AN12775

Integrating the OTAP Client Service into a Bluetooth LE Peripheral Device

Rev. 0 — 11 March 2020 Application Note

Contents

1 Introduction

Over The Air Programming (OTAP) NXPs custom Bluetooth LE service provides the developer a solution to upgrade the software that the MCU contains. It removes the need of cables and a physical link between the OTAP client (the device that is reprogrammed) and the OTAP server (the device that contains the software update).

The best way to take advantage of the OTAP service is by integrating it into the Bluetooth LE application, that way, you can reprogram the device many times as required.

This document is intended for developers that are familiarized with the OTAP software.

2 Basics of the OTAP Client Software

Chapter 2.1 contains a description of the actual implementation of the OTAP client software included in the SDK package for FRDM-KW36. Chapter 2.2 explains the importance of integrating OTAP client software into your application, and what it is expected to achieve.

2.1 OTAP Memory Management During the Update Process

1. The KW36 Flash is partitioned into:
    - One 256 KB Program Flash array (P-Flash) divided into 2 KB sectors with a flash address range from 0x0000_0000 to 0x0003_FFFF.
    - One 256 KB FlexNVM array divided in 2 KB sectors with address range from 0x1000_0000 to 0x1003_FFFF.
    - Alias memory with address range from 0x0004_0000 to 0x0007_FFFF. Writes or reads at the Alias range address modifies or returns the FlexNVM content, respectively.

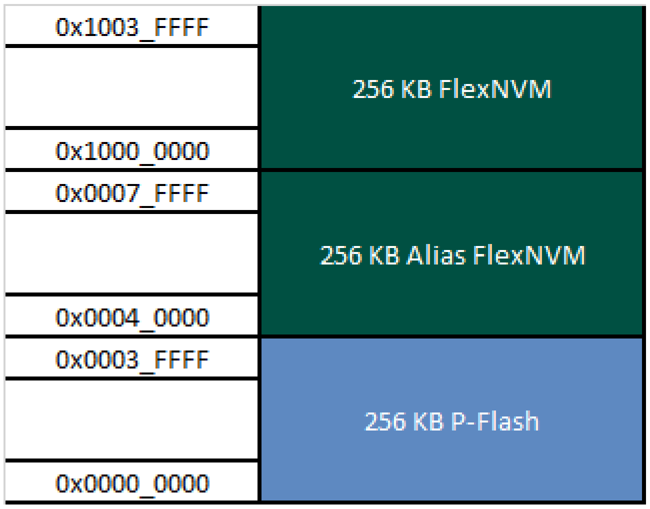


Figure 1. MCU On-Chip memory

2. The OTAP application splits the flash into two independent parts, the OTAP bootloader, and the OTAP client. The OTAP bootloader verifies if there is a new image available in the OTAP client to reprogram the device. The OTAP client software provides the Bluetooth LE custom service needed to communicate the OTAP client device with the OTAP server that contains the new image file. Therefore, the OTAP client device needs to be programmed twice, first with the OTAP bootloader, then with the Bluetooth LE application supporting OTAP client. The mechanism created to have two different software coexisting in the same device is storing each one in different memory regions. This is implemented by the linker file. In the KW36 device, the bootloader application has reserved an 8 KB slot of memory from 0x0000_0000 to 0x0000_1FFF, thus the left memory is reserved, among other things, by the OTAP client application.

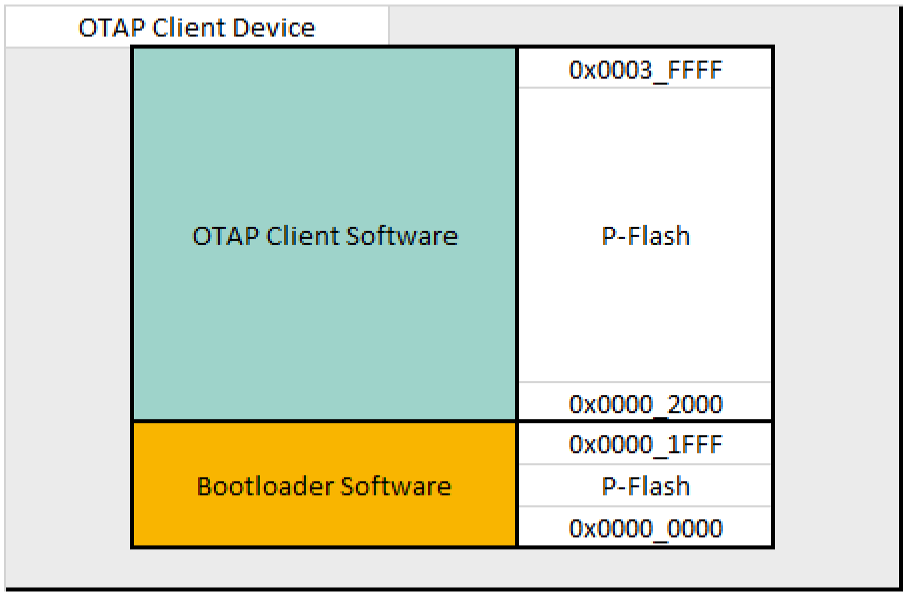


Figure 2. OTAP Client Software

3. To create a new image file for the client device, the developer needs to specify that the code will be stored with an offset of 8 KB since the first addresses must be reserved for the bootloader (making use of the linker script). The new application should also contain the Bootloader Flags at the corresponding address to work properly.

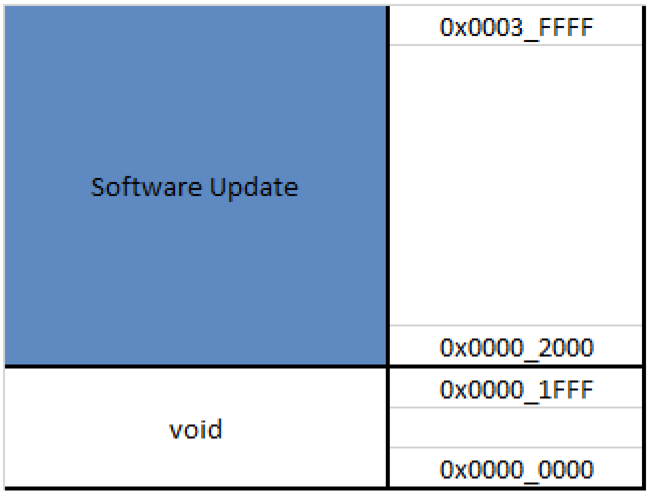


Figure 3. Software update

4. At the connection state, the OTAP server sends the image packets (known as chunks) to the OTAP client via Bluetooth LE. The OTAP client device can store these chunks, in the first instance, at the external SPI flash (only available on the FRDM-KW36 board) or at the On-Chip FlexNVM memory. The destination of the code is selectable in the OTAP client software.

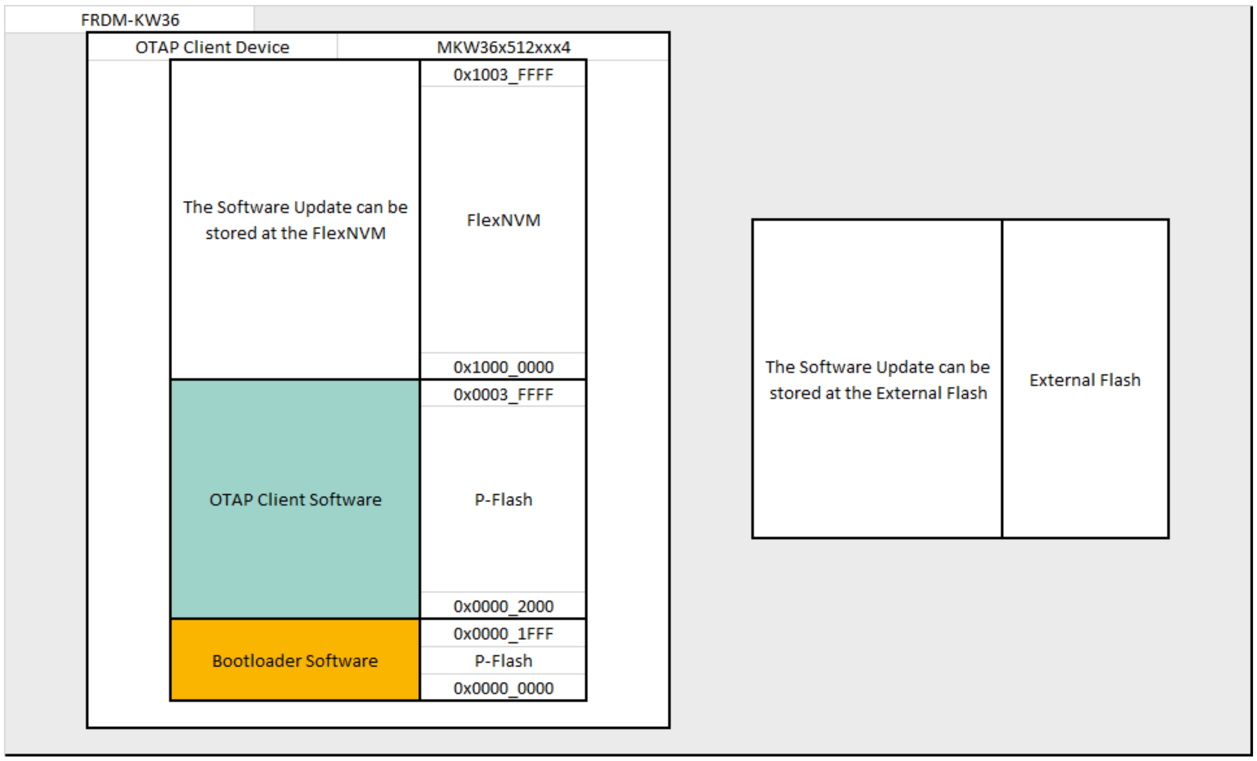


Figure 4. Storage of the software update

5. When the transfer of the image has finished, and all chunks were sent from the OTAP server to the OTAP client, the OTAP client software writes information, such as the source of the image update (external flash or FlexNVM) in a portion of memory known as Bootloader Flags, and then resets the MCU to execute the OTAP bootloader code. The OTAP bootloader reads the Bootloader Flags to get the information needed to program the device and triggers a command to reprogram the MCU with the new application. Due to the new application was built with an offset of 8 KB,

the OTAP bootloader programs the device starting from the 0x0000_2000 address and the OTAP client application is overwritten by the new image. Then, the OTAP bootloader triggers a command to start the execution of the new image. If the new image does not contain the OTAP service included, the device would not be able to be programmed again due to the lack of OTAP functionality. This is discussed further in Section 2.2 Advantages of the OTAP Service Integration.

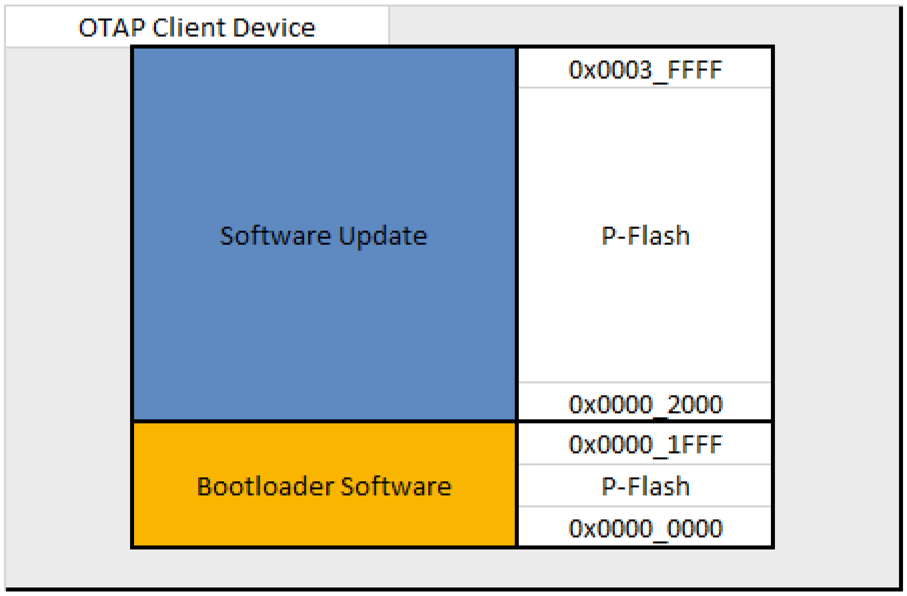


Figure 5. Memory content at the end of the software update process

NOTE

In practice, the boundary created between the OTAP client software and the software update addresses when the internal storage is enabled is not placed exactly in the boundary of the P-Flash and FlexNVM memory regions. Even more, these values might change depending on your linker settings. You can inspect the effective memory addresses in your project.

2.2 Advantages of the OTAP Service Integration

As explained in chapter 2.1 OTAP Memory Management During the Update Process. The OTAP client software is a single-programming demo application. Suppose that an OTAP client device is programmed with the OTAP client software and this device requests an update, for example, a Heart Rate Sensor (HRS). The image that the OTAP server sends to the OTAP client must be the HRS. After the reprogramming process the device that was the OTAP client, now, has turned into a Heart Rate Sensor. The HRS does not have the capabilities to communicate with the OTAP server and request for another update. But if the HRS image had included the OTAP client service as well, the device would have the possibility to request another software update, for example, a modified Glucose Sensor example with OTAP Service. Due to the Glucose Sensor software includes the OTAP client, the device can request another software update from the OTAP server. That way, the developer can continue upgrading the software many times as needed. In other words, to be able to upgrade the software on the OTAP client device in the future, the application sent over the air should support OTAP service.

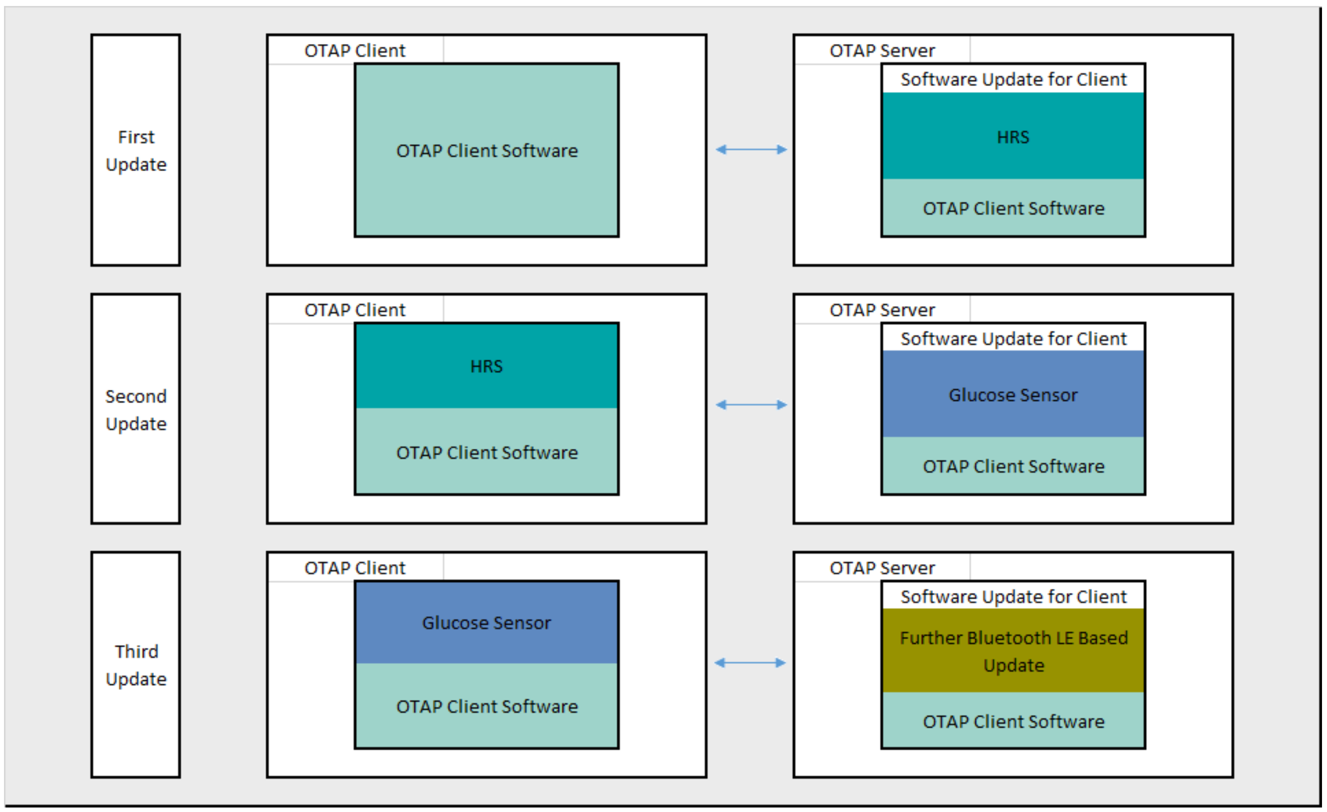


Figure 6. OTAP integration functionality example

This application note is intended as guidance to add the OTAP service to a Bluetooth LE application.

3 Prerequisites

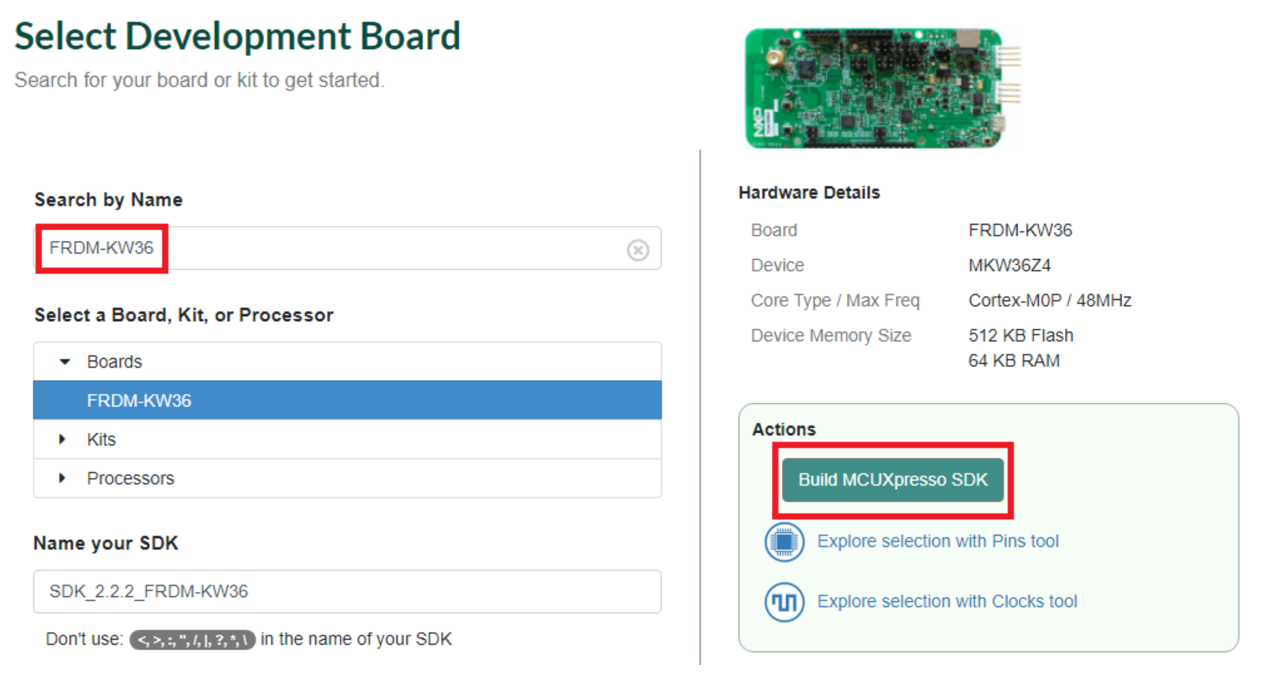
This document is provided together with a functional demo of the OTAP service integration. The example was based on the Heart Rate Sensor project, available in the FRDM-KW36 SDK package and developed on the MCUXpresso IDE platform. The following are required to complete the implementation of the HRS-OTAP integration demo.

- MCUXpresso IDE v11.0.0 or later
- FRDM-KW36 SDK
- HRS – OTAP demo package
- FRDM-KW36 board
- A smartphone with IoT Toolbox NXP app (available for Android and iOS)

3.1 Software Development Kit Download and Install

This chapter provides all the steps needed to download the SDK (Software Development Kit) for the FRDM-KW36 used as a starting point.

1. Navigate to the MCUXpresso web page
2. Click on “Select Development Board”. Log in with your registered account.
3. Search for the “FRDM-KW36” board in the “Search by Name” textbox. Then click on the suggested board and click on “Build MCUXpresso SDK”.



Select Development Board

Search for your board or kit to get started.

Search by Name

FRDM-KW36

Select a Board, Kit, or Processor

- Boards
  - FRDM-KW36
- Kits
- Processors

Name your SDK

SDK_2.2.2_FRDM-KW36

Don't use: < >, :, ", /, |, ?, *, \ in the name of your SDK

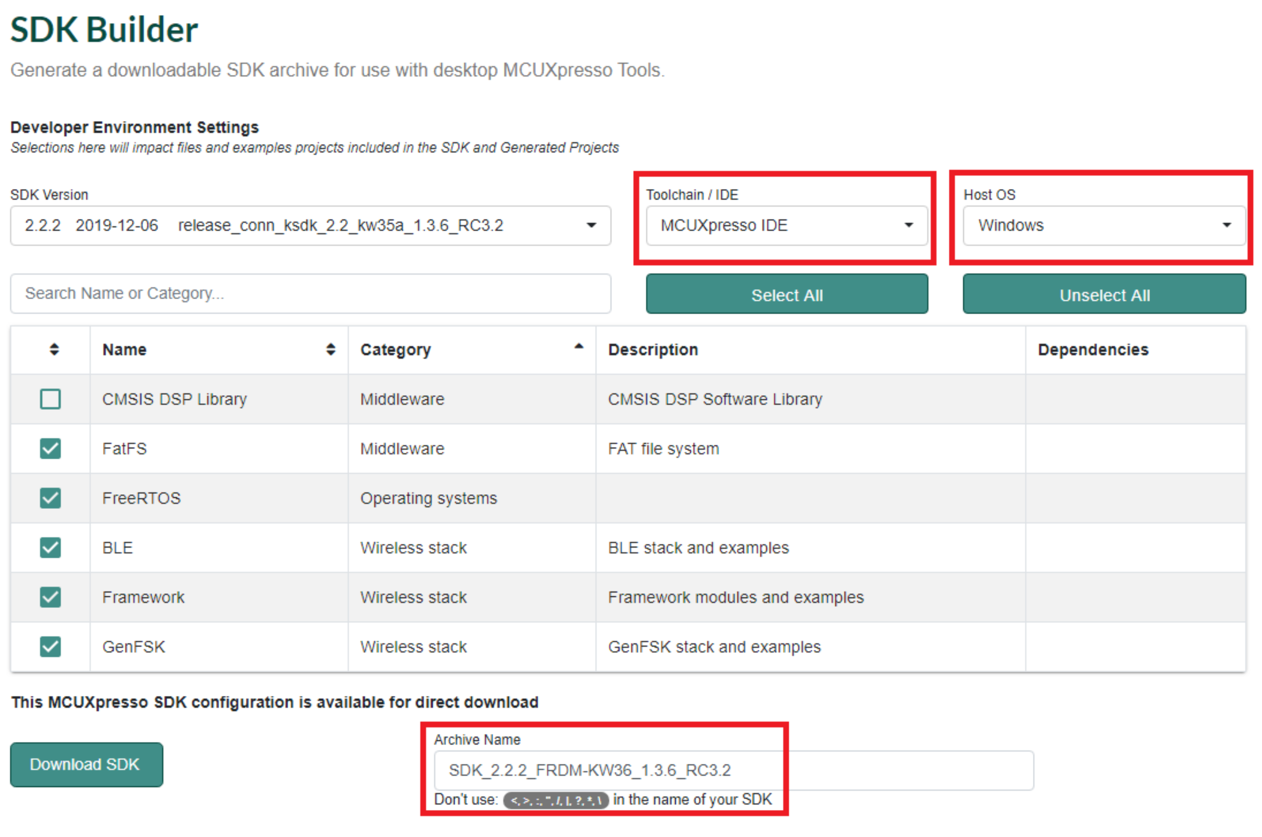
Hardware Details

Board	FRDM-KW36
Device	MKW36Z4
Core Type / Max Freq	Cortex-M0P / 48MHz
Device Memory Size	512 KB Flash 64 KB RAM

Actions
Build MCUXpresso SDK
Explore selection with Pins tool
Explore selection with Clocks tool

Figure 7. Building the FRDM-KW36 SDK package

4. Select MCUXpresso IDE in the “Toolchain / IDE” combo box. Select the supported OS and provide the name to identify the package in your MCUXpresso Dashboard.



SDK Builder

Generate a downloadable SDK archive for use with desktop MCUXpresso Tools.

Developer Environment Settings

Selections here will impact files and examples projects included in the SDK and Generated Projects

SDK Version: 2.2.2 2019-12-06 release_conn_ksdk_2.2_kw35a_1.3.6_RC3.2

Toolchain / IDE: MCUXpresso IDE

Host OS: Windows

	Name	Category	Description	Dependencies
☐	CMSIS DSP Library	Middleware	CMSIS DSP Software Library	
✓	FatFS	Middleware	FAT file system	
✓	FreeRTOS	Operating systems		
✓	BLE	Wireless stack	BLE stack and examples	
✓	Framework	Wireless stack	Framework modules and examples	
✓	GenFSK	Wireless stack	GenFSK stack and examples	

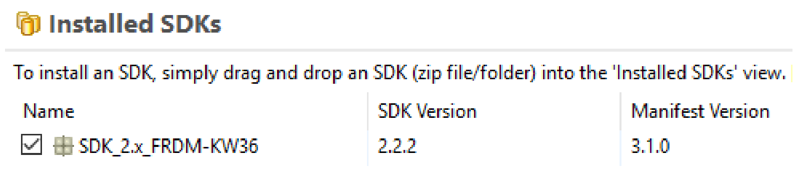
This MCUXpresso SDK configuration is available for direct download

Archive Name: SDK_2.2.2_FRDM-KW36_1.3.6_RC3.2

Don't use: < >, :, ", /, |, ?, *, \ in the name of your SDK

Figure 8. Customizing the installation settings

5. Click the “Download SDK” button. It takes a few minutes until the system gets the package into your account on the MCUXpresso webpage. Read and accept the license agreement. The SDK automatically downloads on your PC.
6. Open MCUXpresso IDE. Drag and drop the FRDM-KW36 SDK zip in the “Installed SDKs” perspective.



To install an SDK, simply drag and drop an SDK (zip file/folder) into the 'Installed SDKs' view.

Name	SDK Version	Manifest Version
✓ SDK_2.x_FRDM-KW36	2.2.2	3.1.0

Figure 9. Importing SDK package to MCUXpresso IDE

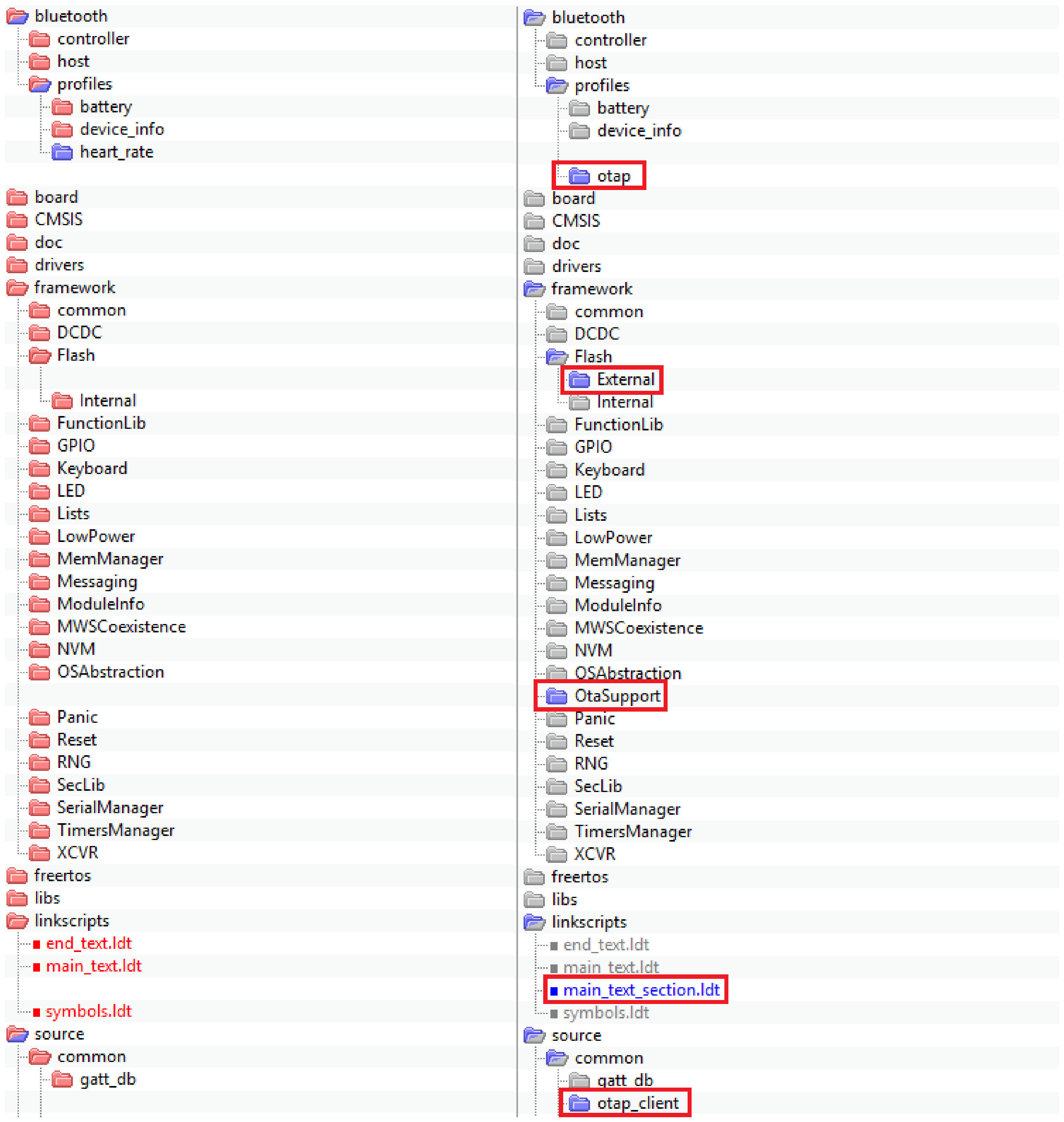
At this point, you have downloaded and installed the SDK package for the FRDM-KW36 board.

4 Customizing a Based Bluetooth LE Demo to Integrate the OTAP Service

The following steps describe the process of customizing a Bluetooth LE demo imported from the SDK to integrate the OTAP service. This guide uses a Heart Rate Sensor project (HRS) as a starting point, so, some steps may differ for another Bluetooth LE SDK example.

4.1 Importing the OTAP Service and Framework Services into the HRS

The OTAP client software makes use of Framework functionalities that are not included for the HRS demo. So, the first step for the OTAP integration must be to compare which folders and files in the project source tree are different between your project and the OTAP Client. Then you must include it to enable these functionalities. A comparison between the HRS (left) and the OTAP Client (right) is shown in Figure 10.



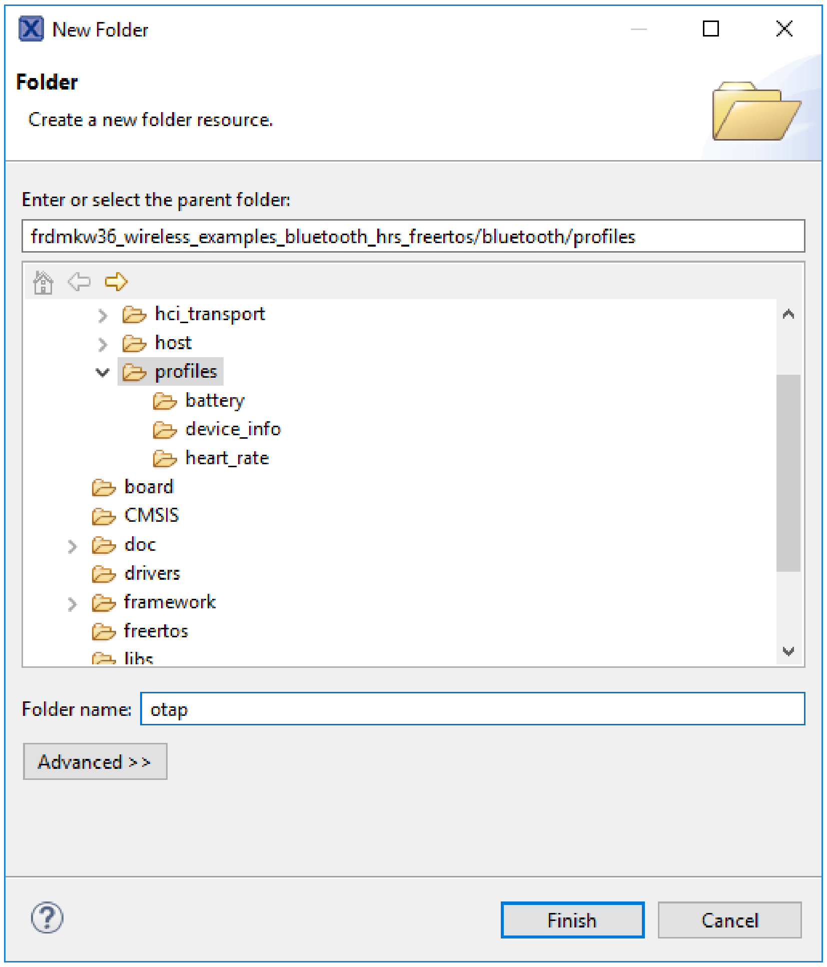
```
bluetooth
    controller
    host
    profiles
        battery
        device_info
        heart_rate

board
CMSIS
doc
drivers
framework
    common
    DCDC
    Flash

        Internal
    FunctionLib
    GPIO
    Keyboard
    LED
    Lists
    LowPower
    MemManager
    Messaging
    ModuleInfo
    MWSCoexistence
    NVM
    OSAbstraction

    Panic
    Reset
    RNG
    SecLib
    SerialManager
    TimersManager
    XCVR
freertos
libs
linkscripts
    end_text.ldt
    main_text.ldt

    symbols.ldt
source
    common
        gatt_db
```

```
bluetooth
    controller
    host
    profiles
        battery
        device_info

        otap
board
CMSIS
doc
drivers
framework
    common
    DCDC
    Flash
        External
        Internal
    FunctionLib
    GPIO
    Keyboard
    LED
    Lists
    LowPower
    MemManager
    Messaging
    ModuleInfo
    MWSCoexistence
    NVM
    OSAbstraction
    OtaSupport
    Panic
    Reset
    RNG
    SecLib
    SerialManager
    TimersManager
    XCVR
freertos
libs
linkscripts
    end_text.ldt
    main_text.ldt
    main_text_section.ldt
    symbols.ldt
source
    common
        gatt_db
        otap_client
```

Figure 10. Source tree comparison

The folders and files that are in OTAP Client but not in HRS, must be imported in the HRS project. The following steps are to include the folders and source files in your project.

1. Expand the “bluetooth” and the “framework” folders in your workspace. Select the folder needed for updates and click the right mouse button. Select “New->Folder”. A new window is shown. Provide the same name as the missing folder in the source directory.

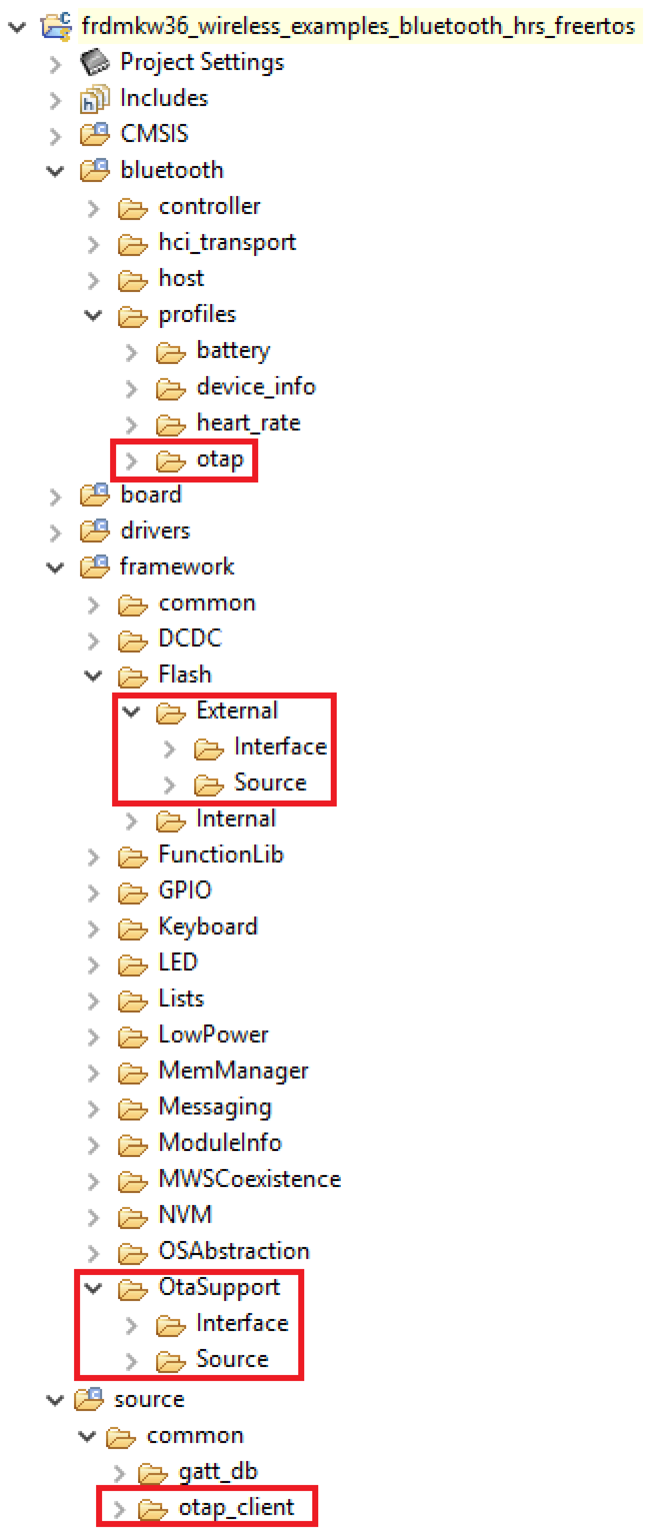

New Folder
Folder
Create a new folder resource.
Enter or select the parent folder:

frdmkw36_wireless_examples_bluetooth_hrs_freertos/bluetooth/profiles

Folder name: otap

Figure 11. Creating the Bluetooth and Framework folders

2. Repeat step 1 for the left folders. The result must look similar as shown in Figure 12.



```
frdmkw36_wireless_examples_bluetooth_hrs_freertos
    Project Settings
    Includes
    CMSIS
    bluetooth
        controller
        hci_transport
        host
        profiles
            battery
            device_info
            heart_rate
            otap
    board
    drivers
    framework
        common
        DCDC
        Flash
            External
                Interface
                Source
            Internal
        FunctionLib
        GPIO
        Keyboard
        LED
        Lists
        LowPower
        MemManager
        Messaging
        ModuleInfo
        MWSCoexistence
        NVM
        OSAbstraction
        OtaSupport
            Interface
            Source
    source
        common
            gatt_db
            otap_client
```

Figure 12. HRS directory updated

3. Copy the files inside all the recently created folders, from the OTAP client and save it into your project. Ensure that all the files are in the same folder from the HRS side. For this example, these files are listed below.
    - “otap_interface.h” and “otap_service.c” in “bluetooth->profiles->otap” folder.

- “Eeprom.h” in “framework->Flash->External->Interface” folder.
- Eeprom source files in “framework->Flash->External->Source” folder.
- “OtaSupport.h” in “framework->OtaSupport->Interface” folder.
- “OtaSupport.c” in “framework->OtaSupport->Source” folder.
- “main_text_section.ldt” in “linkscripts” folder.
- “otap_client.h” and “otap_client.c” in “source->common->otap_client”

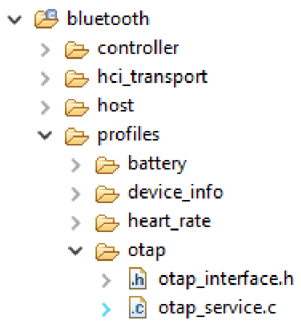
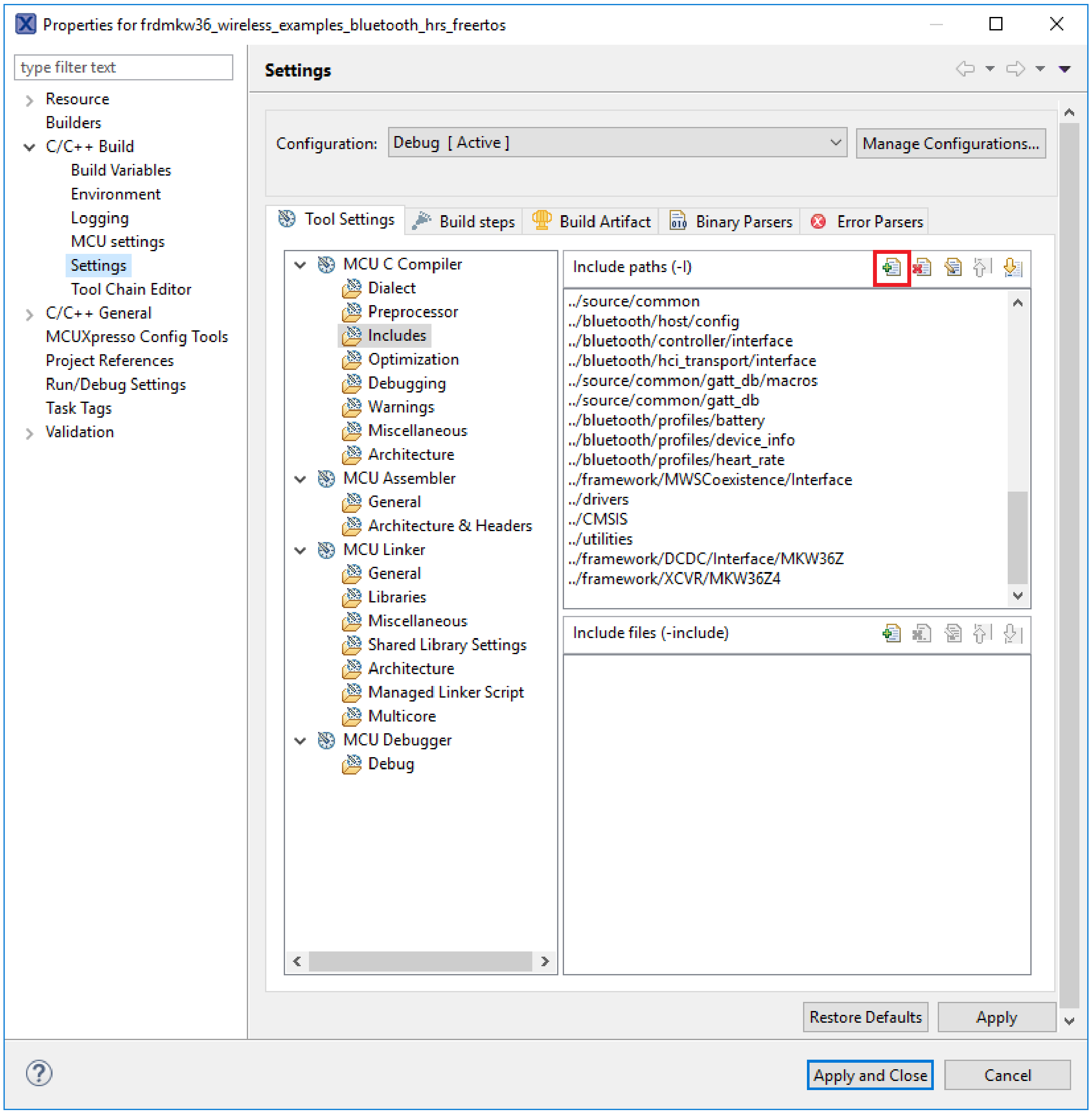


Figure 13. OTAP files integrated into the HRS project

4. Navigate to “Project->Properties” in MCUXpresso IDE. Go to “C/C++ Build->Settings->Tool Settings->MCU C Compiler->Includes”. Click on the icon next to “Include paths” textbox (see Figure 14). A new window will appear, then click on the “Workspace” button.

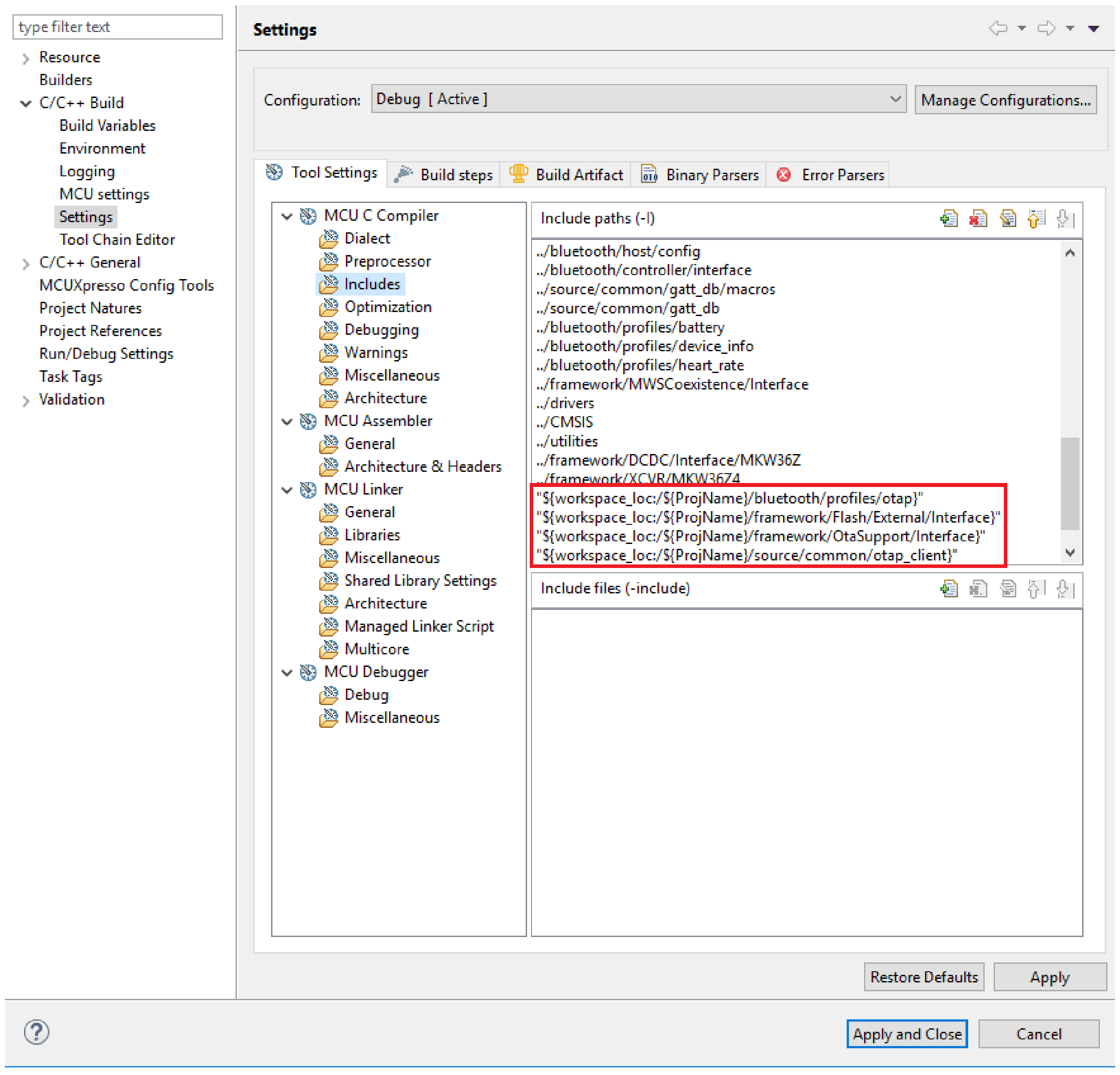


```
../source/common
../bluetooth/host/config
../bluetooth/controller/interface
../bluetooth/hci_transport/interface
../source/common/gatt_db/macros
../source/common/gatt_db
../bluetooth/profiles/battery
../bluetooth/profiles/device_info
../bluetooth/profiles/heart_rate
../framework/MWSCoexistence/Interface
../drivers
../CMSIS
../utilities
../framework/DCDC/Interface/MKW36Z
../framework/XCVR/MKW36Z4
```

Figure 14. Include paths perspective

5. Deploy your directory tree in the folder selection window. Select the following folders and click the “OK” button to save the changes:
   - bluetooth->profiles->otap
   - framework->Flash->External->Interface
   - framework->OtaSupport->Interface
   - source->common->otap_client

   Ensure that these paths were imported onto the “Include paths” view.



Configuration: Debug [Active]

Include paths (-I)

```
../bluetooth/host/config
../bluetooth/controller/interface
../source/common/gatt_db/macros
../source/common/gatt_db
../bluetooth/profiles/battery
../bluetooth/profiles/device_info
../bluetooth/profiles/heart_rate
../framework/MWSCoexistence/Interface
../drivers
../CMSIS
../utilities
../framework/DCDC/Interface/MKW36Z
../framework/XCVR/MKW36Z4
"${workspace_loc:/${ProjName}/bluetooth/profiles/otap}"
"${workspace_loc:/${ProjName}/framework/Flash/External/Interface}"
"${workspace_loc:/${ProjName}/framework/OtaSupport/Interface}"
"${workspace_loc:/${ProjName}/source/common/otap_client}"
```

Figure 15. Including the OTAP folders in the project paths

At this point, you have included the OTAP client Bluetooth and Framework services in the HRS project.

4.2 Main Modifications in the Source Files

Once you have included the OTAP client folders and files in your custom project, the next step is to inspect the differences between the source files of the OTAP client and your Bluetooth LE application and add the code needed to integrate the OTAP Service.The following sections explain the main aspects that you should focus on.

4.2.1 app_preinclude.h

The “app_preinclude.h” file, contains many preprocessor directives that configure some functionalities of the project, such as low power enablement, DCDC configuration, Bluetooth LE security definitions, and the hardware board macros. The OTAP client software requires some definitions that are not included for other Bluetooth LE SDK projects. These definitions that you must include in your software update, are the following:

- gEepromType_d
- gEepromParams_WriteAlignment_c
- gOtapClientAtt_d

The OTAP – HRS demo, sets the following values:

1. gEepromType_d: Defines the storage method between the AT45DB041E external flash on the FRDM-KW36 board (default value) or the FlexNVM on-chip memory. You can also select among other memory devices for custom boards (see the list of suppoted EEPROM in the Eeprom.h header file at framework/Flash/External/Interface).

   ```
   /* Specifies the type of EEPROM available on the target board */
   #define gEepromType_d gEepromDevice_AT45DB041E_c
   ```

2. gEepromParams_WriteAlignment: Defines the offset of the software update for programming. Do not modify the default value.

   ```
   /* Eeprom Write alignment for Bootloader flags. */
   #define gEepromParams_WriteAlignment_c 8
   ```

3. gOtapClientAtt_d: It sets the ATT transference method for OTA updates. It must be set to 1 for own purpose.

   ```
   #define gOtapClientAtt_d 1
   ```

4.2.2 app_config.c

The “app_config.c” source file, contains some structures that configure the advertising and scanning parameters and data. It also contains the access security requirements for each service in the device.

The advertising data announces the list of services that the Bluetooth LE advertiser device (HRS – OTAP) contains. This information is used by the Bluetooth LE scanner, to filter out the advertiser devices that do not contain the services required. Hence, you must include the OTAP client service in the advertising data, to announce to the OTAP server, the availability of this service. This is done at the scan response data as shown in the code below.

```
static const gapAdStructure_t scanResponseStruct[1] = {
{
.length = NumberOfElements(uuid_service_otap) + 1,
.adType = gAdIncomplete128bitServiceList_c,
.aData = (uint8_t *)uuid_service_otap
}
};
gapScanResponseData_t gAppScanRspData =
{
NumberOfElements(scanResponseStruct),
(void *)scanResponseStruct
};
```

CAUTION

Due to the OTAP client service is announced in the scan response, you must ensure that the OTAP server device is configured to perform active scanning. This is already done by the IoT Toolbox App, but the OTAP server SDK example does not. You can change the scanning settings of the OTAP server SDK example at the “app_config.c” file, in the “gScanParams” struct.

Additionally, you need to include the access security requirements for the OTAP service. This is done at the "gapServiceSecurityRequirements_t" struct. You can customize these parameters for your purpose. The HRS – OTAP demo sets the following parameters, focus on the OTAP service handle:

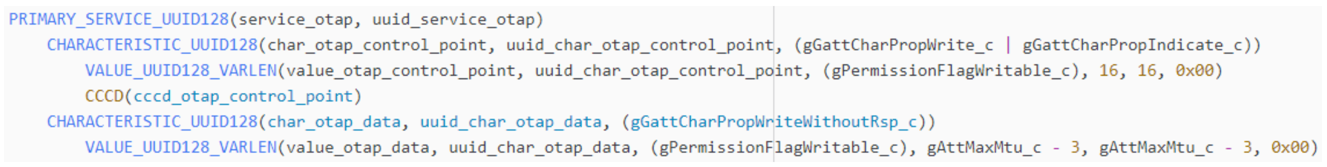
```
static const gapServiceSecurityRequirements_t serviceSecurity[4] = {
{
.requirements = {
.securityModeLevel = gSecurityMode_1_Level_3_c,
.authorization = FALSE,
.minimumEncryptionKeySize = gDefaultEncryptionKeySize_d
},
.serviceHandle = service_heart_rate
},
.requirements = {
.securityModeLevel = gSecurityMode_1_Level_3_c,
.authorization = FALSE,
.minimumEncryptionKeySize = gDefaultEncryptionKeySize_d
},
.serviceHandle = service_otap
},
{
.requirements = {
.securityModeLevel = gSecurityMode_1_Level_3_c,
.authorization = FALSE,
.minimumEncryptionKeySize = gDefaultEncryptionKeySize_d
},
.serviceHandle = service_battery
},
{
.requirements = {
.securityModeLevel = gSecurityMode_1_Level_3_c,
.authorization = FALSE,
.minimumEncryptionKeySize = gDefaultEncryptionKeySize_d
},
.serviceHandle = service_device_info
}
};
```

Last modification requires as well, to increase the index of the number of services in "deviceSecurityRequirements" struct:

```
gapDeviceSecurityRequirements_t deviceSecurityRequirements = {
.pMasterSecurityRequirements = (void*)&masterSecurity,
.cNumServices = 4,
.aServiceSecurityRequirements = (void*)serviceSecurity
};
```

4.2.3 gatt_db.h and gatt_uuid128.h

The "gatt_db.h" header file, contains the list of attributes that, together, shapes the profile of the GATT server (HRS-OTAP client device). The most important step of this guide is to include the list of the OTAP client attributes into the device's database. It is recommended to open the OTAP client SDK example, and your Bluetooth LE demo in order to compare both GATT databases. Figure 16 shows the OTAP client portion of the database that defines the OTAP client service.



```
PRIMARY_SERVICE_UUID128(service_otap, uuid_service_otap)
    CHARACTERISTIC_UUID128(char_otap_control_point, uuid_char_otap_control_point, (gGattCharPropWrite_c | gGattCharPropIndicate_c))
        VALUE_UUID128_VARLEN(value_otap_control_point, uuid_char_otap_control_point, (gPermissionFlagWritable_c), 16, 16, 0x00)
        CCCD(cccd_otap_control_point)
    CHARACTERISTIC_UUID128(char_otap_data, uuid_char_otap_data, (gGattCharPropWriteWithoutRsp_c))
        VALUE_UUID128_VARLEN(value_otap_data, uuid_char_otap_data, (gPermissionFlagWritable_c), gAttMaxMtu_c - 3, gAttMaxMtu_c - 3, 0x00)
```

Figure 16. OTAP client service

The profile built within the "gatt_db.h" database for the HRS – OTAP demo has the architecture depicted in Figure 17.

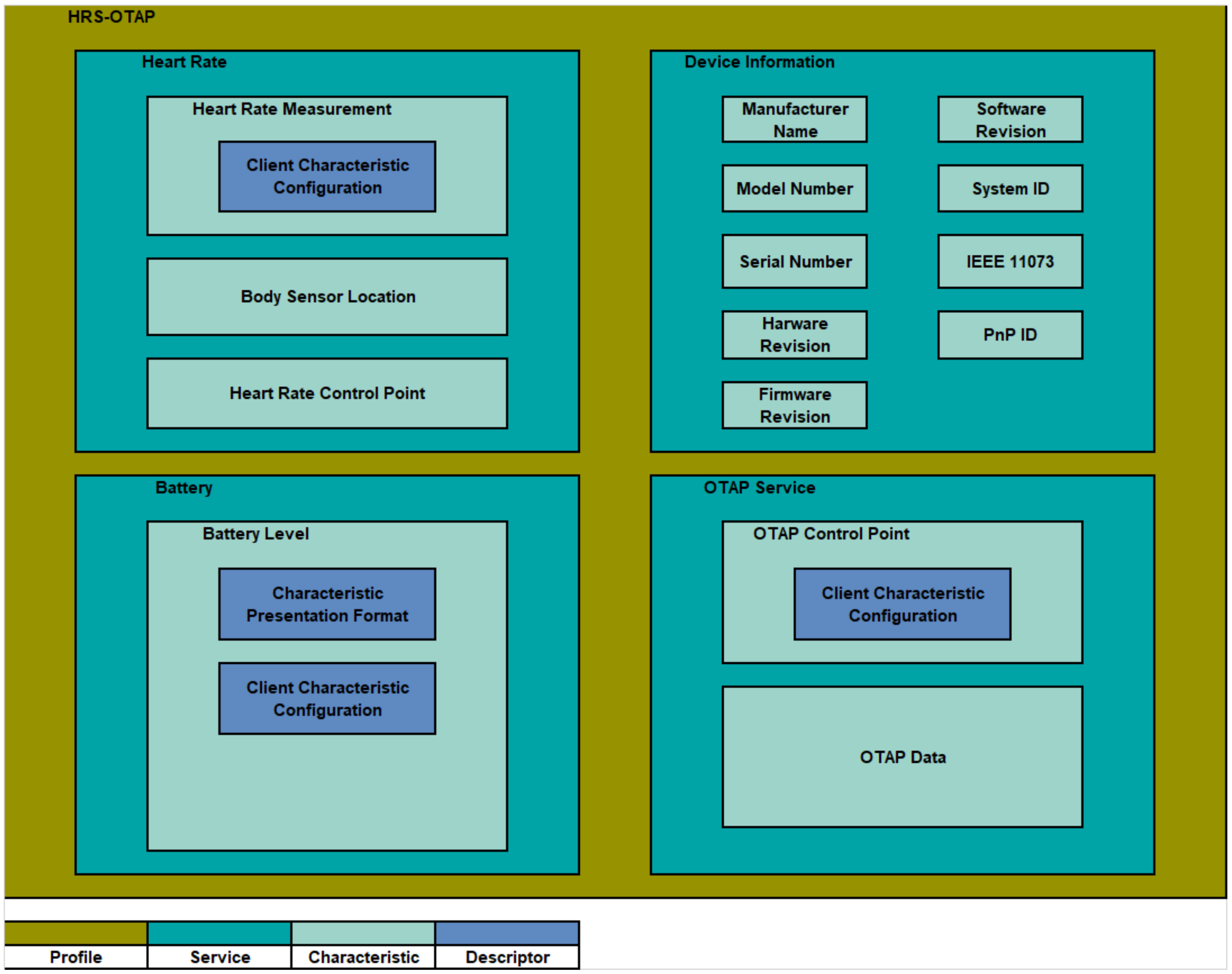
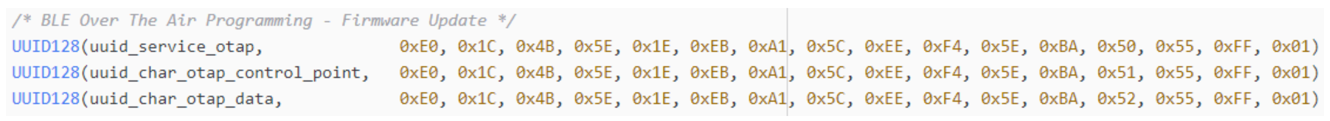


Figure 17. HRS – OTAP profile

The "gatt_uuid128.h" header file contains all the "custom" UUID definitions and its assignation. The "gatt_uuid128.h" does not contain definitions in the original HRS SDK project because the heart rate and the battery services are adopted services by the Bluetooth SIG. However, the OTAP service and its characteristics need to be specified by the developer as a 128 – UUID. Figure 18 shows how to implement the 128 – UUID assignation for the OTAP service.



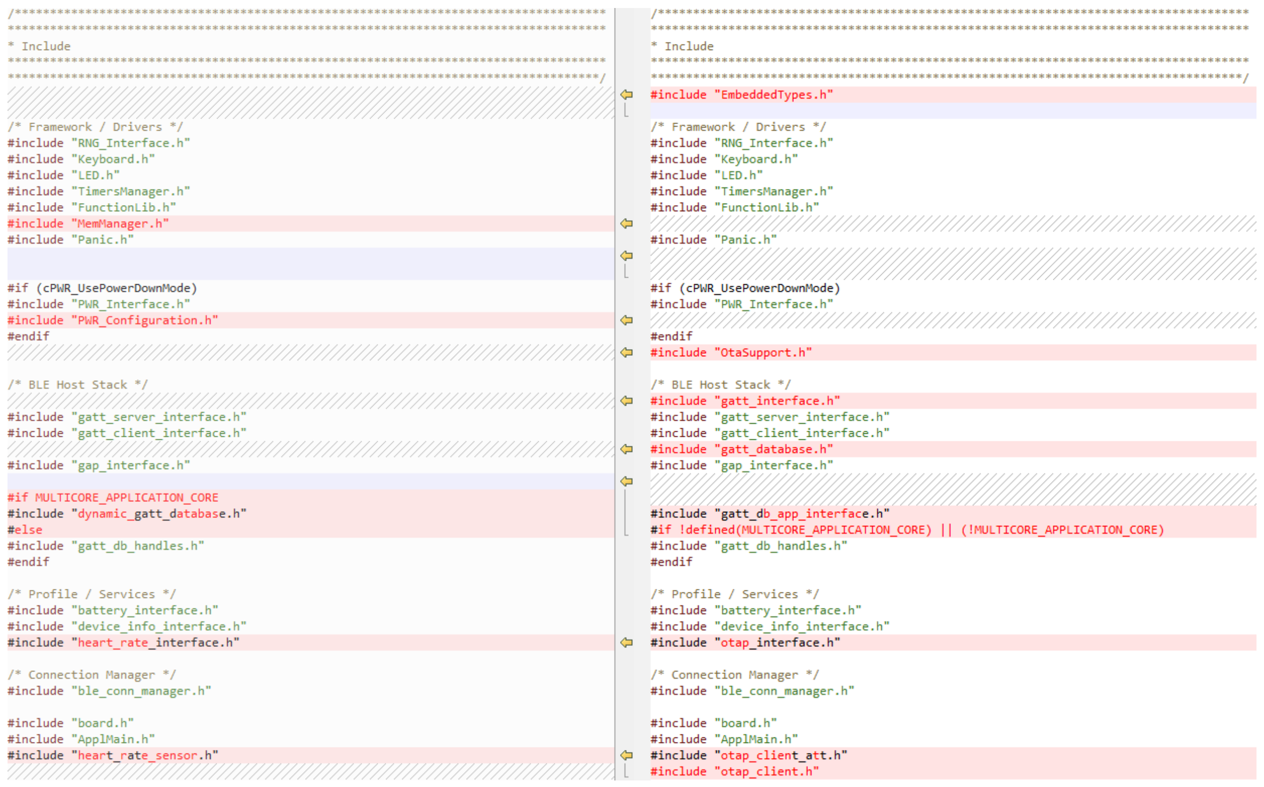
```
/* BLE Over The Air Programming - Firmware Update */
UUID128(uuid_service_otap,              0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA, 0x50, 0x55, 0xFF, 0x01)
UUID128(uuid_char_otap_control_point,   0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA, 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)
```

Figure 18. HRS – OTAP 128 – UUID definitions

4.2.4 heart_rate_sensor.c

The “heart_rate_sensor.c” is the main source file at the application level. Here are managed all the procedures that the device performs, before, during and after to create a connection. The following steps are the main changes to integrate the OTAP service.

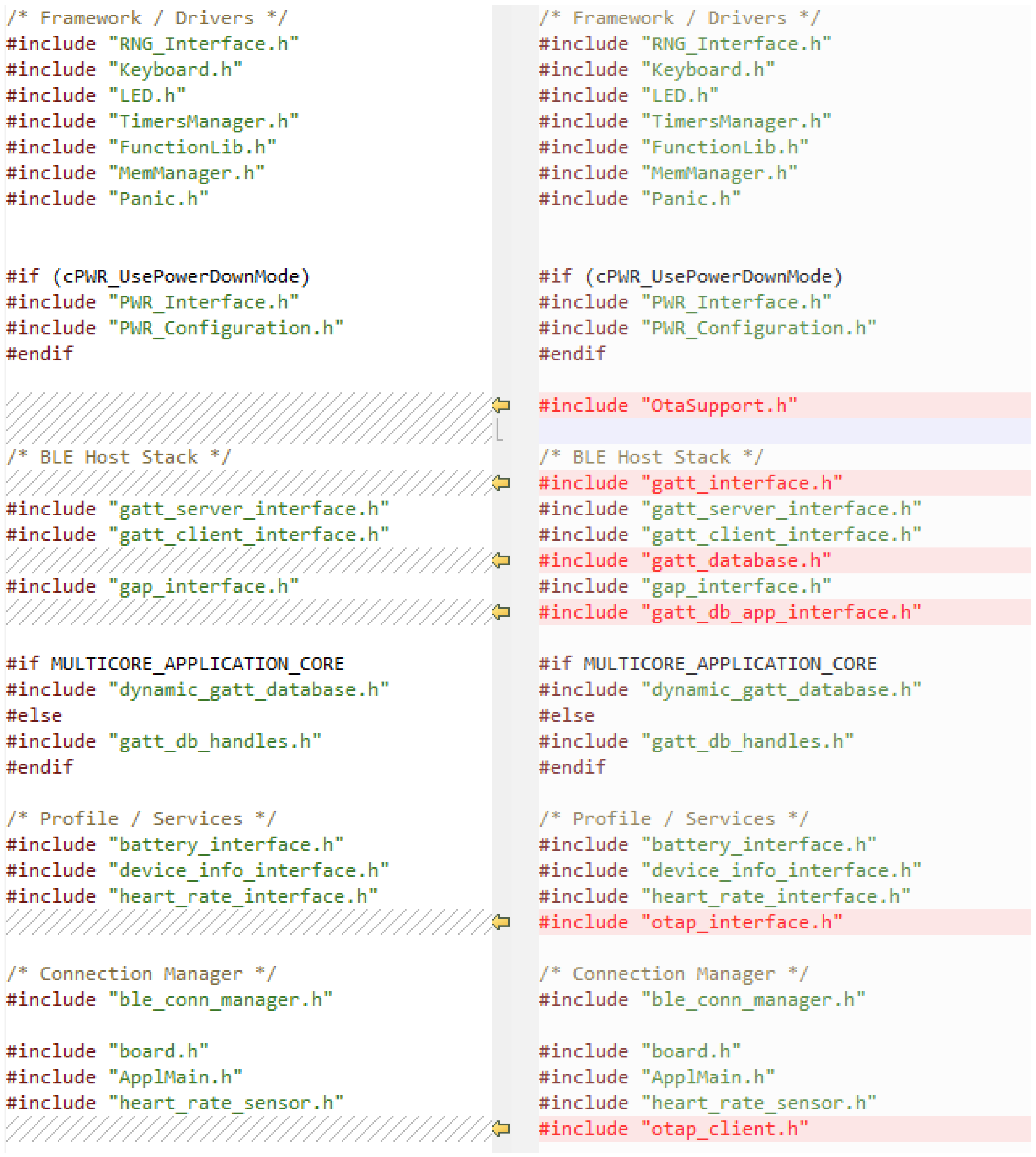
1. Merge the missing “#include” preprocessor directives to reference the OTAP files on your project (except otap_client_att.h). See Figure 19, which is a comparison between HRS (left) and OTAP client (right) application files. This step depends on your software since it might share different files than this example. The results should be similar as depicted in Figure 20, before (HRS left), after (HRS-OTAP right).



```
/************************************************************************************
*************************************************************************************
* Include
*************************************************************************************
************************************************************************************/

/* Framework / Drivers */
#include "RNG_Interface.h"
#include "Keyboard.h"
#include "LED.h"
#include "TimersManager.h"
#include "FunctionLib.h"
#include "MemManager.h"
#include "Panic.h"

#if (cPWR_UsePowerDownMode)
#include "PWR_Interface.h"
#include "PWR_Configuration.h"
#endif

/* BLE Host Stack */
#include "gatt_server_interface.h"
#include "gatt_client_interface.h"
#include "gap_interface.h"

#if MULTICORE_APPLICATION_CORE
#include "dynamic_gatt_database.h"
#else
#include "gatt_db_handles.h"
#endif

/* Profile / Services */
#include "battery_interface.h"
#include "device_info_interface.h"
#include "heart_rate_interface.h"

/* Connection Manager */
#include "ble_conn_manager.h"

#include "board.h"
#include "ApplMain.h"
#include "heart_rate_sensor.h"
```

```
/************************************************************************************
*************************************************************************************
* Include
*************************************************************************************
************************************************************************************/
#include "EmbeddedTypes.h"

/* Framework / Drivers */
#include "RNG_Interface.h"
#include "Keyboard.h"
#include "LED.h"
#include "TimersManager.h"
#include "FunctionLib.h"
#include "Panic.h"

#if (cPWR_UsePowerDownMode)
#include "PWR_Interface.h"
#endif
#include "OtaSupport.h"

/* BLE Host Stack */
#include "gatt_interface.h"
#include "gatt_server_interface.h"
#include "gatt_client_interface.h"
#include "gatt_database.h"
#include "gap_interface.h"

#include "gatt_db_app_interface.h"
#if !defined(MULTICORE_APPLICATION_CORE) || (!MULTICORE_APPLICATION_CORE)
#include "gatt_db_handles.h"
#endif

/* Profile / Services */
#include "battery_interface.h"
#include "device_info_interface.h"
#include "otap_interface.h"

/* Connection Manager */
#include "ble_conn_manager.h"

#include "board.h"
#include "ApplMain.h"
#include "otap_client_att.h"
#include "otap_client.h"
```

Figure 19. Comparison between HRS (left) and OTAP (right) includes



Left (HRS):

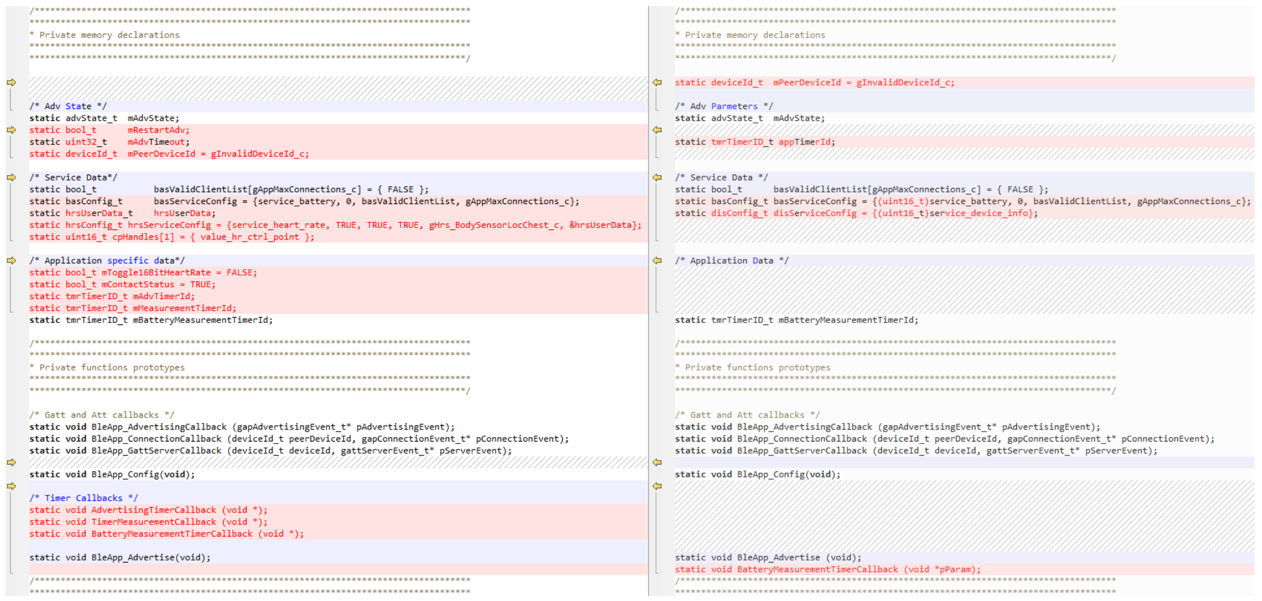
```
/* Framework / Drivers */
#include "RNG_Interface.h"
#include "Keyboard.h"
#include "LED.h"
#include "TimersManager.h"
#include "FunctionLib.h"
#include "MemManager.h"
#include "Panic.h"


#if (cPWR_UsePowerDownMode)
#include "PWR_Interface.h"
#include "PWR_Configuration.h"
#endif


/* BLE Host Stack */

#include "gatt_server_interface.h"
#include "gatt_client_interface.h"

#include "gap_interface.h"


#if MULTICORE_APPLICATION_CORE
#include "dynamic_gatt_database.h"
#else
#include "gatt_db_handles.h"
#endif

/* Profile / Services */
#include "battery_interface.h"
#include "device_info_interface.h"
#include "heart_rate_interface.h"


/* Connection Manager */
#include "ble_conn_manager.h"

#include "board.h"
#include "ApplMain.h"
#include "heart_rate_sensor.h"
```

Right (HRS-OTAP):

```
/* Framework / Drivers */
#include "RNG_Interface.h"
#include "Keyboard.h"
#include "LED.h"
#include "TimersManager.h"
#include "FunctionLib.h"
#include "MemManager.h"
#include "Panic.h"


#if (cPWR_UsePowerDownMode)
#include "PWR_Interface.h"
#include "PWR_Configuration.h"
#endif

#include "OtaSupport.h"

/* BLE Host Stack */
#include "gatt_interface.h"
#include "gatt_server_interface.h"
#include "gatt_client_interface.h"
#include "gatt_database.h"
#include "gap_interface.h"
#include "gatt_db_app_interface.h"

#if MULTICORE_APPLICATION_CORE
#include "dynamic_gatt_database.h"
#else
#include "gatt_db_handles.h"
#endif

/* Profile / Services */
#include "battery_interface.h"
#include "device_info_interface.h"
#include "heart_rate_interface.h"
#include "otap_interface.h"

/* Connection Manager */
#include "ble_conn_manager.h"

#include "board.h"
#include "ApplMain.h"
#include "heart_rate_sensor.h"
#include "otap_client.h"
```

Figure 20. Merging the OTAP files into the project. Before (HRS left) and after (HRS-OTAP right).

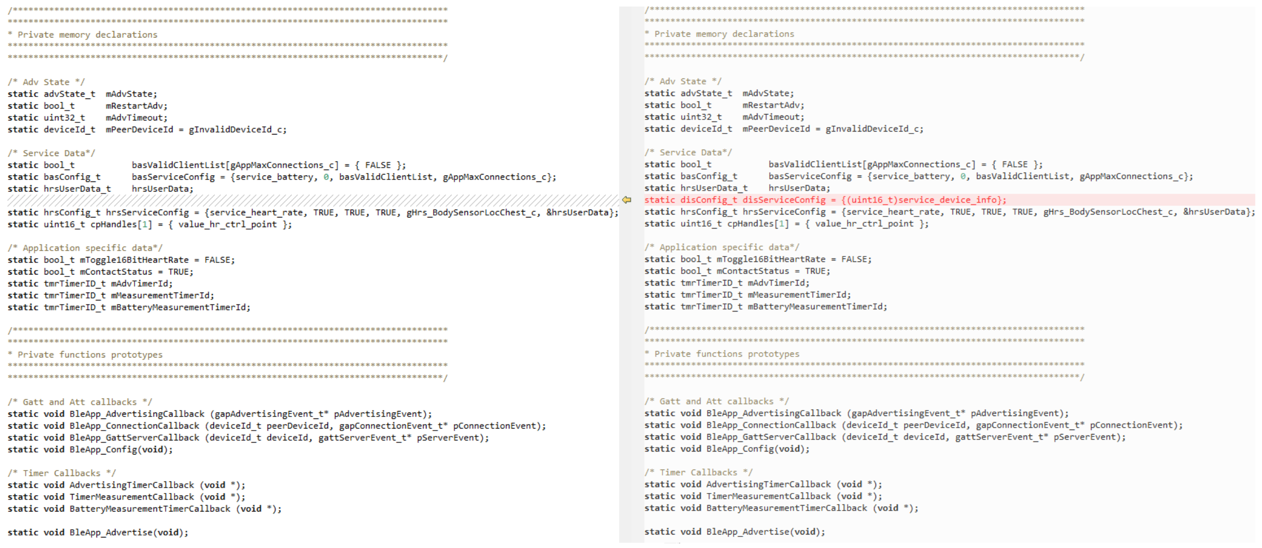
2. Add the function prototypes and global variables that are used by the OTAP client software. See the comparison in Figure 21 between HRS (left) and OTAP (right). As mentioned in the last step, this might depend on your application. The results should be similar as depicted in Figure 22.



```
/*******************************************************************************
********************************************************************************
* Private memory declarations
********************************************************************************
*******************************************************************************/

/* Adv State */
static advState_t  mAdvState;
static bool_t      mRestartAdv;
static uint32_t    mAdvTimeout;
static deviceId_t  mPeerDeviceId = gInvalidDeviceId_c;

/* Service Data*/
static bool_t           basValidClientList[gAppMaxConnections_c] = { FALSE };
static basConfig_t      basServiceConfig = {service_battery, 0, basValidClientList, gAppMaxConnections_c};
static hrsUserData_t    hrsUserData;
static hrsConfig_t hrsServiceConfig = {service_heart_rate, TRUE, TRUE, TRUE, gHrs_BodySensorLocChest_c, &hrsUserData};
static uint16_t cpHandles[1] = { value_hr_ctrl_point };

/* Application specific data*/
static bool_t mToggle16BitHeartRate = FALSE;
static bool_t mContactStatus = TRUE;
static tmrTimerID_t mAdvTimerId;
static tmrTimerID_t mMeasurementTimerId;
static tmrTimerID_t mBatteryMeasurementTimerId;

/*******************************************************************************
********************************************************************************
* Private functions prototypes
********************************************************************************
*******************************************************************************/

/* Gatt and Att callbacks */
static void BleApp_AdvertisingCallback (gapAdvertisingEvent_t* pAdvertisingEvent);
static void BleApp_ConnectionCallback (deviceId_t peerDeviceId, gapConnectionEvent_t* pConnectionEvent);
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t* pServerEvent);

static void BleApp_Config(void);

/* Timer Callbacks */
static void AdvertisingTimerCallback (void *);
static void TimerMeasurementCallback (void *);
static void BatteryMeasurementTimerCallback (void *);

static void BleApp_Advertise(void);

/*******************************************************************************
********************************************************************************
```

```
/*******************************************************************************
********************************************************************************
* Private memory declarations
********************************************************************************
*******************************************************************************/

static deviceId_t  mPeerDeviceId = gInvalidDeviceId_c;

/* Adv Parmeters */
static advState_t  mAdvState;

static tmrTimerID_t appTimerId;


/* Service Data */
static bool_t      basValidClientList[gAppMaxConnections_c] = { FALSE };
static basConfig_t basServiceConfig = {(uint16_t)service_battery, 0, basValidClientList, gAppMaxConnections_c};
static disConfig_t disServiceConfig = {(uint16_t)service_device_info};


/* Application Data */

static tmrTimerID_t mBatteryMeasurementTimerId;

/*******************************************************************************
********************************************************************************
* Private functions prototypes
********************************************************************************
*******************************************************************************/

/* Gatt and Att callbacks */
static void BleApp_AdvertisingCallback (gapAdvertisingEvent_t* pAdvertisingEvent);
static void BleApp_ConnectionCallback (deviceId_t peerDeviceId, gapConnectionEvent_t* pConnectionEvent);
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t* pServerEvent);

static void BleApp_Config(void);

static void BleApp_Advertise (void);
static void BatteryMeasurementTimerCallback (void *pParam);
/*******************************************************************************
********************************************************************************
```

Figure 21. Comparison between HRS (left) and OTAP (right) prototypes



```
/*******************************************************************************
********************************************************************************
* Private memory declarations
********************************************************************************
*******************************************************************************/

/* Adv State */
static advState_t  mAdvState;
static bool_t      mRestartAdv;
static uint32_t    mAdvTimeout;
static deviceId_t  mPeerDeviceId = gInvalidDeviceId_c;

/* Service Data*/
static bool_t           basValidClientList[gAppMaxConnections_c] = { FALSE };
static basConfig_t      basServiceConfig = {service_battery, 0, basValidClientList, gAppMaxConnections_c};
static hrsUserData_t    hrsUserData;

static hrsConfig_t hrsServiceConfig = {service_heart_rate, TRUE, TRUE, TRUE, gHrs_BodySensorLocChest_c, &hrsUserData};
static uint16_t cpHandles[1] = { value_hr_ctrl_point };

/* Application specific data*/
static bool_t mToggle16BitHeartRate = FALSE;
static bool_t mContactStatus = TRUE;
static tmrTimerID_t mAdvTimerId;
static tmrTimerID_t mMeasurementTimerId;
static tmrTimerID_t mBatteryMeasurementTimerId;

/*******************************************************************************
********************************************************************************
* Private functions prototypes
********************************************************************************
*******************************************************************************/

/* Gatt and Att callbacks */
static void BleApp_AdvertisingCallback (gapAdvertisingEvent_t* pAdvertisingEvent);
static void BleApp_ConnectionCallback (deviceId_t peerDeviceId, gapConnectionEvent_t* pConnectionEvent);
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t* pServerEvent);
static void BleApp_Config(void);

/* Timer Callbacks */
static void AdvertisingTimerCallback (void *);
static void TimerMeasurementCallback (void *);
static void BatteryMeasurementTimerCallback (void *);

static void BleApp_Advertise(void);
```

```
/*******************************************************************************
********************************************************************************
* Private memory declarations
********************************************************************************
*******************************************************************************/

/* Adv State */
static advState_t  mAdvState;
static bool_t      mRestartAdv;
static uint32_t    mAdvTimeout;
static deviceId_t  mPeerDeviceId = gInvalidDeviceId_c;

/* Service Data*/
static bool_t           basValidClientList[gAppMaxConnections_c] = { FALSE };
static basConfig_t      basServiceConfig = {service_battery, 0, basValidClientList, gAppMaxConnections_c};
static hrsUserData_t    hrsUserData;
static disConfig_t disServiceConfig = {(uint16_t)service_device_info};
static hrsConfig_t hrsServiceConfig = {service_heart_rate, TRUE, TRUE, TRUE, gHrs_BodySensorLocChest_c, &hrsUserData};
static uint16_t cpHandles[1] = { value_hr_ctrl_point };

/* Application specific data*/
static bool_t mToggle16BitHeartRate = FALSE;
static bool_t mContactStatus = TRUE;
static tmrTimerID_t mAdvTimerId;
static tmrTimerID_t mMeasurementTimerId;
static tmrTimerID_t mBatteryMeasurementTimerId;

/*******************************************************************************
********************************************************************************
* Private functions prototypes
********************************************************************************
*******************************************************************************/

/* Gatt and Att callbacks */
static void BleApp_AdvertisingCallback (gapAdvertisingEvent_t* pAdvertisingEvent);
static void BleApp_ConnectionCallback (deviceId_t peerDeviceId, gapConnectionEvent_t* pConnectionEvent);
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t* pServerEvent);
static void BleApp_Config(void);

/* Timer Callbacks */
static void AdvertisingTimerCallback (void *);
static void TimerMeasurementCallback (void *);
static void BatteryMeasurementTimerCallback (void *);

static void BleApp_Advertise(void);
```

Figure 22. Merging the OTAP prototypes into the project. Before (HRS left) and after (HRS-OTAP right).

3. Locate the “BleApp_Config” function. The “BleApp_Config” function configures the GAP role of the device (HRS – OTAP is a peripheral device), registers the notifiable attributes, prepares the services built on the database, and allocates some application timers. Add the “OtapClient_Config” and “Dis_Start” functions to initialize these services. See the following portion of the code.

```
/* Start services */
    hrsServiceConfig.sensorContactDetected = mContactStatus;
#if gHrs_EnableRRIntervalMeasurements_d
    hrsServiceConfig.pUserData->pStoredRrIntervals = MEM_BufferAlloc (sizeof (uint16_t) *
gHrs_NumOfRRIntervalsRecorded_c);
#endif
    Hrs_Start(&hrsServiceConfig);
    basServiceConfig.batteryLevel = BOARD_GetBatteryLevel();
    Bas_Start(&basServiceConfig);    (void)Dis_Start(&disServiceConfig);
```

```
if (OtapClient_Config() == FALSE)
{
    /* An error occured in configuring the OTAP Client */
    panic(0,0,0,0);
}
```

4. Locate the “BleApp_ConnectionCallback”. The connection callback is triggered whenever a connection event happens, such as a connection or disconnection.
   a. Go to the connection case. Include the “OtapCS_Subscribe” and the “OtapClient_HandleConnectionEvent” functions, this is implemented in the following portion of the code.

```
case gConnEvtConnected_c:
{
    /* Subscribe client*/
    Bas_Subscribe(&basServiceConfig, peerDeviceId);
    Hrs_Subscribe(peerDeviceId);
    (void)OtapCS_Subscribe(peerDeviceId);

    mPeerDeviceId = peerDeviceId;

    /* Stop Advertising Timer*/
    mAdvState.advOn = FALSE;
    TMR_StopTimer(mAdvTimerId);

    /* Start measurements */
    TMR_StartLowPowerTimer(mMeasurementTimerId, gTmrLowPowerIntervalMillisTimer_c,
    TmrSeconds(mHeartRateReportInterval_c), TimerMeasurementCallback, NULL);

    /* Start battery measurements */
    TMR_StartLowPowerTimer(mBatteryMeasurementTimerId, gTmrLowPowerIntervalMillisTimer_c,
    TmrSeconds(mBatteryLevelReportInterval_c), BatteryMeasurementTimerCallback, NULL);

    * Handle OTAP connection event */
    OtapClient_HandleConnectionEvent (peerDeviceId);
#if (cPWR_UsePowerDownMode)
    #ifdef MULTICORE_APPLICATION_CORE
        #if gErpcLowPowerApiServiceIncluded_c
        PWR_ChangeBlackBoxDeepSleepMode(gAppDeepSleepMode_c);
        PWR_AllowBlackBoxToSleep();
    #endif
    #else
    PWR_ChangeDeepSleepMode(gAppDeepSleepMode_c);
    PWR_AllowDeviceToSleep();
    #endif
#else
/* UI */
    LED_StopFlashingAllLeds();    Led1On();
#endif
}
break;
```

   b. Go to the disconnection case. Include the “OtapCS_Unsubscribe” and the “OtapClient_HandleDisconnectionEvent” functions, the implementation is shown in the following portion of the code.

```
case gConnEvtDisconnected_c:
{
    /* Unsubscribe client */
    Bas_Unsubscribe(&basServiceConfig, peerDeviceId);
    Hrs_Unsubscribe();
    (void)OtapCS_Unsubscribe();

    mPeerDeviceId = gInvalidDeviceId_c;

    /* Stop Timers*/
    TMR_StopTimer(mMeasurementTimerId);
    TMR_StopTimer(mBatteryMeasurementTimerId);
    OtapClient_HandleDisconnectionEvent(peerDeviceId);
if (cPWR_UsePowerDownMode)
    /* UI */
    Led1Off();

    /* Go to sleep */
#ifdef MULTICORE_APPLICATION_CORE
    #if gErpcLowPowerApiServiceIncluded_c
        PWR_ChangeBlackBoxDeepSleepMode(cPWR_DeepSleepMode);
    #endif
#else
    PWR_ChangeDeepSleepMode(cPWR_DeepSleepMode);
#endif
#else
    /* Restart advertising */
    BleApp_Start();
#endif
}
break;
```

5. Locate the “BleApp_GattServerCallback”, it manages all the incoming communications from the client devices. Add the GATT events that need to be handled by the OTAP client software (“gEvtAttributeWritten_c”, “gEvtMtuChanged”, “gEvtCharacteristicCccdWritten_c”, “gEvtAttributeWrittenWithoutResponse_c”, “gEvtHandleValueConfirmation_c” and “gEvtError”). Your Bluetooth LE project might share some common GATT events, if it is the case, you will need to add a conditional structure per each attribute handle. Focus on the “gEvtAttributeWritten_c” case, observe the conditional structure that was included for the “HRS control point” and the “OTAP control point” handling.

```

case gEvtAttributeWritten_c:
{
    handle = pServerEvent->eventData.attributeWrittenEvent.handle;
    status = gAttErrCodeNoError_c;
    if (handle == value_hr_ctrl_point)
    {
    status = Hrs_ControlPointHandler(&hrsUserData,
    pServerEvent->eventData.attributeWrittenEvent.aValue[0]);
    GattServer_SendAttributeWrittenStatus(deviceId, handle, status);
    }
     else
    {
    OtapClient_AttributeWritten (deviceId,
        pServerEvent->eventData.attributeWrittenEvent.handle,
        pServerEvent->eventData.attributeWrittenEvent.cValueLength,
        pServerEvent->eventData.attributeWrittenEvent.aValue);
    }
}
break;
case gEvtMtuChanged_c:
{
    OtapClient_AttMtuChanged (deviceId,
        pServerEvent->eventData.mtuChangedEvent.newMtu);
}
break;
case gEvtCharacteristicCccdWritten_c:
{
    OtapClient_CccdWritten (deviceId,
    pServerEvent->eventData.charCccdWrittenEvent.handle,
    pServerEvent->eventData.charCccdWrittenEvent.newCccd);
}
break;
case gEvtAttributeWrittenWithoutResponse_c:
{
    OtapClient_AttributeWrittenWithoutResponse (deviceId,
        pServerEvent->eventData.attributeWrittenEvent.handle,
        pServerEvent->eventData.attributeWrittenEvent.cValueLength,
        pServerEvent->eventData.attributeWrittenEvent.aValue);
}
break;
case gEvtHandleValueConfirmation_c:
{
    OtapClient_HandleValueConfirmation (deviceId);
}
break;
case gEvtError_c:
{
    attErrorCode_t attError = (attErrorCode_t) (pServerEvent->eventData.procedureError.error &
0xFF);    if (attError == gAttErrCodeInsufficientEncryption_c ||
    attError == gAttErrCodeInsufficientAuthorization_c ||
    attError == gAttErrCodeInsufficientAuthentication_c)
{
#if gAppUsePairing_d
#if gAppUseBonding_d
        bool_t isBonded = FALSE;
        /* Check if the devices are bonded and if this is true than the bond may have
        * been lost on the peer device or the security properties may not be sufficient.
        * In this case try to restart pairing and bonding. */
        if (gBleSuccess_c == Gap_CheckIfBonded(deviceId, &isBonded) &&
TRUE == isBonded)
#endif /* gAppUseBonding_d */
    {
    (void)Gap_SendSlaveSecurityRequest(deviceId, &gPairingParameters);
    }
#endif /* gAppUsePairing_d */
    }
}
break;
default:
break;
```

At this point, you have integrated the OTAP Client code into the HRS.

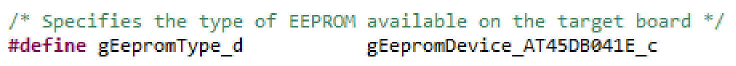
4.3 Modifications in the Project Settings and Storage Configurations

The OTAP client software included in the SDK package contains some linker configurations to generate the application offset needed for the OTAP Bootloader software and split the flash memory in accord of the storage method desired. Such configurations are not part of the HRS demo, so it should be included to integrate the OTAP on the application. Follow the next steps to set the project settings and the storage configurations.

1. Locate the “app_preinclude.h” file under the source folder of the project.

a. To select **external flash storage** method, set the “gEepromType” define to “gEepromDevice_AT45DB041E_c” (default in the HRS-OTAP software attached).

b. To select **internal flash storage** method, set the “gEepromType” define to “gEepromDevice_InternalFlash_c”.



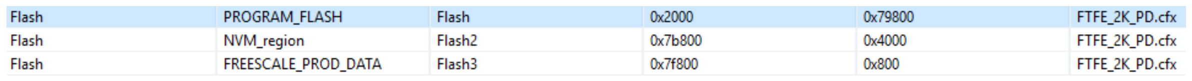
```
/* Specifies the type of EEPROM available on the target board */
#define gEepromType_d          gEepromDevice_AT45DB041E_c
```

Figure 23. Configuring the storage method at the preinclude file

2. Click the HRS-OTAP demo in the MCUXpresso workspace.

3. Navigate to “Project->Properties” in MCUXpresso IDE. Go to “C/C++ Build->MCU settings”.

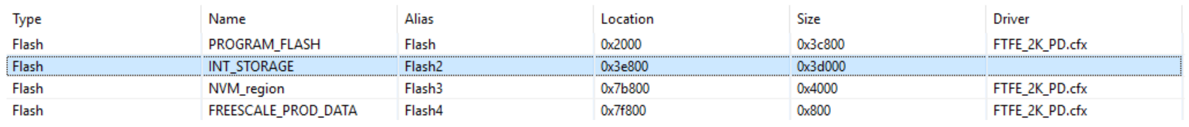
a. To select **external flash storage** method, configure the fields depicted in Figure 24 in the “Memory details” pane (default in the HRS-OTAP software attached).



Flash	PROGRAM_FLASH	Flash	0x2000	0x79800	FTFE_2K_PD.cfx
Flash	NVM_region	Flash2	0x7b800	0x4000	FTFE_2K_PD.cfx
Flash	FREESCALE_PROD_DATA	Flash3	0x7f800	0x800	FTFE_2K_PD.cfx

Figure 24. Configuring external storage method

b. To select **internal flash storage** method, configure the fields depicted in Figure 25 in the “Memory details” pane.



Type	Name	Alias	Location	Size	Driver
Flash	PROGRAM_FLASH	Flash	0x2000	0x3c800	FTFE_2K_PD.cfx
Flash	INT_STORAGE	Flash2	0x3e800	0x3d000	
Flash	NVM_region	Flash3	0x7b800	0x4000	FTFE_2K_PD.cfx
Flash	FREESCALE_PROD_DATA	Flash4	0x7f800	0x800	FTFE_2K_PD.cfx

Figure 25. Configuring internal storage method

4. Clean and build the project.

At this point, you have finally integrated the OTAP service on the Bluetooth LE-based application.

4.4 Adding Low Power Support on the Application

In order to include low power support for OTAP as well, it is necessary to take some considerations.

1. The “OTA_PushImageChunk” function must be altered to disallow the device to sleep while is writing the data into the flash device and to allow to the device return in low power mode when it has finished this procedure. Locate the “OTA_PushImageChunk” function in framework->OtaSupport->Source->OtaSupport.c file. Call “PWR_DisallowDeviceToSleep” before entering on the “OTA_PushImageChunk” code and call “PWR_AllowDeviceToSleep” before return from the function. See the following example:

```
/* Include */
#if (cPWR_UsePowerDownMode)
#include "PWR_Interface.h"
#include "PWR_Configuration.h"
#endif
/* Public functions */
otaResult_t OTA_PushImageChunk(uint8_t* pData, uint16_t length, uint32_t* pImageLength, uint32_t
*pImageOffset)
{
#if (cPWR_UsePowerDownMode)
PWR_DisallowDeviceToSleep();
#endif
/*************************** OTA_PushImageChunk content init **************************/
/*************************** OTA_PushImageChunk content end ***************************/
#if (cPWR_UsePowerDownMode)
PWR_AllowDeviceToSleep();
#endif
return status;
}
```

2. For Deep Sleep 5 and 8 modes (DSM5 and DSM8), who were developed based on VLLS modes, the wake-up routine performs an SW reset, so the SFR's values and the application context are lost and must be recovered after leaving low power state. The warm boot callback restores the application context and the clocking configuration, but SPI peripheral needed for the external storage method is not restored. In other words, the SPI must be initialized in the warm boot callback. The following portion of code can be found in the HRS-OTAP example in the board->board.c file.

```

/* Include */
#include "SPI_Adapter.h"

/* Private type definitions and macros */
ifndef gEepromSpiInstance_c
#define gEepromSpiInstance_c 0
#endif
static spiState_t mEepromSpiState;

/* Private functions prototypes */
static void SPI_Hardware_Init(void);
    /* Private functions */
    static void SPI_Hardware_Init(void){
    spiBusConfig_t spiConfig = {
    .bitsPerSec = 8000000,
    .master = TRUE,
    .clkActiveHigh = TRUE,
    .clkPhaseFirstEdge = TRUE,
    .MsbFirst = TRUE
};

    gpioOutputPinConfig_t mEepromSpiCsCfg = {
        .gpioPort = gpioPort_C_c,
        .gpioPin = 19,
        .outputLogic = 1,
        .slewRate = pinSlewRate_Fast_c,
        .driveStrength = pinDriveStrength_Low_c
};
    Spi_Init(gEepromSpiInstance_c, &mEepromSpiState, NULL, NULL);
    Spi_Configure(gEepromSpiInstance_c, &spiConfig);
    GpioOutputPinInit(&mEepromSpiCsCfg, 1);
}

void BOARD_WarmbootCb(void){
/*****************************************************************************************/
/**************************** Warmboot Callback Development ****************************/
/*****************************************************************************************/
SPI_Hardware_Init();
}
```

3. The application files must also contain the APIs required for low power management, changing the DSM mode depending on if the device is on advertising, connection or idle state and should be able to go to sleep whenever the idle task is active. You could base on the HRS-OTAP application as a reference for the implementation of low power support on your code.

5 Testing the HRS-OTAP Demo

The test case example that was designed to demonstrate the OTAP integration at the *5.4 Testing the HRS-OTAP Software* section, makes use of the listed software:

- OTAP Client SDK software, programmed in the FRDM-KW36 board.
- An SREC software update of the HRS-OTAP example.
- An SREC software update of the HRS SDK example.

The following sections explain how to build the software required for the testing example proposed in this document. However, you are free to decide which software or steps are not relevant to you.

5.1 Preparing the OTAP Client SDK Software

1. Attach your FRDM-KW36 board on the PC.
2. Program the OTAP Bootloader on the FRDM-KW36, you can simply drag and drop the prebuilt binary file from the following path on the board:

   <FRDM-KW36_SDK_root>\tools\wireless\binaries\bootloader_otap_frdmkw36.bin

3. Open MCUXpresso IDE. Click the “Import SDK example(s)” option in the “Quickstart Panel” view.

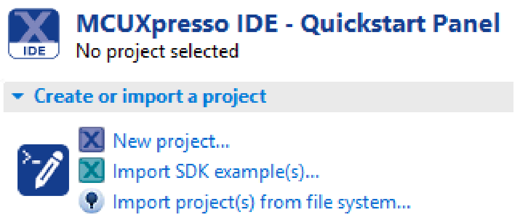
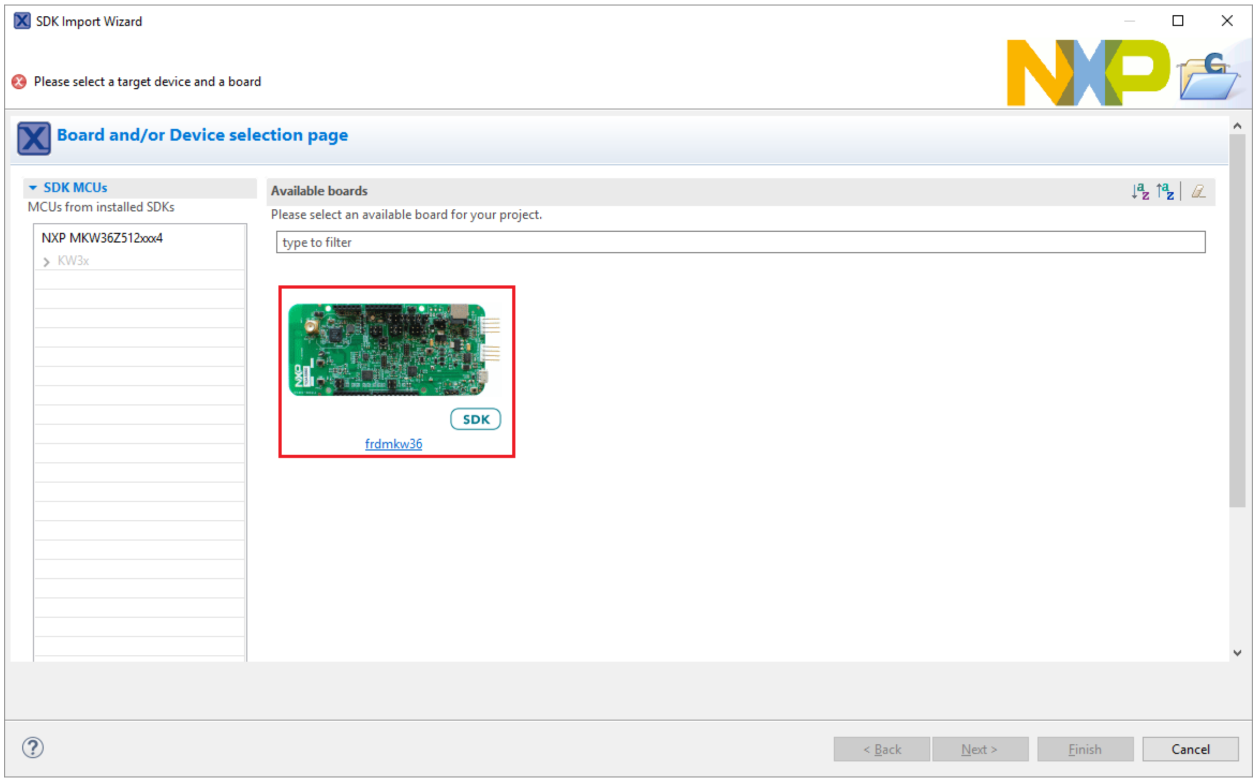

MCUXpresso IDE - Quickstart Panel
No project selected
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...

Figure 26. Quickstart Panel Perspective

4. Click twice on the frdmkw36 icon.



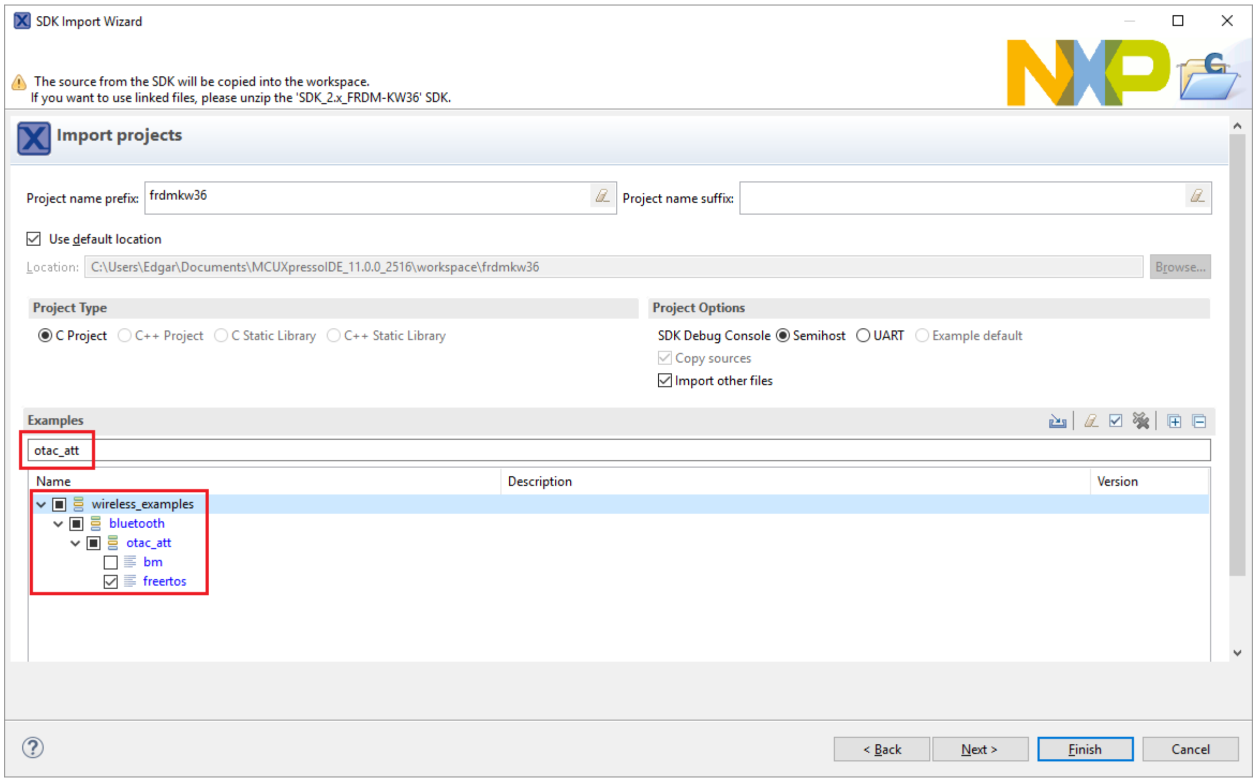
SDK Import Wizard

Please select a target device and a board

Board and/or Device selection page

Figure 27. Device Selection Perspective

5. Type “otac_att” in the examples textbox and select the freertos project at “wireless_examples->bluetooth->otac_att->freertos”. Click the “Finish” button.



SDK Import Wizard

The source from the SDK will be copied into the workspace.
If you want to use linked files, please unzip the 'SDK_2.x_FRDM-KW36' SDK.

Import projects

Project name prefix: frdmkw36 Project name suffix:

☑ Use default location

Location: C:\Users\Edgar\Documents\MCUXpressoIDE_11.0.0_2516\workspace\frdmkw36

Project Type: ● C Project ○ C++ Project ○ C Static Library ○ C++ Static Library

Project Options: SDK Debug Console ● Semihost ○ UART ○ Example default; ☑ Copy sources; ☑ Import other files

Examples: otac_att

```
Name
wireless_examples
  bluetooth
    otac_att
      ☐ bm
      ☑ freertos
```

Figure 28. Importing the OTAP client project on the workspace

6. Set the storage configurations:

   a. Open the “app_preinclude.h” file located in the source folder of the project:

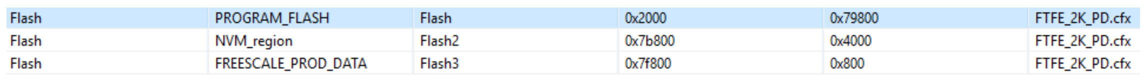
      - To select the external flash storage method (AT45DB041E_c external flash), set the “gEepromType” define to “gEepromDevice_AT45DB041E_c”
      - To select the internal flash storage method (On-chip FlexNVM memory), set the “gEepromType” define to “gEepromDevice_InternalFlash_c”

```
/* Specifies the type of EEPROM available on the target board */
#define gEepromType_d           gEepromDevice_AT45DB041E_c
```

Figure 29. Configuring the storage method at the preinclude file

   b. Navigate to “Project->Properties” in MCUXpresso IDE. Go to “C/C++ Build->Settings->Tool Settings->MCU Linker->Miscellaneous” perspective.

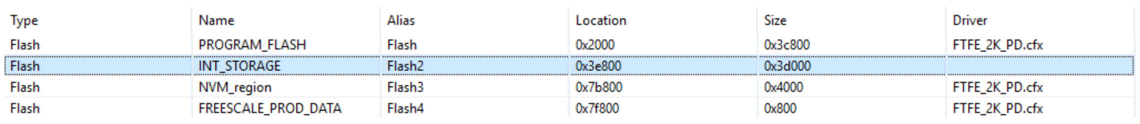
      - To select **external flash storage** method, configure the fields depicted in the following figure in “Memory details” pane.



Flash	PROGRAM_FLASH	Flash	0x2000	0x79800	FTFE_2K_PD.cfx
Flash	NVM_region	Flash2	0x7b800	0x4000	FTFE_2K_PD.cfx
Flash	FREESCALE_PROD_DATA	Flash3	0x7f800	0x800	FTFE_2K_PD.cfx

Figure 30. Configuring external storage method

      - To select **internal flash storage** method, configure the fields depicted in the following figure in “Memory details” pane.



Type	Name	Alias	Location	Size	Driver
Flash	PROGRAM_FLASH	Flash	0x2000	0x3c800	FTFE_2K_PD.cfx
Flash	INT_STORAGE	Flash2	0x3e800	0x3d000	
Flash	NVM_region	Flash3	0x7b800	0x4000	FTFE_2K_PD.cfx
Flash	FREESCALE_PROD_DATA	Flash4	0x7f800	0x800	FTFE_2K_PD.cfx

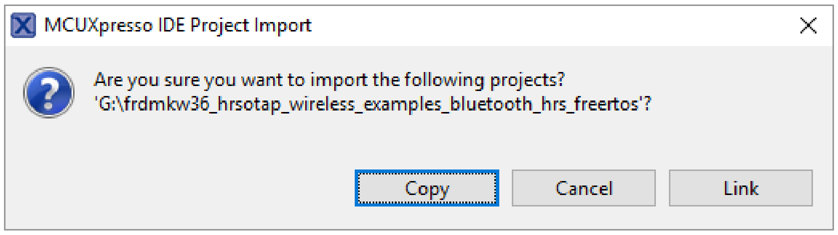
Figure 31. Configuring internal storage method

7. Clean and build the project. Flash the project on the “FRDM-KW36” board.

At this point, you have programming and configuring the OTAP client software on your board. You can communicate to a server and request for a software update.

5.2 Creating an HRS-OTAP S-Record Image to Update the Software

1. Install the HRS-OTAP demo provided with this document in your MCUXpresso IDE. You can drag and drop the project from your installation path to the MCUXpresso workspace. A warning message is shown, click the “Copy” button to clone the original example.

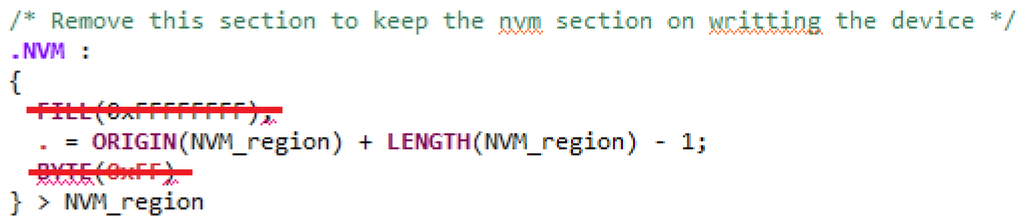


MCUXpresso IDE Project Import

Are you sure you want to import the following projects?
'G:\frdmkw36_hrsotap_wireless_examples_bluetooth_hrs_freertos'?

Figure 32. Importing the HRS-OTAP demo on the MCUXpresso workspace

2. Open the “end_text.ldt” linker script located at the linkscripts folder in the workspace. Locate the section placement of Figure 33 and remove the “FILL” and the “BYTE” statements. This step is needed only to build the SREC image file to reprogram the device.



```
/* Remove this section to keep the nvm section on writting the device */
.NVM :
{
  ~~FILL(0xFFFFFFFF);~~
  . = ORIGIN(NVM_region) + LENGTH(NVM_region) - 1;
  ~~BYTE(0xFF)~~
} > NVM_region
```

Figure 33. Preparing the linker file

3. Clean and build the project.

4. Deploy the “Binaries” icon in the workspace. Click the right mouse button on the “.axf” file. Select the “Binary Utilities->Create S-Record” option. The S-Record file is saved at the “Debug” folder in the workspace with “.s19” extension.

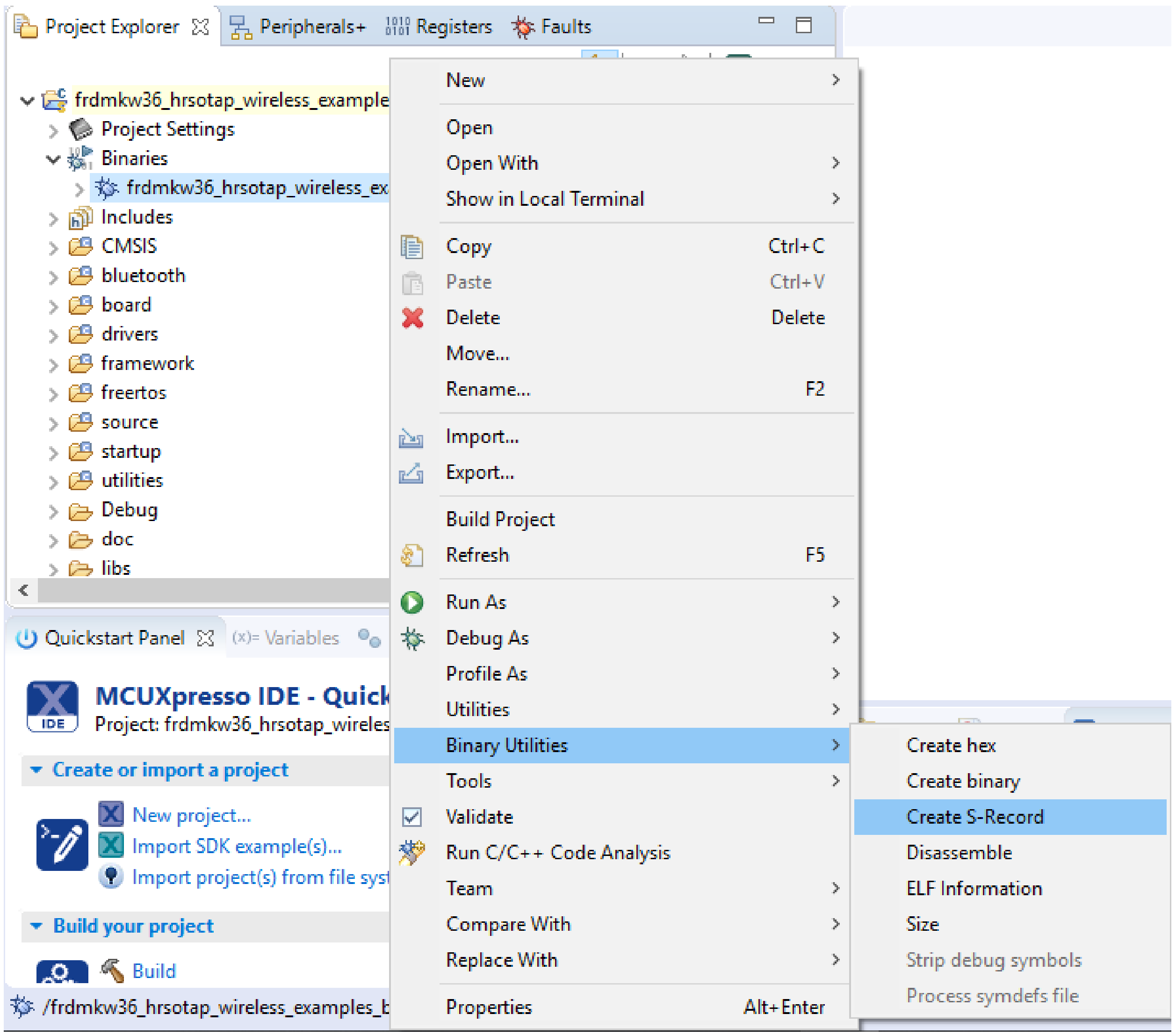


Figure 34. Creating the SREC file

5. Save this file in a known location on your smartphone.

5.3 Creating an HRS S-Record Image to Update the Software

1. Open MCUXpresso IDE. Click the “Import SDK example(s)” option in the “Quickstart Panel” view, the device selection perspective is shown. Click twice on the frdmkw36 icon.
2. Type “hrs” in the examples textbox and select the freertos project at “wireless_examples->bluetooth->hrs->freertos”. Click the “Finish” button.

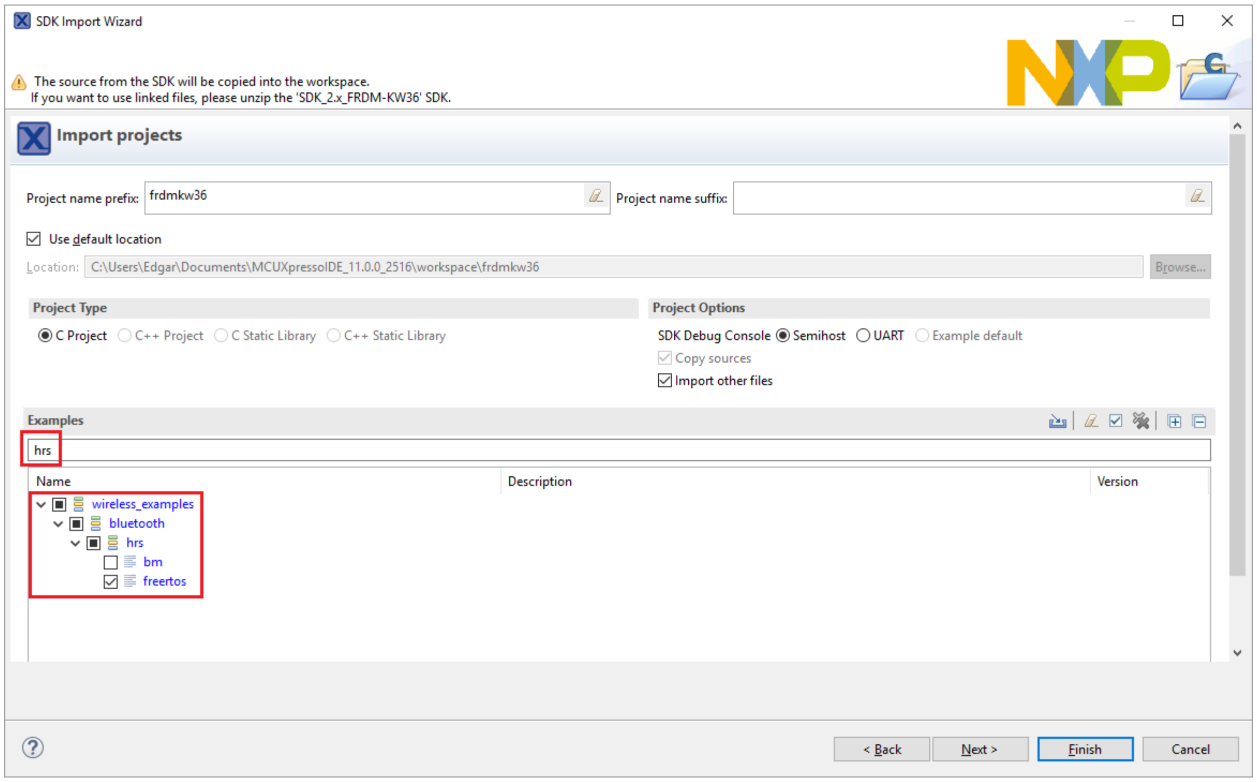


Figure 35. Importing the HRS project on the workspace

3. Open the “app_preinclude.h” file under the source folder at the MCUXpresso workspace. Locate the “cPWR_UsePowerDownMode” macro and change its value to zero. This step is not mandatory, but it is useful at running time to confirm whenever the software update has been successfully programmed by the OTAP bootloader.

```
/* Enable/Disable PowerDown functionality in PwrLib */
#define cPWR_UsePowerDownMode 0
```

4. Navigate to “Project->Properties->C/C++ Build->MCU settings”. Configure the following fields and save the changes.

Flash	PROGRAM_FLASH	Flash	0x2000	0x79800	FTFE_2K_PD.cfx
Flash	NVM_region	Flash2	0x7b800	0x4000	FTFE_2K_PD.cfx
Flash	FREESCALE_PROD_DATA	Flash3	0x7f800	0x800	FTFE_2K_PD.cfx

Figure 36. Configuring the memory layout

5. Navigate to the workspace. Locate the “linkscripts” folder and include into it the “main_text_section.ldt” linker script. You can copy and paste from the OTAP client SDK example.

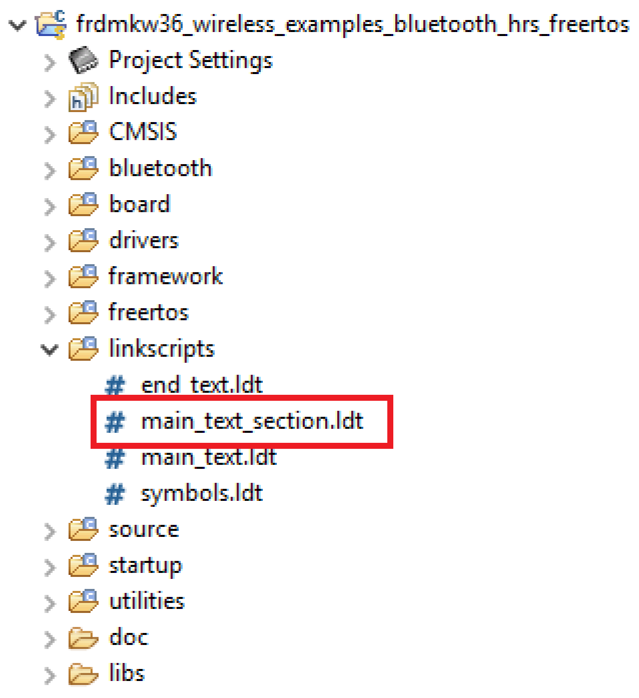
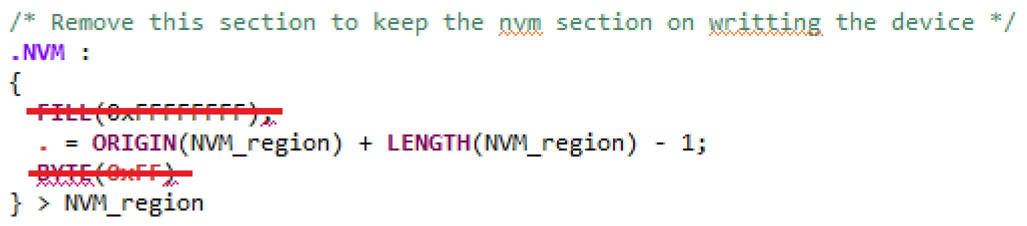


Figure 37. Importing linker scripts

6. Open the “end_text.ldt” linker script located at the linkscripts folder in the workspace. Locate the section placement of Figure 38 and remove the “FILL” and the “BYTE” statements.



```
/* Remove this section to keep the nvm section on writting the device */
.NVM :
{
  ~~FILL(0xFFFFFFFF);~~
  . = ORIGIN(NVM_region) + LENGTH(NVM_region) - 1;
  ~~BYTE(0xFF);~~
} > NVM_region
```

Figure 38. Preparing the linker file

7. Include the “OtaSupport” folder and its files in the “framework” folder. Include “External” folder and its files in the “framework->Flash” folder. This step can be done in the same way as explained in *4.1. Importing the OTAP Service and Framework Services into the HRS* section.

8. Clean and build the project.

9. Deploy the “Binaries” icon in the workspace. Click the right mouse button on the “.axf” file. Select the “Binary Utilities->Create S-Record” option. The S-Record file is saved at the “Debug” folder in the workspace with “.s19” extension.

10. Save this file in a known location on your smartphone.

5.4 Testing the HRS-OTAP Software

To exemplify the testing case of this section, see Figure 39. The FRDM-KW36 contains the OTAP client software. The OTAP client requests a software update from the OTAP server (the smartphone). This software image is the HRS-OTAP demo. The FRDM-KW36 at this point has been updated and can handle all the incoming communication from an HR central or the OTAP server. To demonstrate that you can continue updating the software of the KW36 device, you can connect the HRS-OTAP to an OTAP server and request a software update that only contains the HRS example. From this point, you cannot continue updating the software since the OTAP service was not included in the last software upgrade. This example was designed to understand the key points of the OTAP integration. However, the main purpose of this application note is creating software updates that include the OTAP service and continue upgrading and improving the KW36 device.

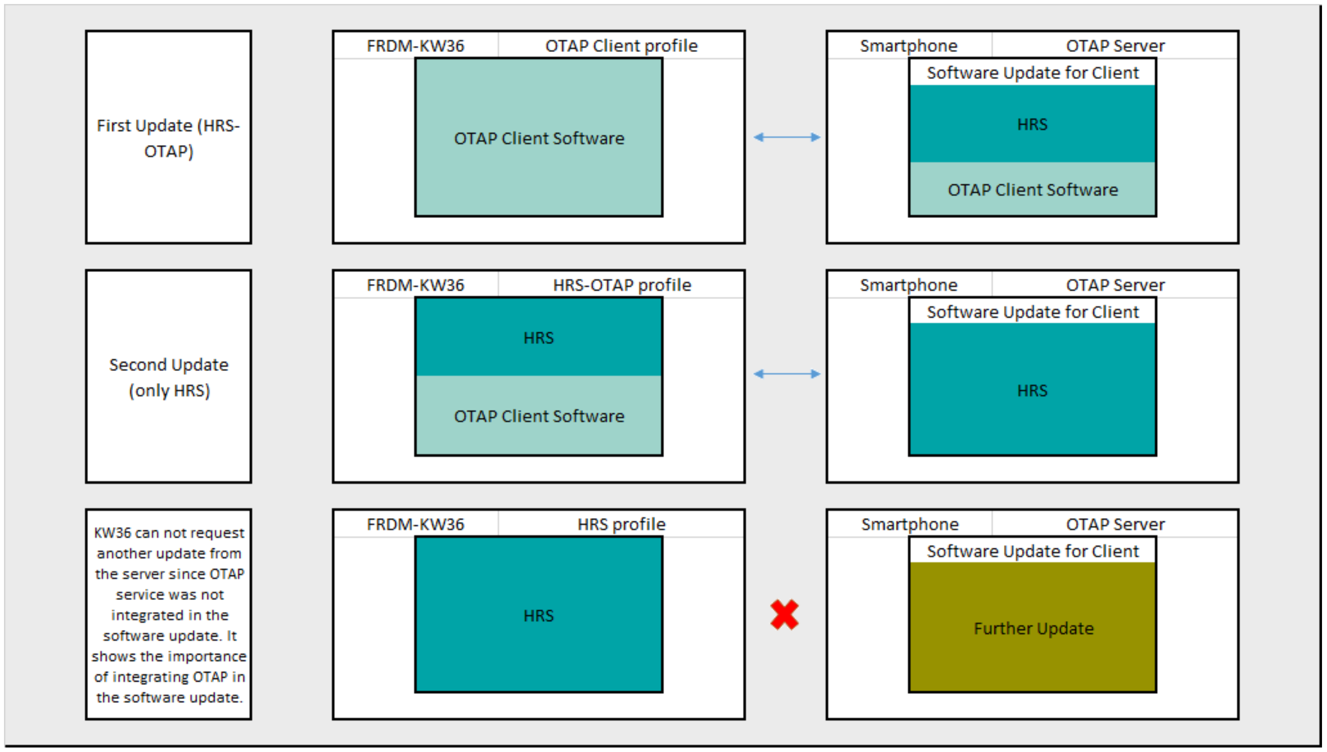


Figure 39. Proposed Test

1. Open the IoT Toolbox App and select the OTAP demo. Click “SCAN” to start scanning for a suitable advertiser.

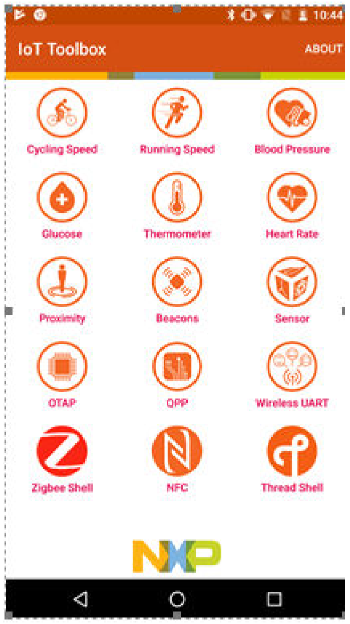


Figure 40. IoT Toolbox Interface

2. Press the ADV button (SW2) on the FRDM-KW36 board to start advertising.
3. Create a connection with the “NXP_OTAA” device. Then, the OTAP interface is displayed on your smartphone.

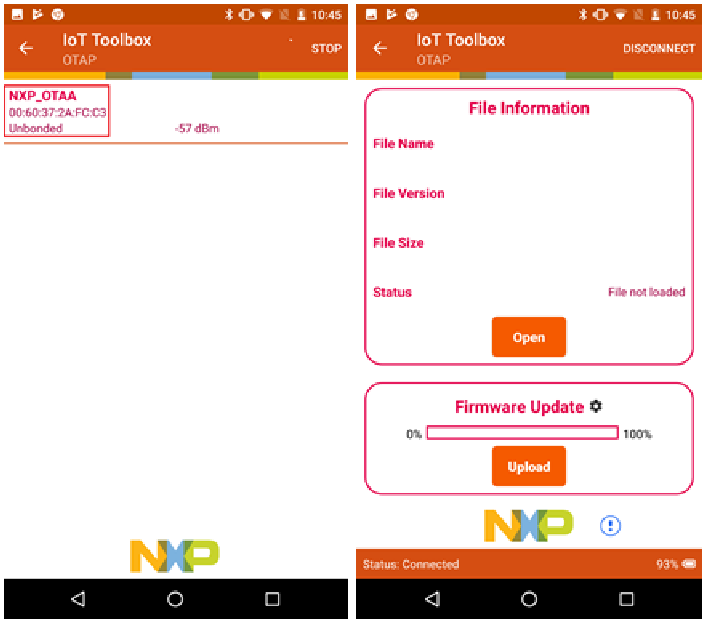


Figure 41. Connecting the OTAP client and the OTAP server

4. Click the “Open” button and search for the “HRS-OTAP” SREC file.
5. Click “Upload” to start the transfer. Wait until the confirmation message is displayed.

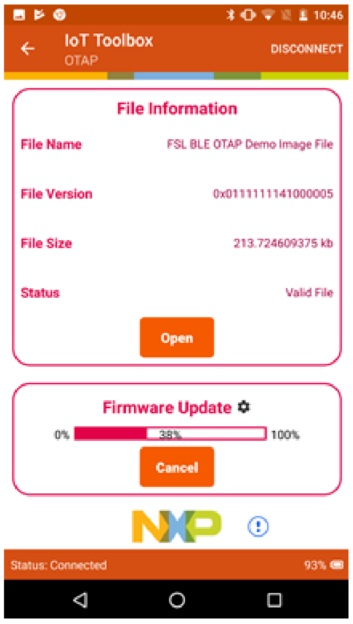


Figure 42. Updating the OTAP client to HRS-OTAP

6. Wait few seconds until the OTAP bootloader has finished programming the new image. The HRS-OTAP application starts automatically (The RGB LED blinks).
7. Press the ADV button (SW2) on the FRDM-KW36 board to start advertising. Verify that the device can be detected by both, HRS and OTAP applications of the IoT Toolbox. The device is named as “NXP_HRS_OTAP”. You can create a connection and interact with both demos.

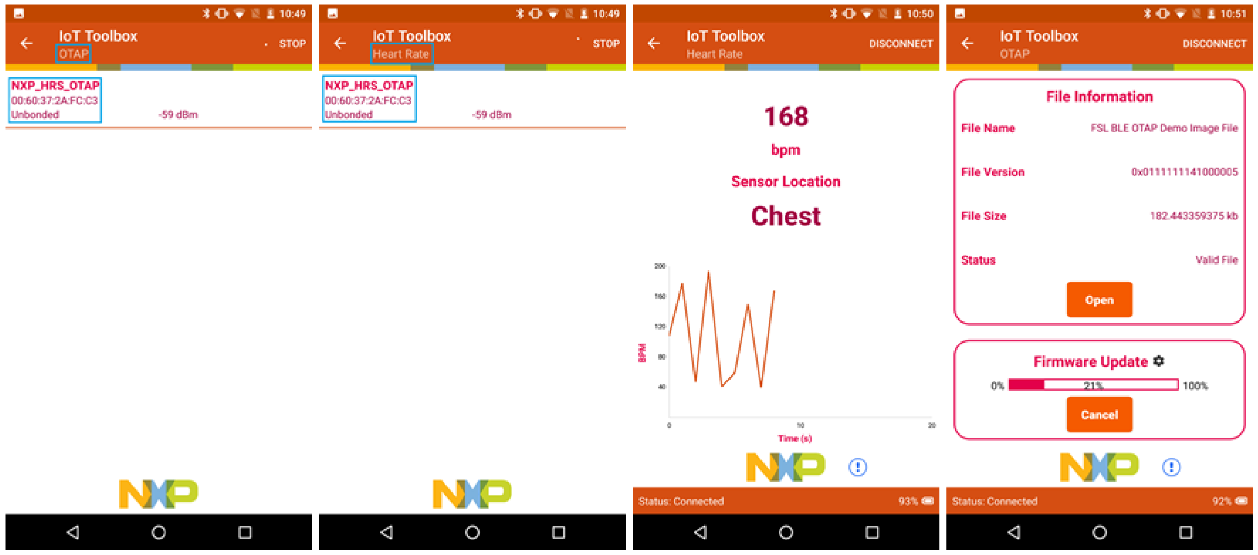


Figure 43. HRS-OTAP device detected by both applications

8. Connect the HRS-OTAP device with the OTAP smartphone application. Update the software using the “HRS” SREC file.
9. Confirm that the device has been updated to a simple HRS, making use of the HRS-OTAP demo. Press the ADV button (SW2) on the FRDM-KW36 board to start advertising. Now the device’s name is “NXP_HRS”. Connect the device with the HRS IoT Toolbox app and verify that it works as expected.

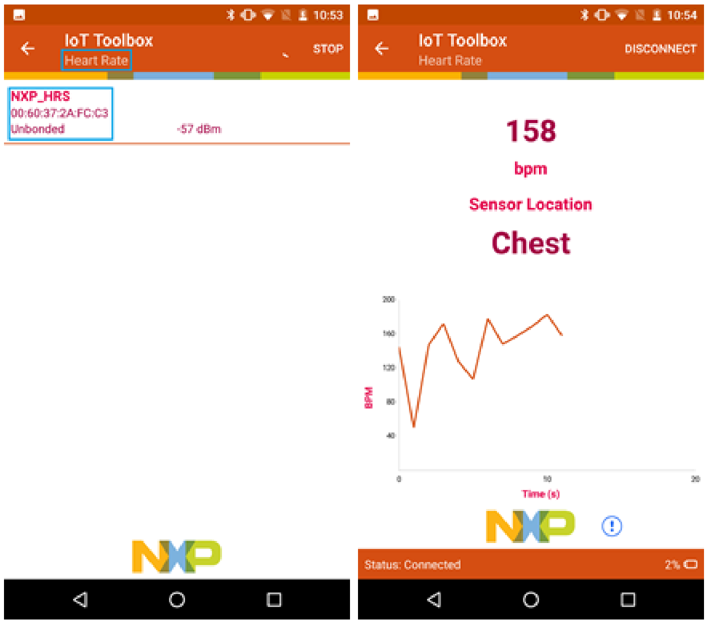


Figure 44. HRS-OTAP device detected by both applications

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 11 March 2020
Document identifier: AN12775