AN15123

Integrating the OTAP Client Service into a MCX W72 Bluetooth LE Peripheral Device

Rev. 1.0 — 3 August 2026 Application note

Document information

Information	Content
Keywords	AN15123, OTAP, Bluetooth LE, and MCX W72
Abstract	This application note provides the steps to integrate the OTAP client service into a Bluetooth LE peripheral device to upgrade software on an MCU.

1 Introduction

This application note outlines the use of the NXP Over the Air Programming (OTAP) custom Bluetooth Low Energy (Bluetooth LE) service to upgrade software on a Microcontroller Unit (MCU) without using physical cables.

The OTAP custom Bluetooth LE service of NXP provides the developers to upgrade the software on the MCU wirelessly. It removes the need for a physical connection between the device to be upgraded and the device that contains the new software.

The demo applications implement a typical scenario where a PC sends a new image through a serial interface to a Bluetooth LE OTAP. The OTAP then transmits the image over the air to an OTAP client, which serves as the upgrade target.

To maximize the benefits of the OTAP service, developers integrate it into the Bluetooth LE application. This approach allows for repeated reprogramming of the device as required.

2 OTAP client software

The OTAP memory management during the update process describes the implementation of the OTAP client software included in the Software Development Kit (SDK) package for the FRDM-MCXW72. The advantages of OTAP service integration highlight the importance of incorporating OTAP client software into your application and the expected outcomes.

2.1 OTAP memory management during the update process

This section describes how the OTAP update mechanism works on the MCX W72 device. To know how the flow logically progresses, perform the steps below:

1. Flash memory layout: The MCX W72 device provides 2 MB of on-chip Program Flash, mapped to the address range 0x00000000 to 0x001FFFFF. To support the OTA (Over-the-Air) update mechanism, the flash memory is divided into different regions. The active application image is stored in a dedicated application region that begins at 0x00040000 and spans 0x000E0000 bytes (896 kB). Within this region, the first 0x400 bytes are reserved for the image header, followed by the interrupt vector table and the application code. The remaining Flash area is reserved for storing a candidate image during the OTA update process.
2. Image generation: When generating a new image file for the OTAP client device, the developer specifies that the code gets stored with an offset using the linker script. The new application must include the bootloader flags at the corresponding address to work properly.
3. Image transfer: At the connection state, the OTAP server sends the image packets, known as chunks, to the OTAP client through Bluetooth LE. The OTAP client stores these chunks in the external Serial Peripheral Interface (SPI) flash or in the on-chip FlexNVM region. The OTAP client software allows the selection of the code destination.
4. Bootloader activation and update execution: After the image transfer completes and all the chunks reach the OTAP client from the OTAP server, the OTAP client software writes metadata. This metadata includes details such as the image source (external flash or FlexNVM) into 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 required to program the device. It also triggers a command to reprogram the MCU with the new application.
5. Programming and executing the new image: After the update process is completed, the new application becomes the active image and starts executing from the application region beginning at address 0x00040000. The device then resumes normal operation using the updated firmware. If the new application does not include OTAP functionality, future wireless firmware updates are no longer available until the device is reprogrammed through another supported interface.

2.2 Advantages of the OTAP service integration

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

Suppose that an OTAP client device runs the OTAP client software and requests an update, such as a Wireless UART. The image that the OTAP server sends to the OTAP client must be the Wireless UART. After the reprogramming process, the device becomes a Wireless UART. The Wireless UART does not have the capabilities to communicate with the OTAP server or request for another update.

If the Wireless UART image includes the OTAP client service as well, the device retains the possibility to request another software update, such as a modified baud rate with OTAP service.

As the baud rate software includes the OTAP client, the device can request another software update from the OTAP server. This approach allows the developer to continue upgrading the software as many times as required. In other words, to enable future software upgrades on the OTAP client device, the application sent over the air must include OTAP service support.

3 Prerequisites

This document accompanies a functional demo of the OTAP service integration. The example is based on the Wireless UART project, available in the FRDM-MCXW72 SDK package, and is developed on the MCUXpresso Integrated Development Environment (IDE) platform. The following components are required to complete the implementation of the Wireless UART-OTAP integration demo:

- MCUXpresso for VS Code
- MCX W72 SDK v.25_06_00
- FRDM-MCXW72 board
- Smartphone with IoT Toolbox NXP app (available for Android and iOS)

Note: *Burn the fuses for the Secure Boot 3 Key Derivation Key (SB3KDK) and Root of Trust Key Table Hash (RoTKTH) keys to use the OTAP feature on the FRDM-MCXW72 board. For more information, refer to MCX W72 Secure Boot using SEC Tool (AN14613).*

3.1 Downloading the MCX W72 Package

MCUXpresso for VS Code provides an optimized embedded development environment for code editing and development. The extension enables NXP developers to use one of the most popular embedded editor tools. The extension also provides an easy and fast way to create, build, and debug applications based on the MCUXpresso SDK.

To install, follow the steps below:

1. Download VS Code from the Microsoft store or VS Code webpage: Download Visual Studio Code - Mac, Linux, Windows.
2. Go to the MCUXpresso for VS Code Wiki and download the MCUXpresso Installer: Dependency Installation – MCUXpresso for VS Code Wiki.
3. Launch the *MCUXpresso Installer* and install the following components:
   - MCUXpresso SDK Developer
   - Matter Developer
   - Arm GNU Toolchain
   - Standalone Toolchain Add-ons
   - Linkserver
   - PEmicro

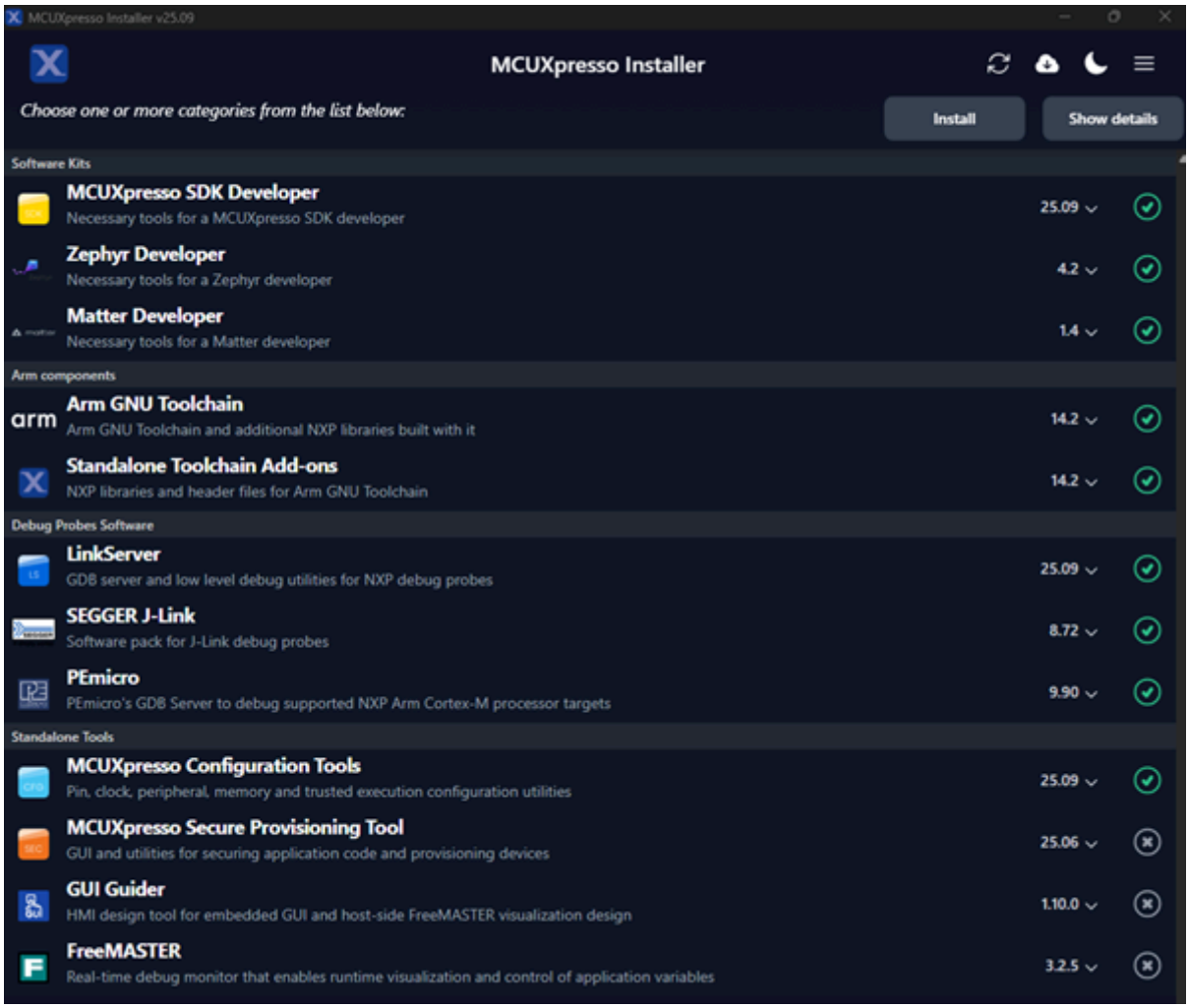

Figure 1. MCUXpresso Installer with required components selected for installation

4. After the installation is complete, verify that all selected components are successfully installed and available in the MCUXpresso Installer.

3.2 Installing the MCX W72 SDK

Each MCU has its own SDK that includes driver, examples, middleware, docs, and other components. To get and build the demo, let's install the SDK into VS Code.

To install the NXP GitHub SDK, follow the steps below:

1. After installing MCUXpresso for VS Code is installed, open VS Code.
2. Go to MCUXpresso for VS Code extension from the Activity Bar on the left side of the window.
3. Look for **INSTALLED REPOSITORIES** option and click '+' (**Import Local/Remote Repository**). For detailed instructions, refer to the GitHub wiki page: nxp-mcuxpresso/vscode-for-mcux.

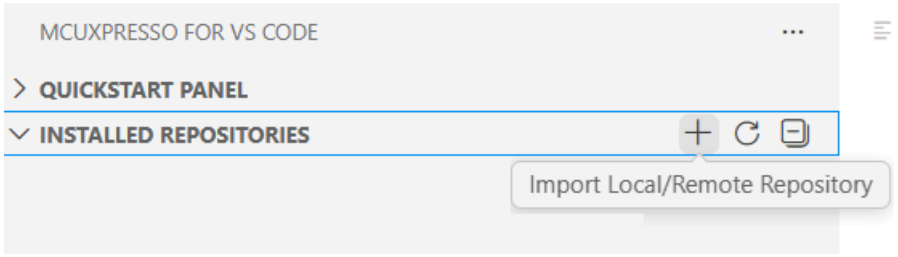

Figure 2. Accessing the Import Repository option

4. In the **Import Repository** window, select the **REMOTE** tab.

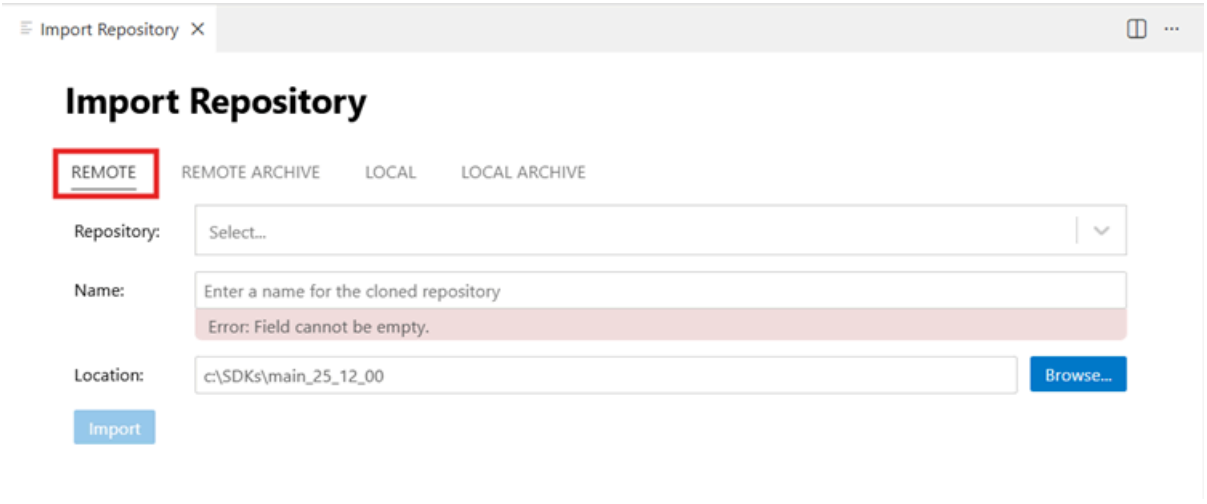

Figure 3. Selecting the Remote Repository source

5. From the **Repository** drop-down list, select **MCUXpresso SDK** to download the GitHub SDK. Then:
   - Select the desired **Revision**:
     - **main** for the latest available version, or
     - a specific revision, if required.
   - Optionally modify the repository **Name** and **Location**.

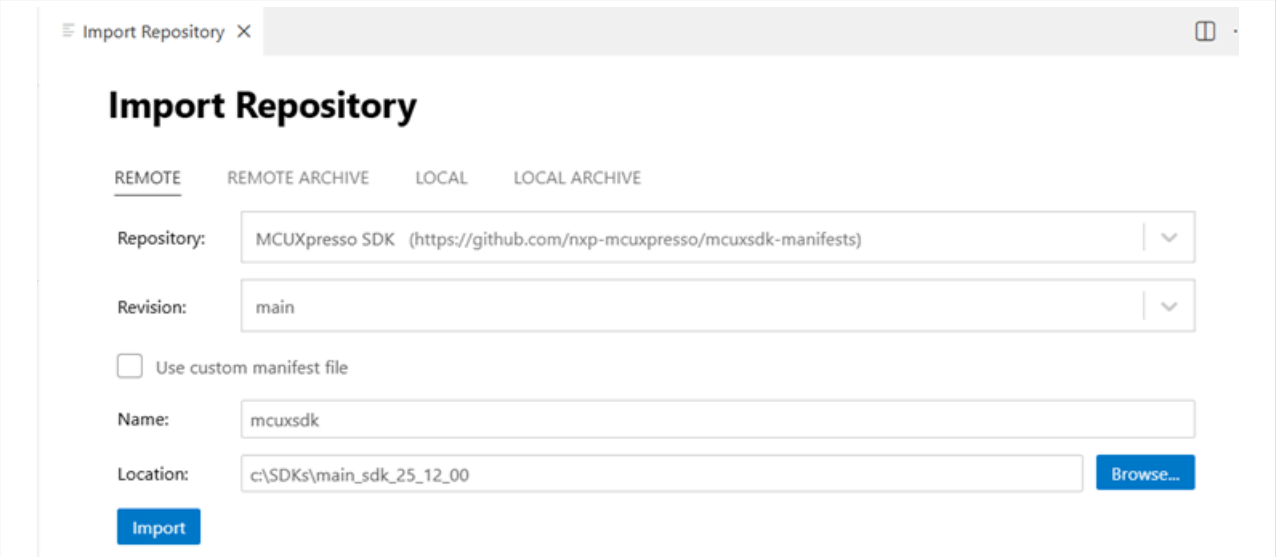

Figure 4. Configuring MCUXpresso SDK Repository settings before import

6. To start downloading and importing the SDK, click Import.
7. Wait for the import process to complete. The SDK then appears under Installed Repositories and is available for project creation and development.

4 Customizing a Bluetooth LE demo to integrate the OTAP service

This section outlines the process of customizing a Bluetooth LE demo imported from the SDK to integrate the OTAP service. The guide uses the Wireless UART project as a starting point, so some steps can vary when using a different 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 an application, import other software that is not included in other SDK examples by default. To locate which files must merge to support this service in the application, begin by comparing the project with the OTAP client SDK project.

To include these folders and source files in your project, perform the steps below:

1. Open prj.conf.
2. Add the required configuration macros for your driver, as shown below:

```
CONFIG_MCUX_COMPONENT_component.messaging=y
CONFIG_MCUX_COMPONENT_component.panic=y
CONFIG_MCUX_COMPONENT_component.mem_manager_light=y
CONFIG_MCUX_COMPONENT_component.timer_manager=y
CONFIG_MCUX_COMPONENT_component.lists=y
CONFIG_MCUX_COMPONENT_component.log=y
CONFIG_MCUX_COMPONENT_component.osa=y
CONFIG_MCUX_PRJSEG_component.osa_interface.generated_config=n
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.function_lib=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.platform=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.platform.ble=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.module_info=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.nvm=y
```

```
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng=y
CONFIG_MCUX_PRJSEG_module.board.wireless.app_ble=y
CONFIG_MCUX_PRJSEG_module.board.wireless.linker_script_ble=y
CONFIG_MCUX_COMPONENT_utility.debug_console_lite=y
CONFIG_MCUX_COMPONENT_utility.format=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_host=y
CONFIG_MCUX_COMPONENT_middleware.wireless.controller_api=y
CONFIG_gNvFragmentation_Enabled_d=n
CONFIG_MCUX_COMPONENT_component.osa_bm=y
CONFIG_MCUX_COMPONENT_component.power_manager_framework=y
CONFIG_MCUX_COMPONENT_component.reset_adapter=y
CONFIG_MCUX_COMPONENT_component.serial_manager_uart=y
CONFIG_MCUX_DEPENDENCY_COMPONENT_component.serial_manager_uart_DEPEND_COMPONENT_driver.
CONFIG_MCUX_COMPONENT_middleware.multicore.rpmsg-lite.bm=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.sensors=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.lowpower=y
CONFIG_MCUX_PRJSEG_module.board.wireless.board.lowpower=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support.no_flash_selected=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_host_peripheral_component_lib=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_host_common=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_host_component_lib=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_gatt_db=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_gatt_service_discovery=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_profiles_wireless_uart=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_profiles_battery_service=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_profiles_otap=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_profiles_battery_service=y
CONFIG_MCUX_COMPONENT_middleware.wireless.ble_profiles_device_info_service=y
# Add source files for Low Power - adding Kconfig board.lowpower is enough to
 get all underlying dependancies
CONFIG_MCUX_PRJSEG_module.board.wireless.board.lowpower=y
```

3. Clean and rebuild the project.
4. Verify that:
   - No build errors occur.
   - The driver is successfully compiled and linked.

4.2 Main modifications in the source files

After including the OTAP client folders and files in the custom project, the next step involves inspecting the differences between the source files of the OTAP client and the Bluetooth LE application. To integrate the OTAP service, add the required code. The subsections that follow explain the main aspects to focus on.

4.2.1 app_preinclude.h

gOtaClientAtt_d 1: It sets the Attribute (ATT) protocol transference method for OTA updates. It must be set to 1 for its own purpose.

```
/*!
 *************************************************************************
 *     BLE OTAP Configuration
 *************************************************************************/
#define gOtapClientAtt_d   1

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

```
/*! 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 means 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)
```

In this file, you must add some OTA characteristics:

```
/*! 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
/* Enable/Disable the FSCI BLE OTAP module */
#define gFsciBleOtapEnabled_d           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 means 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 defines structures that configure the advertising and scanning parameters and data. It also specifies 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) supports. The Bluetooth LE scanner uses this information to filter out the advertiser devices that do not offer the required services.

Therefore, the OTAP client service must appear in the advertising data to notify the OTAP server of its availability.

The code below demonstrates how the scan response data includes this information.

```
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, shape the profile of the Generic Attribute (GATT) server (WU-OTAP client device). The most important step in this guide involves including the list of the OTAP client attributes in the database of the device. Opening the OTAP client SDK example and the Bluetooth LE demo helps in comparing both GATT databases.

The following code snippet 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 custom UUID definitions and their assignments. The gatt_uuid128.h does not contain definitions in the original Wireless UART SDK project, as the Wireless UART and the battery services follow the Bluetooth Special Interest Group (SIG) standards. However, the OTAP service and its characteristics require manual specification by the developer using a 128–Universally Unique Identifier (UUID). The following shows how to implement the 128-bit UUID assignment 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 functions as the main source file at the application level. It manages all the procedures that the device performs before, during, and after establishing a connection. The following steps describe the main changes required to integrate the OTAP service.

1. To reference the OTAP files in the project, except otap_client_att.h merge the missing #include preprocessor directives.
   Note: *This step depends on the software configuration. Since we are integrating the OTAP feature into another example, we always include these files.*

```
/*OTAP includes*/
#include "OtaSupport.h"
#include "device_info_interface.h"
#include "otap_interface.h"
#include "fwk_platform_ble.h"
#include "otap_client.h"
```

2. Add the function prototypes and global variables used by the OTAP client software. As noted in step 1, these additions can vary depending on the specific application. In this example, skip merging the appTimerId variable in the Wireless UART project. It is used in the OTAP client to create a timer instance that is not implemented here.
   - Add the following code under the private type declarations section:

```
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;
```

   - Add the following code under the private memory declarations section:

```
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 and typically handles settings for configuring advertising, filter accept list, services, and more. Insert the following code at the beginning of this function:

```
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 function. The connection callback triggers whenever a connection event occurs, such as a connection or disconnection.
   a. In the connection case [**case***gConnEvtConnected_c*:], include the OtapCS_Subscribe and OtapClient_HandleConnectionEvent functions, as shown below:

```
/* Subscribe client*/
           mPeerDeviceId = peerDeviceId;
           void)Bas_Subscribe(&basServiceConfig, peerDeviceId);
           void)OtapCS_Subscribe(peerDeviceId);

           /* UI */
           LedStopFlashingAllLeds();
           Led1On();

           OtapClient_HandleConnectionEvent (peerDeviceId);
```

   b. In the disconnection case [**case***gConnEvtDisconnected_c*:], include the OtapCS_Unsubscribe and OtapClient_HandleDisconnectionEvent functions. In the Bonding condition, also include the OtapClient_HandleEncryptionChangedEvent function.

```
/* Unsubscribe client */
            mPeerDeviceId = gInvalidDeviceId_c;
            (void)Bas_Unsubscribe(&basServiceConfig, peerDeviceId);
            (void)OtapCS_Unsubscribe();

            /* UI */
            LedStopFlashingAllLeds();
            Led1Flashing();
            Led2Flashing();

            /* Restart advertising*/
            BleApp_Start(gGapPeripheral_c);

            OtapClient_HandleDisconnectionEvent (peerDeviceId);
```

5. Locate the BleApp_GattServerCallback. It manages all the incoming communications from the client devices. Add the GATT events that the OTAP client software must handle: gEvtAttributeWritten_c, gEvtMtuChanged, gEvtCharacteristicCccdWritten_c, gEvtAttributeWrittenWithoutResponse_c, gEvtHandleValueConfirmation_c, and gEvtError
   The Bluetooth LE project can share some common GATT events. If it is the case, add a conditional structure for each attribute handle. Focus on the gEvtAttributeWritten_c case and observe the conditional structure included for handling the Wireless UART control point and the OTAP control point.

```
    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:
        {
            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, uses the listed software:

- OTAP client SDK software, programmed in the FRDM-MCXW72 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.

The following subsections that follow explain how to build the software required for the testing case proposed by this document.

5.1 Preparing the OTAP client SDK software

Flash the OTAP client firmware onto the FRDM-MCXW72 board using MCUXpresso for VS Code.

5.2 Creating a Wireless UART-OTAP SREC image to update the software

1. Clean and build the WU-OTAP project.
2. Deploy the explorer icon in the toolbar, then look for the debug folder.
3. Right-click the .axf file and select **Binary Utilities** → **Create S-Record**. This step generates an SREC file that saves in the debug folder in the workspace with a .srec extension.

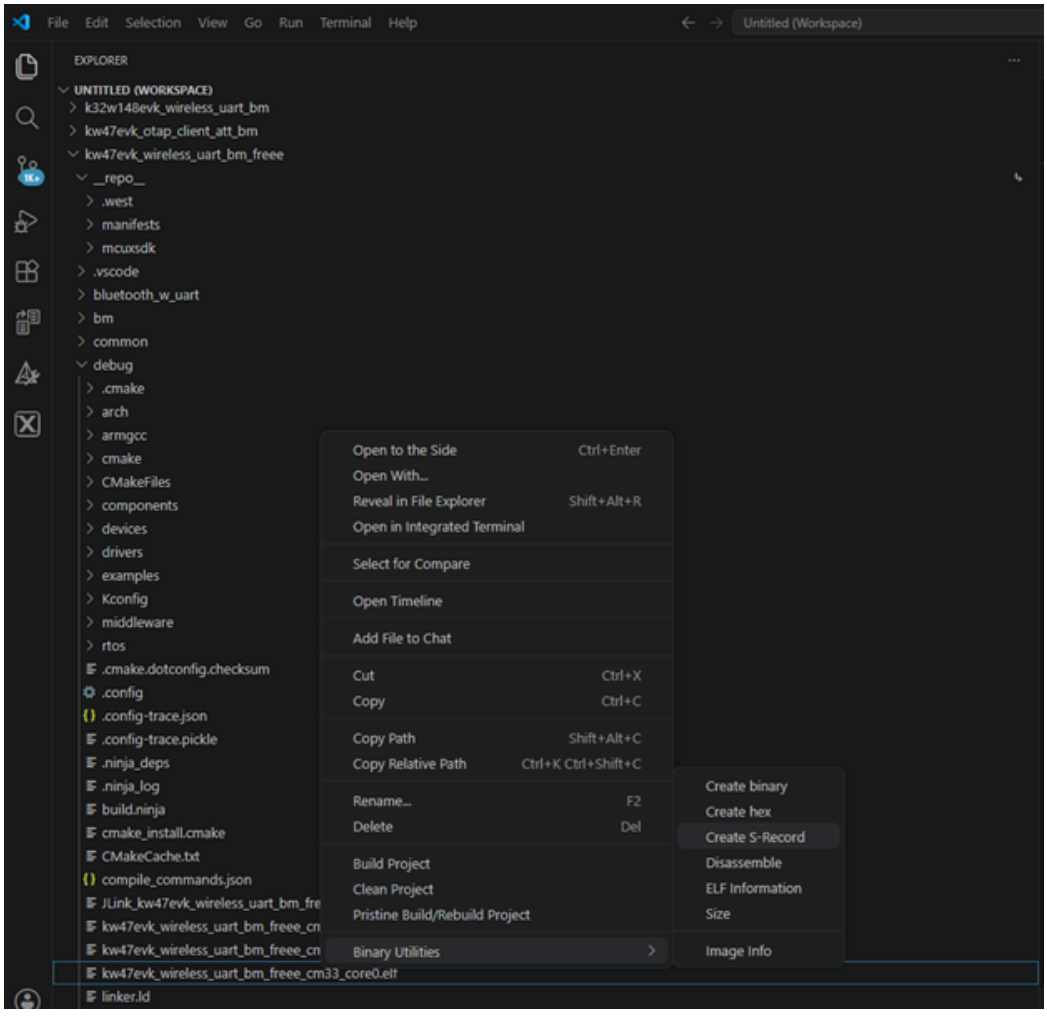

Figure 5. Generate an SREC file in MCUXpresso for VS Code

4. To generate the .bleota image, follow the instructions provided in *Creation of Firmware Update Image for KW47 using Over the Air Programming Tool* (AN14720)

5.3 Testing the Wireless UART-OTAP software

To test the Wireless UART-OTAP software using the IoT Toolbox application, follow the steps below:

1. Open the **IoT Toolbox** App and select the **OTAP** demo.
2. To start scanning for a suitable advertiser, click the *SCAN* button.

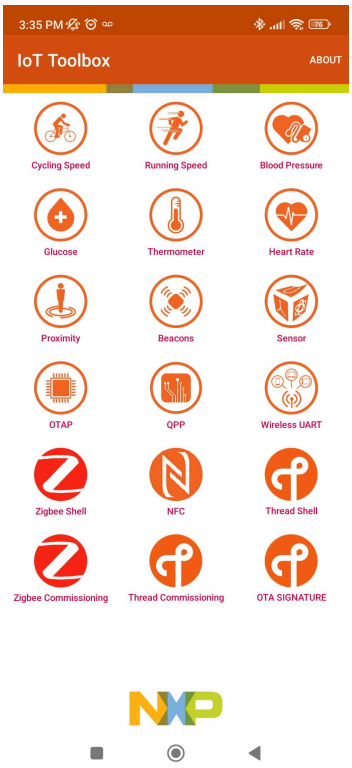

Figure 6. Selecting the OTAP demo in the IoT toolbox

3. To start advertising, click the ADV button, then SW2, followed by SW4 on the FRDM-MCXW72.
4. Create a connection with the NXP_WU device. Then, the OTAP interface gets displayed on your smartphone.

Figure 7. Connecting to NXP_WU and uploading .bleota file

5. Click the **Open** button and browse for the Wireless UART-OTAP .bleota file stored on the smartphone.
6. To start the transfer, click *Upload*. Wait until the confirmation message indicating a successful upload.

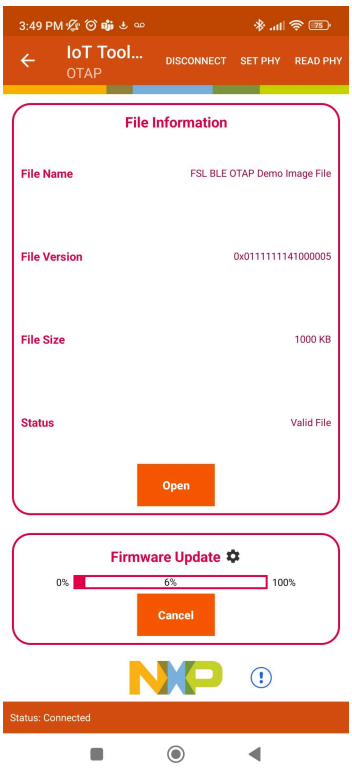

Figure 8. Verifying firmware update through IoT toolbox

7. Wait until the OTAP bootloader finishes programming the new image. The wireless UART application starts automatically, indicated by the RGB LED blinking.
8. To start advertising, click the ADV, then SW3, followed by the SW2 buttons again on the FRDM-MCXW72 board.
9. Verify that the device appears in both the Wireless UART and OTAP applications of the IoT Toolbox. The name of the device is NXP_WU. Establish a connection and interact with both demos.
10. Connect the WU-OTAP device with the OTAP smartphone application and update the software using the Wireless UART .bleota file.
11. Confirm that the device has been updated to a simple Wireless UART, using the Wireless UART-OTAP demo.
12. Once again, click the ADV, then SW3, followed by the SW2 buttons on the FRDM-MCXW72 board to start advertising. Now, the name is NXP_WU.
13. Connect the device using the Wireless UART demo in the IoT Toolbox application and verify the update functions correctly.

6 Acronyms

Section 6 lists the acronyms used in this document along with their description.

Table 1. Acronyms

Term	Description
ATT	Attribute
Bluetooth LE	Bluetooth Low Energy
GATT	Generic Attribute
IDE	Integrated Development Environment
MCU	Microcontroller Unit
OTAP	Over the Air Programming
RGB	Red, Green, and Blue
RoTKTH	Root of Trust Key Table Hash
SB3KDK	Secure Boot 3 Key Derivation Key
SDK	Software Development Kit
SIG	Special Interest Group
SPI	Serial Peripheral Interface
SREC	S-Record
UART	Universal Asynchronous Receiver/Transmitter
UUID	Universally Unique Identifier
WU	Wireless UART

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2026 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 2 summarizes the revisions to this document.

Table 2. Revision history

Document ID	Release date	Description
AN15123 v.1.0	3 August 2026	Initial public release

Legal information

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.

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 https://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 document 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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

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.

Contents