AN12775

将 OTAP 客户端服务集成到 Bluetooth LE 外围设备中

版本 0 — 2020 年 3 月 11 日

应用笔记

目录

1 前言

恩智浦自定义的 Bluetooth LE 空中升级（OTAP）服务为开发人员提供了一个升级 MCU 固件的解决方案。它消除了 OTAP 客户端（需要重新编程的设备）与 OTAP 服务器（包含新的固件的设备）之间的电缆或其它物理连接。

使用 OTAP 服务的最佳方法是，将其集成到 Bluetooth LE 应用程序中，这样，您就可以根据需要对设备进行多次编程。

本文档面向已经熟悉 OTAP 软件的开发人员。

2 OTAP 客户端软件的基础

OTAP 更新过程中的闪存管理描述了 FRDM-KW36 的 SDK 包中所含 OTAP 客户端软件的实际使用过程。

集成 OTAP 服务的优点解释了将 OTAP 客户端软件集成到应用程序中的重要性，以及它将实现什么。

2.1 OTAP 更新过程中的闪存管理

1. KW36 的 Flash 划分为如下部分：
   - 一个 256 KB 的程序闪存阵列（P-Flash），并将其划分为 2 KB 的扇区，闪存地址范围为 0x0000_0000 至 0x0003_FFFF。
   - 一个 256 KB 的 FlexNVM 闪存阵列，2 KB 的扇区大小，地址范围从 0x1000_0000 到 0x1003_FFFF。
   - 地址范围从 0x0004_0000 到 0x0007_FFFF 为 FlexNVM 的别名区。在别名区的地址范围上，进行写入或读取，将对应的修改或返回 FlexNVM 的内容。

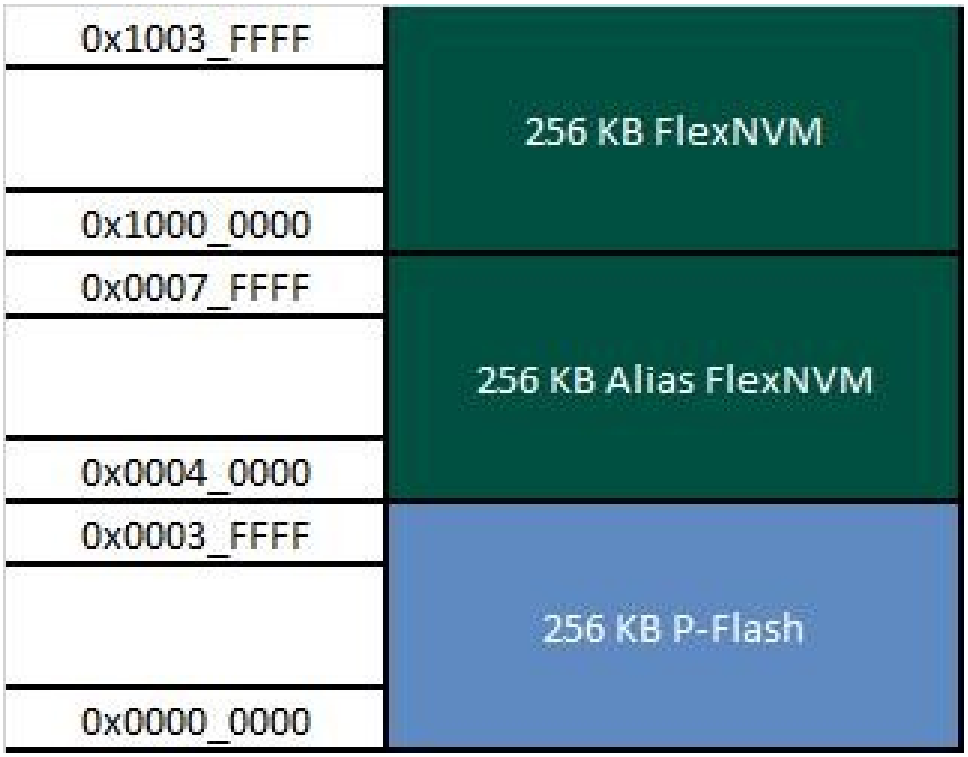


图 1. MCU 片上内存

2. OTAP 应用程序将闪存区划分为两个独立的部分：OTAP bootloader 程序和 OTAP 客户端。OTAP bootloader 程序会验证 OTAP 客户端是否有新的镜像可用于重新编程设备。OTAP 客户端软件提供了自定义的 Bluetooth LE 服务， 实现 OTAP 客户端设备与包含新镜像文件的 OTAP 服务器之间的通信。因此，OTAP 客户端设备需要编程两次，首先下载 OTAP bootloader 程序，然后下载支持 OTAP 客户端的 Bluetooth LE 程序。在同一设备中共存两个不同固件，是通过将不同固件存储在不同的闪存区域实现的，最终由链接器文件实现。在 KW36 中，bootloader 程序占用 8 KB 的闪存，地址从 0x0000_0000 到 0x0000_1FFF，因此，OTAP 客户端程序和其它部分将使用其余的闪存。

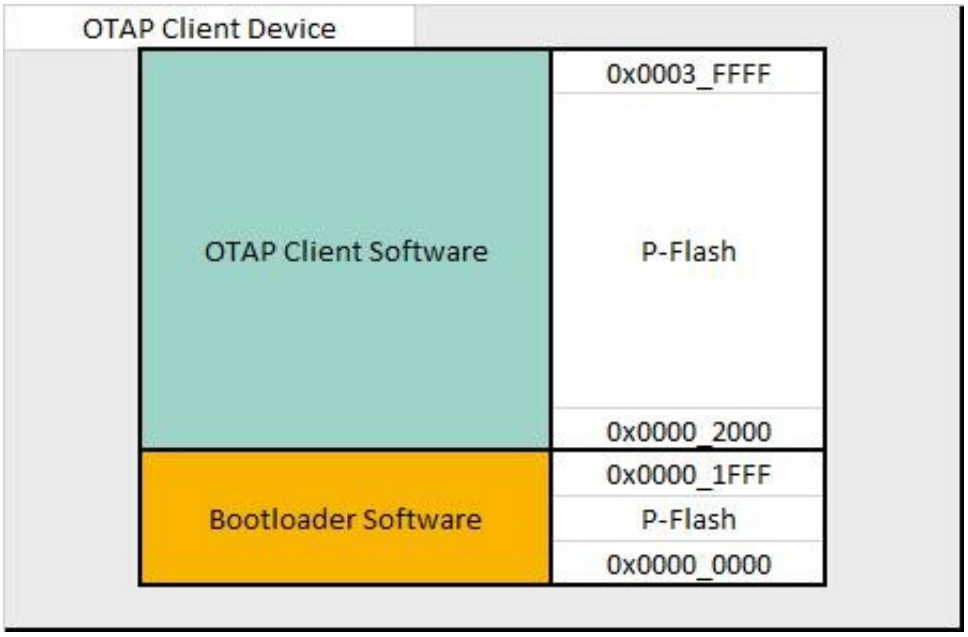


图 2. OTAP 客户端软件

3. 为了给客户端设备创建一个新的镜像文件，开发人员需要明确：代码应该以 8 KB 的偏移进行存储，因为前面的闪存是为 bootloader 程序保留的（使用链接器实现）。新应用程序还应该包含相应地址的 Bootloader Flag，以便正常工作。

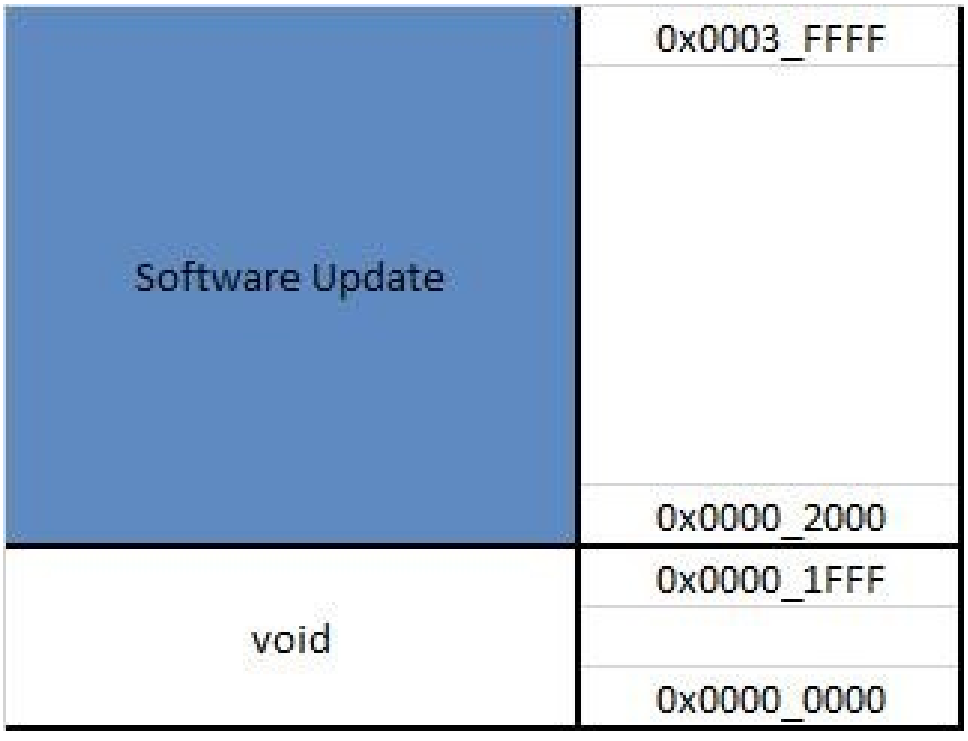


图 3. 软件更新

4. 在连接状态下，OTAP 服务器通过 Bluetooth LE 将镜像数据包（称为数据块）发送到 OTAP 客户端。OTAP 客户端设备可以将这些数据块存储在外部 SPI 闪存（第一个实例存储到了此位置，仅在 FRDM-KW36 板上可用）或片上 FlexNVM 存储器中。在 OTAP 客户端软件中可选择固件的目标存储区域。

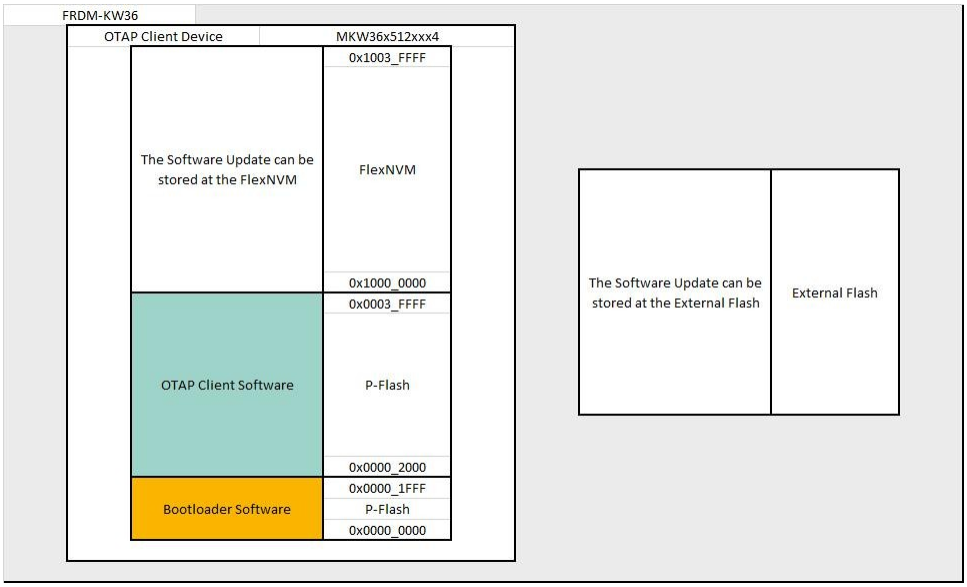


图 4. 软件更新的存储

5. 当完成镜像的传输，且所有数据块都从 OTAP 服务器送达 OTAP 客户端后，OTAP 客户端软件将在被称为 Bootloader Flag 的闪存中写入标记信息，例如镜像更新的来源（外部闪存或 FlexNVM），然后复位 MCU 以执行 OTAP bootloader 代码。OTAP bootloader 读取 Bootloader Flag 以获取设备升级所需的信息，并触发用新的应用程序固件重新编程 MCU 的命令。

   由于新的应用程序是以 8 KB 的偏移构建的，OTAP bootloader 程序从地址 0x0000_2000 开始编程设备，OTAP 客户端程序将被新镜像固件覆盖。然后，OTAP bootloader 触发执行新镜像固件的命令。如果新镜像不包含 OTAP 服务，由于缺失 OTAP 功能，将无法再对设备进行编程。这将在集成 OTAP 服务的优点中进一步讨论。

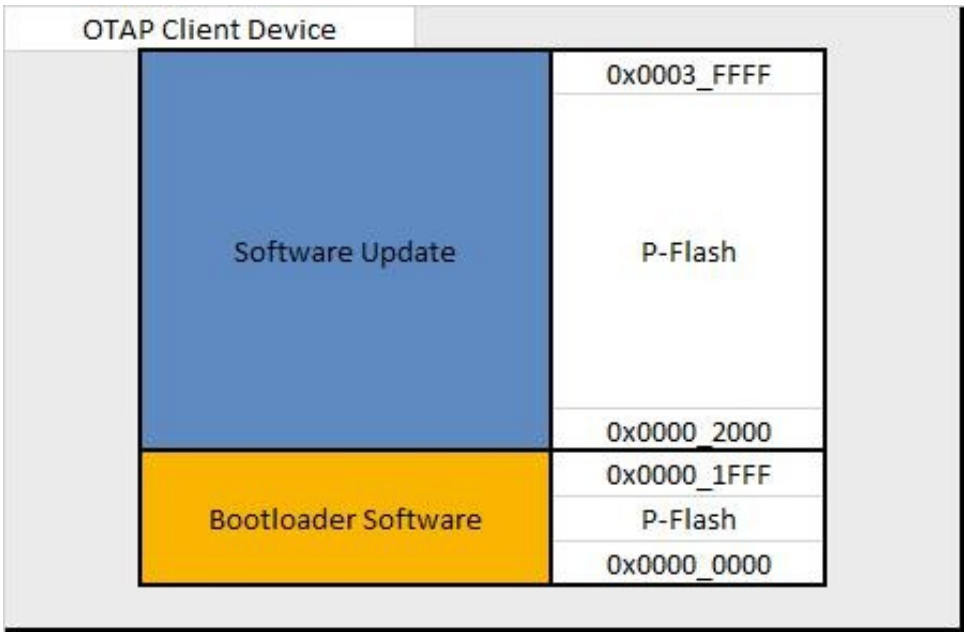


图 5. 软件更新结束后的闪存内容

注意

实际上，如果使用内部 Flash 存储要更新的固件，OTAP 客户端软件和软件更新镜像之间的边界并不是确切位于 P-Flash 和 FlexNVM 闪存区域的边界上。这些数值可以根据链接器的设置而变化。可以根据链接文件，查看项目中的有效闪存地址。

2.2 集成 OTAP 服务的优点

如 OTAP 更新过程中的闪存管理所说明的，OTAP 客户端软件是一个 single-programming 的示例应用。假设 OTAP 客户端设备使用 OTAP 客户端软件编程，该设备请求更新一个没有集成 OTAP 的固件，例如心率传感器（HRS）。OTAP 服务器发送给 OTAP 客户端的镜像是 HRS。在重新编程之后，OTAP 的客户端设备就变成了心率传感器，HRS 无法与 OTAP 服务器通信并请求再次更新。但是如果 HRS 镜像也包含了 OTAP 客户端服务，则该设备可以请求再次更新，例如带 OTAP 服务的优化的葡萄糖传感器示例。由于葡萄糖传感器软件包含 OTAP 客户端服务，设备可以从 OTAP 服务器请求软件更新。这样，开发人员可以根据需求多次升级软件。换句话说，为了将来能够在 OTAP 客户端设备上升级软件，空中升级后的应用程序应该支持 OTAP 服务。

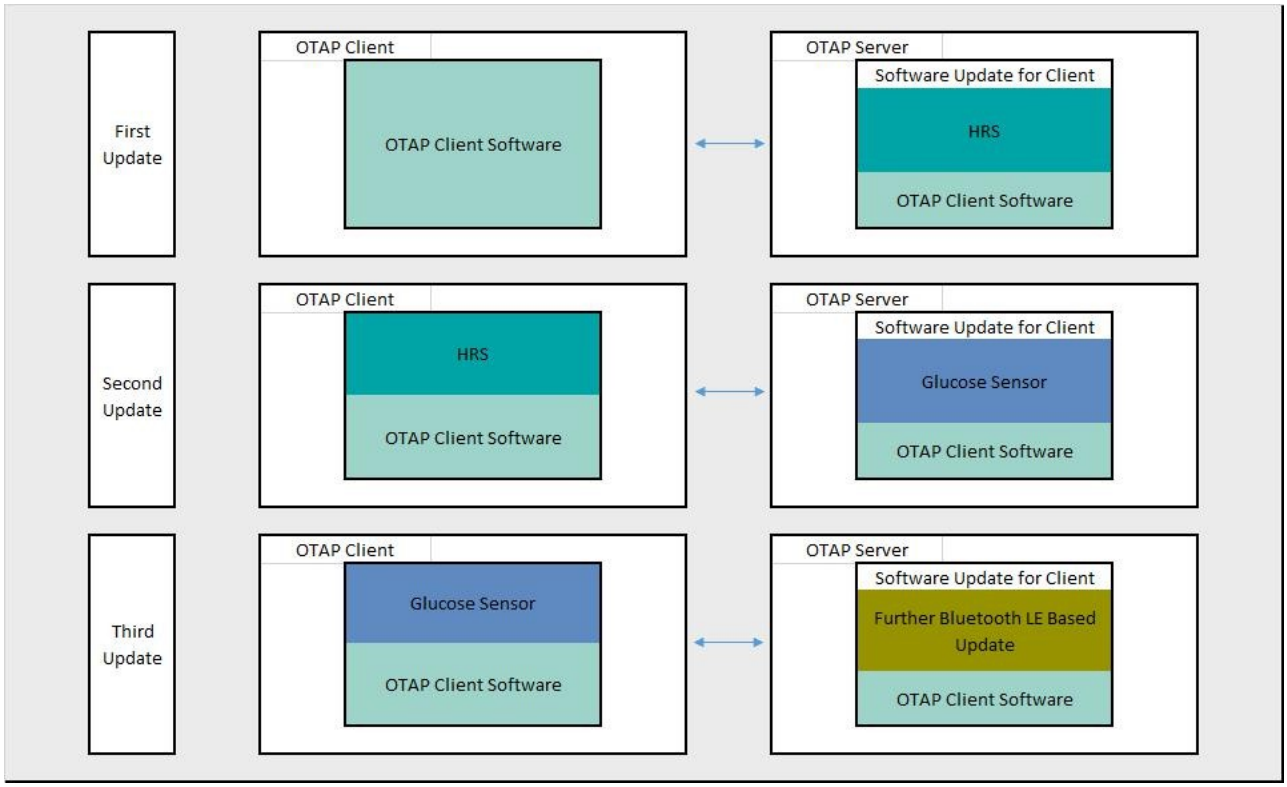


图 6. 集成 OTAP 功能示例

本文旨在为向 Bluetooth LE 应用程序添加 OTAP 服务提供一个指南。

3 准备工作

本文档与集成 OTAP 服务的功能演示实例一起提供。该示例基于心率传感器项目，可在 FRDM-KW36 SDK 中获得，并在 MCUXpresso IDE 平台上开发。以下是完成 HRS-OTAP 集成演示所需要的。

- MCUXpresso IDE v11.0.0 或更高版本
- frdm-kw36 SDK
- HRS-OTAP 演示包
- FRDM-KW36 开发板
- 安装了应用 IoT Toolbox 的智能手机（Android 和 iOS 均可）

3.1 软件开发工具包的下载与安装

本节提供了下载 FRDM-KW36 的 SDK（软件开发工具包）所需的所有步骤。

1. 前往 MCUXpresso SDK Builder。
2. 点击“Select Development Board”。登录或注册帐户。
3. 在“Search by Name”文本框中搜索“FRDM-KW36”开发板。然后点击推荐的板子，点击“Build MCUXpresso SDK”。

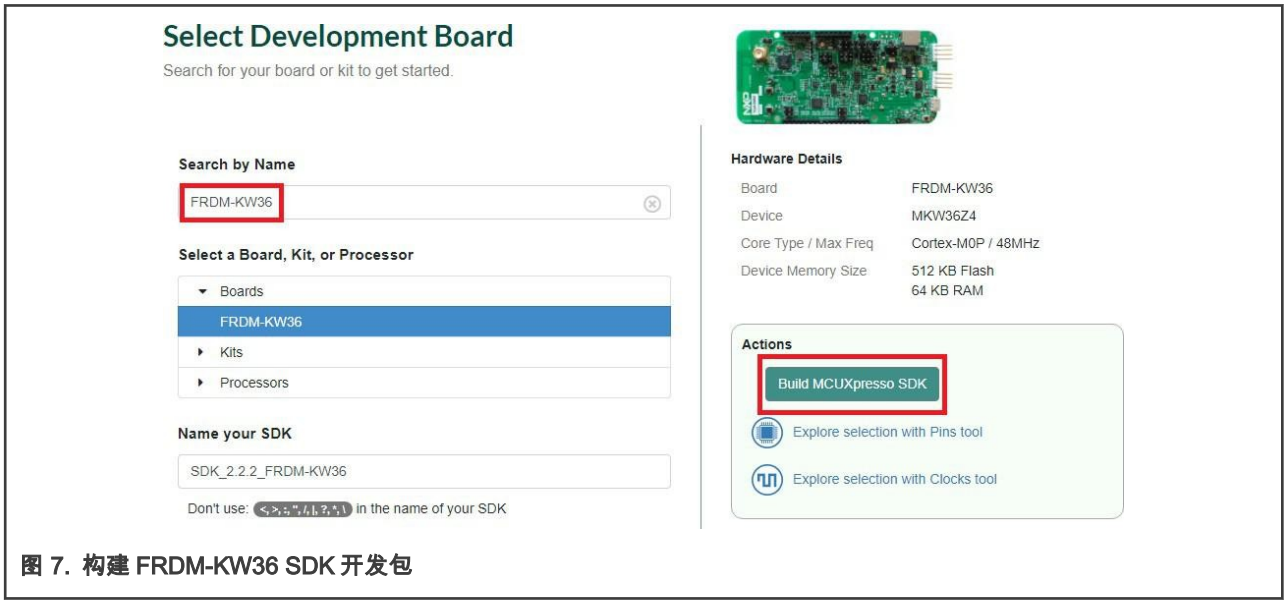


图 7. 构建 FRDM-KW36 SDK 开发包

4. 在“ToolChain/IDE”下拉框中选择 MCUXpresso IDE。选择支持的操作系统，并提供名称以标识 MCUXpresso 仪表板中的软件包。

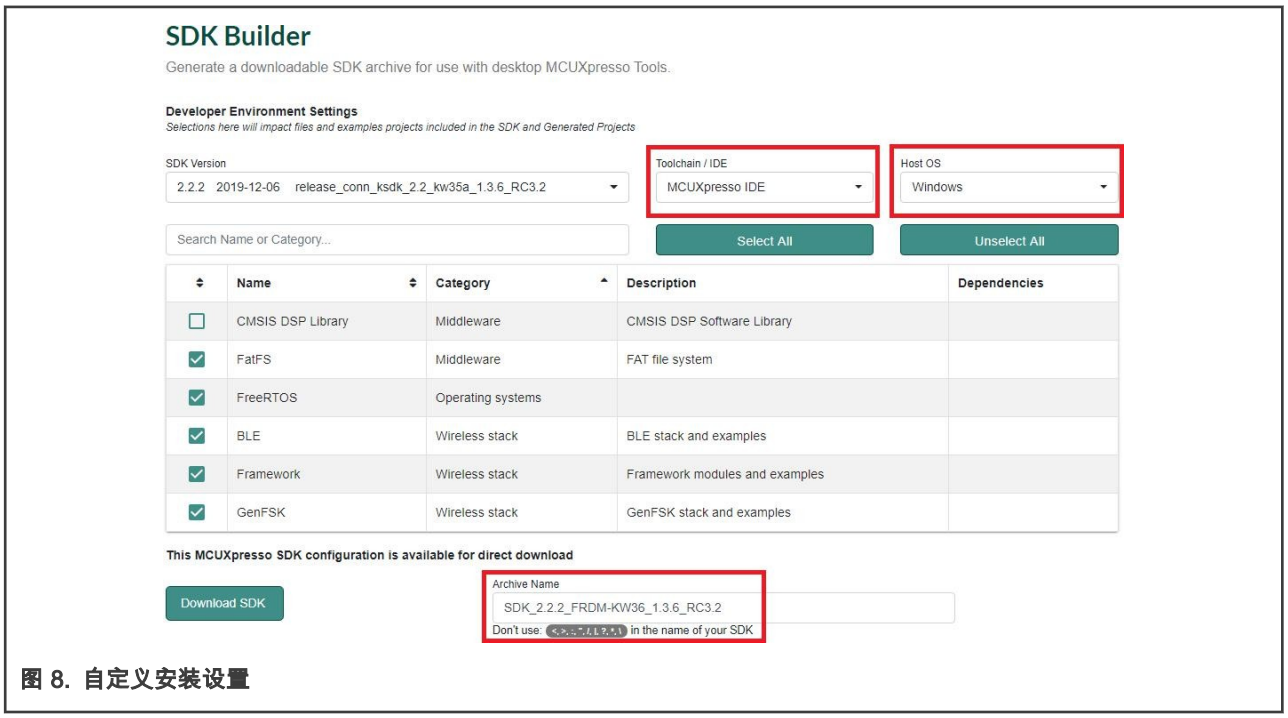


图 8. 自定义安装设置

5. 单击 “Download SDK” 按钮。系统可能需要几分钟才能将软件包打包到 MCUXpresso 网页上。阅读并接受许可协议，SDK 会自动下载到您的电脑上。
6. 打开 MCUXpresso IDE，将 FRDM-KW36 SDK 的 zip 文件拖拽到“Installed SDKs”视图框中。

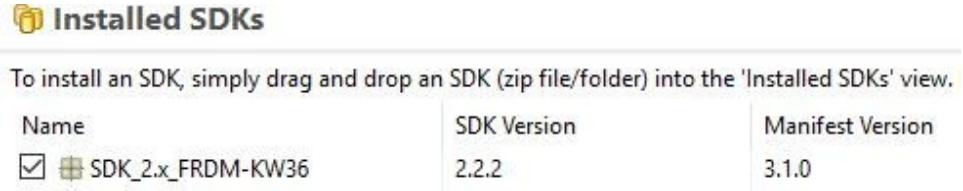


图 9. 将 SDK 开发包导入到 MCUXpresso IDE

至此，您已经下载并安装了 FRDM-KW36 板的 SDK 软件包。

4 自定义一个基于 Bluetooth LE 的 Demo 来集成 OTAP 服务

下面的步骤描述了通过定制的 SDK 中的 Bluetooth LE 例程来集成 OTAP 服务的过程。本指南使用心率传感器例程（HRS）作为例子，对于 Bluetooth LE SDK 中的其它例程，一些步骤可能有所不同。

4.1 将 OTAP 服务和框架服务导入 HRS

OTAP 客户端软件使用了 HRS 中没有的一些框架功能。因此 OTAP 集成的第一步是，比较例程工程和 OTAP 客户端工程之间哪些文件夹和文件是不同的，然后启用这些功能。HRS（左）和 OTAP 客户端（右）之间的比较如 图 10 所示。

图 10. Source tree 比较

在 OTAP 客户端中但不在 HRS 中的文件夹和文件，必须导入到 HRS 项目。以下步骤用于将文件夹和源文件导入到项目中。

1. 展开工作区中的“Bluetooth”和“framework”文件夹。选择更新所需的文件夹，单击鼠标右键。选择“New->Folder”，会出现一个新的窗口。导入文件夹名称，其与源目录中缺少的文件夹同名。

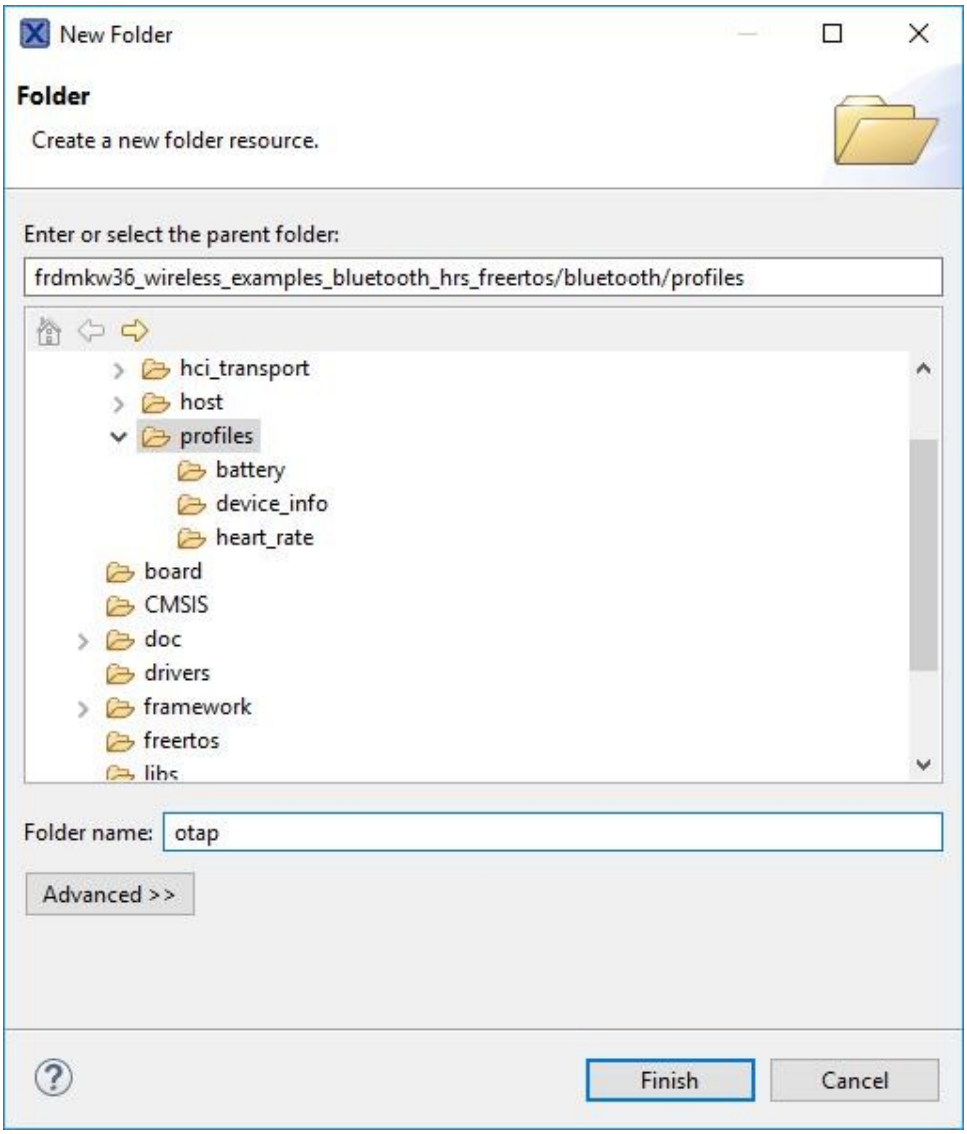


图 11. 创建 Bluetooth 和 Framework 文件夹

2. 对左侧文件夹重复步骤 1，结果如 图 12 所示。

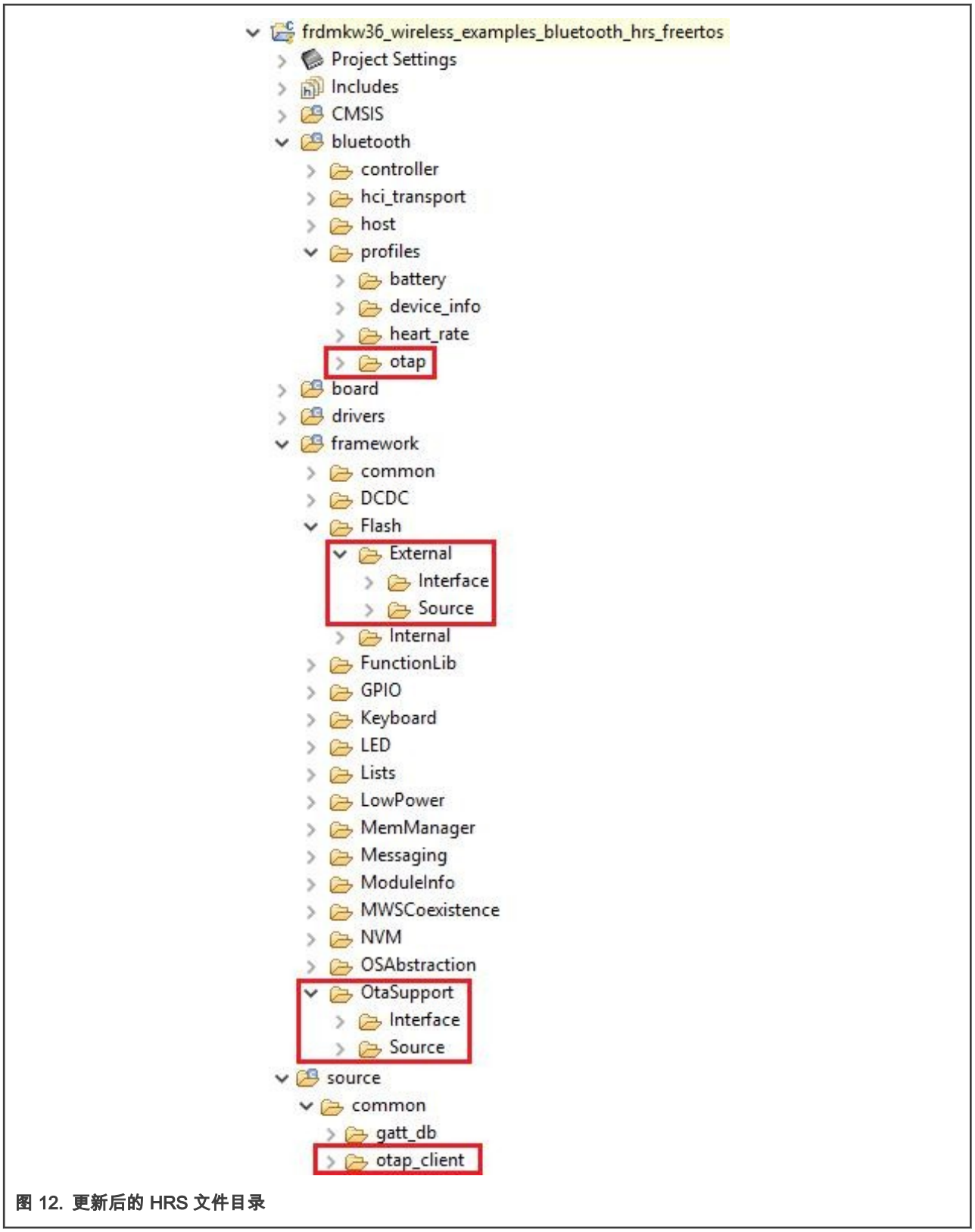


图 12. 更新后的 HRS 文件目录

3. 从 OTAP 客户端复制最近创建的文件夹中的文件，并将其保存到例程中。确保在同一个文件夹名的文件夹下，HRS 端所有文件都与 OTAP 客户端的文件一样，对于本例，这些文件如下所示。
   - “bluetooth->profiles->otap”文件夹中的“otap_interface.h”和“otap_service.c”。

- “framework->Flash->External->Interface”文件夹中的“Eeprom.h”。
- “framework->Flash->External->Source”文件夹中的 Eeprom 源文件。
- “framework->OtaSupport->Interface”文件夹中的“OtaSupport.h”。
- “framework->OtaSupport->Source”文件夹中的“OtaSupport.c”。
- “linkscripts”文件夹中的“main_text_section.ldt”。
- “source->common->otap_client”中的“otap_client.h”和“otap_client.c”。

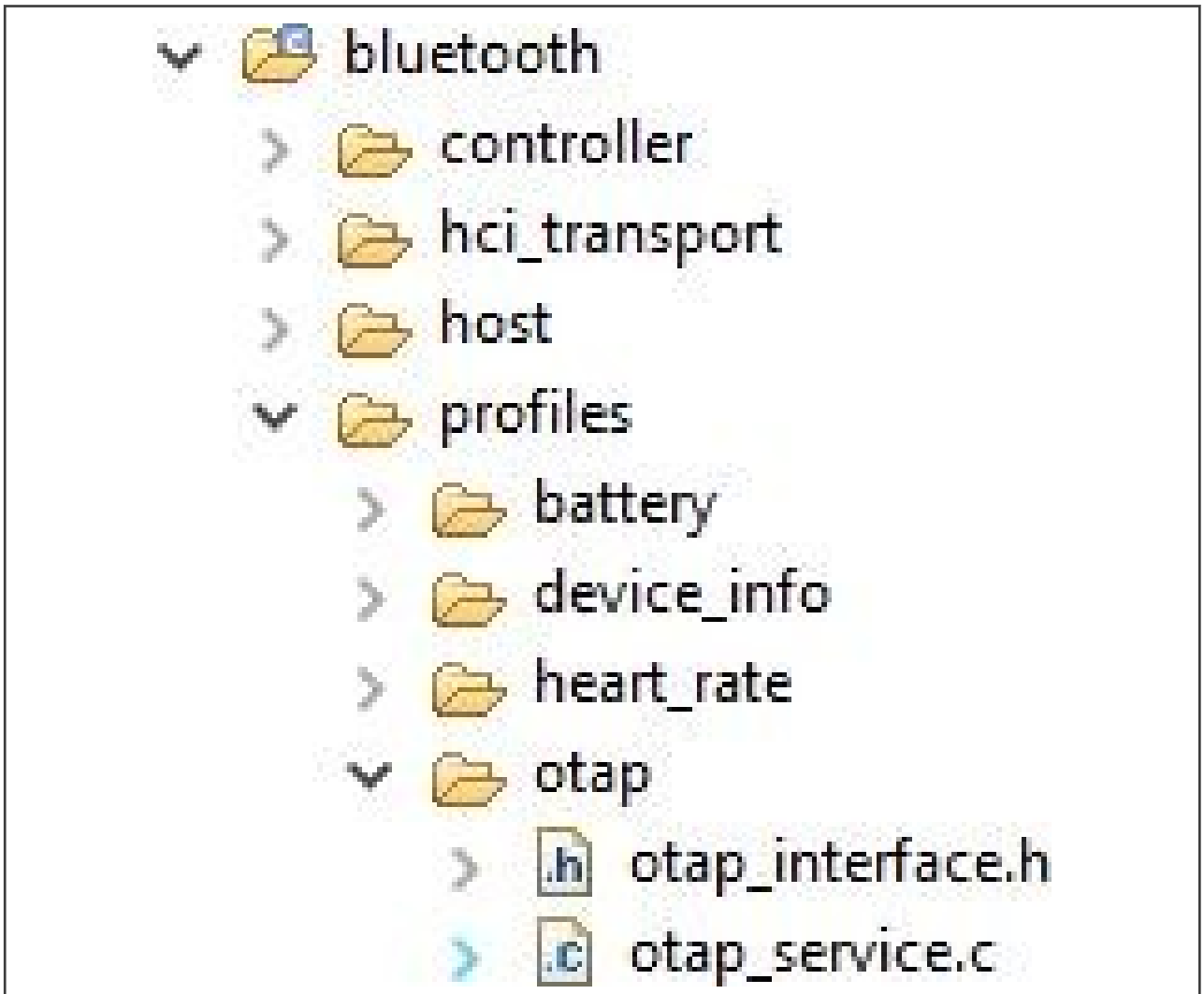


图 13. 集成到 HRS 项目的 OTAP 文件

4. 进入 MCUXpresso IDE 中的“Project->Properties”。进入“C/C++ Build->Settings->Tool Settings->MCU C Compiler->Includes”。单击“Include paths”文本框旁边的图标（参见 图 14 ）。之后会出现一个新窗口，然后单击“Workspace” 按钮。

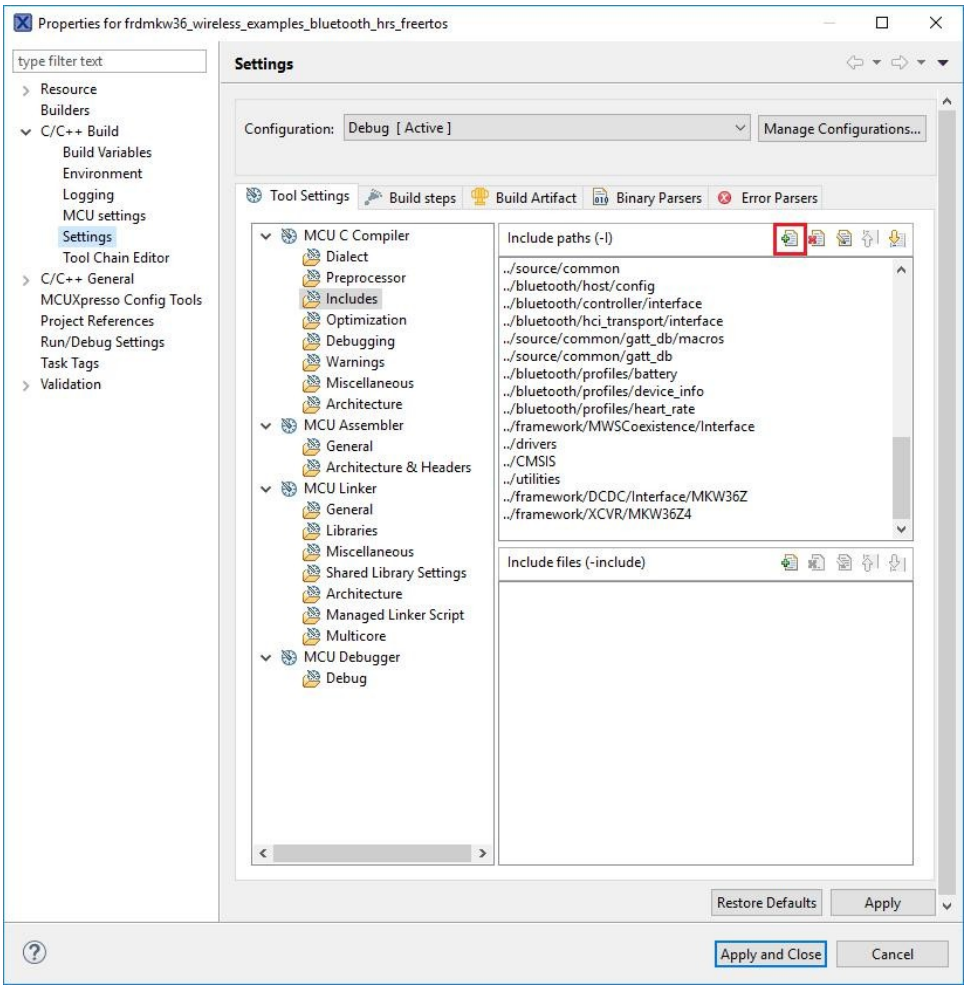


图 14. Include paths 视图

5. 在文件夹选择窗口设置目录树。选择以下文件夹并单击“OK”按钮保存更改：
   - bluetooth->profiles->otap
   - framework->Flash->External->Interface
   - framework->OtaSupport->Interface
   - source->common->otap_client

   确保将这些路径导入到“Include paths”中。

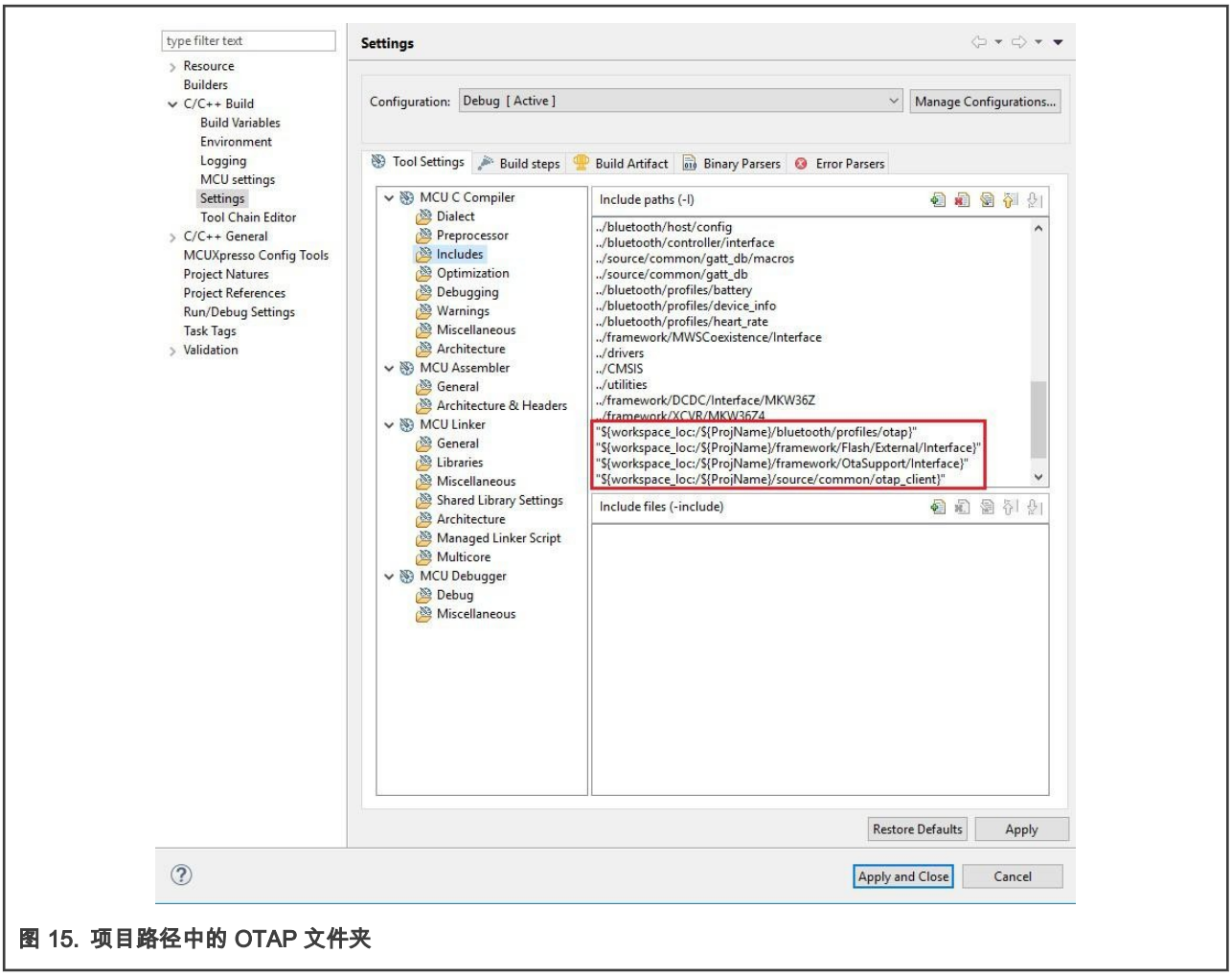


图 15. 项目路径中的 OTAP 文件夹

此时，您已经成功将 OTAP 客户端的蓝牙服务和框架服务导入到了 HRS 项目。

4.2 对源文件的主要修改

一旦将 OTAP 客户端的文件夹和文件导入到自定义例程中，务必比较 OTAP 客户端的源文件与 Bluetooth LE 例程之间的差异，并添加所需的代码以集成 OTAP 服务。下面会介绍应该关注的主要方面。

4.2.1 app_preinclude.h

“app_preinclude.h”文件包含了许多配置项目功能的预处理器指令，如低功耗是否启用、DCDC 配置、Bluetooth LE 安全设置和硬件电路板相关宏定义。OTAP 客户端软件需要一些特别的设置，这些设置不包括在其它 Bluetooth LE SDK 例程中。在软件更新中需要增加以下定义：

- gEepromType_d
- gEepromParams_WriteAlignment_c
- gOtapClientAtt_d

对于 OTAP-HRS demo，设置以下值：

1. **gEepromType_d**：定义使用 FRDM-KW36 板上的外部闪存 AT45DB041E（默认值）还是片上存储器 FlexNVM。也可以使用其它用于开发板的存储设备（请参阅位于 framework/Flash/External/Interface 的 Eeprom.h 头文件中的 EEPROM 列表）。

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

2. **gEepromParams_WriteAlignment**：定义用于编程的软件更新的 offset。不要修改该值。

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

3. **gOtapClientAtt_d**：OTA 更新设置 ATT 传递方式。必须设置为 1。

```
#define gOtapClientAtt_d 1
```

4.2.2 app_config.c

“app_config.c”源文件定义了一些结构，用于配置广播和扫描参数、数据。对于设备上的各个服务，它还可设置安全认证要求。

广播数据显示了 Bluetooth LE 广播设备（HRS-OTAP）包含的服务。Bluetooth LE 扫描设备通过此数据，过滤不包含所需服务的广播设备。因此，必须将 OTAP 客户端服务添加到广播数据，以便向 OTAP 服务器声明该服务的可用性。该交互通过扫描应答来完成，如下面的代码所示。

```
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
};
```

注意

由于在扫描应答中声明了 OTAP 客户端服务，因此必须确保将 OTAP 服务器设备配置为主动扫描。IoT Toolbox App 已经完成了该操作，但是 OTAP 服务器的 SDK 例程并未完成，可以在包含“gScanParams”结构体的“app_config.c”文件中，更改 OTAP 服务器的 SDK 例程的扫描设置。

此外，还需要满足 OTAP 服务的安全认证要求，这是在“gapServiceSecurityRequirements_t”结构体中完成的，可以根据需要自定义这些参数。HRS-OTAP demo 设置了以下参数，请关注 OTAP 服务部分：

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

```
.minimumEncryptionKeySize = gDefaultEncryptionKeySize_d
},
.serviceHandle = service_battery
},
{
.requirements = {
.securityModeLevel = gSecurityMode_1_Level_3_c,
.authorization = FALSE,
.minimumEncryptionKeySize = gDefaultEncryptionKeySize_d
},
.serviceHandle = service_device_info
}
};
```

最后的修改是增加“deviceSecurityRequirements”结构体中服务数量的索引：

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

4.2.3 gatt_db.h 和 gatt_uuid128.h

“gatt_db.h”头文件包含属性列表，这些属性组合在一起构成 GATT 服务器（HRS-OTAP 客户端设备）的配置文件。本指南最重要的步骤是将 OTAP 客户端属性列表添加到设备的数据库中。建议打开 OTAP 客户端 SDK 例程和您的 Bluetooth LE demo，以便比较两个 GATT 数据库的区别。图 16 展示了 OTAP 客户端的数据库。

```
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)
```

图 16. OTAP 客户端服务

HRS-OTAP demo 的“gatt_db.h”数据库构建的配置文件具有如 图 17 所示的体系结构。

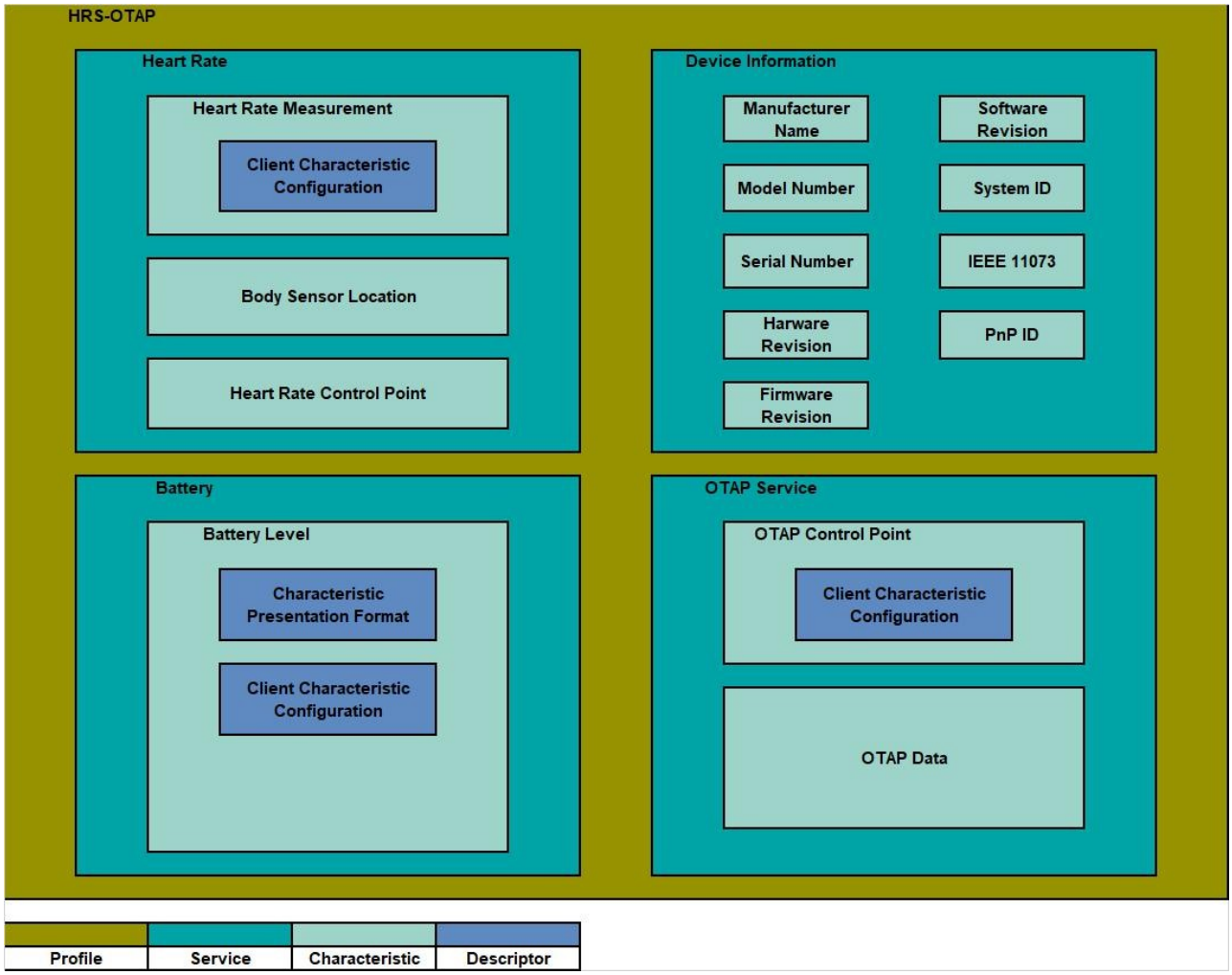


图 17. HRS-OTAP 配置文件

“gatt_uuid128.h”头文件包含所有“自定义”的 UUID 定义及其分配。在最初的 HRS SDK 项目中，“gatt_uuid128.h”不包含其它定义，因为心率和电池服务是 Bluetooth SIG 采用的标准服务。然而，开发人员需要将 OTAP 服务及其属性指定为 128 – UUID。图 18 演示了如何实现 OTAP 服务的 128-UUID 分配。

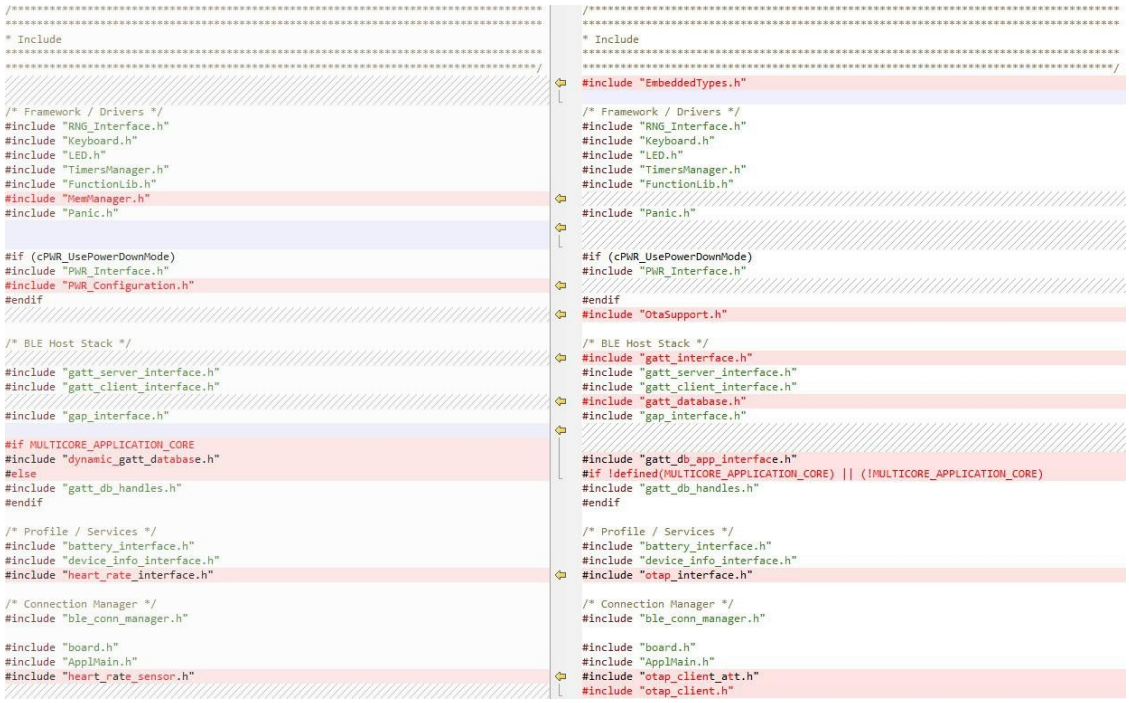
```
/* BLE Over The Air Programming - Firmware Update */
UUID128(uuid_service_otap,              0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA, 0x50, 0x55, 0xFF, 0x01)
UUID128(uuid_char_otap_control_point,   0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA, 0x51, 0x55, 0xFF, 0x01)
UUID128(uuid_char_otap_data,            0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA, 0x52, 0x55, 0xFF, 0x01)
```

图 18. HRS-OTAP 128-UUID 定义

4.2.4 heart_rate_sensor.c

“heart_rate_sensor.c”是应用程序级别的主要源文件。在创建连接的所有过程中，将对设备所执行的所有进程进行管理。以下步骤用于集成 OTAP 服务。

1. 合并缺少的“#include”预处理器指令，以引用项目中的 OTAP 文件（otap_client_att.h 除外）。请参见 图 19，它是 HRS（左）和 OTAP 客户端程序（右）之间的比较。此步骤取决于采用的例程，因为它可能具有与本示例不同的文件。结果如图 20 所示，合并前（HRS 左）、合并后（HRS-OTAP 右）。



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

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


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

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

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

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

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

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

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

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

#include "Panic.h"


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

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


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

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

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

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

图 19. 比较 HRS（左）和 OTAP（右）的 includes

图 20. 将 OTAP 文件合并到项目中. 合并前（HRS 左）和合并后（HRS-OTAP 右）

2. 添加 OTAP 客户端使用的函数原型和全局变量。请对比 HRS（左）和 OTAP（右），如 图 21 所示。如在第一步中提到的，这取决于具体例程。结果类似 图 22 所示。

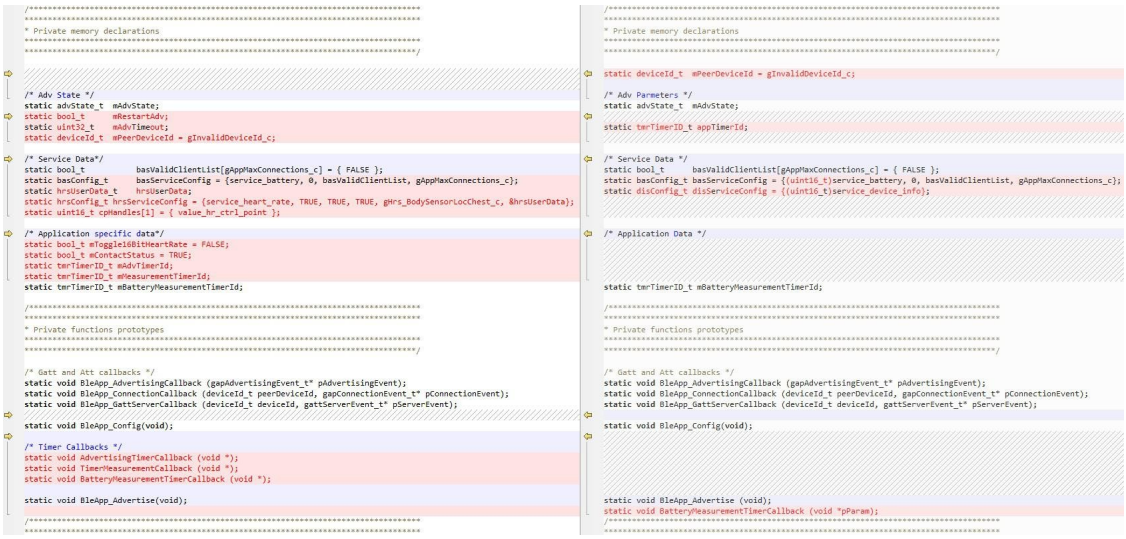


图 21. 比较 HRS（左）和 OTAP（右）的函数原型

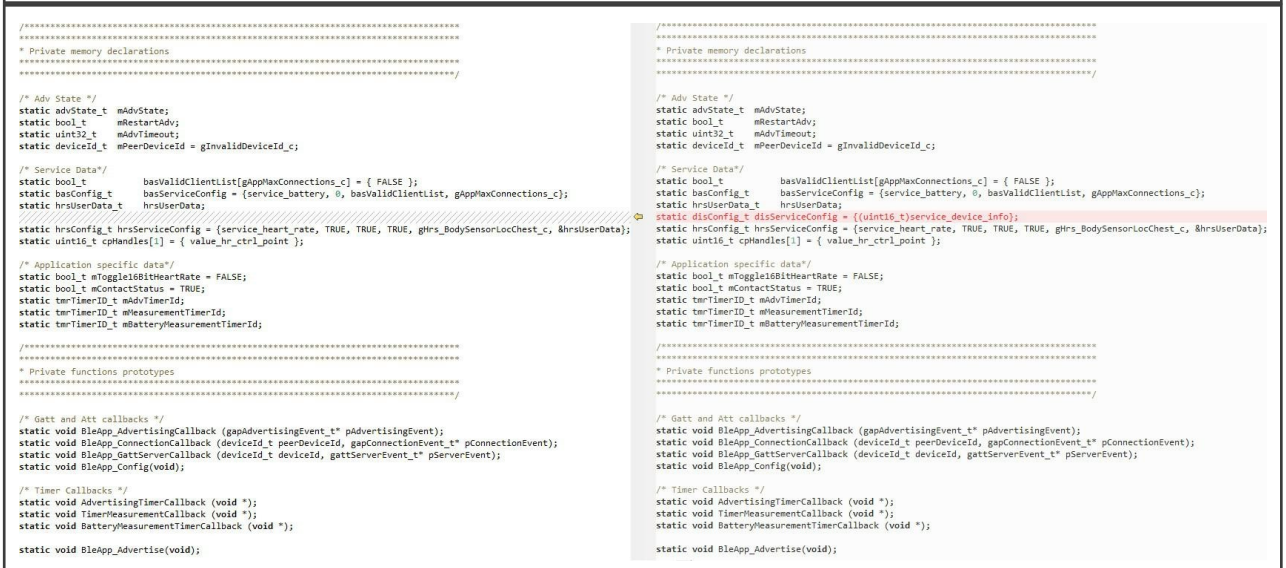


图 22. 将 OTAP 函数原型合并到项目中. 合并前（HRS 左）和合并后（HRS-OTAP 右）

3. 找到“BleApp_Config”函数。“BleApp_Config”函数用于配置设备的 GAP（HRS-OTAP 是一个外围设备），注册相关属性，为在数据库上构建的服务做准备，并分配一些应用计时器。通过添加“OtapClient_Config”和“Dis_Start”函数来初始化这些服务。请参阅以下代码。

```
/* Start services */
    hrsServiceConfig.sensorContactDetected = mContactStatus;
#if gHrs_EnableRRIntervalMeasurements_d
    hrsServiceConfig.pUserData->pStoredRrIntervals = MEM_BufferAlloc (sizeof (uint16_t) *
gHrs_NumOfRRIntervalsRecorded_c);
#endif
    Hrs_Start(&hrsServiceConfig);
    basServiceConfig.batteryLevel = BOARD_GetBatteryLevel();
    Bas_Start(&basServiceConfig);    (void)Dis_Start(&disServiceConfig);
if (OtapClient_Config() == FALSE)
{
    /* An error occured in configuring the OTAP Client */
```

```
    panic(0,0,0,0);
}
```

4. 找到“BleApp_ConnectionCallback”。每当发生连接事件，例如连接成功或断开连接时，都会触发连接回调。
   a. 连接成功。包括“OtapCS_Subscribe”和“OtapClient_HandleConnectionEvent”函数，通过以下代码实现。

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

    mPeerDeviceId = peerDeviceId;

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

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

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

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

   b. 断开连接。包括“OtapCS_Unsubscribe”和“OtapClient_HandleDisconnectionEvent”函数，通过以下代码实现。

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

    mPeerDeviceId = gInvalidDeviceId_c;

    /* Stop Timers*/
    TMR_StopTimer(mMeasurementTimerId);
```

```
    TMR_StopTimer(mBatteryMeasurementTimerId);
    OtapClient_HandleDisconnectionEvent(peerDeviceId);
if (cPWR_UsePowerDownMode)
    /* UI */
    Led1Off();

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

5. 找到“BleApp_GattServerCallback”，它管理来自客户端设备的通信交互。添加需要由 OTAP 客户端处理的 GATT 事件（“gEvtAttributeWritten_c”、“gEvtMtuChanged”、“gEvtCharacteristicCccdWritten_c”、“gEvtAttributeWrittenWithoutResponse_c”、“gEvtHandleValueConfirmation_c”和“gEvtError”）。定制的 Bluetooth LE 例程可能会共享一些常见的 GATT 事件，如果是这样的话，需要为每一个属性句柄添加一个条件结构。请关注“gEvtAttributeWritten_c”，参考“HRS control point”和“OTAP control point”的条件结构。

```
case gEvtAttributeWritten_c:
{
   handle = pServerEvent->eventData.attributeWrittenEvent.handle;
   status = gAttErrCodeNoError_c;
   if (handle == value_hr_ctrl_point)
   {
   status = Hrs_ControlPointHandler(&hrsUserData,
   pServerEvent->eventData.attributeWrittenEvent.aValue[0]);
   GattServer_SendAttributeWrittenStatus(deviceId, handle, status);
   }
     else
   {
   OtapClient_AttributeWritten (deviceId,
     pServerEvent->eventData.attributeWrittenEvent.handle,
     pServerEvent->eventData.attributeWrittenEvent.cValueLength,
     pServerEvent->eventData.attributeWrittenEvent.aValue);
    }
}
break;
case gEvtMtuChanged_c:
{
    OtapClient_AttMtuChanged (deviceId,
    pServerEvent->eventData.mtuChangedEvent.newMtu);
}
break;
case gEvtCharacteristicCccdWritten_c:
{
    OtapClient_CccdWritten (deviceId,
    pServerEvent->eventData.charCccdWrittenEvent.handle,
    pServerEvent->eventData.charCccdWrittenEvent.newCccd);
}
break;
```

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

至此，您已经将 OTAP 客户端代码集成到了 HRS 中。

4.3 项目设置和存储配置中的修改

SDK 开发包中的 OTAP 客户端软件包含一些链接器配置，以生成 OTAP Bootloader 软件所需的应用程序 offset，并按照所需的存储方法划分闪存。这些配置不是 HRS demo 的一部分，因此需要将其导入实例工程，以将 OTAP 集成到应用程序中。按照下列步骤，进行项目设置和存储配置。

1. 在项目的源文件夹下找到“app_preinclude.h”文件。
   a. **外部闪存方法**，请将“gEepromType”定义为“gEepromDevice_AT45DB041E_c”（附带的 HRS-OTAP 软件包中的默认设置）。
   b. **内部闪存方法**，请将“gEepromType”定义为“gEepromDevice_InternalFlash_c”。

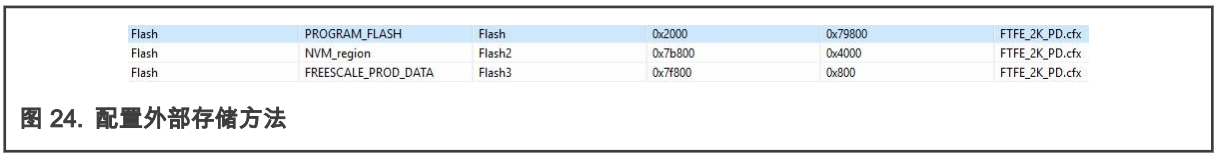
```
/* Specifies the type of EEPROM available on the target board */
#define gEepromType_d           gEepromDevice_AT45DB041E_c
```

图 23. 在 preinclude 文件中配置存储的方法

2. 单击 MCUXpresso 工作区中的 HRS-OTAP demo。

3. 找到 MCUXpresso IDE 中的“Project->Properties”。进入“C/C++ Build->MCU settings”。

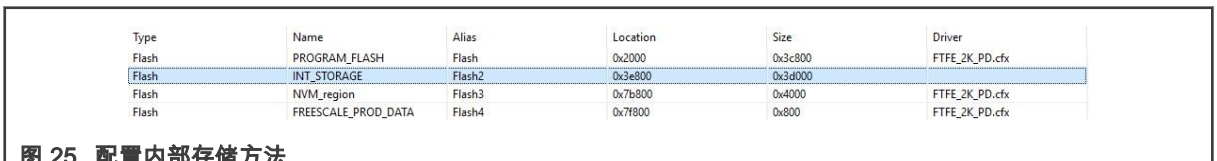
   a. **外部闪存方法**，如 图 24 所示，请配置“Memory details”窗格中描述的字段（附带的 HRS-OTAP 软件包中的默认设置）。



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

图 24. 配置外部存储方法

   b. **内部闪存方法**，如 图 25 所示，请配置“Memory details”窗格中描述的字段。



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

图 25. 配置内部存储方法

4. 清理并构建项目。

至此，基于 Bluetooth LE 的应用程序集成 OTAP 服务全部完成。

4.4 在应用程序上添加低功耗支持功能

为了使 OTAP 支持低功耗，需要考虑以下内容。

1. 必须更改“OTA_PushImageChunk”功能，以禁止设备在将数据写入闪存时进入睡眠，并允许设备在完成进程后返回低功耗模式。在 framework->OtaSupport->Source->OtaSupport.c 文件中找到“OTA_PushImageChunk”函数。进入“OTA_PushImageChunk”代码之前，调用“PWR_DisallowDeviceToSleep”；从函数返回之前，调用“PWR_AllowDeviceToSleep”。

   见以下示例：

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

2. 基于 VLLS 模式开发的深度睡眠模式 5 和模式 8（DSM5 和 DSM8），唤醒例程执行 SW 复位，因此 SFR 的值和程序文本会丢失，必须在退出低功耗状态后恢复原状。热启动回调恢复了程序文本和时钟配置，但是没有恢复外部存储所需的外设 SPI。换句话说，必须在热启动回调中增加 SPI 初始化处理。以下代码可在 board-> board.c 文件的 HRS-OTAP 示例中找到。

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

/* Private type definitions and macros */
ifndef gEepromSpiInstance_c
#define gEepromSpiInstance_c 0
#endif
```

```
static spiState_t mEepromSpiState;

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

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

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

3. 应用程序文件还必须包含低功耗管理所需的 API，根据设备是处于广播状态、连接状态或空闲状态来更改 DSM 模式，并且只要处于空闲状态就能够进入睡眠模式。可以用 HRS-OTAP 应用程序作为参考，用自己的代码来实现低功耗功能。

5 测试 HRS-OTAP Demo

演示 OTAP 集成的示例，使用了以下软件：

- OTAP 客户端 SDK，用于在 FRDM-KW36 板编程。
- HRS-OTAP 示例的 SREC 软件。
- HRS SDK 示例的 SREC 软件。

下面的部分将解释，如何构建本文档中测试所需的软件，开发者可以自己决定哪些软件或步骤是不需要的。

5.1 准备 OTAP 客户端 SDK

1. 将您的 FRDM-KW36 板连接到 PC 上。
2. 在 FRDM-KW36 上编程 OTAP Bootloader，您可以从板子上的以下路径，拖拽预构建的二进制文件：

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

3. 打开 MCUXpresso IDE。单击“Quickstart Panel”视图中的“Import SDK example(s)”选项。

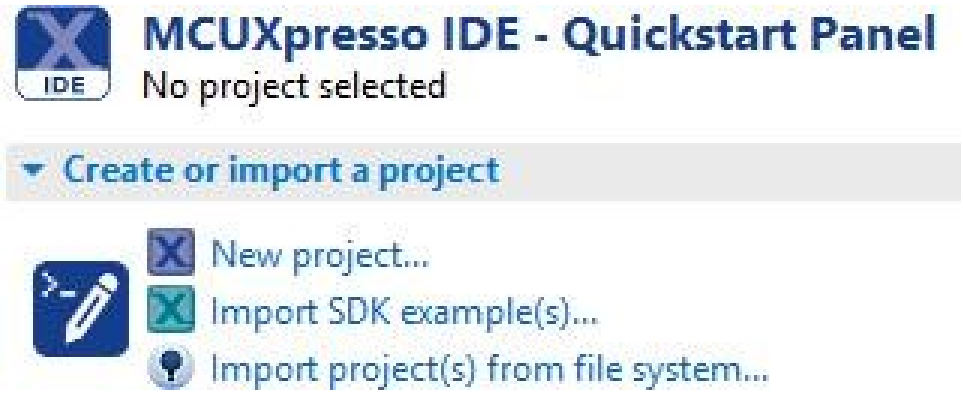


图 26. Quickstart Panel 视图

4. 双击 frdmkw36 图标。

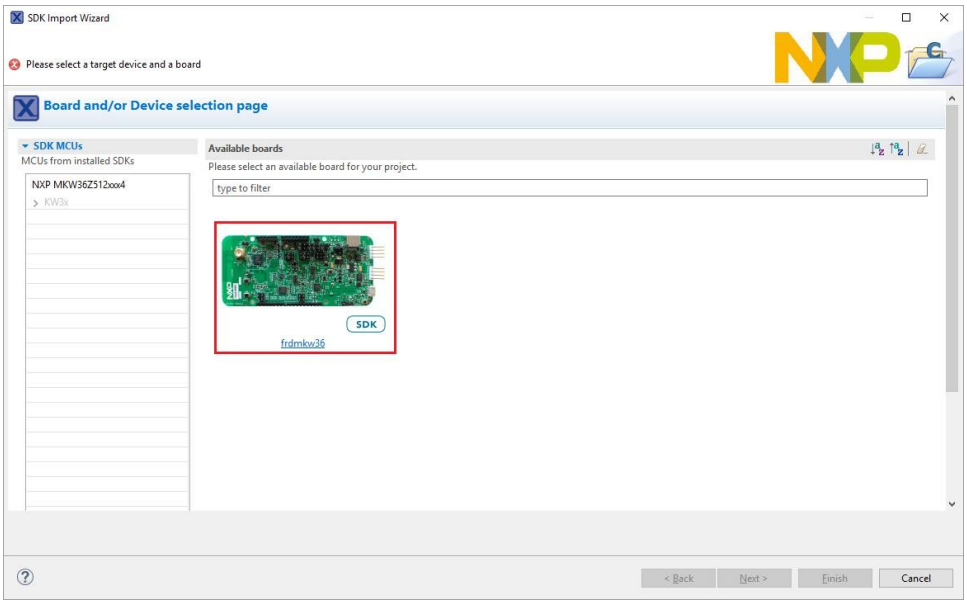


图 27. 设备选择视图

5. 在示例文本框中输入“otac_att”，然后在“wireless_examples-> bluetooth-> otac_att-> freertos”中选择 freertos 项目。点击“Finish”按钮。

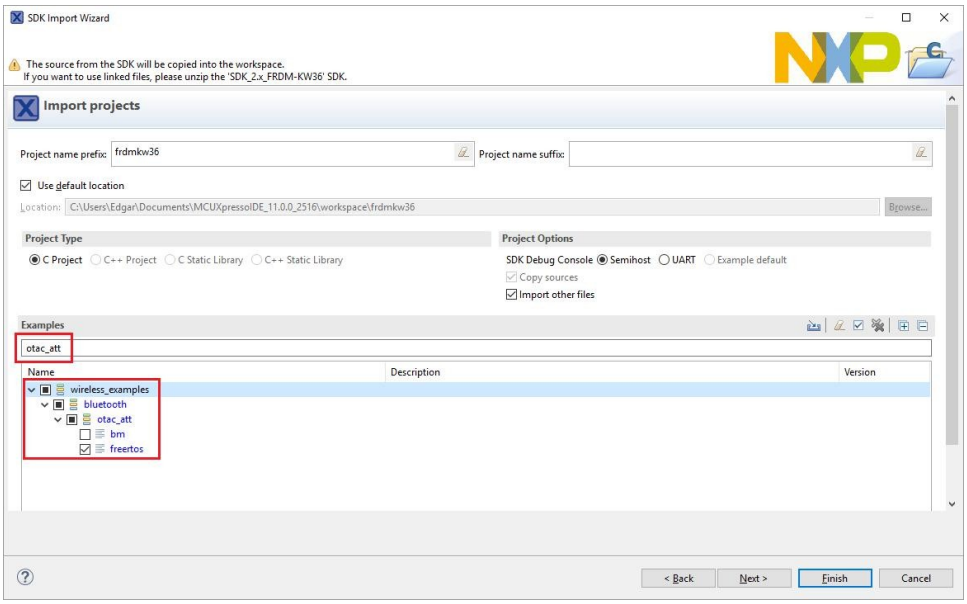


图 28. 在工作区导入 OTAP 客户端项目

6. 设置存储配置：

   a. 打开位于项目源文件夹中的“app_preinclude.h”文件：

   - 外部闪存方法（AT45DB041E_c 外部闪存），请将“gEepromType”定义为“gEepromDevice_AT45DB041E_c”。
   - 内部闪存方法（片上 FlexNVM 内存），请将“gEepromType”定义为“gEepromDevice_InternalFlash_c”。

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

图 29. 在 preinclude 文件中配置存储的方法

   b. 找到 MCUXpresso IDE 中的“Project->Properties”。转到“C/C++ Build->Settings->Tool Settings->MCU Linker->Miscellaneous”视图。

   - **外部闪存方法**，请在“Memory details”窗口配置 图 30 所示的字段。

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

图 30. 配置外部存储的方法

   - **内部闪存方法**，请在“Memory details”窗口配置 图 31 所示的字段。

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

图 31. 配置内部存储的方法

7. 清理并构建项目。 将项目烧写到“FRDM-KW36”板上。

此时，您已经在您的板子上编程并配置了 OTAP 客户端。您可以与服务器通信并请求软件更新。

5.2 创建一个 HRS-OTAPS-Record 镜像来更新软件

1. 在 MCUXpresso IDE 中安装此文档附带的 HRS-OTAP demo。可以将例程从安装路径拖拽到 MCUXpresso 工作区。显示消息弹窗，单击“Copy”按钮复制原始示例。

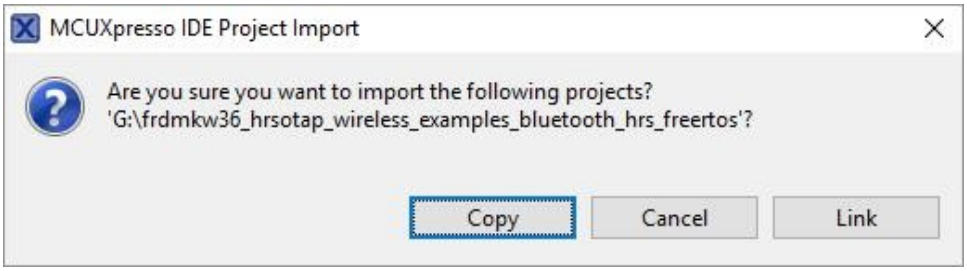


图 32. 在 MCUXpresso 工作区导入 HRS-OTAP demo

2. 打开工作区 linkscripts 文件夹中的“end_text.ldt”链接器脚本，找到 图 33 的位置，删除“FILL”和“BYTE”语句。只有通过构建 SREC 镜像文件来重新编程设备时，才需要此步骤。

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

图 33. 准备链接器文件

3. 清理并构建项目。
4. 在工作区中展开“Binaries”图标。右键“.axf”文件，选择“Binary Utilities->Create S-Record”选项。S-Record 文件保存在工作区中的“Debug”文件夹中，扩展名为“.s19”。

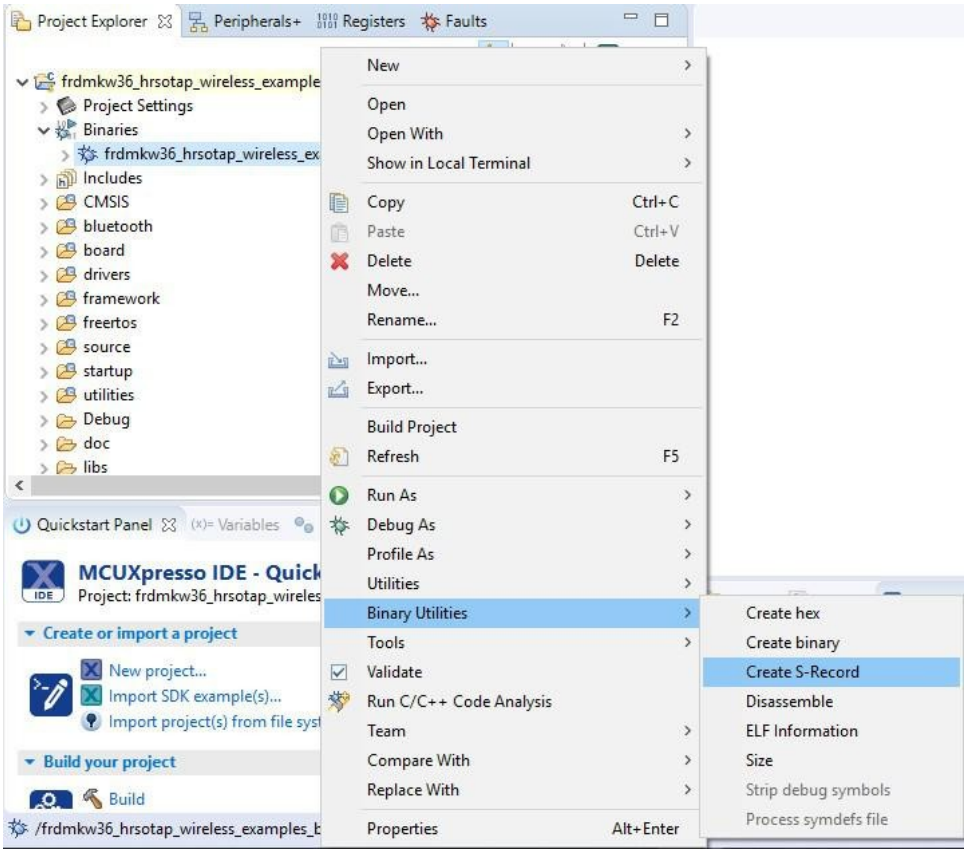


图 34. 创建 SREC 文

5. 将生成的文件导入到智能手机中。

5.3 创建一个 HRS S-Record 镜像来更新软件

1. 打开 MCUXpresso IDE。单击“Quickstart Pane”视图中的“Import SDK example(s)”选项，将显示设备选择。双击 frdmkw36 图标。
2. 在 examples 文本框中键入“hrs”并在“wireless_examples->bluetooth->hrs->freertos”中选择 freertos 项目，单击“Finish”按钮。

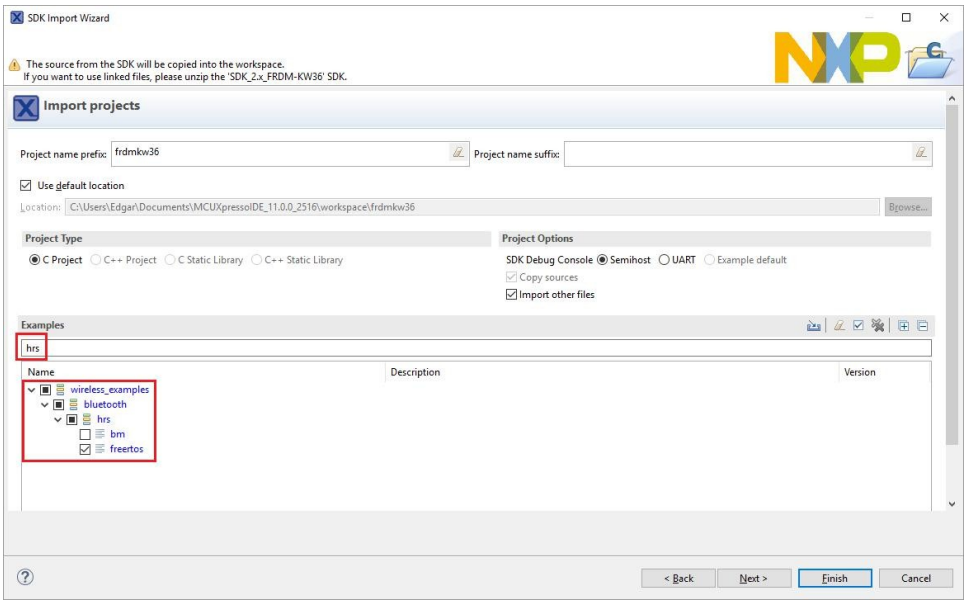
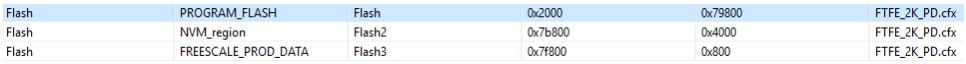


图 35. 在工作区导入 HRS 项目

3. 在 MCUXpresso 工作区打开源文件夹下的“app_preinclude.h”文件。找到“cPWR_UsePowerDownMode”宏并将其值更改为零。此步骤不是强制的，但在运行时有助于确认 OTAP bootloader 是否成功完成了软件更新。

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

4. 找到“Project->Properties->C/C++ Build->MCU settings”。配置以下字段并保存更改。



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

图 36. 配置内存

5. 进入工作区。找到“linkscripts”文件夹，向其中添加“main_text_section.ldt”链接描述文件。可以从 OTAP 客户端 SDK 进行复制和粘贴。

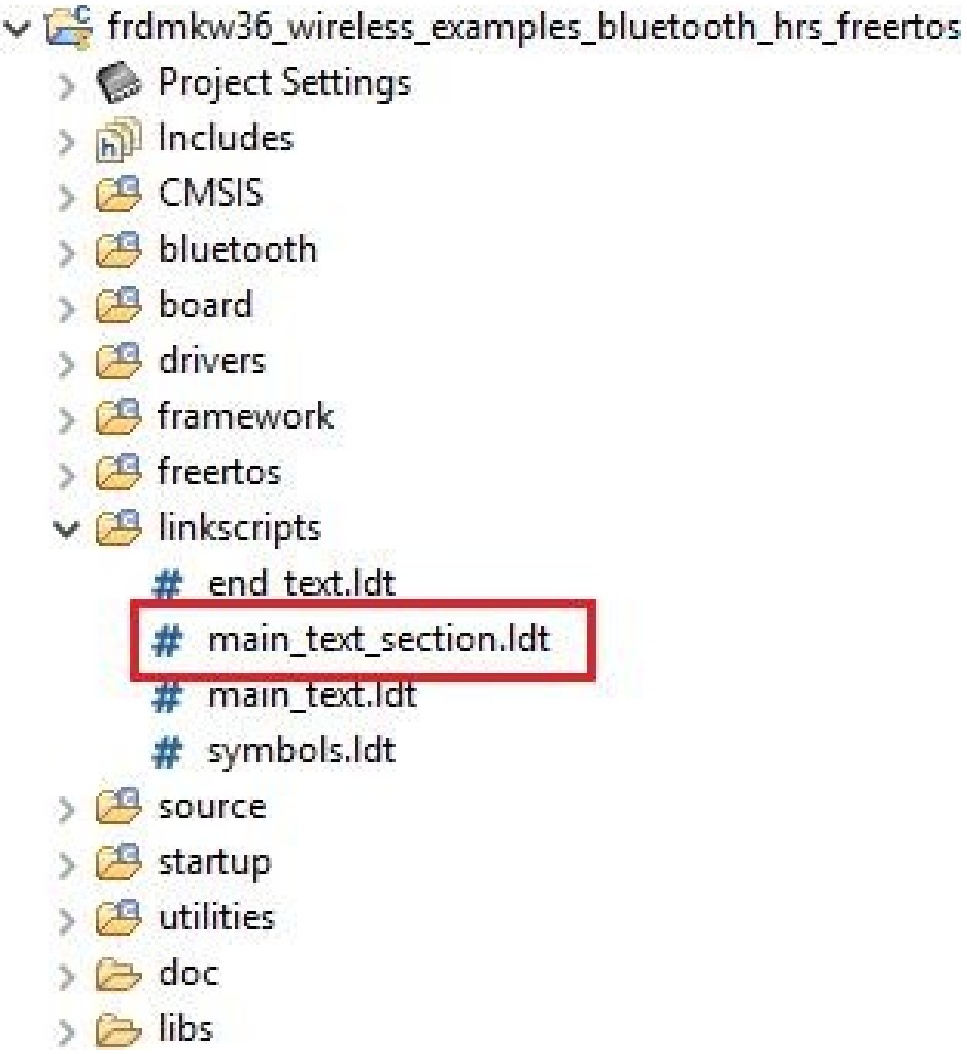


图 37. 导入链接器脚本

6. 打开工作区中的 linkscripts 文件夹中的“end_text.ldt”链接器脚本。找到 图 38 所示位置，并删除“FILL”和“BYTE”语句。

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

图 38. 准备链接器文件

7. 向“framework”文件夹添加“OtaSupport”文件夹及其文件。向“framework-> Flash”文件夹添加“Externa”文件夹及其文件。该步骤可以按照将 OTAP 服务和框架服务导入 HRS 中相同的方法完成。
8. 清理并构建项目。
9. 在工作区中展开“Binaries”图标。右键“.axf”文件。选择“Binary Utilities->Create S-Record”选项。S-Record 文件保存在工作区中的“Debug”文件夹中，扩展名为“.s19”。
10. 将该文件导入到智能手机。

5.4 测试 HRS-OTAP 软件

为了说明本节的测试案例，请参见 图 39。FRDM-KW36 包含 OTAP 客户端软件，OTAP 客户端从 OTAP 服务器（智能手机）请求软件更新，该软件镜像是 HRS-OTAP demo。到这步，FRDM-KW36 已经更新，可以处理来自 HR 服务器或 OTAP 服务器的所有通信。为了说明可以继续更新 KW36 设备的软件，可以将 HRS-OTAP 连接到 OTAP 服务器，并请求仅包含 HRS 示例的软件更新。更新后，不能继续再更新软件，因为上次软件升级中未包含 OTAP 服务。

此示例旨在介绍 OTAP 集成的重要性。但是本文的主要目的是创建包含 OTAP 服务的软件更新功能，并继续升级和改进 KW36 固件。

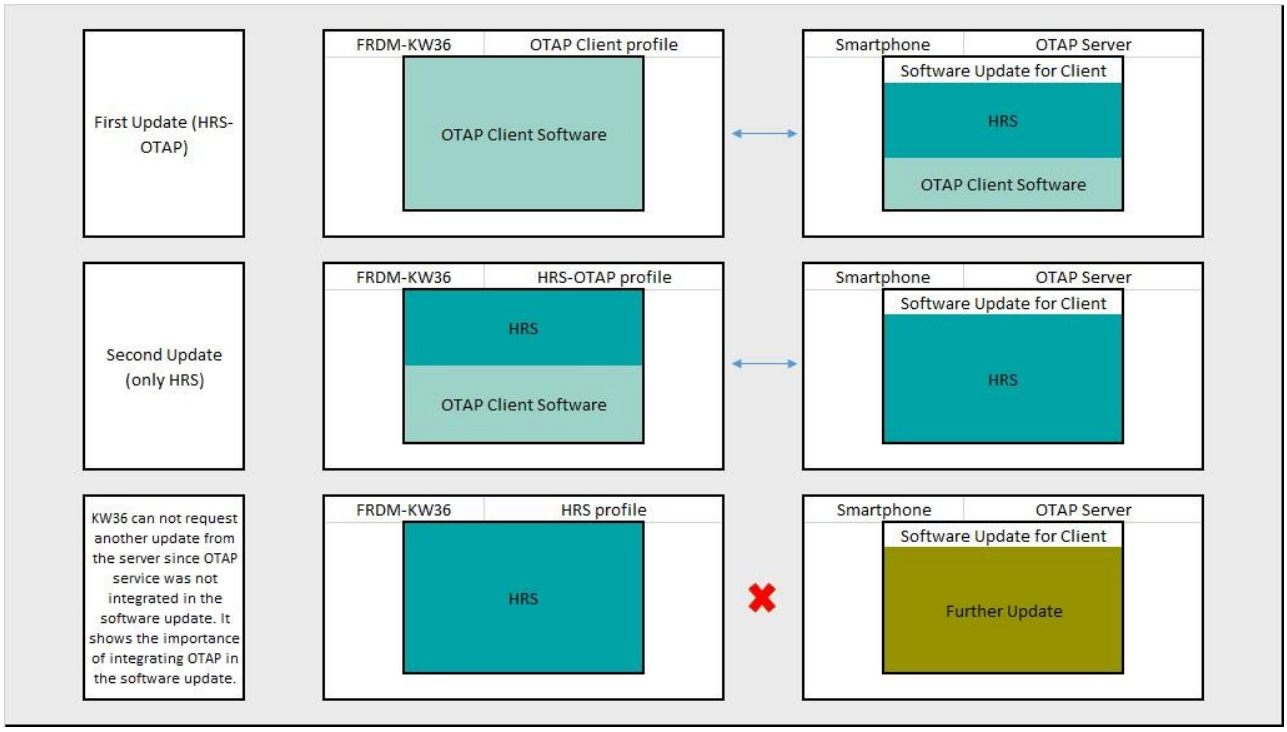


图 39. 建议进行的测试

1. 打开 IoT Toolbox App，然后选择 OTAP demo。 点击“SCAN”开始扫描合适的广播对象。

图 40. IoT Toolbox 接口

2. 按下 FRDM-KW36 板上的 ADV 按钮（SW2）开始广播。
3. 建立与“NXP_OTAA”设备的连接。然后，OTAP 接口将显示在您的智能手机上。

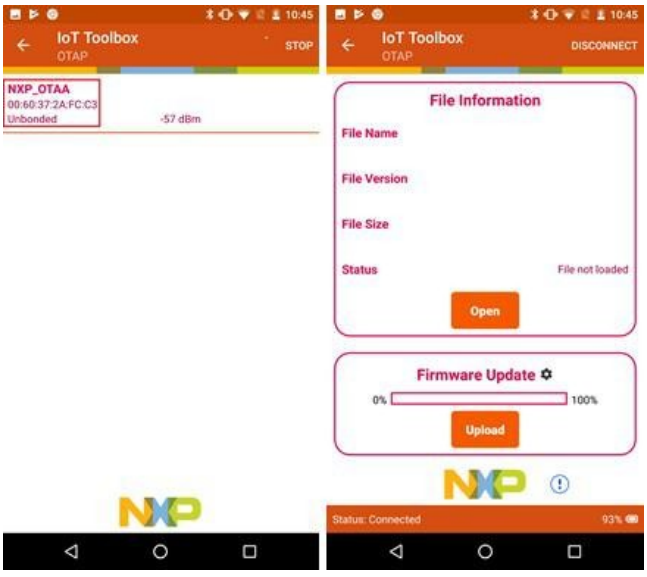


图 41. 连接 OTAP 客户端和 OTAP 服务器

4. 单击“Open”按钮并搜索“HRS-OTAP”SREC 文件。
5. 单击“Upload”启动传输。等待确认消息显示。

图 42. 将 OTAP 客户端更新为 HRS-OTAP

6. 等待几秒钟，直到 OTAP bootloader 完成新镜像的编程，HRS-OTAP 应用程序会自动启动（RGB LED 会闪烁）。
7. 按下 FRDM-KW36 板上的 ADV 按钮（SW2）开始广播。IoT Toolbox 的 HRS 和 OTAP 应用程序都可以检测到该设备，设备命名为“NXP_HRS_OTAP”。HRS 和 OTAP 应用程序都可以和板子创建连接并进行交互。

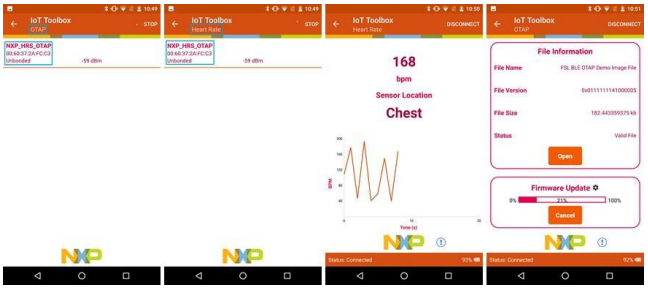


图 43. 两个程序都检测到 HRS-OTAP 设备

8. 将 HRS-OTAP 设备与 OTAP 手机应用程序连接起来。使用 “HRS”SREC 文件更新软件。
9. 使用 HRS-OTAP，更新设备为单一的 HRS。按下 FRDM-KW36 板上的 ADV 按钮（SW2），开始广播。现在，设备的名称是“NXP_HRS”。将设备与 HRS IoT Toolbox app 连接，并验证其是否正常工作。

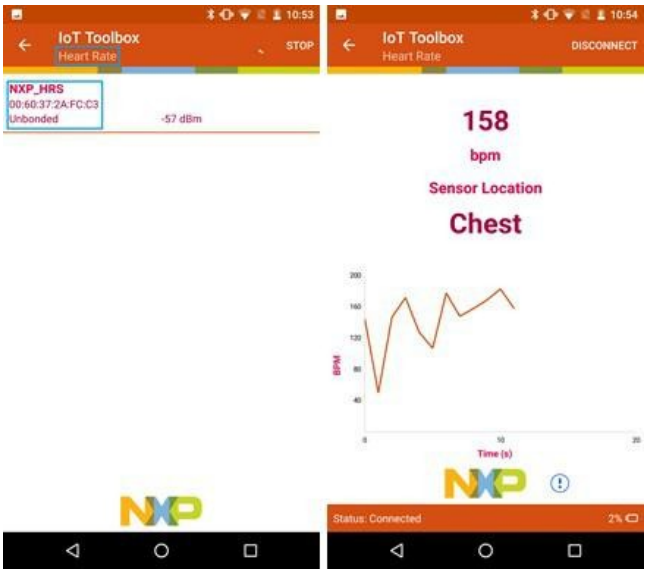


图 44. HRS 检测到设备

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — 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.

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.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. 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.

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

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 2020 年 3 月 11 日

Document identifier: AN12775

arm