AN13855

KW45B41Z/K32W148 - Integrating the OTAP Client Service into a Bluetooth LE Peripheral Device

Rev. 0 — 1 March 2023

Application note

Document information

Information	Content
Keywords	AN13855, KW45B41Z-EVK, K32W148-EVK, Over the Air Programming (OTAP) Client Service, Bluetooth Low Energy Peripheral Device, Wireless UART project, MCUXpresso Integrated Development Environment (IDE), Bluetooth Low Energy SDK projects
Abstract	This document provides the steps and process for integrating Over the Air Programming (OTAP) Client Service into a Bluetooth Low Energy peripheral device.

1 Introduction

This document provides the steps and process for integrating the Over the Air Programming (OTAP) Client Service into a Bluetooth Low Energy peripheral device. It uses the KW45B41Z/K32W148 hardware platforms and other software to implement this demonstration.

The OTAP custom Bluetooth Low Energy service provided by NXP enables the developers to upgrade the software that the MCU contains. It removes the need of cables between the device to be upgraded and the device that contains the new software.

The demo applications implement a typical scenario where a new image is sent from a PC via serial interface to a Bluetooth Low Energy OTAP device. It is transferred over the air to an OTAP client, which is the target of the upgrade image.

The best way to take advantage of the OTAP service is to integrate it into the Bluetooth Low Energy application. In that way, you can reprogram the device as many times as required.

2 OTAP client software

This section describes the OTAP memory management during the OTAP client software update process. It lists steps for implementation of the OTAP client software included in the SDK package for KW45B41Z-EVK or K32W148-EVK. It explains the advantages of integrating OTAP client software into your application and shows the expected results.

2.1 OTAP memory management during the update process

1. By default, the KW45B41Z/K32W148 flash memory is partitioned in three main regions:
   - A 1024 kB program flash array is divided into 7 sectors with a flash address range from `0x0000_0000` to `0x000F_FFFF`.
   - A linker file creates a mechanism to have two different software coexisting in the same device. After the linker file is implemented, each software is stored in a different memory region. In the KW45B41Z device, the application has a reserved slot of memory from **0x0x400 to 0x77BFF**.
2. The developer should use the linker script to specify the code to be stored with an offset while generating a new image file for the OTAP client device. The new application should contain the bootloader flags at the corresponding address to work properly.
3. At the connection state, the OTAP server sends the image packets, known as chunks, to the OTAP client via Bluetooth LE. The OTAP client can store these chunks in the external SPI flash or in the on-chip FlexNVM region. The destination of the code can be selected in the OTAP client software.
4. When all chunks have been sent from the OTAP server to the OTAP client, the image transfer is complete. Then, 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. It then resets the MCU to execute the ROM bootloader code. The ROM bootloader reads the bootloader flags to get the information needed to program the device. It also triggers a command to reprogram the MCU with the new application.
5. If the new application was built with an offset, the ROM bootloader programs the device starting from the 0x400 address and the new image overwrites the OTAP client application. Then the ROM 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 cannot be reprogrammed due to lack of OTAP functionality.

2.2 Advantages of the OTAP service integration

The OTAP client software can reprogram the device only once, because the new application overwrites it.

Consider a case where an OTAP client device is programmed with the OTAP client software and it requests an update, for example, a Wireless UART. For this process, the OTAP server must send a Wireless UART image to the OTAP client device. After the reprogramming process, the device that was previously OTAP client, has now updated into a Wireless UART.

By default, the Wireless UART is not capable of communicating with the OTAP server and request for another update. However, in case the Wireless UART image includes the OTAP client service as well, it is possible for the client device to request another software update.

For example, it could have a modified baud rate with OTAP client service. Later, the device could request another software update from the OTAP server as the baud rate software includes the OTAP client. That way, the developer can continue upgrading the software as many times as needed. In other words, the application sent over the air should include OTAP service support, to be able to upgrade the software on the OTAP client device in future.

3 Pre-requisites

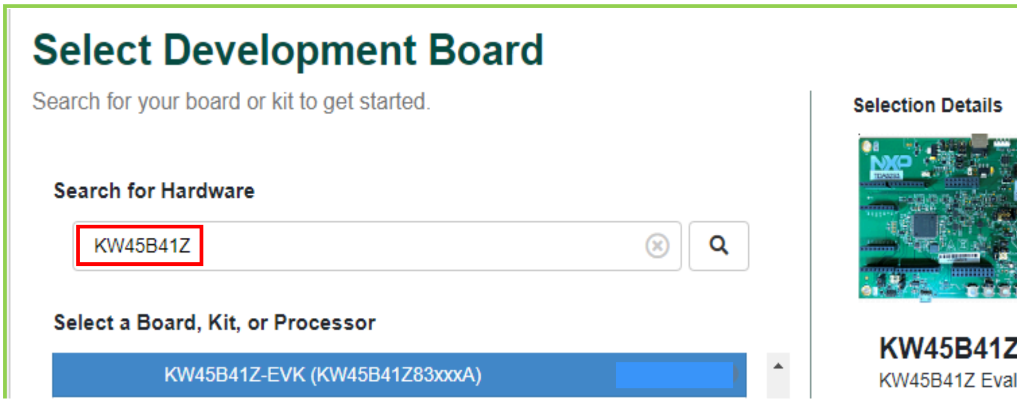
This document also provides a functional demo of the OTAP service integration based on the Wireless UART project. The demo is available in the KW45B41Z/K32W148 SDK package and developed using the MCUXpresso Integrated Development Environment (IDE). The following are required for implementing the Wireless UART-OTAP integration demo:

- KW45B41Z/K32W148 Evaluation Kit (KW45B41Z-EVK / K32W148-EVK)
- A smartphone with the NXP application- IoT Toolbox, available for Android and iOS.
- MCUXpresso IDE v11.6.0 or later
- KW45B41Z SDK v.2_12_1
- Wireless UART – OTAP demo package

3.1 Downloading and installing the software development kit

This section provides all the steps needed to download the SDK for the KW45B41Z-EVK used as a starting point.

- Navigate to the MCUXpresso website.
- Click **Select Development Board**. Log in with your registered account.
- In the **Search by Name** field, search for KW45B41Z-EVK / K32W148-EVK. Then click the suggested board and click **Build MCUXpresso SDK**.



Select Development Board

Search for your board or kit to get started.

Search for Hardware

KW45B41Z

Select a Board, Kit, or Processor

KW45B41Z-EVK (KW45B41Z83xxxA)

Selection Details

KW45B41Z

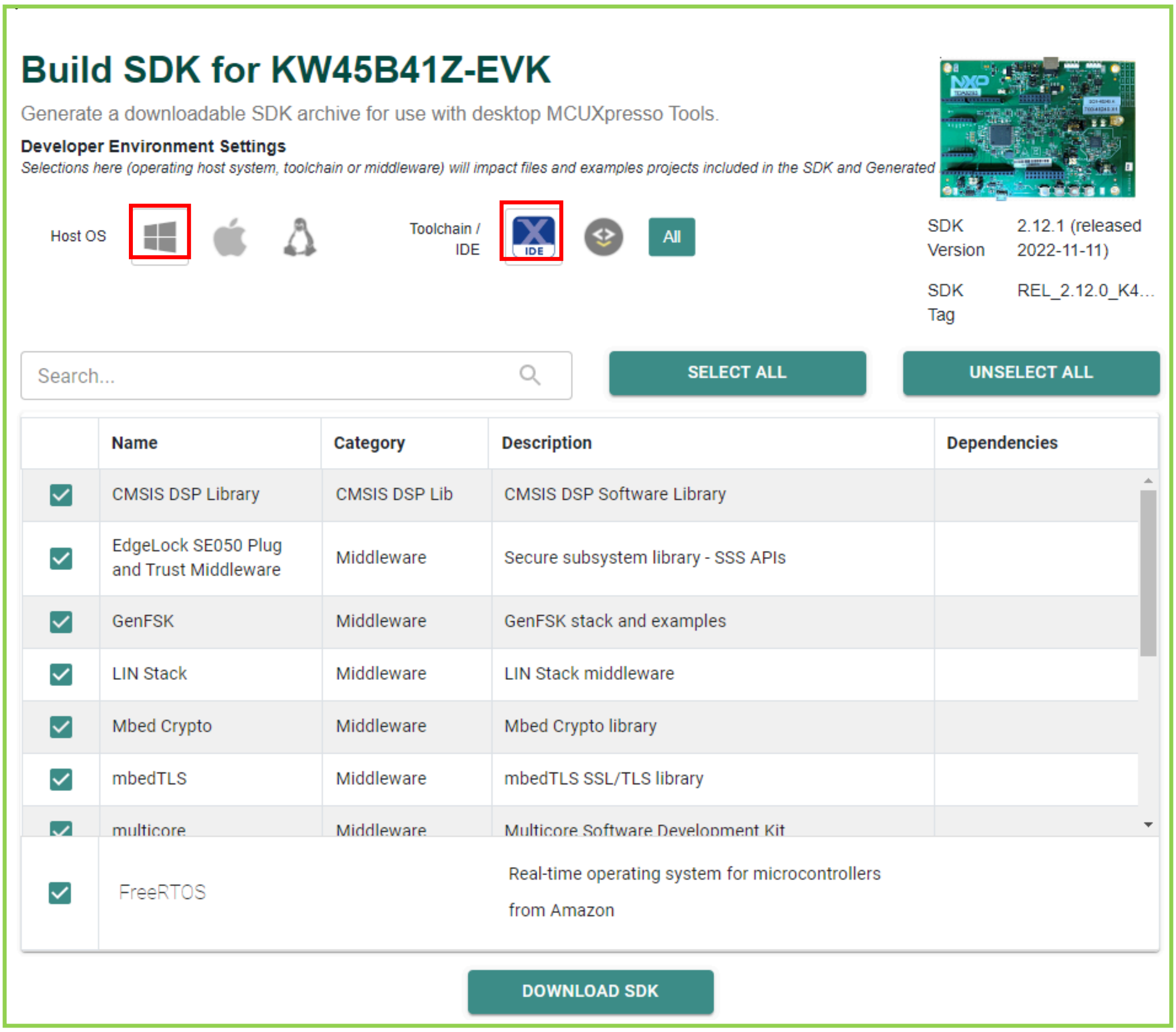
KW45B41Z Eval

Figure 1. Selecting the Development Board

- Select **MCUXpresso IDE** in the **Toolchain/IDE** combo box. Select the supported OS. Click **Download SDK**. Wait for a few minutes until the system gets the package into your account on the MCUXpresso webpage. Read and accept the license agreement. The SDK download starts automatically on your PC.

.

Figure 2. Downloading the SDK for the supported OS



Build SDK for KW45B41Z-EVK

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

Developer Environment Settings

Selections here (operating host system, toolchain or middleware) will impact files and examples projects included in the SDK and Generated

Host OS

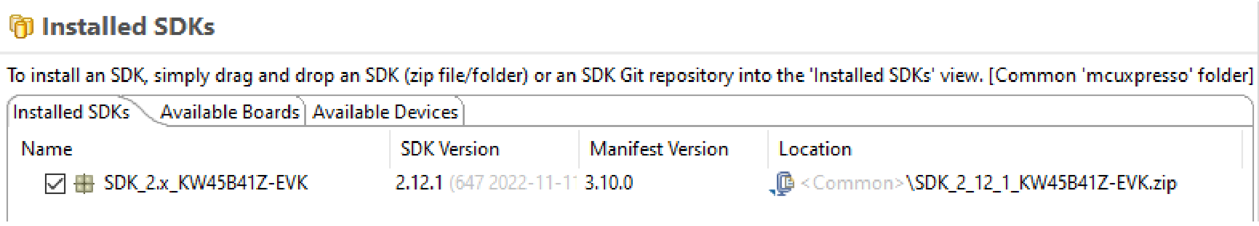
Toolchain / IDE

SDK Version: 2.12.1 (released 2022-11-11)

SDK Tag: REL_2.12.0_K4...

	Name	Category	Description	Dependencies
☑	CMSIS DSP Library	CMSIS DSP Lib	CMSIS DSP Software Library	
☑	EdgeLock SE050 Plug and Trust Middleware	Middleware	Secure subsystem library - SSS APIs	
☑	GenFSK	Middleware	GenFSK stack and examples	
☑	LIN Stack	Middleware	LIN Stack middleware	
☑	Mbed Crypto	Middleware	Mbed Crypto library	
☑	mbedTLS	Middleware	mbedTLS SSL/TLS library	
☑	multicore	Middleware	Multicore Software Development Kit	
☑	FreeRTOS		Real-time operating system for microcontrollers from Amazon	

- Open **MCUXpresso IDE**. Drag and drop the KW45B41Z-EVK SDK zip in the list that displays the installed SDKs.



Installed SDKs

To install an SDK, simply drag and drop an SDK (zip file/folder) or an SDK Git repository into the 'Installed SDKs' view. [Common 'mcuxpresso' folder]

Name	SDK Version	Manifest Version	Location
☑ SDK_2.x_KW45B41Z-EVK	2.12.1 (647 2022-11-1	3.10.0	<Common>\SDK_2_12_1_KW45B41Z-EVK.zip

Figure 3. Viewing installed SDKs

Now, you have downloaded and installed the SDK package for the KW45B41Z-EVK.

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 Wireless UART project as a starting point. Therefore, some steps might be different for another Bluetooth LE SDK example.

4.1 Importing the OTAP Bluetooth LE service and framework software into the wireless UART

To integrate the OTAP client service in your application, you must import additional software that is not included in other SDK examples by default. Therefore, the first step is to compare files in your project (Wireless UART) and the OTAP client SDK project. This step enables you to locate the files that you must merge in your project to support the OTAP Bluetooth LE service in your application.

Figure 4 shows a comparison between the Wireless UART (left) and the OTAP client (right). Files and folders highlighted in red are part of the OTAP client software, but not in the Wireless UART. Consequently, we must incorporate these files in our Wireless UART example to add the OTAP feature in this project. If you are interested in adding OTAP to other Bluetooth LE SDK projects or in your custom Bluetooth LE project, you must look for the missing files and incorporate them following the same methodology described in this example.

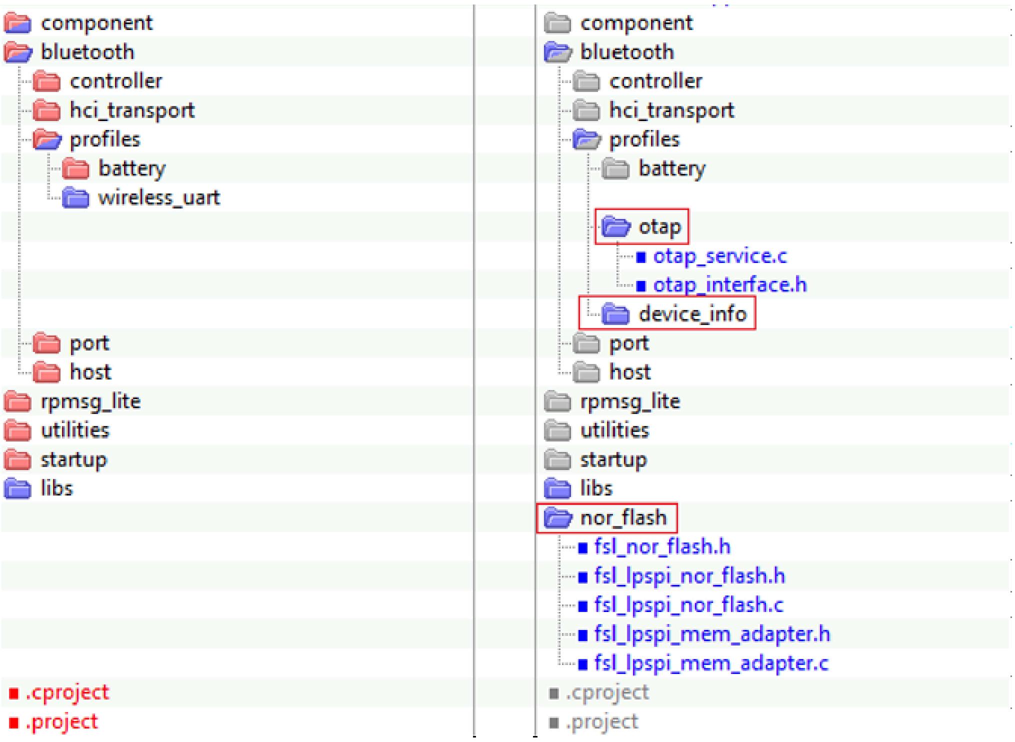
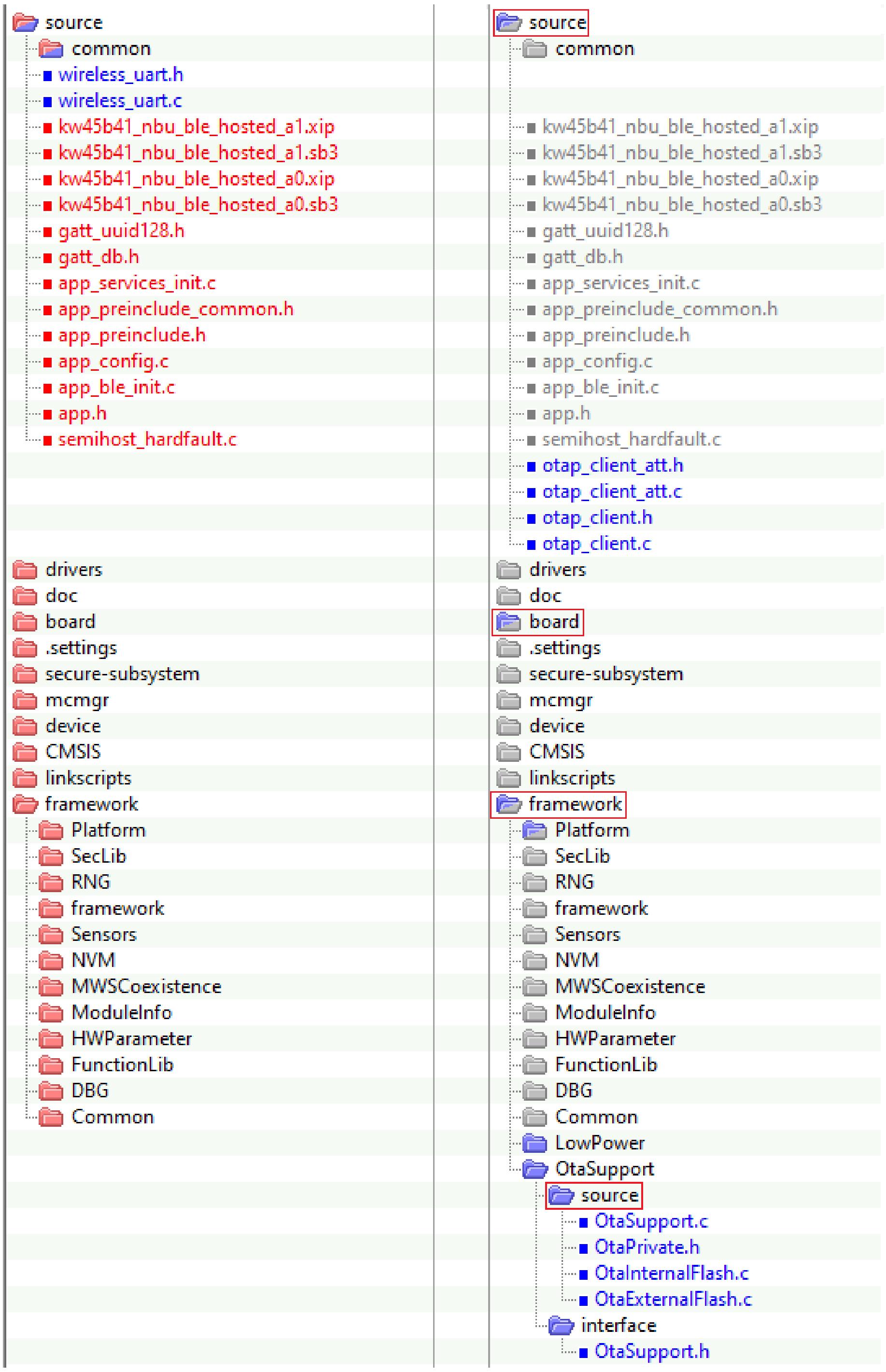


Figure 4. A comparison of files in Wireless UART (left) and the OTAP client SDK project (right)



```
source
    common
    wireless_uart.h
    wireless_uart.c
    kw45b41_nbu_ble_hosted_a1.xip
    kw45b41_nbu_ble_hosted_a1.sb3
    kw45b41_nbu_ble_hosted_a0.xip
    kw45b41_nbu_ble_hosted_a0.sb3
    gatt_uuid128.h
    gatt_db.h
    app_services_init.c
    app_preinclude_common.h
    app_preinclude.h
    app_config.c
    app_ble_init.c
    app.h
    semihost_hardfault.c
drivers
doc
board
.settings
secure-subsystem
mcmgr
device
CMSIS
linkscripts
framework
    Platform
    SecLib
    RNG
    framework
    Sensors
    NVM
    MWSCoexistence
    ModuleInfo
    HWParameter
    FunctionLib
    DBG
    Common
```

```
source
    common
    kw45b41_nbu_ble_hosted_a1.xip
    kw45b41_nbu_ble_hosted_a1.sb3
    kw45b41_nbu_ble_hosted_a0.xip
    kw45b41_nbu_ble_hosted_a0.sb3
    gatt_uuid128.h
    gatt_db.h
    app_services_init.c
    app_preinclude_common.h
    app_preinclude.h
    app_config.c
    app_ble_init.c
    app.h
    semihost_hardfault.c
    otap_client_att.h
    otap_client_att.c
    otap_client.h
    otap_client.c
drivers
doc
board
.settings
secure-subsystem
mcmgr
device
CMSIS
linkscripts
framework
    Platform
    SecLib
    RNG
    framework
    Sensors
    NVM
    MWSCoexistence
    ModuleInfo
    HWParameter
    FunctionLib
    DBG
    Common
    LowPower
    OtaSupport
        source
            OtaSupport.c
            OtaPrivate.h
            OtaInternalFlash.c
            OtaExternalFlash.c
        interface
            OtaSupport.h
```

Figure 5. Files and folders highlighted in red are part of the OTAP client software, but not included in the Wireless UART

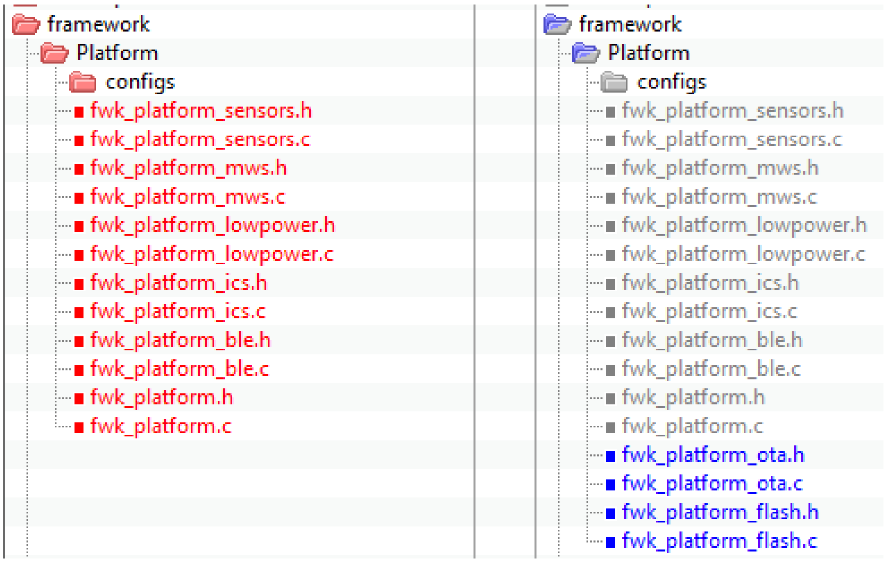


Figure 6. Files in `framework` directory

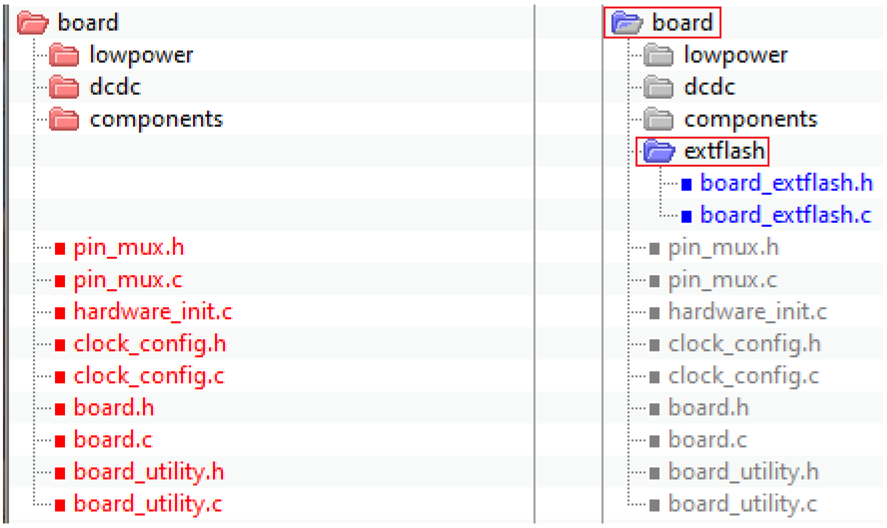


Figure 7. Files in `board` directory

The folders and files that are in OTAP but not in Wireless UART, must be imported in your Wireless UART project. For instance, the following must be imported:

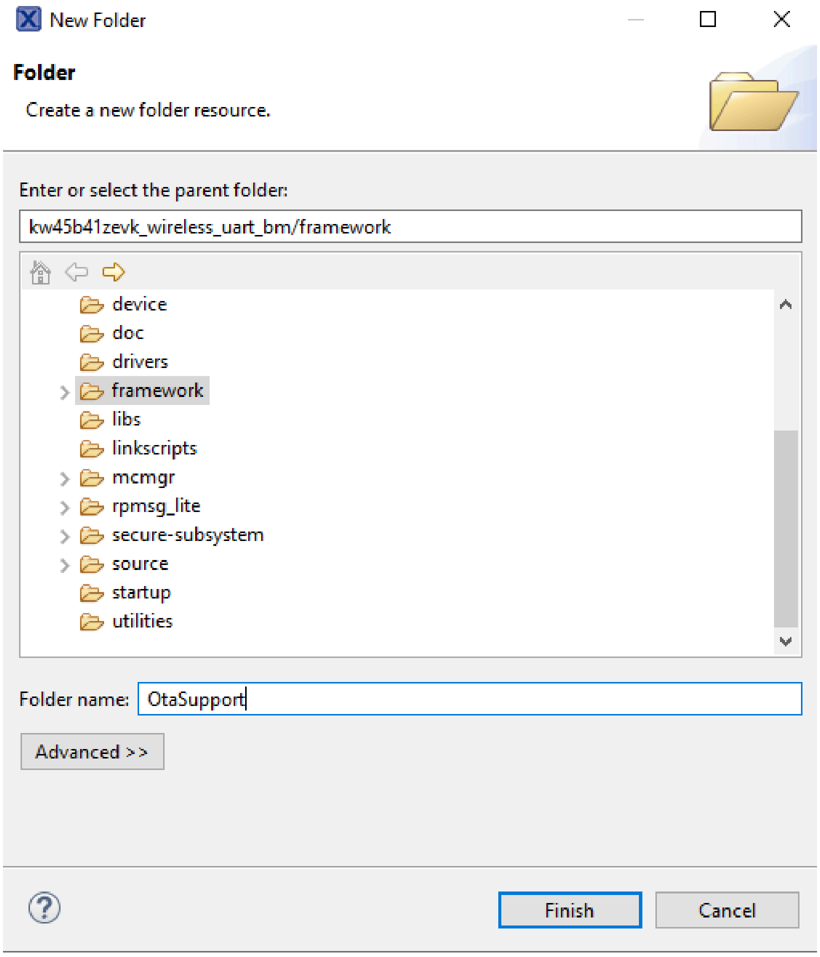
- **bluetooth -> profiles -> otap**
- **bluetooth -> profiles -> device_info**
- **framework ->OtaSupport**
- **framework ->Platform->configs**
- **source -> common -> otap_client**
- **board -> extflash**
- **nor_flash**

These files are displayed in Figure 4, Figure 5, Figure 6, and Figure 7.

To include these folders and source files in your project, perform the following steps.

1. In your workspace, expand the **bluetooth** and **framework** folders. Select the folder needed for updates and click the right mouse button.

2. Select **New -> Folder**. The **Folder** window appears to provide the same name as the missing folder in the source directory, as shown in Figure 8.



New Folder

Folder

Create a new folder resource.

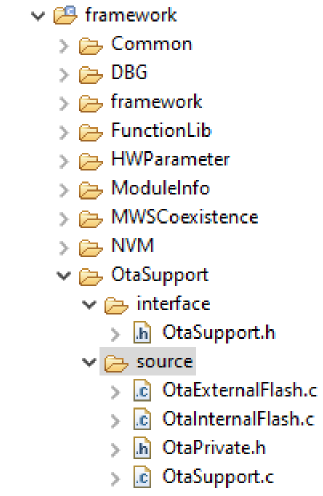
Enter or select the parent folder:

kw45b41zevk_wireless_uart_bm/framework

Folder name: OtaSupport

Figure 8. New Folder Window

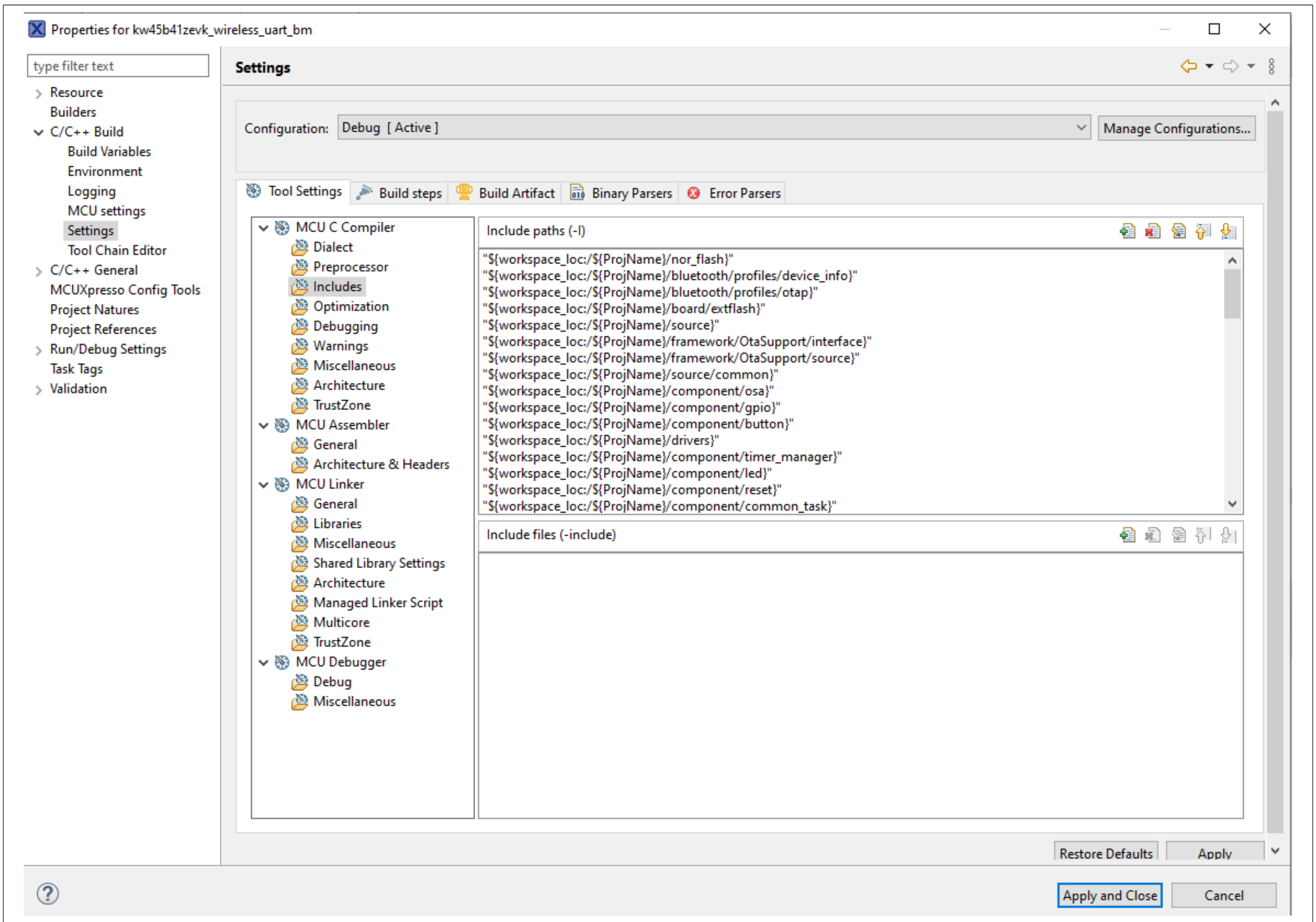
3. Repeat Step 2 for the left folders.
4. Copy the files inside all the recently created folders from the OTAP client and save it into your project. Ensure that all the files are in the same folder from the Wireless UART side. For this example, these files are listed in Figure 9.



- framework
  - Common
  - DBG
  - framework
  - FunctionLib
  - HWParameter
  - ModuleInfo
  - MWSCoexistence
  - NVM
  - OtaSupport
    - interface
      - OtaSupport.h
    - source
      - OtaExternalFlash.c
      - OtaInternalFlash.c
      - OtaPrivate.h
      - OtaSupport.c

Figure 9. Files in the Wireless UART project

5. Ensure that the following files are in the respective folders:
   - `otap_interface.h` and `otap_service.c` in the **bluetooth -> profiles -> otap** folder.
   - `OtaSupport.h` in the **framework -> OtaSupport -> Interface** folder.
   - `OtaSupport.c` in the **framework -> OtaSupport -> Source** folder.
   - `otap_client.h` and `otap_client.c` in the **Source** folder.
   - `fsl_lspi_mem_adapter.c`, `fsl_lspi_mem_adapter.h`, `fsl_lspi_nor_flash.h`, `fsl_lspi_nor_flash.c` and `fsl_nor_flash.h` in the **nor_flash** folder.
6. Navigate to **Project -> Properties** in **MCUXpresso IDE**. Go to **C/C++ Build -> Settings -> Tool Settings -> MCU C Compiler -> Includes**. Click the icon next to the **Include paths** textbox, as shown in Figure 10. In the new window that appears, click the **Workspace** button.



```
"${workspace_loc:/${ProjName}/nor_flash}"
"${workspace_loc:/${ProjName}/bluetooth/profiles/device_info}"
"${workspace_loc:/${ProjName}/bluetooth/profiles/otap}"
"${workspace_loc:/${ProjName}/board/extflash}"
"${workspace_loc:/${ProjName}/source}"
"${workspace_loc:/${ProjName}/framework/OtaSupport/interface}"
"${workspace_loc:/${ProjName}/framework/OtaSupport/source}"
"${workspace_loc:/${ProjName}/source/common}"
"${workspace_loc:/${ProjName}/component/osa}"
"${workspace_loc:/${ProjName}/component/gpio}"
"${workspace_loc:/${ProjName}/component/button}"
"${workspace_loc:/${ProjName}/drivers}"
"${workspace_loc:/${ProjName}/component/timer_manager}"
"${workspace_loc:/${ProjName}/component/led}"
"${workspace_loc:/${ProjName}/component/reset}"
"${workspace_loc:/${ProjName}/component/common_task}"
```

Figure 10. Include paths Window

7. Deploy your directory tree in the folder selection window.
8. Select the following folders and click **OK** to save the changes.
   - **Bluetooth -> profiles -> otap**
   - **board -> extflash**
   - **nor_flash**
   - **Framework -> OtaSupport -> interface**
   - **Framework -> OtaSupport -> source**

   Ensure that these paths were imported onto the **Include Paths** view.

4.2 Main modifications in the source files

The previous sections describe how to include OTAP client folders and files in your custom Wireless UART project. Once this is done, you should inspect the differences between the source files of the OTAP client and the Bluetooth LE application. The code should be modified to integrate the OTAP Service. The following sections explain the main aspects that should be taken care.

4.2.1 app_preinclude.h

`gOtaClientAtt_d` `1`: it sets the ATT transference method for OTA updates. It must be set to `1` for the purpose listed in this document.

```
#ifndef _APP_PREINCLUDE_H_
```

```
#define _APP_PREINCLUDE_H_

/*! Applicatiom specific configuration file only
* Board specific configuration shall be added to board.h file directly such as:
* -Number of buttons on the board,
* -Number of LEDs,
* -etc...
*/

/*!*****************************************************************
     Board  configuration
*****************************************************************/
/*Number of buttons required by the application: */
#define gAppButtonCnt_c         1


/* Number of LEDs required by the application:
  * Beware, on KW45 PB0 is tied to SPI NOR Flash chip select and monochrome LED
  *If external Flash is to be used, avoid using monochrome LED
*/

#define gAppLedCnt_c            2
#define gAppRequireRgbLed_c     1

/*!Enable Debug Consile (PRINTF) */
#define gDebugConsoleEnable_d

/*! ***************************************************************
     App  configuration
*****************************************************************/
#define gOtapClientAtt_d  1
```

In this file, you should add the OTA characteristics as shown in the below codeblock:

```
/*! Define as 1 to place OTA storage in external flash */
#define gAppOtaExternalStorage_c        (1U)

/*! Define the offset where to place the OTA partition storage in external flash
 */
#define gAppOtaStoragePartitionOffset_c   (0U)

/*! Define to 1 to post OTA transactions to a queue. The queue will be processed
 in the idle task.
 * This avoids blocking the system for too long in critical tasks
 * as the write to flash operations will be done during idle period. */
#define gAppOtaASyncFlashTransactions_c    1

/* Define max of consecutive OTA transactions processed during idle task (if
 gAppOtaASyncFlashTransactions_c is enabled)
 * This can be tuned accordingly to an acceptable block time in idle. More
 transactions mean
 * higher block time in idle (if the queue reaches this number of transactions)
 */
#define gAppOtaMaxConsecutiveTransactions_c (4U)

/*! If gAppOtaASyncFlashTransactions_c is enabled the queue of pending
 transactions
 *  must be dimensioned to store at least 4 operations to avoid buffer shortage
 */
```

```
#define gAppOtaNumberOfTransactions_c       (4U)
```

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 (WU – OTAP) contains. This information is used by the Bluetooth LE scanner, to filter out the advertiser devices that do not contain the services required. Therefore, 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 shown 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
};
```

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 (WU-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. The table below 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)
```

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 Wireless UART SDK project because the Bluetooth Special Interest Group (SIG) adopts the wireless UART and the battery services. However, the OTAP service and its characteristics need to be specified by the developer as a 128 – UUID. The example code below shows how to implement the 128 – UUID assignation for the OTAP service.

```
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)
```

4.2.4 wireless_uart.c

The `wireless_uart.c` is the main source file at the application level. This file contains information for managing all the procedures that the device performs, before, during, and after creating 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`.
   Note: This step depends on your software since it might share different files than this example.

```
# include "OtaSupport.h"
# include "device_info_interface.h"
# include "otap_interface.h"
# include "otap_client.h"
# include "fwk_platform_ble.h"
```

2. Add the function prototypes and global variables that are used by the OTAP client software. As mentioned in the last step, this might depend on your application. For this example, you can skip merging the `appTimerId` variable in your wireless UART project, since this is used in the OTAP client to create an instance of a timer that will not be implemented in this example.
   a. In the Private type declarations you need to add the code below:

```
typedef enum
{
#if (defined(gAppUseBonding_d) && (gAppUseBonding_d == 1U)) &&\
    (defined(gAppUsePrivacy_d) && (gAppUsePrivacy_d == 1U)) &&\
    (defined(gBleEnableControllerPrivacy_d) &&
 (gBleEnableControllerPrivacy_d > 0))
    filterAcceptListAdvState_c,
#endif /* gAppUseBonding_d && gAppUsePrivacy_d &&
 gBleEnableControllerPrivacy_d */
fastAdvState_c,
slowAdvState_c
}advType_t;
typedef struct advState_tag
{
bool_t advOn;
advType_t advType;
} advState_t;
```

   b. In the private memory declarations, add the below code:

```
static deviceId_t  mPeerDeviceId = gInvalidDeviceId_c;

static advState_t mAdvState;
static TIMER_MANAGER_HANDLE_DEFINE(mAdvTimerId);
static disConfig_t disServiceConfig = {(uint16_t)service_device_info};
static TIMER_MANAGER_HANDLE_DEFINE(mBatteryMeasurementTimerId);
```

3. Locate the `BluetoothLEHost_Initialized` function. The function configures the Bluetooth Low Energy stack after initialization. The function is used for configuring advertising, filter accepts list, services, and similar tasks. Modify the beginning of the function, as shown below:

```
mAdvState.advOn = FALSE;

    /* Start services */
    basServiceConfig.batteryLevel = SENSORS_GetBatteryLevel();
    (void)Bas_Start(&basServiceConfig);
    (void)Dis_Start(&disServiceConfig);

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

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 [**case** *gConnEvtConnected_c*:]. Include the `OtapCS_Subscribe` and `OtapClient_HandleConnectionEvent` functions. The below code block shows the implementation (see the commands highlighted in bold letters).

```
/* Subscribe client */
 (void)Wus_Subscribe(peerDeviceId);
 (void)Bas_Subscribe(&mBasServiceConfig, peerDeviceId);
 (void)OtapCS_Subscribe(peerDeviceId);


/* UI */
 LedStopFlashingAllLeds();
 OtapClient_HandleConnectionEvent(peerDeviceId);
```

   b. Go to the disconnection case [**case** *gConnEvtDisconnected_c*:]. Include the `OtapCS_Unsubscribe` and `OtapClient_HandleDisconnectionEvent` functions and the Bonding condition include the `OtapClient_HandleEncryptionChangedEvent` function. The below code block shows the implementation (see the commands highlighted in bold letters).

```
/* Unsubscribe client */
  (void)Wus_Unsubscribe();
  (void)Bas_Unsubscribe(&mBasServiceConfig, peerDeviceId);
 (void)OtapCS_Unsubscribe();
  (void)TM_Stop((timer_handle_t)mBatteryMeasurementTimerId);
  OtapClient_HandleDisconnectionEvent(peerDeviceId);

/* Restored custom connection information. Encrypt link */
 (void)Gap_EncryptLink(peerDeviceId);
 OtapClient_HandleEncryptionChangedEvent(peerDeviceId);
```

5. Locate the `BleApp_GattServerCallback`. This function manages all the incoming communications from the client devices. Add the GATT events that the OTAP client software must handle. These are `gEvtAttributeWritten_c`, `gEvtMtuChanged`, `gEvtCharacteristicCccdWritten_c, gEvtAttributeWrittenWithoutResponse_c, gEvtHandleValueConfirmation_c`, and `gEvtError.` Your Bluetooth LE project might share some common GATT events. In such a case, you should add a conditional structure per each attribute handle. Focus on the `gEvtAttributeWritten_c` case and

observe the conditional structure that was included for the wireless UART control point and the OTAP control point handling.

```
switch (pServerEvent->eventType)
{
 case gEvtAttributeWrittenWithoutResponse_c:
    {
    if (pServerEvent->eventData.attributeWrittenEvent.handle ==
 (uint16_t)value_uart_stream)
    {
    BleApp_ReceivedUartStream(deviceId, pServerEvent-
>eventData.attributeWrittenEvent.aValue,
    pServerEvent->eventData.attributeWrittenEvent.cValueLength);
    OtapClient_AttributeWrittenWithoutResponse (deviceId,
    pServerEvent->eventData.attributeWrittenEvent.handle,
    pServerEvent->eventData.attributeWrittenEvent.cValueLength,
    pServerEvent->eventData.attributeWrittenEvent.aValue);
    }
break;
    }
 case gEvtMtuChanged_c:
    {
    /* update stream length with minimum of new MTU */
    (void)Gatt_GetMtu(deviceId, &tempMtu);
    tempMtu = gAttMaxWriteDataSize_d(tempMtu);
    mAppUartBufferSize = mAppUartBufferSize <= tempMtu ? mAppUartBufferSize :
 tempMtu;
    OtapClient_AttMtuChanged (deviceId,
    pServerEvent->eventData.mtuChangedEvent.newMtu);
    }
break;
 case gEvtCharacteristicCccdWritten_c:
    {
    OtapClient_CccdWritten (deviceId,
    pServerEvent->eventData.charCccdWrittenEvent.handle,
    pServerEvent->eventData.charCccdWrittenEvent.newCccd);
    }
break;
 case gEvtHandleValueConfirmation_c:
    {
    OtapClient_HandleValueConfirmation (deviceId);
    }
break;
 case gEvtAttributeWritten_c:
    {
    OtapClient_AttributeWritten (deviceId,
    pServerEvent->eventData.attributeWrittenEvent.handle,
    pServerEvent->eventData.attributeWrittenEvent.cValueLength,
    pServerEvent->eventData.attributeWrittenEvent.aValue);
    }
break;
 case gEvtError_c:
    {
    Testing the Wireless UART-OTAP demo

    uint8_t tempError = (uint8_t)pServerEvent->eventData.procedureError.error
 & 0xFFU;
    attErrorCode_t attError = (attErrorCode_t)tempError;
    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, NULL) &&
    TRUE == isBonded)
    #endif /* gAppUseBonding_d */
    {
    (void)Gap_SendPeripheralSecurityRequest(deviceId, &gPairingParameters);
    }
    #endif /* gAppUsePairing_d */
    }
    }
    default:
    {
    ; /* No action required */
    }
break;
    }
}
```

At this point, you have integrated the OTAP Client code into the Wireless UART project.

5 Testing the Wireless UART-OTAP demo

The test case example, designed to demonstrate the OTAP integration in Testing the Wireless UART-OTAP software, makes use of the listed software:

- OTAP Client SDK software, programmed on the KW45B41Z-EVK board.
- An SREC software update of the Wireless UART-OTAP example.
- An SREC software update of the Wireless UART SDK example. The following sections explain how to build the software required for the testing case proposed by this document.

5.1 Preparing the OTAP client SDK software

1. Connect the KW45B41Z-EVK board to the PC.
2. Open **MCUXpresso IDE**. In the Quickstart Panel view, click **Import SDK example(s)**. See Figure 11.

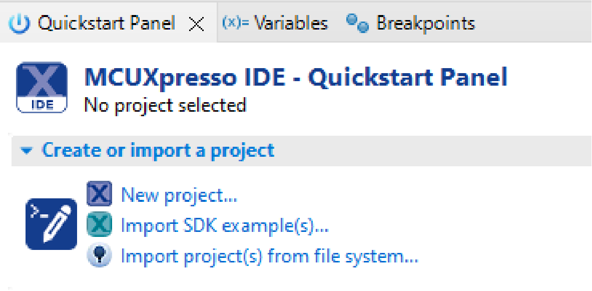
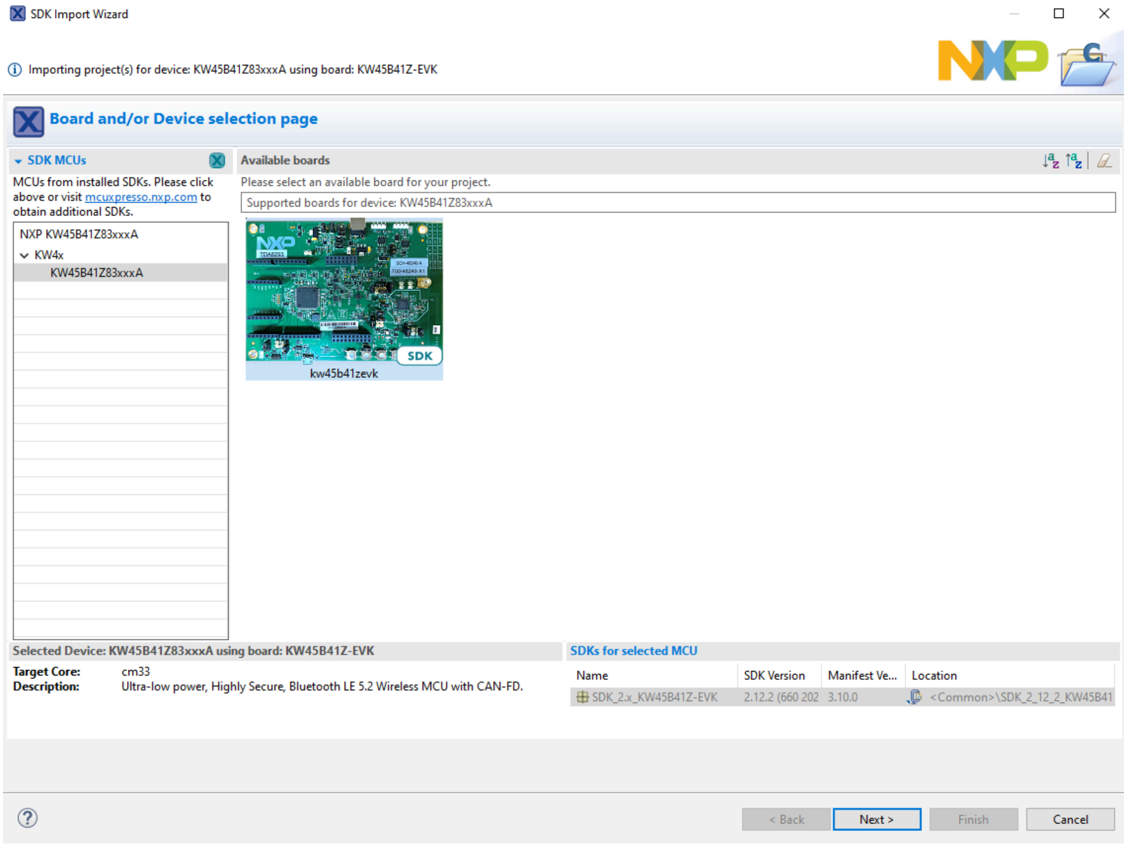


Figure 11. MCU Expresso IDE

3. Click the KW45B41Z-EVK icon to select the board.

See Figure 12.



Importing project(s) for device: KW45B41Z83xxxA using board: KW45B41Z-EVK

Board and/or Device selection page

MCUs from installed SDKs. Please click above or visit mcuxpresso.nxp.com to obtain additional SDKs.

Please select an available board for your project.

Supported boards for device: KW45B41Z83xxxA

Selected Device: KW45B41Z83xxxA using board: KW45B41Z-EVK

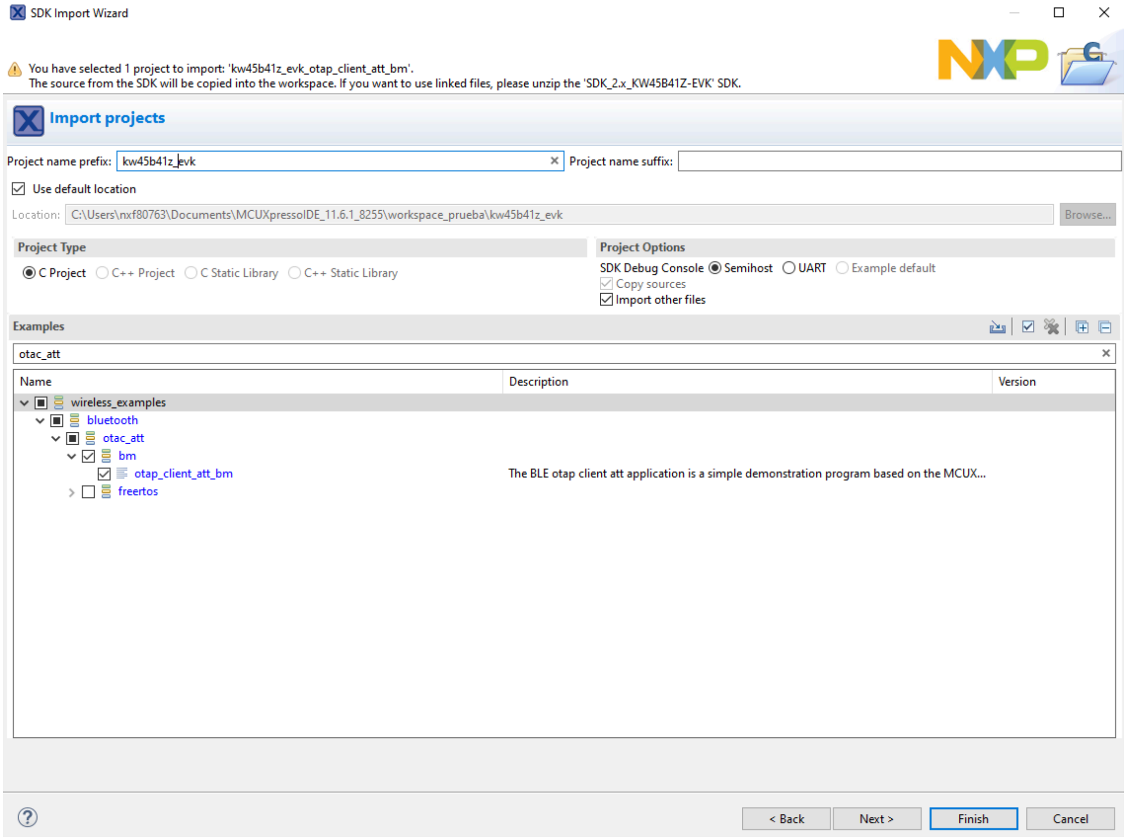
Target Core: cm33
Description: Ultra-low power, Highly Secure, Bluetooth LE 5.2 Wireless MCU with CAN-FD.

SDKs for selected MCU

Name	SDK Version	Manifest Ve...	Location
SDK_2.x_KW45B41Z-EVK	2.12.2 (660 202	3.10.0	<Common>\SDK_2_12_2_KW45B41

Figure 12. Selecting the Board and/or Device

4. In the **Examples** textbox, type `otac_att`. Select the suggested project by **wireless_examples -> framework -> otac_att -> bm**. See Figure 13



You have selected 1 project to import: 'kw45b41z_evk_otap_client_att_bm'.
The source from the SDK will be copied into the workspace. If you want to use linked files, please unzip the 'SDK_2.x_KW45B41Z-EVK' SDK.

Project name prefix: kw45b41z_evk

Project name suffix:

☑ Use default location

Location: C:\Users\nxf80763\Documents\MCUXpressoIDE_11.6.1_8255\workspace_prueba\kw45b41z_evk

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	Description	Version
wireless_examples		
bluetooth		
otac_att		
☑ bm		
☑ otap_client_att_bm	The BLE otap client att application is a simple demonstration program based on the MCUX...	
□ freertos		

Figure 13. Selecting the project

Click **Finish**.

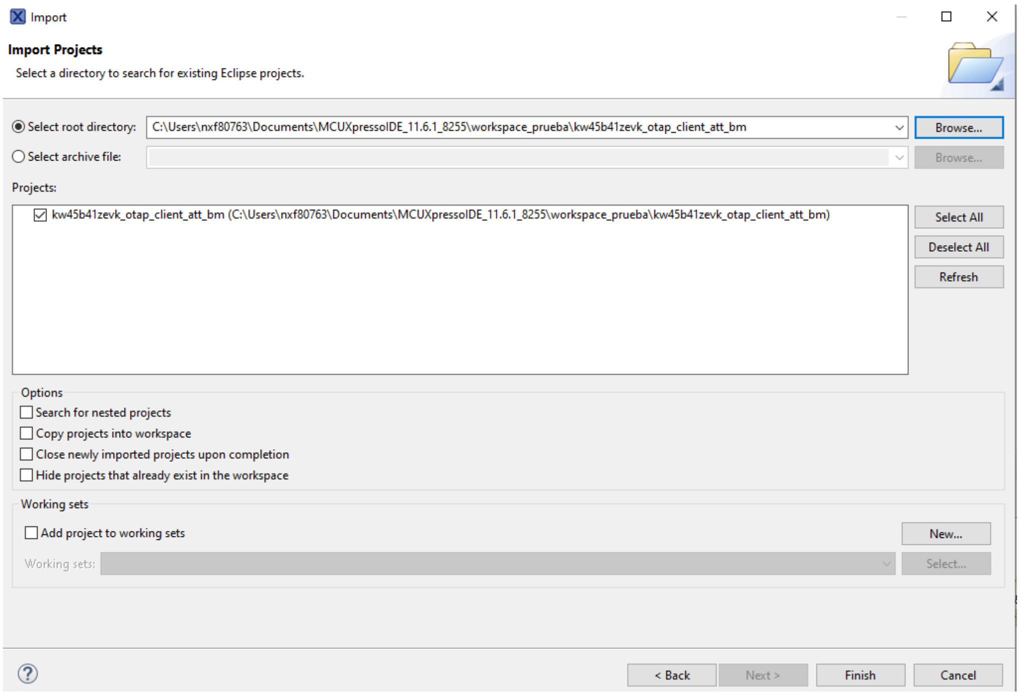
5. Flash the `otac_att` project on the KW45B41Z-EVK board.
6. Set the storage configurations on the OTAP Client software:
   - Open the `app_preinclude.h` file located in the source folder of the project.
   - To configure the software for external flash storage method, set the `gAppOtaExternalStorage` to `1`.
7. Clean and build the project. Flash the OTAP Client project on the KW45B41Z-EVK board.

Now, you have completed 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 a Wireless UART-OTAP S-record image to update the software

This section outlines steps for creating a Wireless UART-OTAP S-record image to update the software.

1. Install the Wireless UART-OTAP demo provided with this document in your MCUXpresso IDE. You can import the project from your installation path to the MCUXpresso workspace as shown in Figure 14.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\Users\nxf80763\Documents\MCUXpressoIDE_11.6.1_8255\workspace_prueba\kw45b41zevk_otap_client_att_bm

○ Select archive file:

Projects:

☑ kw45b41zevk_otap_client_att_bm (C:\Users\nxf80763\Documents\MCUXpressoIDE_11.6.1_8255\workspace_prueba\kw45b41zevk_otap_client_att_bm)

Options

☐ Search for nested projects

☐ Copy projects into workspace

☐ Close newly imported projects upon completion

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

Figure 14. Importing Wireless UART-OTAP project to the MCUXpresso workspace

2. Open the `end_text.ldt` linker script located at the `linkscripts` folder in the workspace. Locate the section placement and remove the `FILL` and `BYTE` statements. (You need to delete the text in bold shown in the code block below). This step is needed only to build the SREC image file to reprogram the device.

```
 /* Remove the FILL and BYTE statements */
 /* to build the SREC image file to reprogram the device */

  _etext= .;

 .NVM_region :
{
   FILL(0xFFFFFFFF)
   . = ORIGIN(NVM_region) + LENGTH(NVM_region) - 1;
   BYTE(0xFF);
 } > NVM_region
```

3. Clean and build the project.
4. Deploy the **Binaries** icon in the workspace.
5. Click the right mouse button on the `.axf` file and select **Binary Utilities -> Create S-Record**. The S-Record file is saved at the Debug folder in the workspace with `.s19` extension. This is shown in Figure 15.

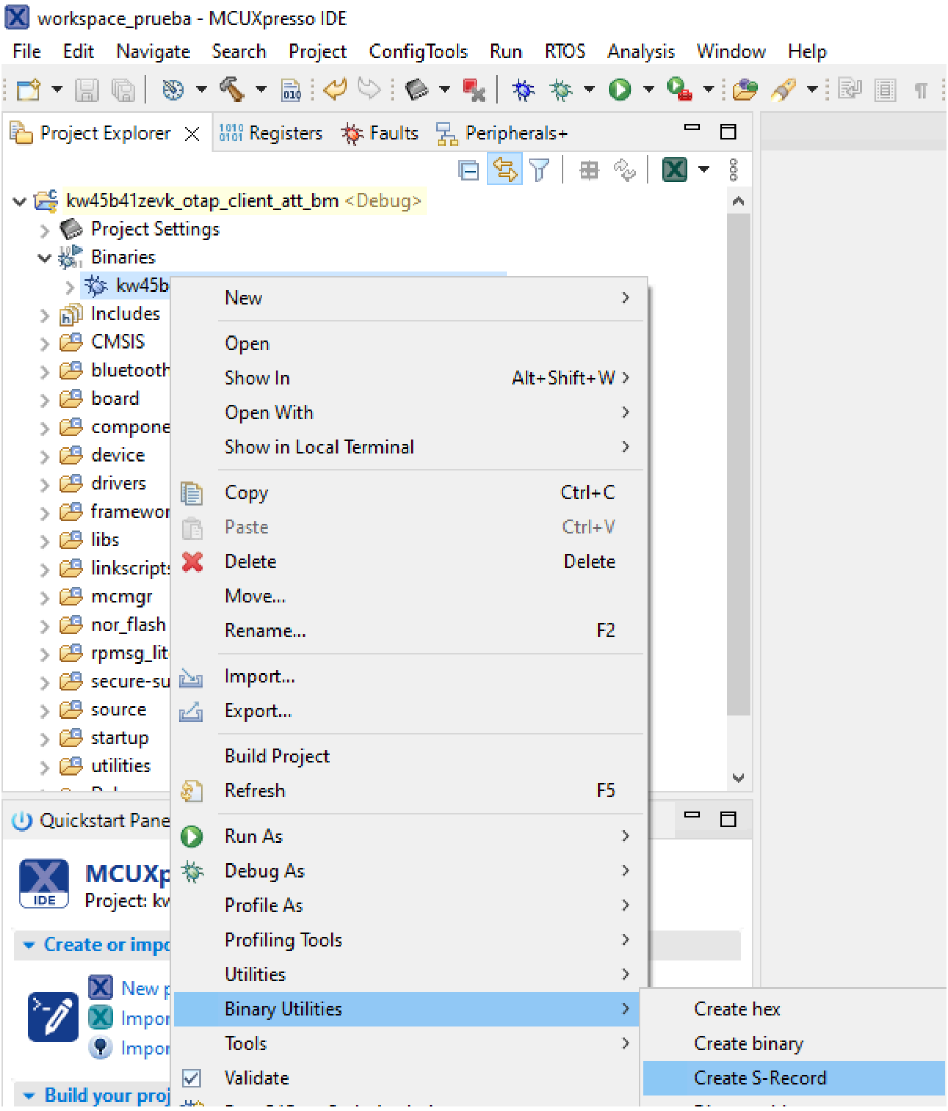


Figure 15. Creating an S-Record

6. Save this file to a known location on your smartphone.

5.3 Testing the wireless UART-OTAP software

This section describes the steps for testing wireless UART-OTAP software using the NXP IoT Toolbox app.

1. Open the IoT Toolbox App and select the **OTAP** demo. Click the **SCAN** button to start scanning for a suitable advertiser. See Figure 16.

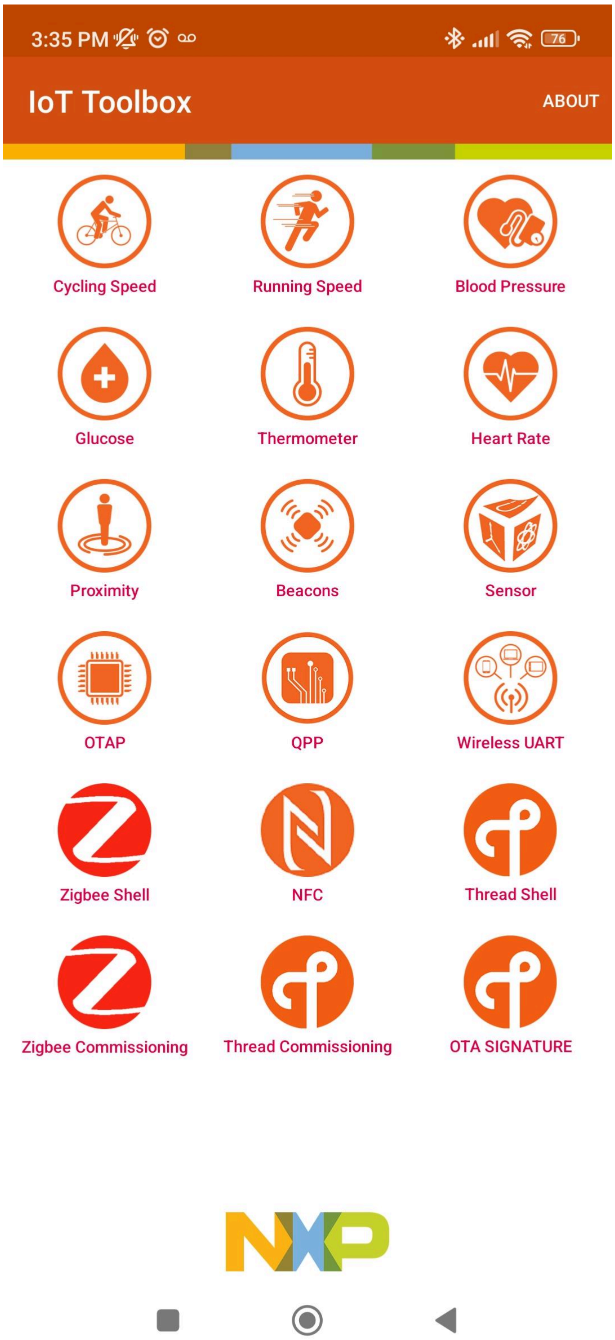


Figure 16. Testing the application using NXP IoT Toolbox App

2. To start advertising, press the **ADV** button, **SW3**, and then **SW2** button, on the KW45B41Z-EVK board.
3. Create a connection with the **NXP_WU** device. Then, the OTAP interface would be displayed on your smartphone. See Figure 17.

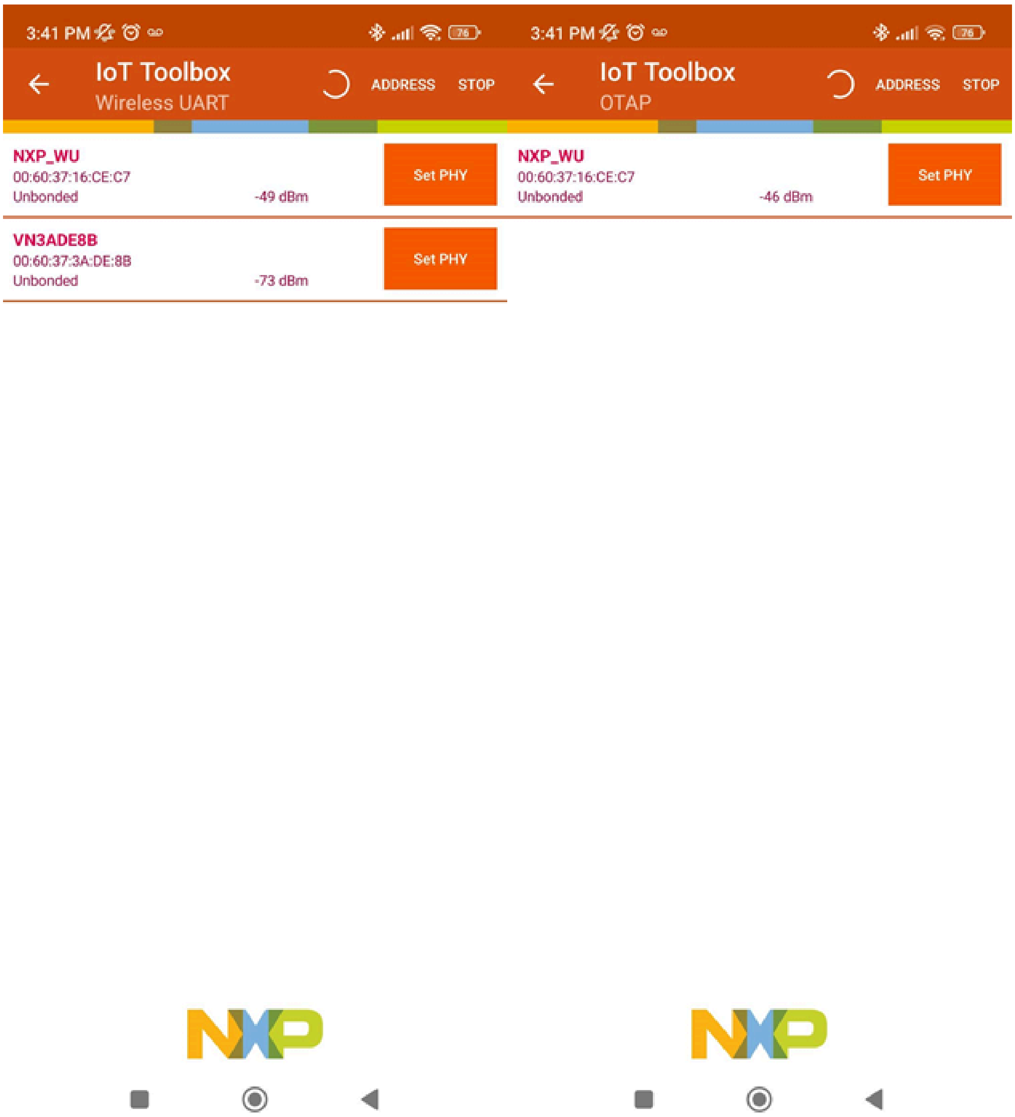
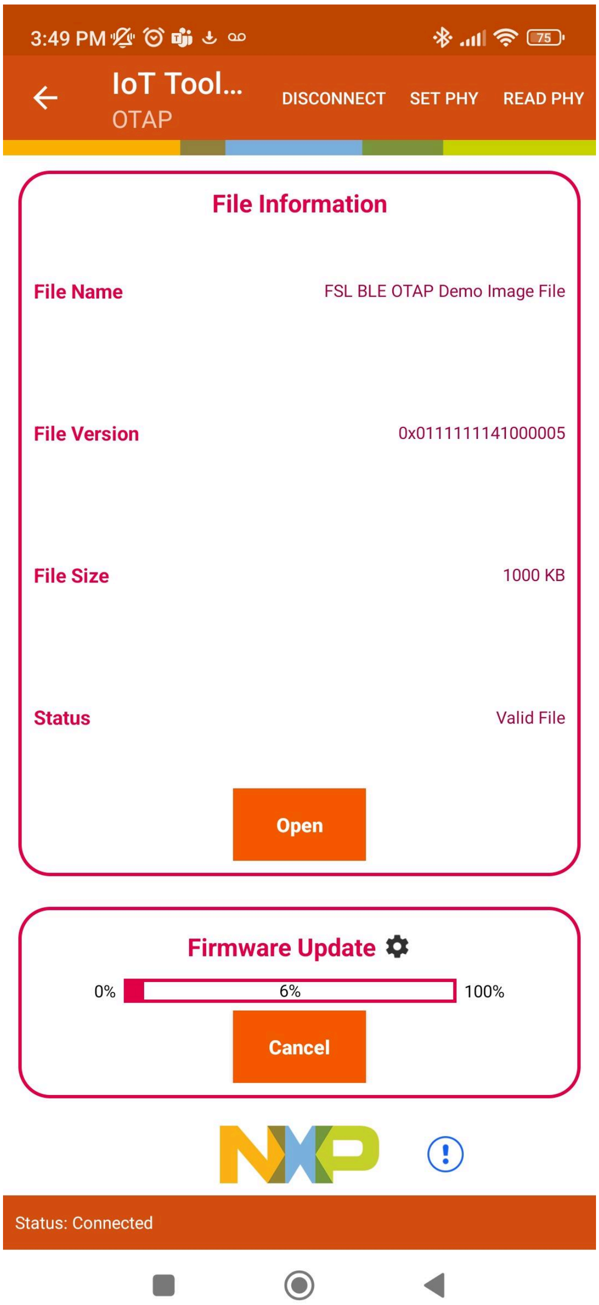


Figure 17. OTAP interface displayed on smartphone

4. Click the **Open** button and search for the Wireless UART-OTAP SREC file.
5. Click **Upload** to start the transfer. Wait until the confirmation message is displayed. See Figure 18.



File Information

File Name: FSL BLE OTAP Demo Image File

File Version: 0x0111111141000005

File Size: 1000 KB

Status: Valid File

Status: Connected

Figure 18. Uploading the Wireless UART-OTAP SREC file

6. Wait for a few seconds until the OTAP bootloader finishes programming the new image. After this interval, the wireless UART application starts automatically, with the RGB LED blinking.
7. Press the **ADV** button, **SW3**, and then **SW2** buttons, on the KW45B41Z-EVK board to start advertising. Verify that the device can be detected by both, Wireless UART and OTAP applications of the IoT Toolbox. The device is named as **NXP_WU**. You can create a connection and interact with both demos.
8. Connect the WU-OTAP device with the OTAP smartphone application. Update the software using the Wireless UART SREC file.
9. Confirm that the device has been updated to a simple Wireless UART, making use of the Wireless UART-OTAP demo. Press the **ADV** button, **SW3**, and then **SW2** buttons on the KW45B41Z-EVK board to start advertising. Now the device’s name is **NXP_WU**.

Connect the device with the Wireless UART IoT Toolbox app and verify that it works as expected.

6 Revision history

Table 1 summarizes revisions to this document.

Table 1. Revision history

Revision number	Date	Substantive changes
0	1 March 2023	Initial release

7 Legal information

7.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

7.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

eIQ — is a trademark of NXP B.V.

i.MX — is a trademark of NXP B.V.

Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

For more information, please visit: http://www.nxp.com

Date of release: 1 March 2023
Document identifier: AN13855