DA14592

DA14592 Crowd-Sourced Locationing (CSL) Accessory

This document presents the implementation of Crowd-Sourced Locationing (CSL) Accessory on DA14592, combining both solutions involving a finding network for locationing of registered devices, Apple Find My™ network and Google Fast Pair with Find My Device Network Extension, based on Apple Find My Network Accessory Specification Release R2, and Google Fast Pair specification v3.1.2 and Find My Device Network Extension specification v1.3, respectively. A sample project is also presented as an application example, using SDK v10.679.1.12 engineering release for DA14592.

Contents

Contents **1**
Figures **2**
Tables **2**
1. Terms and Definitions **3**
2. References **4**
3. Introduction **5**
3.1 Prerequisites 5
3.2 Crowd-Sourced Locationing (CSL) 5
3.2.1 Apple Find My™ Network 5
3.2.2 Google Fast Pair with Find My Device Network Extension 5
4. Crowd-Sourced Locationing Accessory Application Project **6**
4.1 Software Architecture Overview 6
4.2 Folder Structure and Files 7
4.3 Application Configuration 10
4.4 Apple Find My™ Network Integration 16
4.4.1 API for Application 16
4.4.2 Porting API 17
4.4.3 Bluetooth LE Event Handling 19
4.4.4 Bluetooth LE Advertising 20
4.4.5 Configuration 20
4.5 Google Fast Pair Service/Find My Device Network Framework Integration 20
4.5.1 API for Application 20
4.5.2 Porting API 22
4.5.3 Bluetooth LE Event Handling 23
4.5.4 Bluetooth LE Advertising 24
4.5.5 Configuration 24
5. CSL Accessory Task **24**
5.1 Application Initialization 24
5.2 Application Main Task Loop 26
Revision History **28**

© 2025 Renesas Electronics

Figures

Figure 1. CSL accessory sample application 6

Tables

Table 1. CSL accessory application project files 7
Table 2. Target device and application project specific configuration 10
Table 3. Apple Find My™ network framework specific configuration 12
Table 4. Google Fast Pair Service/Find My Device Network framework specific configuration for Fast Pair 14
Table 5. Google Fast Pair Service/Find My Device Network framework specific configuration for Find My Device Network 15
Table 6. Apple Find My™ network API functions 17
Table 7. Apple Find My™ network porting API functions 18
Table 8. Google Fast Pair Service/Find My Device Network API functions 21
Table 9. Google Fast Pair Service/Find My Device Network porting API functions 22

1. Terms and Definitions

AD	Advertising Data
ADK	Accessory Development Kit
AES	Advanced Encryption Standard
AFMN	Apple Find My™ Network (used in code)
AK	Account Key
ANOS	Accessory Non-Owner Service
API	Application Programming Interface
Bluetooth LE	Bluetooth® Low Energy
CLI	Command Line Interface
CSL	Crowd-Sourced Locationing
DIS	Device Information Service
E2EE	End-to-End Encrypted
EID	Ephemeral ID
EIK	Ephemeral Identity Key
FMDN	Find My Device Network (used in code)
FMNA	Find My™ Network Accessory (used in code)
FP	Fast Pair
GAP	Generic Access Profile
GATT	Generic ATTribute profile
GFP	Google Fast Pair (used in code)
GFPS	Google Fast Pair Service (used in code)
HMAC	Hash-based Message Authentication Code
iOS	Internetwork Operating System
MFi	Made for iPhone/iPod/iPad
MTU	Maximum Transmission Unit
NVM	Non-Volatile Memory
NVMS	Non-Volatile Memory Storage
OS	Operating System
RPA	Random Resolvable Private Address
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SUOTA	Software Update Over-The-Air
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmitter
UTPM	Unwanted Tracking Protection Mode
UUID	Universally Unique Identifier
VES	Virtual EEPROM Storage

2. References

[1] DA14592, Datasheet, Renesas Electronics.

[2] DA14592 Getting Started with the Development Kit, https://lpccs-docs.renesas.com/um-b-166-da1459x_getting_started/index.html.

[3] Find My Network Accessory Specification, Release R2, Apple Inc.

[4] Google Fast Pair Procedure, https://developers.google.com/nearby/fast-pair/specifications/service/gatt.

[5] UM-B-179, DA14592 Apple Find My Network Accessory Reference Application, User Manual, Revision 1.1, Renesas Electronics.

[6] UM-B-178, DA14592 Google Fast Pair with Find My Device Network Extension Reference Application, User Manual, Revision 1.1, Renesas Electronics.

Note 1 References are for the latest published version, unless otherwise indicated.

3. Introduction

The key goals of this document are:

- To present a complete sample project demonstrating a Crowd-Sourced Locationing (CSL) accessory device supporting both Apple Find My™ network and Google Fast Pair/ Find My Device Network.
- To explain extended APIs and additional configurations to what has been already presented in corresponding user manuals in Ref. [5] and Ref. [6], used for integration into application projects.

3.1 Prerequisites

Before starting your work on the Crowd-Sourced Locationing (CSL) accessory implementation on DA14592, you need to download and install the latest e2studio and the latest SDK DA14592 platform. These can be downloaded from DA14592 product page (DA14592 - SmartBond Multi-Core Bluetooth LE 5.2 SoC with Embedded Flash | Renesas). Additionally, for this document a Pro Development Kit (DA14592-016FDEVKT-P - SmartBond™ DA14592 Bluetooth® Low Energy 5.2 SoC Development Kit Pro | Renesas) is required.

3.2 Crowd-Sourced Locationing (CSL)

Crowd-Sourced Locationing (CSL) is made possible with systems like Apple Find My™ network and Google Find My Device Network, which help track lost or stolen personal items using billions of smart devices (e.g. smartphones, tablets etc.) around the world. The principle is that a CSL-enabled device emits a beacon signal, which may be detected by any passing smart device running either the Apple Find My™ Network or Google Find My Device Network software services, reporting the latest known location of the CSL-enabled device based on the reported smart device's location.

3.2.1 Apple Find My™ Network

Apple Find My™ network accessory specification has been introduced by Apple to ease tracking of nearby Bluetooth® Low Energy (Bluetooth® LE) devices. Other iOS devices are used to track down Bluetooth® LE devices.

Apple Find My™ network is a Bluetooth LE service to facilitate Find My™ pairing of Bluetooth LE devices (accessories) with iOS devices. An accessory can only be paired with one Apple ID. A paired device becomes a part of Find My™ network. It starts Bluetooth LE advertising that can be picked up by nearby iOS devices on the network.

Further information regarding implementation of Find My™ network technology can be found in corresponding specification document in Ref. [3]. Registration to Made for iPhone/iPod/iPad (MFi) portal is required.

3.2.2 Google Fast Pair with Find My Device Network Extension

Google Fast Pair Service and Find My Device Network extension specification has been introduced by Google to ease tracking of nearby Bluetooth® Low Energy (Bluetooth® LE) devices.

Google Fast Pair Service is a Bluetooth LE service to facilitate easy pairing of Bluetooth LE devices with as little user interaction as possible. Built on top of it, Find My Device Network is a protocol specification introduced by Google, defining a Bluetooth LE message format for end-to-end encrypted messages exchanged with beacons in proximity, enabling them to be provisioned for Find My Device Network using Bluetooth LE advertising and eventually being tracked by smart devices. More information can be found in Ref. [4] and Ref. [6].

4. Crowd-Sourced Locationing Accessory Application Project

4.1 Software Architecture Overview

The software architecture of the CSL accessory sample application project implementing both finding network device locationing solutions, Apple Find My™ network and Google Fast Pair with Find My Device Network Extension on top of the DA14592 SDK, is shown in Figure 1.

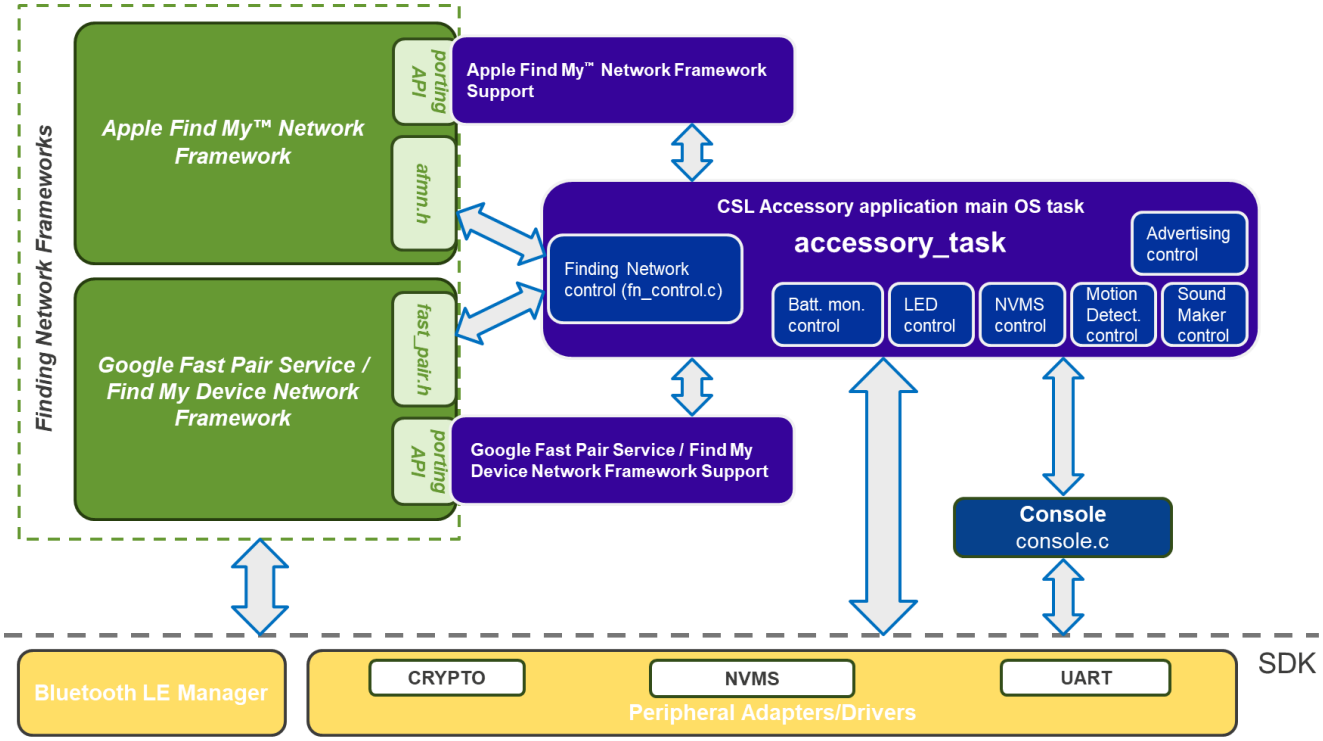


Figure 1. CSL accessory sample application

The DA14592 application software is organized in OS tasks running on top of DA14592 SDK. All corresponding details for application development can be found in Ref. [2].

The main software modules of the CSL accessory sample application are the following:

- **Accessory Task**: This is the main task of the application, and it is responsible, except for controlling the Apple Find My™ network framework and the Google Fast Pair Service/Find My Device Network framework, and implementing their porting APIs, also for controlling access to peripherals, such as NVM storage, battery level monitoring, LEDs (simulating the status of a motion detector and a sound maker) and the user interface with the application executing on DA14592, which includes a button and a console for writing commands and checking the status of the application.
- **Apple Find My™ network framework**: It is essentially a software layer residing on top of Bluetooth LE manager, employing the Apple Find My™ network Bluetooth LE service as well as the underlying system resources (Crypto, NVM storage), to manipulate the Bluetooth LE communication and the corresponding data transfers, the encryption and the storage of required data based on the Apple Find My™ network accessory specification.
- **Google Fast Pair Service/Find My Device Network framework**: It is essentially a software layer residing on top of Bluetooth LE manager, mainly employing the Google Fast Pair and Accessory Non-Owner Bluetooth LE services, as well as the underlying system resources (Crypto, NVM storage), to manipulate the Bluetooth LE communication and the corresponding data transfers, the encryption and the storage of required data based on the Google Fast Pair Service/Find My Device Network specification.

- **Console**: This is UART CLI providing the following commands:
  - **Help** – to show the list of available commands.
  - **Reset** – to reboot the system or Reset factory – to wipe out all pairing data.
  - **Pairmode** – to put the device in pairing mode.
  - **Stopring** – to stop device ringing (applicable for Google Finde My Device Network provisioned device).
  - **Userconsent** – to enable user consent mode.
  - **Advertise** – to start/stop advertising
  - **Motion** – to simulate a motion detection event.
  - **Serial** – to read and display serial number of the accessory.

The accessory has a few operation states that are manipulated by either the Apple Find My™ network framework or the Google Fast Pair Service/Find My Device Network framework. For the user, there are two states that mainly matter, namely paired, where Bluetooth LE advertising frames are sent to enable tracking of the accessory, and unpaired, where the accessory is not yet provisioned for any finding network and a pairing process can be initiated. The state is kept and updated accordingly in NVM storage.

4.2 Folder Structure and Files

Table 1 lists and briefly describes all source and header files of the CSL accessory application project.

Table 1. CSL accessory application project files

Tree	File name	Description
csl_accessory	main.c	File main.c contains main function and initializations of the system.
	accessory_task.c	File accessory_task.c contains the main application task, which handles Bluetooth LE communication, task handlers, control of finding network -related operations, and user interface.
	console.c	File console.c contains implementation for communicating with a terminal over UART.
	platform_devices.c	File plarform_devices.c contains the definition of UART and GPADC controller configuration structures used by the application for communicating with a terminal and monitoring battery voltage level.
└─ afmn_support	afmn_battery_state.c	File afmn_battery_state.c implements functions declared in Apple Find My™ network framework's porting API in afmn_battery_state.h file, enabling access to battery level.
	afmn_ble.c (Note 2)	File afmn_ble.c implements functions declared in Apple Find My™ network framework's porting API in afmn_ble.h file, enabling control of BLE operations (advertising).
	afmn_conn_params.c	File afmn_conn_params.c implements functions declared in Apple Find My™ network framework's porting API in afmn_conn_params.h file, enabling access to corresponding connectivity parameters stored in NVM storage.
	afmn_motion_detector.c	File afmn_motion_detector.c implements functions declared in Apple Find My™ network framework's porting API in afmn_motion_detector.h file, enabling motion detector control.
	afmn_sound_maker.c	File afmn_sound_maker.c implements functions declared in Apple Find My™ network framework's porting API in afmn_sound_maker.h file, enabling control of sound maker.
	afmn_sys_ctrl.c	File afmn_sys_ctrl.c implements functions declared in Apple Find My™ network framework's porting API in afmn_sys_ctrl.h file, enabling control of system resources (system clock frequency).

Tree	File name	Description
└─ config	app_nvparam_values.h	This folder contains all configuration header files for: ● the target device and application ● the Apple Find My™ network framework ● the Google Fast Pair Service/Find My Device Network framework ● the NVM (eFlash/QSPI Flash) storage (partition table and parameters).
	app_nvparam.h	
	custom_config_eflash_suota.h	
	custom_config_eflash.h	
	custom_config_qspi_suota.h	
	custom_config_qspi.h	
	platform_devices.h	
	fw_version.h (Note 2)	
	accessory_config.h	
	afmn_config.h	
	fast_pair_config.h	
	EFLASH/suota/partition_table.h	
	EFLASH/partition_table.h	
	4M/suota/partition_table.h	
	4M/partition_table.h	
└─ fp_support	fp_ble.c (Note 2)	File fp_ble.c implements functions declared in Google Fast Pair Service/Find My Device Network framework's porting API in fp_ble.h file, enabling control of BLE operations (advertising).
	fp_conn_params.c	File fp_conn_params.c implements functions declared in Google Fast Pair Service/Find My Device Network framework's porting API in fp_conn_params.h file, enabling access to corresponding connectivity parameters stored in NVM storage.
	fp_motion_detector.c	File fp_motion_detector.c implements functions declared in Google Fast Pair Service/Find My Device Network framework's fp_motion_detector.h file, enabling motion detector control.
	fp_ring_comp.c	File fp_ring_comp.c implements functions declared in Google Fast Pair Service/Find My Device Network framework's fp_ring_comp.h file, enabling ringing components control.
└─ mbedtls_port	mbedtls_config.h	File mbedtls_config.h includes Mbed TLS configuration macros.
	mbedtls_port.h mbedtls_port.c	Files mbedtls_port.h and mbedtls_port.c implement Mbed TLS integration porting layer.
└─ sdk └─ afmn └─ lib	libfmn_adk_da1459x.a libfmn_adk_dbg_da1459x.a	Apple Find My™ network libraries which contain Apple Find My™ network ADK ported for DA14592. libfmn_adk_dbg_da1459x.a should be used when running Apple Find My™ network accessory application in debug mode, as required by specific test cases.
└─ include	afmn.h	File afmn.h contains top level API functions for controlling Apple Find My™ network framework.
	afmn_battery_state.h	File afmn_battery_state.h contains declaration of functions used by Apple Find My™ network framework for accessing battery level.
	afmn_ble.h (Note 1)	File afmn_ble.h contains declaration of functions used by Apple Find My™ network framework for controlling BLE operations (advertising).
	afmn_conn_params.h	File afmn_conn_params.h contains declaration of functions used by Apple Find My™ network framework for accessing connectivity parameters.
	afmn_defaults.h	File afmn_defaults.h contains default Apple Find My™ network framework configuration parameters.

Tree	File name	Description
	afmn_errors.h	File afmn_errors.h contains enumeration codes for level and category of errors identified by Apple Find My™ network framework.
	afmn_motion_detector.h	File afmn_motion_detector.h contains declaration of functions used by Apple Find My™ network framework for controlling motion detector.
	afmn_sound_maker.h	File afmn_sound_maker.h contains declaration of functions used by Apple Find My™ network framework for controlling sound maker.
	afmn_sys_ctrl.h	File afmn_sys_ctrl.h contains declaration of functions used by Apple Find My™ network framework for controlling system resources (system clock frequency).
	afmn_os_port.h	File afmn_os_port.h contains OS porting API functions used in Apple Find My™ network framework.
└ src	afmn_conf.c	File afmn_conf.c maintains the configuration structure for Apple Find My™ network framework.
	afmn_os_port.c	File afmn_os_port.c implements functions declared in Apple Find My™ network framework's OS porting API in afmn_os_port.h.
└ wolfssl	libwolfssl_afmn_da1459x.a	A port of wolfSSL/wolfCrypt crypto library for DA14592, optimized for use with Apple Find My™ network framework.
└ lib └ port	wc_da1459x.h wc_da1459x.c	
└ sdk └ fast_pair └ ble/services	include/anos.h src/anos.c	Files anos.h and anos.c contain functions for Accessory Non-Owner service handling.
	include/gfps.h src/gfps.c	Files gfps.h and gfps.c contain functions for Google Fast Pair service handling.
└ include	fast_pair.h	File fast_pair.h contains top level API functions for controlling Google Fast Pair Service/Find My Device Network framework.
	fp_ble.h (Note 1)	File fp_ble.h contains declaration of functions used by Google Fast Pair Service/Find My Device Network framework for controlling BLE operations (advertising).
	fp_conn_params.h	File fp_conn_params.h contains declaration of functions used by Google Fast Pair Service/Find My Device Network framework for accessing connectivity parameters.
	fp_defaults.h	File fp_defaults.h contains default Google Fast Pair Service/Find My Device Network framework configuration parameters.
	fp_motion_detector.h	File fp_motion_detector.h contains declaration of functions used by Google Fast Pair Service/Find My Device Network framework for controlling motion detector.
	fp_ring_comp.h	File fp_ring_comp.h contains declaration of functions used by Google Fast Pair Service/Find My Device Network framework for controlling ringing components.
└ src	fp_account_keys.h fp_account_keys.c	Files fp_account_keys.h and fp_account_keys.c implement API for account keys and ephemeral identity key handling operations.
	fp_ano.h fp_ano.c	Files fp_ano.h and fp_ano.c implement the Accessory Non-Owner service module.
	fp_core.h fp_core.c	Files fp_core.h and fp_core.c implement the main module responsible for initialization, Fast Pair and Find My Device Network advertising, factory reset and Fast Pair notification handling for application.
	fp_crypto.h fp_crypto.c	Files fp_crypto.h and fp_crypto.c include utilities for encrypting/decrypting.

Tree	File name	Description
│ │ │ │ │ │ │ │ │ │	fp_fmdn.h fp_fmdn.c	Files fp_fmdn.h and fp_fmdn.c implement Find My Device Network extension module.
	fp_motion_detection.h fp_motion_detection.c	Files fp_motion_detection.h and fp_motion_detection.c implement Find My Device Network motion detection.
	fp_procedure.h fp_procedure.c	Files fp_procedure.h and fp_procedure.c implement Google Fast Pair service module.
	fp_utils.h fp_utils.c	Files fp_utils.h and fp_utils.c include utility functions used internally.
	fp_notifications.h	File fp_notifications.h includes list of notifications handled within Google Fast Pair Service/Find My Device network framework.
└─ utils	adv_control.h (Note 2) adv_control.c (Note 2)	Files adv_control.h and adv_control.c implement functions for controlling Bluetooh LE advertising.
	app_params.h app_params.c	Files app_params.h and app_params.c implement functions supporting access to application parameters stored in NVM storage.
	battery_monitor.h battery_monitor.c	Files battery_monitor.h and battery_monitor.c implement functions for controlling battery level monitoring.
	fn_control.h (Note 2) fn_control.c (Note 2)	Files fn_control.h and fn_control.c implement functions for controlling finding network operations.
	led_control.h led_control.c	Files led_control.h and led_control.c implement functions for controlling device LED.
	motion_detector.h (Note 2) motion_detector.c (Note 2)	Files motion_detector.h and motion_detector.c implement functions for controlling motion detector as required by finding networks.
	sound_maker.h (Note 2) sound_maker.c (Note 2)	Files sound_maker.h and sound_maker.c implement functions for controlling sound maker as required by finding networks.

Note 1 New files added compared to Apple Find My™ network framework and Google Fast Pair Service/Find My Device Network framework implementations presented in Ref. [5] and Ref. [6], respectively.

Note 2 New files added compared to the structure of sample applications presented in Ref. [5] and Ref. [6].

4.3 Application Configuration

Table 2, Table 3, Table 4 and Table 5 list and briefly describe all (compile time) configuration parameters of the CSL accessory application project and of each of the finding network frameworks.

Table 2. Target device and application project specific configuration

Definition	Default value	Header file	Description
`configTOTAL_HEAP_SIZE`	21 * 1024 (+2 * 1024 for SUOTA)	custom_config_eflash.h custom_config_eflash_suota.h custom_config_qspi.h custom_config_qspi_suota.h	The OS total heap size.
`defaultBLE_PPCP_INTERVAL_MIN`	525 ms	custom_config_eflash.h custom_config_eflash_suota.h custom_config_qspi.h custom_config_qspi_suota.h	Minimum connection interval.
`defaultBLE_PPCP_INTERVAL_MAX`	990 ms	custom_config_eflash.h custom_config_eflash_suota.h custom_config_qspi.h custom_config_qspi_suota.h	Maximum connection interval.
`defaultBLE_MTU_SIZE`	512	custom_config_eflash.h custom_config_eflash_suota.h custom_config_qspi.h custom_config_qspi_suota.h	Bluetooth LE MTU size.

Definition	Default value	Header file	Description
ACCESSORY_DEFAULT_NAME	"Renesas CSL Accessory"	accessory_config.h	GAP device name and local name used in scan response.
ADV_CONTROL_MULT_EVENTS_ENABLE (Note 1)	1	accessory_config.h	Support of multiple concurrent advertising events is enabled (using adv_control.h API).
SHORT_PRESS_TIMEOUT_MS	250	accessory_config.h	Short-press button timeout interval used for either stopping ringing (Google Find My Device Network) or enabling user consent mode.
DOUBLE_PRESS_TIMEOUT_MS	800	accessory_config.h	Double-press button timeout interval used for indicating motion detection.
LONG_PRESS_TIMEOUT_MS	4000	accessory_config.h	Long-press button timeout interval used for starting pair mode.
VERY_LONG_PRESS_TIMEOUT_MS	8000	accessory_config.h	Very-long-press button timeout interval used for factory reset.
ADV_STOP_PRESS_TIMEOUT_MS	2000	accessory_config.h	Press button timeout interval used to stop advertising.
ACCESSORY_BATTERIES_COUNT	1	accessory_config.h	Number of available batteries. Values >1 refer to the case where multiple battery-powered peripheral components are controlled.
BATTERY_MONITOR_INTERVAL_MS	5 * 60000	accessory_config.h	Interval for checking battery voltage level periodically.
BATTERY_MONITOR_FULL_LEVEL_MV	3000	accessory_config.h	Fully charged battery voltage level.
BATTERY_MONITOR_EMPTY_LEVEL_MV	1700	accessory_config.h	Empty battery voltage level.
LOW_POWER_MODE_BATTERY_LEVEL	15	accessory_config.h	Battery low power mode threshold (%), applicable only for Google Find My Device Network.
ADV_CONTROL_LOW_POWER_INTERVAL_MS	10 * 60000	accessory_config.h	Battery low power mode advertising interval, applicable only for Google Find My Device Network.
ADV_CONTROL_LOW_POWER_ADV_TIMEOUT_MS	10 * 1000	accessory_config.h	Period during which advertising is enabled in low power mode, applicable only for Google Find My Device Network.
TX_POWER_CONN	GAP_TX_POWER_4_5_dBm	accessory_config.h	TX power used for any GAP connection.
PAIRING_MODE_TIMEOUT_MS	60000	accessory_config.h	Pairing mode timeout interval.
SERIAL_NUMBER_LOOKUP_TIMEOUT_MS	5 * 60000	accessory_config.h	Bluetooth LE serial number lookup timeout interval.

Definition	Default value	Header file	Description
BEACON_TIME_STORAGE_INTERVAL_MS	12 * 60 * 60000	accessory_config.h	Interval at which beacon time is periodically updated in NVM storage, applicable only for Google Find My Device Network.
defaultBLE_DIS_SW_REVISION	SW_VERSION	accessory_config.h	Software revision for Device Information Service. SW_VERSION is in sw_version.h.
defaultBLE_DIS_FW_REVISION	FW_VERSION_MAJOR "." FW_VERSION_MINOR "." FW_VERSION_REVISION	accessory_config.h	Firmware revision for Device Information Service.
USE_CONSOLE (Note 1)	1	accessory_config.h	Console module is enabled, for sending commands over UART.
FW_VERSION_MAJOR (Note 1)	1	fw_version.h	Firmware major version number (value needs to be defined without brackets).
FW_VERSION_MINOR (Note 1)	0	fw_version.h	Firmware minor version number (value needs to be defined without brackets).
FW_VERSION_REVISION (Note 1)	0	fw_version.h	Firmware revision version number (value needs to be defined without brackets).

Note 1 New configuration macros added compared to the configuration of sample applications presented in Ref. [5] and Ref. [6].

Table 3. Apple Find My™ network framework specific configuration

Definition	Default value	Header file	Description
AFMN_CONFIG_FILE	"afmn_config.h"	custom_config_eflash.h custom_config_eflash_suota.h custom_config_qspi.h custom_config_qspi_suota.h	Path of Apple Find My™ network framework configuration file.
AFMN_PRODUCT_DATA	{ 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77 }	afmn_config.h	Product data received from Apple when product plan is submitted and approved. It is 8-byte value.
AFMN_MANUFACTURER_NAME	"Renesas"	afmn_config.h	Manufacturer name.
AFMN_MODEL_NAME	"DA1459x"	afmn_config.h	Model name.
AFMN_ACCESSORY_CATEGORY	1 (Finder)	afmn_config.h	Accessory category.
AFMN_ACCESSORY_CAPABILITY	AFMN_ACCESSORY_CAPABILITY_PLAY_SOUND | AFMN_ACCESSORY_CAPABILITY_UT_MOTION_DETECT | AFMN_ACCESSORY_CAPABILITY_SRNM_LOOKUP_BLE	afmn_config.h	Accessory capability.
AFMN_FW_VERSION_MAJOR	FW_VERSION_MAJOR	afmn_config.h	Firmware major version number (value needs to be defined without brackets).
AFMN_FW_VERSION_MINOR	FW_VERSION_MINOR	afmn_config.h	Firmware minor version number

Definition	Default value	Header file	Description
			(value needs to be defined without brackets).
AFMN_FW_VERSION_REVISION	FW_VERSION_REVISION	afmn_config.h	Firmware revision version number (value needs to be defined without brackets).
AFMN_CONFIG_OPTION (Note 1)	AFMN_CONFIG_OPTION_ BLE_DB_CONTROLLED_BY_APP	afmn_config.h	Configuration option for Apple Find My™ network framework. In CSL accessory application project, BLE database control is performed by the application.
AFMN_BATTERY_TYPE	AFMN_BATTERY_TYPE_ NONRECHARGEABLE	afmn_config.h	Battery type.
__AFMN_PAIR_RETAINED (Note 1)	__attribute__((section("afmn_pair_ retention_mem_zi")))	afmn_defaults.h	Section where retained data are stored that are initialized and used only during Apple Find My™ network pairing process. It cannot be changed.
__AFMN_UNPAIRED_RETAINED (Note 1)	__attribute__((section("afmn_unpaired_ retention_mem_zi")))	afmn_defaults.h	Section where retained data are stored that are used only when the device is not provisioned for Apple Find My™ network. It cannot be changed.
AFMN_BATTERY_LEVEL_MEDIUM	70	afmn_defaults.h	Battery level threshold (%) for medium battery level.
AFMN_BATTERY_LEVEL_LOW	30	afmn_defaults.h	Battery level threshold (%) for low battery level.
AFMN_BATTERY_LEVEL_CRITICAL	10	afmn_defaults.h	Battery level threshold (%) for critical battery level.
AFMN_ADV_TX_POWER	GAP_TX_POWER_4_5_dBm	afmn_defaults.h	TX power used for GAP advertising.
AFMN_TX_POWER_SERVICE_LEVEL	4	afmn_defaults.h	TX power level set in Bluetooth LE TX Power Service.
AFMN_SOUND_DURATION	10 s	afmn_defaults.h	Play sound duration in seconds.
AFMN_TIMER_CLK_ACCURACY	AFMN_TIMER_CLK_ ACCURACY_LOW	afmn_config.h	Accuracy of the OS timer clock source.

Note 1 New configuration macros added, compared to Apple Find My™ network framework configuration presented in Ref. [5].

Table 4. Google Fast Pair Service/Find My Device Network framework specific configuration for Fast Pair

Definition	Default value	Header file	Description
FAST_PAIR_CONFIG_FILE	fast_pair_config.h	custom_config_eflash.h custom_config_eflash_suota.h custom_config_qspi.h custom_config_qspi_suota.h	Path of Google Fast Pair Service/Find My Device Network framework configuration file.
FP_MODEL_ID	0xXXXXXX	fast_pair_config.h	Fast Pair model ID.
FP_BATTERY_NOTIFICATION	1	fp_defaults.h	Battery level notification extension is enabled.
FP_FMDN	1	fast_pair_config.h	Enable the support of Find My Device Network extension.
FP_LOCATOR_TAG	1	fast_pair_config.h	Indicate that the device is a locator tag.
FP_PERSONALIZED_NAME	"Renesas Find My Device Network Accessory" or "Renesas Fast Pair Device"	fast_pair_config.h	Fast Pair personalized name displayed as name for paired device at Android.
FP_PERSONALIZED_NAME_MAX_LENGTH	64	fp_defaults.h	Max. length of Fast Pair personalized name.
FP_ANTI_SPOOFING_PRIVATE_KEY	-	fast_pair_config.h	Assigned by Google anti-spoofing private key for model ID.
FP_ACCOUNT_KEYS_COUNT	5	fp_defaults.h	Maximum number of stored AKs in NVM storage.
FP_CALIBRATED_TX_POWER_LEVEL	-30	fast_pair_config.h	TX power level calculated with Fast Pair Validator app; as received at 0 m (a value in the range [-100, 20]).
FP_ADVERTISE_CALIBRATED_TX_POWER_LEVEL	1	fp_defaults.h	Include TX power level for Fast Pair frames in the advertising payload.
FP_ADV_TX_POWER	GAP_TX_POWER_MINUS_23_dBm	fast_pair_config.h	TX power used for advertising Fast Pair frames.
FP_BATTERIES_COUNT	1	fast_pair_config.h	Number of available batteries.
FP_MAX_PAIRING_FAILURES	10	fp_defaults.h	Allowed number of failed pairing attempts.
FP_AES_HW	1	fp_defaults.h	If 1, then hardware crypto engine is used, otherwise, Mbed TLS library.
FP_DIS_ENABLE (Note 1)	0	fast_pair_config.h	Device Information Service (DIS) is registered by Google Fast Pair Service/Find My Device Network framework. In CSL accessory application project DIS is registered by the application.
FP_CONFIG_OPTION (Note 1)	FP_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP	fast_pair_config.h	Configuration option for Google Fast Pair Service/Find My Device Network framework. In

Definition	Default value	Header file	Description
			CSL accessory application project, BLE database control is performed by the application.
__FP_RETAINED (Note 1)	__RETAINED	fp_defaults.h	Section where retained data are stored that are initialized and used only after Google Fast Pair Service/Find My Device Network framework initialization.
FP_TIMER_CLK_ACCURACY	FP_TIMER_CLK_ACCURACY_LOW	fast_pair_config.h	Accuracy of the OS timer clock source.
FP_LOG_ENABLE	0	fast_pair_config.h	Enable logging for operations related to Fast Pair.

Note 1 New configuration macros added, compared to Google Fast Pair Service/Find My Device Network framework configuration for Fast Pair presented in Ref. [6].

Table 5. Google Fast Pair Service/Find My Device Network framework specific configuration for Find My Device Network

Definition	Default value	Header file	Description
FP_FMDN_MANUFACTURER_NAME	“Renesas”	fast_pair_config.h	Manufacturer name.
FP_FMDN_MODEL_NAME	“DA1459x”	fast_pair_config.h	Model name.
FP_FMDN_ACCESSORY_CATEGORY	1 (location tracker)	fast_pair_config.h	Accessory category.
FP_FMDN_ACCESSORY_CAPABILITIES	FP_FMDN_ACCESSORY_CAPABILITY_PLAY_SOUND | FP_FMDN_ACCESSORY_CAPABILITY_UT_MOTION_DETECT | FP_FMDN_ACCESSORY_CAPABILITY_ID_LOOKUP_BLE	fast_pair_config.h	Capabilities of the Find My Device Network accessory.
FP_FMDN_NETWORK_ID	2	fp_defaults.h	Network ID for Google.
FP_FMDN_FW_VERSION_MAJOR	FW_VERSION_MAJOR (Note 1)	fast_pair_config.h	Firmware major version number (value needs to be defined without brackets).
FP_FMDN_FW_VERSION_MINOR	FW_VERSION_MINOR (Note 1)	fast_pair_config.h	Firmware minor version number (value needs to be defined without brackets).
FP_FMDN_FW_VERSION_REVISION	FW_VERSION_REVISION (Note 1)	fast_pair_config.h	Firmware revision version number (value needs to be defined without brackets).
FP_FMDN_BATTERY_TYPE	FP_FMDN_BATTERY_TYPE_NONRECHARGEABLE	fast_pair_config.h	Battery type.
FP_FMDN_BATTERY_LEVEL_MEDIUM	70	fp_defaults.h	Battery level threshold (%) for medium battery level.
FP_FMDN_BATTERY_LEVEL_LOW	30	fp_defaults.h	Battery level threshold (%) for low battery level.
FP_FMDN_BATTERY_LEVEL_CRITICAL	10	fp_defaults.h	Battery level threshold (%) for critical battery level.
FP_FMDN_CALIBRATED_TX_POWER_LEVEL	2	fast_pair_config.h	Calibrated TX power for Find My Device Network frames as received at 0m.
FP_FMDN_ADV_TX_POWER	GAP_TX_POWER_4_5_dBm	fast_pair_config.h	TX power used by Provider after successful Find My Device Network provisioning.

Definition	Default value	Header file	Description
FP_FMDN_RING_COMPONENTS_NUM	1	fast_pair_config.h	The number of Provider components capable of ringing.
FP_FMDN_RINGING_CAPABILITIES	FP_FMDN_RINGING_VOLUME_AVAILABLE	fast_pair_config.h	Ringing volume selection is available for Provider ringing components.
FP_FMDN_USER_CONSENT_TIMEOUT_MS	5 * 60000	fp_defaults.h	User consent timeout interval for Find My Device Network.
FP_FMDN_PROVISIONING_INIT_TIMEOUT_MS	30 * 1000	fp_defaults.h	Find My Device Network provisioning initiation duration (after power loss).
FP_FMDN_NON_OWNER_PLAY_SOUND_TIMEOUT_MS	12 * 1000	fp_defaults.h	Find My Device Network play sound duration when initiated by non-owner device.
FP_FMDN_MOTION_DETECT_PLAY_SOUND_TIMEOUT_MS	1 * 1000	fp_defaults.h	Find My Device Network play sound duration when motion is detected in separated state.
FP_FMDN_LOG_ENABLE	0	fast_pair_config.h	Enable logging for operations related to Find My Device Network.

Note 1 Changes to configuration macros, compared to Google Fast Pair Service/Find My Device Network framework configuration for Find My Device Network presented in Ref. [6].

4.4 Apple Find My™ Network Integration

The section mainly presents the API and configuration of the Apple Find My™ network framework used for integrating it in an application. Further information, such as how to add Product Data and MFi Token, can be found in Ref. [5].

4.4.1 API for Application

The application can initialize and control the Apple Find My™ network framework using the top-level API defined in **sdk\afmn\include\afmn.h**. Focusing on the Apple Find My™ network accessory implementation, the functions which comprise that top-level API are briefly presented in this section.

The application starts Apple Find My™ network framework by calling `afmn_init(const afmn_config_t *cfg)`. This is the function that needs to be called before any other functions to initialize Apple Find My™ network framework. The configuration structure, which is passed (`cfg`) as argument, contains callback functions that are called during the execution of the corresponding Apple Find My™ network operations for indicating their status to the application. More specifically:

- `execution_cb` – requests application to call `afmn_execution()`, which triggers execution of the Apple Find My™ network framework in application task context.
- `state_cb` – notifies about accessory state changes.
- `db_reset_cb` – notifies about the reset of GATT attribute database initiated by Apple Find My™ network framework. If `AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`, Bluetooth LE services for Apple Find My™ network need to be registered to GATT attribute database by the application, calling `afmn_add_services_to_db()`.
- `error_cb` – notifies about an error occurred during the Apple Find My™ network framework execution.
- `pair_init_cb` – notifies about the initiation of pairing process for Apple Find My™ network provisioning.
- `utc_cb` – notifies about the update of UTC value by Apple Find My™ network.

`pair_init_cb` and `utc_cb` comprise an extension to the configuration structure presented in Ref. [5].

Table 6 lists the rest of the API functions that can be called by an application integrating Apple Find My™ network framework.

Table 6. Apple Find My™ network API functions

API function	Description
`void afmn_deinit(bool suspend)` (Note 1)	Release allocated resources for Apple Find My™ network framework. If argument `suspend` is set to true, then part of framework's state is maintained, so that it can be resumed when calling again `afmn_init()`. If argument `suspend` is set to false, Apple Find My™ network framework cannot be reinitialized, and a device reboot is needed.
`bool afmn_is_initialized(void)` (Note 1)	Check if Apple Find My™ network framework is initialized, so that the rest of API functions can be called.
`void afmn_execution(void)`	Trigger execution of the Apple Find My™ network framework in application task context.
`void afmn_factory_reset(void)`	Force the device into factory default settings for Apple Find My™ network framework. All stored pairing data is wiped out.
`bool afmn_handle_event(ble_evt_hdr_t *evt)` (Note 1)	Handle Bluetooth LE events for the Apple Find My™ network framework. If `AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_NORMAL`, this function must be called instead of `ble_service_handle_event()`, in order for the registered to the Bluetooth LE framework application task to receive Bluetooth LE event notifications. Otherwise, if set to `AFMN_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`, `ble_service_handle_event()` should be called by the application.
`void afmn_set_scan_response(uint8_t structs_count, const gap_adv_ad_struct_t *structs)`	Add structures to Apple Find My™ network scan response data.
`void afmn_set_serial_number_lookup(bool enable)`	Set serial number lookup by Bluetooth LE.
`bool afmn_is_serial_number_lookup_enabled(void)`	Check if serial number lookup by Bluetooth LE is enabled.
`bool afmn_start_pair_mode(void)`	Start Apple Find My™ network pair mode.
`bool afmn_is_pair_mode(void)` (Note 1)	Check if Apple Find My™ network pair mode is enabled.
`AFMN_ACCESSORY_STATE afmn_get_accessory_state(void)`	Return current state of Apple Find My™ network accessory.
`void afmn_add_services_to_db(void)` (Note 1)	Create and register Bluetooth LE services for Apple Find My™ network to the GATT attribute database. It can be used when `AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`.
`void afmn_remove_services_from_db(void)` (Note 1)	Remove Bluetooth LE services for Apple Find My™ network from the GATT attribute database. It can be used when `AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`.

Note 1 Changes in API, compared to Apple Find My™ network API presented in Ref. [5].

4.4.2 Porting API

An Apple Find My™ network accessory requires the use of hardware components like a sound maker, a motion detector, and an NVM storage to support Find My™ network technology. The control of such hardware components, as well as the system underlying resources and Bluetooth LE operations (advertising), is performed using a set of functions comprising the Apple Find My™ network framework's porting API located in **utilities\apple_fmn\include** folder. The declarations of those functions are grouped in several header files, essentially defining the interfaces to those hardware components and system resources required by the framework that need to be implemented by the application on an end-product design. Except for **afmn_os_port.h** which is implemented in **utilities\apple_fmn\src\afmn_os_port.c**, for the rest of the functions declared in the porting API header files, corresponding weak function (empty or default) implementations are provided by Apple Find My™ network libraries, **libfmn_adk_da1459x.a** and **libfmn_adk_dbg_da1459x.a**. Table 7 lists and briefly describes Apple Find My™ network framework's porting API per interface type.

Table 7. Apple Find My™ network porting API functions

Interface	Functions	Description
Battery Monitoring (afmn_battery_state.h)	`uint8_t afmn_battery_state_get_level(void)`	Return the battery level in the range 0 - 100 (%). If not implemented in application, it always returns 100.
Bluetooth LE operations (advertising) control (afmn_ble.h) (Note 1) If not implemented in application, default implementation involves control of Bluetooth LE advertising only for Apple Find My™ network support.	`void afmn_ble_adv_init(void)`	Initialize resources for Bluetooth LE advertising as required for Apple Find My™ network support.
	`void afmn_ble_adv_deinit(void)`	Release resources for Bluetooth LE advertising that have been previously allocated with `afmn_ble_adv_init()`.
	`ble_error_t afmn_ble_adv_set_params(` `const afmn_ble_adv_params_t *params)`	Set Bluetooth LE advertising parameters (discoverability and connectivity modes, interval and TX power level), as required for Apple Find My™ network support, before advertising is started.

	`ble_error_t afmn_ble_adv_set_ad_struct(`
	`size_t ad_len, const gap_adv_ad_struct_t *ad,`
	`size_t sd_len, const gap_adv_ad_struct_t *sd)`
		Set Bluetooth LE advertising and scan response data, as required for Apple Find My™ network support.

	`ble_error_t afmn_ble_adv_start(void)`	Start Bluetooth LE advertising for Apple Find My™ network support.
	`bool afmn_ble_adv_is_started(void)`	Check if Bluetooth LE advertising for Apple Find My™ network support is started.
	`ble_error_t afmn_ble_adv_stop(void)`	Stop Bluetooth LE advertising for Apple Find My™ network support.
	`ble_error_t afmn_ble_adv_stop_all(void)`	Stop sending any Bluetooth LE advertising packets.
Connectivity parameters storage (afmn_conn_params.h) If not implemented in application, default implementation is empty.	`uint16_t afmn_conn_params_get_params(` `afmn_conn_params_t *params, uint8_t n)`	Read a set of Apple Find My™ network connectivity parameters from NVM storage.
	`uint16_t afmn_conn_params_set_params(` `afmn_conn_params_t *params, uint8_t n)`	Write a set of Apple Find My™ network connectivity parameters to NVM storage.
Motion detector (afmn_motion_detector.h) If not implemented in application, default implementation is empty.	`void afmn_motion_detector_init(` `const afmn_motion_detector_config_t *cfg)`	Initialize the motion detector for Apple Find My™ network.
	`void afmn_motion_detector_deinit(void)`	De-initialize the motion detector for Apple Find My™ network.
	`void afmn_motion_detector_enable(void)`	Enable the motion detector for Apple Find My™ network.
	`void afmn_motion_detector_disable(void)`	Disable the motion detector for Apple Find My™ network.
Sound maker (afmn_sound_maker.h) If not implemented in application, default implementation is empty.	`void afmn_sound_maker_init(void)`	Initialize the sound maker for Apple Find My™ network.
	`void afmn_sound_maker_deinit(void)`	De-initialize the sound maker for Apple Find My™ network.
	`void afmn_sound_maker_enable(void)`	Enable the sound maker for Apple Find My™ network.
	`void afmn_sound_maker_disable(void)`	Disable the sound maker for Apple Find My™ network.
System control (afmn_sys_ctrl.h)	`void afmn_sys_ctrl_set_perf(` `AFMN_SYS_CTRL_PERF perf)`	Set processing performance (system clock frequency) for Apple Find My™ network.

Interface	Functions	Description
If not implemented in application, default implementation is empty.	`AFMN_SYS_CTRL_PERF afmn_sys_ctrl_get_perf(void)`	Get current processing performance (system clock frequency) set for Apple Find My™ network. If not implemented in application, it always returns `AFMN_SYS_CTRL_PERF_NORMAL`.
OS resources (afmn_os_port.h) It is implemented in afmn_os_port.c.	`afmn_os_queue_t afmn_os_queue_create( uint32_t elem_size, uint32_t max_elems)`	Create OS queue of a max number of elements of specified size.
	`void afmn_os_queue_delete( afmn_os_queue_t queue)` (Note 1)	Delete OS queue.
	`int afmn_os_queue_put(afmn_os_queue_t queue, const void * const elem, uint32_t timeout)`	Add an element to OS queue.
	`int afmn_os_queue_get(afmn_os_queue_t queue, void * const elem, uint32_t timeout)`	Remove an element from OS queue.
	`int afmn_os_register_task(void)`	Register the current task to OS, to be notified for calling `afmn_os_timer_execution()`.
	`void afmn_os_timer_execution(void)`	Handle pending notifications for OS timer control.
	`afmn_os_timer_t afmn_os_timer_create( AFMN_OS_TIMER_TYPE type, afmn_os_timer_cb_t cb)`	Create OS timer.
	`int afmn_os_timer_start(afmn_os_timer_t timer, uint32_t period)`	Start OS timer.
	`int afmn_os_timer_stop(afmn_os_timer_t timer)`	Stop OS timer.
	`int afmn_os_timer_delete(afmn_os_timer_t timer)`	Delete OS timer.
	`bool afmn_os_timer_is_active( afmn_os_timer_t timer)`	Check if OS time is active.
	`uint32_t afmn_os_timer_get_timer_id( afmn_os_timer_t timer)`	Return OS timer ID.
	`uint32_t afmn_os_ms_to_ticks(uint32_t ms)`	Convert ms to OS ticks.
	`void *afmn_os_malloc(size_t size)`	Allocate memory in OS heap.
	`void afmn_os_free(void *addr)`	De-allocate memory from OS heap.

Note 1 Changes in porting API, compared to Apple Find My™ network porting API presented in Ref. [5].

4.4.3 Bluetooth LE Event Handling

All Apple Find My™ network requests over Bluetooth LE are handled by the Apple Find My™ network framework library. Corresponding Bluetooth LE events are handled by calling `afmn_handle_event()` in application's OS task main loop. In the application's configuration header file for Apple Find My™ network framework, the path of which is defined by `AFMN_CONFIG_FILE`, if `AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_NORMAL`, then `afmn_handle_event()` must be called instead of `ble_service_handle_event()`, as the latter is called internally in the Apple Find My™ network library, in order for the registered to the Bluetooth LE framework application task to receive Bluetooth LE event notifications. Otherwise, if `AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`, then `ble_service_handle_event()` should be called by the application, so that the Bluetooth LE events can be also handled by the registered instances of Bluetooth LE services.

`AFMN_CONFIG_OPTION` value also defines (`AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_NORMAL`) whether Bluetooth LE services instances required for Apple Find My™ network support are registered to Bluetooth LE attribute database internally by the Apple Find My™ network framework, or (`AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`) whether their registration is completely controlled by the application, by calling `afmn_add_services_to_db()` and `afmn_remove_services_from_db()` API functions.

4.4.4 Bluetooth LE Advertising

Apple Find My™ network advertised payload is controlled only within Apple Find My™ network framework library. Corresponding changes to Bluetooth LE advertising parameters and start or stop operations of Apple Find My™ network advertising events are performed by the framework, calling the porting API functions defined in **utilities\apple_fmn\include\afmn_ble.h**, the implementation of which is either provided by the application or it involves, by default (i.e. internal weak function implementations), direct calls to corresponding SDK Bluetooth LE functions defined in **sdk\interfaces\ble\api\include\ble_gap.h**.

More details regarding the advertised payloads can be found in the Apple Find My™ network specification document (Ref. [3]) available in MFi portal.

4.4.5 Configuration

The default configuration for Apple Find My™ network framework is located in **utilities\apple_fmn\include\afmn_defaults.h** header file. The application can overwrite the default configuration preprocessor macro definitions using a configuration file whose path is defined by `AFMN_CONFIG_FILE`. There is a configuration structure (`afmn_const_config`) defined in **utilities\apple_fmn\src\afmn_conf.c**, which is initialized with the values of preprocessor macro definitions and referenced by the Find My™ network ADK library. In the sample application project, this configuration file, called **afmn_config.h**, is located in application's **config** folder, while its relative path is defined by `AFMN_CONFIG_FILE` in **config\custom_config_xx.h** files. The list of definitions that can be changed is presented in Table 3.

4.5 Google Fast Pair Service/Find My Device Network Framework Integration

The section mainly presents the API and configuration of the Google Fast Pair Service/Find My Device Network framework used for integrating it in an application. Further information, such as how to add Fast Pair Model ID and how to perform TX power calibration, as well as implementation details for Google Fast Pair Service/Find My Device Network framework, can be found in Ref. [6].

4.5.1 API for Application

The application can initialize and control the Google Fast Pair Service/Find My Device Network framework using the top-level API defined in **utilities\fast_pair\include\fast_pair.h**. Focusing on the Google Fast Pair Provider with Find My Device Network extension implementation, the functions which comprise that top-level API are briefly presented in this section.

The application starts Google Fast Pair/Find My Device Network framework by calling `fp_init(fp_cfg_t *cfg)`. This is the function that needs to be called before any other to initialize Google Fast Pair Service/Find My Device Network framework. The configuration structure, which is passed (`cfg`) as argument, contains callback functions that are called during the execution of the corresponding Google Fast Pair Service/Find My Device Network operations for indicating their status to the application, as well as system's batteries state information and control of advertising. More specifically:

- `execution_cb` – requests application to call `fp_execution()`, which triggers execution of the Google Fast Pair Service/Find My Device Network framework core module in application task context.
- `beacon_time_cb` – requests application to read and return current beacon time (in seconds), as required by Find My Device Network protocol related operations.
- `pair_status_cb` – notifies about Fast Pair standard Bluetooth LE pairing status.
- `pair_req_status_cb` – notifies about Fast Pair request status.
- `fmdn_prov_status_cb` – notifies about Find My Device Network provisioning status.
- `error_cb` – notifies about an error occurred during the Google Fast Pair Service/Find My Device Network framework execution.
- `batt_info` – defines battery state information.

- `start_adv` – indicates whether advertising will be also started upon the Google Fast Pair Service/Find My Device Network framework initialization.

lists the rest of the API functions that can be called by an application integrating Google Fast Pair Service/Find My Device Network framework.

Table 8. Google Fast Pair Service/Find My Device Network API functions

API function	Description
`void fp_deinit(void)` (Note 1)	Release allocated resources for Google Fast Pair Service/Find My Device Network framework.
`bool fp_is_initialized(void)` (Note 1)	Check if Google Fast Pair Service/Find My Device Network framework is initialized, so that the rest of API functions can be called.
`int fp_execution(void)`	Triggers execution of the Google Fast Pair Service/Find My Device Network framework core module in user application task context.
`int fp_factory_reset(void)`	Forces the device into factory default settings for Google Fast Pair Service/Find My Device Network framework. All stored keys are wiped out.
`bool fp_set_pairing_mode(bool enable)`	Starts or stops the Fast Pair pairing mode and informs the advertising state machine that the device should switch to discoverable pairing mode advertising.
`bool fp_is_pairing_mode(void)`	Informs the application whether Fast Pair mode is enabled or not.
`bool fp_handle_event(ble_evt_hdr_t *evt)`	Handles Bluetooth LE events for the Google Fast Pair Service/Find My Device Network framework and the employed Bluetooth LE services. If `FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_NORMAL`, this function must be called instead of `ble_service_handle_event()`, in order for the registered to the Bluetooth LE framework application task to receive Bluetooth LE event notifications. Otherwise, if set to `FP_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`, `ble_service_handle_event()` should be called by the application.
`int fp_start_advertise(void)`	Starts Google Fast Pair Service/Find My Device Network discoverable or non-discoverable advertising based on whether Fast Pair pairing mode has started or not.
`int fp_stop_advertise(void)`	Stops Google Fast Pair Service/Find My Device Network discoverable or non-discoverable advertising.
`bool fp_is_advertise_stopped(void)`	Checks if Google Fast Pair Service/Find My Device Network advertising is stopped.
`int fp_add_custom_advertise(uint8_t structs_count, const gap_adv_ad_struct_t *structs)`	Adds custom advertise structures to Fast Pair advertising frames payload.
`int fp_set_scan_response(uint8_t structs_count, const gap_adv_ad_struct_t *structs)`	Adds structures to Google Fast Pair Service/Find My Device Network scan response data.
`void fp_set_acc_key_filter_ui_indication( bool enable)`	Sets account key filter UI indication for the Google Fast Pair Service/Find My Device Network framework.
`int fp_set_battery_information( const fp_battery_info_t info[FP_BATTERIES_COUNT])`	Sets current battery status information for the Google Fast Pair Service/Find My Device Network framework.
`void fp_set_battery_ui_indication(bool enable)`	Sets battery UI indication for the Google Fast Pair Service/Find My Device Network framework.
`void fp_add_services_to_db(void)` (Note 1)	Create and register Bluetooth LE services for Google Fast Pair Service/Find My Device Network to the GATT attribute database. It can be used when `FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`.
`void fp_remove_services_from_db(void)` (Note 1)	Remove Bluetooth LE services for Google Fast Pair Service/Find My Device Network from the GATT attribute

API function	Description
	database. It can be used when `FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`.
`int fp_set_advertise_interval(uint16_t interval)`	Sets advertising interval for Find My Device Network frames.
`int fp_stop_ringing(void)`	Stops Google Fast Pair Service/Find My Device Network ringing process.
`bool fp_is_ringing(void)`	Checks if Google Fast Pair Service/Find My Device Network ringing is in progress.
`void fp_set_user_consent(bool enable)`	Enables or disables the Google Fast Pair Service/Find My Device Network user consent period.
`bool fp_is_fmdn_provisioned(void)`	Checks if Provider is Google Fast Pair Service/Find My Device Network provisioned.
`void fp_set_firmware_status(FP_FW_STATUS status)` (Note 1)	Sets Google Fast Pair Service/Find My Device Network firmware status to "normal", "updating", or "abnormal".

Note 1 Changes in API, compared to Google Fast Pair Service/Find My Device Network API presented in Ref. [6].

4.5.2 Porting API

A Google Fast Pair Service/Find My Device Network accessory requires the use of hardware components like a motion detector, a ringing component and an NVM storage. The control of such hardware components, as well as the system underlying resources and Bluetooth LE operations (advertising), is performed using a set of functions comprising the Google Fast Pair Service/Find My Device Network framework's porting API located in **utilities\fast_pair\include** folder. The declarations of those functions are grouped in several header files, essentially defining the interfaces to those hardware components and system resources required by the framework that need to be implemented by the application on an end-product design. For some of the functions declared in the porting API header files, corresponding weak function (empty or default) implementations are provided by Google Fast Pair Service/Find My Device Network framework. Table 9 lists and briefly describes the porting API per interface type.

Table 9. Google Fast Pair Service/Find My Device Network porting API functions

Interface	Functions	Description
Bluetooth LE operations (advertising) control (fp_ble.h) (Note 1) If not implemented in application, default implementation involves control of Bluetooth LE advertising only for Google Fast Pair Service/Find My Device Network support.	`void fp_ble_adv_init(void)`	Initialize resources for Bluetooth LE advertising as required for Google Fast Pair Service/Find My Device Network support.
	`void fp_ble_adv_deinit(void)`	Release resources for Bluetooth LE advertising that have been previously allocated with `fp_ble_adv_init()`.
	`ble_error_t fp_ble_adv_set_params( const fp_ble_adv_params_t *params)`	Set Bluetooth LE advertising parameters (discoverability and connectivity modes, interval and TX power level), as required for Google Fast Pair Service/Find My Device Network support, before advertising is started.
	`ble_error_t fp_ble_adv_set_tx_power( gap_tx_power_t tx_power)`	Set TX power level for Bluetooth LE advertising, as required for Google Fast Pair Service/Find My Device Network support.
	`ble_error_t fp_ble_adv_set_ad_struct( size_t ad_len, const gap_adv_ad_struct_t *ad, size_t sd_len, const gap_adv_ad_struct_t *sd)`	Set Bluetooth LE advertising and scan response data, as required for Google Fast Pair Service/Find My Device Network support.
	`ble_error_t fp_ble_adv_start(void)`	Start Bluetooth LE advertising for Google Fast Pair Service/Find My Device Network support.

Interface	Functions	Description
	`bool fp_ble_adv_is_started(void)`	Check if Bluetooth LE advertising for Google Fast Pair Service/Find My Device Network support is started.
	`ble_error_t fp_ble_adv_stop(void)`	Stop Bluetooth LE advertising for Google Fast Pair Service/Find My Device Network support.
	`ble_error_t fp_ble_adv_stop_all(void)`	Stop sending any Bluetooth LE advertising packets.
Connectivity parameters storage (fp_conn_params.h) Implementation should always be provided by application.	`uint16_t fp_conn_params_get_params( fp_conn_params_t *params, uint8_t n)`	Read a set of Google Fast Pair Service/Find My Device Network connectivity parameters from NVM storage.
	`uint16_t fp_conn_params_set_params( fp_conn_params_t *params, uint8_t n)`	Write a set of Google Fast Pair Service/Find My Device Network connectivity parameters to NVM storage.
Motion detector (fp_motion_detector.h) If not implemented in application, default implementation is empty.	`void fp_motion_detector_init( const fp_motion_detector_config_t *cfg)`	Initialize the motion detector for Google Fast Pair Service/Find My Device Network.
	`void fp_motion_detector_deinit(void)` (Note 1)	De-initialize the motion detector for Google Fast Pair Service/Find My Device Network.
	`void fp_motion_detector_enable(void)`	Enable the motion detector for Google Fast Pair Service/Find My Device Network.
	`void fp_motion_detector_disable(void)`	Disable the motion detector for Google Fast Pair Service/Find My Device Network.
Ringing components (fp_ring_comp.h) Implementation should always be provided by application if `FP_FMDN_RING_COMPONENTS_NUM > 0`.	`void fp_ring_comp_init(void)`	Initialize the sound maker for Google Fast Pair Service/Find My Device Network.
	`void fp_ring_comp_deinit(void)` (Note 1)	De-initialize the sound maker for Google Fast Pair Service/Find My Device Network.
	`int fp_ring_comp_set_state( uint8_t comp_en_msk, FP_RING_COMP_VOLUME volume)`	Enable the sound maker for Google Fast Pair Service/Find My Device Network.

Note 1 Changes in porting API, compared to Google Fast Pair Service/Find My Device Network porting API presented in Ref. [6].

4.5.3 Bluetooth LE Event Handling

All Google Fast Pair Service/Find My Device Network requests over Bluetooth LE are handled by the framework implementation residing in **utilities\fast_pair** folder. Corresponding Bluetooth LE events are handled by calling `fp_handle_event()` in application's OS task main loop. In the application's configuration header file for Google Fast Pair Service/Find My Device Network framework, the path of which is defined by `FP_CONFIG_FILE`, if `FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_NORMAL`, then `fp_handle_event()` must be called instead of `ble_service_handle_event()`, as the latter is called internally in the Google Fast Pair Service/Find My Device Network framework implementation, in order for the registered to the Bluetooth LE framework application task to receive Bluetooth LE event notifications. Otherwise, if `FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`, then `ble_service_handle_event()` should be called by the application, so that the Bluetooth LE events can be also handled by the registered instances of Bluetooth LE services.

`FP_CONFIG_OPTION` value also defines (`FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_NORMAL`) whether Bluetooth LE services instances required for Google Fast Pair Service/Find My Device Network support are registered to Bluetooth LE attribute database internally by the Google Fast Pair Service/Find My Device Network

framework, or (`FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`) whether their registration is completely controlled by the application, by calling `fp_add_services_to_db()` and `fp_remove_services_from_db()` API functions.

4.5.4 Bluetooth LE Advertising

Similarly to Apple Find My™ network framework implementation, Google Fast Pair Service/Find My Device Network advertised payload is controlled only within the corresponding framework. Changes to Bluetooth LE advertising parameters and start or stop operations of Google Fast Pair Service/Find My Device Network advertising events are performed by the framework, calling the porting API functions defined in **utilities\fast_pair\include\fp_ble.h**, the implementation of which is either provided by the application or it involves, by default (i.e. internal weak function implementations), direct calls to corresponding SDK Bluetooth LE functions defined in **sdk\interfaces\ble\api\include\ble_gap.h**.

More details regarding the advertised payloads can be found in Ref. [4] and Ref. [6].

4.5.5 Configuration

The default configuration for Google Fast Pair Service/Find My Device Network framework is located in **utilities\fast_pair\include\fp_defaults.h** header file. The application can overwrite the default configuration preprocessor macro definitions using a configuration file whose path is defined by `FP_CONFIG_FILE`. In the sample application project, this configuration file, called **fast_pair_config.h**, is located in application's **config** folder, while its relative path is defined by `FP_CONFIG_FILE` in **config\custom_config_xx.h** files. The list of definitions that can be changed are presented in Table 4 (Google Fast Pair Service) and Table 5 (Find My Device Network).

5. CSL Accessory Task

The operations performed by the main task of the sample application can be separated in two parts:

- Initialization of the application
- Main task loop.

5.1 Application Initialization

Initialization of Bluetooth LE device as peripheral:

```
ble_peripheral_start();
```

The CSL accessory task is registered to Bluetooth LE framework to receive Bluetooth LE event notifications:

```
ble_register_app();
```

Device name is read from NVM storage and set as GAP device name and local name for scan response.

```
read_name(MAX_NAME_LEN, name_buf);
ble_gap_device_name_set(name_buf, ATT_PERM_READ);
```

Bluetooth Device address is set to private random with address renewal duration of 1024 seconds. `FP_CONFIG_OPTION` is set to `FP_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`, and `AFMN_CONFIG_OPTION` is set to `AFMN_CONFIG_OPTION_BLE_DB_CONTROLLED_BY_APP`, thus preventing Bluetooth LE attribute database to be created (it is reset when calling `ble_gap_device_name_set()`) internally by the finding network frameworks, during initialization.

```
ble_gap_address_set(&addr, 1024);
```

Initialization of state and trigger functionality for buttons is used to control application state:

```
user_button_init();
```

Finding network frameworks are initialized. Their initialization and control are abstracted with a common API, namely **fn_control.h**, easing their integration in the **csl_accessory** sample application project. Based on which finding network framework is initialized and which finding network the device is provisioned for, corresponding finding network -specific API functions are finally called by the **fn_control.h** API functions.

```
fn_control_init(&fn_control_cfg);
```

The configuration structure, which is passed (`fn_control_cfg`) as argument, contains callback functions that are called during the execution of the finding network operations for indicating their status to the application. More specifically:

- `accessory_state_cb` – is called when the state of the accessory is updated, that is, provisioned or removed from a finding network.
- `accessory_pair_stop_cb` – is called when pairing mode is stopped (applicable for Google Fast Pair Service, when pairing mode timeout expires).
- `accessory_db_reset_cb` – is called when GATT attribute database is reset, either because Bluetooth LE services are initialized, or the device has been un-provisioned from a finding network, or Apple Find My™ network framework has triggered database reset.
- `accessory_batt_level_get_cb` – is called when battery level is requested upon initialization of Google Fast Pair Service / Find My Device Network framework.

Advertising is mainly controlled by the finding network frameworks and the abstraction software module, **fn_control.c**. What type of advertising will be started is transparent to the application. Depending on the accessory state, the advertising payload can correspond either to Apple Find My™ network pairing, nearby, and separated states, or Google Fast Pair Service/Find My Device Network pairing and provisioning states. The application only sets Bluetooth LE scan response data to the frameworks, including local name and Software Update Over-The-Air (SUOTA) service, so that they can be maintained across all advertising payload updates performed by the frameworks:

```
fn_control_set_scan_response(ARRAY_LENGTH(scan_rsp_data), scan_rsp_data);
```

In addition to the function calls above, which are required mainly for initiating the support of Apple Find My™ network and Google Fast Pair Service/Find My Device Network frameworks in the **csl_accessory** sample application project, there is also initialization of special modules used for abstracting application functionality related to:

- Advertising control (**adv_control.h/c**) – stop advertising or set advertising to normal or low-power mode. Compared to the corresponding software module used in **fast_pair_device** sample project presented in Ref. [6], now **adv_control.h** API allows for the creation of multiple concurrent Bluetooth LE advertising events with different payloads, employing the standard/primary advertising Bluetooth LE channels, To enable that functionality, `ADV_CONTROL_MULT_EVENTS_ENABLE` is set to 1 in the application's configuration file **accessory_config.h**. API functions in **adv_control.h** are called by the poring API functions of Apple Find My™ network (**afmn_ble.h**) and Google Fast Pair Service/Find My Device Network (**fp_ble.h**) frameworks,
- NVM storage control for application parameters (**app_params.h/c**) – write or read application parameters from NVM storage.
- Battery level monitoring (**battery_monitor.h/c**) – periodically check battery voltage level.
- LED control (**led_control.h/c**) – indicate current status of Apple Find My™ network or Google Fast Pair/Find My Device Network processes, simulate sound maker.
- Motion detector control (**motion_detector.h/c**) – initialize/de-initialize and enable/disable motion detector, create and provide instances to consumer software modules for sharing motion detector control.
- Sound maker control (**sound_maker.h/c**) – initialize/de-initialize and enable/disable sound maker, create and provide instances to consumer software modules for sharing sound maker control.

Finally, there are OS timers created to facilitate the following application activities:

- `user_long_press_tim` – timer used for handling user button long press (start pairing mode).
- `user_short_press_tim` – timer used for handling user button short press (for example, disable sound maker or enter user consent mode).
- `user_double_press_tim` – timer used for checking if user has double pressed the button to simulate the motion.
- `factory_reset_press_tim` – timer used for checking whether user is long-pressing the button to perform a factory reset.
- `adv_stop_press_tim` – timer used for checking if user is pressing the button to stop advertise.

5.2 Application Main Task Loop

Application main OS task (`accessory_task`), as every other OS task, notifies the watchdog and suspends it, while waiting for a task notification. When a notification is received, it resumes the watchdog function and handles the notification accordingly. The handling of notifications is divided according to the notification messages. More specifically, there are the following notification types:

- `BLE_APP_NOTIFY_MASK`: This notification type handles all Bluetooth LE messages/events including GAP and GATT messages. Most of the GATT messages are directed to the corresponding profile. Some Bluetooth LE manager notifications, however, may need special handling by Apple Find My™ network or Google Fast Pair Service/Find My Device Network frameworks. This is done by `fn_control_handle_event()`, which abstracts corresponding calls to `afmn_handle_event()` and `fp_handle_event()`, already presented in Section 4.4.1 and Section 4.5.1, respectively, as well as the call to `ble_service_handle_event()`, required for directing messages/events to Bluetooth LE services (refer to Section 4.4.3 and Section 4.5.3). Essentially, any messages/events that are handled exclusively by the finding network frameworks, are suppressed within the frameworks. `adv_control_handle_event()` is called before `fn_control_handle_event()`, so that the status of multiple concurrent Bluetooth LE advertising events can be updated before accessed by the finding network frameworks. GAP/GATT messages/events that are finally handled by the sample application task are mainly:
  - `BLE_EVT_GAP_CONNECTED`: TX power level is set for the connection and the device is added to the list of connected devices waiting for connection parameters update (for example, connection interval, supervision timeout).
  - `BLE_EVT_GAP_DISCONNECTED`: The device is removed from the list of connected devices.
- `LED_CONTROL_NOTIF`: Handled in `led_control_process_notif()` for controlling LED blinking.
- `BATTERY_MONITOR_NOTIF`: Handled in `battery_monitor_process_notif()` for getting battery status, and in application task for updating battery and advertising status to Google Fast Pair Service/Find My Device Network framework.
- `UPDATE_CONN_PARAM_NOTIF`: Generated by OS timer to update connection parameters after each device connection.
- `LONG_PRESS_TMO_NOTIF`: Received when the user button is long pressed, to start pairing mode.
- `SHORT_PRESS_TMO_NOTIF`: Received when the button is short pressed, in order to disable sound maker or enter user consent mode.
- `DOUBLE_PRESS_TMO_NOTIF`: Received when the button is double pressed, to simulate motion detection.
- `FACTORY_RESET_TMO_NOTIF`: Received when the user button is very long pressed, to start factory reset.
- `ADV_STOP_PRESS_TMO_NOTIF`: Received when the button is long pressed, to stop advertising.
- Set of notifications handled in `fn_control_process_notif()` when either of the following finding network operation -related notifications are received:
  - `APPLE_FMN_NOTIF`: Received when the Apple Find My™ network framework requests execution and `afmn_execution()` must be called.
  - `AFMN_OS_TIMER_EXECUTION_NOTIF`: Received when OS porting module for Apple Find My™ network framework related to OS timer implementation requests application to call `afmn_os_timer_execution()`.

- SERIAL_NUMBER_TMO_NOTIF: Received when serial number lookup timeout expires, to stop serial number lookup consent (applicable for Apple Find My™ network).
- BEACON_TIME_TMO_NOTIF: Generated by OS timer to store the beacon time in NVM storage as indicated by Find My Device Network specification.
- GOOGLE_FAST_PAIR_NOTIF: Received when the Google Fast Pair Service/Find My Device Network framework requests execution and fp_execution() must be called.
- FINDER_NETWORK_STATE_NOTIF: Received when accessory state is changed, that is, provisioned or removed from a finding network, to allow for reinitializing a finding network framework, accordingly.

- CONSOLE_NOTIF: Handled in console_process_notif() when a new character is received over UART for CLI commands.

Regarding the interaction with the finding network frameworks, the main application task controls their operations mainly by calling the top-level API functions listed in **utils\fn_control.h** header file, which subsequently results in calls to corresponding finding network API functions (in **afmn.h** and **fast_pair.h**) depending on the accessory current state. To receive notifications by the framework about the current status of the underlying finding networks functionality, it also registers callback functions as arguments to fn_control_init(). See Section 5.1.

The rest of CSL accessory application's top-level API in **utils\fn_control.h** is summarized:

- fn_control_set_scan_response(): sets custom scan response for the application (i.e. local device name and SUOTA service UUID).
- fn_control_get_finder_network_state(): returns the current state of the accessory.
- fn_control_handle_event(): handles Bluetooth LE events for the finding network frameworks.
- fn_control_stop_ringing(): stops ringing (disables sound maker), called when button is single pressed by user.
- fn_control_factory_reset(): cleans pairing data stored in NVM storage, called when button is very-long pressed (8 s) by user.
- fn_control_set_pairing_mode(): sets the device pairing mode, called when button is long pressed (4 s) by user.
- fn_control_is_pairing_mode(): returns true if pairing process is in progress.
- fn_control_set_user_consent(): sets user consent mode as required by the finding networks to allow for controlling access to specific information, called when button is single pressed by user.

Revision History

Revision	Date	Description
Draft	Feb 14, 2025	Draft version

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Renesas Electronics' suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

Important Notice and Disclaimer

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

© 2025 Renesas Electronics Corporation. All rights reserved.

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
https://www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation.
All trademarks and registered trademarks are the property of their respective owners.