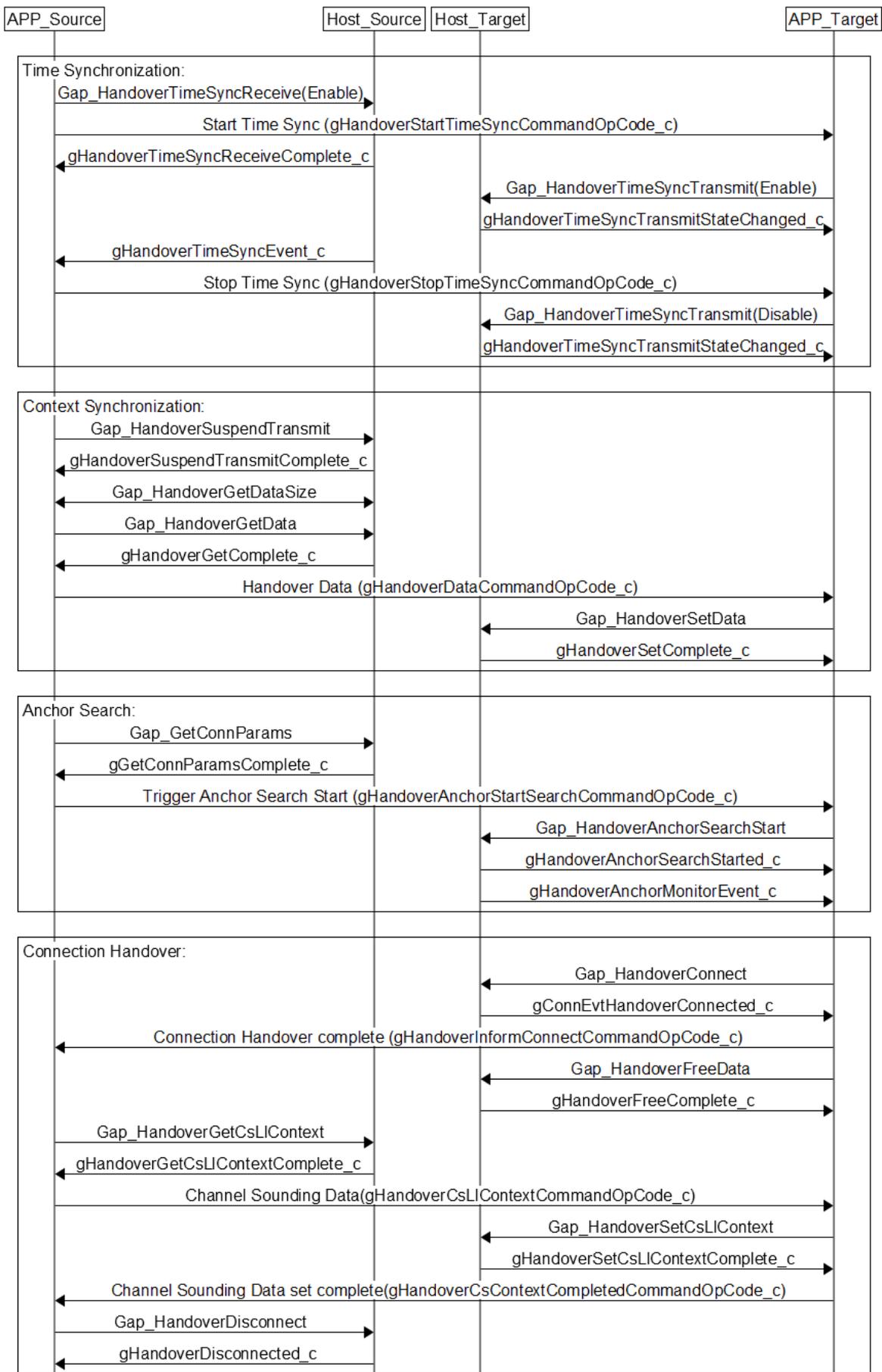


**Connection Handover signaling chart**

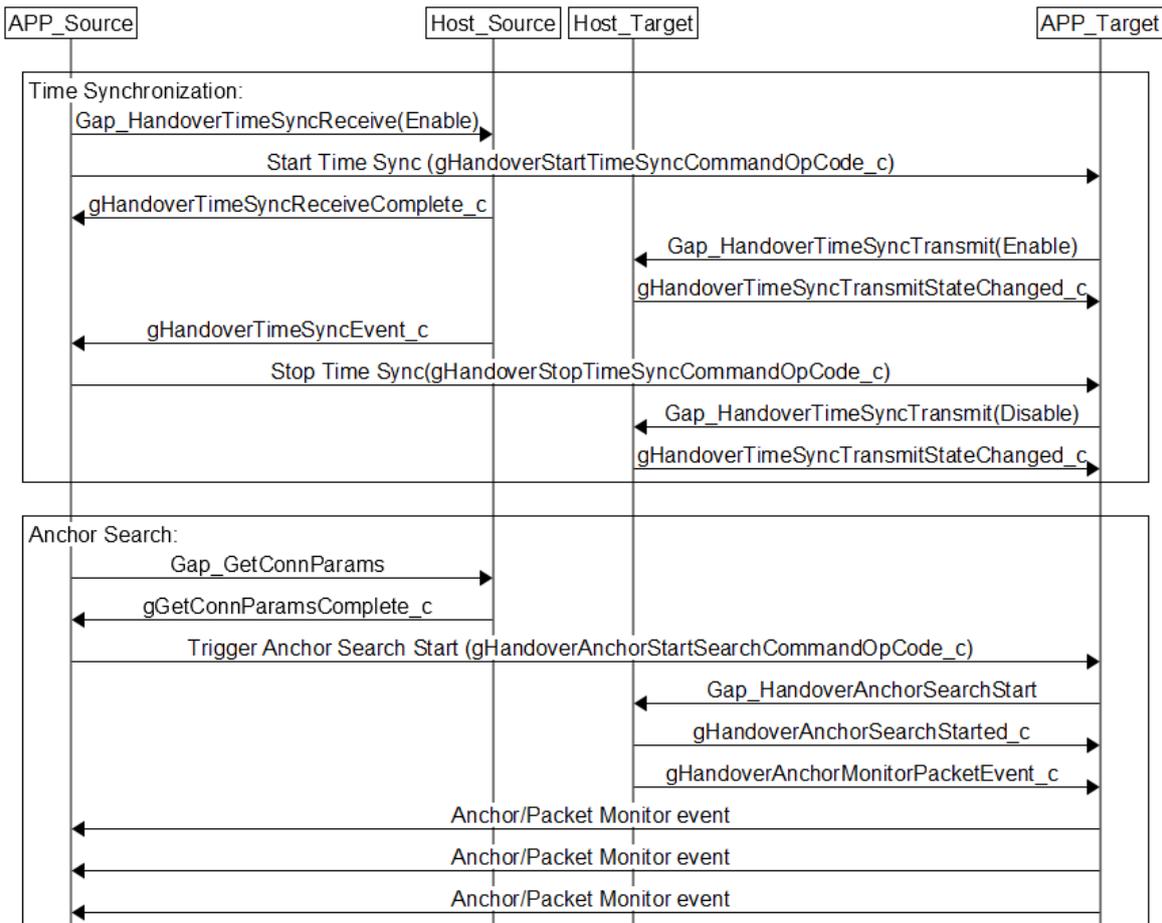
**Procedures - Detail view Connection Handover**



### Connection Handover with Channel Sounding



## 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 '1111111100000111101'...

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

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 `gapConnection-Event_t.eventData.failReason`.

enumerator `gConnEvtParameterUpdateRequest_c`

A connection parameter update request has been received. Data in `gapConnection-Event_t.eventData.connectionUpdateRequest`.

enumerator `gConnEvtParameterUpdateComplete_c`

The connection has new parameters. Data in `gapConnection-Event_t.eventData.connectionUpdateComplete`.

enumerator `gConnEvtLeDataLengthChanged_c`

The new TX/RX Data Length parameters. Data in `gapConnection-Event_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 `gapConnection-Event_t.eventData.channelMap`

enumerator `gConnEvtChannelMapReadFailure_c`

Channel Map reading could not be performed. Data in `gapConnection-Event_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 `gap-ConnectionEvent_t.eventData.connIqReport`.

enumerator `gConnEvtCteRequestFailed_c`

CTE Request to a connected peer has failed. Data in `gapConnection-Event_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 `gapConnection-Event_t.eventData.pathLossThreshold`.
- enumerator `gConnEvtTransmitPowerReporting_c`  
Received a Transmit Power report. Data in `gapConnection-Event_t.eventData.transmitPowerReporting`.
- enumerator `gConnEvtEnhancedReadTransmitPowerLevel_c`  
Local information has been read from Controller. Data in `gapConnection-Event_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 occurred.

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 gapDecisionInstructions-Data\_t structures. containing decision instruction for each test.

**Return values**

- gBleFeatureNotSupported\_c – in case gLeExtendedAdv\_c or gLeDecision-BasedAdvertisingFiltering\_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\_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  $F[\text{MHz}] = 2402 + 2 * \text{index}$ . 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 advertisement 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\_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 on-going 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 `gGapDefaultAdvertisingParameters_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 gGapDefault-PeriodicAdvParameters\_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  $6 \leq \text{dataLength} \leq 8$

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 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.

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 occurred.

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 `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(*gattClientEnhancedMultipleValueNotificationCallback\_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\_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 maxStringLength 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 gSmImproperKeyDistributionField\_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 not 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

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 `gapConnection-Event_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

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

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 gSafeStopLITx\_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).

$\text{connIntervalmin} = \text{Conn\_Interval\_Min} * 1.25 \text{ ms}$

Value of 0xFFFF indicates no specific minimum.

`gcConnectionIntervalMaxDefault_c`

$\text{connIntervalmax} = \text{Conn\_Interval\_Max} * 1.25 \text{ ms}$

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 Diversifier size

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

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_ExtractTwoByteValue(buf)`  
Returns a `uint16_t` from a buffer, big-endian

`Utils_ExtractThreeByteValue(buf)`  
Returns a 3-byte value from a buffer, big-endian

`Utils_ExtractFourByteValue(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:  $\text{transmitWindowSize} = \text{winSize} * 1.25 \text{ ms}$

uint16\_t winOffset

Used to indicate the transmitWindowOffset value as:  $\text{transmitWindowOffset} = \text{winOffset} * 1.25 \text{ ms}$

uint16\_t interval

Used to indicate the connInterval value, as:  $\text{connInterval} = \text{interval} * 1.25 \text{ ms}$

uint16\_t latency

Used to indicate the connPeripheralLatency value as:  $\text{connPeripheralLatency} = \text{latency}$

uint16\_t timeout

Used to indicate the connSupervisionTimeout value as:  $\text{connSupervisionTimeout} = \text{timeout} * 10 \text{ ms}$

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 handoverLIPendingDataIndication\_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

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.  $((1 \ll 13) - 1) * 300$

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  $((\text{max param length}) - \text{sizeof}(\text{numTests})) / ((\text{sizeof}(\text{TestFlag}) + \text{sizeof}(\text{TestField}) + \text{sizeof}(\text{TestParameters})) = (255-1)/(1 + 1 + 16)$

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

## 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 gSourceCidAlreadyAllocated\_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

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

*handoverLIPendingDataIndication\_t* handoverLIPendingDataIndication

*handoverConnectionUpdateProcedureEvent\_t* handoverConnectionUpdateProcedure

*handoverApplyConnectionUpdateProcedure\_t* handoverApplyConnectionUpdateProcedure

*vendorUnitaryTestEvent\_t* unitaryTestData

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 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

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 (rpmmsg, 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

### 7.0.3 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 TSTM\_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\_H SLOT) 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 `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 gRng-MaxRequests\_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 algorithm 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\_config.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, frdmxcw72 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 SBT-SNOOP\_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.
    - \* Explicated RNG error codes
    - \* Added argument checks for all APIs and return gRngBadArguments\_d (-2) when wrong
    - \* added checks of RNG initialization and return gRngNotInitialized\_d (-3) when not done
    - \* fixed correctness 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 defragmentation 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()` renamed 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

### Unsupported 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 firmware 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\_HSL0T 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() has an important fix for reading the Timestamp in TSTMRO
  - 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 the 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 gPlatformUseUniqueIdFor15\_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 functionality, 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 (dc/dc)
  - 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 trimming 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 oscillator 32k to 3
- [platform] Switch back to RNG for MAC Address 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 gBoard-UseUart0HwFlowControl, 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

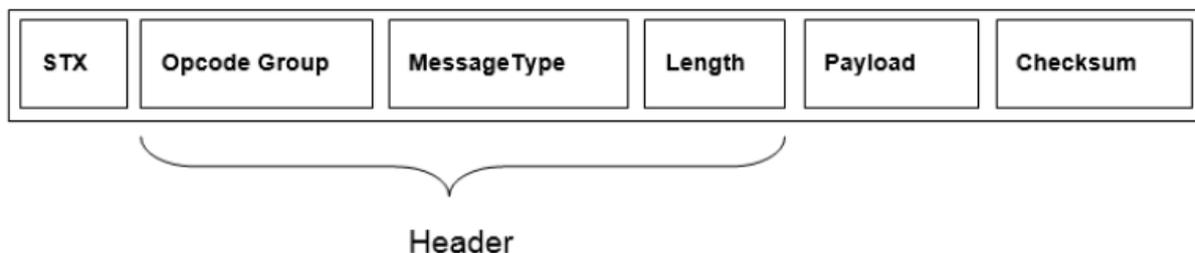
**Overview** The Framework Serial Communication Interface (FSCI) is both a software module and a protocol that allows monitoring and extensive testing of the protocol layers. It also allows separation of the protocol stack between two protocol layers in a two processing entities setup, the host processor (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.

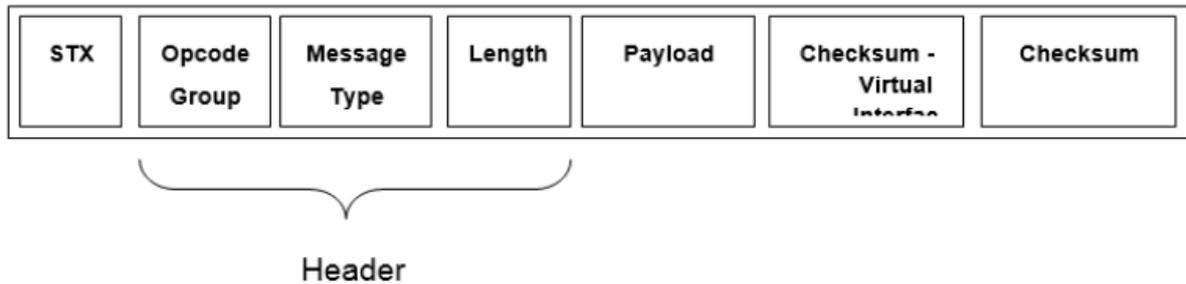
An entity which needs to be interfaced to the FSCI module can use the API to register opcodes to specific interfaces. After doing so, any packet coming from that interface with the same opcode triggers a callback execution. Two or more entities cannot register the same opcode on the same interface, but 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 configures 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 re-sent 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_transmitFormattedPacket(pClientPacket, interfaceId);
 return;
 }
 case 0x02:
 {
 /* Allocate 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_transmitFormattedPacket(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 allotted 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 Main-Flash 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 fulfill 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 resources (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.

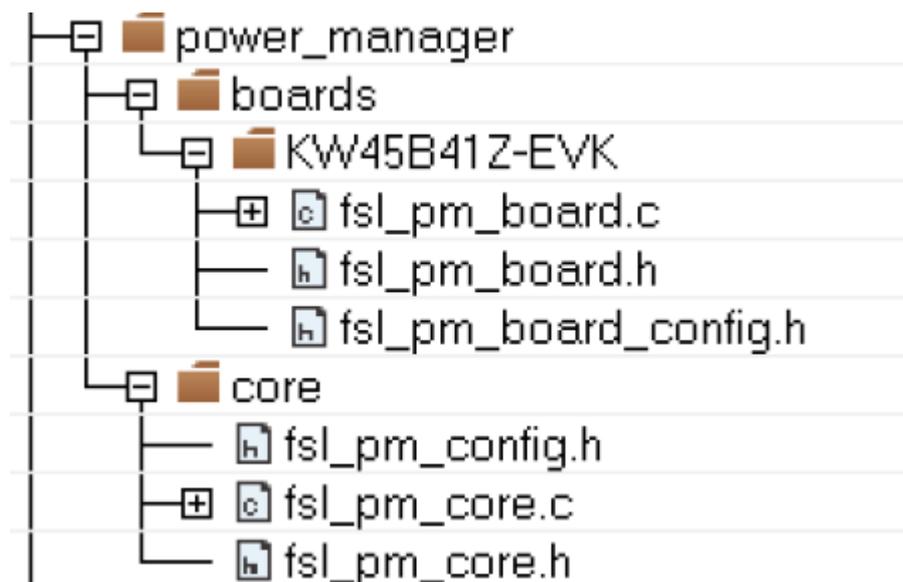
Other peripherals that require specific action on low power entry and restore on low power exit should be added to low power board files. For more details, refer to section Low power board file update

**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.

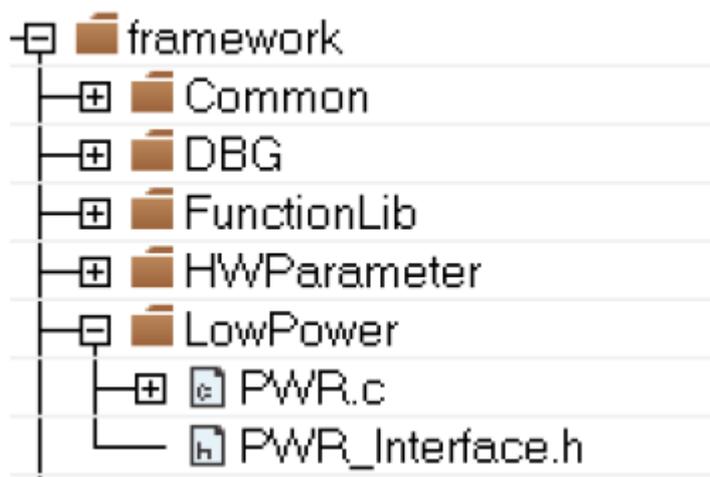
However, users can start directly from the application project and implement low power in it, by performing the steps described in the following sections.

**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.

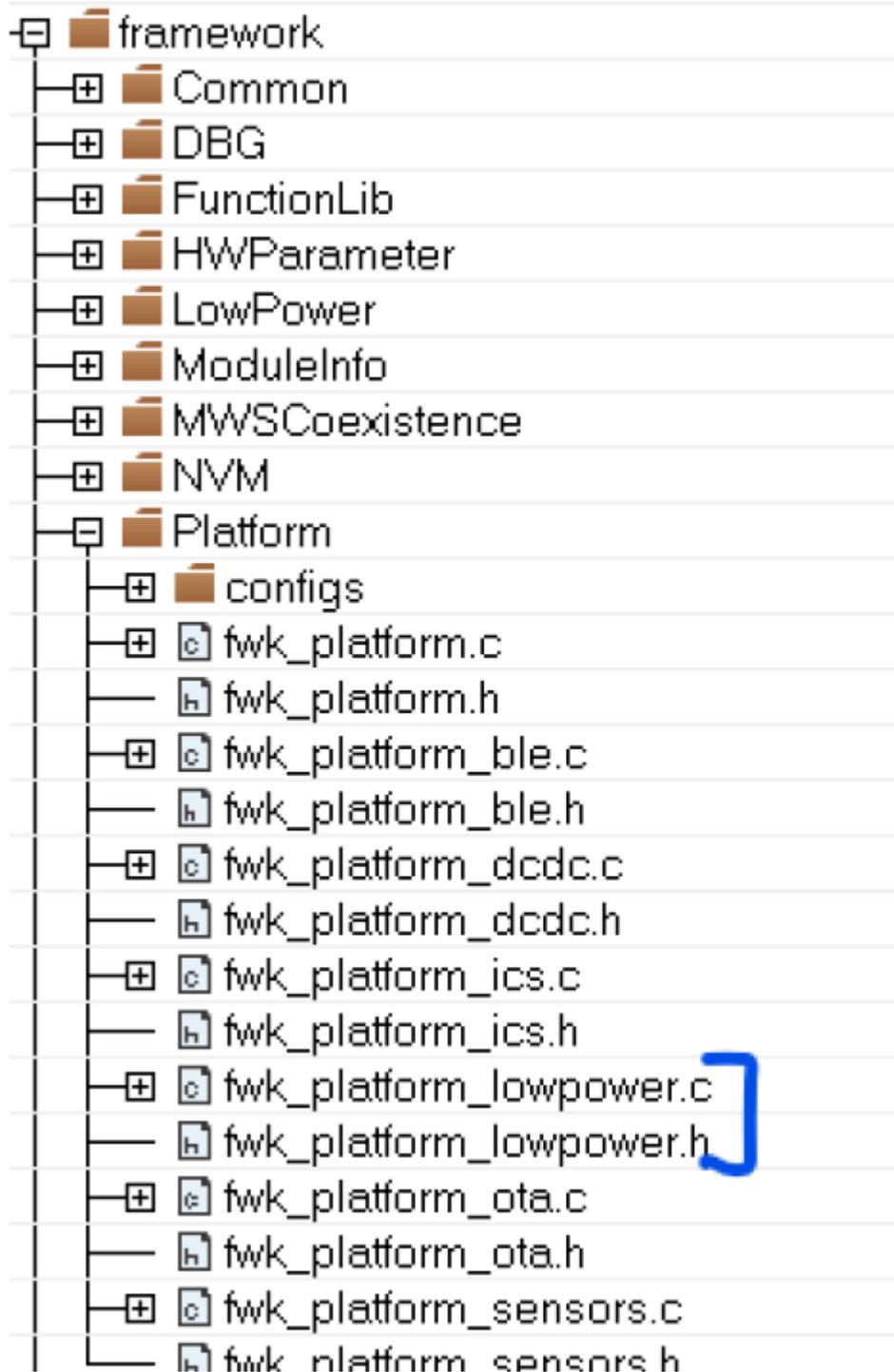
**2.1.2 - PWR connectivity framework module** `PWR.c` `PWR_Interface.h` shall be added to your application projects :



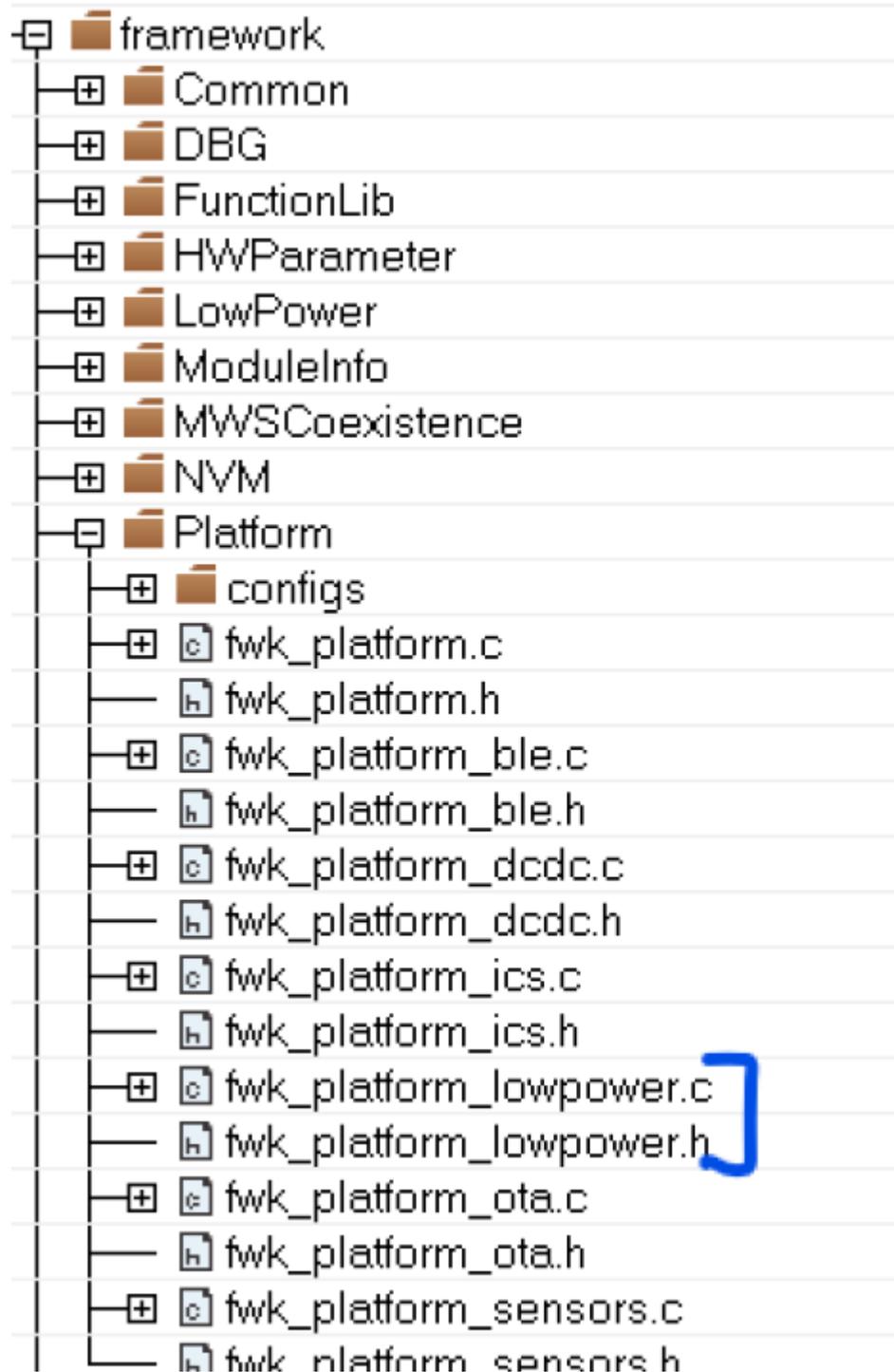
Optionally, in order to support SysTick less mode, `PWR_systicks.c` or `PWR_systicks_bm.c` could also be added.

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 tickless 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);

#ifdef (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 serial 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 fonctionnal to trigger upcoming interrupts */
 }
}
```

(continues on next 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 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 aggressive 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 written 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 `gApp-TaskWaitTimeout_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 suspended, 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 manual 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 save 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 take 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 than 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.

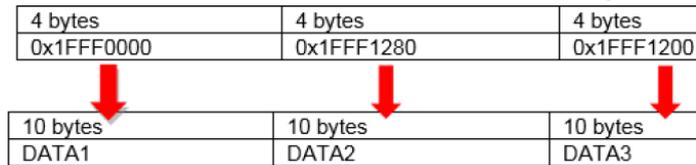
A RAM table entry has the following structure:

- 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  $0x1FFF8000 + 2 * sizeof(void*)$ .

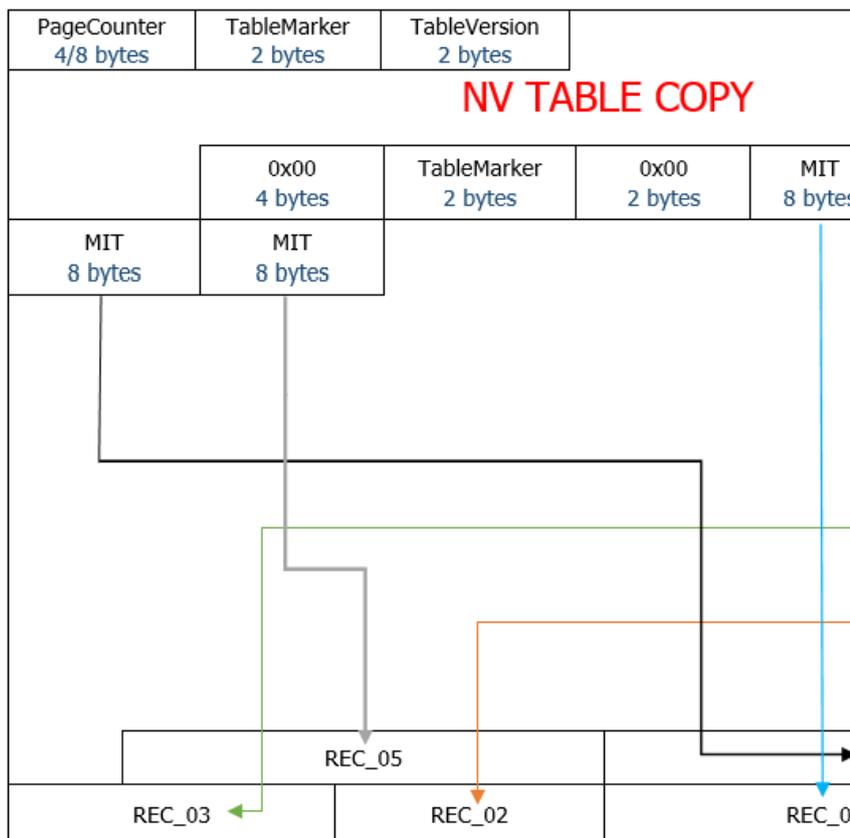
The page validity is guaranteed by the page counter. The page counter is a 32-bit value and is written at the beginning and at the end of the active page. The values need to be equal to consider the page a valid one. The value of the page counter is incremented after each page copy operation. A page 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

entryId	entryType	elemCnt	elemSz
2 bytes	2 bytes	2 bytes	2 bytes

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:

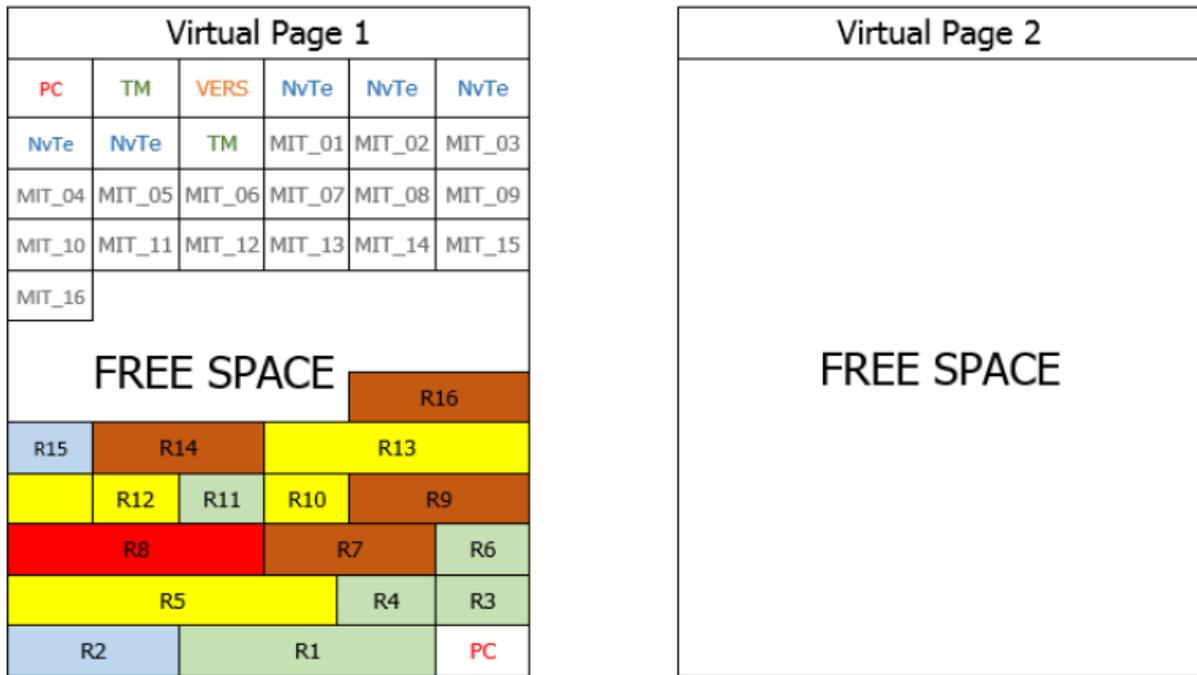
VSB	entryID	elemIdx	recordOffset	VEB
1 byte	2 bytes	2 bytes	2 bytes	

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.

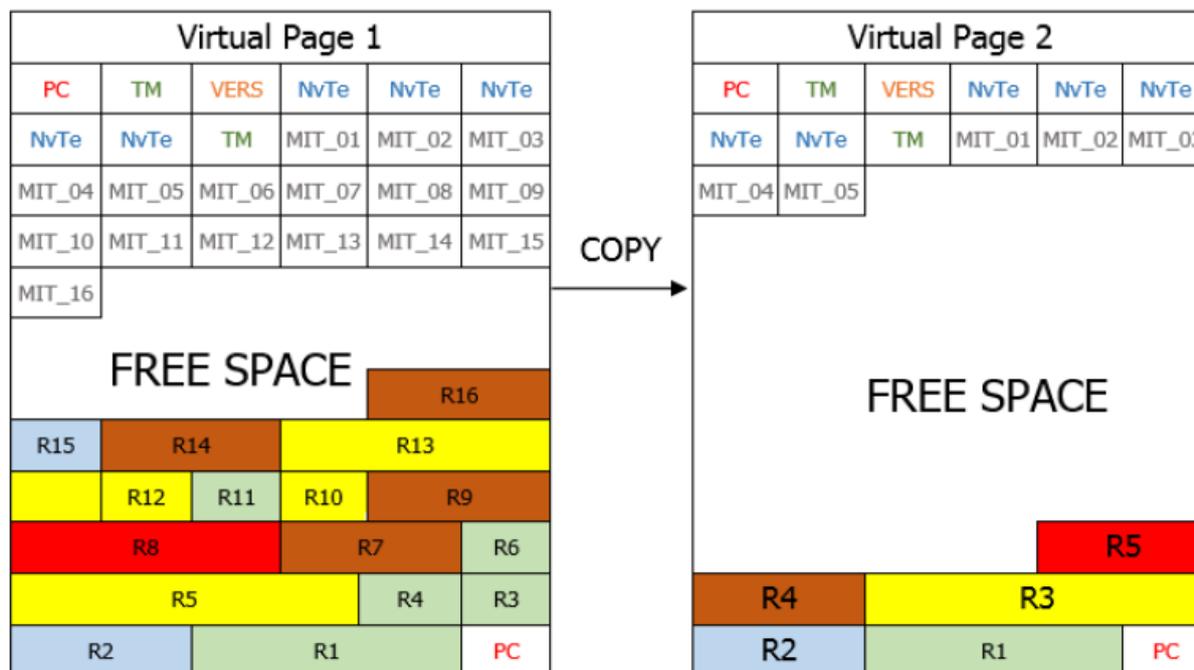


In this example, there are five datasets (one color for each dataset) with both 'full' and 'single' record types.

- 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 bus 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 `NvCopy-Page`, 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 upgradeIt 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 0xFFFFFU.
- *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

### 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 alternatively 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 subsystem. On the NBU code side, a request is emitted via RPSMSG 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 from 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 RPSMSG exchange. On receipt of this request the Host sets a flag notifying of this request from the RPSMSG ISR context. From the Idle thread, this flag is polled via the `RNG_IsReseedNeeded` API. If set the seed is regenerated and forwarded to the NBU via RPSMSG.

`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.

**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.

## 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_e	Mbedtls Seclib_mbec	nccl (part of Seclib.c)	Usage example
AES_128		SecLib_aes.c	x		x		
AES_128_ECB			x		x		
AES_128_CBC		x	x		x		
AES_128_CTR encryption	en-	x	x				
AES_128_OFB encryption	En-	x					
AES_128_CMAC		x	x		x		BLE connection, ieee 15.4
AES_128_EAX		x					
AES_128_CCM		x	x		x		BLE, ieee 15.4
SHA1		SecLib_sha.c	x		x		
SHA256		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	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

## 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 preventing lowpower (only WFI) during its processing. Platform specific code can be find in `fwk_platform_sensors.c/h`.

#### 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 synchronizaton 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 switch 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:  $new\_estimation = (new\_measurement + ((1 \ll n) - 1) * last\_estimation) \gg n$

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 FRO32K. 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 4

## RTOS

### 4.1 FreeRTOS

#### 4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

**FreeRTOS kernel for MCUXpresso SDK Readme**

**FreeRTOS kernel for MCUXpresso SDK**

**Overview** The purpose of this document is to describes the [FreeRTOS kernel repo](#) integration into the [NXP MCUXpresso Software Development Kit: mcuxsdk](#). MCUXpresso SDK provides a comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on MCUs from NXP. This project involves the FreeRTOS kernel repo fork with:

- cmake and Kconfig support to allow the configuration and build in MCUXpresso SDK ecosystem
- FreeRTOS OS additions, such as [FreeRTOS driver wrappers](#), RTOS ready FatFs file system, and the implementation of FreeRTOS tickless mode

The history of changes in FreeRTOS kernel repo for MCUXpresso SDK are summarized in [CHANGELOG\\_mcuxsdk.md](#) file.

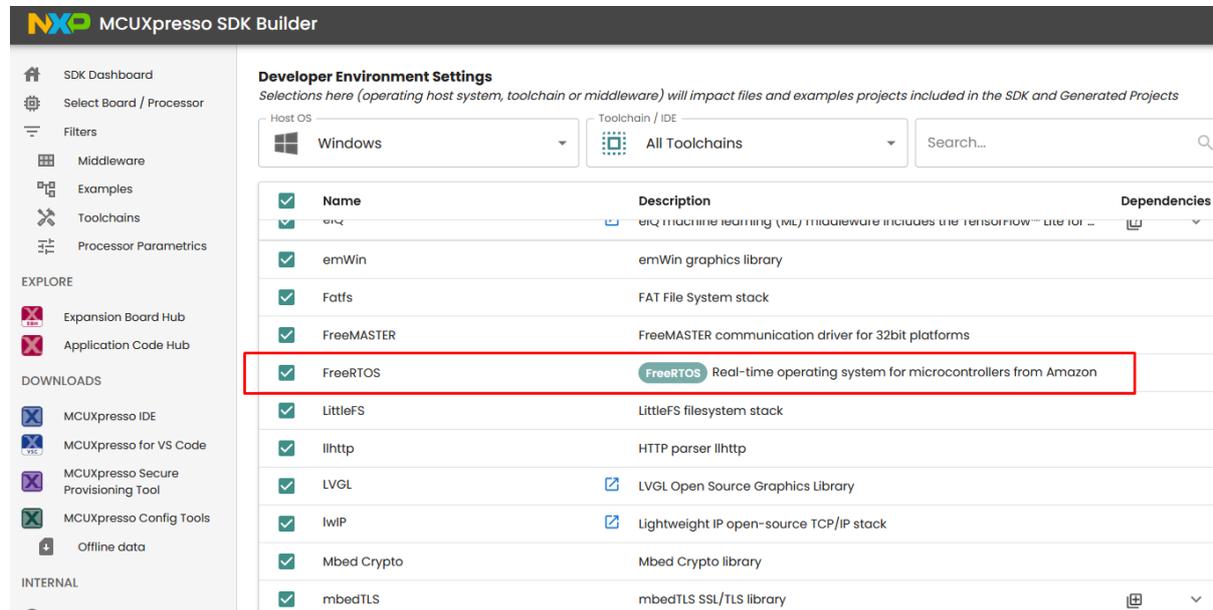
The MCUXpresso SDK framework also contains a set of FreeRTOS examples which show basic FreeRTOS OS features. This makes it easy to start a new FreeRTOS project or begin experimenting with FreeRTOS OS. Selected drivers and middleware are RTOS ready with related FreeRTOS adaptation layer.

**FreeRTOS example applications** The FreeRTOS examples are written to demonstrate basic FreeRTOS features and the interaction between peripheral drivers and the RTOS.

**List of examples** The list of `freertos_examples`, their description and availability for individual supported MCUXpresso SDK development boards can be obtained here: [https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos\\_examples/index.html](https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos_examples/index.html)

**Location of examples** The FreeRTOS examples are located in [mcuxsdk-examples](#) repository, see the `freertos_examples` folder.

Once using MCUXpresso SDK zip packages created via the [MCUXpresso SDK Builder](#) the FreeRTOS kernel library and associated `freertos_examples` are added into final zip package once FreeRTOS components is selected on the Developer Environment Settings page:



The FreeRTOS examples in MCUXpresso SDK zip packages are located in `<MCUXpressoSDK_install_dir>/boards/<board_name>/freertos_examples/` subfolders.

**Building a FreeRTOS example application** For information how to use the `cmake` and `Kconfig` based build and configuration system and how to build `freertos_examples` visit: [MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide](#)

Tip: To list all FreeRTOS example projects and targets that can be built via the `west` build command, use this `west list_project` command in `mcuxsdk` workspace:

```
west list_project -p examples/freertos_examples
```

**FreeRTOS aware debugger plugin** NXP provides FreeRTOS task aware debugger for GDB. The plugin is compatible with Eclipse-based (MCUXpressoIDE) and is available after the installation.

TCB#	Task Name	Task Handle	Task State	Priority	Stack Usage	Event Object	Runtime
1	task_one	0x1ffffcc8	Blocked	1 (1)	0 B / 880 B	MyCountingSemaphore (Rx)	0x0 (0.0%)
2	task_two	0x1ffff130	Blocked	2 (2)	0 B / 888 B	MyCountingSemaphore (Rx)	0x1 (0.1%)
3	IDLE	0x1ffff330	Running	0 (0)	0 B / 296 B		0x3e5 (99.6%)
4	Tmr Svc	0x1ffff6b8	Blocked	17 (17)	28 B / 672 B	TmrQ (Rx)	0x3 (0.3%)

### FreeRTOS kernel for MCUXpresso SDK ChangeLog

**Changelog FreeRTOS kernel for MCUXpresso SDK** All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

**[Unreleased]****Added**

- Kconfig added CONFIG\_FREERTOS\_USE\_CUSTOM\_CONFIG\_FRAGMENT config to optionally include custom FreeRTOSConfig fragment include file FreeRTOSConfig\_frag.h. File must be provided by application.
- Added missing Kconfig option for configUSE\_PICOLIBC\_TLS.
- Add correct header files to build when configUSE\_NEWLIB\_REENTRANT and configUSE\_PICOLIBC\_TLS is selected in config.

**[11.1.0\_rev0]**

- update amazon freertos version

**[11.0.1\_rev0]**

- update amazon freertos version

**[10.5.1\_rev0]**

- update amazon freertos version

**[10.4.3\_rev1]**

- Apply CM33 security fix from 10.4.3-LTS-Patch-2. See rtos\freertos\freertos\_kernel\History.txt
- Apply CM33 security fix from 10.4.3-LTS-Patch-1. See rtos\freertos\freertos\_kernel\History.txt

**[10.4.3\_rev0]**

- update amazon freertos version.

**[10.4.3\_rev0]**

- update amazon freertos version.

**[9.0.0\_rev3]**

- New features:
  - Tickless idle mode support for Cortex-A7. Add fsl\_tickless\_epit.c and fsl\_tickless\_generic.h in portable/IAR/ARM\_CA9 folder.
  - Enabled float context saving in IAR for Cortex-A7. Added configUSE\_TASK\_FPU\_SUPPORT macros. Modified port.c and portmacro.h in portable/IAR/ARM\_CA9 folder.
- Other changes:
  - Transformed ARM\_CM core specific tickless low power support into generic form under freertos/Source/portable/low\_power\_tickless/.

### [9.0.0\_rev2]

- New features:
  - Enabled MCUXpresso thread aware debugging. Add `freertos_tasks_c_additions.h` and `configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H` and `configFREERTOS_MEMORY_SCHEME` macros.

### [9.0.0\_rev1]

- New features:
  - Enabled `-flto` optimization in GCC by adding `attribute((used))` for `vTaskSwitchContext`.
  - Enabled KDS Task Aware Debugger. Apply FreeRTOS patch to enable `configRECORD_STACK_HIGH_ADDRESS` macro. Modified files are `task.c` and `FreeRTOS.h`.

### [9.0.0\_rev0]

- New features:
  - Example `freertos_sem_static`.
  - Static allocation support RTOS driver wrappers.
- Other changes:
  - Tickless idle rework. Support for different timers is in separated files (`fsl_tickless_systick.c`, `fsl_tickless_lptmr.c`).
  - Removed configuration option `configSYSTICK_USE_LOW_POWER_TIMER`. Low power timer is now selected by linking of appropriate file `fsl_tickless_lptmr.c`.
  - Removed `configOVERRIDE_DEFAULT_TICK_CONFIGURATION` in RVDS port. Use of `attribute((weak))` is the preferred solution. Not same as `_weak!`

### [8.2.3]

- New features:
  - Tickless idle mode support.
  - Added template application for Kinetis Expert (KEx) tool (`template_application`).
- Other changes:
  - Folder structure reduction. Keep only Kinetis related parts.

## FreeRTOS kernel Readme

**MCUXpresso SDK: FreeRTOS kernel** This repository is a fork of FreeRTOS kernel (<https://github.com/FreeRTOS/FreeRTOS-Kernel>)(11.1.0). Modifications have been made to adapt to NXP MCUXpresso SDK. `CMakeLists.txt` and `Kconfig` added to enable FreeRTOS kernel repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository `mcuxsdk-manifests`(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

For more information about the FreeRTOS kernel repo adoption see [README\\_mcuxsdk.md: FreeRTOS kernel for MCUXpresso SDK Readme](#) document.



**Getting started** This repository contains FreeRTOS kernel source/header files and kernel ports only. This repository is referenced as a submodule in [FreeRTOS/FreeRTOS](#) repository, which contains pre-configured demo application projects under [FreeRTOS/Demo](#) directory.

The easiest way to use FreeRTOS is to start with one of the pre-configured demo application projects. That way you will have the correct FreeRTOS source files included, and the correct include paths configured. Once a demo application is building and executing you can remove the demo application files, and start to add in your own application source files. See the [FreeRTOS Kernel Quick Start Guide](#) for detailed instructions and other useful links.

Additionally, for FreeRTOS kernel feature information refer to the [Developer Documentation](#), and [API Reference](#).

Also for contributing and creating a Pull Request please refer to *the instructions here*.

**Getting help** If you have any questions or need assistance troubleshooting your FreeRTOS project, we have an active community that can help on the [FreeRTOS Community Support Forum](#).

## To consume FreeRTOS-Kernel

**Consume with CMake** If using CMake, it is recommended to use this repository using FetchContent. Add the following into your project's main or a subdirectory's CMakeLists.txt:

- Define the source and version/tag you want to use:

```
FetchContent_Declare(freertos_kernel
 GIT_REPOSITORY https://github.com/FreeRTOS/FreeRTOS-Kernel.git
 GIT_TAG main #Note: Best practice to use specific git-hash or tagged version
)
```

In case you prefer to add it as a git submodule, do:

```
git submodule add https://github.com/FreeRTOS/FreeRTOS-Kernel.git <path of the submodule>
git submodule update --init
```

- Add a freertos\_config library (typically an INTERFACE library) The following assumes the directory structure:

– include/FreeRTOSConfig.h

```
add_library(freertos_config INTERFACE)

target_include_directories(freertos_config SYSTEM
INTERFACE
 include
)

target_compile_definitions(freertos_config
INTERFACE
 projCOVERAGE_TEST=0
)
```

In case you installed FreeRTOS-Kernel as a submodule, you will have to add it as a subdirectory:

```
add_subdirectory(${FREERTOS_PATH})
```

- Configure the FreeRTOS-Kernel and make it available
  - this particular example supports a native and cross-compiled build option.

```
set(FREERTOS_HEAP "4" CACHE STRING "" FORCE)
Select the native compile PORT
set(FREERTOS_PORT "GCC_POSIX" CACHE STRING "" FORCE)
Select the cross-compile PORT
if (CMAKE_CROSSCOMPILING)
 set(FREERTOS_PORT "GCC_ARM_CA9" CACHE STRING "" FORCE)
endif()

FetchContent_MakeAvailable(freertos_kernel)
```

- In case of cross compilation, you should also add the following to `freertos_config`:

```
target_compile_definitions(freertos_config INTERFACE ${definitions})
target_compile_options(freertos_config INTERFACE ${options})
```

### Consuming stand-alone - Cloning this repository

To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Kernel.git
```

#### Using SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Kernel.git
```

### Repository structure

- The root of this repository contains the three files that are common to every port - `list.c`, `queue.c` and `tasks.c`. The kernel is contained within these three files. `croutine.c` implements the optional co-routine functionality - which is normally only used on very memory limited systems.
- The `./portable` directory contains the files that are specific to a particular microcontroller and/or compiler. See the readme file in the `./portable` directory for more information.
- The `./include` directory contains the real time kernel header files.
- The `./template_configuration` directory contains a sample `FreeRTOSConfig.h` to help jumpstart a new project. See the `FreeRTOSConfig.h` file for instructions.

**Code Formatting** FreeRTOS files are formatted using the “`uncrustify`” tool. The configuration file used by `uncrustify` can be found in the `FreeRTOS/CI-CD-GitHub-Actions`’s `uncrustify.cfg` file.

**Line Endings** File checked into the `FreeRTOS-Kernel` repository use unix-style LF line endings for the best compatibility with `git`.

For optimal compatibility with Microsoft Windows tools, it is best to enable the `git autocrlf` feature. You can enable this setting for the current repository using the following command:

```
git config core.autocrlf true
```

**Git History Optimizations** Some commits in this repository perform large refactors which touch many lines and lead to unwanted behavior when using the `git blame` command. You can configure `git` to ignore the list of large refactor commits in this repository with the following command:

```
git config blame.ignoreRevsFile .git-blame-ignore-revs
```

**Spelling and Formatting** We recommend using [Visual Studio Code](#), commonly referred to as VSCode, when working on the FreeRTOS-Kernel. The FreeRTOS-Kernel also uses [cSpell](#) as part of its spelling check. The config file for which can be found at [cspell.config.yaml](#) There is additionally a [cSpell plugin for VSCode](#) that can be used as well. `.cSpellWords.txt` contains words that are not traditionally found in an English dictionary. It is used by the spellchecker to verify the various jargon, variable names, and other odd words used in the FreeRTOS code base are correct. If your pull request fails to pass the spelling and you believe this is a mistake, then add the word to `.cSpellWords.txt`. When adding a word please then sort the list, which can be done by running the bash command: `sort -u .cSpellWords.txt -o .cSpellWords.txt` Note that only the FreeRTOS-Kernel Source Files, *include*, *portable/MemMang*, and *portable/Common* files are checked for proper spelling, and formatting at this time.

### 4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

### 4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

#### Readme

**MCUXpresso SDK: backoffAlgorithm Library** This repository is a fork of backoffAlgorithm library (<https://github.com/FreeRTOS/backoffalgorithm>)(1.3.0). Modifications have been made to adapt to NXP MCUXpresso SDK. `CMakeLists.txt` and `Kconfig` added to enable backoffAlgorithm repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository `mcuxsdk-manifests`(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**backoffAlgorithm Library** This repository contains the backoffAlgorithm library, a utility library to calculate backoff period using an exponential backoff with jitter algorithm for retrying network operations (like failed network connection with server). This library uses the “Full Jitter” strategy for the exponential backoff with jitter algorithm. More information about the algorithm can be seen in the [Exponential Backoff and Jitter](#) AWS blog.

The backoffAlgorithm library is distributed under the *MIT Open Source License*.

Exponential backoff with jitter is typically used when retrying a failed network connection or operation request with the server. An exponential backoff with jitter helps to mitigate failed network operations with servers, that are caused due to network congestion or high request load on the server, by spreading out retry requests across multiple devices attempting network operations. Besides, in an environment with poor connectivity, a client can get disconnected at any time. A backoff strategy helps the client to conserve battery by not repeatedly attempting reconnections when they are unlikely to succeed.

See memory requirements for this library [here](#).

**backoffAlgorithm v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.**

**backoffAlgorithm v1.0.0 source code is part of the FreeRTOS 202012.00 LTS release.**

**Reference example** The example below shows how to use the backoffAlgorithm library on a POSIX platform to retry a DNS resolution query for amazon.com.

```
#include "backoff_algorithm.h"
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <unistd.h>
#include <time.h>

/* The maximum number of retries for the example code. */
#define RETRY_MAX_ATTEMPTS (5U)

/* The maximum back-off delay (in milliseconds) for between retries in the example. */
#define RETRY_MAX_BACKOFF_DELAY_MS (5000U)

/* The base back-off delay (in milliseconds) for retry configuration in the example. */
#define RETRY_BACKOFF_BASE_MS (500U)

int main()
{
 /* Variables used in this example. */
 BackoffAlgorithmStatus_t retryStatus = BackoffAlgorithmSuccess;
 BackoffAlgorithmContext_t retryParams;
 char serverAddress[] = "amazon.com";
 uint16_t nextRetryBackoff = 0;

 int32_t dnsStatus = -1;
 struct addrinfo hints;
 struct addrinfo ** pListHead = NULL;
 struct timespec tp;

 /* Add hints to retrieve only TCP sockets in getaddrinfo. */
 (void) memset(&hints, 0, sizeof(hints));

 /* Address family of either IPv4 or IPv6. */
 hints.ai_family = AF_UNSPEC;
 /* TCP Socket. */
 hints.ai_socktype = (int32_t) SOCK_STREAM;
 hints.ai_protocol = IPPROTO_TCP;

 /* Initialize reconnect attempts and interval. */
 BackoffAlgorithm_InitializeParams(&retryParams,
 RETRY_BACKOFF_BASE_MS,
 RETRY_MAX_BACKOFF_DELAY_MS,
 RETRY_MAX_ATTEMPTS);

 /* Seed the pseudo random number generator used in this example (with call to
 * rand() function provided by ISO C standard library) for use in backoff period
 * calculation when retrying failed DNS resolution. */

 /* Get current time to seed pseudo random number generator. */
 (void) clock_gettime(CLOCK_REALTIME, &tp);
 /* Seed pseudo random number generator with seconds. */
 srand(tp.tv_sec);

 do
 {
 /* Perform a DNS lookup on the given host name. */
 dnsStatus = getaddrinfo(serverAddress, NULL, &hints, pListHead);
 }
}
```

(continues on next page)

(continued from previous page)

```

/* Retry if DNS resolution query failed. */
if(dnsStatus != 0)
{
 /* Generate a random number and get back-off value (in milliseconds) for the next retry.
 * Note: It is recommended to use a random number generator that is seeded with
 * device-specific entropy source so that backoff calculation across devices is different
 * and possibility of network collision between devices attempting retries can be avoided.
 *
 * For the simplicity of this code example, the pseudo random number generator, rand()
 * function is used. */
 retryStatus = BackoffAlgorithm_GetNextBackoff(&retryParams, rand(), &nextRetryBackoff);

 /* Wait for the calculated backoff period before the next retry attempt of querying DNS.
 * As usleep() takes nanoseconds as the parameter, we multiply the backoff period by 1000. */
 (void) usleep(nextRetryBackoff * 1000U);
}
} while((dnsStatus != 0) && (retryStatus != BackoffAlgorithmRetriesExhausted));

return dnsStatus;
}

```

**Building the library** A compiler that supports **C90 or later** such as *gcc* is required to build the library.

Additionally, the library uses a header file introduced in ISO C99, *stdint.h*. For compilers that do not provide this header file, the *source/include* directory contains *stdint.readme*, which can be renamed to *stdint.h* to build the *backoffAlgorithm* library.

For instance, if the example above is copied to a file named *example.c*, *gcc* can be used like so:

```
gcc -I source/include example.c source/backoff_algorithm.c -o example
./example
```

*gcc* can also produce an output file to be linked:

```
gcc -I source/include -c source/backoff_algorithm.c
```

## Building unit tests

**Checkout Unity Submodule** By default, the submodules in this repository are configured with `update=none` in *.gitmodules*, to avoid increasing clone time and disk space usage of other repositories (like [amazon-freertos](#) that submodules this repository).

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/Unity
```

## Platform Prerequisites

- For running unit tests
  - C89 or later compiler like *gcc*
  - CMake 3.13.0 or later
- For running the coverage target, *gcov* is additionally required.

### Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described [above](#).)
2. Create build directory: `mkdir build && cd build`
3. Run `cmake` while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

**Contributing** See *CONTRIBUTING.md* for information on contributing.

## 4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

### MCUXpresso SDK: coreHTTP Client Library

This repository is a fork of coreHTTP Client library (<https://github.com/FreeRTOS/corehttp>)(3.0.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreHTTP Client repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

### coreHTTP Client Library

This repository contains a C language HTTP client library designed for embedded platforms. It has no dependencies on any additional libraries other than the standard C library, [llhttp](#), and a customer-implemented transport interface. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8. This library has also undergone both static code analysis from [Coverity static analysis](#), and validation of memory safety and data structure invariance through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**coreHTTP v3.0.0 source code is part of the FreeRTOS 202210.00 LTS release.**

**coreHTTP v2.0.0 source code is part of the FreeRTOS 202012.00 LTS release.**

**coreHTTP Config File** The HTTP client library exposes configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *core\_http\_config\_defaults.h*. To provide custom values for the configuration macros, a custom config file named *core\_http\_config.h* can be provided by the user application to the library.

By default, a *core\_http\_config.h* custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide `HTTP_DO_NOT_USE_CUSTOM_CONFIG` as a compile time preprocessor macro.

**The HTTP client library can be built by either:**

- Defining a `core_http_config.h` file in the application, and adding it to the include directories for the library build. **OR**
- Defining the `HTTP_DO_NOT_USE_CUSTOM_CONFIG` preprocessor macro for the library build.

**Building the Library** The `httpFilePaths.cmake` file contains the information of all source files and header include paths required to build the HTTP client library.

As mentioned in the *previous section*, either a custom config file (i.e. `core_http_config.h`) OR `HTTP_DO_NOT_USE_CUSTOM_CONFIG` macro needs to be provided to build the HTTP client library.

For a CMake example of building the HTTP library with the `httpFilePaths.cmake` file, refer to the `coverity_analysis` library target in `test/CMakeLists.txt` file.

## Building Unit Tests

### Platform Prerequisites

- For running unit tests, the following are required:
  - **C90 compiler** like `gcc`
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is required for this repository's [CMock test framework](#).
- For running the coverage target, the following are required:
  - **gcov**
  - **lcov**

### Steps to build Unit Tests

1. Go to the root directory of this repository.
2. Run the `cmake` command: `cmake -S test -B build -DBUILD_CLONE_SUBMODULES=ON`
3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Reference examples** The AWS IoT Device SDK for Embedded C repository contains demos of using the HTTP client library [here](#) on a POSIX platform. These can be used as reference examples for the library API.

## Documentation

**Existing Documentation** For pre-generated documentation, please see the documentation linked in the locations below:

Location
<a href="https://www.FreeRTOS.org">AWS IoT Device SDK for Embedded C FreeRTOS.org</a>

Note that the latest included version of coreHTTP may differ across repositories.

**Generating Documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

**Contributing** See *CONTRIBUTING.md* for information on contributing.

## 4.1.5 corejson

JSON parser.

### Readme

**MCUXpresso SDK: coreJSON Library** This repository is a fork of coreJSON library (<https://github.com/FreeRTOS/corejson>)(3.2.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreJSON repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**coreJSON Library** This repository contains the coreJSON library, a parser that strictly enforces the ECMA-404 JSON standard and is suitable for low memory footprint embedded devices. The coreJSON library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8, and checks against deviations from mandatory rules in the [MISRA coding standard](#). Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from [Coverity static analysis](#), and validation of memory safety through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**coreJSON v3.2.0 source code is part of the [FreeRTOS 202210.00 LTS release](#).**

**coreJSON v3.0.0 source code is part of the [FreeRTOS 202012.00 LTS release](#).**

### Reference example

```

#include <stdio.h>
#include "core_json.h"

int main()
{
 // Variables used in this example.
 JSONStatus_t result;
 char buffer[] = "{\"foo\": \"abc\", \"bar\": {\"foo\": \"xyz\"}}";
 size_t bufferLength = sizeof(buffer) - 1;
 char queryKey[] = "bar.foo";
 size_t queryKeyLength = sizeof(queryKey) - 1;
 char * value;
 size_t valueLength;

 // Calling JSON_Validate() is not necessary if the document is guaranteed to be valid.
 result = JSON_Validate(buffer, bufferLength);

 if(result == JSONSuccess)
 {
 result = JSON_Search(buffer, bufferLength, queryKey, queryKeyLength,
 &value, &valueLength);
 }

 if(result == JSONSuccess)
 {
 // The pointer "value" will point to a location in the "buffer".
 char save = value[valueLength];
 // After saving the character, set it to a null byte for printing.
 value[valueLength] = '\\0';
 // "Found: bar.foo -> xyz" will be printed.
 printf("Found: %s -> %s\\n", queryKey, value);
 // Restore the original character.
 value[valueLength] = save;
 }

 return 0;
}

```

A search may descend through nested objects when the `queryKey` contains matching key strings joined by a separator, .. In the example above, `bar` has the value `{"foo": "xyz"}`. Therefore, a search for query key `bar.foo` would output `xyz`.

**Building coreJSON** A compiler that supports **C90 or later** such as `gcc` is required to build the library.

Additionally, the library uses 2 header files introduced in ISO C99, `stdbool.h` and `stdint.h`. For compilers that do not provide this header file, the *source/include* directory contains *stdbool.readme* and *stdint.readme*, which can be renamed to `stdbool.h` and `stdint.h` respectively.

For instance, if the example above is copied to a file named `example.c`, `gcc` can be used like so:

```
gcc -I source/include example.c source/core_json.c -o example
./example
```

`gcc` can also produce an output file to be linked:

```
gcc -I source/include -c source/core_json.c
```

## Documentation

**Existing documentation** For pre-generated documentation, please see the documentation linked in the locations below:

Location
<a href="#">AWS IoT Device SDK for Embedded C FreeRTOS.org</a>

Note that the latest included version of the coreJSON library may differ across repositories.

**Generating documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

### Building unit tests

**Checkout Unity Submodule** By default, the submodules in this repository are configured with `update=none` in `.gitmodules`, to avoid increasing clone time and disk space usage of other repositories (like [amazon-freertos](#) that submodules this repository).

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/Unity
```

### Platform Prerequisites

- For running unit tests
  - C90 compiler like gcc
  - CMake 3.13.0 or later
  - Ruby 2.0.0 or later is additionally required for the Unity test framework (that we use).
- For running the coverage target, gcov is additionally required.

### Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described [above](#).)
2. Create build directory: `mkdir build && cd build`
3. Run `cmake` while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Contributing** See *CONTRIBUTING.md* for information on contributing.

#### 4.1.6 coremqtt

MQTT publish/subscribe messaging library.

##### MCUXpresso SDK: coreMQTT Library

This repository is a fork of coreMQTT library (<https://github.com/FreeRTOS/coremqtt>)(2.1.1). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreMQTT repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

##### coreMQTT Client Library

This repository contains the coreMQTT library that has been optimized for a low memory footprint. The coreMQTT library is compliant with the [MQTT 3.1.1](#) standard. It has no dependencies on any additional libraries other than the standard C library, a customer-implemented network transport interface, and *optionally* a user-implemented platform time function. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8, and checks against deviations from mandatory rules in the [MISRA coding standard](#). Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from [Coverity static analysis](#), and validation of memory safety through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**coreMQTT v2.1.1 source code is part of the FreeRTOS 202210.01 LTS release.**

**MQTT Config File** The MQTT client library exposes build configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *core\_mqtt\_config\_defaults.h*. To provide custom values for the configuration macros, a custom config file named *core\_mqtt\_config.h* can be provided by the application to the library.

By default, a *core\_mqtt\_config.h* custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide `MQTT_DO_NOT_USE_CUSTOM_CONFIG` as a compile time preprocessor macro.

**Thus, the MQTT library can be built by either:**

- Defining a *core\_mqtt\_config.h* file in the application, and adding it to the include directories list of the library
- OR**
- Defining the `MQTT_DO_NOT_USE_CUSTOM_CONFIG` preprocessor macro for the library build.

**Sending metrics to AWS IoT** When establishing a connection with AWS IoT, users can optionally report the Operating System, Hardware Platform and MQTT client version information of their device to AWS. This information can help AWS IoT provide faster issue resolution and technical support. If users want to report this information, they can send a specially formatted string (see below) in the username field of the MQTT CONNECT packet.

#### Format

The format of the username string with metrics is:

```
<Actual_Username>?SDK=<OS_Name>&Version=<OS_Version>&Platform=<Hardware_Platform>&MQTTLib=<MQTT_Library_name>@<MQTT_Library_version>
```

#### Where

- <Actual\_Username> is the actual username used for authentication, if username and password are used for authentication. When username and password based authentication is not used, this is an empty value.
- <OS\_Name> is the Operating System the application is running on (e.g. FreeRTOS)
- <OS\_Version> is the version number of the Operating System (e.g. V10.4.3)
- <Hardware\_Platform> is the Hardware Platform the application is running on (e.g. WinSim)
- <MQTT\_Library\_name> is the MQTT Client library being used (e.g. coreMQTT)
- <MQTT\_Library\_version> is the version of the MQTT Client library being used (e.g. 1.0.2)

#### Example

- Actual\_Username = "iotuser", OS\_Name = FreeRTOS, OS\_Version = V10.4.3, Hardware\_Platform\_Name = WinSim, MQTT\_Library\_Name = coremqtt, MQTT\_Library\_version = 2.1.1. If username is not used, then "iotuser" can be removed.

```
/* Username string:
 * iotuser?SDK=FreeRTOS&Version=v10.4.3&Platform=WinSim&MQTTLib=coremqtt@2.1.1
 */

#define OS_NAME "FreeRTOS"
#define OS_VERSION "V10.4.3"
#define HARDWARE_PLATFORM_NAME "WinSim"
#define MQTT_LIB "coremqtt@2.1.1"

#define USERNAME_STRING "iotuser?SDK=" OS_NAME "&Version=" OS_VERSION "&Platform=" HARDWARE_PLATFORM_NAME "&MQTTLib=" MQTT_LIB
#define USERNAME_STRING_LENGTH ((uint16_t) (sizeof(USERNAME_STRING) - 1))

MQTTConnectInfo_t connectInfo;
connectInfo.pUserName = USERNAME_STRING;
connectInfo.userNameLength = USERNAME_STRING_LENGTH;
mqttStatus = MQTT_Connect(pMqttContext, &connectInfo, NULL, CONNACK_RECV_TIMEOUT_MS,
↳ pSessionPresent);
```

**Upgrading to v2.0.0 and above** With coreMQTT versions >=v2.0.0, there are breaking changes. Please refer to the *coreMQTT version >=v2.0.0 Migration Guide*.

**Building the Library** The *mqttFilePaths.cmake* file contains the information of all source files and the header include path required to build the MQTT library.

Additionally, the MQTT library requires two header files that are not part of the ISO C90 standard library, *stdbool.h* and *stdint.h*. For compilers that do not provide these header files, the

*source/include* directory contains the files *stdbool.readme* and *stdint.readme*, which can be renamed to *stdbool.h* and *stdint.h*, respectively, to provide the type definitions required by MQTT.

As mentioned in the previous section, either a custom config file (i.e. *core\_mqtt\_config.h*) OR `MQTT_DO_NOT_USE_CUSTOM_CONFIG` macro needs to be provided to build the MQTT library.

For a CMake example of building the MQTT library with the *mqttFilePaths.cmake* file, refer to the *coverity\_analysis* library target in *test/CMakeLists.txt* file.

## Building Unit Tests

**Checkout CMock Submodule** By default, the submodules in this repository are configured with `update=none` in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories (like [amazon-freertos](#) that submodules this repository).

To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/CMock
```

## Platform Prerequisites

- Docker

or the following:

- For running unit tests
  - **C90 compiler** like `gcc`
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

## Steps to build Unit Tests

1. If using docker, launch the container:
  1. `docker build -t coremqtt .`
  2. `docker run -it -v "$PWD":/workspaces/coreMQTT -w /workspaces/coreMQTT coremqtt`
2. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described [above](#))
3. Run the *cmake* command: `cmake -S test -B build`
4. Run this command to build the library and unit tests: `make -C build all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `cd build && ctest` to execute all tests and view the test run summary.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Reference examples** Please refer to the demos of the MQTT client library in the following locations for reference examples on POSIX and FreeRTOS platforms:

Platform	Location	Transport Interface Implementation
POSIX	<a href="#">AWS IoT Device SDK for Embedded C</a>	POSIX sockets for TCP/IP and OpenSSL for TLS stack
FreeRTOS	<a href="#">FreeRTOS/FreeRTOS</a>	FreeRTOS+TCP for TCP/IP and mbedTLS for TLS stack
FreeRTOS	<a href="#">FreeRTOS AWS Reference Integrations</a>	Based on Secure Sockets Abstraction

## Documentation

**Existing Documentation** For pre-generated documentation, please see the documentation linked in the locations below:

Location
<a href="#">AWS IoT Device SDK for Embedded C</a>
<a href="#">FreeRTOS.org</a>

Note that the latest included version of coreMQTT may differ across repositories.

**Generating Documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

**Contributing** See *CONTRIBUTING.md* for information on contributing.

### 4.1.7 corepkcs11

PKCS #11 key management library.

#### Readme

**MCUXpresso SDK: corePKCS11 Library** This repository is a fork of PKCS #11 key management library (<https://github.com/FreeRTOS/corePKCS11/tree/v3.5.0>)(v3.5.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable corepkcs11 repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**corePKCS11 Library** PKCS #11 is a standardized and widely used API for manipulating common cryptographic objects. It is important because the functions it specifies allow application software to use, create, modify, and delete cryptographic objects, without ever exposing those objects to the application's memory. For example, FreeRTOS AWS reference integrations use a small subset of the PKCS #11 API to, among other things, access the secret (private) key necessary to create a network connection that is authenticated and secured by the [Transport Layer Security \(TLS\)](#) protocol – without the application ever 'seeing' the key.

The Cryptoki or PKCS #11 standard defines a platform-independent API to manage and use cryptographic tokens. The name, "PKCS #11", is used interchangeably to refer to the API itself and the standard which defines it.

This repository contains a software based mock implementation of the PKCS #11 interface (API) that uses the cryptographic functionality provided by Mbed TLS. Using a software mock enables rapid development and flexibility, but it is expected that the mock be replaced by an implementation specific to your chosen secure key storage in production devices.

Only a subset of the PKCS #11 standard is implemented, with a focus on operations involving asymmetric keys, random number generation, and hashing.

The targeted use cases include certificate and key management for TLS authentication and code-sign signature verification, on small embedded devices.

corePKCS11 is implemented on PKCS #11 v2.4.0, the full PKCS #11 standard can be found on the [oasis website](#).

This library has gone through code quality checks including verification that no function has a [GNU Complexity](#) score over 8, and checks against deviations from mandatory rules in the [MISRA coding standard](#). Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from [Coverity static analysis](#) and validation of memory safety through the [CBMC automated reasoning tool](#).

See memory requirements for this library [here](#).

**corePKCS11 v3.5.0 source code is part of the FreeRTOS 202210.00 LTS release.**

**corePKCS11 v3.0.0 source code is part of the FreeRTOS 202012.00 LTS release.**

**Purpose** Generally vendors for secure cryptoprocessors such as Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element, or any other type of secure hardware enclave, distribute a PKCS #11 implementation with the hardware. The purpose of the corePKCS11 software only mock library is therefore to provide a non hardware specific PKCS #11 implementation that allows for rapid prototyping and development before switching to a cryptoprocessor specific PKCS #11 implementation in production devices.

Since the PKCS #11 interface is defined as part of the PKCS #11 [specification](#) replacing this library with another implementation should require little porting effort, as the interface will not change. The system tests distributed in this repository can be leveraged to verify the behavior of a different implementation is similar to corePKCS11.

**corePKCS11 Configuration** The corePKCS11 library exposes preprocessor macros which must be defined prior to building the library. A list of all the configurations and their default values are defined in the doxygen documentation for this library.

## Build Prerequisites

**Library Usage** For building the library the following are required:

- A C99 compiler

- **mbedcrypto** library from [mbedtls](#) version 2.x or 3.x.
- **pkcs11 API header(s)** available from [OASIS](#) or [OpenSC](#)

Optionally, variables from the `pkcsFilePaths.cmake` file may be referenced if your project uses `cmake`.

**Integration and Unit Tests** In order to run the integration and unit test suites the following are dependencies are necessary:

- **C Compiler**
- **CMake 3.13.0 or later**
- **Ruby 2.0.0 or later** required by `CMock`.
- **Python 3** required for configuring `mbedtls`.
- **git** required for fetching dependencies.
- **GNU Make** or **Ninja**

The `mbedtls`, `CMock`, and `Unity` libraries are downloaded and built automatically using the `cmake` `FetchContent` feature.

**Coverage Measurement and Instrumentation** The following software is required to run the coverage target:

- Linux, MacOS, or another POSIX-like environment.
- A recent version of **GCC** or **Clang** with support for `gcov`-like coverage instrumentation.
- **gcov** binary corresponding to your chosen compiler
- **lcov** from the [Linux Test Project](#)
- **perl** needed to run the `lcov` utility.

Coverage builds are validated on recent versions of Ubuntu Linux.

### Running the Integration and Unit Tests

1. Navigate to the root directory of this repository in your shell.
2. Run **cmake** to construct a build tree: `cmake -S test -B build`
  - You may specify your preferred build tool by appending `-G'Unix Makefiles'` or `-GNinja` to the command above.
  - You may append `-DUNIT_TESTS=0` or `-DSYSTEM_TESTS=0` to disable Unit Tests or Integration Tests respectively.
3. Build the test binaries: `cmake --build ./build --target all`
4. Run `ctest --test-dir ./build` or `cmake --build ./build --target test` to run the tests without capturing coverage.
5. Run `cmake --build ./build --target coverage` to run the tests and capture coverage data.

**CBMC** To learn more about CBMC and proofs specifically, review the training material [here](#).

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

**Reference examples** The FreeRTOS-Labs repository contains demos using the PKCS #11 library [here](#) using FreeRTOS on the Windows simulator platform. These can be used as reference examples for the library API.

**Porting Guide** Documentation for porting corePKCS11 to a new platform can be found on the [AWS docs](#) web page.

corePKCS11 is not meant to be ported to projects that have a TPM, HSM, or other hardware for offloading crypto-processing. This library is specifically meant to be used for development and prototyping.

**Related Example Implementations** These projects implement the PKCS #11 interface on real hardware and have similar behavior to corePKCS11. It is preferred to use these, over corePKCS11, as they allow for offloading Cryptography to separate hardware.

- [ARM's Platform Security Architecture](#).
- [Microchip's cryptoauthlib](#).
- [Infineon's Optiga Trust X](#).

## Documentation

**Existing Documentation** For pre-generated documentation, please see the documentation linked in the locations below:

Location
<a href="#">AWS IoT Device SDK for Embedded C</a> <a href="#">FreeRTOS.org</a>

Note that the latest included version of corePKCS11 may differ across repositories.

**Generating Documentation** The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

**Security** See *CONTRIBUTING* for more information.

**License** This library is licensed under the MIT-0 License. See the LICENSE file.

### 4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

## Readme

**MCUXpresso SDK: FreeRTOS-Plus-TCP Library** This repository is a fork of FreeRTOS-Plus-TCP library (<https://github.com/FreeRTOS/freertos-plus-tcp>)(4.3.3). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable FreeRTOS-Plus-TCP repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(<https://github.com/nxp-mcuxpresso/mcuxsdk-manifests>) for the complete delivery of MCUXpresso SDK.

**FreeRTOS-Plus-TCP Library** FreeRTOS-Plus-TCP is a lightweight TCP/IP stack for FreeRTOS. It provides a familiar Berkeley sockets interface, making it as simple to use and learn as possible. FreeRTOS-Plus-TCP's features and RAM footprint are fully scalable, making FreeRTOS-Plus-TCP equally applicable to smaller lower throughput microcontrollers as well as larger higher throughput microprocessors.

This library has undergone static code analysis and checks for compliance with the [MISRA coding standard](#). Any deviations from the MISRA C:2012 guidelines are documented under [MISRA Deviations](#). The library is validated for memory safety and data structure invariance through the [CBMC automated reasoning tool](#) for the functions that parse data originating from the network. The library is also protocol tested using Maxwell protocol tester for both IPv4 and IPv6.

**FreeRTOS-Plus-TCP Library V4.2.2 source code is part of the FreeRTOS 202406.01 LTS release.**

**Getting started** The easiest way to use version 4.0.0 and later of FreeRTOS-Plus-TCP is to refer the Getting started Guide (found [here](#)) Another way is to start with the pre-configured IPv4 Windows Simulator demo (found in [this directory](#)) or IPv6 Multi-endpoint Windows Simulator demo (found in [this directory](#)). That way you will have the correct FreeRTOS source files included, and the correct include paths configured. Once a demo application is building and executing you can remove the demo application files, and start to add in your own application source files. See the [FreeRTOS Kernel Quick Start Guide](#) for detailed instructions and other useful links.

Additionally, for FreeRTOS-Plus-TCP source code organization refer to the [Documentation](#), and [API Reference](#).

**Getting help** If you have any questions or need assistance troubleshooting your FreeRTOS project, we have an active community that can help on the [FreeRTOS Community Support Forum](#). Please also refer to [FAQ](#) for frequently asked questions.

Also see the [Submitting a bugs/feature request](#) section of CONTRIBUTING.md for more details.

**Note:** All the remaining sections are generic and applies to all the versions from V3.0.0 onwards.

**Upgrading to V4.3.0 and above** For users of STM32 network interfaces:

Starting from version V4.3.0, the STM32 network interfaces have been consolidated into a single unified implementation located at `source/portable/NetworkInterface/STM32/NetworkInterface.c`, supporting STM32 F4, F7, and H7 series microcontrollers, with newly added support for STM32 H5. The new interface has been tested with the STM32 HAL Ethernet (ETH) drivers, available at `source/portable/NetworkInterface/STM32/Drivers`. For compatibility, the legacy interfaces (STM32Fxx and STM32Hxx) have been retained and relocated to `source/portable/NetworkInterface/STM32/Legacy`.

**Upgrading to V3.0.0 and V3.1.0** In version 3.0.0 or 3.1.0, the folder structure of FreeRTOS-Plus-TCP has changed and the files have been broken down into smaller logically separated modules. This change makes the code more modular and conducive to unit-tests. FreeRTOS-Plus-TCP V3.0.0 improves the robustness, security, and modularity of the library. Version 3.0.0 adds comprehensive unit test coverage for all lines and branches of code and has undergone protocol testing, and penetration testing by AWS Security to reduce the exposure to security vulnerabilities. Additionally, the source files have been moved to a `source` directory. This change requires modification of any existing project(s) to include the modified source files and directories.

**FreeRTOS-Plus-TCP V3.1.0 source code(.c .h) is part of the FreeRTOS 202210.00 LTS release.**

**Generating pre V3.0.0 folder structure for backward compatibility:** If you wish to continue using a version earlier than V3.0.0 i.e. continue to use your existing source code organization, a script is provided to generate the folder structure similar to [this](#).

**Note:** After running the script, while the `.c` files will have same names as the pre V3.0.0 source, the files in the `include` directory will have different names and the number of files will differ as well. This should, however, not pose any problems to most projects as projects generally include all files in a given directory.

Running the script to generate pre V3.0.0 folder structure: For running the script, you will need Python version > 3.7. You can download/install it from [here](#).

Once python is downloaded and installed, you can verify the version from your terminal/command window by typing `python --version`.

To run the script, you should switch to the FreeRTOS-Plus-TCP directory Then run `python <Path/to/the/script>/GenerateOriginalFiles.py`.

## To consume FreeRTOS+TCP

**Consume with CMake** If using CMake, it is recommended to use this repository using FetchContent. Add the following into your project's main or a subdirectory's `CMakeLists.txt`:

- Define the source and version/tag you want to use:

```
FetchContent_Declare(freertos_plus_tcp
 GIT_REPOSITORY https://github.com/FreeRTOS/FreeRTOS-Plus-TCP.git
 GIT_TAG main #Note: Best practice to use specific git-hash or tagged version
 GIT_SUBMODULES "" # Don't grab any submodules since not latest
)
```

- Configure the FreeRTOS-Kernel and make it available
  - this particular example supports a native and cross-compiled build option.

```
Select the native compile PORT
set(FREERTOS_PLUS_TCP_NETWORK_IF "POSIX" CACHE STRING "" FORCE)
Or: select a cross-compile PORT
if (CMAKE_CROSSCOMPILING)
 # Eg. STM32Hxx version of port
 set(FREERTOS_PLUS_TCP_NETWORK_IF "STM32HXX" CACHE STRING "" FORCE)
endif()

FetchContent_MakeAvailable(freertos_plus_tcp)
```

**Consuming stand-alone** This repository uses Git Submodules to bring in dependent components.

Note: If you download the ZIP file provided by GitHub UI, you will not get the contents of the submodules. (The ZIP file is also not a valid Git repository)

To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Plus-TCP.git ./FreeRTOS-Plus-TCP
cd ./FreeRTOS-Plus-TCP
git submodule update --checkout --init --recursive tools/CMock test/FreeRTOS-Kernel
```

Using SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Plus-TCP.git ./FreeRTOS-Plus-TCP
cd ./FreeRTOS-Plus-TCP
git submodule update --checkout --init --recursive tools/CMock test/FreeRTOS-Kernel
```

**Porting** The porting guide is available on [this page](#).

**Repository structure** This repository contains the FreeRTOS-Plus-TCP repository and a number of supplementary libraries for testing/PR Checks. Below is the breakdown of what each directory contains:

- tools
  - This directory contains the tools and related files (CMock/uncrustify) required to run tests/checks on the TCP source code.
- tests
  - This directory contains all the tests (unit tests and CBMC) and the dependencies (FreeRTOS-Kernel/Litani-port) the tests require.
- source/portable
  - This directory contains the portable files required to compile the FreeRTOS-Plus-TCP source code for different hardware/compilers.
- source/include
  - The include directory has all the ‘core’ header files of FreeRTOS-Plus-TCP source.
- source
  - This directory contains all the [.c] source files.

**Note** At this time it is recommended to use BufferAllocation\_2.c in which case it is essential to use the heap\_4.c memory allocation scheme. See [memory management](#).

**Kernel sources** The FreeRTOS Kernel Source is in [FreeRTOS/FreeRTOS-Kernel repository](#), and it is consumed by testing/PR checks as a submodule in this repository.

The version of the FreeRTOS Kernel Source in use could be accessed at `./test/FreeRTOS-Kernel` directory.

**CBMC** The `test/cbmc/proofs` directory contains CBMC proofs.

To learn more about CBMC and proofs specifically, review the training material [here](#).

In order to run these proofs you will need to install CBMC and other tools by following the instructions [here](#).

# Index

## Non-alphabetical

`__noreturn` (*C macro*), 1381

## A

`addDifferentSecurityModes` (*C macro*), 1230

`addMode1AndMode2` (*C macro*), 1230

`addSameSecurityModes` (*C macro*), 1230

`attErrorCode_t` (*C enum*), 1271

`attErrorCode_t.gAttErrCodeAttributeNotFound_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeAttributeNotLong_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeCccdImproperlyConfigured_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeDatabaseOutOfSync_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeInsufficientAuthentication_c` (*C enumerator*), 1271

`attErrorCode_t.gAttErrCodeInsufficientAuthorization_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeInsufficientEncryption_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeInsufficientEncryptionKeySize_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeInsufficientResources_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeInvalidAttributeValueLength_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeInvalidHandle_c` (*C enumerator*), 1271

`attErrorCode_t.gAttErrCodeInvalidOffset_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeInvalidPdu_c` (*C enumerator*), 1271

`attErrorCode_t.gAttErrCodeNoError_c` (*C enumerator*), 1271

`attErrorCode_t.gAttErrCodeOutOfRange_c` (*C enumerator*), 1273

`attErrorCode_t.gAttErrCodePrepareQueueFull_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeProcedureAlreadyInProgress_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeReadNotPermitted_c` (*C enumerator*), 1271

`attErrorCode_t.gAttErrCodeRequestNotSupported_c` (*C enumerator*), 1271

`attErrorCode_t.gAttErrCodeUnlikelyError_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeUnsupportedGroupType_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeValueNotAllowed_c` (*C enumerator*), 1272

`attErrorCode_t.gAttErrCodeWriteNotPermitted_c` (*C enumerator*), 1271

`attErrorCode_t.gAttErrCodeWriteRequestRejected_c` (*C enumerator*), 1272

## B

`bearerId_t` (*C type*), 1364

`bearerStatus_t` (*C type*), 1364

`bearerStatus_tag` (*C enum*), 1337

`bearerStatus_tag.gEattBearerActive_c` (*C enumerator*), 1337

`bearerStatus_tag.gEattBearerAllocated_c` (*C enumerator*), 1337

`bearerStatus_tag.gEattBearerAlreadyAllocated_c` (*C enumerator*), 1337

`bearerStatus_tag.gEattBearerFree_c` (*C enumerator*), 1337

`bearerStatus_tag.gEattBearerReconfInProgress_c` (*C enumerator*), 1337

`bearerStatus_tag.gEattBearerSuspended_c` (*C enumerator*), 1337

`Ble_CheckMemoryStorage` (*C function*), 1368

`Ble_CopyDeviceAddress` (*C macro*), 1370

`Ble_DeviceAddressesMatch` (*C macro*), 1370

[Ble\\_HciRecv \(C function\), 1368](#)  
[Ble\\_HciRecvFromIsr \(C function\), 1368](#)  
[Ble\\_HostDeInitialize \(C function\), 1367](#)  
[Ble\\_HostInitialize \(C function\), 1367](#)  
[Ble\\_IsPrivateNonresolvableDeviceAddress \(C macro\), 1370](#)  
[Ble\\_IsPrivateResolvableDeviceAddress \(C macro\), 1370](#)  
[Ble\\_IsRandomStaticDeviceAddress \(C macro\), 1370](#)  
[bleAddressType\\_t \(C type\), 1364](#)  
[bleAdvertiserClockAccuracy\\_t \(C type\), 1364](#)  
[bleAdvertiserClockAccuracy\\_tag \(C enum\), 1339](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc20ppm\\_c \(C enumerator\), 1340](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc30ppm\\_c \(C enumerator\), 1340](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc50ppm\\_c \(C enumerator\), 1340](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc75ppm\\_c \(C enumerator\), 1340](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc100ppm\\_c \(C enumerator\), 1340](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc150ppm\\_c \(C enumerator\), 1340](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc250ppm\\_c \(C enumerator\), 1340](#)  
[bleAdvertiserClockAccuracy\\_tag.gBleAdvertiserClkAcc500ppm\\_c \(C enumerator\), 1339](#)  
[bleAdvertisingChannelMap\\_t \(C struct\), 1383, 1417](#)  
[bleAdvertisingChannelMap\\_t.enableChannel37 \(C var\), 1383, 1417](#)  
[bleAdvertisingChannelMap\\_t.enableChannel38 \(C var\), 1383, 1417](#)  
[bleAdvertisingChannelMap\\_t.enableChannel39 \(C var\), 1383, 1417](#)  
[bleAdvertisingChannelMap\\_t.reserved \(C var\), 1383, 1417](#)  
[bleAdvertisingFilterPolicy\\_t \(C enum\), 1339](#)  
[bleAdvertisingFilterPolicy\\_t.gBleAdvFilterAllowScanFromAnyAllowConnFromAny\\_c \(C enumerator\), 1339](#)  
[bleAdvertisingFilterPolicy\\_t.gBleAdvFilterAllowScanFromAnyAllowConnFromWL\\_c \(C enumerator\), 1339](#)  
[bleAdvertisingFilterPolicy\\_t.gBleAdvFilterAllowScanFromWLLowConnFromAny\\_c \(C enumerator\), 1339](#)  
[bleAdvertisingFilterPolicy\\_t.gBleAdvFilterAllowScanFromWLLowConnFromWL\\_c \(C enumerator\), 1339](#)  
[bleAdvertisingReportEventType\\_t \(C enum\), 1340](#)  
[bleAdvertisingReportEventType\\_t.gBleAdvRepAdvDirectInd\\_c \(C enumerator\), 1340](#)  
[bleAdvertisingReportEventType\\_t.gBleAdvRepAdvInd\\_c \(C enumerator\), 1340](#)  
[bleAdvertisingReportEventType\\_t.gBleAdvRepAdvNonconnInd\\_c \(C enumerator\), 1340](#)  
[bleAdvertisingReportEventType\\_t.gBleAdvRepAdvScanInd\\_c \(C enumerator\), 1340](#)  
[bleAdvertisingReportEventType\\_t.gBleAdvRepScanRsp\\_c \(C enumerator\), 1340](#)  
[bleAdvertisingType\\_t \(C enum\), 1337](#)  
[bleAdvertisingType\\_t.gAdvConnectableUndirected\\_c \(C enumerator\), 1337](#)  
[bleAdvertisingType\\_t.gAdvDirectedHighDutyCycle\\_c \(C enumerator\), 1337](#)  
[bleAdvertisingType\\_t.gAdvDirectedLowDutyCycle\\_c \(C enumerator\), 1337](#)  
[bleAdvertisingType\\_t.gAdvNonConnectable\\_c \(C enumerator\), 1337](#)  
[bleAdvertisingType\\_t.gAdvScannable\\_c \(C enumerator\), 1337](#)  
[bleAdvIndexType\\_t \(C enum\), 1363](#)  
[bleAdvIndexType\\_t.gAdvIndexAscend\\_c \(C enumerator\), 1363](#)  
[bleAdvIndexType\\_t.gAdvIndexDescend\\_c \(C enumerator\), 1363](#)  
[bleAdvIndexType\\_t.gAdvIndexRandom\\_c \(C enumerator\), 1364](#)  
[bleAdvIndexType\\_t.gAdvIndexUser\\_c \(C enumerator\), 1363](#)  
[bleAdvReportEventProperties\\_t \(C type\), 1364](#)  
[bleAdvReportEventProperties\\_tag \(C enum\), 1337](#)  
[bleAdvReportEventProperties\\_tag.gAdvEventAnonymous\\_c \(C enumerator\), 1338](#)  
[bleAdvReportEventProperties\\_tag.gAdvEventConnectable\\_c \(C enumerator\), 1337](#)  
[bleAdvReportEventProperties\\_tag.gAdvEventDirected\\_c \(C enumerator\), 1338](#)  
[bleAdvReportEventProperties\\_tag.gAdvEventLegacy\\_c \(C enumerator\), 1338](#)  
[bleAdvReportEventProperties\\_tag.gAdvEventScannable\\_c \(C enumerator\), 1338](#)  
[bleAdvReportEventProperties\\_tag.gAdvEventScanResponse\\_c \(C enumerator\), 1338](#)  
[bleAdvRequestProperties\\_t \(C type\), 1364](#)

bleAdvRequestProperties\_tag (C enum), 1338  
 bleAdvRequestProperties\_tag.gAdvIncludeADIinDecisionPDU\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvIncludeAdvAinDecisionPDU\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvIncludeTxPower\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvReqAnonymous\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvReqConnectable\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvReqDirected\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvReqHighDutyCycle\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvReqLegacy\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvReqScannable\_c (C enumerator), 1338  
 bleAdvRequestProperties\_tag.gAdvUseDecisionPDU\_c (C enumerator), 1338  
 bleAntennaInformation\_t (C type), 1365  
 bleAntennaInformation\_tag (C struct), 1386, 1419  
 bleAntennaInformation\_tag.maxCteLength (C var), 1420  
 bleAntennaInformation\_tag.maxSwitchingPatternLength (C var), 1420  
 bleAntennaInformation\_tag.numAntennae (C var), 1420  
 bleAntennaInformation\_tag.supportedSwitchingSamplingRates (C var), 1420  
 bleBondCreatedEvent\_t (C type), 1365  
 bleBondCreatedEvent\_tag (C struct), 1385, 1419  
 bleBondCreatedEvent\_tag.address (C var), 1385, 1419  
 bleBondCreatedEvent\_tag.addressType (C var), 1385, 1419  
 bleBondCreatedEvent\_tag.nvmIndex (C var), 1385, 1419  
 bleBondDataBlob\_t (C struct), 1397, 1430  
 bleBondDataBlob\_t.bondDataBlobDeviceInfo (C var), 1430  
 bleBondDataBlob\_t.bondDataBlobDynamic (C var), 1430  
 bleBondDataBlob\_t.bondDataBlobLegacy (C var), 1430  
 bleBondDataBlob\_t.bondDataBlobStatic (C var), 1430  
 bleBondDataBlob\_t.bondDataDescriptors (C var), 1430  
 bleBondDataBlob\_t.bondHeader (C var), 1430  
 bleBondDataDescriptorBlob\_t (C struct), 1397, 1430  
 bleBondDataDescriptorBlob\_t.raw (C var), 1430  
 bleBondDataDeviceInfoBlob\_t (C struct), 1397, 1430  
 bleBondDataDeviceInfoBlob\_t.raw (C var), 1430  
 bleBondDataDynamicBlob\_t (C struct), 1397, 1429  
 bleBondDataDynamicBlob\_t.raw (C var), 1429  
 bleBondDataLegacyBlob\_t (C struct), 1397, 1429  
 bleBondDataLegacyBlob\_t.raw (C var), 1430  
 bleBondDataRam\_t (C struct), 1397, 1430  
 bleBondDataRam\_t.bondDataBlobDeviceInfo (C var), 1431  
 bleBondDataRam\_t.bondDataBlobDynamic (C var), 1431  
 bleBondDataRam\_t.bondDataBlobLegacy (C var), 1431  
 bleBondDataRam\_t.bondDataBlobStatic (C var), 1431  
 bleBondDataRam\_t.bondDataDescriptors (C var), 1431  
 bleBondDataRam\_t.bondHeader (C var), 1431  
 bleBondDataStaticBlob\_t (C struct), 1397, 1429  
 bleBondDataStaticBlob\_t.raw (C var), 1429  
 bleBondIdentityHeaderBlob\_t (C struct), 1397, 1429  
 bleBondIdentityHeaderBlob\_t.raw (C var), 1429  
 bleCentralClockAccuracy\_t (C type), 1364  
 bleCentralClockAccuracy\_tag (C enum), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc20ppm\_c (C enumerator), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc30ppm\_c (C enumerator), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc50ppm\_c (C enumerator), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc75ppm\_c (C enumerator), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc100ppm\_c (C enumerator), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc150ppm\_c (C enumerator), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc250ppm\_c (C enumerator), 1339  
 bleCentralClockAccuracy\_tag.gBleCentralClkAcc500ppm\_c (C enumerator), 1339

`bleChannelFrequency_t` (*C enum*), [1341](#)  
`bleChannelFrequency_t.gBleFreq2402MHz_c` (*C enumerator*), [1341](#)  
`bleChannelFrequency_t.gBleFreq2404MHz_c` (*C enumerator*), [1341](#)  
`bleChannelFrequency_t.gBleFreq2406MHz_c` (*C enumerator*), [1341](#)  
`bleChannelFrequency_t.gBleFreq2408MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2410MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2412MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2414MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2416MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2418MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2420MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2422MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2424MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2426MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2428MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2430MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2432MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2434MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2436MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2438MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2440MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2442MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2444MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2446MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2448MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2450MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2452MHz_c` (*C enumerator*), [1342](#)  
`bleChannelFrequency_t.gBleFreq2454MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2456MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2458MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2460MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2462MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2464MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2466MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2468MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2470MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2472MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2474MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2476MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2478MHz_c` (*C enumerator*), [1343](#)  
`bleChannelFrequency_t.gBleFreq2480MHz_c` (*C enumerator*), [1343](#)  
`bleChannelMap_t` (*C type*), [1364](#)  
`bleChannelOverrideMode_t` (*C enum*), [1341](#)  
`bleChannelOverrideMode_t.gOverrideAdvChannels_c` (*C enumerator*), [1341](#)  
`bleChannelOverrideMode_t.gOverrideInitChannels_c` (*C enumerator*), [1341](#)  
`bleChannelOverrideMode_t.gOverrideScanChannels_c` (*C enumerator*), [1341](#)  
`bleCteAllowedTypesMap_t` (*C struct*), [1397](#), [1431](#)  
`bleCteAllowedTypesMap_t.allowAoA` (*C var*), [1398](#), [1431](#)  
`bleCteAllowedTypesMap_t.allowAoD1us` (*C var*), [1398](#), [1431](#)  
`bleCteAllowedTypesMap_t.allowAoD2us` (*C var*), [1398](#), [1431](#)  
`bleCteAllowedTypesMap_t.reserved` (*C var*), [1398](#), [1431](#)  
`bleCteReqEnable_t` (*C enum*), [1360](#)  
`bleCteReqEnable_t.gCteReqDisable_c` (*C enumerator*), [1360](#)  
`bleCteReqEnable_t.gCteReqEnable_c` (*C enumerator*), [1360](#)  
`bleCteRspEnable_t` (*C enum*), [1360](#)  
`bleCteRspEnable_t.gCteRspDisable_c` (*C enumerator*), [1360](#)  
`bleCteRspEnable_t.gCteRspEnable_c` (*C enumerator*), [1360](#)  
`bleCteTransmitEnable_t` (*C enum*), [1360](#)

bleCteTransmitEnable\_t.gCteTransmitDisable\_c (C enumerator), 1360  
 bleCteTransmitEnable\_t.gCteTransmitEnable\_c (C enumerator), 1360  
 bleCteType\_t (C type), 1366  
 bleCteType\_tag (C enum), 1360  
 bleCteType\_tag.gCteTypeAoA\_c (C enumerator), 1360  
 bleCteType\_tag.gCteTypeAoD1us\_c (C enumerator), 1360  
 bleCteType\_tag.gCteTypeAoD2us\_c (C enumerator), 1360  
 bleCteType\_tag.gCteTypeNoCte\_c (C enumerator), 1361  
 bleDeviceAddress\_t (C type), 1364  
 bleGetConnParamsMode\_t (C enum), 1363  
 bleGetConnParamsMode\_t.gSendEventAfterLLProcUpdate\_c (C enumerator), 1363  
 bleGetConnParamsMode\_t.gSendEventAfterRemoteSNChange\_c (C enumerator), 1363  
 bleHandoverAnchorNotificationEnable\_t (C enum), 1363  
 bleHandoverAnchorNotificationEnable\_t.gAnchorNotificationDisable\_c (C enumerator), 1363  
 bleHandoverAnchorNotificationEnable\_t.gAnchorNotificationEnable\_c (C enumerator), 1363  
 bleHandoverAnchorSearchMode\_t (C enum), 1363  
 bleHandoverAnchorSearchMode\_t.gPacketMode\_c (C enumerator), 1363  
 bleHandoverAnchorSearchMode\_t.gRssiSniffingMode\_c (C enumerator), 1363  
 bleHandoverAnchorSearchMode\_t.gSuspendTxMode\_c (C enumerator), 1363  
 bleHandoverSuspendTransmitMode\_t (C enum), 1362  
 bleHandoverSuspendTransmitMode\_t.gDoNotUseEventCounter\_c (C enumerator), 1362  
 bleHandoverSuspendTransmitMode\_t.gSafeStopHostLlTx\_c (C enumerator), 1362  
 bleHandoverSuspendTransmitMode\_t.gSafeStopLlTx\_c (C enumerator), 1362  
 bleHandoverSuspendTransmitMode\_t.gUseEventCounter\_c (C enumerator), 1362  
 bleHandoverTimeSyncEnable\_t (C enum), 1363  
 bleHandoverTimeSyncEnable\_t.gTimeSyncDisable\_c (C enumerator), 1363  
 bleHandoverTimeSyncEnable\_t.gTimeSyncEnable\_c (C enumerator), 1363  
 bleHandoverTimeSyncStopWhenFound\_t (C enum), 1363  
 bleHandoverTimeSyncStopWhenFound\_t.gTimeSyncDoNotStopWhenFound\_c (C enumerator), 1363  
 bleHandoverTimeSyncStopWhenFound\_t.gTimeSyncStopWhenFound\_c (C enumerator), 1363  
 bleHardwareErrorCode\_t (C enum), 1344  
 bleHardwareErrorCode\_t.bleHwErrCodeNoError\_c (C enumerator), 1344  
 bleIdentityAddress\_t (C struct), 1382, 1416  
 bleIdentityAddress\_t.idAddress (C var), 1383, 1416  
 bleIdentityAddress\_t.idAddressType (C var), 1382, 1416  
 bleInitiatorFilterPolicy\_t (C type), 1364  
 bleIqReportPacketStatus\_t (C type), 1366  
 bleIqReportPacketStatus\_tag (C enum), 1361  
 bleIqReportPacketStatus\_tag.gIqReportPacketStatusCorrectCrc\_c (C enumerator), 1361  
 bleIqReportPacketStatus\_tag.gIqReportPacketStatusCrcIncorrectUsedLength\_c (C enumerator), 1361  
 bleIqReportPacketStatus\_tag.gIqReportPacketStatusCrcIncorrectUsedOther\_c (C enumerator), 1361  
 bleIqReportPacketStatus\_tag.gIqReportPacketStatusInsufficientResources\_c (C enumerator), 1361  
 bleIqSamplingEnable\_t (C enum), 1360  
 bleIqSamplingEnable\_t.gIqSamplingDisable\_c (C enumerator), 1360  
 bleIqSamplingEnable\_t.gIqSamplingEnable\_c (C enumerator), 1360  
 bleLlConnectionRole\_t (C enum), 1339  
 bleLlConnectionRole\_t.gBleLlConnectionCentral\_c (C enumerator), 1339  
 bleLlConnectionRole\_t.gBleLlConnectionPeripheral\_c (C enumerator), 1339  
 bleLocalKeysBlob\_t (C struct), 1397, 1430  
 bleLocalKeysBlob\_t.\_\_unnamed2\_\_ (C union), 1402, 1434  
 bleLocalKeysBlob\_t.\_\_unnamed2\_\_.pKey (C var), 1402, 1434  
 bleLocalKeysBlob\_t.\_\_unnamed2\_\_.raw (C var), 1402, 1434  
 bleLocalKeysBlob\_t.keyGenerated (C var), 1430  
 bleNotificationEvent\_t (C struct), 1384, 1418  
 bleNotificationEvent\_t.adv\_handle (C var), 1385, 1418  
 bleNotificationEvent\_t.ce\_counter (C var), 1384, 1418  
 bleNotificationEvent\_t.channel (C var), 1384, 1418  
 bleNotificationEvent\_t.deviceId (C var), 1384, 1418

`bleNotificationEvent_t.eventType` (*C var*), [1384](#), [1418](#)  
`bleNotificationEvent_t.rssi` (*C var*), [1384](#), [1418](#)  
`bleNotificationEvent_t.scanned_addr` (*C var*), [1385](#), [1418](#)  
`bleNotificationEvent_t.status` (*C var*), [1384](#), [1418](#)  
`bleNotificationEvent_t.timestamp` (*C var*), [1384](#), [1418](#)  
`bleNotificationEventType_t` (*C type*), [1365](#)  
`bleNotificationEventType_tag` (*C enum*), [1359](#)  
`bleNotificationEventType_tag.gNotifAdvConnReqRx_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifAdvEventOver_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifAdvScanReqRx_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifAdvTx_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifChannelMatrix_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifConnChannelMapUpdate_c` (*C enumerator*), [1360](#)  
`bleNotificationEventType_tag.gNotifConnCreated_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifConnEventOver_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifConnInd_c` (*C enumerator*), [1360](#)  
`bleNotificationEventType_tag.gNotifConnRxPdu_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifEventNone_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifPhyReq_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifPhyUpdateInd_c` (*C enumerator*), [1360](#)  
`bleNotificationEventType_tag.gNotifScanAdvPktRx_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifScanEventOver_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifScanReqTx_c` (*C enumerator*), [1359](#)  
`bleNotificationEventType_tag.gNotifScanRspRx_c` (*C enumerator*), [1359](#)  
`blePathLossReportingEnable_t` (*C enum*), [1362](#)  
`blePathLossReportingEnable_t.gPathLossReportingDisable_c` (*C enumerator*), [1362](#)  
`blePathLossReportingEnable_t.gPathLossReportingEnable_c` (*C enumerator*), [1362](#)  
`blePathLossThresholdZoneEntered_t` (*C type*), [1367](#)  
`blePathLossThresholdZoneEntered_tag` (*C enum*), [1361](#)  
`blePathLossThresholdZoneEntered_tag.gPathLossThresholdHighZone_c` (*C enumerator*), [1361](#)  
`blePathLossThresholdZoneEntered_tag.gPathLossThresholdLowZone_c` (*C enumerator*), [1361](#)  
`blePathLossThresholdZoneEntered_tag.gPathLossThresholdMiddleZone_c` (*C enumerator*), [1361](#)  
`blePowerControlPhyType_t` (*C type*), [1367](#)  
`blePowerControlPhyType_tag` (*C enum*), [1362](#)  
`blePowerControlPhyType_tag.gPowerControlLePhy1M_c` (*C enumerator*), [1362](#)  
`blePowerControlPhyType_tag.gPowerControlLePhy2M_c` (*C enumerator*), [1362](#)  
`blePowerControlPhyType_tag.gPowerControlLePhyCodedS2_c` (*C enumerator*), [1362](#)  
`blePowerControlPhyType_tag.gPowerControlLePhyCodedS8_c` (*C enumerator*), [1362](#)  
`blePrivacyMode_t` (*C type*), [1364](#)  
`bleResult_t` (*C type*), [1364](#)  
`bleResult_tag` (*C enum*), [1323](#)  
`bleResult_tag.gAttStatusBase_c` (*C enumerator*), [1334](#)  
`bleResult_tag.gAttSuccess_c` (*C enumerator*), [1334](#)  
`bleResult_tag.gBleAlreadyInitialized_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleFeatureNotSupported_c` (*C enumerator*), [1323](#)  
`bleResult_tag.gBleInvalidParameter_c` (*C enumerator*), [1323](#)  
`bleResult_tag.gBleInvalidState_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleNVMEError_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleOsError_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleOutOfMemory_c` (*C enumerator*), [1323](#)  
`bleResult_tag.gBleOverflow_c` (*C enumerator*), [1323](#)  
`bleResult_tag.gBleReassemblyInProgress_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleRngError_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleSecLibError_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleStatusBase_c` (*C enumerator*), [1323](#)  
`bleResult_tag.gBleSuccess_c` (*C enumerator*), [1323](#)  
`bleResult_tag.gBleTimerError_c` (*C enumerator*), [1324](#)  
`bleResult_tag.gBleUnavailable_c` (*C enumerator*), [1323](#)

bleResult\_tag.gBleUnexpectedError\_c (C enumerator), 1324  
 bleResult\_tag.gCsCallbackAlreadyInstalled\_c (C enumerator), 1337  
 bleResult\_tag.gCsSecurityCheckFail\_c (C enumerator), 1337  
 bleResult\_tag.gCsStatusBase\_c (C enumerator), 1336  
 bleResult\_tag.gCsSuccess\_c (C enumerator), 1336  
 bleResult\_tag.gCtrlStatusBase\_c (C enumerator), 1329  
 bleResult\_tag.gCtrlSuccess\_c (C enumerator), 1329  
 bleResult\_tag.gDevDbCccdLimitReached\_c (C enumerator), 1336  
 bleResult\_tag.gDevDbCccdNotFound\_c (C enumerator), 1336  
 bleResult\_tag.gDevDbStatusBase\_c (C enumerator), 1336  
 bleResult\_tag.gDevDbSuccess\_c (C enumerator), 1336  
 bleResult\_tag.gGapAdvDataTooLong\_c (C enumerator), 1335  
 bleResult\_tag.gGapAnotherProcedureInProgress\_c (C enumerator), 1336  
 bleResult\_tag.gGapDeviceNotBonded\_c (C enumerator), 1336  
 bleResult\_tag.gGapScanRspDataTooLong\_c (C enumerator), 1335  
 bleResult\_tag.gGapStatusBase\_c (C enumerator), 1335  
 bleResult\_tag.gGapSuccess\_c (C enumerator), 1335  
 bleResult\_tag.gGattAnotherProcedureInProgress\_c (C enumerator), 1334  
 bleResult\_tag.gGattClientConfirmationTimeout\_c (C enumerator), 1335  
 bleResult\_tag.gGattConnectionSecurityRequirementsNotMet\_c (C enumerator), 1335  
 bleResult\_tag.gGattDbCccdNotFound\_c (C enumerator), 1336  
 bleResult\_tag.gGattDbCharacteristicNotFound\_c (C enumerator), 1336  
 bleResult\_tag.gGattDbDescriptorNotFound\_c (C enumerator), 1336  
 bleResult\_tag.gGattDbInvalidHandle\_c (C enumerator), 1336  
 bleResult\_tag.gGattDbServiceNotFound\_c (C enumerator), 1336  
 bleResult\_tag.gGattDbServiceOrCharAlreadyDeclared\_c (C enumerator), 1336  
 bleResult\_tag.gGattDbStatusBase\_c (C enumerator), 1336  
 bleResult\_tag.gGattDbSuccess\_c (C enumerator), 1336  
 bleResult\_tag.gGattIndicationAlreadyInProgress\_c (C enumerator), 1335  
 bleResult\_tag.gGattInvalidPduReceived\_c (C enumerator), 1335  
 bleResult\_tag.gGattInvalidValueLength\_c (C enumerator), 1335  
 bleResult\_tag.gGattLongAttributePacketsCorrupted\_c (C enumerator), 1334  
 bleResult\_tag.gGattMtuExchangeInProgress\_c (C enumerator), 1335  
 bleResult\_tag.gGattMultipleAttributesOverflow\_c (C enumerator), 1335  
 bleResult\_tag.gGattOutOfSyncProceduresOngoing\_c (C enumerator), 1335  
 bleResult\_tag.gGattPeerDisconnected\_c (C enumerator), 1335  
 bleResult\_tag.gGattServerTimeout\_c (C enumerator), 1335  
 bleResult\_tag.gGattStatusBase\_c (C enumerator), 1334  
 bleResult\_tag.gGattSuccess\_c (C enumerator), 1334  
 bleResult\_tag.gGattUnexpectedReadMultipleResponseLength\_c (C enumerator), 1335  
 bleResult\_tag.gHciAclConnectionAlreadyExists\_c (C enumerator), 1325  
 bleResult\_tag.gHciAlreadyInit\_c (C enumerator), 1329  
 bleResult\_tag.gHciAuthenticationFailure\_c (C enumerator), 1324  
 bleResult\_tag.gHciCallbackAlreadyInstalled\_c (C enumerator), 1329  
 bleResult\_tag.gHciCallbackNotInstalled\_c (C enumerator), 1329  
 bleResult\_tag.gHciChannelClassificationNotSupported\_c (C enumerator), 1327  
 bleResult\_tag.gHciCoarseClockAdjustmentRejected\_c (C enumerator), 1328  
 bleResult\_tag.gHciCommandDisallowed\_c (C enumerator), 1325  
 bleResult\_tag.gHciCommandNotSupported\_c (C enumerator), 1329  
 bleResult\_tag.gHciConnectionAcceptTimeoutExceeded\_c (C enumerator), 1325  
 bleResult\_tag.gHciConnectionFailedToBeEstablishedOrSyncTimeout\_c (C enumerator), 1328  
 bleResult\_tag.gHciConnectionLimitExceeded\_c (C enumerator), 1325  
 bleResult\_tag.gHciConnectionRejectedDueToLimitedResources\_c (C enumerator), 1325  
 bleResult\_tag.gHciConnectionRejectedDueToNoSuitableChannelFound\_c (C enumerator), 1328  
 bleResult\_tag.gHciConnectionRejectedDueToSecurityReasons\_c (C enumerator), 1325  
 bleResult\_tag.gHciConnectionRejectedDueToUnacceptableBdAddr\_c (C enumerator), 1325  
 bleResult\_tag.gHciConnectionTerminatedByLocalHost\_c (C enumerator), 1326  
 bleResult\_tag.gHciConnectionTerminatedDueToMicFailure\_c (C enumerator), 1328

bleResult\_tag.gHciConnectionTimeout\_c (C enumerator), 1325

bleResult\_tag.gHciControllerBusy\_c (C enumerator), 1328

bleResult\_tag.gHciDifferentTransactionCollision\_c (C enumerator), 1327

bleResult\_tag.gHciDirectedAdvertisingTimeout\_c (C enumerator), 1328

bleResult\_tag.gHciEncryptionModeNotAcceptable\_c (C enumerator), 1327

bleResult\_tag.gHciEventNotSupported\_c (C enumerator), 1329

bleResult\_tag.gHciExtendedInquiryResponseTooLarge\_c (C enumerator), 1328

bleResult\_tag.gHciHardwareFailure\_c (C enumerator), 1324

bleResult\_tag.gHciHostBusyPairing\_c (C enumerator), 1328

bleResult\_tag.gHciInstantPassed\_c (C enumerator), 1327

bleResult\_tag.gHciInsufficientSecurity\_c (C enumerator), 1327

bleResult\_tag.gHciInvalidHciCommandParameters\_c (C enumerator), 1325

bleResult\_tag.gHciInvalidLpmParameters\_c (C enumerator), 1326

bleResult\_tag.gHciInvalidParameter\_c (C enumerator), 1329

bleResult\_tag.gHciLimitReached\_c (C enumerator), 1329

bleResult\_tag.gHciLinkKeyCannotBeChanged\_c (C enumerator), 1327

bleResult\_tag.gHciLLResponseTimeout\_c (C enumerator), 1326

bleResult\_tag.gHciLmpErrorTransactionCollision\_c (C enumerator), 1326

bleResult\_tag.gHciLmpPduNotAllowed\_c (C enumerator), 1326

bleResult\_tag.gHciMacConnectionFailed\_c (C enumerator), 1328

bleResult\_tag.gHciMemoryCapacityExceeded\_c (C enumerator), 1325

bleResult\_tag.gHciOperationCancelledByHost\_c (C enumerator), 1329

bleResult\_tag.gHciPacketTooEarly\_c (C enumerator), 1329

bleResult\_tag.gHciPacketTooLate\_c (C enumerator), 1329

bleResult\_tag.gHciPacketTooLong\_c (C enumerator), 1329

bleResult\_tag.gHciPageTimeout\_c (C enumerator), 1324

bleResult\_tag.gHciPairingNotAllowed\_c (C enumerator), 1326

bleResult\_tag.gHciPairingWithUnitKeyNotSupported\_c (C enumerator), 1327

bleResult\_tag.gHciParameterOutOfMandatoryRange\_c (C enumerator), 1327

bleResult\_tag.gHciPinOrKeyMissing\_c (C enumerator), 1325

bleResult\_tag.gHciQosNotAcceptableParameter\_c (C enumerator), 1327

bleResult\_tag.gHciQosRejected\_c (C enumerator), 1327

bleResult\_tag.gHciRemoteDeviceTerminatedConnectionLowResources\_c (C enumerator), 1325

bleResult\_tag.gHciRemoteDeviceTerminatedConnectionPowerOff\_c (C enumerator), 1325

bleResult\_tag.gHciRemoteUserTerminatedConnection\_c (C enumerator), 1325

bleResult\_tag.gHciRepeatedAttempts\_c (C enumerator), 1326

bleResult\_tag.gHciRequestedQosNotSupported\_c (C enumerator), 1327

bleResult\_tag.gHciReserved\_0x2B\_c (C enumerator), 1327

bleResult\_tag.gHciReserved\_0x31\_c (C enumerator), 1327

bleResult\_tag.gHciReserved\_0x33\_c (C enumerator), 1327

bleResult\_tag.gHciReservedSlotViolation\_c (C enumerator), 1327

bleResult\_tag.gHciRoleChangeNotAllowed\_c (C enumerator), 1326

bleResult\_tag.gHciRoleSwitchFailed\_c (C enumerator), 1328

bleResult\_tag.gHciRoleSwitchPending\_c (C enumerator), 1327

bleResult\_tag.gHciScoAirModeRejected\_c (C enumerator), 1326

bleResult\_tag.gHciScoIntervalRejected\_c (C enumerator), 1326

bleResult\_tag.gHciScoOffsetRejected\_c (C enumerator), 1326

bleResult\_tag.gHciSecureSimplePairingNotSupportedByHost\_c (C enumerator), 1328

bleResult\_tag.gHciStatusBase\_c (C enumerator), 1324

bleResult\_tag.gHciSuccess\_c (C enumerator), 1324

bleResult\_tag.gHciSynchronousConnectionLimitToADeviceExceeded\_c (C enumerator), 1325

bleResult\_tag.gHciTransportError\_c (C enumerator), 1329

bleResult\_tag.gHciType0SubmapNotDefined\_c (C enumerator), 1328

bleResult\_tag.gHciUnacceptableConnectionParameters\_c (C enumerator), 1328

bleResult\_tag.gHciUnknownAdvertisingIdentifier\_c (C enumerator), 1328

bleResult\_tag.gHciUnknownConnectionIdentifier\_c (C enumerator), 1324

bleResult\_tag.gHciUnknownHciCommand\_c (C enumerator), 1324

bleResult\_tag.gHciUnknownLpmPdu\_c (C enumerator), 1326

bleResult\_tag.gHciUnspecifiedError\_c (C enumerator), 1326  
bleResult\_tag.gHciUnsupportedFeatureOrParameterValue\_c (C enumerator), 1325  
bleResult\_tag.gHciUnsupportedLpmParameterValue\_c (C enumerator), 1326  
bleResult\_tag.gHciUnsupportedRemoteFeature\_c (C enumerator), 1326  
bleResult\_tag.gL2caAlreadyInit\_c (C enumerator), 1330  
bleResult\_tag.gL2caCallbackAlreadyInstalled\_c (C enumerator), 1330  
bleResult\_tag.gL2caCallbackNotInstalled\_c (C enumerator), 1330  
bleResult\_tag.gL2caChannelAlreadyConnected\_c (C enumerator), 1330  
bleResult\_tag.gL2caChannelBusy\_c (C enumerator), 1330  
bleResult\_tag.gL2caChannelClosed\_c (C enumerator), 1330  
bleResult\_tag.gL2caChannelInvalid\_c (C enumerator), 1330  
bleResult\_tag.gL2caConnectionParametersRejected\_c (C enumerator), 1330  
bleResult\_tag.gL2caInsufficientResources\_c (C enumerator), 1330  
bleResult\_tag.gL2caInternalError\_c (C enumerator), 1330  
bleResult\_tag.gL2caInvalidParameter\_c (C enumerator), 1330  
bleResult\_tag.gL2caLePsmAlreadyRegistered\_c (C enumerator), 1330  
bleResult\_tag.gL2caLePsmInsufficientResources\_c (C enumerator), 1330  
bleResult\_tag.gL2caLePsmInvalid\_c (C enumerator), 1330  
bleResult\_tag.gL2caLePsmNotRegistered\_c (C enumerator), 1330  
bleResult\_tag.gL2caStatusBase\_c (C enumerator), 1329  
bleResult\_tag.gL2caSuccess\_c (C enumerator), 1330  
bleResult\_tag.gSmCommandNotSupported\_c (C enumerator), 1331  
bleResult\_tag.gSmImproperKeyDistributionField\_c (C enumerator), 1331  
bleResult\_tag.gSmInsufficientResources\_c (C enumerator), 1332  
bleResult\_tag.gSmInvalidCommandCode\_c (C enumerator), 1331  
bleResult\_tag.gSmInvalidCommandLength\_c (C enumerator), 1331  
bleResult\_tag.gSmInvalidCommandParameter\_c (C enumerator), 1331  
bleResult\_tag.gSmInvalidConnectionHandle\_c (C enumerator), 1331  
bleResult\_tag.gSmInvalidDeviceId\_c (C enumerator), 1331  
bleResult\_tag.gSmInvalidHciEventParameter\_c (C enumerator), 1332  
bleResult\_tag.gSmInvalidInternalOperation\_c (C enumerator), 1331  
bleResult\_tag.gSmInvalidPeerPublicKey\_c (C enumerator), 1333  
bleResult\_tag.gSmInvalidSmpPacketLength\_c (C enumerator), 1332  
bleResult\_tag.gSmInvalidSmpPacketParameter\_c (C enumerator), 1332  
bleResult\_tag.gSmKeySessionKeyDerivationFailed\_c (C enumerator), 1334  
bleResult\_tag.gSmLlConnectionEncryptionFailure\_c (C enumerator), 1332  
bleResult\_tag.gSmLlConnectionEncryptionInProgress\_c (C enumerator), 1332  
bleResult\_tag.gSmNullCBFunction\_c (C enumerator), 1331  
bleResult\_tag.gSmOobDataAddressMismatch\_c (C enumerator), 1332  
bleResult\_tag.gSmPairingAlreadyStarted\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorAuthenticationRequirements\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorBusy\_c (C enumerator), 1334  
bleResult\_tag.gSmPairingErrorCommandNotSupported\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorConfirmValueFailed\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorDhKeyCheckFailed\_c (C enumerator), 1334  
bleResult\_tag.gSmPairingErrorEncryptionKeySize\_c (C enumerator), 1334  
bleResult\_tag.gSmPairingErrorInvalidParameters\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorKeyRejected\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorNumericComparisonFailed\_c (C enumerator), 1334  
bleResult\_tag.gSmPairingErrorOobNotAvailable\_c (C enumerator), 1334  
bleResult\_tag.gSmPairingErrorPairingNotSupported\_c (C enumerator), 1334  
bleResult\_tag.gSmPairingErrorPasskeyEntryFailed\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorRepeatedAttempts\_c (C enumerator), 1334  
bleResult\_tag.gSmPairingErrorTimeout\_c (C enumerator), 1333  
bleResult\_tag.gSmPairingErrorUnknownReason\_c (C enumerator), 1333  
bleResult\_tag.gSmReceivedHciEventFromUnknownDevice\_c (C enumerator), 1332  
bleResult\_tag.gSmReceivedSmpPacketFromUnknownDevice\_c (C enumerator), 1332  
bleResult\_tag.gSmReceivedTimerEventForUnknownDevice\_c (C enumerator), 1333

`bleResult_tag.gSmReceivedUnexpectedHciEvent_c` (*C enumerator*), 1332

`bleResult_tag.gSmReceivedUnexpectedSmpPacket_c` (*C enumerator*), 1332

`bleResult_tag.gSmSmpPacketReceivedAfterTimeoutOccurred_c` (*C enumerator*), 1332

`bleResult_tag.gSmSmpTimeoutOccurred_c` (*C enumerator*), 1332

`bleResult_tag.gSmStatusBase_c` (*C enumerator*), 1331

`bleResult_tag.gSmSuccess_c` (*C enumerator*), 1331

`bleResult_tag.gSmTbInvalidDataSignature_c` (*C enumerator*), 1334

`bleResult_tag.gSmTbResolvableAddressDoesNotMatchIrk_c` (*C enumerator*), 1334

`bleResult_tag.gSmUnattainableLocalDeviceMinKeySize_c` (*C enumerator*), 1333

`bleResult_tag.gSmUnattainableLocalDeviceSecRequirements_c` (*C enumerator*), 1333

`bleResult_tag.gSmUnattainablePeripheralSecReqRequirements_c` (*C enumerator*), 1333

`bleResult_tag.gSmUnexpectedCommand_c` (*C enumerator*), 1331

`bleResult_tag.gSmUnexpectedKeyset_c` (*C enumerator*), 1331

`bleResult_tag.gSmUnexpectedKeyType_c` (*C enumerator*), 1331

`bleResult_tag.gSmUnexpectedPairingTerminationReason_c` (*C enumerator*), 1331

`bleResult_tag.gSmUnknownSmpPacketType_c` (*C enumerator*), 1332

`bleScanningFilterPolicy_t` (*C type*), 1364

`bleScanType_t` (*C enum*), 1340

`bleScanType_t.gScanTypeActive_c` (*C enumerator*), 1341

`bleScanType_t.gScanTypePassive_c` (*C enumerator*), 1341

`BleSig_IsGroupingAttributeUuid16` (*C macro*), 1381

`BleSig_IsServiceDeclarationUuid16` (*C macro*), 1381

`bleSlotDurations_t` (*C type*), 1366

`bleSlotDurations_tag` (*C enum*), 1361

`bleSlotDurations_tag.gSlotDurations1us_c` (*C enumerator*), 1361

`bleSlotDurations_tag.gSlotDurations2us_c` (*C enumerator*), 1361

`bleSupportedSwitchingSamplingRates_t` (*C struct*), 1385, 1419

`bleSupportedSwitchingSamplingRates_t.reserved` (*C var*), 1386, 1419

`bleSupportedSwitchingSamplingRates_t.samplingSupportedAodReception` (*C var*), 1385, 1419

`bleSupportedSwitchingSamplingRates_t.switchingSamplingSupportedAoaReception` (*C var*), 1386, 1419

`bleSupportedSwitchingSamplingRates_t.switchingSupportedAodTransmission` (*C var*), 1385, 1419

`bleSyncCteType_t` (*C type*), 1366

`bleSyncCteType_tag` (*C struct*), 1398, 1431

`bleSyncCteType_tag.doNotSyncWithAoA` (*C var*), 1398, 1431

`bleSyncCteType_tag.doNotSyncWithAoD1us` (*C var*), 1398, 1431

`bleSyncCteType_tag.doNotSyncWithAoD2us` (*C var*), 1398, 1431

`bleSyncCteType_tag.doNotSyncWithoutCte` (*C var*), 1398, 1431

`bleSyncCteType_tag.doNotSyncWithType3` (*C var*), 1398, 1431

`bleSyncCteType_tag.reserved` (*C var*), 1398, 1431

`bleTransmitPowerChannelType_t` (*C enum*), 1341

`bleTransmitPowerChannelType_t.gTxPowerAdvChannel_c` (*C enumerator*), 1341

`bleTransmitPowerChannelType_t.gTxPowerConnChannel_c` (*C enumerator*), 1341

`bleTransmitPowerLevelType_t` (*C enum*), 1341

`bleTransmitPowerLevelType_t.gReadCurrentTxPowerLevel_c` (*C enumerator*), 1341

`bleTransmitPowerLevelType_t.gReadMaximumTxPowerLevel_c` (*C enumerator*), 1341

`bleTxPowerLevelFlags_t` (*C struct*), 1398, 1431

`bleTxPowerLevelFlags_t.maximum` (*C var*), 1399, 1432

`bleTxPowerLevelFlags_t.minimum` (*C var*), 1398, 1432

`bleTxPowerLevelFlags_t.reserved` (*C var*), 1399, 1432

`bleTxPowerReportingEnable_t` (*C enum*), 1362

`bleTxPowerReportingEnable_t.gTxPowerReportingDisable_c` (*C enumerator*), 1362

`bleTxPowerReportingEnable_t.gTxPowerReportingEnable_c` (*C enumerator*), 1362

`bleTxPowerReportingReason_t` (*C type*), 1367

`bleTxPowerReportingReason_tag` (*C enum*), 1361

`bleTxPowerReportingReason_tag.gLocalTxPowerChanged_c` (*C enumerator*), 1361

`bleTxPowerReportingReason_tag.gReadRemoteTxPowerLevelCommandCompleted_c` (*C enumerator*), 1362

bleTxPowerReportingReason\_tag.gRemoteTxPowerChanged\_c (C enumerator), 1362  
 bleTxTestPacketPayload\_t (C enum), 1343  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPattern00001111\_c (C enumerator), 1343  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPattern01010101\_c (C enumerator), 1344  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPattern10101010\_c (C enumerator), 1343  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPattern11110000\_c (C enumerator), 1343  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPatternAllBits0\_c (C enumerator), 1343  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPatternAllBits1\_c (C enumerator), 1343  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPrbs9\_c (C enumerator), 1343  
 bleTxTestPacketPayload\_t.gBleTestPacketPayloadPrbs15\_c (C enumerator), 1343  
 bleUuid\_t (C union), 1383, 1416  
 bleUuid\_t.uuid16 (C var), 1383, 1417  
 bleUuid\_t.uuid32 (C var), 1383, 1417  
 bleUuid\_t.uuid128 (C var), 1383, 1417  
 bleUuidType\_t (C type), 1364

## D

deviceId\_t (C type), 1364

## G

gAdvDataChange\_c (C macro), 1370  
 gaGattDynamicAttrBlob (C var), 1316  
 gaGattDynamicValBlob (C var), 1316  
 Gap\_AcceptPairingRequest (C function), 1181  
 Gap\_AddDeviceToFilterAcceptList (C function), 1194  
 Gap\_AddSecurityModesAndLevels (C macro), 1228  
 Gap\_Authorize (C function), 1176  
 Gap\_BleAdvIndexChange (C function), 1206  
 Gap\_CancelInitiatingConnection (C macro), 1228  
 Gap\_CheckIfBonded (C function), 1192  
 Gap\_CheckIfConnected (C function), 1193  
 Gap\_CheckIndicationStatus (C function), 1178  
 Gap\_CheckNotificationStatus (C function), 1178  
 Gap\_CheckNvmIndex (C function), 1193  
 Gap\_ClearFilterAcceptList (C function), 1194  
 Gap\_Connect (C function), 1189  
 Gap\_ConnectFromPawr (C function), 1189  
 Gap\_ControllerEnhancedNotification (C function), 1205  
 Gap\_ControllerTest (C function), 1203  
 Gap\_CreateRandomDeviceAddress (C function), 1194  
 Gap\_DecryptAdvertisingData (C function), 1227  
 Gap\_DenyLongTermKey (C function), 1185  
 Gap\_Disconnect (C function), 1190  
 Gap\_EattConnectionAccept (C function), 1223  
 Gap\_EattConnectionRequest (C function), 1222  
 Gap\_EattDisconnect (C function), 1224  
 Gap\_EattReconfigureRequest (C function), 1223  
 Gap\_EattSendCredits (C function), 1224  
 Gap\_EnableConnectionCteRequest (C function), 1220  
 Gap\_EnableConnectionCteResponse (C function), 1220  
 Gap\_EnableConnectionlessCteTransmit (C function), 1218  
 Gap\_EnableConnectionlessIqSampling (C function), 1219  
 Gap\_EnableControllerPrivacy (C function), 1202  
 Gap\_EnableHostPrivacy (C function), 1202  
 Gap\_EnablePathLossReporting (C function), 1221  
 Gap\_EnableTransmitPowerReporting (C function), 1222  
 Gap\_EnableUpdateConnectionParameters (C function), 1201  
 Gap\_EncryptAdvertisingData (C function), 1227

Gap\_EncryptLink (C function), 1180  
Gap\_EnhancedReadTransmitPowerLevel (C function), 1220  
Gap\_EnterPasskey (C function), 1181  
Gap\_GenerateDhKeyV2 (C function), 1222  
Gap\_GetBondedDeviceName (C function), 1196  
Gap\_GetBondedDevicesCount (C function), 1196  
Gap\_GetBondedDevicesIdentityInformation (C function), 1179  
Gap\_GetConnectionHandleFromDeviceId (C function), 1225  
Gap\_GetConnParams (C macro), 1229  
Gap\_GetConnParamsMonitoring (C function), 1225  
Gap\_GetDeviceIdFromConnHandle (C function), 1224  
Gap\_GetHostVersion (C function), 1225  
Gap\_LeChannelOverride (C function), 1225  
Gap\_LeReadPhy (C function), 1204  
Gap\_LeScGetLocalOobData (C function), 1183  
Gap\_LeScRegeneratePublicKey (C function), 1183  
Gap\_LeScSendKeypressNotification (C function), 1184  
Gap\_LeScSetPeerOobData (C function), 1184  
Gap\_LeScValidateNumericValue (C function), 1183  
Gap\_LeSetHostFeature (C function), 1226  
Gap\_LeSetPhy (C function), 1205  
Gap\_LeSetSchedulerPriority (C function), 1226  
Gap\_LoadCustomBondedDeviceInformation (C function), 1191  
Gap\_LoadCustomPeerInformation (C function), 1191  
Gap\_LoadEncryptionInformation (C function), 1185  
Gap\_LoadKeys (C function), 1206  
Gap\_ModifySleepClockAccuracy (C function), 1204  
Gap\_Pair (C function), 1179  
Gap\_PeriodicAdvCreateSync (C function), 1215  
Gap\_PeriodicAdvReceiveDisable (C function), 1216  
Gap\_PeriodicAdvReceiveEnable (C function), 1216  
Gap\_PeriodicAdvSetInfoTransfer (C function), 1217  
Gap\_PeriodicAdvSyncTransfer (C function), 1217  
Gap\_PeriodicAdvTerminateSync (C function), 1216  
Gap\_ProvideLongTermKey (C function), 1184  
Gap\_ProvideOob (C function), 1182  
Gap\_ReadAdvertisingTxPowerLevel (C macro), 1228  
Gap\_ReadAntennaInformation (C function), 1220  
Gap\_ReadChannelMap (C function), 1208  
Gap\_ReadControllerLocalRPA (C function), 1200  
Gap\_ReadFilterAcceptListSize (C function), 1193  
Gap\_ReadPublicDeviceAddress (C function), 1194  
Gap\_ReadRadioPowerLevel (C function), 1198  
Gap\_ReadRemoteTransmitPowerLevel (C function), 1221  
Gap\_ReadRemoteVersionInformation (C function), 1225  
Gap\_ReadRssi (C macro), 1229  
Gap\_ReadTxPowerLevelInConnection (C macro), 1229  
Gap\_RegisterDeviceSecurityRequirements (C function), 1174  
Gap\_RejectKeyExchangeRequest (C function), 1183  
Gap\_RejectPairing (C function), 1181  
Gap\_RejectPasskeyRequest (C function), 1182  
Gap\_RemoveAdvSet (C function), 1211  
Gap\_RemoveAllBonds (C function), 1198  
Gap\_RemoveBond (C function), 1197  
Gap\_RemoveDeviceFromFilterAcceptList (C function), 1194  
Gap\_ResumeLeScStateMachine (C function), 1218  
Gap\_SaveCccd (C function), 1177  
Gap\_SaveCustomPeerInformation (C function), 1190

Gap\_SaveDeviceName (C function), 1195  
 Gap\_SaveKeys (C function), 1207  
 Gap\_SendPeripheralSecurityRequest (C function), 1180  
 Gap\_SendSmpKeys (C function), 1182  
 Gap\_SetAdvertisingData (C function), 1175  
 Gap\_SetAdvertisingParameters (C function), 1174  
 Gap\_SetBondedDeviceName (C function), 1197  
 Gap\_SetChannelMap (C function), 1207  
 Gap\_SetConnectionCallback (C function), 1227  
 Gap\_SetConnectionCteReceiveParameters (C function), 1219  
 Gap\_SetConnectionCteTransmitParameters (C function), 1219  
 Gap\_SetConnectionlessCteTransmitParameters (C function), 1218  
 Gap\_SetDataRelatedAddressChanges (C function), 1173  
 Gap\_SetDecisionInstructions (C function), 1187  
 Gap\_SetDefaultPairingParameters (C function), 1200  
 Gap\_SetDefaultPeriodicAdvSyncTransferParams (C function), 1218  
 Gap\_SetExtAdvertisingData (C function), 1209  
 Gap\_SetExtAdvertisingDecisionData (C function), 1210  
 Gap\_SetExtAdvertisingParameters (C function), 1208  
 Gap\_SetLocalPasskey (C function), 1186  
 Gap\_SetPathLossReportingParameters (C function), 1221  
 Gap\_SetPeriodicAdvertisingData (C function), 1213  
 Gap\_SetPeriodicAdvParameters (C function), 1212  
 Gap\_SetPeriodicAdvResponseData (C function), 1213  
 Gap\_SetPeriodicAdvSubeventData (C function), 1212  
 Gap\_SetPeriodicAdvSyncTransferParams (C function), 1217  
 Gap\_SetPeriodicSyncSubevent (C function), 1213  
 Gap\_SetPrivacyMode (C function), 1203  
 Gap\_SetRandomAddress (C function), 1199  
 Gap\_SetScanMode (C function), 1187  
 Gap\_SetScanningCallback (C function), 1228  
 Gap\_SetTxPowerLevel (C function), 1198  
 Gap\_StartAdvertising (C function), 1175  
 Gap\_StartExtAdvertising (C function), 1210  
 Gap\_StartPeriodicAdvertising (C function), 1214  
 Gap\_StartScanning (C function), 1188  
 Gap\_StopAdvertising (C function), 1176  
 Gap\_StopExtAdvertising (C function), 1211  
 Gap\_StopPeriodicAdvertising (C function), 1214  
 Gap\_StopScanning (C function), 1189  
 Gap\_UpdateConnectionParameters (C function), 1200  
 Gap\_UpdateLeDataLength (C function), 1201  
 Gap\_UpdatePeriodicAdvList (C function), 1215  
 Gap\_VerifyPrivateResolvableAddress (C function), 1199  
 gapAddrReadyEvent\_t (C type), 1365  
 gapAddrReadyEvent\_t\_tag (C struct), 1385, 1419  
 gapAddrReadyEvent\_t\_tag.aAddress (C var), 1385, 1419  
 gapAddrReadyEvent\_t\_tag.advHandle (C var), 1385, 1419  
 gapAdStructure\_t (C struct), 1250, 1444  
 gapAdStructure\_t.aData (C var), 1250, 1444  
 gapAdStructure\_t.adType (C var), 1250, 1444  
 gapAdStructure\_t.length (C var), 1250, 1444  
 gapAdType\_t (C enum), 1158  
 gapAdType\_t.gAd3dInformationData\_c (C enumerator), 1160  
 gapAdType\_t.gAdAdvertisingInterval\_c (C enumerator), 1160  
 gapAdType\_t.gAdAdvertisingIntervalLong\_c (C enumerator), 1160  
 gapAdType\_t.gAdAppearance\_c (C enumerator), 1159  
 gapAdType\_t.gAdChannelMapUpdateIndication\_c (C enumerator), 1160

gapAdType\_t.gAdClassOfDevice\_c (C enumerator), 1159  
gapAdType\_t.gAdComplete16bitServiceList\_c (C enumerator), 1158  
gapAdType\_t.gAdComplete32bitServiceList\_c (C enumerator), 1158  
gapAdType\_t.gAdComplete128bitServiceList\_c (C enumerator), 1158  
gapAdType\_t.gAdCompleteLocalName\_c (C enumerator), 1158  
gapAdType\_t.gAdEncryptedAdvertisingData\_c (C enumerator), 1160  
gapAdType\_t.gAdFlags\_c (C enumerator), 1158  
gapAdType\_t.gAdIncomplete16bitServiceList\_c (C enumerator), 1158  
gapAdType\_t.gAdIncomplete32bitServiceList\_c (C enumerator), 1158  
gapAdType\_t.gAdIncomplete128bitServiceList\_c (C enumerator), 1158  
gapAdType\_t.gAdLeDeviceAddress\_c (C enumerator), 1160  
gapAdType\_t.gAdLeRole\_c (C enumerator), 1160  
gapAdType\_t.gAdLeSupportedFeatures\_c (C enumerator), 1160  
gapAdType\_t.gAdManufacturerSpecificData\_c (C enumerator), 1160  
gapAdType\_t.gAdPeripheralConnectionIntervalRange\_c (C enumerator), 1159  
gapAdType\_t.gAdPublicTargetAddress\_c (C enumerator), 1159  
gapAdType\_t.gAdRandomTargetAddress\_c (C enumerator), 1159  
gapAdType\_t.gAdSecurityManagerOobFlags\_c (C enumerator), 1159  
gapAdType\_t.gAdSecurityManagerTkValue\_c (C enumerator), 1159  
gapAdType\_t.gAdServiceData16bit\_c (C enumerator), 1159  
gapAdType\_t.gAdServiceData32bit\_c (C enumerator), 1159  
gapAdType\_t.gAdServiceData128bit\_c (C enumerator), 1159  
gapAdType\_t.gAdServiceSolicitationList16bit\_c (C enumerator), 1159  
gapAdType\_t.gAdServiceSolicitationList32bit\_c (C enumerator), 1159  
gapAdType\_t.gAdServiceSolicitationList128bit\_c (C enumerator), 1159  
gapAdType\_t.gAdShortenedLocalName\_c (C enumerator), 1158  
gapAdType\_t.gAdSimplePairingHashC192\_c (C enumerator), 1159  
gapAdType\_t.gAdSimplePairingHashC256\_c (C enumerator), 1160  
gapAdType\_t.gAdSimplePairingRandomizerR192\_c (C enumerator), 1159  
gapAdType\_t.gAdSimplePairingRandomizerR256\_c (C enumerator), 1160  
gapAdType\_t.gAdTxPowerLevel\_c (C enumerator), 1159  
gapAdType\_t.gAdUniformResourceIdentifier\_c (C enumerator), 1160  
gapAdTypeFlags\_t (C type), 1173  
gapAdvertisingCallback\_t (C type), 1173  
gapAdvertisingChannelMapFlags\_t (C enum), 1156  
gapAdvertisingChannelMapFlags\_t.gAdvChanMapFlag37\_c (C enumerator), 1156  
gapAdvertisingChannelMapFlags\_t.gAdvChanMapFlag38\_c (C enumerator), 1156  
gapAdvertisingChannelMapFlags\_t.gAdvChanMapFlag39\_c (C enumerator), 1156  
gapAdvertisingData\_t (C struct), 1250, 1445  
gapAdvertisingData\_t.aAdStructures (C var), 1251, 1445  
gapAdvertisingData\_t.cNumAdStructures (C var), 1251, 1445  
gapAdvertisingDecisionData\_t (C struct), 1251, 1445  
gapAdvertisingDecisionData\_t.dataLength (C var), 1251, 1445  
gapAdvertisingDecisionData\_t.pDecisionData (C var), 1251, 1445  
gapAdvertisingDecisionData\_t.pKey (C var), 1251, 1445  
gapAdvertisingDecisionData\_t.pPrand (C var), 1251, 1445  
gapAdvertisingDecisionData\_t.resolvableTagPresent (C var), 1251, 1445  
gapAdvertisingEvent\_t (C struct), 1254, 1448  
gapAdvertisingEvent\_t.eventData (C union), 1267, 1459  
gapAdvertisingEvent\_t.eventData (C var), 1254, 1448  
gapAdvertisingEvent\_t.eventData.advHandle (C var), 1268, 1460  
gapAdvertisingEvent\_t.eventData.advSetTerminated (C var), 1268, 1460  
gapAdvertisingEvent\_t.eventData.failReason (C var), 1267, 1460  
gapAdvertisingEvent\_t.eventData.perAdvResponse (C var), 1268, 1460  
gapAdvertisingEvent\_t.eventData.scanNotification (C var), 1268, 1460  
gapAdvertisingEvent\_t.eventData.subeventDataRequest (C var), 1268, 1460  
gapAdvertisingEvent\_t.eventType (C var), 1254, 1448  
gapAdvertisingEventType\_t (C enum), 1164

[gapAdvertisingEventType\\_t.gAdvertisingCommandFailed\\_c \(C enumerator\), 1164](#)  
[gapAdvertisingEventType\\_t.gAdvertisingSetTerminated\\_c \(C enumerator\), 1164](#)  
[gapAdvertisingEventType\\_t.gAdvertisingStateChanged\\_c \(C enumerator\), 1164](#)  
[gapAdvertisingEventType\\_t.gExtAdvertisingStateChanged\\_c \(C enumerator\), 1164](#)  
[gapAdvertisingEventType\\_t.gExtScanNotification\\_c \(C enumerator\), 1164](#)  
[gapAdvertisingEventType\\_t.gPerAdvResponse\\_c \(C enumerator\), 1164](#)  
[gapAdvertisingEventType\\_t.gPerAdvSubeventDataRequest\\_c \(C enumerator\), 1164](#)  
[gapAdvertisingFilterPolicy\\_t \(C enum\), 1156](#)  
[gapAdvertisingFilterPolicy\\_t.gProcessAll\\_c \(C enumerator\), 1157](#)  
[gapAdvertisingFilterPolicy\\_t.gProcessConnAllScanWL\\_c \(C enumerator\), 1157](#)  
[gapAdvertisingFilterPolicy\\_t.gProcessFilterAcceptListOnly\\_c \(C enumerator\), 1157](#)  
[gapAdvertisingFilterPolicy\\_t.gProcessScanAllConnWL\\_c \(C enumerator\), 1157](#)  
[gapAdvertisingParameters\\_t \(C struct\), 1239, 1437](#)  
[gapAdvertisingParameters\\_t.advertisingType \(C var\), 1239, 1437](#)  
[gapAdvertisingParameters\\_t.channelMap \(C var\), 1239, 1437](#)  
[gapAdvertisingParameters\\_t.filterPolicy \(C var\), 1240, 1437](#)  
[gapAdvertisingParameters\\_t.maxInterval \(C var\), 1239, 1437](#)  
[gapAdvertisingParameters\\_t.minInterval \(C var\), 1239, 1437](#)  
[gapAdvertisingParameters\\_t.ownAddressType \(C var\), 1239, 1437](#)  
[gapAdvertisingParameters\\_t.peerAddress \(C var\), 1239, 1437](#)  
[gapAdvertisingParameters\\_t.peerAddressType \(C var\), 1239, 1437](#)  
[gapAdvertisingSetTerminated\\_t \(C struct\), 1253, 1447](#)  
[gapAdvertisingSetTerminated\\_t.deviceId \(C var\), 1253, 1447](#)  
[gapAdvertisingSetTerminated\\_t.handle \(C var\), 1253, 1447](#)  
[gapAdvertisingSetTerminated\\_t.numCompletedExtAdvEvents \(C var\), 1253, 1447](#)  
[gapAdvertisingSetTerminated\\_t.status \(C var\), 1253, 1447](#)  
[gapAppearance\\_t \(C type\), 1173](#)  
[gapAppearance\\_tag \(C enum\), 1170](#)  
[gapAppearance\\_tag.gBarcodeScanner\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gBloodPressureArm\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gBloodPressureWrist\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gCardReader\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gCyclingCadenceSensor\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gCyclingComputer\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gCyclingPowerSensor\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gCyclingSpeedandCadenceSensor\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gCyclingSpeedSensor\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gDigitalPen\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gDigitizerTablet\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gFingertip\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gGamepad\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gGenericBarcodeScanner\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericBloodPressure\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericClock\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericComputer\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericCycling\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gGenericDisplay\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericEyeglasses\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericGlucoseMeter\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gGenericHeartRateSensor\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericKeyring\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericMediaPlayer\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericOutdoorSportsActivity\\_c \(C enumerator\), 1172](#)  
[gapAppearance\\_tag.gGenericPhone\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericPulseOximeter\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gGenericRemoteControl\\_c \(C enumerator\), 1170](#)  
[gapAppearance\\_tag.gGenericRunningWalkingSensor\\_c \(C enumerator\), 1171](#)  
[gapAppearance\\_tag.gGenericTag\\_c \(C enumerator\), 1170](#)

gapAppearance\_tag.gGenericThermometer\_c (C enumerator), 1170  
gapAppearance\_tag.gGenericWatch\_c (C enumerator), 1170  
gapAppearance\_tag.gGenericWeightScale\_c (C enumerator), 1172  
gapAppearance\_tag.gHeartRateSensorHeartRateBelt\_c (C enumerator), 1170  
gapAppearance\_tag.gHumanInterfaceDevice\_c (C enumerator), 1171  
gapAppearance\_tag.gJoystick\_c (C enumerator), 1171  
gapAppearance\_tag.gKeyboard\_c (C enumerator), 1171  
gapAppearance\_tag.gLocationandNavigationDisplayDevice\_c (C enumerator), 1172  
gapAppearance\_tag.gLocationAndNavigationPod\_c (C enumerator), 1172  
gapAppearance\_tag.gLocationDisplayDevice\_c (C enumerator), 1172  
gapAppearance\_tag.gLocationPod\_c (C enumerator), 1172  
gapAppearance\_tag.gMouse\_c (C enumerator), 1171  
gapAppearance\_tag.gRunningWalkingSensorInShoe\_c (C enumerator), 1171  
gapAppearance\_tag.gRunningWalkingSensorOnHip\_c (C enumerator), 1171  
gapAppearance\_tag.gRunningWalkingSensorOnShoe\_c (C enumerator), 1171  
gapAppearance\_tag.gSportsWatch\_c (C enumerator), 1170  
gapAppearance\_tag.gThermometerEar\_c (C enumerator), 1170  
gapAppearance\_tag.gUnknown\_c (C enumerator), 1170  
gapAppearance\_tag.gWristWorn\_c (C enumerator), 1171  
gapAuthenticationRejectedEvent\_t (C struct), 1261, 1453  
gapAuthenticationRejectedEvent\_t.rejectReason (C var), 1261, 1453  
gapAuthenticationRejectReason\_t (C type), 1172  
gapAutoConnectParams\_t (C struct), 1266, 1458  
gapAutoConnectParams\_t.aAutoConnectData (C var), 1267, 1458  
gapAutoConnectParams\_t.cNumAddresses (C var), 1267, 1458  
gapAutoConnectParams\_t.writeInFilterAcceptList (C var), 1267, 1458  
gapCarSupport\_t (C enum), 1169  
gapCarSupport\_t.CAR\_Supported (C enumerator), 1169  
gapCarSupport\_t.CAR\_Unavailable (C enumerator), 1169  
gapCarSupport\_t.CAR\_Unknown (C enumerator), 1169  
gapCarSupport\_t.CAR\_Unsupported (C enumerator), 1169  
gapConnCteRequestFailed\_t (C struct), 1264, 1455  
gapConnCteRequestFailed\_t.status (C var), 1264, 1455  
gapConnectedEvent\_t (C struct), 1259, 1452  
gapConnectedEvent\_t.advHandle (C var), 1260, 1452  
gapConnectedEvent\_t.connectionRole (C var), 1260, 1452  
gapConnectedEvent\_t.connParameters (C var), 1259, 1452  
gapConnectedEvent\_t.localRpa (C var), 1260, 1452  
gapConnectedEvent\_t.localRpaUsed (C var), 1260, 1452  
gapConnectedEvent\_t.peerAddress (C var), 1260, 1452  
gapConnectedEvent\_t.peerAddressType (C var), 1259, 1452  
gapConnectedEvent\_t.peerRpa (C var), 1260, 1452  
gapConnectedEvent\_t.peerRpaResolved (C var), 1260, 1452  
gapConnectedEvent\_t.syncHandle (C var), 1260, 1452  
gapConnectionCallback\_t (C type), 1173  
gapConnectionCteReceiveParams\_t (C struct), 1249, 1443  
gapConnectionCteReceiveParams\_t.aAntennaIds (C var), 1249, 1444  
gapConnectionCteReceiveParams\_t.iqSamplingEnable (C var), 1249, 1444  
gapConnectionCteReceiveParams\_t.slotDurations (C var), 1249, 1444  
gapConnectionCteReceiveParams\_t.switchingPatternLength (C var), 1249, 1444  
gapConnectionCteReqEnableParams\_t (C struct), 1249, 1444  
gapConnectionCteReqEnableParams\_t.cteReqEnable (C var), 1249, 1444  
gapConnectionCteReqEnableParams\_t.cteReqInterval (C var), 1249, 1444  
gapConnectionCteReqEnableParams\_t.requestedCteLength (C var), 1250, 1444  
gapConnectionCteReqEnableParams\_t.requestedCteType (C var), 1250, 1444  
gapConnectionCteTransmitParams\_t (C struct), 1249, 1443  
gapConnectionCteTransmitParams\_t.aAntennaIds (C var), 1249, 1443  
gapConnectionCteTransmitParams\_t.cteTypes (C var), 1249, 1443

[gapConnectionCteTransmitParams\\_t.switchingPatternLength \(C var\)](#), 1249, 1443  
[gapConnectionEvent\\_t \(C struct\)](#), 1266, 1457  
[gapConnectionEvent\\_t.eventData \(C union\)](#), 1269, 1460  
[gapConnectionEvent\\_t.eventData \(C var\)](#), 1266, 1457  
[gapConnectionEvent\\_t.eventData.authenticationRejectedEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.channelMap \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.connectedEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.connectionUpdateComplete \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.connectionUpdateRequest \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.connIqReport \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.cteRequestFailedEvent \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.disconnectedEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.eattBearerStatusNotification \(C var\)](#), 1271, 1462  
[gapConnectionEvent\\_t.eventData.eattConnectionComplete \(C var\)](#), 1271, 1462  
[gapConnectionEvent\\_t.eventData.eattConnectionRequest \(C var\)](#), 1270, 1462  
[gapConnectionEvent\\_t.eventData.eattReconfigureResponse \(C var\)](#), 1271, 1462  
[gapConnectionEvent\\_t.eventData.encryptionChangedEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.failReason \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.handoverConnectedEvent \(C var\)](#), 1271, 1462  
[gapConnectionEvent\\_t.eventData.handoverDisconnectedEvent \(C var\)](#), 1271, 1462  
[gapConnectionEvent\\_t.eventData.incomingKeypressNotification \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.keyExchangeRequestEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.keysReceivedEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.leDataLengthChanged \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.longTermKeyRequestEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.numericValueForDisplay \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.pairingCompleteEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.pairingEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.passkeyForDisplay \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.pathLossThreshold \(C var\)](#), 1270, 1462  
[gapConnectionEvent\\_t.eventData.perAdvSyncTransferStatus \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.peripheralSecurityRequestEvent \(C var\)](#), 1269, 1461  
[gapConnectionEvent\\_t.eventData.rssi\\_dBm \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventData.smError \(C var\)](#), 1271, 1462  
[gapConnectionEvent\\_t.eventData.transmitPowerInfo \(C var\)](#), 1270, 1462  
[gapConnectionEvent\\_t.eventData.transmitPowerReporting \(C var\)](#), 1270, 1462  
[gapConnectionEvent\\_t.eventData.txPowerLevel\\_dBm \(C var\)](#), 1270, 1461  
[gapConnectionEvent\\_t.eventType \(C var\)](#), 1266, 1457  
[gapConnectionEventType\\_t \(C enum\)](#), 1165  
[gapConnectionEventType\\_t.gConnEvtAuthenticationRejected\\_c \(C enumerator\)](#), 1165  
[gapConnectionEventType\\_t.gConnEvtChannelMapRead\\_c \(C enumerator\)](#), 1167  
[gapConnectionEventType\\_t.gConnEvtChannelMapReadFailure\\_c \(C enumerator\)](#), 1167  
[gapConnectionEventType\\_t.gConnEvtChanSelectionAlgorithm2\\_c \(C enumerator\)](#), 1167  
[gapConnectionEventType\\_t.gConnEvtConnected\\_c \(C enumerator\)](#), 1165  
[gapConnectionEventType\\_t.gConnEvtCteReceiveParamsSetupComplete\\_c \(C enumerator\)](#), 1167  
[gapConnectionEventType\\_t.gConnEvtCteReqStateChanged\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtCteRequestFailed\\_c \(C enumerator\)](#), 1167  
[gapConnectionEventType\\_t.gConnEvtCteRspStateChanged\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtCteTransmitParamsSetupComplete\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtDisconnected\\_c \(C enumerator\)](#), 1166  
[gapConnectionEventType\\_t.gConnEvtEattBearerStatusNotification\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtEattChannelReconfigureResponse\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtEattConnectionComplete\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtEattConnectionRequest\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtEncryptionChanged\\_c \(C enumerator\)](#), 1166  
[gapConnectionEventType\\_t.gConnEvtEnhancedReadTransmitPowerLevel\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtHandoverConnected\\_c \(C enumerator\)](#), 1168  
[gapConnectionEventType\\_t.gConnEvtIqReportReceived\\_c \(C enumerator\)](#), 1167

gapConnectionEventType\_t.gConnEvtKeyExchangeRequest\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtKeysReceived\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtLeDataLengthChanged\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtLeScDisplayNumericValue\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtLeScKeypressNotification\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtLeScOobDataRequest\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtLeSetDataLengthFailure\_c (C enumerator), 1169  
gapConnectionEventType\_t.gConnEvtLongTermKeyRequest\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtOobRequest\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtPairingAlreadyStarted\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtPairingComplete\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtPairingNoLtk\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtPairingRequest\_c (C enumerator), 1165  
gapConnectionEventType\_t.gConnEvtPairingResponse\_c (C enumerator), 1165  
gapConnectionEventType\_t.gConnEvtParameterUpdateComplete\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtParameterUpdateRequest\_c (C enumerator), 1167  
gapConnectionEventType\_t.gConnEvtPasskeyDisplay\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtPasskeyRequest\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtPathLossReportingParamsSetupComplete\_c (C enumerator), 1168  
gapConnectionEventType\_t.gConnEvtPathLossReportingStateChanged\_c (C enumerator), 1168  
gapConnectionEventType\_t.gConnEvtPathLossThreshold\_c (C enumerator), 1168  
gapConnectionEventType\_t.gConnEvtPeripheralSecurityRequest\_c (C enumerator), 1165  
gapConnectionEventType\_t.gConnEvtPowerReadFailure\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtRssiRead\_c (C enumerator), 1166  
gapConnectionEventType\_t.gConnEvtSmError\_c (C enumerator), 1169  
gapConnectionEventType\_t.gConnEvtTransmitPowerReporting\_c (C enumerator), 1168  
gapConnectionEventType\_t.gConnEvtTransmitPowerReportingStateChanged\_c (C enumerator), 1168  
gapConnectionEventType\_t.gConnEvtTxPowerLevelRead\_c (C enumerator), 1166  
gapConnectionEventType\_t.gHandoverDisconnected\_c (C enumerator), 1168  
gapConnectionFromPawrParameters\_t (C struct), 1246, 1441  
gapConnectionFromPawrParameters\_t.advHandle (C var), 1247, 1442  
gapConnectionFromPawrParameters\_t.connEventLengthMax (C var), 1247, 1442  
gapConnectionFromPawrParameters\_t.connEventLengthMin (C var), 1246, 1442  
gapConnectionFromPawrParameters\_t.connIntervalMax (C var), 1246, 1442  
gapConnectionFromPawrParameters\_t.connIntervalMin (C var), 1246, 1442  
gapConnectionFromPawrParameters\_t.connLatency (C var), 1246, 1442  
gapConnectionFromPawrParameters\_t.filterPolicy (C var), 1246, 1441  
gapConnectionFromPawrParameters\_t.initiatingPHYs (C var), 1247, 1442  
gapConnectionFromPawrParameters\_t.ownAddressType (C var), 1246, 1441  
gapConnectionFromPawrParameters\_t.peerAddress (C var), 1246, 1441  
gapConnectionFromPawrParameters\_t.peerAddressType (C var), 1246, 1441  
gapConnectionFromPawrParameters\_t.scanInterval (C var), 1246, 1441  
gapConnectionFromPawrParameters\_t.scanWindow (C var), 1246, 1441  
gapConnectionFromPawrParameters\_t.subevent (C var), 1247, 1442  
gapConnectionFromPawrParameters\_t.supervisionTimeout (C var), 1246, 1442  
gapConnectionFromPawrParameters\_t.usePeerIdentityAddress (C var), 1247, 1442  
gapConnectionlessCteTransmitParams\_t (C struct), 1248, 1442  
gapConnectionlessCteTransmitParams\_t.aAntennaIds (C var), 1248, 1443  
gapConnectionlessCteTransmitParams\_t.cteCount (C var), 1248, 1443  
gapConnectionlessCteTransmitParams\_t.cteLength (C var), 1248, 1443  
gapConnectionlessCteTransmitParams\_t.cteType (C var), 1248, 1443  
gapConnectionlessCteTransmitParams\_t.handle (C var), 1248, 1443  
gapConnectionlessCteTransmitParams\_t.switchingPatternLength (C var), 1248, 1443  
gapConnectionlessIqReport\_t (C struct), 1258, 1451  
gapConnectionlessIqReport\_t.aI\_samples (C var), 1259, 1451  
gapConnectionlessIqReport\_t.aQ\_samples (C var), 1259, 1451

[gapConnectionlessIqReport\\_t.channelIndex \(C var\)](#), [1258](#), [1451](#)  
[gapConnectionlessIqReport\\_t.cteType \(C var\)](#), [1259](#), [1451](#)  
[gapConnectionlessIqReport\\_t.packetStatus \(C var\)](#), [1259](#), [1451](#)  
[gapConnectionlessIqReport\\_t.periodicEventCounter \(C var\)](#), [1259](#), [1451](#)  
[gapConnectionlessIqReport\\_t.rssi \(C var\)](#), [1258](#), [1451](#)  
[gapConnectionlessIqReport\\_t.rssiAntennaId \(C var\)](#), [1259](#), [1451](#)  
[gapConnectionlessIqReport\\_t.sampleCount \(C var\)](#), [1259](#), [1451](#)  
[gapConnectionlessIqReport\\_t.slotDurations \(C var\)](#), [1259](#), [1451](#)  
[gapConnectionlessIqReport\\_t.syncHandle \(C var\)](#), [1258](#), [1451](#)  
[gapConnectionlessIqSamplingParams\\_t \(C struct\)](#), [1248](#), [1443](#)  
[gapConnectionlessIqSamplingParams\\_t.aAntennaIds \(C var\)](#), [1248](#), [1443](#)  
[gapConnectionlessIqSamplingParams\\_t.iqSamplingEnable \(C var\)](#), [1248](#), [1443](#)  
[gapConnectionlessIqSamplingParams\\_t.maxSampledCtes \(C var\)](#), [1248](#), [1443](#)  
[gapConnectionlessIqSamplingParams\\_t.slotDurations \(C var\)](#), [1248](#), [1443](#)  
[gapConnectionlessIqSamplingParams\\_t.switchingPatternLength \(C var\)](#), [1248](#), [1443](#)  
[gapConnectionParameters\\_t \(C struct\)](#), [1247](#), [1442](#)  
[gapConnectionParameters\\_t.centralClockAccuracy \(C var\)](#), [1247](#), [1442](#)  
[gapConnectionParameters\\_t.connInterval \(C var\)](#), [1247](#), [1442](#)  
[gapConnectionParameters\\_t.connLatency \(C var\)](#), [1247](#), [1442](#)  
[gapConnectionParameters\\_t.supervisionTimeout \(C var\)](#), [1247](#), [1442](#)  
[gapConnectionRequestParameters\\_t \(C struct\)](#), [1244](#), [1440](#)  
[gapConnectionRequestParameters\\_t.connEventLengthMax \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.connEventLengthMin \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.connIntervalMax \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.connIntervalMin \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.connLatency \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.filterPolicy \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.initiatingPHYS \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.ownAddressType \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.peerAddress \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.peerAddressType \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.scanInterval \(C var\)](#), [1244](#), [1441](#)  
[gapConnectionRequestParameters\\_t.scanWindow \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.supervisionTimeout \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionRequestParameters\\_t.usePeerIdentityAddress \(C var\)](#), [1245](#), [1441](#)  
[gapConnectionSecurityInformation\\_t \(C struct\)](#), [1238](#), [1436](#)  
[gapConnectionSecurityInformation\\_t.authenticated \(C var\)](#), [1238](#), [1436](#)  
[gapConnectionSecurityInformation\\_t.authorizedToRead \(C var\)](#), [1238](#), [1436](#)  
[gapConnectionSecurityInformation\\_t.authorizedToWrite \(C var\)](#), [1238](#), [1436](#)  
[gapConnIqReport\\_t \(C struct\)](#), [1263](#), [1454](#)  
[gapConnIqReport\\_t.aI\\_samples \(C var\)](#), [1264](#), [1455](#)  
[gapConnIqReport\\_t.aQ\\_samples \(C var\)](#), [1264](#), [1455](#)  
[gapConnIqReport\\_t.connEventCounter \(C var\)](#), [1264](#), [1455](#)  
[gapConnIqReport\\_t.cteType \(C var\)](#), [1263](#), [1455](#)  
[gapConnIqReport\\_t.dataChannelIndex \(C var\)](#), [1263](#), [1454](#)  
[gapConnIqReport\\_t.packetStatus \(C var\)](#), [1264](#), [1455](#)  
[gapConnIqReport\\_t.rssi \(C var\)](#), [1263](#), [1454](#)  
[gapConnIqReport\\_t.rssiAntennaId \(C var\)](#), [1263](#), [1454](#)  
[gapConnIqReport\\_t.rxPhy \(C var\)](#), [1263](#), [1454](#)  
[gapConnIqReport\\_t.sampleCount \(C var\)](#), [1264](#), [1455](#)  
[gapConnIqReport\\_t.slotDurations \(C var\)](#), [1263](#), [1455](#)  
[gapConnLeDataLengthChanged\\_t \(C struct\)](#), [1263](#), [1454](#)  
[gapConnLeDataLengthChanged\\_t.maxRxOctets \(C var\)](#), [1263](#), [1454](#)  
[gapConnLeDataLengthChanged\\_t.maxRxTime \(C var\)](#), [1263](#), [1454](#)  
[gapConnLeDataLengthChanged\\_t.maxTxOctets \(C var\)](#), [1263](#), [1454](#)  
[gapConnLeDataLengthChanged\\_t.maxTxTime \(C var\)](#), [1263](#), [1454](#)  
[gapConnParamsUpdateComplete\\_t \(C struct\)](#), [1262](#), [1454](#)  
[gapConnParamsUpdateComplete\\_t.connInterval \(C var\)](#), [1262](#), [1454](#)

gapConnParamsUpdateComplete\_t.connLatency (C var), 1262, 1454  
gapConnParamsUpdateComplete\_t.status (C var), 1454  
gapConnParamsUpdateComplete\_t.supervisionTimeout (C var), 1262, 1454  
gapConnParamsUpdateReq\_t (C struct), 1262, 1453  
gapConnParamsUpdateReq\_t.intervalMax (C var), 1262, 1454  
gapConnParamsUpdateReq\_t.intervalMin (C var), 1262, 1454  
gapConnParamsUpdateReq\_t.peripheralLatency (C var), 1262, 1454  
gapConnParamsUpdateReq\_t.timeoutMultiplier (C var), 1262, 1454  
gapControllerTestCmd\_t (C enum), 1163  
gapControllerTestCmd\_t.gControllerTestCmdEnd\_c (C enumerator), 1163  
gapControllerTestCmd\_t.gControllerTestCmdStartRx\_c (C enumerator), 1163  
gapControllerTestCmd\_t.gControllerTestCmdStartTx\_c (C enumerator), 1163  
gapControllerTestEvent\_t (C struct), 1384, 1417  
gapControllerTestEvent\_t.receivedPackets (C var), 1418  
gapControllerTestEvent\_t.testEventType (C var), 1418  
gapControllerTestEventType\_t (C enum), 1357  
gapControllerTestEventType\_t.gControllerReceiverTestStarted\_c (C enumerator), 1357  
gapControllerTestEventType\_t.gControllerTestEnded\_c (C enumerator), 1357  
gapControllerTestEventType\_t.gControllerTransmitterTestStarted\_c (C enumerator), 1357  
gapControllerTestTxType\_t (C type), 1173  
gapControllerTestTxType\_tag (C enum), 1163  
gapControllerTestTxType\_tag.gControllerTestTx00\_c (C enumerator), 1163  
gapControllerTestTxType\_tag.gControllerTestTx0F\_c (C enumerator), 1163  
gapControllerTestTxType\_tag.gControllerTestTx55\_c (C enumerator), 1163  
gapControllerTestTxType\_tag.gControllerTestTxAA\_c (C enumerator), 1163  
gapControllerTestTxType\_tag.gControllerTestTxF0\_c (C enumerator), 1163  
gapControllerTestTxType\_tag.gControllerTestTxFF\_c (C enumerator), 1163  
gapControllerTestTxType\_tag.gControllerTestTxPrbs9\_c (C enumerator), 1163  
gapControllerTestTxType\_tag.gControllerTestTxPrbs15\_c (C enumerator), 1163  
gapCreateSyncReqFilterPolicy\_t (C type), 1173  
gapCreateSyncReqFilterPolicy\_tag (C enum), 1157  
gapCreateSyncReqFilterPolicy\_tag.gUseCommandParameters\_c (C enumerator), 1158  
gapCreateSyncReqFilterPolicy\_tag.gUsePeriodicAdvList\_c (C enumerator), 1158  
gapCreateSyncReqOptions\_t (C type), 1173  
gapCreateSyncReqOptions\_tag (C struct), 1244, 1440  
gapCreateSyncReqOptions\_tag.duplicateFilteringEnabled (C var), 1440  
gapCreateSyncReqOptions\_tag.filterPolicy (C var), 1440  
gapCreateSyncReqOptions\_tag.reportingEnabled (C var), 1440  
gapDecisionInstructionsAdvAChecks\_t (C enum), 1162  
gapDecisionInstructionsAdvAChecks\_t.gDIAAC\_AdvAinFilterAcceptList\_c (C enumerator), 1162  
gapDecisionInstructionsAdvAChecks\_t.gDIAAC\_AdvAmatchAddress1\_c (C enumerator), 1162  
gapDecisionInstructionsAdvAChecks\_t.gDIAAC\_AdvAmatchAddress1orAddress2\_c (C enumerator), 1162  
gapDecisionInstructionsAdvMode\_t (C type), 1173  
gapDecisionInstructionsData\_t (C type), 1173  
gapDecisionInstructionsData\_tag (C struct), 1251, 1445  
gapDecisionInstructionsData\_tag.passCriteria (C var), 1445  
gapDecisionInstructionsData\_tag.relevantField (C var), 1445  
gapDecisionInstructionsData\_tag.testGroup (C var), 1445  
gapDecisionInstructionsData\_tag.testParameters (C union), 1267, 1458  
gapDecisionInstructionsData\_tag.testParameters (C var), 1445  
gapDecisionInstructionsData\_tag.testParameters.advA (C struct), 1267, 1459  
gapDecisionInstructionsData\_tag.testParameters.advA (C var), 1267, 1458  
gapDecisionInstructionsData\_tag.testParameters.advA.address1 (C var), 1459  
gapDecisionInstructionsData\_tag.testParameters.advA.address1Type (C var), 1459  
gapDecisionInstructionsData\_tag.testParameters.advA.address2 (C var), 1459  
gapDecisionInstructionsData\_tag.testParameters.advA.address2Type (C var), 1459  
gapDecisionInstructionsData\_tag.testParameters.advA.check (C var), 1459

[gapDecisionInstructionsData\\_tag.testParameters.advMode \(C var\), 1267, 1458](#)  
[gapDecisionInstructionsData\\_tag.testParameters.arbitraryData \(C struct\), 1267, 1458](#)  
[gapDecisionInstructionsData\\_tag.testParameters.arbitraryData \(C var\), 1267, 1458](#)  
[gapDecisionInstructionsData\\_tag.testParameters.arbitraryData.mask \(C var\), 1459](#)  
[gapDecisionInstructionsData\\_tag.testParameters.arbitraryData.target \(C var\), 1459](#)  
[gapDecisionInstructionsData\\_tag.testParameters.pathLoss \(C struct\), 1267, 1459](#)  
[gapDecisionInstructionsData\\_tag.testParameters.pathLoss \(C var\), 1267, 1458](#)  
[gapDecisionInstructionsData\\_tag.testParameters.pathLoss.max \(C var\), 1459](#)  
[gapDecisionInstructionsData\\_tag.testParameters.pathLoss.min \(C var\), 1459](#)  
[gapDecisionInstructionsData\\_tag.testParameters.resolvableTagKey \(C var\), 1267, 1458](#)  
[gapDecisionInstructionsData\\_tag.testParameters.rssi \(C struct\), 1267, 1459](#)  
[gapDecisionInstructionsData\\_tag.testParameters.rssi \(C var\), 1267, 1458](#)  
[gapDecisionInstructionsData\\_tag.testParameters.rssi.max \(C var\), 1459](#)  
[gapDecisionInstructionsData\\_tag.testParameters.rssi.min \(C var\), 1459](#)  
[gapDecisionInstructionsRelevantField\\_t \(C enum\), 1160](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_AdvAddress\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_AdvMode\\_c \(C enumerator\), 1160](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_1Byte\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_2Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_3Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_4Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_5Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_6Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_7Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtLeast\\_8Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_1Byte\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_2Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_3Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_4Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_5Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_6Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_7Bytes\\_c \(C enumerator\), 1162](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfAtMost\\_8Bytes\\_c \(C enumerator\), 1162](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfExactly\\_1Byte\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfExactly\\_2Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfExactly\\_3Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfExactly\\_4Bytes\\_c \(C enumerator\), 1161](#)  
[gapDecisionInstructionsRelevantField\\_t.gDIRF\\_ArbitraryDataOfExactly\\_5Bytes\\_c \(C enumerator\), 1161](#)

[1161](#)  
gapDecisionInstructionsRelevantField\_t.gDIRF\_ArbitraryDataOfExactly\_6Bytes\_c (C enumerator), [1161](#)  
gapDecisionInstructionsRelevantField\_t.gDIRF\_ArbitraryDataOfExactly\_7Bytes\_c (C enumerator), [1161](#)  
gapDecisionInstructionsRelevantField\_t.gDIRF\_ArbitraryDataOfExactly\_8Bytes\_c (C enumerator), [1161](#)  
gapDecisionInstructionsRelevantField\_t.gDIRF\_PathLoss\_c (C enumerator), [1160](#)  
gapDecisionInstructionsRelevantField\_t.gDIRF\_ResolvableTag\_c (C enumerator), [1160](#)  
gapDecisionInstructionsRelevantField\_t.gDIRF\_RSSI\_c (C enumerator), [1160](#)  
gapDecisionInstructionsTestGroup\_t (C enum), [1162](#)  
gapDecisionInstructionsTestGroup\_t.gDITG\_NewTestGroup\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestGroup\_t.gDITG\_SameTestGroup\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t (C enum), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_Always\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_CheckFails\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_CheckFailsOrRelevantFieldNotPresent\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_CheckPasses\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_CheckPassesOrRelevantFieldNotPresent\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_Never\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_RelevantFieldNotPresent\_c (C enumerator), [1162](#)  
gapDecisionInstructionsTestPassCriteria\_t.gDITPC\_RelevantFieldPresent\_c (C enumerator), [1162](#)  
gapDeviceSecurityRequirements\_t (C struct), [1237](#), [1435](#)  
gapDeviceSecurityRequirements\_t.aServiceSecurityRequirements (C var), [1237](#), [1436](#)  
gapDeviceSecurityRequirements\_t.cNumServices (C var), [1237](#), [1436](#)  
gapDeviceSecurityRequirements\_t.pSecurityRequirements (C var), [1237](#), [1436](#)  
gapDisconnectedEvent\_t (C struct), [1262](#), [1453](#)  
gapDisconnectedEvent\_t.reason (C var), [1262](#), [1453](#)  
gapDisconnectionReason\_t (C type), [1173](#)  
gapEattBearerStatus\_t (C enum), [1169](#)  
gapEattBearerStatus\_t.gEnhancedBearerActive\_c (C enumerator), [1169](#)  
gapEattBearerStatus\_t.gEnhancedBearerDisconnected\_c (C enumerator), [1169](#)  
gapEattBearerStatus\_t.gEnhancedBearerNoPeerCredits\_c (C enumerator), [1169](#)  
gapEattBearerStatus\_t.gEnhancedBearerStatusEnd\_c (C enumerator), [1169](#)  
gapEattBearerStatus\_t.gEnhancedBearerSuspendedNoLocalCredits\_c (C enumerator), [1169](#)  
gapEattBearerStatusNotification\_t (C struct), [1265](#), [1457](#)  
gapEattBearerStatusNotification\_t.bearerId (C var), [1457](#)  
gapEattBearerStatusNotification\_t.status (C var), [1457](#)  
gapEattConnectionComplete\_t (C struct), [1265](#), [1456](#)  
gapEattConnectionComplete\_t.aBearerIds (C var), [1456](#)  
gapEattConnectionComplete\_t.cBearers (C var), [1456](#)  
gapEattConnectionComplete\_t.mtu (C var), [1456](#)  
gapEattConnectionComplete\_t.status (C var), [1456](#)  
gapEattConnectionRequest\_t (C struct), [1265](#), [1456](#)  
gapEattConnectionRequest\_t.cBearers (C var), [1456](#)  
gapEattConnectionRequest\_t.initialCredits (C var), [1456](#)  
gapEattConnectionRequest\_t.mtu (C var), [1456](#)  
gapEattReconfigureResponse\_t (C struct), [1265](#), [1456](#)  
gapEattReconfigureResponse\_t.aBearerIds (C var), [1457](#)  
gapEattReconfigureResponse\_t.cBearers (C var), [1456](#)  
gapEattReconfigureResponse\_t.localMtu (C var), [1456](#)  
gapEattReconfigureResponse\_t.status (C var), [1456](#)  
gapEncryptionChangedEvent\_t (C struct), [1261](#), [1453](#)  
gapEncryptionChangedEvent\_t.newEncryptionState (C var), [1262](#), [1453](#)  
gapExtAdvertisingParameters\_t (C type), [1172](#)

[gapExtAdvertisingParameters\\_tag \(C struct\), 1240, 1437](#)  
[gapExtAdvertisingParameters\\_tag.channelMap \(C var\), 1240, 1438](#)  
[gapExtAdvertisingParameters\\_tag.enableScanReqNotification \(C var\), 1241, 1438](#)  
[gapExtAdvertisingParameters\\_tag.extAdvProperties \(C var\), 1241, 1438](#)  
[gapExtAdvertisingParameters\\_tag.filterPolicy \(C var\), 1240, 1438](#)  
[gapExtAdvertisingParameters\\_tag.handle \(C var\), 1240, 1437](#)  
[gapExtAdvertisingParameters\\_tag.maxInterval \(C var\), 1240, 1437](#)  
[gapExtAdvertisingParameters\\_tag.minInterval \(C var\), 1240, 1437](#)  
[gapExtAdvertisingParameters\\_tag.ownAddressType \(C var\), 1240, 1438](#)  
[gapExtAdvertisingParameters\\_tag.ownRandomAddr \(C var\), 1240, 1438](#)  
[gapExtAdvertisingParameters\\_tag.peerAddress \(C var\), 1240, 1438](#)  
[gapExtAdvertisingParameters\\_tag.peerAddressType \(C var\), 1240, 1438](#)  
[gapExtAdvertisingParameters\\_tag.primaryAdvPhyOptions \(C var\), 1241, 1438](#)  
[gapExtAdvertisingParameters\\_tag.primaryPHY \(C var\), 1241, 1438](#)  
[gapExtAdvertisingParameters\\_tag.secondaryAdvMaxSkip \(C var\), 1241, 1438](#)  
[gapExtAdvertisingParameters\\_tag.secondaryAdvPhyOptions \(C var\), 1241, 1438](#)  
[gapExtAdvertisingParameters\\_tag.secondaryPHY \(C var\), 1241, 1438](#)  
[gapExtAdvertisingParameters\\_tag.SID \(C var\), 1240, 1437](#)  
[gapExtAdvertisingParameters\\_tag.txPower \(C var\), 1241, 1438](#)  
[gapExtScannedDevice\\_t \(C struct\), 1255, 1448](#)  
[gapExtScannedDevice\\_t.aAddress \(C var\), 1255, 1449](#)  
[gapExtScannedDevice\\_t.addressType \(C var\), 1255, 1449](#)  
[gapExtScannedDevice\\_t.advertisingAddressResolved \(C var\), 1255, 1449](#)  
[gapExtScannedDevice\\_t.advEventProperties \(C var\), 1255, 1449](#)  
[gapExtScannedDevice\\_t.dataLength \(C var\), 1256, 1449](#)  
[gapExtScannedDevice\\_t.directRpa \(C var\), 1256, 1449](#)  
[gapExtScannedDevice\\_t.directRpaType \(C var\), 1256, 1449](#)  
[gapExtScannedDevice\\_t.directRpaUsed \(C var\), 1256, 1449](#)  
[gapExtScannedDevice\\_t.pData \(C var\), 1256, 1449](#)  
[gapExtScannedDevice\\_t.periodicAdvInterval \(C var\), 1256, 1449](#)  
[gapExtScannedDevice\\_t.primaryPHY \(C var\), 1255, 1449](#)  
[gapExtScannedDevice\\_t.rssi \(C var\), 1255, 1449](#)  
[gapExtScannedDevice\\_t.secondaryPHY \(C var\), 1256, 1449](#)  
[gapExtScannedDevice\\_t.SID \(C var\), 1255, 1449](#)  
[gapExtScannedDevice\\_t.txPower \(C var\), 1255, 1449](#)  
[gapExtScanNotification\\_t \(C struct\), 1253, 1446](#)  
[gapExtScanNotification\\_t.aScannerAddr \(C var\), 1253, 1447](#)  
[gapExtScanNotification\\_t.handle \(C var\), 1253, 1447](#)  
[gapExtScanNotification\\_t.scannerAddrResolved \(C var\), 1253, 1447](#)  
[gapExtScanNotification\\_t.scannerAddrType \(C var\), 1253, 1447](#)  
[gapFilterDuplicates\\_t \(C enum\), 1157](#)  
[gapFilterDuplicates\\_t.gGapDuplicateFilteringDisable\\_c \(C enumerator\), 1157](#)  
[gapFilterDuplicates\\_t.gGapDuplicateFilteringEnable\\_c \(C enumerator\), 1157](#)  
[gapFilterDuplicates\\_t.gGapDuplicateFilteringPeriodicEnable\\_c \(C enumerator\), 1157](#)  
[gapGenerateDHKeyV2Params\\_t \(C struct\), 1247, 1442](#)  
[gapGenerateDHKeyV2Params\\_t.keyType \(C var\), 1248, 1442](#)  
[gapGenerateDHKeyV2Params\\_t.remoteP256PublicKey \(C var\), 1247, 1442](#)  
[gapGenericCallback\\_t \(C type\), 1366](#)  
[gapGenericEvent\\_t \(C struct\), 1397, 1429](#)  
[gapGenericEvent\\_t.eventData \(C union\), 1399, 1432](#)  
[gapGenericEvent\\_t.eventData \(C var\), 1397, 1429](#)  
[gapGenericEvent\\_t.eventData.aAddress \(C var\), 1399, 1432](#)  
[gapGenericEvent\\_t.eventData.aControllerLocalRPA \(C var\), 1400, 1433](#)  
[gapGenericEvent\\_t.eventData.addrReady \(C var\), 1399, 1432](#)  
[gapGenericEvent\\_t.eventData.advHandle \(C var\), 1399, 1432](#)  
[gapGenericEvent\\_t.eventData.advTxPowerLevel\\_dBm \(C var\), 1399, 1432](#)  
[gapGenericEvent\\_t.eventData.antennaInformation \(C var\), 1400, 1433](#)  
[gapGenericEvent\\_t.eventData.bondCreatedEvent \(C var\), 1400, 1433](#)

gapGenericEvent\_\_t.eventData.deviceId (*C var*), 1400, 1432  
gapGenericEvent\_\_t.eventData.filterAcceptListSize (*C var*), 1399, 1432  
gapGenericEvent\_\_t.eventData.gapRemoteVersionInfoRead (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.gapSkdReport (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.getConnParams (*C var*), 1400, 1433  
gapGenericEvent\_\_t.eventData.handoverAnchorMonitor (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverAnchorMonitorPacket (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.handoverAnchorMonitorPacketContinue (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.handoverAnchorNotificationStateChanged (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.handoverAnchorSearchStart (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverAnchorSearchStop (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverApplyConnectionUpdateProcedure (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.handoverConnect (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverConnectionUpdateProcedure (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.handoverConnParamUpdate (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.handoverGetCsLlContext (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverGetData (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverGetTime (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverLlPendingDataIndication (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.handoverResumeTransmitComplete (*C var*), 1401, 1434  
gapGenericEvent\_\_t.eventData.handoverSetData (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverSuspendTransmitComplete (*C var*), 1401, 1434  
gapGenericEvent\_\_t.eventData.handoverTimeSync (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.handoverUpdateConnParams (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.initCompleteData (*C var*), 1400, 1432  
gapGenericEvent\_\_t.eventData.internalError (*C var*), 1399, 1432  
gapGenericEvent\_\_t.eventData.leGenerateDhKeyCompleteEvent (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.localOobData (*C var*), 1399, 1432  
gapGenericEvent\_\_t.eventData.newControllerPrivacyState (*C var*), 1399, 1432  
gapGenericEvent\_\_t.eventData.newHostPrivacyState (*C var*), 1399, 1432  
gapGenericEvent\_\_t.eventData.notifEvent (*C var*), 1400, 1432  
gapGenericEvent\_\_t.eventData.pawrAdvHandle (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.pawrSyncHandle (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.perAdvSetDefaultPerAdvSyncTransferParams (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.perAdvSetInfoTransfer (*C var*), 1400, 1433  
gapGenericEvent\_\_t.eventData.perAdvSetSyncTransferParams (*C var*), 1400, 1433  
gapGenericEvent\_\_t.eventData.perAdvSyncTransfer (*C var*), 1400, 1433  
gapGenericEvent\_\_t.eventData.perAdvSyncTransferEnable (*C var*), 1400, 1433  
gapGenericEvent\_\_t.eventData.perAdvSyncTransferReceived (*C var*), 1401, 1433  
gapGenericEvent\_\_t.eventData.phyEvent (*C var*), 1400, 1432  
gapGenericEvent\_\_t.eventData.setupFailError (*C var*), 1399, 1432  
gapGenericEvent\_\_t.eventData.syncHandle (*C var*), 1400, 1433  
gapGenericEvent\_\_t.eventData.testEvent (*C var*), 1400, 1432  
gapGenericEvent\_\_t.eventData.txPowerLevelSetStatus (*C var*), 1400, 1432  
gapGenericEvent\_\_t.eventData.unitaryTestData (*C var*), 1402, 1434  
gapGenericEvent\_\_t.eventData.verified (*C var*), 1399, 1432  
gapGenericEvent\_\_t.eventType (*C var*), 1397, 1429  
gapGenericEventType\_\_t (*C enum*), 1345  
gapGenericEventType\_\_t.gAdvertisingDataSetupComplete\_\_c (*C enumerator*), 1345  
gapGenericEventType\_\_t.gAdvertisingParametersSetupComplete\_\_c (*C enumerator*), 1345  
gapGenericEventType\_\_t.gAdvertisingSetupFailed\_\_c (*C enumerator*), 1345  
gapGenericEventType\_\_t.gAdvTxPowerLevelRead\_\_c (*C enumerator*), 1346  
gapGenericEventType\_\_t.gAntennaInformationRead\_\_c (*C enumerator*), 1348  
gapGenericEventType\_\_t.gBondCreatedEvent\_\_c (*C enumerator*), 1347  
gapGenericEventType\_\_t.gChannelMapSet\_\_c (*C enumerator*), 1347  
gapGenericEventType\_\_t.gConnectionlessCteTransmitParamsSetupComplete\_\_c (*C enumerator*), 1347  
gapGenericEventType\_\_t.gConnectionlessCteTransmitStateChanged\_\_c (*C enumerator*), 1347  
gapGenericEventType\_\_t.gConnectionlessIqSamplingStateChanged\_\_c (*C enumerator*), 1347

[gapGenericEventType\\_t.gConnEvtLeGenerateDhKeyComplete\\_c \(C enumerator\), 1348](#)  
[gapGenericEventType\\_t.gControllerLocalRPAREad\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gControllerNotificationEvent\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gControllerPrivacyStateChanged\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gControllerTestEvent\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gCreateConnectionCanceled\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gDecisionInstructionsSetupComplete\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gDeInitializationComplete\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gDeviceAddedToFilterAcceptList\\_c \(C enumerator\), 1345](#)  
[gapGenericEventType\\_t.gDeviceRemovedFromFilterAcceptList\\_c \(C enumerator\), 1345](#)  
[gapGenericEventType\\_t.gExtAdvertisingDataSetupComplete\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gExtAdvertisingDecisionDataSetupComplete\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gExtAdvertisingParametersSetupComplete\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gExtAdvertisingSetRemoveComplete\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gFilterAcceptListCleared\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gFilterAcceptListSizeRead\\_c \(C enumerator\), 1345](#)  
[gapGenericEventType\\_t.gGetConnParamsComplete\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gHandoverAnchorMonitorEvent\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverAnchorMonitorPacketContinueEvent\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gHandoverAnchorMonitorPacketEvent\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverAnchorNotificationStateChanged\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverAnchorSearchStarted\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverAnchorSearchStopped\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverApplyConnectionUpdateProcedureComplete\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gHandoverConnectionUpdateProcedureEvent\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gHandoverConnParamUpdateEvent\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverFreeComplete\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gHandoverGetComplete\\_c \(C enumerator\), 1348](#)  
[gapGenericEventType\\_t.gHandoverGetCsLlContextComplete\\_c \(C enumerator\), 1348](#)  
[gapGenericEventType\\_t.gHandoverGetTime\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverLlPendingData\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gHandoverResumeTransmitComplete\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverSetComplete\\_c \(C enumerator\), 1348](#)  
[gapGenericEventType\\_t.gHandoverSetCsLlContextComplete\\_c \(C enumerator\), 1348](#)  
[gapGenericEventType\\_t.gHandoverSuspendTransmitComplete\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverTimeSyncEvent\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverTimeSyncReceiveComplete\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverTimeSyncTransmitStateChanged\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gHandoverUpdateConnParamsComplete\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gHostPrivacyStateChanged\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gInitializationComplete\\_c \(C enumerator\), 1345](#)  
[gapGenericEventType\\_t.gInternalError\\_c \(C enumerator\), 1345](#)  
[gapGenericEventType\\_t.gLeChannelOverrideComplete\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gLePhyEvent\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gLeScLocalOobData\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gLeScPublicKeyRegenerated\\_c \(C enumerator\), 1346](#)  
[gapGenericEventType\\_t.gLeSetDataRelatedAddressChangesComplete\\_c \(C enumerator\), 1350](#)  
[gapGenericEventType\\_t.gLeSetSchedulerPriorityComplete\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gLlSkdReportEvent\\_c \(C enumerator\), 1349](#)  
[gapGenericEventType\\_t.gModifiedSleepClockAccuracy\\_c \(C enumerator\), 1348](#)  
[gapGenericEventType\\_t.gPeriodicAdvCreateSyncCancelled\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gPeriodicAdvDataSetupComplete\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gPeriodicAdvertisingStateChanged\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gPeriodicAdvListUpdateComplete\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gPeriodicAdvParamSetupComplete\\_c \(C enumerator\), 1347](#)  
[gapGenericEventType\\_t.gPeriodicAdvRecvEnableComplete\\_c \(C enumerator\), 1348](#)  
[gapGenericEventType\\_t.gPeriodicAdvSetInfoTransferComplete\\_c \(C enumerator\), 1348](#)

gapGenericEventType\_t.gPeriodicAdvSetResponseDataComplete\_c (C enumerator), 1350  
gapGenericEventType\_t.gPeriodicAdvSetSubeventDataComplete\_c (C enumerator), 1350  
gapGenericEventType\_t.gPeriodicAdvSyncTransferComplete\_c (C enumerator), 1348  
gapGenericEventType\_t.gPeriodicAdvSyncTransferFailed\_c (C enumerator), 1348  
gapGenericEventType\_t.gPeriodicAdvSyncTransferSucceeded\_c (C enumerator), 1348  
gapGenericEventType\_t.gPeriodicSyncSubeventComplete\_c (C enumerator), 1350  
gapGenericEventType\_t.gPrivateResolvableAddressVerified\_c (C enumerator), 1346  
gapGenericEventType\_t.gPublicAddressRead\_c (C enumerator), 1346  
gapGenericEventType\_t.gRandomAddressReady\_c (C enumerator), 1346  
gapGenericEventType\_t.gRandomAddressSet\_c (C enumerator), 1346  
gapGenericEventType\_t.gRemoteVersionInformationRead\_c (C enumerator), 1349  
gapGenericEventType\_t.gSetDefaultPeriodicAdvSyncTransferParamsComplete\_c (C enumerator), 1348  
gapGenericEventType\_t.gSetPeriodicAdvSyncTransferParamsComplete\_c (C enumerator), 1348  
gapGenericEventType\_t.gTxEntryAvailable\_c (C enumerator), 1347  
gapGenericEventType\_t.gTxPowerLevelSetComplete\_c (C enumerator), 1346  
gapGenericEventType\_t.gVendorUnitaryTestComplete\_c (C enumerator), 1350  
gapHandleList\_t (C struct), 1237, 1436  
gapHandleList\_t.aHandles (C var), 1237, 1436  
gapHandleList\_t.cNumHandles (C var), 1237, 1436  
gapHandoverConnectedEvent\_t (C struct), 1265, 1457  
gapHandoverConnectedEvent\_t.connectionRole (C var), 1266, 1457  
gapHandoverConnectedEvent\_t.peerAddress (C var), 1266, 1457  
gapHandoverConnectedEvent\_t.peerAddressType (C var), 1266, 1457  
gapHandoverDisconnectedEvent\_t (C struct), 1266, 1457  
gapHandoverDisconnectedEvent\_t.status (C var), 1266, 1457  
gapHostVersion\_t (C type), 1173  
gapHostVersion\_tag (C struct), 1267, 1458  
gapHostVersion\_tag.bleHostVerMajor (C var), 1458  
gapHostVersion\_tag.bleHostVerMinor (C var), 1458  
gapHostVersion\_tag.bleHostVerPatch (C var), 1458  
gapIdentityInformation\_t (C struct), 1266, 1457  
gapIdentityInformation\_t.identityAddress (C var), 1266, 1458  
gapIdentityInformation\_t.irk (C var), 1266, 1458  
gapIdentityInformation\_t.privacyMode (C var), 1266, 1458  
gapInitComplete\_t (C struct), 1385, 1418  
gapInitComplete\_t.maxAdvDataSize (C var), 1419  
gapInitComplete\_t.numOfSupportedAdvSets (C var), 1419  
gapInitComplete\_t.periodicAdvListSize (C var), 1419  
gapInitComplete\_t.supportedFeatures (C var), 1419  
gapInternalError\_t (C struct), 1384, 1417  
gapInternalError\_t.errorCode (C var), 1384, 1417  
gapInternalError\_t.errorSource (C var), 1384, 1417  
gapInternalError\_t.hciCommandOpcode (C var), 1384, 1417  
gapInternalErrorSource\_t (C enum), 1350  
gapInternalErrorSource\_t.gAddDeviceToFilterAcceptList\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gAddNewConnection\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gCancelCreateConnection\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gCheckPrivateResolvableAddress\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gClearFilterAcceptList\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gConnect\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gConnectionCteReqEnable\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gConnectionCteRspEnable\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gCreateRandomAddress\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gDefaultPairingProcedure\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gDenyLongTermKey\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gDisconnect\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gEattConnectionAccept\_c (C enumerator), 1354

gapInternalErrorSource\_t.gEattConnectionRequest\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEattDisconnectRequest\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEattL2caCancelConnection\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEattL2caDisconnectLePsm\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEattL2caEnhancedCancelConnection\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gEattL2caEnhancedReconfigureReq\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gEattL2caHandleRecvLeCbData\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gEattL2caHandleSendLeCbData\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gEattL2caSendLeFlowControlCredit\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEattReconfigureRequest\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEattSendCreditsRequest\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEnableControllerPrivacy\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gEnableHostPrivacy\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gEnableLdmTimer\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gEncryptLink\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gEnhancedReadTransmitPowerLevel\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gEnterPasskey\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gExtAdvReportProcess\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gGenerateDHKeyV2\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gGetConnParams\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandover\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverAnchorNotifHciTx\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverAnchorNotifLl\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverAnchorSearchStartHciTx\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverAnchorSearchStartLl\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverAnchorSearchStopHciTx\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverAnchorSearchStopLl\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gHandoverApplyConnectionUpdateProcedure\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gHandoverConnect\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverDisconnect\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverGetCsLlContext\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverGetLlContext\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverGetTimeLl\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverResumeTransmitHciTx\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverResumeTransmitLl\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverSetCsLlContext\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverSetLlPendingData\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gHandoverSetSkd\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverSuspendTransmitHciTx\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverSuspendTransmitHostTxInProgress\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverSuspendTransmitLl\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverSuspendTransmitLlProcInProgress\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHandoverTimeSyncRxHciTx\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverTimeSyncRxLl\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gHandoverTimeSyncTxHciTx\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gHandoverTimeSyncTxLl\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gHandoverUpdateConnParams\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHciCommandStatus\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gHciDataDiscardedAlloc\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHciDataDiscardedInvalidStateParam\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gHciEventReceiveHandler\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gHciRecvFragmentOfPacket\_c (C enumerator), 1355  
gapInternalErrorSource\_t.gL2capRxPacket\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeAddDeviceToPeriodicAdvList\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeChannelOverride\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gLeClearPeriodicAdvList\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeControllerTest\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLePeriodicAdvResponseReport\_c (C enumerator), 1357

gapInternalErrorSource\_t.gLePeriodicAdvSyncEstb\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLePeriodicAdvSyncLost\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeReadMaxAdvDataLen\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeReadNumOfSupportedAdvSets\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeReadPeriodicAdvListSize\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeReadPhy\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeRemoveDeviceFromPeriodicAdvList\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gLeScGetLocalOobData\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeScRegeneratePublicKey\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeScSendKeypressNotification\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeScSetPeerOobData\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeScValidateNumericValue\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeSetHostFeature\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gLeSetPeriodicAdvParamsV2\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gLeSetPeriodicAdvResponseData\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gLeSetPeriodicAdvSubeventData\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gLeSetPeriodicSyncSubevent\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gLeSetPhy\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeSetResolvablePrivateAddressTimeout\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gLeSetSchedulerPriority\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gModifySleepClockAccuracy\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gPeriodicAdvCancelSync (C enumerator), 1353  
gapInternalErrorSource\_t.gPeriodicAdvCreateSync (C enumerator), 1353  
gapInternalErrorSource\_t.gPeriodicAdvRcvEnable\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gPeriodicAdvSetInfoTransfer\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gPeriodicAdvSyncTransfer\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gPeriodicAdvTerminateSync (C enumerator), 1353  
gapInternalErrorSource\_t.gProvideLongTermKey\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gProvideOob\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gReadAntennaInformation\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gReadControllerLocalRPA\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gReadDeviceAddress\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gReadFilterAcceptListSize\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gReadLeBufferSize\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gReadLocalSupportedCommands\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gReadLocalSupportedFeatures\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gReadRadioPower\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gReadRemoteTransmitPowerLevel\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gReadRemoteVersionInfo\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gReadSuggestedDefaultDataLength\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gRemoveAdvertisingSet\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gRemoveDeviceFromFilterAcceptList\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gResetController\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gSaveKeys\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gSendPeripheralSecurityRequest\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gSendSmpKeys\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gSetChannelMap\_c (C enumerator), 1352  
gapInternalErrorSource\_t.gSetConnectionCteReceiveParams\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gSetConnectionCteTransmitParams\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gSetConnectionlessCteTransmitEnable\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gSetConnectionlessCteTransmitParams\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gSetConnectionlessIqSamplingEnable\_c (C enumerator), 1353  
gapInternalErrorSource\_t.gSetDataRelatedAddressChanges\_c (C enumerator), 1357  
gapInternalErrorSource\_t.gSetDecisionInstructions\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gSetDefaultPeriodicAdvSyncTransferParams\_c (C enumerator), 1354  
gapInternalErrorSource\_t.gSetEventMask\_c (C enumerator), 1351  
gapInternalErrorSource\_t.gSetExpMSupportedFeatures\_c (C enumerator), 1356  
gapInternalErrorSource\_t.gSetExtAdvDecisionData\_c (C enumerator), 1356

[gapInternalErrorSource\\_t.gSetLeEventMask\\_c \(C enumerator\), 1351](#)  
[gapInternalErrorSource\\_t.gSetPathLossReportingEnable\\_c \(C enumerator\), 1354](#)  
[gapInternalErrorSource\\_t.gSetPathLossReportingParams\\_c \(C enumerator\), 1354](#)  
[gapInternalErrorSource\\_t.gSetPeriodicAdvSyncTransferParams\\_c \(C enumerator\), 1354](#)  
[gapInternalErrorSource\\_t.gSetRandomAddress\\_c \(C enumerator\), 1351](#)  
[gapInternalErrorSource\\_t.gSetTransmitPowerReportingEnable\\_c \(C enumerator\), 1354](#)  
[gapInternalErrorSource\\_t.gTerminatePairing\\_c \(C enumerator\), 1352](#)  
[gapInternalErrorSource\\_t.gUpdateLeDataLength\\_c \(C enumerator\), 1352](#)  
[gapInternalErrorSource\\_t.gVendorUnitaryTest\\_c \(C enumerator\), 1357](#)  
[gapInternalErrorSource\\_t.gVerifySignature\\_c \(C enumerator\), 1351](#)  
[gapInternalErrorSource\\_t.gWriteSuggestedDefaultDataLength\\_c \(C enumerator\), 1352](#)  
[gapIoCapabilities\\_t \(C type\), 1172](#)  
[gapKeyExchangeRequestEvent\\_t \(C struct\), 1260, 1452](#)  
[gapKeyExchangeRequestEvent\\_t.requestedKeys \(C var\), 1260, 1452](#)  
[gapKeyExchangeRequestEvent\\_t.requestedLtkSize \(C var\), 1260, 1452](#)  
[gapKeypressNotification\\_t \(C type\), 1172](#)  
[gapKeypressNotification\\_tag \(C enum\), 1155](#)  
[gapKeypressNotification\\_tag \(C type\), 1172](#)  
[gapKeypressNotification\\_tag.gKnPasskeyCleared\\_c \(C enumerator\), 1156](#)  
[gapKeypressNotification\\_tag.gKnPasskeyDigitErased\\_c \(C enumerator\), 1156](#)  
[gapKeypressNotification\\_tag.gKnPasskeyDigitStarted\\_c \(C enumerator\), 1156](#)  
[gapKeypressNotification\\_tag.gKnPasskeyEntryCompleted\\_c \(C enumerator\), 1156](#)  
[gapKeypressNotification\\_tag.gKnPasskeyEntryStarted\\_c \(C enumerator\), 1156](#)  
[gapKeysReceivedEvent\\_t \(C struct\), 1260, 1452](#)  
[gapKeysReceivedEvent\\_t.pKeys \(C var\), 1261, 1453](#)  
[gapLeAllPhyFlags\\_t \(C enum\), 1357](#)  
[gapLeAllPhyFlags\\_t.gLeRxPhyNoPreference\\_c \(C enumerator\), 1358](#)  
[gapLeAllPhyFlags\\_t.gLeTxPhyNoPreference\\_c \(C enumerator\), 1358](#)  
[gapLEGenerateDhKeyCompleteEvent\\_t \(C struct\), 1396, 1428](#)  
[gapLEGenerateDhKeyCompleteEvent\\_t.aDHKey \(C var\), 1428](#)  
[gapLePhyFlags\\_t \(C type\), 1365](#)  
[gapLePhyMode\\_t \(C type\), 1365](#)  
[gapLePhyMode\\_tag \(C enum\), 1358](#)  
[gapLePhyMode\\_tag.gLePhy1M\\_c \(C enumerator\), 1358](#)  
[gapLePhyMode\\_tag.gLePhy2M\\_c \(C enumerator\), 1358](#)  
[gapLePhyMode\\_tag.gLePhyCoded\\_c \(C enumerator\), 1358](#)  
[gapLePhyOptionsFlags\\_t \(C enum\), 1358](#)  
[gapLePhyOptionsFlags\\_t.gLeCodingNoPreference\\_c \(C enumerator\), 1358](#)  
[gapLePhyOptionsFlags\\_t.gLeCodingS2\\_c \(C enumerator\), 1358](#)  
[gapLePhyOptionsFlags\\_t.gLeCodingS2Req\\_c \(C enumerator\), 1358](#)  
[gapLePhyOptionsFlags\\_t.gLeCodingS8\\_c \(C enumerator\), 1358](#)  
[gapLePhyOptionsFlags\\_t.gLeCodingS8Req\\_c \(C enumerator\), 1358](#)  
[gapLeScOobData\\_t \(C struct\), 1383, 1417](#)  
[gapLeScOobData\\_t.confirmValue \(C var\), 1383, 1417](#)  
[gapLeScOobData\\_t.randomValue \(C var\), 1383, 1417](#)  
[gapLlSkdReport\\_t \(C type\), 1366](#)  
[gapLlSkdReport\\_tag \(C struct\), 1396, 1428](#)  
[gapLlSkdReport\\_tag.aSKD \(C var\), 1397, 1429](#)  
[gapLlSkdReport\\_tag.deviceId \(C var\), 1397, 1429](#)  
[gapLongTermKeyRequestEvent\\_t \(C struct\), 1261, 1453](#)  
[gapLongTermKeyRequestEvent\\_t.aRand \(C var\), 1261, 1453](#)  
[gapLongTermKeyRequestEvent\\_t.ediv \(C var\), 1261, 1453](#)  
[gapLongTermKeyRequestEvent\\_t.randSize \(C var\), 1261, 1453](#)  
[gApp2Host\\_TaskQueue \(C var\), 1367](#)  
[gapPairingCompleteEvent\\_t \(C struct\), 1261, 1453](#)  
[gapPairingCompleteEvent\\_t.pairingCompleteData \(C union\), 1269, 1460](#)  
[gapPairingCompleteEvent\\_t.pairingCompleteData \(C var\), 1261, 1453](#)  
[gapPairingCompleteEvent\\_t.pairingCompleteData.failReason \(C var\), 1269, 1460](#)

gapPairingCompleteEvent\_t.pairingCompleteData.withBonding (C var), 1269, 1460  
gapPairingCompleteEvent\_t.pairingSuccessful (C var), 1261, 1453  
gapPairingParameters\_t (C struct), 1238, 1436  
gapPairingParameters\_t.centralKeys (C var), 1238, 1436  
gapPairingParameters\_t.leSecureConnectionSupported (C var), 1238, 1436  
gapPairingParameters\_t.localIoCapabilities (C var), 1238, 1436  
gapPairingParameters\_t.maxEncryptionKeySize (C var), 1238, 1436  
gapPairingParameters\_t.oobAvailable (C var), 1238, 1436  
gapPairingParameters\_t.peripheralKeys (C var), 1238, 1436  
gapPairingParameters\_t.securityModeAndLevel (C var), 1238, 1436  
gapPairingParameters\_t.useKeypressNotifications (C var), 1239, 1437  
gapPairingParameters\_t.withBonding (C var), 1238, 1436  
gapPathLossReportingParams\_t (C struct), 1250, 1444  
gapPathLossReportingParams\_t.highHysteresis (C var), 1250, 1444  
gapPathLossReportingParams\_t.highThreshold (C var), 1250, 1444  
gapPathLossReportingParams\_t.lowHysteresis (C var), 1250, 1444  
gapPathLossReportingParams\_t.lowThreshold (C var), 1250, 1444  
gapPathLossReportingParams\_t.minTimeSpent (C var), 1250, 1444  
gapPathLossThresholdEvent\_t (C struct), 1264, 1455  
gapPathLossThresholdEvent\_t.currentPathLoss (C var), 1264, 1455  
gapPathLossThresholdEvent\_t.zoneEntered (C var), 1264, 1455  
gapPerAdvResponse\_t (C struct), 1254, 1447  
gapPerAdvResponse\_t.aData (C var), 1448  
gapPerAdvResponse\_t.advHandle (C var), 1447  
gapPerAdvResponse\_t.cteType (C var), 1447  
gapPerAdvResponse\_t.dataLength (C var), 1448  
gapPerAdvResponse\_t.responseSlot (C var), 1448  
gapPerAdvResponse\_t.rssi (C var), 1447  
gapPerAdvResponse\_t.subevent (C var), 1447  
gapPerAdvResponse\_t.txPower (C var), 1447  
gapPerAdvSubeventDataRequest\_t (C struct), 1253, 1447  
gapPerAdvSubeventDataRequest\_t.handle (C var), 1254, 1447  
gapPerAdvSubeventDataRequest\_t.subeventDataCount (C var), 1254, 1447  
gapPerAdvSubeventDataRequest\_t.subeventStart (C var), 1254, 1447  
gapPeriodicAdvertisingResponseData\_t (C struct), 1252, 1446  
gapPeriodicAdvertisingResponseData\_t.pResponseData (C var), 1252, 1446  
gapPeriodicAdvertisingResponseData\_t.requestEvent (C var), 1252, 1446  
gapPeriodicAdvertisingResponseData\_t.requestSubevent (C var), 1252, 1446  
gapPeriodicAdvertisingResponseData\_t.responseSlot (C var), 1252, 1446  
gapPeriodicAdvertisingResponseData\_t.responseSubevent (C var), 1252, 1446  
gapPeriodicAdvertisingSubeventData\_t (C struct), 1252, 1446  
gapPeriodicAdvertisingSubeventData\_t.aSubeventDataStructures (C var), 1252, 1446  
gapPeriodicAdvertisingSubeventData\_t.cNumSubevents (C var), 1252, 1446  
gapPeriodicAdvListOperation\_t (C enum), 1169  
gapPeriodicAdvListOperation\_t.gAddDevice\_c (C enumerator), 1169  
gapPeriodicAdvListOperation\_t.gRemoveAllDevices\_c (C enumerator), 1170  
gapPeriodicAdvListOperation\_t.gRemoveDevice\_c (C enumerator), 1169  
gapPeriodicAdvParameters\_t (C type), 1172  
gapPeriodicAdvParameters\_tag (C struct), 1241, 1438  
gapPeriodicAdvParameters\_tag.addTxPowerInAdv (C var), 1241, 1438  
gapPeriodicAdvParameters\_tag.handle (C var), 1241, 1438  
gapPeriodicAdvParameters\_tag.maxInterval (C var), 1241, 1438  
gapPeriodicAdvParameters\_tag.minInterval (C var), 1241, 1438  
gapPeriodicAdvParameters\_tag.numResponseSlots (C var), 1242, 1439  
gapPeriodicAdvParameters\_tag.numSubevents (C var), 1242, 1438  
gapPeriodicAdvParameters\_tag.responseSlotDelay (C var), 1242, 1439  
gapPeriodicAdvParameters\_tag.responseSlotSpacing (C var), 1242, 1439  
gapPeriodicAdvParameters\_tag.subeventInterval (C var), 1242, 1438

[gapPeriodicAdvSetInfoTransfer\\_t \(C type\), 1173](#)  
[gapPeriodicAdvSetInfoTransfer\\_tag \(C struct\), 1242, 1439](#)  
[gapPeriodicAdvSetInfoTransfer\\_tag.advHandle \(C var\), 1242, 1439](#)  
[gapPeriodicAdvSetInfoTransfer\\_tag.deviceId \(C var\), 1242, 1439](#)  
[gapPeriodicAdvSetInfoTransfer\\_tag.serviceData \(C var\), 1242, 1439](#)  
[gapPeriodicAdvSyncMode\\_t \(C type\), 1172](#)  
[gapPeriodicAdvSyncMode\\_tag \(C enum\), 1157](#)  
[gapPeriodicAdvSyncMode\\_tag.gapPeriodicSyncNoReports\\_c \(C enumerator\), 1157](#)  
[gapPeriodicAdvSyncMode\\_tag.gapPeriodicSyncNoSyncMode\\_c \(C enumerator\), 1157](#)  
[gapPeriodicAdvSyncMode\\_tag.gapPeriodicSyncReportsEnabled\\_c \(C enumerator\), 1157](#)  
[gapPeriodicAdvSyncMode\\_tag.gapPeriodicSyncReportsEnabledWithDF\\_c \(C enumerator\), 1157](#)  
[gapPeriodicAdvSyncReq\\_t \(C type\), 1173](#)  
[gapPeriodicAdvSyncReq\\_tag \(C struct\), 1244, 1440](#)  
[gapPeriodicAdvSyncReq\\_tag.cteType \(C var\), 1244, 1440](#)  
[gapPeriodicAdvSyncReq\\_tag.options \(C var\), 1244, 1440](#)  
[gapPeriodicAdvSyncReq\\_tag.peerAddress \(C var\), 1244, 1440](#)  
[gapPeriodicAdvSyncReq\\_tag.peerAddressType \(C var\), 1244, 1440](#)  
[gapPeriodicAdvSyncReq\\_tag.SID \(C var\), 1244, 1440](#)  
[gapPeriodicAdvSyncReq\\_tag.skipCount \(C var\), 1244, 1440](#)  
[gapPeriodicAdvSyncReq\\_tag.timeout \(C var\), 1244, 1440](#)  
[gapPeriodicAdvSyncTransfer\\_t \(C type\), 1172](#)  
[gapPeriodicAdvSyncTransfer\\_tag \(C struct\), 1242, 1439](#)  
[gapPeriodicAdvSyncTransfer\\_tag.deviceId \(C var\), 1242, 1439](#)  
[gapPeriodicAdvSyncTransfer\\_tag.serviceData \(C var\), 1242, 1439](#)  
[gapPeriodicAdvSyncTransfer\\_tag.syncHandle \(C var\), 1242, 1439](#)  
[gapPeriodicScannedDevice\\_t \(C struct\), 1256, 1449](#)  
[gapPeriodicScannedDevice\\_t.cteType \(C var\), 1256, 1449](#)  
[gapPeriodicScannedDevice\\_t.dataLength \(C var\), 1256, 1449](#)  
[gapPeriodicScannedDevice\\_t.pData \(C var\), 1256, 1450](#)  
[gapPeriodicScannedDevice\\_t.rssi \(C var\), 1256, 1449](#)  
[gapPeriodicScannedDevice\\_t.syncHandle \(C var\), 1256, 1449](#)  
[gapPeriodicScannedDevice\\_t.txPower \(C var\), 1256, 1449](#)  
[gapPeriodicScannedDeviceV2\\_t \(C struct\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.cteType \(C var\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.dataLength \(C var\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.pData \(C var\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.periodicEventCounter \(C var\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.rssi \(C var\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.subevent \(C var\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.syncHandle \(C var\), 1257, 1450](#)  
[gapPeriodicScannedDeviceV2\\_t.txPower \(C var\), 1257, 1450](#)  
[gapPeriodicSyncSubeventParameters\\_t \(C struct\), 1252, 1446](#)  
[gapPeriodicSyncSubeventParameters\\_t.aSubevents \(C var\), 1253, 1446](#)  
[gapPeriodicSyncSubeventParameters\\_t.numSubevents \(C var\), 1253, 1446](#)  
[gapPeriodicSyncSubeventParameters\\_t.perAdvProperties \(C var\), 1253, 1446](#)  
[gapPeripheralSecurityRequestParameters\\_t \(C struct\), 1239, 1437](#)  
[gapPeripheralSecurityRequestParameters\\_t.authenticationRequired \(C var\), 1239, 1437](#)  
[gapPeripheralSecurityRequestParameters\\_t.bondAfterPairing \(C var\), 1239, 1437](#)  
[gapPhyEvent\\_t \(C struct\), 1384, 1418](#)  
[gapPhyEvent\\_t.deviceId \(C var\), 1418](#)  
[gapPhyEvent\\_t.phyEventType \(C var\), 1418](#)  
[gapPhyEvent\\_t.rxPhy \(C var\), 1418](#)  
[gapPhyEvent\\_t.txPhy \(C var\), 1418](#)  
[gapPhyEventType\\_t \(C enum\), 1358](#)  
[gapPhyEventType\\_t.gPhyRead\\_c \(C enumerator\), 1358](#)  
[gapPhyEventType\\_t.gPhySetDefaultComplete\\_c \(C enumerator\), 1358](#)  
[gapPhyEventType\\_t.gPhyUpdateComplete\\_c \(C enumerator\), 1359](#)  
[gapPrivateKeyType\\_t \(C enum\), 1158](#)

gapPrivateKeyType\_t.gUseDebugKey\_c (C enumerator), 1158  
gapPrivateKeyType\_t.gUseGeneratedKey\_c (C enumerator), 1158  
gapRadioPowerLevelReadType\_t (C enum), 1162  
gapRadioPowerLevelReadType\_t.gRssi\_c (C enumerator), 1163  
gapRadioPowerLevelReadType\_t.gTxPowerCurrentLevelInConnection\_c (C enumerator), 1162  
gapRadioPowerLevelReadType\_t.gTxPowerLevelForAdvertising\_c (C enumerator), 1163  
gapRadioPowerLevelReadType\_t.gTxPowerMaximumLevelInConnection\_c (C enumerator), 1163  
gapRemoteVersionInfoRead\_t (C type), 1366  
gapRemoteVersionInfoRead\_tag (C struct), 1396, 1428  
gapRemoteVersionInfoRead\_tag.deviceId (C var), 1428  
gapRemoteVersionInfoRead\_tag.manufacturerName (C var), 1428  
gapRemoteVersionInfoRead\_tag.status (C var), 1428  
gapRemoteVersionInfoRead\_tag.subversion (C var), 1428  
gapRemoteVersionInfoRead\_tag.version (C var), 1428  
gapRole\_t (C enum), 1155  
gapRole\_t.gGapBroadcaster\_c (C enumerator), 1155  
gapRole\_t.gGapCentral\_c (C enumerator), 1155  
gapRole\_t.gGapObserver\_c (C enumerator), 1155  
gapRole\_t.gGapPeripheral\_c (C enumerator), 1155  
gapScanMode\_t (C enum), 1156  
gapScanMode\_t.gAutoConnect\_c (C enumerator), 1156  
gapScanMode\_t.gDefaultScan\_c (C enumerator), 1156  
gapScanMode\_t.gGeneralDiscovery\_c (C enumerator), 1156  
gapScanMode\_t.gLimitedDiscovery\_c (C enumerator), 1156  
gapScannedDevice\_t (C struct), 1254, 1448  
gapScannedDevice\_t.aAddress (C var), 1254, 1448  
gapScannedDevice\_t.addressType (C var), 1254, 1448  
gapScannedDevice\_t.advertisingAddressResolved (C var), 1255, 1448  
gapScannedDevice\_t.advEventType (C var), 1254, 1448  
gapScannedDevice\_t.data (C var), 1254, 1448  
gapScannedDevice\_t.dataLength (C var), 1254, 1448  
gapScannedDevice\_t.directRpa (C var), 1255, 1448  
gapScannedDevice\_t.directRpaUsed (C var), 1255, 1448  
gapScannedDevice\_t.rssi (C var), 1254, 1448  
gapScanningCallback\_t (C type), 1173  
gapScanningEvent\_t (C struct), 1259, 1451  
gapScanningEvent\_t.eventData (C union), 1268, 1460  
gapScanningEvent\_t.eventData (C var), 1259, 1452  
gapScanningEvent\_t.eventData.extScannedDevice (C var), 1268, 1460  
gapScanningEvent\_t.eventData.failReason (C var), 1268, 1460  
gapScanningEvent\_t.eventData.iqReport (C var), 1268, 1460  
gapScanningEvent\_t.eventData.periodicScannedDevice (C var), 1268, 1460  
gapScanningEvent\_t.eventData.periodicScannedDeviceV2 (C var), 1268, 1460  
gapScanningEvent\_t.eventData.scannedDevice (C var), 1268, 1460  
gapScanningEvent\_t.eventData.syncEstb (C var), 1268, 1460  
gapScanningEvent\_t.eventData.syncLost (C var), 1268, 1460  
gapScanningEvent\_t.eventType (C var), 1259, 1452  
gapScanningEventType\_t (C enum), 1164  
gapScanningEventType\_t.gConnectionlessIqReportReceived\_c (C enumerator), 1165  
gapScanningEventType\_t.gDeviceScanned\_c (C enumerator), 1165  
gapScanningEventType\_t.gExtDeviceScanned\_c (C enumerator), 1165  
gapScanningEventType\_t.gPeriodicAdvSyncEstablished\_c (C enumerator), 1165  
gapScanningEventType\_t.gPeriodicAdvSyncLost\_c (C enumerator), 1165  
gapScanningEventType\_t.gPeriodicAdvSyncTerminated\_c (C enumerator), 1165  
gapScanningEventType\_t.gPeriodicDeviceScanned\_c (C enumerator), 1165  
gapScanningEventType\_t.gPeriodicDeviceScannedV2\_c (C enumerator), 1165  
gapScanningEventType\_t.gScanCommandFailed\_c (C enumerator), 1164  
gapScanningEventType\_t.gScanStateChanged\_c (C enumerator), 1164

[gapScanningParameters\\_t \(C struct\), 1243, 1439](#)  
[gapScanningParameters\\_t.filterPolicy \(C var\), 1243, 1440](#)  
[gapScanningParameters\\_t.interval \(C var\), 1243, 1440](#)  
[gapScanningParameters\\_t.ownAddressType \(C var\), 1243, 1440](#)  
[gapScanningParameters\\_t.scanningPHYs \(C var\), 1244, 1440](#)  
[gapScanningParameters\\_t.type \(C var\), 1243, 1440](#)  
[gapScanningParameters\\_t.window \(C var\), 1243, 1440](#)  
[gapScanResponseData\\_t \(C type\), 1173](#)  
[gapSecurityLevel\\_t \(C type\), 1172](#)  
[gapSecurityMode\\_t \(C type\), 1172](#)  
[gapSecurityModeAndLevel\\_t \(C type\), 1172](#)  
[gapSecurityRequirements\\_t \(C struct\), 1236, 1435](#)  
[gapSecurityRequirements\\_t.authorization \(C var\), 1237, 1435](#)  
[gapSecurityRequirements\\_t.minimumEncryptionKeySize \(C var\), 1237, 1435](#)  
[gapSecurityRequirements\\_t.securityModeLevel \(C var\), 1236, 1435](#)  
[gapServiceSecurityRequirements\\_t \(C struct\), 1237, 1435](#)  
[gapServiceSecurityRequirements\\_t.requirements \(C var\), 1237, 1435](#)  
[gapServiceSecurityRequirements\\_t.serviceHandle \(C var\), 1237, 1435](#)  
[gapSetPeriodicAdvSyncTransferParams\\_t \(C type\), 1173](#)  
[gapSetPeriodicAdvSyncTransferParams\\_tag \(C struct\), 1242, 1439](#)  
[gapSetPeriodicAdvSyncTransferParams\\_tag.CTETType \(C var\), 1243, 1439](#)  
[gapSetPeriodicAdvSyncTransferParams\\_tag.deviceId \(C var\), 1243, 1439](#)  
[gapSetPeriodicAdvSyncTransferParams\\_tag.mode \(C var\), 1243, 1439](#)  
[gapSetPeriodicAdvSyncTransferParams\\_tag.skip \(C var\), 1243, 1439](#)  
[gapSetPeriodicAdvSyncTransferParams\\_tag.syncTimeout \(C var\), 1243, 1439](#)  
[gapSleepClockAccuracy\\_t \(C enum\), 1164](#)  
[gapSleepClockAccuracy\\_t.gSwitchToLessAccurateClock\\_c \(C enumerator\), 1164](#)  
[gapSleepClockAccuracy\\_t.gSwitchToMoreAccurateClock\\_c \(C enumerator\), 1164](#)  
[gapSmpKeyFlags\\_t \(C type\), 1172](#)  
[gapSmpKeys\\_t \(C struct\), 1236, 1435](#)  
[gapSmpKeys\\_t.aAddress \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.aCsrk \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.addressType \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.aIrk \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.aLtk \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.aRand \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.cLtkSize \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.cRandSize \(C var\), 1236, 1435](#)  
[gapSmpKeys\\_t.ediv \(C var\), 1236, 1435](#)  
[gapSubeventDataStructure\\_t \(C struct\), 1251, 1445](#)  
[gapSubeventDataStructure\\_t.pAdvertisingData \(C var\), 1252, 1446](#)  
[gapSubeventDataStructure\\_t.responseSlotCount \(C var\), 1252, 1446](#)  
[gapSubeventDataStructure\\_t.responseSlotStart \(C var\), 1252, 1445](#)  
[gapSubeventDataStructure\\_t.subevent \(C var\), 1251, 1445](#)  
[gapSyncEstbEventData\\_t \(C struct\), 1257, 1450](#)  
[gapSyncEstbEventData\\_t.advertiserClockAccuracy \(C var\), 1258, 1450](#)  
[gapSyncEstbEventData\\_t.numSubevents \(C var\), 1258, 1450](#)  
[gapSyncEstbEventData\\_t.peerAddress \(C var\), 1258, 1450](#)  
[gapSyncEstbEventData\\_t.peerAddressType \(C var\), 1257, 1450](#)  
[gapSyncEstbEventData\\_t.periodicAdvInterval \(C var\), 1258, 1450](#)  
[gapSyncEstbEventData\\_t.PHY \(C var\), 1258, 1450](#)  
[gapSyncEstbEventData\\_t.responseSlotDelay \(C var\), 1258, 1451](#)  
[gapSyncEstbEventData\\_t.responseSlotSpacing \(C var\), 1258, 1451](#)  
[gapSyncEstbEventData\\_t.SID \(C var\), 1257, 1450](#)  
[gapSyncEstbEventData\\_t.status \(C var\), 1257, 1450](#)  
[gapSyncEstbEventData\\_t.subeventInterval \(C var\), 1258, 1451](#)  
[gapSyncEstbEventData\\_t.syncHandle \(C var\), 1257, 1450](#)  
[gapSyncLostEventData\\_t \(C struct\), 1258, 1451](#)

gapSyncLostEventData\_t.syncHandle (C var), 1258, 1451  
gapSyncTransferReceivedEventData\_t (C type), 1365  
gapSyncTransferReceivedEventData\_tag (C struct), 1386, 1420  
gapSyncTransferReceivedEventData\_tag.advAddress (C var), 1386, 1421  
gapSyncTransferReceivedEventData\_tag.advAddressType (C var), 1386, 1421  
gapSyncTransferReceivedEventData\_tag.advClockAccuracy (C var), 1387, 1421  
gapSyncTransferReceivedEventData\_tag.advPhy (C var), 1386, 1421  
gapSyncTransferReceivedEventData\_tag.advSID (C var), 1386, 1421  
gapSyncTransferReceivedEventData\_tag.deviceId (C var), 1386, 1420  
gapSyncTransferReceivedEventData\_tag.numSubevents (C var), 1387, 1421  
gapSyncTransferReceivedEventData\_tag.periodicAdvInt (C var), 1386, 1421  
gapSyncTransferReceivedEventData\_tag.responseSlotDelay (C var), 1387, 1421  
gapSyncTransferReceivedEventData\_tag.responseSlotSpacing (C var), 1387, 1421  
gapSyncTransferReceivedEventData\_tag.serviceData (C var), 1386, 1420  
gapSyncTransferReceivedEventData\_tag.status (C var), 1420  
gapSyncTransferReceivedEventData\_tag.subeventInterval (C var), 1387, 1421  
gapSyncTransferReceivedEventData\_tag.syncHandle (C var), 1386, 1421  
gapTransmitPowerInfo\_t (C struct), 1265, 1456  
gapTransmitPowerInfo\_t.currTxPowerLevel (C var), 1265, 1456  
gapTransmitPowerInfo\_t.maxTxPowerLevel (C var), 1265, 1456  
gapTransmitPowerInfo\_t.phy (C var), 1265, 1456  
gapTransmitPowerReporting\_t (C struct), 1264, 1455  
gapTransmitPowerReporting\_t.delta (C var), 1265, 1455  
gapTransmitPowerReporting\_t.flags (C var), 1265, 1455  
gapTransmitPowerReporting\_t.phy (C var), 1265, 1455  
gapTransmitPowerReporting\_t.reason (C var), 1264, 1455  
gapTransmitPowerReporting\_t.txPowerLevel (C var), 1265, 1455  
Gatt\_GetMtu (C function), 1275  
Gatt\_Init (C function), 1274  
gattAttribute\_t (C struct), 1275, 1462  
gattAttribute\_t.handle (C var), 1275, 1462  
gattAttribute\_t.maxValueLength (C var), 1276, 1462  
gattAttribute\_t.paValue (C var), 1276, 1462  
gattAttribute\_t.uuid (C var), 1276, 1462  
gattAttribute\_t.uuidType (C var), 1276, 1462  
gattAttribute\_t.valueLength (C var), 1276, 1462  
gattAttributePermissionsBitFields\_t (C type), 1316  
gattCachingClientState\_c (C enum), 1273  
gattCachingClientState\_c.gGattClientChangeAware\_c (C enumerator), 1273  
gattCachingClientState\_c.gGattClientChangeUnaware\_c (C enumerator), 1273  
gattCachingClientState\_c.gGattClientStateChangePending\_c (C enumerator), 1273  
gattCccdFlags\_t (C type), 1274  
gattCharacteristic\_t (C struct), 1276, 1462  
gattCharacteristic\_t.aDescriptors (C var), 1276, 1463  
gattCharacteristic\_t.cNumDescriptors (C var), 1276, 1463  
gattCharacteristic\_t.properties (C var), 1276, 1463  
gattCharacteristic\_t.value (C var), 1276, 1463  
gattCharacteristicPropertiesBitFields\_t (C type), 1316  
gattCharacteristicPropertiesBitFields\_tag (C enum), 1315  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropAuthSignedWrites\_c (C enumerator), 1316  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropBroadcast\_c (C enumerator), 1315  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropExtendedProperties\_c (C enumerator), 1316  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropIndicate\_c (C enumerator), 1316  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropNone\_c (C enumerator), 1315  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropNotify\_c (C enumerator), 1315  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropRead\_c (C enumerator), 1315  
gattCharacteristicPropertiesBitFields\_tag.gGattCharPropWrite\_c (C enumerator), 1315

[gattCharacteristicPropertiesBitFields\\_tag.gGattCharPropWriteWithoutRsp\\_c \(C enumerator\), 1315](#)  
[GattClient\\_CharacteristicSignedWrite \(C macro\), 1301](#)  
[GattClient\\_CharacteristicWriteWithoutResponse \(C macro\), 1301](#)  
[GattClient\\_DiscoverAllCharacteristicDescriptors \(C function\), 1286](#)  
[GattClient\\_DiscoverAllCharacteristicsOfService \(C function\), 1285](#)  
[GattClient\\_DiscoverAllPrimaryServices \(C function\), 1283](#)  
[GattClient\\_DiscoverCharacteristicOfServiceByUuid \(C function\), 1285](#)  
[GattClient\\_DiscoverPrimaryServicesByUuid \(C function\), 1284](#)  
[GattClient\\_EnhancedDiscoverAllCharacteristicDescriptors \(C function\), 1295](#)  
[GattClient\\_EnhancedDiscoverAllCharacteristicsOfService \(C function\), 1294](#)  
[GattClient\\_EnhancedDiscoverAllPrimaryServices \(C function\), 1292](#)  
[GattClient\\_EnhancedDiscoverCharacteristicOfServiceByUuid \(C function\), 1295](#)  
[GattClient\\_EnhancedDiscoverPrimaryServicesByUuid \(C function\), 1293](#)  
[GattClient\\_EnhancedFindIncludedServices \(C function\), 1293](#)  
[GattClient\\_EnhancedReadCharacteristicDescriptor \(C function\), 1299](#)  
[GattClient\\_EnhancedReadCharacteristicValue \(C function\), 1296](#)  
[GattClient\\_EnhancedReadMultipleCharacteristicValues \(C function\), 1298](#)  
[GattClient\\_EnhancedReadMultipleVariableCharacteristicValues \(C function\), 1300](#)  
[GattClient\\_EnhancedReadUsingCharacteristicUuid \(C function\), 1297](#)  
[GattClient\\_EnhancedWriteCharacteristicDescriptor \(C function\), 1299](#)  
[GattClient\\_EnhancedWriteCharacteristicValue \(C function\), 1298](#)  
[GattClient\\_ExchangeMtu \(C function\), 1283](#)  
[GattClient\\_FindIncludedServices \(C function\), 1284](#)  
[GattClient\\_Init \(C function\), 1281](#)  
[GattClient\\_ReadCharacteristicDescriptor \(C function\), 1289](#)  
[GattClient\\_ReadCharacteristicValue \(C function\), 1287](#)  
[GattClient\\_ReadMultipleCharacteristicValues \(C function\), 1288](#)  
[GattClient\\_ReadMultipleVariableCharacteristicValues \(C function\), 1290](#)  
[GattClient\\_ReadUsingCharacteristicUuid \(C function\), 1287](#)  
[GattClient\\_RegisterEnhancedIndicationCallback \(C function\), 1292](#)  
[GattClient\\_RegisterEnhancedMultipleValueNotificationCallback \(C function\), 1292](#)  
[GattClient\\_RegisterEnhancedNotificationCallback \(C function\), 1291](#)  
[GattClient\\_RegisterEnhancedProcedureCallback \(C function\), 1291](#)  
[GattClient\\_RegisterIndicationCallback \(C function\), 1282](#)  
[GattClient\\_RegisterMultipleValueNotificationCallback \(C function\), 1282](#)  
[GattClient\\_RegisterNotificationCallback \(C function\), 1282](#)  
[GattClient\\_RegisterProcedureCallback \(C function\), 1281](#)  
[GattClient\\_ResetProcedure \(C function\), 1281](#)  
[GattClient\\_SimpleCharacteristicWrite \(C macro\), 1301](#)  
[GattClient\\_WriteCharacteristicDescriptor \(C function\), 1290](#)  
[GattClient\\_WriteCharacteristicValue \(C function\), 1289](#)  
[gattClientEnhancedIndicationCallback\\_t \(C type\), 1281](#)  
[gattClientEnhancedMultipleValueNotificationCallback\\_t \(C type\), 1281](#)  
[gattClientEnhancedNotificationCallback\\_t \(C type\), 1281](#)  
[gattClientEnhancedProcedureCallback\\_t \(C type\), 1280](#)  
[gattClientHashUpdateType\\_t \(C enum\), 1273](#)  
[gattClientHashUpdateType\\_t.gattClientActiveConnectionUpdate \(C enumerator\), 1273](#)  
[gattClientHashUpdateType\\_t.gattClientFirstConnection\\_c \(C enumerator\), 1273](#)  
[gattClientHashUpdateType\\_t.gattClientNoChange \(C enumerator\), 1273](#)  
[gattClientHashUpdateType\\_t.gattClientReconnectBondedPeer\\_c \(C enumerator\), 1273](#)  
[gattClientIndicationCallback\\_t \(C type\), 1280](#)  
[gattClientMultipleValueNotificationCallback\\_t \(C type\), 1280](#)  
[gattClientNotificationCallback\\_t \(C type\), 1280](#)  
[gattClientProcedureCallback\\_t \(C type\), 1280](#)  
[gattDatabase \(C var\), 1316](#)  
[GattDb\\_ComputeDatabaseHash \(C function\), 1317](#)  
[GattDb\\_Deinit \(C function\), 1318](#)  
[GattDb\\_FindCccdHandleForCharValueHandle \(C function\), 1321](#)

[GattDb\\_FindCharValueHandleInService \(C function\), 1320](#)  
[GattDb\\_FindDescriptorHandleForCharValueHandle \(C function\), 1321](#)  
[GattDb\\_FindServiceHandle \(C function\), 1319](#)  
[GattDb\\_FindServiceRange \(C function\), 1317](#)  
[GattDb\\_GetAttributeValueSize \(C function\), 1317](#)  
[GattDb\\_GetIndexOfHandle \(C function\), 1316](#)  
[GattDb\\_Init \(C function\), 1317](#)  
[GattDb\\_ReadAttribute \(C function\), 1319](#)  
[GattDb\\_ServiceStartHandle \(C function\), 1317](#)  
[GattDb\\_WriteAttribute \(C function\), 1318](#)  
[gattDbAccessType\\_t \(C enum\), 1316](#)  
[gattDbAccessType\\_t.gAccessNotify\\_c \(C enumerator\), 1316](#)  
[gattDbAccessType\\_t.gAccessRead\\_c \(C enumerator\), 1316](#)  
[gattDbAccessType\\_t.gAccessWrite\\_c \(C enumerator\), 1316](#)  
[gattDbAttribute\\_t \(C struct\), 1322, 1468](#)  
[gattDbAttribute\\_t.handle \(C var\), 1322, 1468](#)  
[gattDbAttribute\\_t.maxVariableValueLength \(C var\), 1323, 1468](#)  
[gattDbAttribute\\_t.permissions \(C var\), 1323, 1468](#)  
[gattDbAttribute\\_t.pValue \(C var\), 1323, 1468](#)  
[gattDbAttribute\\_t.uuid \(C var\), 1323, 1468](#)  
[gattDbAttribute\\_t.uuidType \(C var\), 1323, 1468](#)  
[gattDbAttribute\\_t.valueLength \(C var\), 1323, 1468](#)  
[gattDbCharPresFormat\\_t \(C struct\), 1277, 1463](#)  
[gattDbCharPresFormat\\_t.description \(C var\), 1277, 1463](#)  
[gattDbCharPresFormat\\_t.exponent \(C var\), 1277, 1463](#)  
[gattDbCharPresFormat\\_t.format \(C var\), 1277, 1463](#)  
[gattDbCharPresFormat\\_t.ns \(C var\), 1277, 1463](#)  
[gattDbCharPresFormat\\_t.unitUuid16 \(C var\), 1277, 1463](#)  
[gAttDefaultMtu\\_c \(C macro\), 1404](#)  
[gattHandleRange\\_t \(C struct\), 1277, 1463](#)  
[gattHandleRange\\_t.endHandle \(C var\), 1277, 1464](#)  
[gattHandleRange\\_t.startHandle \(C var\), 1277, 1464](#)  
[gAttMaxMtu\\_c \(C macro\), 1404](#)  
[gAttMaxValueLength\\_c \(C macro\), 1404](#)  
[gattProcedurePhase\\_t \(C enum\), 1274](#)  
[gattProcedurePhase\\_t.gattProcPhaseInitiated \(C enumerator\), 1274](#)  
[gattProcedurePhase\\_t.gattProcPhaseRunning \(C enumerator\), 1274](#)  
[gattProcedureResult\\_t \(C enum\), 1280](#)  
[gattProcedureResult\\_t.gGattProcError\\_c \(C enumerator\), 1280](#)  
[gattProcedureResult\\_t.gGattProcSuccess\\_c \(C enumerator\), 1280](#)  
[gattProcedureType\\_t \(C enum\), 1273](#)  
[gattProcedureType\\_t.gGattProcDiscoverAllCharacteristicDescriptors\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcDiscoverAllCharacteristics\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcDiscoverAllPrimaryServices\\_c \(C enumerator\), 1273](#)  
[gattProcedureType\\_t.gGattProcDiscoverCharacteristicByUuid\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcDiscoverPrimaryServicesByUuid\\_c \(C enumerator\), 1273](#)  
[gattProcedureType\\_t.gGattProcExchangeMtu\\_c \(C enumerator\), 1273](#)  
[gattProcedureType\\_t.gGattProcFindIncludedServices\\_c \(C enumerator\), 1273](#)  
[gattProcedureType\\_t.gGattProcReadCharacteristicDescriptor\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcReadCharacteristicValue\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcReadMultipleCharacteristicValues\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcReadMultipleVarLengthCharValues\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcReadUsingCharacteristicUuid\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcWriteCharacteristicDescriptor\\_c \(C enumerator\), 1274](#)  
[gattProcedureType\\_t.gGattProcWriteCharacteristicValue\\_c \(C enumerator\), 1274](#)  
[GattServer\\_EnhancedSendAttributeReadStatus \(C function\), 1310](#)  
[GattServer\\_EnhancedSendAttributeWrittenStatus \(C function\), 1309](#)  
[GattServer\\_EnhancedSendIndication \(C function\), 1311](#)

GattServer\_EnhancedSendInstantValueIndication (C function), 1312  
 GattServer\_EnhancedSendInstantValueNotification (C function), 1311  
 GattServer\_EnhancedSendMultipleHandleValueNotification (C function), 1312  
 GattServer\_EnhancedSendNotification (C function), 1311  
 GattServer\_Init (C function), 1303  
 GattServer\_RegisterCallback (C function), 1303  
 GattServer\_RegisterEnhancedCallback (C function), 1309  
 GattServer\_RegisterHandlesForReadNotifications (C function), 1305  
 GattServer\_RegisterHandlesForWriteNotifications (C function), 1304  
 GattServer\_RegisterUniqueHandlesForNotifications (C function), 1309  
 GattServer\_SendAttributeReadStatus (C function), 1306  
 GattServer\_SendAttributeWrittenStatus (C function), 1305  
 GattServer\_SendIndication (C function), 1307  
 GattServer\_SendInstantValueIndication (C function), 1308  
 GattServer\_SendInstantValueNotification (C function), 1308  
 GattServer\_SendMultipleHandleValueNotification (C function), 1308  
 GattServer\_SendNotification (C function), 1307  
 GattServer\_UnregisterHandlesForReadNotifications (C function), 1306  
 GattServer\_UnregisterHandlesForWriteNotifications (C function), 1304  
 gattServerAttributeReadEvent\_t (C struct), 1314, 1466  
 gattServerAttributeReadEvent\_t.handle (C var), 1314, 1467  
 gattServerAttributeWrittenEvent\_t (C struct), 1313, 1466  
 gattServerAttributeWrittenEvent\_t.aValue (C var), 1313, 1466  
 gattServerAttributeWrittenEvent\_t.bearerId (C var), 1313, 1466  
 gattServerAttributeWrittenEvent\_t.cValueLength (C var), 1313, 1466  
 gattServerAttributeWrittenEvent\_t.handle (C var), 1313, 1466  
 gattServerCallback\_t (C type), 1303  
 gattServerCccdWrittenEvent\_t (C struct), 1313, 1466  
 gattServerCccdWrittenEvent\_t.handle (C var), 1314, 1466  
 gattServerCccdWrittenEvent\_t.newCccd (C var), 1314, 1466  
 gattServerEnhancedCallback\_t (C type), 1303  
 gattServerEvent\_t (C struct), 1314, 1467  
 gattServerEvent\_t.eventData (C union), 1315, 1467  
 gattServerEvent\_t.eventData (C var), 1314, 1467  
 gattServerEvent\_t.eventData.attributeOpCode (C var), 1315, 1467  
 gattServerEvent\_t.eventData.attributeReadEvent (C var), 1315, 1467  
 gattServerEvent\_t.eventData.attributeWrittenEvent (C var), 1315, 1467  
 gattServerEvent\_t.eventData.charCccdWrittenEvent (C var), 1315, 1467  
 gattServerEvent\_t.eventData.longCharWrittenEvent (C var), 1315, 1467  
 gattServerEvent\_t.eventData.mtuChangedEvent (C var), 1315, 1467  
 gattServerEvent\_t.eventData.procedureError (C var), 1315, 1467  
 gattServerEvent\_t.eventType (C var), 1314, 1467  
 gattServerEventType\_t (C enum), 1302  
 gattServerEventType\_t.gEvtAttributeRead\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtAttributeWritten\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtAttributeWrittenWithoutResponse\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtCharacteristicCccdWritten\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtError\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtHandleValueConfirmation\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtInvalidPduReceived\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtLongCharacteristicWritten\_c (C enumerator), 1302  
 gattServerEventType\_t.gEvtMtuChanged\_c (C enumerator), 1302  
 gattServerInvalidPdu\_t (C struct), 1314, 1467  
 gattServerInvalidPdu\_t.attOpCode (C var), 1314, 1467  
 gattServerLongCharacteristicWrittenEvent\_t (C struct), 1313, 1466  
 gattServerLongCharacteristicWrittenEvent\_t.aValue (C var), 1313, 1466  
 gattServerLongCharacteristicWrittenEvent\_t.cValueLength (C var), 1313, 1466  
 gattServerLongCharacteristicWrittenEvent\_t.handle (C var), 1313, 1466

`gattServerMtuChangedEvent_t` (C struct), 1313, 1466  
`gattServerMtuChangedEvent_t.newMtu` (C var), 1313, 1466  
`gattServerProcedureError_t` (C struct), 1314, 1467  
`gattServerProcedureError_t.error` (C var), 1314, 1467  
`gattServerProcedureError_t.procedureType` (C var), 1314, 1467  
`gattServerProcedureType_t` (C enum), 1303  
`gattServerProcedureType_t.gSendAttributeReadStatus_c` (C enumerator), 1303  
`gattServerProcedureType_t.gSendAttributeWrittenStatus_c` (C enumerator), 1303  
`gattServerProcedureType_t.gSendIndication_c` (C enumerator), 1303  
`gattServerProcedureType_t.gSendMultipleValNotification_c` (C enumerator), 1303  
`gattServerProcedureType_t.gSendNotification_c` (C enumerator), 1303  
`gattService_t` (C type), 1274  
`gattService_tag` (C struct), 1276, 1463  
`gattService_tag.aCharacteristics` (C var), 1277, 1463  
`gattService_tag.aIncludedServices` (C var), 1277, 1463  
`gattService_tag.cNumCharacteristics` (C var), 1277, 1463  
`gattService_tag.cNumIncludedServices` (C var), 1277, 1463  
`gattService_tag.endHandle` (C var), 1276, 1463  
`gattService_tag.startHandle` (C var), 1276, 1463  
`gattService_tag.uuid` (C var), 1277, 1463  
`gattService_tag.uuidType` (C var), 1276, 1463  
`gBasicFilteredScan_c` (C macro), 1371  
`gBasicUnfilteredScan_c` (C macro), 1371  
`gBleAddrTypePublic_c` (C macro), 1370  
`gBleAddrTypeRandom_c` (C macro), 1370  
`gBleAdvTxPowerNoPreference_c` (C macro), 1405  
`gBleBondDataDescriptorSize_c` (C macro), 1403  
`gBleBondDataDeviceInfoSize_c` (C macro), 1403  
`gBleBondDataDynamicSize_c` (C macro), 1403  
`gBleBondDataLegacySize_c` (C macro), 1403  
`gBleBondDataSize_c` (C macro), 1403  
`gBleBondDataStaticSize_c` (C macro), 1403  
`gBleCteMaxTxOctets_c` (C macro), 1405  
`gBleCteMaxTxTime_c` (C macro), 1405  
`gBleCteMinTxOctets_c` (C macro), 1404  
`gBleCteMinTxTime_c` (C macro), 1405  
`gBleEattMaxConnectionChannels` (C var), 1367  
`gBleEattPsmMtu` (C var), 1367  
`gBleExtAdvDefaultSetHandle_c` (C macro), 1405  
`gBleExtAdvDefaultSetId_c` (C macro), 1405  
`gBleExtAdvLegacySetHandle_c` (C macro), 1405  
`gBleExtAdvLegacySetId_c` (C macro), 1405  
`gBleExtAdvMaxAuxOffsetUsec_c` (C macro), 1406  
`gBleExtAdvMaxSetId_c` (C macro), 1405  
`gBleExtAdvNoDuration_c` (C macro), 1405  
`gBleExtAdvNoMaxEvents_c` (C macro), 1405  
`gBleHighDutyDirectedAdvDuration` (C macro), 1405  
`gBleMaxActiveConnections` (C var), 1367  
`gBleMaxADStructureLength_c` (C macro), 1406  
`gBleMaxExtAdvDataLength_c` (C macro), 1406  
`gBleMaxTxOctets_c` (C macro), 1404  
`gBleMaxTxTime_c` (C macro), 1404  
`gBleMaxTxTimeCodedPhy_c` (C macro), 1404  
`gBleMinTxOctets_c` (C macro), 1404  
`gBleMinTxTime_c` (C macro), 1404  
`gBleP256KeyLength_c` (C macro), 1406  
`gBlePeriodicAdvDefaultHandle_c` (C macro), 1405  
`gBlePeriodicAdvMaxSyncHandle_c` (C macro), 1405

[gBlePeriodicAdvOngoingSyncCancelHandle \(C macro\), 1233](#)  
[gBlePeriodicAdvSkipMax\\_c \(C macro\), 1405](#)  
[gBlePeriodicAdvSyncTimeoutMax\\_c \(C macro\), 1405](#)  
[gBlePeriodicAdvSyncTimeoutMin\\_c \(C macro\), 1405](#)  
[gBleSig\\_AlertLevel\\_d \(C macro\), 1375](#)  
[gBleSig\\_AlertNotifControlPoint\\_d \(C macro\), 1377](#)  
[gBleSig\\_AlertNotificationService\\_d \(C macro\), 1374](#)  
[gBleSig\\_AlertStatus\\_d \(C macro\), 1377](#)  
[gBleSig\\_BatteryLevel\\_d \(C macro\), 1376](#)  
[gBleSig\\_BatteryService\\_d \(C macro\), 1374](#)  
[gBleSig\\_BloodPressureFeature\\_d \(C macro\), 1378](#)  
[gBleSig\\_BloodPressureService\\_d \(C macro\), 1374](#)  
[gBleSig\\_BodySensorLocation\\_d \(C macro\), 1377](#)  
[gBleSig\\_BootKeyboardInputReport\\_d \(C macro\), 1376](#)  
[gBleSig\\_BootKeyboardOutputReport\\_d \(C macro\), 1376](#)  
[gBleSig\\_BootMouseInputReport\\_d \(C macro\), 1377](#)  
[gBleSig\\_BpEnhancedMeasurement\\_d \(C macro\), 1377](#)  
[gBleSig\\_BpMeasurement\\_d \(C macro\), 1377](#)  
[gBleSig\\_BpRecord\\_d \(C macro\), 1377](#)  
[gBleSig\\_BtpService\\_d \(C macro\), 1375](#)  
[gBleSig\\_CAR\\_NotSupported\\_d \(C macro\), 1380](#)  
[gBleSig\\_CAR\\_Supported\\_d \(C macro\), 1380](#)  
[gBleSig\\_CCCD\\_d \(C macro\), 1373](#)  
[gBleSig\\_CentralAddressResolution\\_d \(C macro\), 1379](#)  
[gBleSig\\_Characteristic\\_d \(C macro\), 1372](#)  
[gBleSig\\_CharAggregateFormat\\_d \(C macro\), 1373](#)  
[gBleSig\\_CharExtendedProperties\\_d \(C macro\), 1373](#)  
[gBleSig\\_CharPresFormatDescriptor\\_d \(C macro\), 1373](#)  
[gBleSig\\_CharUserDescription\\_d \(C macro\), 1373](#)  
[gBleSig\\_CpControlPoint\\_d \(C macro\), 1379](#)  
[gBleSig\\_CpFeature\\_d \(C macro\), 1379](#)  
[gBleSig\\_CpMeasurement\\_d \(C macro\), 1379](#)  
[gBleSig\\_CpVector\\_d \(C macro\), 1379](#)  
[gBleSig\\_CscFeature\\_d \(C macro\), 1378](#)  
[gBleSig\\_CscMeasurement\\_d \(C macro\), 1378](#)  
[gBleSig\\_CurrentTime\\_d \(C macro\), 1376](#)  
[gBleSig\\_CurrentTimeService\\_d \(C macro\), 1373](#)  
[gBleSig\\_CyclingPowerService\\_d \(C macro\), 1374](#)  
[gBleSig\\_CyclingSpeedAndCadenceService\\_d \(C macro\), 1374](#)  
[gBleSig\\_DeviceInformationService\\_d \(C macro\), 1374](#)  
[gBleSig\\_EncryptedDataKeyMaterial\\_d \(C macro\), 1380](#)  
[gBleSig\\_FirmwareRevisionString\\_d \(C macro\), 1376](#)  
[gBleSig\\_GapAppearance\\_d \(C macro\), 1375](#)  
[gBleSig\\_GapDeviceName\\_d \(C macro\), 1375](#)  
[gBleSig\\_GapPpcp\\_d \(C macro\), 1375](#)  
[gBleSig\\_GattClientSupportedFeatures\\_d \(C macro\), 1375](#)  
[gBleSig\\_GattDatabaseHash\\_d \(C macro\), 1375](#)  
[gBleSig\\_GattSecurityLevels\\_d \(C macro\), 1380](#)  
[gBleSig\\_GattServerSupportedFeatures\\_d \(C macro\), 1375](#)  
[gBleSig\\_GattServiceChanged\\_d \(C macro\), 1375](#)  
[gBleSig\\_GenericAccessProfile\\_d \(C macro\), 1373](#)  
[gBleSig\\_GenericAttributeProfile\\_d \(C macro\), 1373](#)  
[gBleSig\\_GlucoseFeature\\_d \(C macro\), 1378](#)  
[gBleSig\\_GlucoseMeasurement\\_d \(C macro\), 1375](#)  
[gBleSig\\_GlucoseMeasurementContext\\_d \(C macro\), 1377](#)  
[gBleSig\\_GlucoseService\\_d \(C macro\), 1373](#)  
[gBleSig\\_HardwareRevisionString\\_d \(C macro\), 1376](#)  
[gBleSig\\_HealthThermometerService\\_d \(C macro\), 1374](#)

[gBleSig\\_HeartRateService\\_d \(C macro\), 1374](#)  
[gBleSig\\_HidBootMouseInputReport\\_d \(C macro\), 1378](#)  
[gBleSig\\_HidCtrlPoint\\_d \(C macro\), 1378](#)  
[gBleSig\\_HidInformation\\_d \(C macro\), 1378](#)  
[gBleSig\\_HidService\\_d \(C macro\), 1374](#)  
[gBleSig\\_HrControlPoint\\_d \(C macro\), 1377](#)  
[gBleSig\\_HrMeasurement\\_d \(C macro\), 1377](#)  
[gBleSig\\_HTTP\\_ControlPoint\\_d \(C macro\), 1380](#)  
[gBleSig\\_HTTP\\_EntityBody\\_d \(C macro\), 1380](#)  
[gBleSig\\_HTTP-Headers\\_d \(C macro\), 1380](#)  
[gBleSig\\_HTTP\\_StatusCode\\_d \(C macro\), 1380](#)  
[gBleSig\\_HTTPProxyService\\_d \(C macro\), 1374](#)  
[gBleSig\\_HTTPS\\_Security\\_d \(C macro\), 1380](#)  
[gBleSig\\_IeeeRcdl\\_d \(C macro\), 1376](#)  
[gBleSig\\_ImmediateAlertService\\_d \(C macro\), 1373](#)  
[gBleSig\\_Include\\_d \(C macro\), 1372](#)  
[gBleSig\\_IntermediateCuffPressure\\_d \(C macro\), 1377](#)  
[gBleSig\\_IntermediateTemperature\\_d \(C macro\), 1376](#)  
[gBleSig\\_IpsService\\_d \(C macro\), 1374](#)  
[gBleSig\\_LinkLossService\\_d \(C macro\), 1373](#)  
[gBleSig\\_LnControlPoint\\_d \(C macro\), 1379](#)  
[gBleSig\\_LnFeature\\_d \(C macro\), 1379](#)  
[gBleSig\\_LocalTimeInformation\\_d \(C macro\), 1375](#)  
[gBleSig\\_LocationAndNavigationService\\_d \(C macro\), 1374](#)  
[gBleSig\\_LocationAndSpeed\\_d \(C macro\), 1379](#)  
[gBleSig\\_ManufacturerNameString\\_d \(C macro\), 1376](#)  
[gBleSig\\_MeasurementInterval\\_d \(C macro\), 1376](#)  
[gBleSig\\_MeshProvDataIn\\_d \(C macro\), 1380](#)  
[gBleSig\\_MeshProvDataOut\\_d \(C macro\), 1380](#)  
[gBleSig\\_MeshProvisioningService\\_d \(C macro\), 1380](#)  
[gBleSig\\_MeshProxyDataIn\\_d \(C macro\), 1380](#)  
[gBleSig\\_MeshProxyDataOut\\_d \(C macro\), 1380](#)  
[gBleSig\\_MeshProxyService\\_d \(C macro\), 1380](#)  
[gBleSig\\_ModelNumberString\\_d \(C macro\), 1376](#)  
[gBleSig\\_Navigation\\_d \(C macro\), 1379](#)  
[gBleSig\\_NewAlert\\_d \(C macro\), 1377](#)  
[gBleSig\\_NextDSTChangeService\\_d \(C macro\), 1373](#)  
[gBleSig\\_PhoneAlertStatusService\\_d \(C macro\), 1374](#)  
[gBleSig\\_PlxContMeasurement\\_d \(C macro\), 1379](#)  
[gBleSig\\_PlxSCMeasurement\\_d \(C macro\), 1379](#)  
[gBleSig\\_PnpId\\_d \(C macro\), 1378](#)  
[gBleSig\\_PositionQuality\\_d \(C macro\), 1379](#)  
[gBleSig\\_PrimaryService\\_d \(C macro\), 1372](#)  
[gBleSig\\_ProtocolMode\\_d \(C macro\), 1378](#)  
[gBleSig\\_PulseOximeterFeature\\_d \(C macro\), 1379](#)  
[gBleSig\\_PulseOximeterService\\_d \(C macro\), 1374](#)  
[gBleSig\\_RaCtrlPoint\\_d \(C macro\), 1378](#)  
[gBleSig\\_RangingService\\_d \(C macro\), 1381](#)  
[gBleSig\\_RasControlPoint\\_d \(C macro\), 1381](#)  
[gBleSig\\_RasFeature\\_d \(C macro\), 1381](#)  
[gBleSig\\_RasOnDemandProcData\\_d \(C macro\), 1381](#)  
[gBleSig\\_RasprocDataOverwritten\\_d \(C macro\), 1381](#)  
[gBleSig\\_RasProcDataReady\\_d \(C macro\), 1381](#)  
[gBleSig\\_RasRealTimeProcData\\_d \(C macro\), 1381](#)  
[gBleSig\\_ReferenceTimeInformation\\_d \(C macro\), 1375](#)  
[gBleSig\\_ReferenceTimeUpdateService\\_d \(C macro\), 1373](#)  
[gBleSig\\_Report\\_d \(C macro\), 1378](#)  
[gBleSig\\_ResolvablePrivateAddressOnly\\_d \(C macro\), 1380](#)

[gBleSig\\_RingerControlPoint\\_d \(C macro\), 1377](#)  
[gBleSig\\_RingerSetting\\_d \(C macro\), 1377](#)  
[gBleSig\\_RPAO\\_Used\\_d \(C macro\), 1381](#)  
[gBleSig\\_RscFeature\\_d \(C macro\), 1378](#)  
[gBleSig\\_RscMeasurement\\_d \(C macro\), 1378](#)  
[gBleSig\\_RunningSpeedAndCadenceService\\_d \(C macro\), 1374](#)  
[gBleSig\\_ScanIntervalWindow\\_d \(C macro\), 1378](#)  
[gBleSig\\_SCCD\\_d \(C macro\), 1373](#)  
[gBleSig\\_ScControlPoint\\_d \(C macro\), 1378](#)  
[gBleSig\\_SecondaryService\\_d \(C macro\), 1372](#)  
[gBleSig\\_SensorLocation\\_d \(C macro\), 1379](#)  
[gBleSig\\_SerialNumberString\\_d \(C macro\), 1376](#)  
[gBleSig\\_SoftwareRevisionString\\_d \(C macro\), 1376](#)  
[gBleSig\\_SupportedNewAlertCategory\\_d \(C macro\), 1377](#)  
[gBleSig\\_SupportedUnreadAlertCategory\\_d \(C macro\), 1378](#)  
[gBleSig\\_SystemId\\_d \(C macro\), 1376](#)  
[gBleSig\\_Temperature\\_d \(C macro\), 1379](#)  
[gBleSig\\_TemperatureMeasurement\\_d \(C macro\), 1376](#)  
[gBleSig\\_TemperatureType\\_d \(C macro\), 1376](#)  
[gBleSig\\_TimeUpdateControlPoint\\_d \(C macro\), 1375](#)  
[gBleSig\\_TimeUpdateState\\_d \(C macro\), 1375](#)  
[gBleSig\\_TimeWithDST\\_d \(C macro\), 1375](#)  
[gBleSig\\_TxPower\\_d \(C macro\), 1375](#)  
[gBleSig\\_TxPowerService\\_d \(C macro\), 1373](#)  
[gBleSig\\_UnreadAlertStatus\\_d \(C macro\), 1377](#)  
[gBleSig\\_URI\\_d \(C macro\), 1379](#)  
[gBleSig\\_ValidRangeDescriptor\\_d \(C macro\), 1373](#)  
[gBleSig\\_WPTService\\_d \(C macro\), 1374](#)  
[gBleUuidType16\\_c \(C macro\), 1370](#)  
[gBleUuidType32\\_c \(C macro\), 1370](#)  
[gBleUuidType128\\_c \(C macro\), 1370](#)  
[gbmpDIAM\\_ConnectableUndirected\\_c \(C macro\), 1236](#)  
[gbmpDIAM\\_NonConnectableNonScannableUndirected\\_c \(C macro\), 1235](#)  
[gbmpDIAM\\_ScannableUndirected\\_c \(C macro\), 1236](#)  
[gBrEdrNotSupported\\_c \(C macro\), 1233](#)  
[gCancelOngoingInitiatingConnection\\_d \(C macro\), 1230](#)  
[gcBleChannelMapSize\\_c \(C macro\), 1404](#)  
[gcBleDeviceAddressSize\\_c \(C macro\), 1403](#)  
[gcBleLongUuidSize\\_c \(C macro\), 1403](#)  
[gCccdEmpty\\_c \(C macro\), 1275](#)  
[gCccdIndication\\_c \(C macro\), 1275](#)  
[gCccdNotification\\_c \(C macro\), 1275](#)  
[gcConnectionEventMaxDefault\\_c \(C var\), 1367](#)  
[gcConnectionEventMinDefault\\_c \(C var\), 1367](#)  
[gcConnectionIntervalMax\\_c \(C macro\), 1369](#)  
[gcConnectionIntervalMaxDefault\\_c \(C macro\), 1369](#)  
[gcConnectionIntervalMin\\_c \(C macro\), 1369](#)  
[gcConnectionIntervalMinDefault\\_c \(C macro\), 1369](#)  
[gcConnectionPeripheralLatencyDefault\\_c \(C macro\), 1369](#)  
[gcConnectionPeripheralLatencyMax\\_c \(C macro\), 1369](#)  
[gcConnectionSupervisionTimeoutDefault\\_c \(C macro\), 1369](#)  
[gcConnectionSupervisionTimeoutMax\\_c \(C macro\), 1369](#)  
[gcConnectionSupervisionTimeoutMin\\_c \(C macro\), 1369](#)  
[gcDecisionDataKeySize\\_c \(C macro\), 1406](#)  
[gcDecisionDataMaxSize\\_c \(C macro\), 1406](#)  
[gcDecisionDataPrandSize\\_c \(C macro\), 1406](#)  
[gcDecisionDataResolvableTagSize\\_c \(C macro\), 1406](#)  
[gcDecisionInstructionsArbitraryDataMaskSize\\_c \(C macro\), 1406](#)

gcDecisionInstructionsArbitraryDataTargetSize\_c (C macro), 1406  
gcDecisionInstructionsParamSize\_c (C macro), 1406  
gcEadAadValue\_c (C macro), 1407  
gcEadIvSize\_c (C macro), 1406  
gcEadKeySize\_c (C macro), 1406  
gcEadMicSize\_c (C macro), 1406  
gcEadRandomizerSize\_c (C macro), 1406  
gcGapMaxAdvertisingDataLength\_c (C macro), 1404  
gcGapMaxAuthorizationHandles\_c (C macro), 1403  
gcGapMaxDeviceNameSize\_c (C macro), 1404  
gcGapMaximumSavedCccds\_c (C macro), 1403  
gcMaxNumTestsInDecisionInstruction\_c (C macro), 1406  
gcReservedFlashSizeForCustomInformation\_c (C macro), 1404  
gcSecureModeSavedLocalKeysNo\_c (C macro), 1403  
gcSmpCsrkSize\_c (C macro), 1403  
gcSmpIrkSize\_c (C macro), 1403  
gcSmpMaxBlobSize\_c (C macro), 1403  
gcSmpMaxIrkBlobSize\_c (C macro), 1403  
gcSmpMaxLtkSize\_c (C macro), 1403  
gcSmpMaxRandSize\_c (C macro), 1403  
gcSmpOobSize\_c (C macro), 1403  
gCsrk\_c (C macro), 1233  
gDefaultEncryptionKeySize\_d (C macro), 1230  
gDevicePrivacy\_c (C macro), 1372  
gDHKeySize\_c (C macro), 1372  
gEattMaxMtu\_c (C macro), 1404  
gEattMinMtu\_c (C macro), 1404  
gEnhancedL2capMinimumMps\_c (C macro), 1414  
getConnParams\_t (C type), 1365  
getConnParams\_tag (C struct), 1387, 1421  
getConnParams\_tag.aChMapBm (C var), 1387, 1421  
getConnParams\_tag.aCrcInitVal (C var), 1387, 1421  
getConnParams\_tag.aucRemoteMasRxPHY (C var), 1388, 1422  
getConnParams\_tag.connectionHandle (C var), 1387, 1421  
getConnParams\_tag.seqNum (C var), 1388, 1422  
getConnParams\_tag.ucCentralSCA (C var), 1388, 1422  
getConnParams\_tag.ucChannelSelection (C var), 1387, 1421  
getConnParams\_tag.ucHop (C var), 1387, 1421  
getConnParams\_tag.ucRole (C var), 1388, 1422  
getConnParams\_tag.ucUnMapChIdx (C var), 1388, 1422  
getConnParams\_tag.uiAnchorDelay (C var), 1388, 1422  
getConnParams\_tag.uiConnEvent (C var), 1388, 1422  
getConnParams\_tag.uiConnInterval (C var), 1387, 1421  
getConnParams\_tag.uiConnLatency (C var), 1387, 1421  
getConnParams\_tag.uiSuperTO (C var), 1387, 1421  
getConnParams\_tag.ulAnchorClk (C var), 1388, 1422  
getConnParams\_tag.ulRxInstant (C var), 1388, 1422  
getConnParams\_tag.ulTxAccCode (C var), 1387, 1421  
getSecurityLevel (C macro), 1230  
getSecurityMode (C macro), 1230  
gExpandAsEnum\_m (C macro), 1414  
gExpandAsTable\_m (C macro), 1414  
gExtendedFilteredScan\_c (C macro), 1371  
gExtendedUnfilteredScan\_c (C macro), 1371  
gGapAdvertisingChannelMapDefault\_c (C macro), 1231  
gGapAdvertisingIntervalDefault\_c (C macro), 1230  
gGapAdvertisingIntervalRangeMaximum\_c (C macro), 1230  
gGapAdvertisingIntervalRangeMinimum\_c (C macro), 1230

[gGapChSelAlgorithmNo2 \(C macro\), 1233](#)  
[gGapConnEventLengthMax\\_d \(C macro\), 1232](#)  
[gGapConnEventLengthMin\\_d \(C macro\), 1232](#)  
[gGapConnIntervalMax\\_d \(C macro\), 1232](#)  
[gGapConnIntervalMin\\_d \(C macro\), 1232](#)  
[gGapConnLatencyMax\\_d \(C macro\), 1232](#)  
[gGapConnLatencyMin\\_d \(C macro\), 1232](#)  
[gGapConnSuperTimeoutMax\\_d \(C macro\), 1232](#)  
[gGapConnSuperTimeoutMin\\_d \(C macro\), 1232](#)  
[gGapCteMaxCount\\_c \(C macro\), 1235](#)  
[gGapCteMaxLength\\_c \(C macro\), 1235](#)  
[gGapCteMinCount\\_c \(C macro\), 1235](#)  
[gGapCteMinLength\\_c \(C macro\), 1235](#)  
[gGapDefaultAdvertisingParameters\\_d \(C macro\), 1231](#)  
[gGapDefaultConnectionLatency\\_d \(C macro\), 1232](#)  
[gGapDefaultConnectionRequestParameters\\_d \(C macro\), 1233](#)  
[gGapDefaultDeviceSecurity\\_d \(C macro\), 1230](#)  
[gGapDefaultExtAdvertisingParameters\\_d \(C macro\), 1231](#)  
[gGapDefaultMaxConnectionInterval\\_d \(C macro\), 1232](#)  
[gGapDefaultMinConnectionInterval\\_d \(C macro\), 1232](#)  
[gGapDefaultPeriodicAdvParameters\\_d \(C macro\), 1231](#)  
[gGapDefaultScanningParameters\\_d \(C macro\), 1232](#)  
[gGapDefaultSecurityRequirements\\_d \(C macro\), 1230](#)  
[gGapDefaultSupervisionTimeout\\_d \(C macro\), 1232](#)  
[gGapEattMaxBearers \(C macro\), 1235](#)  
[gGapExtAdvertisingIntervalDefault\\_c \(C macro\), 1230](#)  
[gGapExtAdvertisingIntervalRangeMaximum\\_c \(C macro\), 1231](#)  
[gGapExtAdvertisingIntervalRangeMinimum\\_c \(C macro\), 1230](#)  
[gGapInvalidSyncHandle \(C macro\), 1233](#)  
[gGapMaxSwitchingPatternLength\\_c \(C macro\), 1235](#)  
[gGapMinSwitchingPatternLength\\_c \(C macro\), 1235](#)  
[gGapPeriodicAdvIntervalDefault\\_c \(C macro\), 1231](#)  
[gGapPeriodicAdvIntervalRangeMaximum\\_c \(C macro\), 1231](#)  
[gGapPeriodicAdvIntervalRangeMinimum\\_c \(C macro\), 1231](#)  
[gGapRssiMax\\_d \(C macro\), 1231](#)  
[gGapRssiMin\\_d \(C macro\), 1231](#)  
[gGapRssiNotAvailable\\_d \(C macro\), 1232](#)  
[gGapScanContinuously\\_d \(C macro\), 1232](#)  
[gGapScanIntervalDefault\\_d \(C macro\), 1231](#)  
[gGapScanIntervalMax\\_d \(C macro\), 1231](#)  
[gGapScanIntervalMin\\_d \(C macro\), 1231](#)  
[gGapScanPeriodicDisabled\\_d \(C macro\), 1232](#)  
[gGapScanWindowDefault\\_d \(C macro\), 1231](#)  
[gGapScanWindowMax\\_d \(C macro\), 1231](#)  
[gGapScanWindowMin\\_d \(C macro\), 1231](#)  
[gGattDatabaseHashSize\\_c \(C macro\), 1322](#)  
[gGattDbAttributeCount\\_c \(C var\), 1316](#)  
[gGattDbInvalidHandle\\_d \(C macro\), 1322](#)  
[gGattDbInvalidHandleIndex\\_d \(C macro\), 1322](#)  
[gGattDynamicAttrSize \(C var\), 1316](#)  
[gGattDynamicValSize \(C var\), 1316](#)  
[gHci2Host\\_TaskQueue \(C var\), 1367](#)  
[gHciTransportUartChannel\\_c \(C macro\), 1404](#)  
[gIncompatibleIoCapabilities\\_c \(C macro\), 1234](#)  
[gInvalidDeviceId\\_c \(C macro\), 1369](#)  
[gInvalidNvmIndex\\_c \(C macro\), 1369](#)  
[gIoDisplayOnly\\_c \(C macro\), 1235](#)  
[gIoDisplayYesNo\\_c \(C macro\), 1235](#)

[gIoKeyboardDisplay\\_c \(C macro\), 1235](#)  
[gIoKeyboardOnly\\_c \(C macro\), 1235](#)  
[gIoNone\\_c \(C macro\), 1235](#)  
[gIrk\\_c \(C macro\), 1233](#)  
[gL2caLePsmDynamicFirst\\_c \(C macro\), 1414](#)  
[gL2caLePsmDynamicLast\\_c \(C macro\), 1414](#)  
[gL2caLePsmSigAssignedEatt\\_c \(C macro\), 1414](#)  
[gL2caLePsmSigAssignedFirst\\_c \(C macro\), 1414](#)  
[gL2caLePsmSigAssignedLast\\_c \(C macro\), 1414](#)  
[gL2capCidAtt\\_c \(C macro\), 1414](#)  
[gL2capCidLePsmDynamicFirst\\_c \(C macro\), 1414](#)  
[gL2capCidLePsmDynamicLast\\_c \(C macro\), 1414](#)  
[gL2capCidNotApplicable\\_c \(C macro\), 1414](#)  
[gL2capCidNull\\_c \(C macro\), 1413](#)  
[gL2capCidSigAssignedFirst\\_c \(C macro\), 1414](#)  
[gL2capCidSigAssignedLast\\_c \(C macro\), 1414](#)  
[gL2capCidSignaling\\_c \(C macro\), 1414](#)  
[gL2capCidSmp\\_c \(C macro\), 1414](#)  
[gL2capDefaultMps\\_c \(C macro\), 1414](#)  
[gL2capDefaultMtu\\_c \(C macro\), 1414](#)  
[gL2capEnhancedMaxChannels\\_c \(C macro\), 1414](#)  
[gL2capHeaderLength\\_c \(C macro\), 1414](#)  
[gL2capMaximumMps\\_c \(C macro\), 1414](#)  
[gLeGeneralDiscoverableMode\\_c \(C macro\), 1233](#)  
[gLeLimitedDiscoverableMode\\_c \(C macro\), 1233](#)  
[gLePhy1MFlag\\_c \(C macro\), 1370](#)  
[gLePhy2MFlag\\_c \(C macro\), 1370](#)  
[gLePhyCodedFlag\\_c \(C macro\), 1370](#)  
[gLePsmSigAssignedNumbersTable\\_m \(C macro\), 1414](#)  
[gLinkEncryptionFailed\\_c \(C macro\), 1235](#)  
[gLowEncryptionKeySize\\_c \(C macro\), 1234](#)  
[gLtk\\_c \(C macro\), 1233](#)  
[gMaxAdvReportQueueSize \(C var\), 1367](#)  
[gMaxAdvSets\\_c \(C macro\), 1369](#)  
[gMaxEncryptionKeySize\\_d \(C macro\), 1230](#)  
[gMode\\_2\\_Mask\\_d \(C macro\), 1230](#)  
[gNetworkPrivacy\\_c \(C macro\), 1372](#)  
[gNoKeys\\_c \(C macro\), 1233](#)  
[gNone\\_c \(C macro\), 1233](#)  
[gOobNotAvailable\\_c \(C macro\), 1234](#)  
[gPairingNotSupported\\_c \(C macro\), 1234](#)  
[gPermissionFlagReadable\\_c \(C macro\), 1322](#)  
[gPermissionFlagReadWithAuthentication\\_c \(C macro\), 1322](#)  
[gPermissionFlagReadWithAuthorization\\_c \(C macro\), 1322](#)  
[gPermissionFlagReadWithEncryption\\_c \(C macro\), 1322](#)  
[gPermissionFlagWritable\\_c \(C macro\), 1322](#)  
[gPermissionFlagWriteWithAuthentication\\_c \(C macro\), 1322](#)  
[gPermissionFlagWriteWithAuthorization\\_c \(C macro\), 1322](#)  
[gPermissionFlagWriteWithEncryption\\_c \(C macro\), 1322](#)  
[gPermissionNone\\_c \(C macro\), 1322](#)  
[gRepeatedAttempts\\_c \(C macro\), 1235](#)  
[gScanAll\\_c \(C macro\), 1371](#)  
[gScanAllPDUs\\_c \(C macro\), 1372](#)  
[gScanOnlyDecisionPDUs\\_c \(C macro\), 1372](#)  
[gScanOnlyNonDecisionPDUs\\_c \(C macro\), 1371](#)  
[gScanRspDataChange\\_c \(C macro\), 1370](#)  
[gScanWithFilterAcceptList\\_c \(C macro\), 1371](#)  
[gSecurityLevel\\_LeSecureConnections\\_c \(C macro\), 1234](#)

[gSecurityLevel\\_NoMitmProtection\\_c \(C macro\), 1234](#)  
[gSecurityLevel\\_NoSecurity\\_c \(C macro\), 1234](#)  
[gSecurityLevel\\_WithMitmProtection\\_c \(C macro\), 1234](#)  
[gSecurityMode\\_1\\_c \(C macro\), 1233](#)  
[gSecurityMode\\_1\\_Level\\_1\\_c \(C macro\), 1234](#)  
[gSecurityMode\\_1\\_Level\\_2\\_c \(C macro\), 1234](#)  
[gSecurityMode\\_1\\_Level\\_3\\_c \(C macro\), 1234](#)  
[gSecurityMode\\_1\\_Level\\_4\\_c \(C macro\), 1234](#)  
[gSecurityMode\\_2\\_c \(C macro\), 1234](#)  
[gSecurityMode\\_2\\_Level\\_1\\_c \(C macro\), 1234](#)  
[gSecurityMode\\_2\\_Level\\_2\\_c \(C macro\), 1234](#)  
[gSimultaneousLeBrEdrCapableController\\_c \(C macro\), 1233](#)  
[gSimultaneousLeBrEdrCapableHost\\_c \(C macro\), 1233](#)  
[gSkdSize\\_c \(C macro\), 1372](#)  
[gSmpLeScRandomConfirmValueSize\\_c \(C macro\), 1404](#)  
[gSmpLeScRandomValueSize\\_c \(C macro\), 1403](#)  
[gUnenhancedBearerId\\_c \(C macro\), 1372](#)  
[gUnspecifiedReason\\_c \(C macro\), 1235](#)  
[gUseDecisionPDU\\_UseFilterAcceptListForOtherPDUs\\_c \(C macro\), 1371](#)  
[gUseDeviceAddress\\_c \(C macro\), 1370](#)  
[gUseDeviceAddressNoDecisionPDUs\\_c \(C macro\), 1371](#)  
[gUseFilterAcceptList\\_c \(C macro\), 1371](#)  
[gUseFilterAcceptListAllPDUs\\_c \(C macro\), 1371](#)  
[gUseFilterAcceptListNoDecisionPDUs\\_c \(C macro\), 1371](#)  
[gUseHciCommandFlowControl \(C var\), 1367](#)  
[gUseOnlyDecisionPDUs\\_c \(C macro\), 1371](#)  
[gVendorHandoverMaxCsLlContextSize\\_c \(C macro\), 1372](#)  
[gVendorUnitaryTestSize\\_c \(C macro\), 1372](#)

## H

[handoverAnchorMonitorEvent\\_t \(C type\), 1365](#)  
[handoverAnchorMonitorEvent\\_tag \(C struct\), 1390, 1423](#)  
[handoverAnchorMonitorEvent\\_tag.anchorClock625Us \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.anchorDelay \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.chIdx \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.connectionHandle \(C var\), 1390, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.connEvent \(C var\), 1390, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.lqiActive \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.lqiRemote \(C var\), 1390, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.rssiActive \(C var\), 1390, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.rssiRemote \(C var\), 1390, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.statusActive \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.statusRemote \(C var\), 1390, 1424](#)  
[handoverAnchorMonitorEvent\\_tag.ucNbReports \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorPacketContinueEvent\\_t \(C type\), 1366](#)  
[handoverAnchorMonitorPacketContinueEvent\\_tag \(C struct\), 1392, 1425](#)  
[handoverAnchorMonitorPacketContinueEvent\\_tag.connectionHandle \(C var\), 1392, 1425](#)  
[handoverAnchorMonitorPacketContinueEvent\\_tag.packetCounter \(C var\), 1392, 1425](#)  
[handoverAnchorMonitorPacketContinueEvent\\_tag.pduSize \(C var\), 1392, 1425](#)  
[handoverAnchorMonitorPacketContinueEvent\\_tag.pPdu \(C var\), 1392, 1425](#)  
[handoverAnchorMonitorPacketEvent\\_t \(C type\), 1365](#)  
[handoverAnchorMonitorPacketEvent\\_tag \(C struct\), 1391, 1424](#)  
[handoverAnchorMonitorPacketEvent\\_tag.anchorClock625Us \(C var\), 1392, 1425](#)  
[handoverAnchorMonitorPacketEvent\\_tag.anchorDelay \(C var\), 1392, 1425](#)  
[handoverAnchorMonitorPacketEvent\\_tag.chIdx \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorPacketEvent\\_tag.connectionHandle \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorPacketEvent\\_tag.connEvent \(C var\), 1391, 1424](#)  
[handoverAnchorMonitorPacketEvent\\_tag.lqiPacket \(C var\), 1391, 1424](#)

handoverAnchorMonitorPacketEvent\_tag.packetCounter (*C var*), 1391, 1424  
handoverAnchorMonitorPacketEvent\_tag.pduSize (*C var*), 1392, 1425  
handoverAnchorMonitorPacketEvent\_tag.phy (*C var*), 1391, 1424  
handoverAnchorMonitorPacketEvent\_tag.pPdu (*C var*), 1392, 1425  
handoverAnchorMonitorPacketEvent\_tag.rssiPacket (*C var*), 1391, 1424  
handoverAnchorMonitorPacketEvent\_tag.statusPacket (*C var*), 1391, 1424  
handoverAnchorMonitorPacketEvent\_tag.ucNbConnIntervals (*C var*), 1392, 1425  
handoverAnchorNotificationStateChanged\_t (*C type*), 1366  
handoverAnchorNotificationStateChanged\_tag (*C struct*), 1396, 1428  
handoverAnchorNotificationStateChanged\_tag.connectionHandle (*C var*), 1396, 1428  
handoverAnchorSearchStart\_t (*C type*), 1365  
handoverAnchorSearchStart\_tag (*C struct*), 1388, 1422  
handoverAnchorSearchStart\_tag.connectionHandle (*C var*), 1389, 1422  
handoverAnchorSearchStart\_tag.status (*C var*), 1389, 1422  
handoverAnchorSearchStop\_t (*C type*), 1365  
handoverAnchorSearchStop\_tag (*C struct*), 1389, 1422  
handoverAnchorSearchStop\_tag.connectionHandle (*C var*), 1389, 1423  
handoverAnchorSearchStop\_tag.status (*C var*), 1389, 1423  
handoverApplyConnectionUpdateProcedure\_t (*C type*), 1366  
handoverApplyConnectionUpdateProcedure\_tag (*C struct*), 1396, 1427  
handoverApplyConnectionUpdateProcedure\_tag.connectionHandle (*C var*), 1396, 1428  
handoverConnect\_t (*C type*), 1365  
handoverConnect\_tag (*C struct*), 1389, 1423  
handoverConnect\_tag.connectionHandle (*C var*), 1389, 1423  
handoverConnect\_tag.status (*C var*), 1389, 1423  
handoverConnectionUpdateProcedureEvent\_t (*C type*), 1366  
handoverConnectionUpdateProcedureEvent\_tag (*C struct*), 1392, 1425  
handoverConnectionUpdateProcedureEvent\_tag.connectionHandle (*C var*), 1392, 1425  
handoverConnectionUpdateProcedureEvent\_tag.currentEventCounter (*C var*), 1393, 1425  
handoverConnectionUpdateProcedureEvent\_tag.instant (*C var*), 1393, 1425  
handoverConnectionUpdateProcedureEvent\_tag.interval (*C var*), 1393, 1425  
handoverConnectionUpdateProcedureEvent\_tag.latency (*C var*), 1393, 1425  
handoverConnectionUpdateProcedureEvent\_tag.timeout (*C var*), 1393, 1425  
handoverConnectionUpdateProcedureEvent\_tag.winOffset (*C var*), 1393, 1425  
handoverConnectionUpdateProcedureEvent\_tag.winSize (*C var*), 1392, 1425  
handoverConnParamUpdateEvent\_t (*C type*), 1366  
handoverConnParamUpdateEvent\_tag (*C struct*), 1393, 1426  
handoverConnParamUpdateEvent\_tag.aChMapBm (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.aCrcInitVal (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.aucRemoteMasRxPHY (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.connectionHandle (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.seqNum (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.status (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.ucCentralSCA (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.ucChannelSelection (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.ucHop (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.ucRole (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.ucUnMapChIdx (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.uiAnchorDelay (*C var*), 1395, 1427  
handoverConnParamUpdateEvent\_tag.uiConnEvent (*C var*), 1394, 1427  
handoverConnParamUpdateEvent\_tag.uiConnInterval (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.uiConnLatency (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.uiSuperTO (*C var*), 1394, 1426  
handoverConnParamUpdateEvent\_tag.ulAnchorClk (*C var*), 1395, 1427  
handoverConnParamUpdateEvent\_tag.ulRxInstant (*C var*), 1395, 1427  
handoverConnParamUpdateEvent\_tag.ulTxAccCode (*C var*), 1394, 1426  
handoverGetCsLlContext\_t (*C type*), 1365  
handoverGetCsLlContext\_tag (*C struct*), 1390, 1423

[handoverGetCsLlContext\\_tag.llContext \(C var\), 1390, 1423](#)  
[handoverGetCsLlContext\\_tag.llContextLength \(C var\), 1390, 1423](#)  
[handoverGetCsLlContext\\_tag.responseMask \(C var\), 1390, 1423](#)  
[handoverGetCsLlContext\\_tag.status \(C var\), 1390, 1423](#)  
[handoverGetData\\_t \(C type\), 1365](#)  
[handoverGetData\\_tag \(C struct\), 1389, 1423](#)  
[handoverGetData\\_tag.pData \(C var\), 1389, 1423](#)  
[handoverGetData\\_tag.status \(C var\), 1389, 1423](#)  
[handoverGetTime\\_t \(C type\), 1365](#)  
[handoverGetTime\\_tag \(C struct\), 1388, 1422](#)  
[handoverGetTime\\_tag.slot \(C var\), 1388, 1422](#)  
[handoverGetTime\\_tag.status \(C var\), 1388, 1422](#)  
[handoverGetTime\\_tag.us\\_offset \(C var\), 1388, 1422](#)  
[handoverLIPendingDataIndication\\_t \(C type\), 1366](#)  
[handoverLIPendingDataIndication\\_tag \(C struct\), 1396, 1428](#)  
[handoverLIPendingDataIndication\\_tag.dataSize \(C var\), 1396, 1428](#)  
[handoverLIPendingDataIndication\\_tag.pData \(C var\), 1396, 1428](#)  
[handoverResumeTransmitCompleteEvent\\_t \(C type\), 1366](#)  
[handoverResumeTransmitCompleteEvent\\_tag \(C struct\), 1395, 1427](#)  
[handoverResumeTransmitCompleteEvent\\_tag.connectionHandle \(C var\), 1395, 1427](#)  
[handoverSetData\\_t \(C type\), 1365](#)  
[handoverSetData\\_tag \(C struct\), 1389, 1423](#)  
[handoverSetData\\_tag.pData \(C var\), 1390, 1423](#)  
[handoverSetData\\_tag.status \(C var\), 1390, 1423](#)  
[handoverSuspendTransmitCompleteEvent\\_t \(C type\), 1366](#)  
[handoverSuspendTransmitCompleteEvent\\_tag \(C struct\), 1395, 1427](#)  
[handoverSuspendTransmitCompleteEvent\\_tag.connectionHandle \(C var\), 1395, 1427](#)  
[handoverSuspendTransmitCompleteEvent\\_tag.noOfPendingAclPackets \(C var\), 1395, 1427](#)  
[handoverSuspendTransmitCompleteEvent\\_tag.sizeOfDataNAckInOldestPacket \(C var\), 1395, 1427](#)  
[handoverSuspendTransmitCompleteEvent\\_tag.sizeOfDataTxInOldestPacket \(C var\), 1395, 1427](#)  
[handoverSuspendTransmitCompleteEvent\\_tag.sizeOfPendingAclPackets \(C var\), 1395, 1427](#)  
[handoverTimeSyncEvent\\_t \(C type\), 1366](#)  
[handoverTimeSyncEvent\\_tag \(C struct\), 1393, 1425](#)  
[handoverTimeSyncEvent\\_tag.rssi \(C var\), 1393, 1426](#)  
[handoverTimeSyncEvent\\_tag.rxClkSlot \(C var\), 1393, 1426](#)  
[handoverTimeSyncEvent\\_tag.rxUs \(C var\), 1393, 1426](#)  
[handoverTimeSyncEvent\\_tag.txClkSlot \(C var\), 1393, 1426](#)  
[handoverTimeSyncEvent\\_tag.txUs \(C var\), 1393, 1426](#)  
[handoverUpdateConnParams\\_t \(C type\), 1366](#)  
[handoverUpdateConnParams\\_tag \(C struct\), 1395, 1427](#)  
[handoverUpdateConnParams\\_tag.connectionHandle \(C var\), 1396, 1427](#)  
[handoverUpdateConnParams\\_tag.status \(C var\), 1396, 1427](#)  
[hciHostToControllerInterface\\_t \(C type\), 1366](#)  
[hciPacketType\\_t \(C enum\), 1340](#)  
[hciPacketType\\_t.gHciCommandPacket\\_c \(C enumerator\), 1340](#)  
[hciPacketType\\_t.gHciDataPacket\\_c \(C enumerator\), 1340](#)  
[hciPacketType\\_t.gHciEventPacket\\_c \(C enumerator\), 1340](#)  
[hciPacketType\\_t.gHciIsoDataPacket\\_c \(C enumerator\), 1340](#)  
[hciPacketType\\_t.gHciSynchronousDataPacket\\_c \(C enumerator\), 1340](#)  
[Host\\_TaskHandler \(C function\), 1369](#)

## I

[isMode\\_1 \(C macro\), 1230](#)  
[isMode\\_2 \(C macro\), 1230](#)  
[isSameMode \(C macro\), 1230](#)

## L

[L2ca\\_CancelConnection \(C function\), 1411](#)

L2ca\_ConnectLePsm (*C function*), 1411  
L2ca\_DeregisterLePsm (*C function*), 1410  
L2ca\_DisconnectLeCbChannel (*C function*), 1411  
L2ca\_EnhancedCancelConnection (*C function*), 1413  
L2ca\_EnhancedChannelReconfigure (*C function*), 1413  
L2ca\_EnhancedConnectLePsm (*C function*), 1412  
L2ca\_RegisterLeCbCallbacks (*C function*), 1410  
L2ca\_RegisterLePsm (*C function*), 1410  
L2ca\_SendLeCbData (*C function*), 1412  
L2ca\_SendLeCredit (*C function*), 1412  
l2caChannelStatus\_t (*C enum*), 1408  
l2caChannelStatus\_t.gL2ca\_ChannelStatusChannelBusy\_c (*C enumerator*), 1408  
l2caChannelStatus\_t.gL2ca\_ChannelStatusChannelIdle\_c (*C enumerator*), 1408  
l2caControlCallback\_t (*C type*), 1410  
l2caEattCallback\_t (*C type*), 1410  
l2caEnhancedConnectionComplete\_t (*C type*), 1409  
l2caEnhancedConnectionComplete\_tag (*C struct*), 1415, 1471  
l2caEnhancedConnectionComplete\_tag.aCids (*C var*), 1471  
l2caEnhancedConnectionComplete\_tag.deviceId (*C var*), 1471  
l2caEnhancedConnectionComplete\_tag.initialCredits (*C var*), 1471  
l2caEnhancedConnectionComplete\_tag.noOfChannels (*C var*), 1471  
l2caEnhancedConnectionComplete\_tag.peerMps (*C var*), 1471  
l2caEnhancedConnectionComplete\_tag.peerMtu (*C var*), 1471  
l2caEnhancedConnectionComplete\_tag.result (*C var*), 1471  
l2caEnhancedConnectionRequest\_t (*C type*), 1409  
l2caEnhancedConnectionRequest\_tag (*C struct*), 1415, 1470  
l2caEnhancedConnectionRequest\_tag.aCids (*C var*), 1471  
l2caEnhancedConnectionRequest\_tag.deviceId (*C var*), 1470  
l2caEnhancedConnectionRequest\_tag.initialCredits (*C var*), 1471  
l2caEnhancedConnectionRequest\_tag.lePsm (*C var*), 1470  
l2caEnhancedConnectionRequest\_tag.noOfChannels (*C var*), 1471  
l2caEnhancedConnectionRequest\_tag.peerMps (*C var*), 1471  
l2caEnhancedConnectionRequest\_tag.peerMtu (*C var*), 1471  
l2caEnhancedReconfigureRequest\_t (*C type*), 1409  
l2caEnhancedReconfigureRequest\_tag (*C struct*), 1415, 1471  
l2caEnhancedReconfigureRequest\_tag.aCids (*C var*), 1471  
l2caEnhancedReconfigureRequest\_tag.deviceId (*C var*), 1471  
l2caEnhancedReconfigureRequest\_tag.newMps (*C var*), 1471  
l2caEnhancedReconfigureRequest\_tag.newMtu (*C var*), 1471  
l2caEnhancedReconfigureRequest\_tag.noOfChannels (*C var*), 1471  
l2caEnhancedReconfigureRequest\_tag.result (*C var*), 1471  
l2caEnhancedReconfigureResponse\_t (*C type*), 1409  
l2caEnhancedReconfigureResponse\_tag (*C struct*), 1415, 1472  
l2caEnhancedReconfigureResponse\_tag.deviceId (*C var*), 1472  
l2caEnhancedReconfigureResponse\_tag.result (*C var*), 1472  
l2caErrorSource\_t (*C enum*), 1407  
l2caErrorSource\_t.gL2ca\_CancelConnection\_c (*C enumerator*), 1407  
l2caErrorSource\_t.gL2ca\_DisconnectLePsm\_c (*C enumerator*), 1407  
l2caErrorSource\_t.gL2ca\_EnhancedCancelConnection\_c (*C enumerator*), 1408  
l2caErrorSource\_t.gL2ca\_EnhancedReconfigureReq\_c (*C enumerator*), 1408  
l2caErrorSource\_t.gL2ca\_HandleLeFlowControlCredit\_c (*C enumerator*), 1408  
l2caErrorSource\_t.gL2ca\_HandleRecvLeCbData\_c (*C enumerator*), 1408  
l2caErrorSource\_t.gL2ca\_HandleSendLeCbData\_c (*C enumerator*), 1407  
l2caErrorSource\_t.gL2ca\_SendLeFlowControlCredit\_c (*C enumerator*), 1407  
l2caGenericCallback\_t (*C type*), 1410  
l2caHandoverConnectionComplete\_t (*C type*), 1409  
l2caHandoverConnectionComplete\_tag (*C struct*), 1415, 1469  
l2caHandoverConnectionComplete\_tag.cId (*C var*), 1469

[l2caHandoverConnectionComplete\\_tag.credits \(C var\), 1469](#)  
[l2caHandoverConnectionComplete\\_tag.deviceId \(C var\), 1469](#)  
[l2caHandoverConnectionComplete\\_tag.peerMps \(C var\), 1469](#)  
[l2caHandoverConnectionComplete\\_tag.peerMtu \(C var\), 1469](#)  
[l2caLeCbChannelStatusNotification\\_t \(C type\), 1409](#)  
[l2caLeCbChannelStatusNotification\\_tag \(C struct\), 1415, 1470](#)  
[l2caLeCbChannelStatusNotification\\_tag.cId \(C var\), 1470](#)  
[l2caLeCbChannelStatusNotification\\_tag.deviceId \(C var\), 1470](#)  
[l2caLeCbChannelStatusNotification\\_tag.status \(C var\), 1470](#)  
[l2caLeCbConnectionComplete\\_t \(C type\), 1409](#)  
[l2caLeCbConnectionComplete\\_tag \(C struct\), 1415, 1468](#)  
[l2caLeCbConnectionComplete\\_tag.cId \(C var\), 1468](#)  
[l2caLeCbConnectionComplete\\_tag.deviceId \(C var\), 1468](#)  
[l2caLeCbConnectionComplete\\_tag.initialCredits \(C var\), 1469](#)  
[l2caLeCbConnectionComplete\\_tag.peerMps \(C var\), 1469](#)  
[l2caLeCbConnectionComplete\\_tag.peerMtu \(C var\), 1469](#)  
[l2caLeCbConnectionComplete\\_tag.result \(C var\), 1469](#)  
[l2caLeCbConnectionRequest\\_t \(C type\), 1409](#)  
[l2caLeCbConnectionRequest\\_tag \(C struct\), 1414, 1468](#)  
[l2caLeCbConnectionRequest\\_tag.deviceId \(C var\), 1468](#)  
[l2caLeCbConnectionRequest\\_tag.initialCredits \(C var\), 1468](#)  
[l2caLeCbConnectionRequest\\_tag.lePsm \(C var\), 1468](#)  
[l2caLeCbConnectionRequest\\_tag.peerMps \(C var\), 1468](#)  
[l2caLeCbConnectionRequest\\_tag.peerMtu \(C var\), 1468](#)  
[l2caLeCbConnectionRequestResult\\_t \(C enum\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gCommandRejected\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gInsufficientAuthentication\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gInsufficientAuthorization\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gInsufficientEncryption\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gInsufficientEncryptionKeySize\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gInvalidParameters\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gInvalidSourceCid\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gLePsmNotSupported\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gNoResourcesAvailable\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gResponseTimeout\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gSourceCidAlreadyAllocated\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gSuccessful\\_c \(C enumerator\), 1407](#)  
[l2caLeCbConnectionRequestResult\\_t.gUnacceptableParameters\\_c \(C enumerator\), 1407](#)  
[l2caLeCbControlCallback\\_t \(C type\), 1410](#)  
[l2caLeCbDataCallback\\_t \(C type\), 1409](#)  
[l2caLeCbDisconnection\\_t \(C type\), 1409](#)  
[l2caLeCbDisconnection\\_tag \(C struct\), 1415, 1469](#)  
[l2caLeCbDisconnection\\_tag.cId \(C var\), 1469](#)  
[l2caLeCbDisconnection\\_tag.deviceId \(C var\), 1469](#)  
[l2caLeCbError\\_t \(C type\), 1409](#)  
[l2caLeCbError\\_tag \(C struct\), 1415, 1470](#)  
[l2caLeCbError\\_tag.deviceId \(C var\), 1470](#)  
[l2caLeCbError\\_tag.errorSource \(C var\), 1470](#)  
[l2caLeCbError\\_tag.result \(C var\), 1470](#)  
[l2caLeCbLocalCreditsNotification\\_t \(C type\), 1409](#)  
[l2caLeCbLocalCreditsNotification\\_tag \(C struct\), 1415, 1470](#)  
[l2caLeCbLocalCreditsNotification\\_tag.cId \(C var\), 1470](#)  
[l2caLeCbLocalCreditsNotification\\_tag.deviceId \(C var\), 1470](#)  
[l2caLeCbLocalCreditsNotification\\_tag.localCredits \(C var\), 1470](#)  
[l2caLeCbLowPeerCredits\\_t \(C type\), 1409](#)  
[l2caLeCbLowPeerCredits\\_tag \(C struct\), 1415, 1469](#)  
[l2caLeCbLowPeerCredits\\_tag.cId \(C var\), 1470](#)  
[l2caLeCbLowPeerCredits\\_tag.deviceId \(C var\), 1470](#)

- [l2caLeCbNoPeerCredits\\_t \(C type\), 1409](#)
- [l2caLeCbNoPeerCredits\\_tag \(C struct\), 1415, 1469](#)
- [l2caLeCbNoPeerCredits\\_tag.cId \(C var\), 1469](#)
- [l2caLeCbNoPeerCredits\\_tag.deviceId \(C var\), 1469](#)
- [l2capControlMessage\\_t \(C type\), 1409](#)
- [l2capControlMessage\\_tag \(C struct\), 1415, 1472](#)
- [l2capControlMessage\\_tag.messageData \(C union\), 1415, 1472](#)
- [l2capControlMessage\\_tag.messageData \(C var\), 1472](#)
- [l2capControlMessage\\_tag.messageData.channelStatusNotification \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.connectionComplete \(C var\), 1415, 1472](#)
- [l2capControlMessage\\_tag.messageData.connectionRequest \(C var\), 1415, 1472](#)
- [l2capControlMessage\\_tag.messageData.disconnection \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.enhancedConnComplete \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.enhancedConnRequest \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.error \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.handoverConnectionComplete \(C var\), 1416, 1473](#)
- [l2capControlMessage\\_tag.messageData.localCreditsNotification \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.lowPeerCredits \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.noPeerCredits \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.reconfigureRequest \(C var\), 1416, 1472](#)
- [l2capControlMessage\\_tag.messageData.reconfigureResponse \(C var\), 1416, 1473](#)
- [l2capControlMessage\\_tag.messageType \(C var\), 1472](#)
- [l2capControlMessageType\\_t \(C enum\), 1408](#)
- [l2capControlMessageType\\_t.gL2ca\\_ChannelStatusNotification\\_c \(C enumerator\), 1409](#)
- [l2capControlMessageType\\_t.gL2ca\\_EnhancedReconfigureRequest\\_c \(C enumerator\), 1409](#)
- [l2capControlMessageType\\_t.gL2ca\\_EnhancedReconfigureResponse\\_c \(C enumerator\), 1409](#)
- [l2capControlMessageType\\_t.gL2ca\\_Error\\_c \(C enumerator\), 1409](#)
- [l2capControlMessageType\\_t.gL2ca\\_HandoverConnectionComplete\\_c \(C enumerator\), 1409](#)
- [l2capControlMessageType\\_t.gL2ca\\_LePsmConnectionComplete\\_c \(C enumerator\), 1408](#)
- [l2capControlMessageType\\_t.gL2ca\\_LePsmConnectRequest\\_c \(C enumerator\), 1408](#)
- [l2capControlMessageType\\_t.gL2ca\\_LePsmDisconnectNotification\\_c \(C enumerator\), 1408](#)
- [l2capControlMessageType\\_t.gL2ca\\_LePsmEnhancedConnectionComplete\\_c \(C enumerator\), 1409](#)
- [l2capControlMessageType\\_t.gL2ca\\_LePsmEnhancedConnectRequest\\_c \(C enumerator\), 1409](#)
- [l2capControlMessageType\\_t.gL2ca\\_LocalCreditsNotification\\_c \(C enumerator\), 1408](#)
- [l2capControlMessageType\\_t.gL2ca\\_LowPeerCredits\\_c \(C enumerator\), 1408](#)
- [l2capControlMessageType\\_t.gL2ca\\_NoPeerCredits\\_c \(C enumerator\), 1408](#)
- [l2capReconfigureResponse\\_t \(C enum\), 1408](#)
- [l2capReconfigureResponse\\_t.gDestinationCidInvalid\\_c \(C enumerator\), 1408](#)
- [l2capReconfigureResponse\\_t.gMtuReductionNotAllowed\\_c \(C enumerator\), 1408](#)
- [l2capReconfigureResponse\\_t.gMultipleChannelMpsReductionNotAllowed\\_c \(C enumerator\), 1408](#)
- [l2capReconfigureResponse\\_t.gReconfigurationSuccessful\\_c \(C enumerator\), 1408](#)
- [l2capReconfigureResponse\\_t.gReconfigurationTimeout\\_c \(C enumerator\), 1408](#)
- [l2capReconfigureResponse\\_t.gReconfReserved\\_c \(C enumerator\), 1408](#)
- [l2capReconfigureResponse\\_t.gUnacceptableReconfParameters\\_c \(C enumerator\), 1408](#)
- [leSupportedFeatures\\_t \(C type\), 1364](#)
- [leSupportedFeatures\\_tag \(C enum\), 1344](#)
- [leSupportedFeatures\\_tag.gAdvertisingCodingSelection\\_c \(C enumerator\), 1345](#)
- [leSupportedFeatures\\_tag.gAdvertisingCodingSelectionHostSupport\\_c \(C enumerator\), 1345](#)
- [leSupportedFeatures\\_tag.gLe2MbPhy\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeChannelSelAlg2\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeCodedPhy\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeConnectionParametersRequestProcedure\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeDataPacketLengthExtension\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeEncryption\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeExtendedAdv\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeExtendedRejectIndication\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeExtendedScannerFilterPolicies\\_c \(C enumerator\), 1344](#)
- [leSupportedFeatures\\_tag.gLeLIPrivacy\\_c \(C enumerator\), 1344](#)

[leSupportedFeatures\\_tag.gLeMinNumOfUsedChanProcedure\\_c \(C enumerator\), 1344](#)  
[leSupportedFeatures\\_tag.gLePathLossMonitoring\\_c \(C enumerator\), 1345](#)  
[leSupportedFeatures\\_tag.gLePawrAdvertiser\\_c \(C enumerator\), 1345](#)  
[leSupportedFeatures\\_tag.gLePawrScanner\\_c \(C enumerator\), 1345](#)  
[leSupportedFeatures\\_tag.gLePeriodicAdv\\_c \(C enumerator\), 1344](#)  
[leSupportedFeatures\\_tag.gLePeriodicAdvSyncTransferReceiver\\_c \(C enumerator\), 1345](#)  
[leSupportedFeatures\\_tag.gLePeriodicAdvSyncTransferSender\\_c \(C enumerator\), 1344](#)  
[leSupportedFeatures\\_tag.gLePeripheralInitiatedFeaturesExchange\\_c \(C enumerator\), 1344](#)  
[leSupportedFeatures\\_tag.gLePing\\_c \(C enumerator\), 1344](#)  
[leSupportedFeatures\\_tag.gLePowerClass1\\_c \(C enumerator\), 1344](#)  
[leSupportedFeatures\\_tag.gLePowerControlRequest1\\_c \(C enumerator\), 1345](#)  
[leSupportedFeatures\\_tag.gLePowerControlRequest2\\_c \(C enumerator\), 1345](#)  
[leSupportedFeatures\\_tag.gLeStableModulationIdxRx\\_c \(C enumerator\), 1344](#)  
[leSupportedFeatures\\_tag.gLeStableModulationIdxTx\\_c \(C enumerator\), 1344](#)  
[LL\\_PHY\\_1M \(C macro\), 1372](#)  
[LL\\_PHY\\_2M \(C macro\), 1372](#)  
[LL\\_PHY\\_S2 \(C macro\), 1372](#)  
[LL\\_PHY\\_S8 \(C macro\), 1372](#)

## M

[mServerServiceChangedCCCDHandle \(C var\), 1316](#)  
[mServerServiceChangedCharHandle \(C var\), 1316](#)

## P

[PACKED\\_STRUCT \(C macro\), 1381](#)  
[periodicAdvSetInfoTransferEvent\\_t \(C type\), 1365](#)  
[periodicAdvSetInfoTransferEvent\\_tag \(C struct\), 1386, 1420](#)  
[periodicAdvSetInfoTransferEvent\\_tag.deviceId \(C var\), 1420](#)  
[periodicAdvSetInfoTransferEvent\\_tag.status \(C var\), 1420](#)  
[periodicAdvSetSyncTransferParamsEvent\\_t \(C type\), 1365](#)  
[periodicAdvSetSyncTransferParamsEvent\\_tag \(C struct\), 1386, 1420](#)  
[periodicAdvSetSyncTransferParamsEvent\\_tag.deviceId \(C var\), 1420](#)  
[periodicAdvSetSyncTransferParamsEvent\\_tag.status \(C var\), 1420](#)  
[periodicAdvSyncTransferEvent\\_t \(C type\), 1365](#)  
[periodicAdvSyncTransferEvent\\_tag \(C struct\), 1386, 1420](#)  
[periodicAdvSyncTransferEvent\\_tag.deviceId \(C var\), 1420](#)  
[periodicAdvSyncTransferEvent\\_tag.status \(C var\), 1420](#)  
[pProcedureData \(C var\), 1281](#)  
[procDataStruct\\_t \(C struct\), 1278, 1464](#)  
[procDataStruct\\_t.array \(C union\), 1279, 1465](#)  
[procDataStruct\\_t.array \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.array.aBytes \(C var\), 1279, 1465](#)  
[procDataStruct\\_t.array.aChars \(C var\), 1279, 1465](#)  
[procDataStruct\\_t.array.aDescriptors \(C var\), 1279, 1465](#)  
[procDataStruct\\_t.array.aHandles \(C var\), 1279, 1465](#)  
[procDataStruct\\_t.array.aServices \(C var\), 1279, 1465](#)  
[procDataStruct\\_t.bAllocatedArray \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.charUuid \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.charUuidType \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.index \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.max \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.pOutActualCount \(C union\), 1279, 1464](#)  
[procDataStruct\\_t.pOutActualCount \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.pOutActualCount.pCount8b \(C var\), 1279, 1465](#)  
[procDataStruct\\_t.pOutActualCount.pCount16b \(C var\), 1279, 1465](#)  
[procDataStruct\\_t.reliableLongWrite \(C var\), 1278, 1464](#)  
[procDataStruct\\_t.reqParams \(C union\), 1279, 1465](#)  
[procDataStruct\\_t.reqParams \(C var\), 1278, 1464](#)

procDataStruct\_t.reqParams.ewParams (*C var*), 1280, 1465  
procDataStruct\_t.reqParams.fbtvParams (*C var*), 1279, 1465  
procDataStruct\_t.reqParams.fiParams (*C var*), 1279, 1465  
procDataStruct\_t.reqParams.pwParams (*C var*), 1280, 1465  
procDataStruct\_t.reqParams.rbgParams (*C var*), 1279, 1465  
procDataStruct\_t.reqParams.rbParams (*C var*), 1280, 1465  
procDataStruct\_t.reqParams.rbtParams (*C var*), 1279, 1465  
procDataStruct\_t.reqParams.rmParams (*C var*), 1280, 1465  
procDataStruct\_t.reqParams.rParams (*C var*), 1280, 1465  
procDataStruct\_t.reqParams.swParams (*C var*), 1280, 1465  
procDataStruct\_t.reqParams.wParams (*C var*), 1280, 1465  
procStatus\_t (*C struct*), 1278, 1464  
procStatus\_t.isOngoing (*C var*), 1278, 1464  
procStatus\_t.ongoingProcedurePhase (*C var*), 1278, 1464  
procStatus\_t.ongoingProcedureType (*C var*), 1278, 1464

## S

STATIC (*C macro*), 1369

## U

Utils\_ExtractFourByteValue (*C macro*), 1382  
Utils\_ExtractThreeByteValue (*C macro*), 1382  
Utils\_ExtractTwoByteValue (*C macro*), 1382  
Utils\_PackFourByteValue (*C macro*), 1382  
Utils\_PackThreeByteValue (*C macro*), 1382  
Utils\_PackTwoByteValue (*C macro*), 1382  
Utils\_Copy8 (*C macro*), 1382  
Utils\_Copy16 (*C macro*), 1382  
Utils\_Copy32 (*C macro*), 1382  
Utils\_Copy64 (*C macro*), 1382  
Utils\_ExtractEightByteValue (*C macro*), 1382  
Utils\_ExtractFourByteValue (*C macro*), 1382  
Utils\_ExtractThreeByteValue (*C macro*), 1381  
Utils\_ExtractTwoByteValue (*C macro*), 1381  
Utils\_PackEightByteValue (*C macro*), 1382  
Utils\_PackFourByteValue (*C macro*), 1382  
Utils\_PackThreeByteValue (*C macro*), 1382  
Utils\_PackTwoByteValue (*C macro*), 1382  
Utils\_RevertByteArray (*C macro*), 1382  
Uuid16 (*C macro*), 1381  
Uuid32 (*C macro*), 1381  
UuidArray (*C macro*), 1381

## V

vendorUnitaryTestEvent\_t (*C type*), 1366  
vendorUnitaryTestEvent\_tag (*C struct*), 1397, 1429  
vendorUnitaryTestEvent\_tag.aParam (*C var*), 1429  
vendorUnitaryTestEvent\_tag.paramLength (*C var*), 1429  
vendorUnitaryTestEvent\_tag.status (*C var*), 1429