BLE OTA Fixed Stack Bootloader and Bootloadable

1.30

Features

- Over-the-Air (OTA) firmware update
- Shared BLE Bootloader Service to receive bootloadable images
- HID Keyboard

General Description

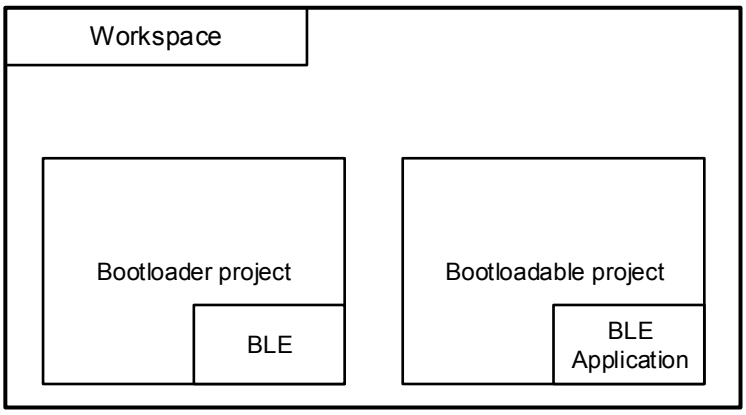
This example shows how to use the custom linker scripts to share a block of memory between the bootloader and bootloadable projects. It demonstrates how the bootloader can place the API functions so that the bootloadable can also call them. This allows creating an OTA bootloader.

The purpose of the Bootloader project is to replace a bootloadable image on the device with an image sent OTA by the Bluetooth protocol.

The bootloadable project uses BLE APIs implemented in the bootloader part of the memory (Figure 1).

Note Currently only the GCC 4.9.3, MDK, and IAR compilers are supported.

Figure 1. OTA Fixed Stack Workspace



By default, both bootloader and bootloadable projects are expected to be located in the same workspace. However, the user can save projects in any location and modify paths in the build script.

Figure 2 shows the architecture of the bootloader project; Figure 3 shows the architecture of the bootloadable project.

Figure 2. Bootloader Project Architecture

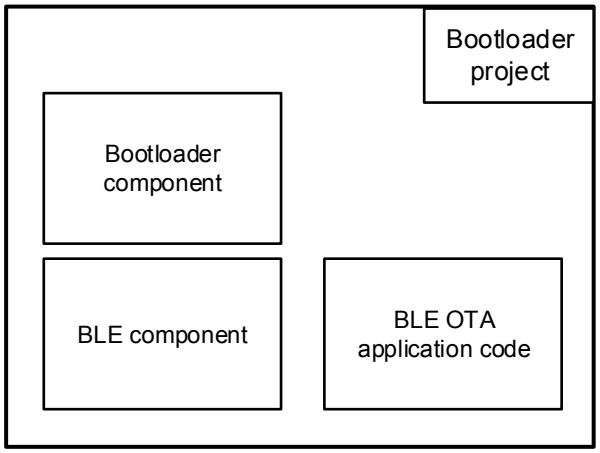
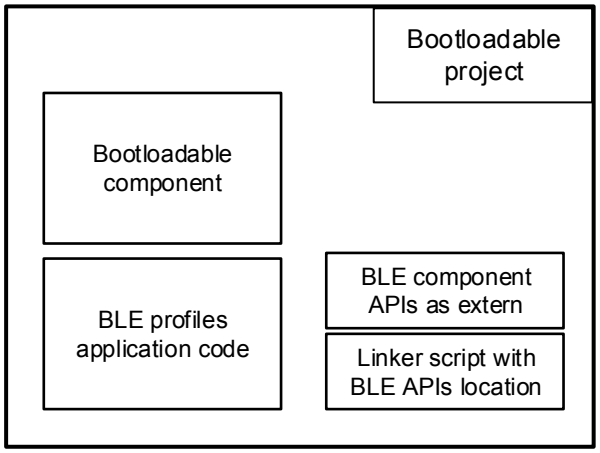


Figure 3. Bootloadable Project Architecture



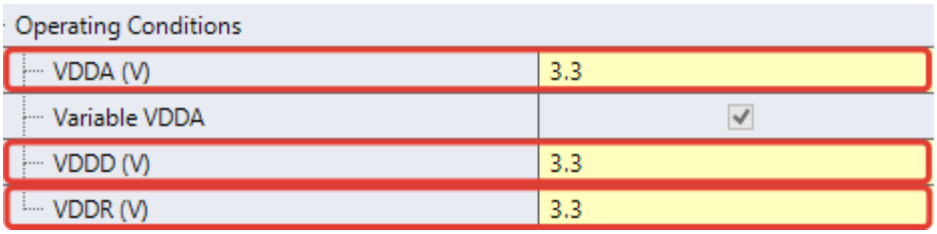
The bootloadable project may use APIs from the bootloader project if:

1. These APIs are listed as ‘extern’ in a dedicated header.
2. These APIs addresses are provided in the bootloadable project linker script by the post-build script of the bootloader.
3. Bootloader *.hex/*.elf has not been modified since step 2.

Development Kit Configuration

This example project is designed to run on the Cypress CY8CKIT-042-BLE kit. A description of the kit along with more example programs and ordering information can be found at http://www.cypress.com/go/cy8ckit-042-ble.

The kit must be powered by 3.3 V (J16 is set to 1 and 2). No connection on the kit board is required to use this example project. If you want to power the kit with 5V supply you should modify the Operating Conditions in the *.cydwr files under System tab for both bootloader and bootloadable projects:



Refer to the Setup and Run Example Project section of this document for instructions on how to use this example project.

If the kit will be powered with different from 3.3 voltage – projects settings have to be updated appropriately in the System DWR settings on System tab.

Bootloader Project Configuration

The Bootloader project contains the following components:

- Bluetooth Low Energy (BLE)
- Bootloader
- Software Transmit UART
- Watchdog timer (WDT) that is used as general timer

Linker Settings

Check the correctness of the project linker settings:

1. Right-click on the bootloader project in **Workspace Explorer**.
2. Click **Build Settings**.
3. Expand **Linker** under the selected compiler:
    a. If **ARM GCC 4.9.3** is selected:

        Check the values of the following options in the **General** section:

Section	Option	Value
General	Custom Linker Script	.\cm0gcc.ld

b. In case of **ARM MDK Generic**:

Check the values of the following options in the **General** and **Command line** section:

Section	Option	Value
General	Use MicroLIB	False
Command line	Custom flags	--keep=Advertising_LED_1_SetDriveMode -- keep=Advertising_LED_2_SetDriveMode -- keep=Bootloading_LED_SetDriveMode -- keep=Bootloader_Service_Activation_Read -- keep=Bootloader_Service_Activation_SetDriveMode -- keep=Bootloader_Service_Activation_Write --keep=CyBle_StackInit -- keep=CyBle_Shutdown --keep=CyBle_SoftReset --keep=CyBle_EnterLPM -- keep=CyBle_ExitLPM --keep=CyBle_ProcessEvents -- keep=CyBle_SetDeviceAddress --keep=CyBle_GetDeviceAddress -- keep=CyBle_GetRssi --keep=CyBle_GetTxPowerLevel -- keep=CyBle_SetTxPowerLevel --keep=CyBle_GetBleClockCfgParam -- keep=CyBle_SetBleClockCfgParam --keep=CyBle_GenerateRandomNumber -- keep=CyBle_SetCeLengthParam --keep=CyBle_WriteAuthPayloadTimeout -- keep=CyBle_ReadAuthPayloadTimeout --keep=CyBle_GetStackLibraryVersion - -keep=CyBle_GetBleSsState --keep=CyBle_AesEncrypt -- keep=CyBle_AesCcmInit --keep=CyBle_AesCcmEncrypt -- keep=CyBle_AesCcmDecrypt --keep=CyBle_SetTxGainMode -- keep=CyBle_SetRxGainMode --keep=CyBle_GattGetMtuSize -- keep=CyBle_ApplCallback --keep=cyBle_deviceAddress -- keep=cyBle_sflashDeviceAddress --keep=cyBle_discoveryModeInfo -- keep=cyBle_scanRspData --keep=cyBle_discoveryData -- keep=cyBle_discoveryParam --keep=CyBle_Init --keep=CyBle_ServiceInit -- keep=CyBle_Start --keep=CyBle_Stop --keep=CyBle_StoreBondingData -- keep=CyBle_GappStartAdvertisement --keep=CyBle_GappStopAdvertisement -- keep=CyBle_ChangeAdDeviceAddress --keep=CyBle_GapSetLocalName -- keep=CyBle_GapGetLocalName --keep=CyBle_Get16ByPtr -- keep=CyBle_Set16ByPtr --keep=cyBle_bass --keep=CyBle_BasInit -- keep=CyBle_BasRegisterAttrCallback -- keep=CyBle_BassSetCharacteristicValue -- keep=CyBle_BassGetCharacteristicValue -- keep=CyBle_BassGetCharacteristicDescriptor -- keep=CyBle_BassWriteEventHandler --keep=CyBle_BassSendNotification -- keep=cyBle_diss --keep=CyBle_DisInit --keep=CyBle_DisRegisterAttrCallback -- keep=CyBle_DissSetCharacteristicValue -- keep=CyBle_DissGetCharacteristicValue --keep=cyBle_hidss -- keep=CyBle_HidsInit --keep=CyBle_HidsRegisterAttrCallback -- keep=CyBle_HidssSetCharacteristicValue -- keep=CyBle_HidssGetCharacteristicValue -- keep=CyBle_HidssGetCharacteristicDescriptor -- keep=CyBle_HidssWriteEventHandler --keep=CyBle_HidssSendNotification -- keep=CyBle_HidssOnDeviceConnected --keep=cyBle_scpss -- keep=CyBle_ScpsInit --keep=CyBle_ScpsRegisterAttrCallback -- keep=CyBle_ScpssSetCharacteristicValue -- keep=CyBle_ScpssGetCharacteristicValue -- keep=CyBle_ScpssGetCharacteristicDescriptor -- keep=CyBle_ScpssWriteEventHandler --keep=CyBle_ScpssSendNotification -- keep=CyBle_GattsEnableAttribute --apcs=/pic --info unused

Note MicroLib has compilation limitations which prevent the usage of code sharing. Cause of this MircoLib cannot be used neither in BLE OTA Bootloader nor in BLE OTA Bootloadable projects.

c. And if the project was exported to IAR:

Set the following settings in the project options:

Section	Option	Value
C/C++ Compiler	Command line options	--no_fragments --no_inline --no_wrap_diagnostics --interwork --no_unaligned_access --cpu_mode=thumb
Linker	Command line options	--keep=Bootloader_Checksum --keep=Advertising_LED_1_SetDriveMode --keep=Advertising_LED_2_SetDriveMode --keep=Bootloading_LED_SetDriveMode --keep=Bootloader_Service_Activation_Read --keep=Bootloader_Service_Activation_SetDriveMode --keep=Bootloader_Service_Activation_Write --keep=CyBle_StackInit --keep=CyBle_Shutdown --keep=CyBle_SoftReset --keep=CyBle_EnterLPM --keep=CyBle_ExitLPM --keep=CyBle_ProcessEvents --keep=CyBle_SetDeviceAddress --keep=CyBle_GetDeviceAddress --keep=CyBle_GetRssi --keep=CyBle_GetTxPowerLevel --keep=CyBle_SetTxPowerLevel --keep=CyBle_GetBleClockCfgParam --keep=CyBle_SetBleClockCfgParam --keep=CyBle_GenerateRandomNumber --keep=CyBle_SetCeLengthParam --keep=CyBle_WriteAuthPayloadTimeout --keep=CyBle_ReadAuthPayloadTimeout --keep=CyBle_GetStackLibraryVersion --keep=CyBle_GetBleSsState --keep=CyBle_AesEncrypt --keep=CyBle_AesCcmInit --keep=CyBle_AesCcmEncrypt --keep=CyBle_AesCcmDecrypt --keep=CyBle_SetTxGainMode --keep=CyBle_SetRxGainMode --keep=CyBle_GattGetMtuSize --keep=CyBle_ApplCallback --keep=cyBle_deviceAddress --keep=cyBle_sflashDeviceAddress --keep=cyBle_discoveryModeInfo --keep=cyBle_scanRspData --keep=cyBle_discoveryData --keep=cyBle_discoveryParam --keep=CyBle_Init --keep=CyBle_ServiceInit --keep=CyBle_Start --keep=CyBle_Stop --keep=CyBle_StoreBondingData --keep=CyBle_GappStartAdvertisement --keep=CyBle_GappStopAdvertisement --keep=CyBle_GattsEnableAttribute --keep=CyBle_ChangeAdDeviceAddress --keep=CyBle_GapSetLocalName --keep=CyBle_GapGetLocalName --keep=CyBle_Get16ByPtr --keep=CyBle_Set16ByPtr --keep=cyBle_bass --keep=CyBle_BasInit --keep=CyBle_BasRegisterAttrCallback --keep=CyBle_BassSetCharacteristicValue

		--keep=CyBle_BassGetCharacteristicValue --keep=CyBle_BassGetCharacteristicDescriptor --keep=CyBle_BassWriteEventHandler --keep=CyBle_BassSendNotification --keep=cyBle_diss --keep=CyBle_DisInit --keep=CyBle_DisRegisterAttrCallback --keep=CyBle_DissSetCharacteristicValue --keep=CyBle_DissGetCharacteristicValue --keep=cyBle_hidss --keep=CyBle_HidsInit --keep=CyBle_HidsRegisterAttrCallback --keep=CyBle_HidssSetCharacteristicValue --keep=CyBle_HidssGetCharacteristicValue --keep=CyBle_HidssGetCharacteristicDescriptor --keep=CyBle_HidssWriteEventHandler --keep=CyBle_HidssSendNotification --keep=CyBle_HidssOnDeviceConnected --keep=cyBle_scpss --keep=CyBle_ScpsInit --keep=CyBle_ScpsRegisterAttrCallback --keep=CyBle_ScpssSetCharacteristicValue --keep=CyBle_ScpssGetCharacteristicValue --keep=CyBle_ScpssGetCharacteristicDescriptor --keep=CyBle_ScpssWriteEventHandler --keep=CyBle_ScpssSendNotification --semihosting

Bluetooth Low Energy (BLE)

The BLE component is used to implement the BLE Human Interface Device (HID over GATT keyboard), Battery Service (BAS), Scan Parameters (SCPS), Device Information (DIS), and a hidden Bootloader Service. The purpose of the Bootloader Service is to receive other bootloadable images. The Bootloader Service is available only in the bootloader. When a device is in the bootloadable mode (the device operation is the same as in the BLE_HID_Keyboard example, except for button SW2) in this example project, it forces the device to go to the bootloader mode.

Figure 4. BLE GATT Settings

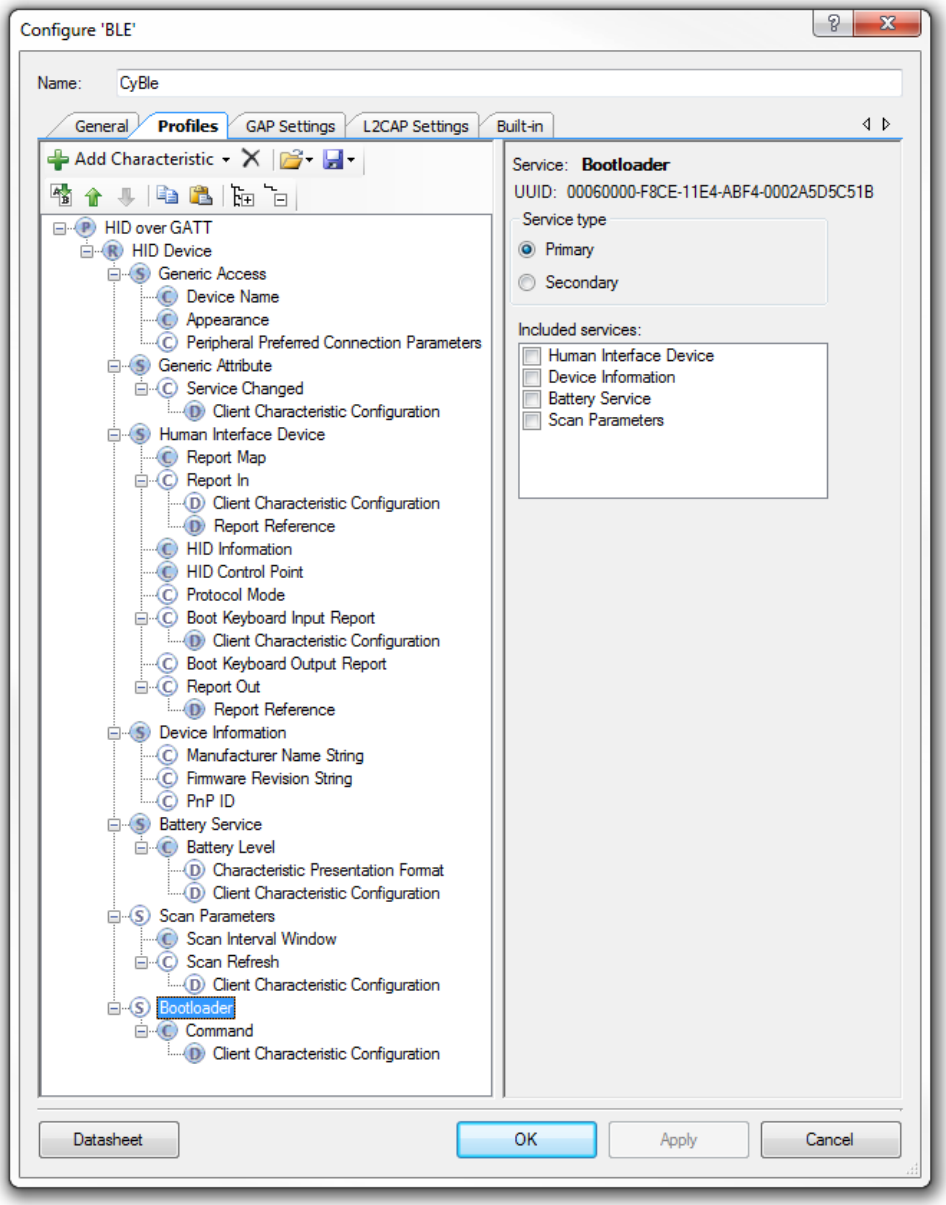


Figure 5. BLE GATT Settings

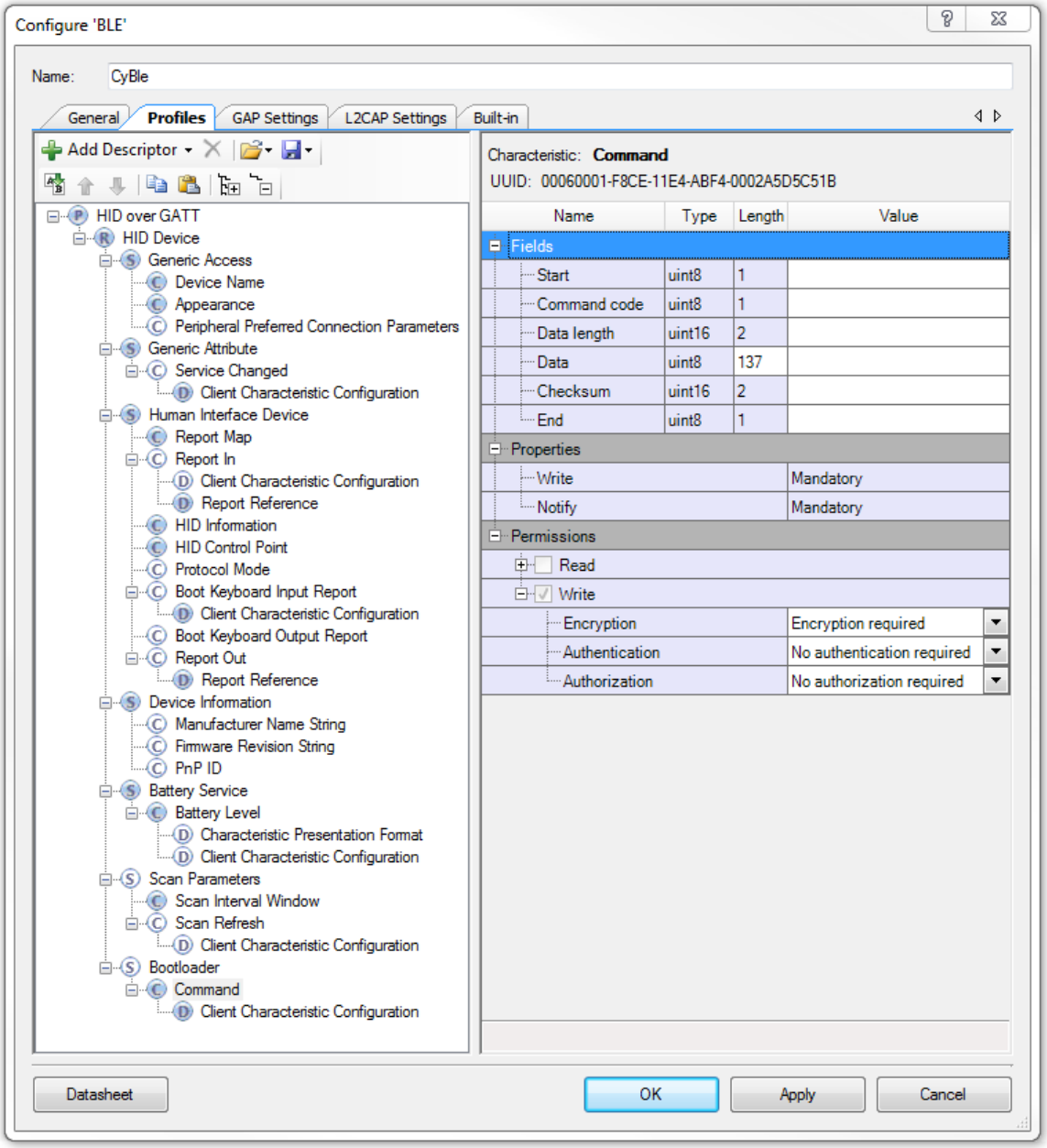
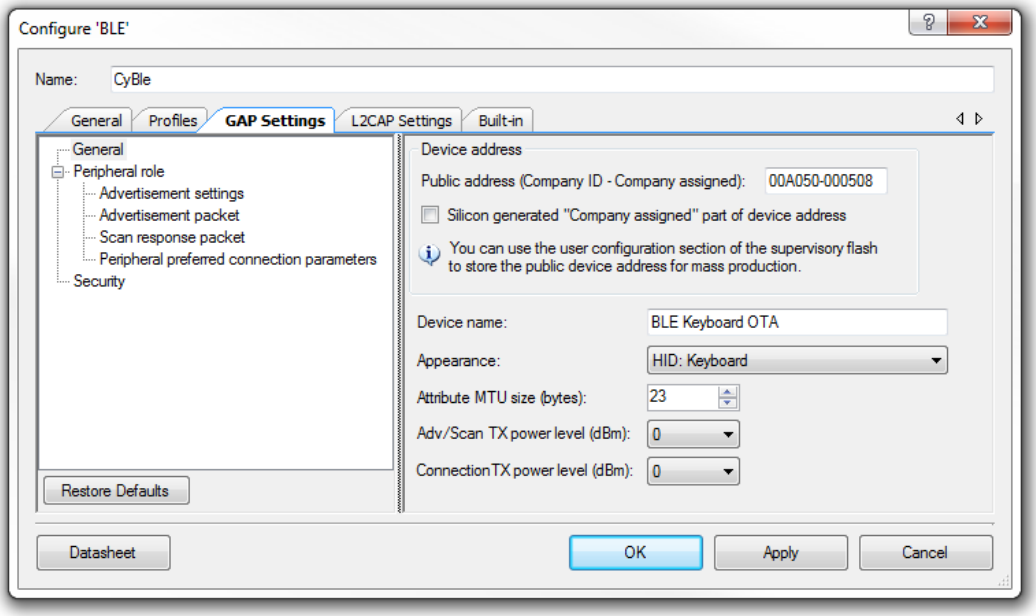


Figure 6. BLE GAP Settings



The Bootloader Service is enabled by default in the bootloader mode for communication with the Bootloader Host tool.

The BLE Bootloader Service has one characteristic that supports the “Write/Write Long Characteristic Values” procedure and notifications. The described characteristic also has a characteristic descriptor - Client Characteristic Configuration.

For communication, the Bootloader Host tool writes a command to the Command Characteristic and enables notifications in the characteristic descriptor.

The bootloader emulator reads the command from the Characteristic, processes it, and, if notifications are enabled in the characteristic descriptor, writes a response to the notification that is sent to Bootloader Host Tool.

The Bootloader Host tool receives the notification and, depending on its content (a bootloader emulator response), either sends the next command or reports an error.

The BLE component is configured to have MTU of 23 byte and the Bootloader Service UUID, which is 00060000-F8CE-11E4-ABF4-0002A5D5C51B.

After pressing onboard SW2, the LED changes its indication to red. This can be applied to both “advertising” and “connected’ modes.

Note The bootloader project is performing Flash writes. Due to this maximum connection interval value is set to 15 ms. Changing this parameter value to smaller values might result in disconnect during bootloading in case when Flash write will take longer. Since Flash writes take about 10ms (refer to the device datasheet) this value cannot be smaller. If device Flash writes take more than 15ms this value should be changed.

The bootloadable project shares key BLE component settings this value is inherited from the bootloader project.

Adding Services

This section describes how to add and use another service in the BLE component of the bootloader project. The basic steps include:

1. Add a service in the BLE component's customizer of the BLE_OTA_FixedStack_Bootloader project.
2. **Optional** Add code to disable the new service in the bootloader project as it is done in main.c using API CyBle_GattsDisableAttribute(). This is done to hide unused functionality in the bootloader mode.

Figure 7. Code Example of Service Disabling

```
91      CyBle_GattsDisableAttribute(cyBle_hidss[0].serviceHandle);
92      CyBle_GattsDisableAttribute(cyBle_diss.serviceHandle);
93      CyBle_GattsDisableAttribute(cyBle_bass[0].serviceHandle);
94      CyBle_GattsDisableAttribute(cyBle_scpss.serviceHandle);
95
```

3. Build the BLE_OTA_FixedStack_Bootloader project.
4. Rerun the mk.bat script to update the bootloadable project linker scripts.
5. Add code that uses the added BLE API service in the BLE_OTA_FixedStack_Bootloadable project.
6. Add all the used BLE APIs to the OTAMandatory.h file as extern declarations in the BLE_OTA_FixedStack_Bootloadable project.
7. If APIs are to be used from the bootloadable project:
    a. For GCC compiler add those APIs to the section with KEEP instructions in the linker script of the bootloader project.
    b. For MDK compiler and for IAR same procedure should be performed with the linker command line argumens of the bootloader project.
    c. If services were removed APIs should be removed.
8. Add all the used BLE types and defines to the OTAMandatory.h file of the BLE_OTA_FixedStack_Bootloadable project.
9. Build the BLE_OTA_FixedStack_Bootloadable project.

Bootloader

Enable the Bootloader application validation option only if the BLE profiles do not require saving bonding information or if this information is not saved during the BLE component operation.

Otherwise, saving bonding information will update the Bootloader project image, which may cause the validation to fail.

Software Transmit UART

This component is used for printing debug information. Refer to the Using UART for Debugging section for details.

Watch Dog Timer (WDT)

The WDT is used here as a general purpose timer for LED indication.

Bootloadable Project Configuration

The bootloadable project contains the following components:

- Bootloadable
- Software Transmit UART

Linker Settings

Check the correctness of the project linker settings:

1. Right-click on the bootloadable project in **Workspace Explorer**.
2. Click **Build Settings**.
3. Expand **Linker** under the selected compiler:
    a. If **ARM GCC 4.9.3** is selected:

    Check the values of the following options in the **General** and **Optimization** sections:

Section	Option	Value
General	Additional Library Directories	.\LinkerScripts\
General	Custom Linker Script	.\LinkerScripts\cm0gcc.ld
Optimization	Removed Unused Function	False

    b. In case of **ARM MDK Generic**:

    Check the values of the following options in the **Linker** section:

Section	Option	Value
General	Custom Linker Script	.\LinkerScripts\Cm0Mdk.scat

General	Use MicroLIB	False
Command line	Custom flags	.\LinkerScripts\BootloaderSymbolsMdk.txt

c. And if the project was exported to IAR:

Set the following settings in the project options:

Section	Option	Value
C/C++ Compiler	Command line options	--no_fragments --no_cse --no_unroll --no_inline --no_code_motion --no_tbaa --no_scheduling --debug --no_wrap_diagnostics --interwork --no_unaligned_access --cpu_mode=thumb
Linker	Command line options	--no_dynamic_rtti_elimination --no_range_reservations --no_fragments

Bootloadable

The Bootloadable component is used to create an image with the bootloadable firmware that can be updated without affecting the bootloader.

Software Transmit UART

The component is used for printing debug information. Refer to the Using UART for Debugging section for the details.

Watch Dog Timer (WDT)

The WDT is used here as a general purpose timer for LED indication.

Custom Linker Scripts

The custom linker scripts are intended for keeping the Bootloader's RAM regions not re-initialized by the Bootloadable application and including the list of exported APIs.

Custom Linker Script Description

Bootloader and Bootloadable projects use the custom linker scripts for the GCC compiler. For other supported compilers the custom linker scripts are used only for bootloadable projects.

Compared to a generated script, the bootloader's custom linker script for GCC (cm0gcc.ld), contains a few KEEP instructions to prevent the required symbols from being removed by the linker and this script is located in the Bootloader sources folder (\BLE_OTA_FixedStack_Bootloader.cydsn).

For the GCC compiler, the bootloadable's custom linker script (cm0gcc.ld) includes a generated file with a list of exported symbols (BootloaderSymbolsGcc.ld) and declares the ".bootloader_data" section as NOLOAD, so that it can't be re-initialized. The Bootloadable's custom linker script is located in the Bootloadable's folder (\BLE_OTA_FixedStack_Bootloadable.cydsn\LinkerScripts\).

For the MDK compiler, the bootloader does not use custom linker scripts – instead all the options are sent via a command line to the linker:

```
--keep=Advertising_LED_1_SetDriveMode --keep=Advertising_LED_2_SetDriveMode --
keep=Bootloading_LED_SetDriveMode --keep=Bootloader_Service_Activation_Read --
keep=Bootloader_Service_Activation_SetDriveMode --keep=Bootloader_Service_Activation_Write --keep=CyBle_StackInit --
keep=CyBle_Shutdown --keep=CyBle_SoftReset --keep=CyBle_EnterLPM --keep=CyBle_ExitLPM --
keep=CyBle_ProcessEvents --keep=CyBle_SetDeviceAddress --keep=CyBle_GetDeviceAddress --keep=CyBle_GetRssi --
keep=CyBle_GetTxPowerLevel --keep=CyBle_SetTxPowerLevel --keep=CyBle_GetBleClockCfgParam --
keep=CyBle_SetBleClockCfgParam --keep=CyBle_GenerateRandomNumber --keep=CyBle_SetCeLengthParam --
keep=CyBle_WriteAuthPayloadTimeout --keep=CyBle_ReadAuthPayloadTimeout --keep=CyBle_GetStackLibraryVersion --
keep=CyBle_GetBleSsState --keep=CyBle_AesEncrypt --keep=CyBle_AesCcmInit --keep=CyBle_AesCcmEncrypt --
keep=CyBle_AesCcmDecrypt --keep=CyBle_SetTxGainMode --keep=CyBle_SetRxGainMode --
keep=CyBle_GattGetMtuSize --keep=CyBle_ApplCallback --keep=cyBle_deviceAddress --keep=cyBle_sflashDeviceAddress
--keep=cyBle_discoveryModeInfo --keep=cyBle_scanRspData --keep=cyBle_discoveryData --keep=cyBle_discoveryParam --
keep=CyBle_Init --keep=CyBle_ServiceInit --keep=CyBle_Start --keep=CyBle_Stop --keep=CyBle_StoreBondingData --
keep=CyBle_GappStartAdvertisement --keep=CyBle_GappStopAdvertisement --keep=CyBle_ChangeAdDeviceAddress --
keep=CyBle_GapSetLocalName --keep=CyBle_GapGetLocalName --keep=CyBle_Get16ByPtr --keep=CyBle_Set16ByPtr --
keep=cyBle_bass --keep=CyBle_BasInit --keep=CyBle_BasRegisterAttrCallback --keep=CyBle_BassSetCharacteristicValue -
-keep=CyBle_BassGetCharacteristicValue --keep=CyBle_BassGetCharacteristicDescriptor --
keep=CyBle_BassWriteEventHandler --keep=CyBle_BassSendNotification --keep=cyBle_diss --keep=CyBle_DisInit --
keep=CyBle_DisRegisterAttrCallback --keep=CyBle_DissSetCharacteristicValue --keep=CyBle_DissGetCharacteristicValue -
-keep=cyBle_hidss --keep=CyBle_HidsInit --keep=CyBle_HidsRegisterAttrCallback --
keep=CyBle_HidssSetCharacteristicValue --keep=CyBle_HidssGetCharacteristicValue --
keep=CyBle_HidssGetCharacteristicDescriptor --keep=CyBle_HidssWriteEventHandler --keep=CyBle_HidssSendNotification
--keep=CyBle_HidssOnDeviceConnected --keep=cyBle_scpss --keep=CyBle_ScpsInit --
keep=CyBle_ScpsRegisterAttrCallback --keep=CyBle_ScpssSetCharacteristicValue --
keep=CyBle_ScpssGetCharacteristicValue --keep=CyBle_ScpssGetCharacteristicDescriptor --
keep=CyBle_ScpssWriteEventHandler --keep=CyBle_ScpssSendNotification --keep=CyBle_GattsEnableAttribute --apcs=/pic
--info unused
```

For the MDK compiler, the bootloadable project uses a custom linker script that provides placement information of APIs from the bootloader project. The same approach is used for IAR.

Compilation Script Description

A compilation batch script file (mk.bat in the Bootloader project) generates custom link files. This script file parses the *.elf file generated from the bootloader project and generates a file for the bootloadable project linker with shared symbols placement. The script generates files into the \BLE_OTA_FixedStack_Bootloadable.cydsn\LinkerScripts\ directory. The script consists of two sections:

- Setup section
- Main section

The user must check the correctness of all paths in the Setup section. The setup options are as follows:

- **LOADER_PRJ_NAME** – the name of the bootloader project.
- **LOADER_PRJ_PATH** – the path to the bootloader project.
- **LOADABLE_PRJ_NAME** – the name of the bootloadable project.
- **OUTPUT_DIR –** the path to the directory where to put generated files.
- **BUILD_OUTPUT_DIR** – the path to the directory with *.elf file. There is a list of all possible options, choose only one, uncomment it, and comment some other option.
- **LOADER_ELF_FILE** – the full name with a path to the *.elf file.
- **COMPILER** – the compiler identifier.
- **COMPILE_OPTION** – the Debug/Release option.
- **KEEP_TMP_FILES** – the options to keep temporary generated by this script files.
- **UTILS_NM** – the path to the Creator directory with the arm-none-eabi-nm.exe tool. Check if this path is correct.
- The names of input files with section and symbols names are defined in **SECTION_LIST** and **SEARCHING_SYMBOLS**.
- The names of temporary files defined in **PARSED_ELF**, **EXTRACTED_BLE_INFO**, **BLE_INFO_ALL**, and **BLE_INFO_ALL_SORTED**.
- The main section checks the input, searches specified symbols in *.elf files and generates an output for linker scripts.

Note The names of the bootloader project component to be exported to the bootloadable project must be aligned with the Symbol_list.txt file. It is not recommended to change the default values. The script file may work in a wrong way if these names are misrecognized inside other names (e.g., BLE as part of TABLE). CyBle is the recommended name for the BLE component.

Custom linker scripts limitations:

For project(s) that use custom linker scripts heap and stack size that was set in the PSoC Creator will not be applied. Size will have to be specified manually in corresponding linker scripts.

Setup and Run Example Project

Building Example Project

The following steps are described for compiler GCC 4.9.3 that is shipped with PSoC Creator. If the MDK compiler is used, variable COMPILER should be changed to MDK_COMPILER and it should be set to IAR_COMPILER for IAR. **Note** The same compiler should be used for the bootloader and bootloadable projects.

For more information on changing the compiler from a default one (GCC), refer to section Changing projects compiler.

1. Build the *BLE_OTA_FixedStack_Bootloader* example project. Check that the proper compiler toolchain is set in **Project -> Build Settings**.
2. **Optional** Add the *BLE_OTA_FixedStack_Bootloadable* example project to the workspace).
   a. Setup the mk.bat file. Open the mk.bat file in the text editor. Set the following variables (see the chain of file generation in the table below):

Variable name	Default value	Description
LOADER_PRJ_NAME	BLE_OTA_FixedStack_Bootloader	The Bootloader project's name. **Note** The Bootloader project's name is different. For each example project, PSoC Creator appends a number starting from 01 to the example project's name in a selected folder
LOADABLE_PRJ_NAME	BLE_OTA_FixedStack_Bootloadable	The Bootloadable project's name. **Note** The Bootloadable project's name is different. For each example project PSoC Creator appends a number starting from 01 to the example project's name in a selected folder.
LOADER_PRJ_PATH	"%~dp0..\" that means Bootloader's sources folder.	The path to the Bootloader project. By default it takes the current path, but it can be set as the absolute path.
OUTPUT_DIR	%~dp0..\..\%LOADABLE_PRJ_NAME%.cydsn\LinkerScripts\ That means Bootloadable's \LinkerScripts\ folder	The path to the Bootloadable's sources, where the "LinkerScripts" folder is located. You should put the absolute path to the Bootloadable project here. **Note** The default path is valid if the Bootloader and Bootloadable projects are in the same workspace.

Variable name	Default value	Description
CREATOR_VERSION	3.3	Version of PSoC Creator. For example, 3.2, 3.3 or any other that matches installed PSoC Creator version. This variable must contain version of PSoC Creator installed. **Note** PSoC Creator Service Packs, Device Packs do not matter here. You need to specify only version number.
CREATOR_LOCATION	%PROGRAM_FILES_PATH%\Cypress\ PSoC Creator	By default, this value should not be changed. This path needs to be modified only if PSoC Creator is installed to a different location. In this case full path to PSoC Creator installation should be specified.
COMPILER	GCC_COMPILER=ARM_GCC_493 COMPILER= %GCC_COMPILER%	Keeps the compiler identifier. COMPILER=%GCC_COMPILER% or COMPILER=%MDK_COMPILER% or COMPILER=%IAR_COMPILER%
COMPILE_OPTION	COMPILE_OPTION=%COMPILE_OPTION_DEBUG%	Keeps either the Debug or Release option. You can choose it from the following list: COMPILE_OPTION_DEBUG=Debug COMPILE_OPTION_RELEASE=Release
KEEP_TMP_FILES	KEEP_TMP_FILES=YES	If set to YES, it keeps temporary files generated during a batch file run.

b. Use the following link for detailed information on what a BATCH file is and how it works:
http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/batch.mspx?mfr=true

The temporary files that are generated during a BATCH file run. **Note** They are available if variable **KEEP_TMP_FILES** is set to YES.

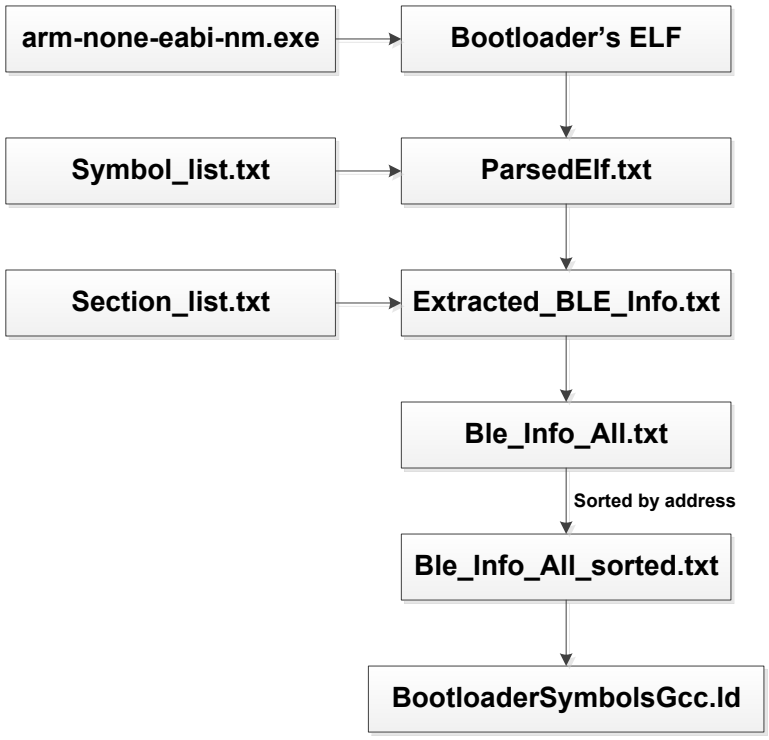
- "ParsedElf.txt" – the output file that contains extracted information from the bootloader's ELF file in the following format:
- **<Address> <section symbol> <symbol name>**
- "Extracted_BLE_Info.txt" – the output file that contains filtered information about the symbols that are searched by the masks defined in "Symbol_list.txt". If you want some instance/component to be available in the Bootloader, you can add your own mask in that file. Those symbols names and their addresses are provided to Bootloadable project.
- "Ble_Info_All.txt" – the output file that contains filtered information from the "Extracted_BLE_Info.txt" file for certain required sections (T/t-text, B/b-bss, D/d-

data, R/r-read-only, a-absolute). These sections are listed in the "Section_list.txt" file.

- Ble_Info_All_Sorted.txt – the output file that contains the final sorted by address information required for the Bootloadable project.

All symbol lists for GCC are generated from the content of this file.

Figure 8. Batch Script Workflow



- "Symbol_list" – the input file that should be permanently present at the \Scripts folder. It is not regenerated. It basically contains the list of symbols names masks, such as: "CyBle_", "LED_", "Bootloader_Service_Activation_", "WakeUp_Interrupt_" and so on.

  Do not modify compiler-specific symbols under comment ";GCC". Also, do not delete or modify the "CyBle_" symbol, otherwise the BLE component-related info will not be found. The search is not case-sensitive. We strongly recommend NOT to modify the BLE component name (CyBle), because it is hard to find only proper symbols with such names as Ble, because it takes also different "Table_..." and "Enable_..." symbols that are redundant. "CyBle" is quite unique.

  Therefore, symbols with names that contain any of those masks are filtered from the "ParsedElf.txt" file in "Extracted_BLE_Info.txt". If you want some instance/component to be available in the Bootloader, you can add your own

mask in that file. Those symbols names and their addresses will be provided to the Bootloadable project.

- "Section_list.txt" – the list of sections required for filtering (T/t-text, B/b-bss, D/d-data, R/r-read-only, a-absolute). Do not modify this file, unless you know what you are doing.

3. Run mk.bat file:
   a. Open the mk.bat directory in **Explorer** or other file manager. ([Project path]\Scripts\)
   b. Double click on the mk.bat file.

      If everything is correct, it reports the following:

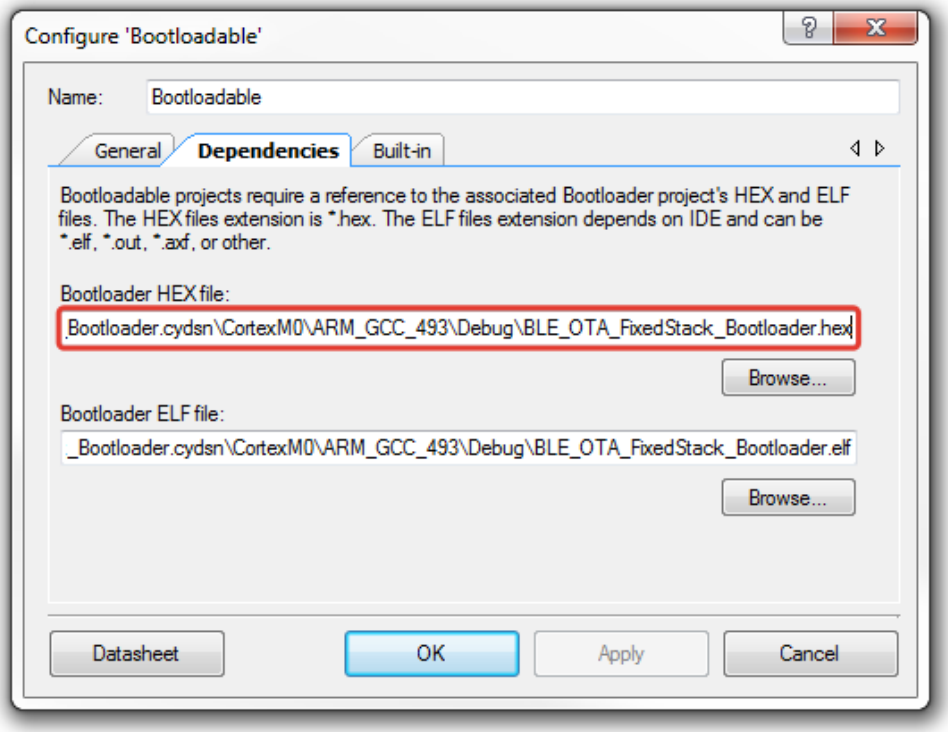
      - Parsing the ELF file...
      - Filtering exported symbols...
      - Filtering by the section list...
      - Sorting by the address...
      - Generating output file(s) ...
      - Press any key to continue …

      No error should be allowed on the output and a compiler-specific file to be created (**Note** Such a file is regenerated each time the mk.bat file runs):

      - BootloaderSymbolsGcc.ld – the file with a list of the PROVIDE() directive for all exported symbols required for the GCC compiler.
      - BootloaderSymbolsMdk.txt – exported symbol definition for MDK compiler
      - BootloaderSymbolsIar.icf – exported symbol definitions for IAR compiler

4. Open the top design schematic of the *BLE_OTA_FixedStack_Bootloadable* project. Specify the path to the bootloader project HEX and ELF files by double-clicking on the Bootloadable component and going to the **Dependencies** tab and link Bootloadable to the BLE_OTA_FixedStack_Bootloader.hex file, as Figure 9 shows.

Figure 9. Bootloadable Component Configuration



5. Program the BLE_OTA_FixedStack_Bootloader project.

   At this point CYC8CKIT-042-BLE contains firmware that can receive new updates over-the-air. The LED3 flashes red.

Upgrading Projects images

To perform any application update, Bootloader project image should be running. If the current running project image is Bootloadable project image, rebooting to Bootloader project image can be done by a SW2 button press. While device is running Bootloader project image code, and is ready to receive Application project image updates (*.cyacd files), the red LDE should be blinking.

There are two software tools that can send Bootloadable project image updates to device:

- Bootloader Host Tool (BHT), which is provided with PSoC Creator.
- CySmart, external software that needs to be downloaded and installed separately.

There are two hardware Dongles that communicate with BLE devices and can be used to update firmware:

- CY5670, with 128KB Flash chip, supports BLE 4.1. It can be used with both BHT and CySmart.
- CY5677, with 256KB Flash chip, supports BLE 4.2. It can be used only with CySmart. It also supports higher transfer rates, up to 1Mbps.

If BHT is used for the update, ensure that the dongle used is CY5670. You can refer to the documentation that came with the dongle to get this information. You can also get this information from CySmart.

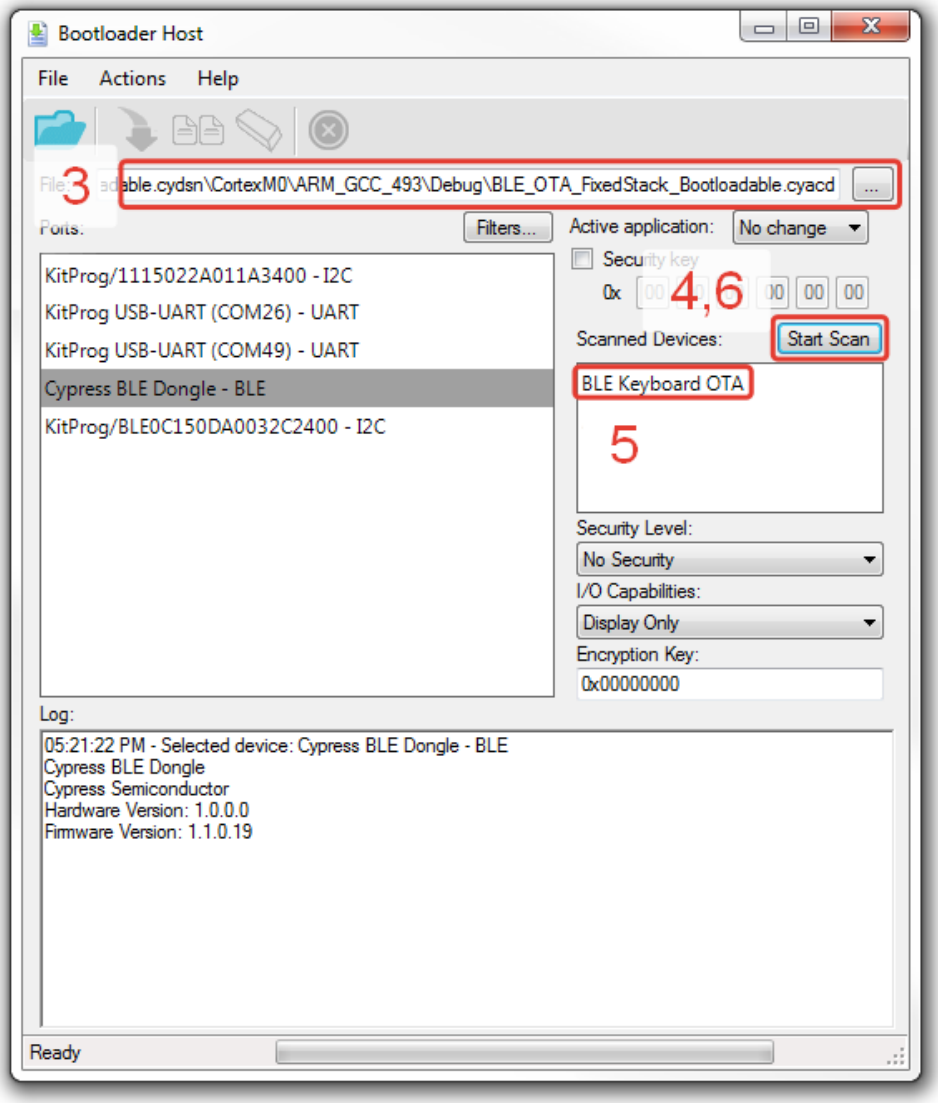
1. Connect your dongle, and install its drivers if you have not done it already.
2. Open CySmart.
3. In the “Select BLE Dongle Target” dialog that appears, look for the dongle name under “Supported devices” or “Unsupported devices”. If the dongle name contains **4.2**, for e.g. "CySmart BLE **4.2** USB Dongle (COM3)", then the dongle supports BLE 4.2 and can be used only with CySmart.
4. If your dongle is not listed by CySmart, click on the **Refresh** button in the “Select BLE Dongle Target” dialog window.

Upgrade Project images with BTH

To upgrade the device firmware OTA follow the next steps:

1. Open the Bootloader Host tool (BHT) by navigating to **Tools > Bootloader Host** in PSoC Creator.
2. In the Bootloader Host Tool select **Cypress BLE Dongle**
3. Press **Open** button and point the path to the *.cyacd file
4. Press the **Start** button next to the **Scanned Devices** field.
5. Wait until the list **BLE Keyboard** in **Scanned Devices** appears and select it.
6. Now press the **Stop** button.
7. Through the **File > Open** menu, point BHT to the *.cyacd file from the appropriate project folder. It is located in the project folder ([*project folder*]\CortexM0\[*compiler name*])

Figure 10. Bootloader Host Tool



8. Press the **Program** button in BHT and wait while a new application image uploads.
9. When image uploading finishes, the Bootloadable Example Project starts to work.
10. If you want to upload another Bootloadable image, press the **SW2** button on the pioneer kit board and repeat steps 1-8.

Note If the update fails at authentication, then check that **Security Level** options are equal to the ones selected in Stack project's BLE component.

Upgrade Project images with CySmart

More detailed information about CySmart usage can be found in CySmart User Guide (section 2.7 Updating Peripheral Device Firmware). You can download the CySmart tool and its user guide from here http://cypress.com/cysmart.

To upgrade the device firmware OTA follow the next steps:

1. Make sure OTA device is running Bootloader project image, and is ready to receive updates.
2. Connect BLE Dongle.
3. Open CySmart software, and select BLE Dongle.
4. Press **Start Scan** button to start scanning for the peripheral device.
5. When the device is listed in the Discovered Device list, stop scanning by clicking the **Stop Scan** button.
6. Select the device from the Discovered Device list.
7. Click **Update Firmware** button.
8. Select the firmware update type on OTA firmware update window.

Firmware Update Type	Description
Application and Stack (Combined) update	Application and Stack firmware co-exist in the same memory location. This option to update complete firmware.
Application only update	Application and stack firmware exist in independent memory locations. This option updates only application firmware.
Application and Stack update	Application and Stack firmware exist in independent memory locations. Select this option to update first the stack firmware followed by application update.

9. Click **Next**. In the next page, browse and select the new firmware image (*.cyacd) file.
10. Click on **Update** button.
11. Wait for the device firmware to be updated.

Note If the update fails at authentication, then check if the security settings in CySmart matches with the settings used in the BLE component of the stack project. To change the security settings in CySmart, click on the **Configure Master Settings** button in the tool and go to the **Security parameters** page.

Expected Results

LED Behavior Description

Bootloader project

Color	State	Description
Red	Blinking	Advertising
Red	Static	Bootloading
White (Red+Green+Blue)	Static	Low power mode

At a reset, when Bootloader runs, the red LED is blinking, which means that the device is advertising. Once connected and an application image is sent from Bootloader Host Tool to the device to be updated, the red LED stops blinking and remains static. This means the device is bootloading.

If the LED color changes to white (all 3 LEDs are on – green, blue, and red), the device goes into the low power mode.

Bootloadable project

Color	State	Description
Green	Blinking	Advertising
Blue	Static	Disconnected
Cyan (Green+Blue)	Blinking	Disconnected + Advertising

At a reset when Application runs, the green LED is blinking, which means that the device is advertising. Once connected, it stops blinking. The blue LED ON indicates that the device is disconnected. The Cyan (2 LED – Green+Blue simultaneously) color means that the device is disconnected and advertising.

After the firmware update process is completed, the device should operate as the BLE_HID_Keyboard example. Refer to the BLE_HID_Keyboard example project datasheet for details.

If there are more changes to the *BLE_OTA_FixedStack_Bootloadable* project, repeat all the steps described for OTA firmware update in the **Setup** and run the **Example Project** section.

Using UART for Debugging

In this example projects UART is used to print various debug information (enabled by default).

File Options.h contains define “DEBUG_UART_ENABLED”. It is set to YES and it provides extra debugging information in each of the bootloader or bootloadable projects. This slightly decreases the projects performance, but it provides extra debugging output to UART. If it is not needed – set it to NO.

A HyperTerminal program is required in the PC to receive debugging information. If you don’t have a HyperTerminal program installed, download and install any serial port communication program. Freeware such as Bray’s Terminal, Putty etc. are available on the web.

1. Connect the PC and kit with a USB cable.
2. Open the device manager program in your PC, find the COM port to which the kit is connected, and note the port number.
3. Open the HyperTerminal program and select the COM port to which the kit is connected.
4. Configure Baud rate, Parity, Stop bits and Flow control information in the HyperTerminal configuration window. By default, the settings are the following: Baud rate – 115200, Parity – None, Stop bits – 1 and Flow control – XON/XOFF. These settings must match the configuration of the PSoC Creator UART component in the project.
5. Start communicating with the device as explained in the project description.

File debug.h contains macros used to print various types of data:

- DBG_PRINT_TEXT(a) - prints a text string.
- DBG_PRINT_DEC(a) – prints a decimal number.
- DBG_PRINT_HEX(a) – prints a hexadecimal number.
- DBG_PRINT_ARRAY(a,b) – prints ‘b’ first elements of array ‘a’.
- DBG_PRINTF(...) – prints the function macro.

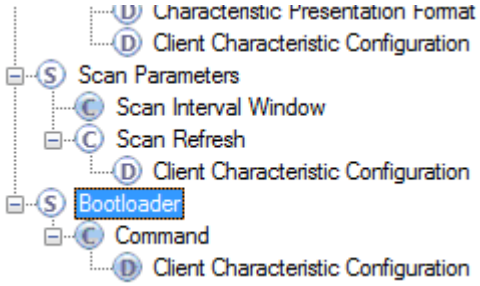
These macros print information to UART only if DEBUG_UART_ENABLED define is set to YES.

Adding OTA Bootloader Support to another BLE Project using shared code

This section describes how to add the OTA shared bootloader capability to your own project with a BLE component.

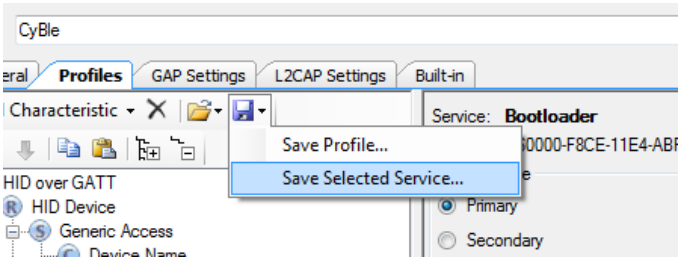
1. Open the BLE_OTA_FixedStack_Bootloader example project.
2. Navigate to project **TopDesign**, open the BLE component customizer by double clicking on the CyBle component and select the Bootloader Service Top node:

Figure 12. Bootloader Service Top Node



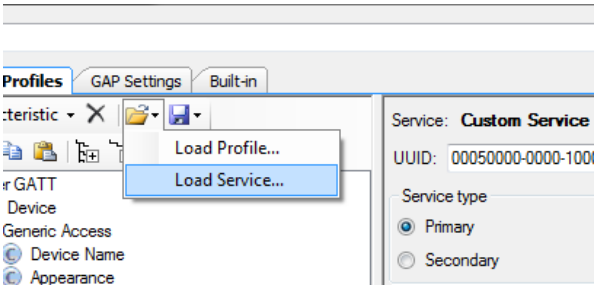
3. Now you can export the Bootloader Service into a file:

Figure 13. Exporting BLE service configuration



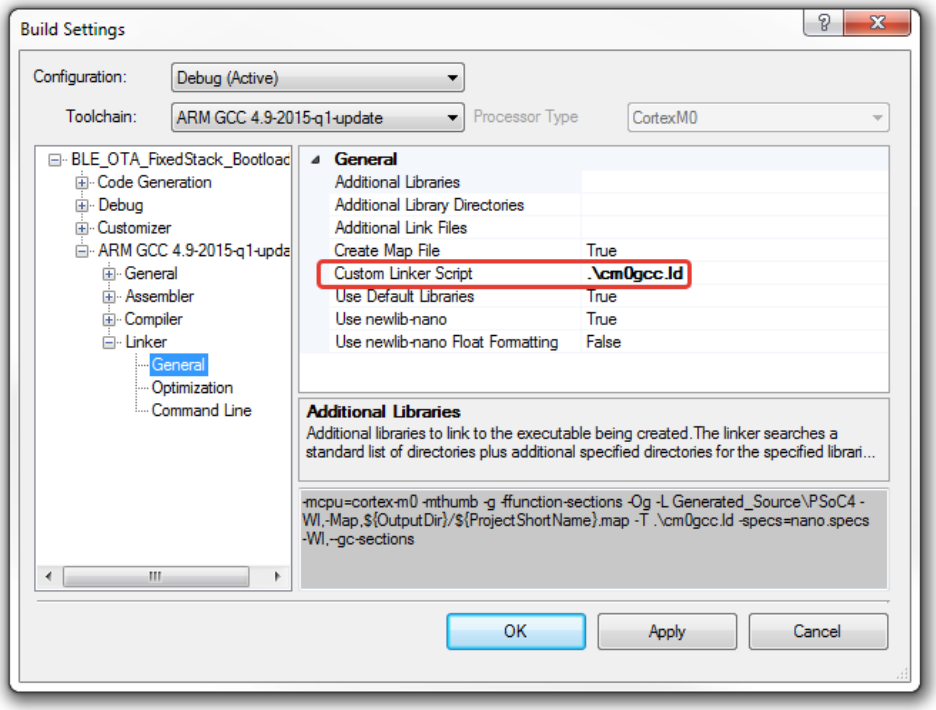
4. Once the BLE Bootloader Service configuration is saved, it can be imported to any project with a BLE component:

Figure 14. Loading service configuration



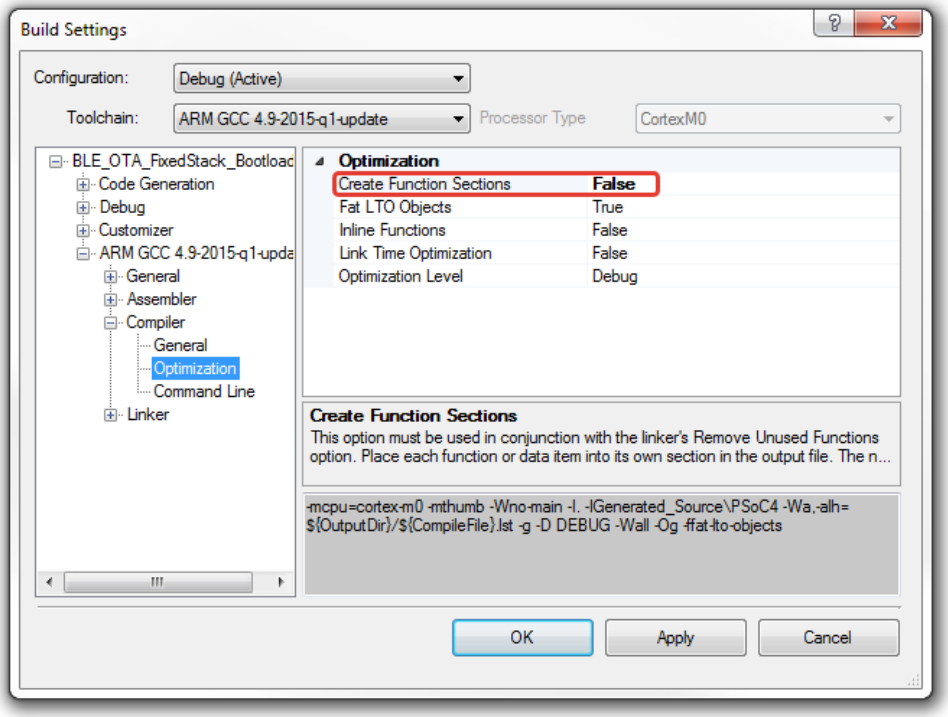
5. To enable the Bootloader Service to receive long packets, the MTU size of the BLE component does not matter.
6. You can copy a bootloader component symbol from the BLE Shared Memory Bootloader example project to your project. Change the project type to bootloader. This can be done by selecting the project, and selecting **Project > Build Settings** and changing Application Type to Bootloader.
7. You need to copy all application layer code from the project that you are converting to a new project and configure it to be bootloadable.
8. Copy all the linker files and scripts from example projects (BLE_OTA_FixedStack_Bootloader/ BLE_OTA_FixedStack_Bootloadable) to your bootloader/bootloadable projects respectively maintaining their placement in the project folders and change the projects names and paths in the mk.bat script.
9. Enable option Custom Linker Script in the bootloader project by selecting the copied linker script in the build settings of the bootloader project:

Figure 15. Custom Linker script configuration:



10. Change the option of the compiler to generate functions sections:

Figure 16. Compilation settings:



11. Now the bootloader project can be compiled.
12. After the bootloader project compilation, batch script mk.bat should be executed. It analyzes the symbols placement in the bootloader image and provide them to the bootloadable project linker script.
13. You can compile your bootloadable project.
14. If there are some compilation errors related to the missing APIs/symbols, you should declare these APIs/symbols as external ones, as it is done in the files OTAMandatory.c/ OTAMandatory.h. This is also true for typedefs/structures or enums (OTAMandatory.h).
15. Now all the changes made to the bootloadable project can be updated OTA. **Note** changes that are made to the bootloader project can be updated only via reprogramming the device.

Functional description

The projects operation is slightly different for GCC, MDK, and IAR compilers with the same approach. The Bootloader project works this way in any generic case. On the other hand, the Bootloadable project uses APIs from the bootloader project and, therefore initializes global and static variables for them by creating two separate segments in RAM – one for bootloader APIs and one for the bootloadable project code. Afterwards, bootloadable project calls either the function generated by the compiler or the custom initialization function to initialize the bootloader APIs RAM.

GCC Compiler

For the GCC compiler, the bootloadable project uses the custom function to initialize the bootloader APIs RAM by passing the bootloader segments addresses to the bootloadable application and copying data from ROM to RAM.

MDK compiler

For the MDK compiler a similar approach is used, but instead of writing a function to initialize RAM the function, already existing in the bootloader project is used. To be able to return to the bootloadable project, the return point address is passed through the not initialized segment of RAM.

IAR compiler

The IAR compiler uses roughly the same approach, but RAM separation should be done manually. After the bootloader project is compiled, the amount of used RAM should be compared

to one, specified in row 48 of the IAR linker script in the bootloadable project. If the size is bigger, increase the number specified in the mentioned line.

Changing projects compiler

Due to the specific implementation of the projects, the compiler should be the same for the bootloader and bootloadable projects. By default the projects are configured to work with the GCC compiler but this can be changed.

MDK compiler

To use the example project with the MDK compiler, follow the next steps:

1. Change the compiler to MDK Generic in the project build settings.
2. Open file mk.bat with Notepad or any other text editor. It is stored in the **Scripts** subfolder of bootloader project.
3. Find the line that contains setting COMPILER variable and change it to MDK_COMPILER:

```
33  ::
34  ::  Compiler choice. Currently only GCC 4.9.3, MDK and IAR compilers are supported.
35  ::
36  set GCC_COMPILER=ARM_GCC_493
37  set IAR_COMPILER=ARM_IAR_Generic
38  set MDK_COMPILER=ARM_MDK_Generic
39
40  set COMPILER=%GCC_COMPILER%
```

Note The symbol '%' should be present on both sides of the value.

4. Build the bootloder project.
5. Verify the path to the bootloader *.elf file as it is described in section **Setup and Run Example Project.**
6. Run the mk.bat script.
7. Compile the bootloadable project and program it.

IAR compiler

To use IAR compiler, export both bootloader and bootloadable projects to IAR IDE as described in PSoC® Creator™ User Guide with the only exception: the linker script for the bootloadable project; you should use the one that is provided in the **LinkerScripts** folder of the bootloadable project.

After the projects are exported and configured, you should also verify that the RAM segment for the bootloader is big enough. This can be done by getting the amount of RAM used in the

bootloader project and comparing it to the size of the BTLDRRAM block defined in the IAR linker script:

```
48  define block BTLDRRAM   with alignment = 8, size = 0x2AFF { readwrite section .bootloaderram };
```

If the amount of RAM used by the bootloader is bigger, the size of the section should be increased, so that the RAM left in the bootloadable was big enough to contain the heap and stack.

Also, the mk.bat file should be modified:

Replace the LOADER_ELF_FILE variable value with the path to the *.out file that is generated by IAR after building the bootloader project:

```
58  :: Bootloader's *.elf file path.
59  set LOADER_ELF_FILE=%LOADER_PRJ_PATH%%BUILD_OUTPUT_DIR%%LOADER_PRJ_NAME%.elf
```

to something like:

```
58  :: Bootloader's *.elf file path.
59  set LOADER_ELF_FILE=C:\Users\User\Desktop\BLE_Shared_Memory_Bootloader01\BLE_Shared_Memory_Bootloadable01.
        cydsn\Debug\Exe\BLE_Shared_Memory_Bootloadable01.out
```

Also, in the batch file that is located in subfolder **\Export** of the bootloadable project (**prebuild.bat**) you should replace path to the *.elf file that is generated in PSoC Creator with the path to the *.out file of the bootloader project. This is needed for the elf tool to include the correct bootloader image into the final *.hex image of bootloadable. It is required only if you plan to program the bootloadable *.hex file into the device and it is not required for generation of the correct *.cyacd file.

Configuring projects for other Cypress BLE devices

BLE OTA Fixed Stack example projects use custom linker scripts so PSoC Creator will not be able to configure linker scripts to provide correct linking and placement. This section describes steps that are to be done if you are not using CY8C4247LQI-BL483 or CYBL10563-56LQXI devices or devices that have different size of RAM or ROM memory.

Following sections describe linker scripts modification after you have changed the device in PSoC Creator.

Steps for GCC compiler

For GCC compiler changes are required for both bootloader and bootloadable linker scripts:

```
21  MEMORY
22  {
23    rom (rx) : ORIGIN = 0x0, LENGTH = 131072
24    ram (rwx) : ORIGIN = 0x20000000, LENGTH = 16384
25  }
26
27
28  CY_APPL_ORIGIN      = 0;
29  CY_FLASH_ROW_SIZE   = 128;
30  CY_APPL_NUM         = 1;
31  CY_METADATA_SIZE    = 64;
32
```

1
2
3

1. Change device flash memory size
2. Change device RAM size
3. Change device row size

Linker scripts are located in ***.\cm0gcc.ld*** for bootloader project and in ***.\LinkerScripts\cm0gcc.ld*** for bootloadable project.

If you are not sure in values that are to be entered for your device you can refer to the values that are in linker script that is generated by PSoC Creator despite it is not used. It is located in the folder: ***%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot*** and has name ***cm0gcc.ld***.

Steps for MDK compiler

In case of MDK compiler only linker script of bootloadable project is to be changed:

```
#include "BootloaderSymbols.h"

#define CY_FLASH_SIZE      131072
#define CY_FLASH_ROW_SIZE  128
#define CY_METADATA_SIZE   64
```

1. Change device flash memory size
2. Change device row size

And a bit further:

```
    ARM_LIB_HEAP (0x20000000 + 16384 - 0x400 - 0x1000) EMPTY 0x400
    {
    }

    ARM_LIB_STACK (0x20000000 + 16384) EMPTY -0x1000
    {
    }
```

3. Change device RAM size (both values)

Linker script is located in folder ***.\LinkerScripts\Cm0Mdk.scat***

If you are not sure in values that are to be entered for your device you can refer to the values that are in linker script that is generated by PSoC Creator despite it is not used after project rebuild. It is located in the folder: ***%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot*** and has name ***Cm0Mdk.scat***.

Steps for IAR compiler

If you use IAR the linker script of bootloader project is safe to use one that is generated by PSoC Creator after device vas changed in PSoC Creator. But the bootloadable project linker script has to be modified:

```
6  define symbol __ICFEDIT_intvec_start__ = 0x00000000;
7  /*-Memory Regions-*/
8  define symbol __ICFEDIT_region_ROM_start__   = 0x0;
9  define symbol __ICFEDIT_region_ROM_end__     = 131072 - 1;
10 define symbol __ICFEDIT_region_RAM_start__   = 0x20000000;
11 define symbol __ICFEDIT_region_RAM_end__     = 0x20000000 + 16384 - 1;
12 /*-Sizes-*/
13 define symbol __ICFEDIT_size_cstack__ = 0x0800;
14 define symbol __ICFEDIT_size_heap__   = 0x400;
15 /**** End of ICF editor section. ###ICF###*/
16
17
18 /******** Definitions ********/
19 define symbol CY_FLASH_SIZE     = 131072;
20 define symbol CY_APPL_ORIGIN    = 0;
21 define symbol CY_FLASH_ROW_SIZE = 128;
22 define symbol CY_APPL_LOADABLE  = 1;
23 define symbol CY_APPL_LOADER    = 0;
24 define symbol CY_APPL_NUM       = 1;
25 define symbol CY_METADATA_SIZE  = 64;
```

1. Change device flash memory size
2. Change device RAM size
3. Change device row size

Linker script is located in folder ***.\LinkerScripts\Cm0Iar.icf***

If you are not sure in values that are to be entered for your device you can refer to the values that are in linker script that is generated by PSoC Creator despite it is not used after project rebuild. It is located in the folder: ***%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot*** and has name ***Cm0Iar.icf***.

Updating BLE component to other version

Following steps are required to be done if you update the bootloader component version of BLE component:

1. Ensure you're using the desired version of BLE component in the bootloader project;
2. Rebuild the bootloader project;
3. Locate CYBLE_EVT_T enum declaration in the bootloadable project (for the BLE_OTA_FixedStack_Bootloadable example project this is **.\OTAMandatory.h** starting from line 1539) and replace it with CYBLE_EVT_T enum declaration that was generated in the bootloader project (BLE_OTA_FixedStack_Bootloader example project this is **.\Generated_Source\PSoC4\BLE_eventHandler.h** starting from line 146);
4. Run the mk.bat script that is located in the bootloader project in Scripts subfolder;

5. After updating the BLE component in the bootloader project you have to program new bootloader image to the device;