1 Introduction

The Over The Air Programming (OTAP) NXP’s custom Bluetooth LE service provides the developers 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 to integrate it into the Bluetooth LE application. In that way, you can reprogram the device as many times as required.

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

2 OTAP client software

OTAP memory management during the update process describes the actual implementation of the OTAP client software included in the SDK package for FRDM-KW36. Advantages of the OTAP service integration explains the importance of integrating OTAP client software into your application, and what it is expected to achieve through this application note.

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 $0x0007_FFFF$. Writing or reading at the Alias range address modifies or returns the FlexNVM content respectively.
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, and then with the Bluetooth LE application supporting OTAP client. The mechanism is created to have two different software coexisting in the same device and store 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 rest of the memory is reserved, among other things, by the OTAP client application.

3. To create a new image file for the OTAP 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 contain the Bootloader Flags at the corresponding address to work properly.
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 the On-Chip FlexNVM memory. The destination of the code is selectable in the OTAP client software.

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. Because 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 is not able to be programmed again due to the lack of OTAP functionality. For more description, see **Advantages of the OTAP service integration**.

![Memory content at the end of the software update process](image)

**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. These values might change with linker settings. You can inspect the effective memory addresses in your project.

### 2.2 Advantages of the OTAP service integration

As explained in **OTAP memory management during the update process**, the OTAP client software is a single-programming demo application. Supposing that an OTAP client device is programmed with the OTAP client software, this device requests an update, for example, a Temperature Collector (Temp Coll). The image that the OTAP server will send to the OTAP client must be the Temp Coll. After the reprogramming process the device that was the OTAP client, now, has turned into a Temperature Collector. The Temp Coll does not have the capabilities to communicate with the OTAP server and request for another update. But if the Temp Coll image had included the OTAP client service as well, the device would have the possibility to request another software update, for example, a modified Wireless UART example with OTAP Service. Due to the WU software already 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.
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 Temp Coll 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 Temp Coll - OTAP integration demo.

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

3.1 Downloading and installing the software development kit

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 MCUXpresso.
2. Click Select Development Board. Log in with your registered account.
3. In the Search by Name field, search for FRDM-KW36. Then click the suggested board and click Build MCUXpresso SDK.
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.
5. Click **Download SDK** and it will take a few minutes until the system gets the package into your account on the MCUXpresso web page. Read and accept the license agreement. The SDK download starts automatically on your PC.

6. Open MCUXpresso IDE. Drag and drop the FRDM-KW36 SDK zip in the **Installed SDK's** list.

![Installed SDKs](image)

Figure 9. Importing SDK package to MCUXpresso IDE

Now, 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 into it. This guide uses a Temp Coll project 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 Temp Coll example

The OTAP client software makes use of Framework functionalities that are not included for the Temp Coll 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 Temp Coll (left) and the OTAP Client (right) is as shown in **Figure 10**.
Figure 10. Source tree comparison
The folders and files, which are in OTAP but not in Temp Coll, must be imported in your Temp Coll project. For example, in Figure 10, the followings are required to be imported:

- `bluetooth -> profiles -> otap`
- `framework -> Flash -> External`
- `framework -> OtaSupport`
- `source -> common -> otap_client`
- `linkscripts -> main_text_section.ldt`

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

1. Expand the `bluetooth` and `framework` folders in your workspace. Select the folder needed for updates and click the right mouse button. Select New -> Folder. The Folder window appears to provide the same name as the missing folder in the source directory, as shown in Figure 11.

2. Repeat Step 1 for the left folders. The result must look similar as shown in Figure 12.
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 Temp Coll side. For this example, these files are listed as below.

- `otap_interface.h` and `otap_service.c` in the `bluetooth -> profiles -> otap` folder.
- `Eeprom.h` in the `framework -> Flash -> External -> Interface` folder.
- Eeprom source files in the framework -> Flash -> External -> Source folder.
- OtaSupport.h in the framework -> OtaSupport -> Interface folder.
- OtaSupport.c in the framework -> OtaSupport -> Source folder.
- main_text_section.ldt in linkscripts folder.
- otap_client.h and otap_client.c in the source -> common -> otap_client folder.

```
frdmkw36_wireless_examples_bluetooth_temp_coll_freertos

  > Project Settings
  > Includes
  > CMSIS
  > bluetooth
    > controller
    > host
  > profiles
    > battery
    > device_info
  > otap
    > otap_interface.h
    > otap_service.c
  > temperature
```

Figure 13. OTAP files integrated into the Temp Coll project

4. Replace `lib_ble_5-0_host_central_cm0p_gcc.a`, included in your project in libs folder, with the `lib_ble_5-0_host_cm0p_gcc.a` library located in `<SDK_path>/middleware/wireless/bluetooth_1.3.6/host/lib`.

You can simply drag and drop the new library on the workspace and remove the original library file. To remove the old library, right-click on the file and select Delete.
5. 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 15. In the new window that appears, click the **Workspace** button.

6. Deploy your directory tree in the folder selection window. Select the following folders:
   - `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.

![Properties for listed - wireless examples bluetooth_temp_call_s往返.png](image)

**Figure 16. Including the OTAP folders in the project paths**

7. Open the MCU Linker -> Libraries view. Remove the `_ble_5-0_host_central_cm0p_gcc` library and replace it with the `_ble_5-0_host_cm0p_gcc` library, as shown in **Figure 17**. Then, click OK to save the changes.
Now, you have included the OTAP client Bluetooth and Framework services in the Temp Coll project.

4.2 Main modifications in the source files

Once you have included the OTAP client folders and files in your custom project, inspect the differences between the source files of the OTAP client and your Bluetooth LE application and add the code needed to integrate the 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 configuration macros. The OTAP client software requires some definitions that are not included for other Bluetooth LE SDK projects. The software update must include the following definitions:

- gEepromType_d
- gEepromParams_WriteAlignment_c
- gOtapClientAtt_d

The OTAP Temp Coll demo, sets the following values:

1. **gEepromType_d**: It defines the storage method between the AT45DB041E external flash on the FRDM-KW36 board, the default value, or the FlexNVM on-chip memory. You can also select from the list of EEPROM devices in the *Eeprom.h* header file at framework/Flash/External/Interface, for custom boards.

```c
/* Specifies the type of EEPROM available on the target board */
#define gEepromType_d       gEepromDevice_AT45DB041E_c
```
2. **gEepromParams_WriteAlignment**: It defines the offset of the software update for programming. Do not modify the default value.

```c
/* 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.

```c
#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, Temp Coll – 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.

The following code must be included in this file to prepare the advertising data:

```c
/* Default Advertising Parameters. Values can be changed at runtime to align with profile requirements */
#define gGapAdvertisingInterval_050ms_c 0x0050
#define gGapAdvertisingInterval_100ms_c 0x00A0
#define gGapAdvertisingInterval_125ms_c 0x00C8
#define gGapAdvertisingInterval_250ms_c 0x0190
#define gGapAdvertisingInterval_500ms_c 0x0320

gapAdvertisingParameters_t gAdvParams = {
    /* minInterval */           gGapAdvertisingInterval_050ms_c,
    /* maxInterval */           gGapAdvertisingInterval_100ms_c,
    /* advertisingType */       gAdvConnectableUndirected_c,
    /* addressType */           gBleAddrTypePublic_c,
    /* directedAddressType */   gBleAddrTypePublic_c,
    /* directedAddress */       {0, 0, 0, 0, 0, 0},
    /* channelMap */            (gapAdvertisingChannelMapFlags_t)(gGapAdvertisingChannelMapDefault_c),
    /* filterPolicy */          gProcessAll_c
};

/* Scanning and Advertising Data */
static const uint8_t adData0[1] = { (gapAdTypeFlags_t)(gLeGeneralDiscoverableMode_c | gBrEdrNotSupported_c) };
static const uint8_t adData1[] = { 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E,
    0xBA, 0x50, 0x55, 0xFF, 0x01};
static const gapAdStructure_t advScanStruct[3] = {
    { .length = NumberOfElements(adData0) + 1,
      .adType = gAdvFlags_c,
      .aData = (uint8_t *)adData0 }
},
    { .length = NumberOfElements(adData1) + 1,
      .adType = gAdvComplete128bitServiceList_c,
      .aData = (uint8_t *)adData1 }
};
```

NXP Semiconductors

Customizing a based Bluetooth LE demo to integrate the OTAP service

KW36 - Integrating the OTAP Client Service into a Bluetooth LE Central Device, Rev. 0, 09/2020

Application Note 15 / 46
Additionally, you need to add the access security requirements for the OTAP service, including the `gapServiceSecurityRequirements_t` struct. You can customize these parameters for your purpose. The Temp Coll – OTAP demo sets the following parameters:

```c
    { .requirements = { .securityModeLevel = gSecurityMode_1_Level_3_c, .authorization = FALSE, .minimumEncryptionKeySize = gDefaultEncryptionKeySize_d }, .serviceHandle = (uint16_t)service_otap },
    { .requirements = { .securityModeLevel = gSecurityMode_1_Level_3_c, .authorization = FALSE, .minimumEncryptionKeySize = gDefaultEncryptionKeySize_d }, .serviceHandle = (uint16_t)service_battery },
    { .requirements = { .securityModeLevel = gSecurityMode_1_Level_3_c, .authorization = FALSE, .minimumEncryptionKeySize = gDefaultEncryptionKeySize_d }, .serviceHandle = (uint16_t)service_device_info }
};
```

Last modification is to register all services into the `deviceSecurityRequirements` struct. See the following portion of code:

```c
gapDeviceSecurityRequirements_t deviceSecurityRequirements = {
    .pMasterSecurityRequirements  = (void*)&masterSecurity, .cNumServices             = 3, .aServiceSecurityRequirements = (void*)&serviceSecurity
};
```
4.2.3  gatt_db.h and gatt_uuid128.h

The `gatt_db.h` header file contains the list of attributes, which shapes the profile of the Temp Coll-OTAP 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 to compare both GATT databases, and include the list of attributes missing in your project. Figure 18 shows the OTAP client portion of the database that must be included in the Temp Coll project.

![Figure 18. OTAP client service](image1)

The `gatt_uuid128.h` header file contains all the custom UUID definitions and its assignation. OTAP service and its characteristics need to be specified by the developer as a 128 – UUID in this file. Figure 19 shows how to implement the 128 – UUID assignation for the OTAP service.

![Figure 19. Temp Coll – OTAP 128 – UUID definitions](image2)

4.2.4 temperature_collector.h and temperature_collector.c

The `temperature_collector.c` file 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 in `temperature_collector.c` file, except the `#include statement for the otap_client_att.h` file. Figure 20 shows the comparison between Temp Coll (left) and OTAP client (right) application files. This step depends on your software as it might share different files than this example. The results are similar as depicted in Figure 21, before (Temp Coll left) and after (Temp Coll – OTAP right).
Figure 20. Comparison between Temp Coll (left) and OTAP (right) include directives
2. Add the function prototypes and global variables that are used by the OTAP client software. Figure 22 and Figure 23 show the comparison between Temp Coll (left) and OTAP (right). As mentioned in Step 1, this might depend on your application. Also, it is required to create another variable: `static gapRole_t mGapRole`. This variable will be used to switch between central and peripheral GAP roles as the OTAP Client device needs to be the Bluetooth LE peripheral to advertise this service and communicate with the OTAP Server. In other words, your KW36 device will be the Bluetooth LE Central device while it is the Temp Coll and will be the Bluetooth LE Peripheral device while it is the OTAP Client. The results are similar as depicted in Figure 24 and Figure 25.
Figure 22. Comparison between Temp Coll (left) and OTAP (right) prototypes (1)
Figure 23. Comparison between Temp Coll (left) and OTAP (right) prototypes (2)
Figure 24. Merging the variables into the project: before (Temp Coll left) and after (Temp Coll - OTAP) (1)
3. Locate the `BleApp_Init` function. You can add in this function, driver initialization API's for the application. The Temp Coll does not support the Battery Service while the OTAP Client does, so you must call the `BOARD_InitAdc` function. See the following portion of code:

```c
void BleApp_Init(void)
{
    /* Initialize application support for drivers */
    BOARD_InitAdc();
    /* Init shell and set prompt */
    shell_init("BLE Temp Collector");
#if defined(MULTICORE_APPLICATION_CORE) && (MULTICORE_APPLICATION_CORE)
    Init eRPC host */
    init_rpc_host();
```
4. Locate the `BleApp_Start` function. This function is used to start the scanning. Modify it as follows to start scanning or advertising depending on the `gapRole` variable and change the `BleApp_Start` prototype at `temperature_collector.h` by

```c
void BleApp_Start(gapRole_t gapRole)
{
    switch (gapRole)
    {
    case gGapCentral_c:
    {
        if (!mScanningOn)
        {
            gPairingParameters.localIoCapabilities = gIoKeyboardDisplay_c;
            /* Stop advertising, Start scanning */
            if (mAdvState.advOn)
            {
                (void)Gap_StopAdvertising();
            }
            (void)App_StartScanning(&gScanParams, BleApp_ScanningCallback,
                                    gGapDuplicateFilteringEnable_c, gGapScanContinuously_d, gGapScanPeriodicDisabled_d);
        }
        break;
    }
    case gGapPeripheral_c:
    {
        if (mPeerInformation.deviceId == gInvalidDeviceId_c)
        {
            /* Device is not connected and not advertising*/
            if (!mAdvState.advOn)
            {
                #if gAppUseBonding_d
                if (gcBondedDevices > 0)
                {
                    mAdvState.advType = whiteListAdvState_c;
                }
                else
                {
                    #endif
                    mAdvState.advType = advState_c;
                    #if gAppUseBonding_d
                    }
                    #endif
                    gPairingParameters.localIoCapabilities = gIoDisplayOnly_c;
                    /* Stop scanning, Start advertising */
                    if (mScanningOn)
                    {
                        (void)Gap_StopScanning();
                    }
                    BleApp_Advertise();
                }
            }
        break;
    }
    default:
    {
        /* No action required */
```
5. Locate the `BleApp_HandleKeys` function. This is executed each time that you press a switch on the FRDM-KW36 board. It must be modified to change the GAP role of the device when the user presses a switch button, allowing to move from Temp Coll to OTAP Client and return to Temp Coll if it is desired.

```c
void BleApp_HandleKeys(key_event_t events)
{
    switch (events)
    {
        /* Start on button press if low-power is disabled */
        case gKBD_EventPressPB1_c:
        {
            BleApp_Start(mGapRole);
            break;
        }
        /* Disconnect on long button press */
        case gKBD_EventLongPB1_c:
        {
            if (mPeerInformation.deviceId != gInvalidDeviceId_c)
            {
                (void)Gap_Disconnect(mPeerInformation.deviceId);
            }
            break;
        }
        /* Toggle Gap Role */
        case gKBD_EventPressPB2_c:
        case gKBD_EventLongPB2_c:
        {
            if (mGapRole == gGapCentral_c)
            {
                mGapRole = gGapPeripheral_c;
            }
            else
            {
                mGapRole = gGapCentral_c;
            }
            break;
        }
        default:
        {
            ; /* No action required */
            break;
        }
    }
}
```

6. Locate the `BleApp_GenericCallback` function. Include the handling of `gAdvertisingParametersSetupComplete_c`, `gAdvertisingDataSetupComplete_c`, and `gAdvertisingSetupFailed_c` events.

```c
void BleApp_GenericCallback (gapGenericEvent_t* pGenericEvent)
{
    /* Call BLE Conn Manager */
    BleConnManager_GenericEvent(pGenericEvent);
    switch (pGenericEvent->eventType)
    {
```
case gInitializationComplete_c:
    {
        BleApp_Config();
    }
    break;

case gAdvertisingParametersSetupComplete_c:
    {
        (void)Gap_SetAdvertisingData(&gAppAdvertisingData, &gAppScanRspData);
    }
    break;

case gAdvertisingDataSetupComplete_c:
    {
        (void)App_StartAdvertising(BleApp_AdvertisingCallback, BleApp_ConnectionCallback);
    }
    break;

case gAdvertisingSetupFailed_c:
    {
        panic(0,0,0,0);
    }
    break;

default:
    {
        /* No action required */
    }
    break;
}

7. Locate the BleApp_Config function. The BleApp_Config function configures the initial GAP role of the device, with the default value that Temp Coll – OTAP is GAP Central, registers the notifiable attributes, prepares the services built on the database, and allocates some application timers. Register the BleApp_GattServerCallback defines an initial GAP role for the device, including the OtapClient_Config and Dis_Start functions to initialize these services and allocate a timer for the battery measurements. The result should look as the following example.

```c
static void BleApp_Config(void)
{
    #if defined(MULTICORE_APPLICATION_CORE) && (MULTICORE_APPLICATION_CORE == 1)
        if (GattDbDynamic_CreateDatabase() != gBleSuccess_c)
            {
                panic(0,0,0,0);
                return;
            }
    #endif /* MULTICORE_APPLICATION_CORE */
    /* Configure as GAP Central */
    BleConnManager_GapCommonConfig();
    /* Register for callbacks*/
    (void)App_RegisterGattServerCallback(BleApp_GattServerCallback);
    (void)App_RegisterGattClientProcedureCallback(BleApp_GattClientCallback);
    (void)App_RegisterGattClientNotificationCallback(BleApp_GattNotificationCallback);
    BleServDisc_RegisterCallback(BleApp_ServiceDiscoveryCallback);
    /* By default, always start node as GAP central */
    mGapRole = gGapCentral_c;
    /* Initialize private variables */
    mPeerInformation.appState = mAppStateIdle_c;
    mPeerInformation.deviceId = gInvalidDeviceId_c;
    mScanningOn = FALSE;
    mAdvState.advOn = FALSE;
    mFoundDeviceToConnect = FALSE;
    /* Start services */
```
8. Include after `BleApp_Config`, the `BleApp_Advertise` function:

```c
static void BleApp_Advertise(void)
{
    switch (mAdvState.advType)
    {
    #if gAppUseBonding_d
        case whiteListAdvState_c:
            
            gAdvParams.filterPolicy = gProcessWhiteListOnly_c;
            
            break;
    #endif
        case advState_c:
            
            gAdvParams.filterPolicy = gProcessAll_c;
            
            break;
        default:
            ; /* For MISRA compliance */
            
            break;
    }
/* Set advertising parameters*/
(void)Gap_SetAdvertisingParameters(&gAdvParams);
}
```

9. Include after `BleApp_ScanningCallback`, the `BleApp_AdvertisingCallback` function:

```c
static void BleApp_AdvertisingCallback (gapAdvertisingEvent_t* pAdvertisingEvent)
{
    switch (pAdvertisingEvent->eventType)
    {
    case gAdvertisingStateChanged_c:
        
        mAdvState.advOn = !mAdvState.advOn;
        if(mAdvState.advOn)
        {
            shell_write("\r\nAdvertising...\r\n");
            LED_StopFlashingAllLeds();
            Led2Flashing();
        }
    }
    
    break;
```
case gAdvertisingCommandFailed_c:
  {
    Led2On();
    panic(0,0,0,0);
  }
break;
default:
  ; /* For MISRA compliance */
break;
}

10. **Locate the BleApp_ConnectionCallback function.** The connection callback is triggered whenever a connection event happens, such as a connection or disconnection. Modify this callback to handle the OTAP service as follows:

```c
static void BleApp_ConnectionCallback (deviceId_t peerDeviceId, gapConnectionEvent_t* pConnectionEvent)
{
  if(mGapRole == gGapCentral_c)
  {
    /* Connection Manager to handle Host Stack interactions */
    BleConnManager_GapCentralEvent(peerDeviceId, pConnectionEvent);
  }
  else
  {
    /* Connection Manager to handle Host Stack interactions */
    BleConnManager_GapPeripheralEvent(peerDeviceId, pConnectionEvent);
  }
  switch (pConnectionEvent->eventType)
  {
    case gConnEvtConnected_c:
    {
      mAdvState.advOn = FALSE;
      /* Subscribe client*/
      (void)Bas_Subscribe(&basServiceConfig, peerDeviceId);
      (void)OtapCS_Subscribe(peerDeviceId);
      /* Update UI */
      LED_StopFlashingAllLeds();
      Led3On();
      shell_write("
Connected!
");
      mPeerInformation.deviceId = peerDeviceId;
      mPeerInformation.isBonded = FALSE;
      #if (cPWR_UsePowerDownMode)
        /* Device does not need to sleep until some information is exchanged with the peer.
        */
        PWR_DisallowDeviceToSleep();
      #endif
      if(!TMR_IsTimerActive(mBatteryMeasurementTimerId)){
        /* Start battery measurements */
        (void)TMR_StartLowPowerTimer(mBatteryMeasurementTimerId, gTmrLowPowerIntervalMillisTimer_c, TmrSeconds(mBatteryLevelReportInterval_c), BatteryMeasurementTimerCallback, NULL);
      }
      if(mGapRole == gGapCentral_c)
      {
        #if defined(gAppUseBonding_d) && (gAppUseBonding_d)
          (void)Gap_CheckIfBonded(peerDeviceId, &mPeerInformation.isBonded);
          if ((mPeerInformation.isBonded) &&
```
(gBleSuccess_c == Gap_LoadCustomPeerInformation(peerDeviceId, (void*) &mPeerInformation.customInfo, 0, sizeof (appCustomInfo_t)))

mRestoringBondedLink = TRUE;
/* Restored custom connection information. Encrypt link */
(void)Gap_EncryptLink(peerDeviceId);

#endif

BleApp_StateMachineHandler(mPeerInformation.deviceId, mAppEvt_PeerConnected_c);
if(mGapRole == gGapPeripheral_c)
{
/* Handle OTAP connection event */
OtapClient_HandleConnectionEvent (peerDeviceId);
}
}
break;

case gConnEvtDisconnected_c:
{
/* Unsubscribe client */
(void)Bas_Unsubscribe(&basServiceConfig, peerDeviceId);
(void)OtapCS_Unsubscribe();
mPeerInformation.deviceId = gInvalidDeviceId_c;
mPeerInformation.appState = mAppIdle_c;
TMR_StopTimer(mBatteryMeasurementTimerId);
if(mGapRole == gGapPeripheral_c)
{
/* Handle OTAP disconnection event */
OtapClient_HandleDisconnectionEvent (peerDeviceId);
}
/* Reset Service Discovery to be sure*/
BleServDisc_Stop(peerDeviceId);
/* Update UI */
shell_write("\r\nDisconnected!\r\n");
LED_TurnOffAllLeds();
LED_StartFlash(LED_ALL);
}
break;
#if gAppUsePairing_d
case gConnEvtPairingComplete_c:
{
/* Notify state machine handler on pairing complete */
if (pConnectionEvent->eventData.pairingCompleteEvent.pairingSuccessful)
{
BleApp_StateMachineHandler(mPeerInformation.deviceId, mAppEvt_PairingComplete_c);
}
}
break;
#if defined(gAppUseBonding_d) && (gAppUseBonding_d)
case gConnEvtEncryptionChanged_c:
{
if( pConnectionEvent->eventData.encryptionChangedEvent.newEncryptionState )
{
if( mRestoringBondedLink )
{
/* Try to enable temperature notifications, disconnect on failure */
if( gBleSuccess_c != BleApp_ConfigureNotifications() )
{
(void)Gap_Disconnect(peerDeviceId);
}

}
11. Develop the `BleApp_GattServerCallback` function. It manages all the incoming communications from the client devices. Add the GATT server events that need to be handled by the OTAP client software. See the following example.

```c
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t* pServerEvent)
{
    switch (pServerEvent->eventType)
    {
    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 gEvtAttributeWritten_c:
        {  
            OtapClient_AttributeWritten (deviceId,  
                                           pServerEvent->eventData.attributeWrittenEvent.handle,  
                                           pServerEvent->eventData.attributeWrittenEvent.cValueLength,  
                                           pServerEvent->eventData.attributeWrittenEvent.aValue);  
        }  
        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,  
                                                 pServerEvent->eventData.handleValueConfirmationEvent.handle,  
                                                 pServerEvent->eventData.handleValueConfirmationEvent.cValueLength,  
                                                 pServerEvent->eventData.handleValueConfirmationEvent.aValue);  
        }  
        break;
    default:
        ;  /* No action required */  
        break;
    }
}
```
```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 */
          #endif /* gAppUsePairing_d */
        (void)Gap_SendSlaveSecurityRequest(deviceId, &gPairingParameters);
      }
      #endif /* gAppUseBonding_d */
      break;
    }
  default:
    ; /* For MISRA compliance */
    break;
}
```

12. Include the timer callback for the battery service:

```c
static void BatteryMeasurementTimerCallback(void * pParam)
{
  basServiceConfig.batteryLevel = BOARD_GetBatteryLevel();
  (void)Bas_RecordBatteryMeasurement(&basServiceConfig);
}
```

Now, you have integrated the OTAP Client code into the Temp Coll.

**NOTE**

The example provided with this application note removes all references related to low power support in the `temperature_collector.c`, except the `PWR_DisallowDeviceToSleep` function at `BleApp_ConnectionCallback`. The `DisconnectTimerCallback` and its associated timer instance and it changes the state machine of the RGB LED. These changes are not listed in the steps above since they are not relevant for this application note.

### 4.3 Modifications in 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 Temp Coll demo, so it should be included to integrate the OTAP on the application. Follow the next steps to set the project settings and storage configurations.
1. Locate the `app_preinclude.h` file under the source folder of the project.
   - To select external flash storage method, set the `gEepromType` define to `gEepromDevice_AT45DB041E_c`, in the attached Temp Coll-OTAP software by default.
   - To select internal flash storage method, set the `gEepromType` define to `gEepromDevice_InternalFlash_c`.

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

   Figure 26. Configuring the storage method at the preinclude file

2. Click on the Temp Coll-OTAP demo in the MCUXpresso workspace.

3. Navigate to Project -> Properties in MCUXpresso IDE. Go to C/C++ Build -> MCU settings.
   - To select external flash storage method, as shown in Figure 27, configure the fields in the Memory details pane. This is the default storage method for the Tempertaure Collector - OTAP software.

   ![Figure 27. Configuring external storage method](image)

   - To select internal flash storage method, as shown in Figure 28, configure the fields in the Memory details pane.

   ![Figure 28. Configuring internal storage method](image)

4. Clean and build the project.

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

5 Testing the Temp Coll-OTAP demo

The test case example, designed to demonstrate the OTAP integration in Testing the Temp Coll-OTAP software, makes use of the following software:

- OTAP Client SDK software, programmed in the FRDM-KW36 board.
- An SREC software update of the Temp Coll-OTAP example.
- An SREC software update of the Temp Coll 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. Attach your FRDM-KW36 board on the PC.
2. Program the OTAP Bootloader on the FRDM-KW36.
   - Drag and drop the prebuilt binary file from the following path on the COM port icon corresponding to the FRDM-KW36 device:
     ```
     <FRDM-KW36_SDK_root>/tools/wireless/binaries/bootloader_otap_frdmkw36.bin
     ```
3. Open MCUXpresso IDE.
In the **Quickstart Panel** view, click **Import SDK example(s)**.

![Quickstart panel perspective](image)

**Figure 29. Quickstart panel perspective**

4. Click twice on the frdmkw36 icon.

![Device selection perspective](image)

**Figure 30. Device selection perspective**

5. In the **Examples** text field, type `otac_att`. In the **Name** pane, click `wireless_examples` -> `bluetooth` -> `otac_att` -> `freertos`. Click **Finish**.
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 `gEepromType` to `gEepromDevice_AT45DB041E_c`.
      - To select the internal flash storage method, On-chip FlexNVM memory, set `gEepromType` to `gEepromDevice_InternalFlash_c`.

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

      Figure 32. Configuring the storage method at the preinclude file

   b. Navigate to **Project -> Properties** in MCUXpresso IDE. Go to **C/C++ Build -> MCU settings** perspective.

      - To select external flash storage method, as shown in Figure 33, configure the fields in the **Memory details** pane.

      ```plaintext
      | Flash | PROGRAM_FLASH | Flash | 0x2000 | 0x7000 | FITF_2K_P0.cf
      |-------|---------------|-------|--------|--------|---------------
      | Flash | NVM_region    | Flash2| 0x76800| 0x4000 | FITF_2K_P0.cf
      | Flash | FRESCALE_PROD_DATA | Flash3 | 0x76800| 0x4000 | FITF_2K_P0.cf
      ```

      Figure 33. Configuring external storage method

      - To select internal flash storage method, as shown in Figure 34, configure the fields in the **Memory details** pane.
7. Clean and build the project. Flash the project on the FRDM-KW36 board. Now, you have programmed and configured the OTAP client software on your board. You can communicate to a server and request for a software update.

5.2 Creating a Temp Coll - OTAP S - record image to update the software

1. Install the Temp Coll-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 appears, as shown in Figure 35. click the Copy button to clone the original example.

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, as shown in Figure 36. This step is needed only to build the SREC image file to reprogram the device.

3. Clean and build the project.

4. Deploy the Binaries icon in the workspace. Right-click the `.axf` file and select Binary Utilities -> Create S-Record. The S-Record file will be saved at the Debug folder in the workspace with the `.s19` extension.
5. Save this file in a known location on your smartphone.

5.3 Creating a Temp Coll S-Record image to update the software

1. Open MCUXpresso IDE. In the Quickstart Panel view, click the Import SDK example(s), and the device selection perspective will appear. Click twice on the frdmkw36 icon.

2. In the Examples text box, type temp_coll and select wireless_examples -> bluetooth -> temp_coll -> freertos. Click Finish.
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.

```c
/* 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.

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.
6. 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, as shown in Figure 41.

7. Include the OtaSupport folder and its files in the framework folder. Include the External folder and its files in the framework->Flash folder. This step can be done in the same way as explained in Importing the OTAP service and framework services into the Temp Coll example.

8. Clean and build the project.

9. Deploy the Binaries icon in the workspace. Right-click the .axf file and select Binary Utilities -> Create S-Record. The S-Record file will be saved at the Debug folder in the workspace with the .s19 extension.

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

5.4 Testing the Temp Coll-OTAP software

Figure 42 exemplifies the testing case of this section. The FRDM-KW36 contains the OTAP client software. The OTAP client will request a software update from the OTAP server, the smartphone. This software image is the Temp Coll-OTAP demo. The FRDM-KW36 at this point has been updated and can handle all the incoming communication from a Temperature Sensor Example or the OTAP server. To demonstrate that you can continue updating the software of the KW36 device, you can connect the Temp Coll-OTAP to an OTAP server and request a software update that only contains the Temp Coll example. From this point, you cannot continue updating the software since the OTAP service was not included in the last software upgrade, demonstrating the importance of including OTAP in the software update sent over the air. This example was designed to understand the key points of the OTAP integration. However, the main purpose of this application note is to create software updates that include the OTAP service and continue upgrading and improving the KW36 device.
1. Open the IoT Toolbox App and select the OTAP demo. Click the SCAN to start scanning for a suitable advertiser.
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 will be displayed on your smartphone.
4. Click the **Open** button and search for the **Temp Coll-OTAP** SREC file.

5. Click the **Upload** button to start the transfer. Wait until the confirmation message is displayed.
6. Connect your FRDM-KW36 with any serial terminal software. Wait few seconds until the OTAP bootloader has finished programming the new image. The Temp Coll-OTAP application will start automatically, when the RGB LED will blink, and a welcome message will be displayed on the serial terminal.
7. You can press the SW2 button on the FRDM-KW36 board to change the GAP role of the device or SW3 to start scanning or advertising. Verify that the device can be detected by both, Temperature Sensor and OTAP applications. To connect with a Temperature Sensor, start the scanning. You can program the Temperature Sensor SDK example on another FRDM-KW36 board, disabling low power, pairing and bonding macros at preinclude file. This example is located at `wireless_examples -> bluetooth -> temp_sens`. The RGB will blink in blue when it is scanning. To connect with OTAP Server, start the advertising, when the RGB is blinking in red. When the device is advertising, it will be named as `NXP_OTAT`. 
8. Connect the Temp Coll-OTAP device with the OTAP smartphone application, when the RGB will be ON in green. Update the software using the Temp Coll SREC file using the OTAP service.

9. Press the SCAN button, SW2, on the FRDM-KW36 board to start scanning. Connect the device with a temperature sensor and verify that it works as expected. Now, you can’t continue updating the software, as OTAP service was not added to the software update. It demonstrate the importance of integrating OTAP in the software update.
Figure 48. Temp Coll device connected
Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. 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 or registered trademarks of Arm Limited (or its subsidiaries) 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. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2020. All rights reserved.

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

Date of release: 09/2020
Document identifier: AN12970