SimpleLink™ *Bluetooth*® low energy CC26X0 Wireless MCU

Over-the-Air Download User's Guide

For BLE-Stack™ Version: 2.2.2

Table of Contents

Table of Contents

1. Introduction 5
  1.1 Purpose 5
  1.2 Scope 5
  1.3 Document Updates 6
  1.4 Legacy OAD Versions 6
  1.5 Supported OAD Downloaders 6
2. OAD Concept Overview 7
  2.1 OAD Types 7
    2.1.1 On-chip OAD 7
    2.1.2 Off-chip OAD 8
  2.2 OAD Topology Overview 8
  2.3 OAD Image Metadata 9
    2.3.1 CRC and CRC Shadow 9
    2.3.2 Version 10
    2.3.3 Length 10
    2.3.4 User Identification (UID) 10
    2.3.5 Start Address 10
    2.3.6 Image Type 10
    2.3.7 Image State 11
  2.4 OAD Service Description 11
    2.4.1 OAD Image Identify (0xFFC1) 13
    2.4.2 OAD Image Block Characteristic (0xFFC2) 14
    2.4.3 OAD Image Count Characteristic (0xFFC3) 14
    2.4.4 OAD Image Status (0xFFC4) 14
  2.5 OAD Process 15
    2.5.1 Initiation of the OAD Process 15
    2.5.2 Image Block Transfers 17
    2.5.3 Completion of the OAD Process 17
  2.6 OAD Reset Service 17

2.6.1 OAD Reset (0xFFD1) 17
2.7 Bootloader 18
3. Off-Chip OAD 18
3.1 Off-chip OAD Memory Map 19
3.2 Constraints and Requirements for Off-chip OAD 20
3.3 Conditions for rejecting Metadata 20
3.4 BIM for Off-chip OAD 20
4. On-Chip OAD 22
4.1 On-chip OAD Memory Map 22
4.2 Constraints and Requirements for On-chip OAD 23
4.3 Conditions for Rejecting Metadata 24
4.4 BIM for On-chip OAD 24
5. Running the Out of the Box Demos 26
5.1 Required Hardware 26
5.2 Required Software 26
5.3 OAD Downloader 26
5.3.1 Setting Up the CC2650 LaunchPad 26
5.3.2 Setting up and connecting to BTool 27
5.4 OAD Target 27
5.4.1 On-Chip OAD 28
5.4.2 Off-Chip OAD 29
5.5 Changing the Device Name to Validate OAD 30
5.6 Performing an OAD with BTool 31
5.6.1 (On-chip OAD only) Reset the device via the OAD Reset Service 31
5.6.2 Performing an OAD 32
6. Adding OAD to an Existing Application 33
6.1 Off-chip OAD 34
6.1.1 Project Changes For IAR 34
6.1.2 Project Changes for CCS 35
6.1.3 Source Code Changes 37
6.1.4 [Optional] Changing the pins of the external flash device 38
6.2 On-chip OAD 39
6.2.1 Steps for IAR 39

6.2.2 Source Code Changes 42
7. TI OAD Image Tool 42
7.1 Dependencies 42
7.2 Using the tool 43
7.3 Building a Production Image 44
8. Appendix 46
8.1 References 46
8.2 Definitions, Abbreviations, Acronyms 47
8.3 RTOS RCFG Section 47
8.4 Prevent On-chip BIM from checking the CRC 47
9. Troubleshooting Guide 48
9.1 General Troubleshooting Guide 48
9.2 Downloaded OAD Image Isn't Starting 48
9.3 Should External Flash be used during OAD? 48
9.4 Individually Flashing Hex Files for On-Chip OAD - App doesn't work! 48
9.5 Debugging BIM/ Pre application 49

Table of Figures/Diagrams

Figure 1. Two Types of OAD 7
Figure 2. OAD Downloader and Target 8
Figure 3. Metadata description 9
Figure 4. Off-chip OAD Image Types 11
Figure 5. OAD Service Overview 12
Figure 6. OAD Service Description 12
Figure 7. Reject Notification in Sniffer Capture 13
Figure 8. Successful OAD Initiation Sniffer Capture 14
Figure 9. Block Request/Response Sniffer Capture 14
Figure 10. OAD Status Codes 15
Figure 11. OAD Sequence Diagram 16
Figure 12. OAD Reset Service 17
Figure 13. Off-chip OAD Target Memory Partition 19
Figure 14. Functional Overview of Off-chip BIM 21
Figure 15. On-chip OAD Target Memory Partition 23
Figure 16. Functional Overview of On-chip BIM 25
Figure 17. OAD Downloader HostTest Setup 27
Figure 18. OAD On-chip Target Setup 29

Figure 19. OAD Advertisement Data before change....30
Figure 20. OAD Advertisement Data after Change....30
Figure 21. OAD Device Name field modified....30
Figure 22. Read all characteristic data in BTool....31
Figure 23. Writing to OAD Reset pt 1....31
Figure 24. Writing to OAD Reset pt 2....32
Figure 25. Sending an OAD Image over the air....33
Figure 26. Verifying an OAD completed....33
Figure 27. OAD Off-Chip Files in IAR....35
Figure 28. OAD Off-chip files in CCS....37
Figure 29. OAD configuro prebuild step (IAR)....40
Figure 30. On-Chip OAD Boundary (compiler)....40
Figure 31. On-Chip OAD Boundary (linker)....41
Figure 32. On-Chip OAD Workspace (IAR)....42
Figure 33. OAD Image Tool Arguments....44
Figure 34. OAD Image Tool Output....44
Figure 35. OnChip OAD Production Image Example invocation.....45
Figure 36. Off-Chip OAD Production Image Example invocation....46

1. Introduction

1.1 Purpose

This document describes the process by which a developer can enable a SimpleLink™ Bluetooth® low energy CC26X0 wireless MCU project and application to successfully implement the TI OAD Profile to remotely upgrade the image on a CC26X0 BLE device, a process further referred to as an Over-the-Air Download (OAD). This process provides tremendous value to a BLE product solution, as a target device does not need to be physically accessed to provide a software upgrade. Our purpose here is to make OAD simple by providing detailed instructions from enabling OAD in an application project to receiving the new image over the air, verifying its correctness, and running it from the bootloader.

1.2 Scope

This document provides instruction on how to setup a BLE-Stack™ project to be OAD enabled, such as our example simple_peripheral project for OAD. An overview of the OAD architecture and how to build, download, and debug the components shall be discussed here as well. Details about the BLE runtime system on CC26xx devices and the interrupt vector tables will be discussed as is necessary to elucidate how an OAD build differs from a project which does not provide OAD capability.

There are two different types of OAD. One is On-chip OAD which doesn't require any additional hardware and the other is Off-chip OAD which supports a system where an external flash memory is equipped. In the sample application projects, it is assumed that the hardware is the CC2650 LaunchPad where 1MB external flash memory is connected to the CC26X0 via SPI.

OAD for the sensortag project is not specifically mentioned in this guide since it uses the same Off-chip OAD method as the simple_peripheral project, see the SensorTag User's Guide wiki for more information on the SensorTag:
http://processors.wiki.ti.com/index.php/CC26X0_SensorTag_User's_Guide.

Note: On-chip OAD is currently only supported by IAR with CC26X0 devices, while Off-chip OAD works with both IAR and CCS.

NOTE: This kit supports development of Bluetooth low energy applications on the following SimpleLink wireless MCUs: CC2640 and CC2650. The multi-standard CC2650 wireless MCU supports Bluetooth low energy as well as other wireless protocols, such as ZigBee® and 6LoWPAN. The CC2640 supports Bluetooth low energy only. All code generated from the BLE-Stack v2.x SDK is binary compatible and exchangeable with both the CC2650 and CC2640 wireless MCUs. Throughout this document, CC26X0 will be used to refer to both devices.

Also, it is assumed that the developer is familiar with the CC26X0 Software Developer's Guide [1], including the dual-image architecture used by the SDK.

For additional support, please visit the following online resources:

- TI Bluetooth LE Wiki-page: www.ti.com/ble-wiki
- TI E2E support forum: www.ti.com/ble-forum

1.3 Document Updates

This document is constantly being improved. For the best user experience, please ensure that you are using the latest version. Newer revisions will be posted online on the OAD User's Guide wiki page.

1.4 Legacy OAD Versions

Prior to the BLE-Stack v2.2.0 release, there were three versions of OAD supported by the SDK:

1. Off-chip OAD with preserved page 0 using BIM
2. Off-chip OAD without preserved page 0 using Baseloader
3. On-chip OAD using BIM

In BLE-Stack SDK versions prior to 2.2.0, the SensorTag embedded project supported the baseloader (Method 2) from above. In conjunction with the embedded project, the SensorTag mobile applications (iOS, Android) only supported the baseloader method (Method 2). This method has been discontinued* as of BLE-Stack v2.2.0, all projects (including the SensorTag) have now standardized on Methods 1 and 3.

This guide (and its previous versions) will describe the BIM based on On-chip and Off-chip methods (Method 1 and Method 3 above).

The baseloader method has been discontinued because there is a risk where a device may become bricked (require wired JTAG access to recover) if a power loss is experienced while updating page 0

1.5 Supported OAD Downloaders

The recommended OAD Downloader for use with this document is BTool v1.41.17. **This is different than the BTool version that is bundled with this BLE-Stack install.** You can download BTool v1.41.17 from CC2640R2 SDK Download Link. This specific version of BTool has been selected because it has support for the legacy TI OAD profile and the HCI command set supported by v1.41.17 is the most compatible with the CC2650 LaunchPad host_test image.

TI-provided mobile applications are out of the scope of this document.

2. OAD Concept Overview

This section aims to explain the major concepts involved in the OAD process from a high level. The concepts here will be expanded on further in the following sections. Some concepts, such as the Boot Image Manager (BIM) may vary in their implementation details. Wherever possible, the concepts will be covered in this chapter with their implementation details covered in the following chapters.

2.1 OAD Types

There are two methods of performing an over the air download: On-chip and Off-chip. The key difference between the two methods is where the downloaded image is to be stored during the OAD. In On-chip OAD, the downloaded image is written to internal flash, allowing for a single chip OAD solution. Off-chip OAD stores the downloaded image in an external flash part, requiring a two chip OAD solution. A graphic showing the different OAD methods is shown below (Figure 1). Each type of OAD has associated tradeoffs and benefits which will be discussed below and then covered in depth in their respective sections

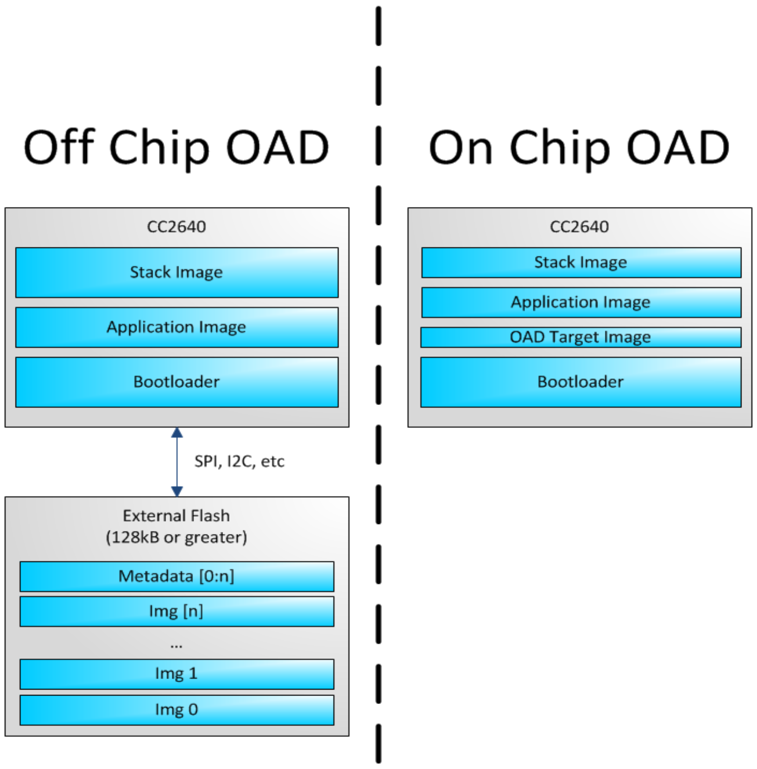


Figure 1. Two Types of OAD

2.1.1 On-chip OAD

On-chip OAD is best suited for applications with a very small flash footprint and very basic functionality.

The BOM cost is also reduced for on-chip applications because no external flash part is needed. The removal of the external flash will also free up GPIO pins to be used for other applications.

The BLE-Stack instance cannot be updated in on-chip OAD, and thus all application updates must respect the flash and RAM boundaries of the stack that is permanently resident on the device. Additionally, in an on-chip OAD system, the device must reboot from the user application in order to perform an OAD.

2.1.2 Off-chip OAD

Off-chip OAD is best suited for complex applications that require a larger flash memory footprint.

The tradeoff with off-chip OAD is that the incoming image must be stored in external flash and thus an external flash part is required. Additionally, the application must have enough free GPIOs to interface to the external flash.

Both BLE-Stack and user application can be updated via OAD. The User application/functionality will be able to operate while the OAD is in progress.

2.2 OAD Topology Overview

Two BLE capable devices are required for performing an OAD. The terms for the devices involved in an OAD exchange are listed below:

- **OAD Target:** The device whose firmware is being upgraded over the air. This is assumed to be a CC26xx device running the TI OAD service.
- **OAD Downloader:** The device responsible for accepting an OAD enabled image from the compiler and transferring it over the air to the OAD target.

OAD roles are independent of the GAP role; they are dependent on which device exposes the OAD service. The OAD target is always the device that runs the OAD service (GATT server), and the OAD downloader is always the device that consumes the OAD service (GATT client).

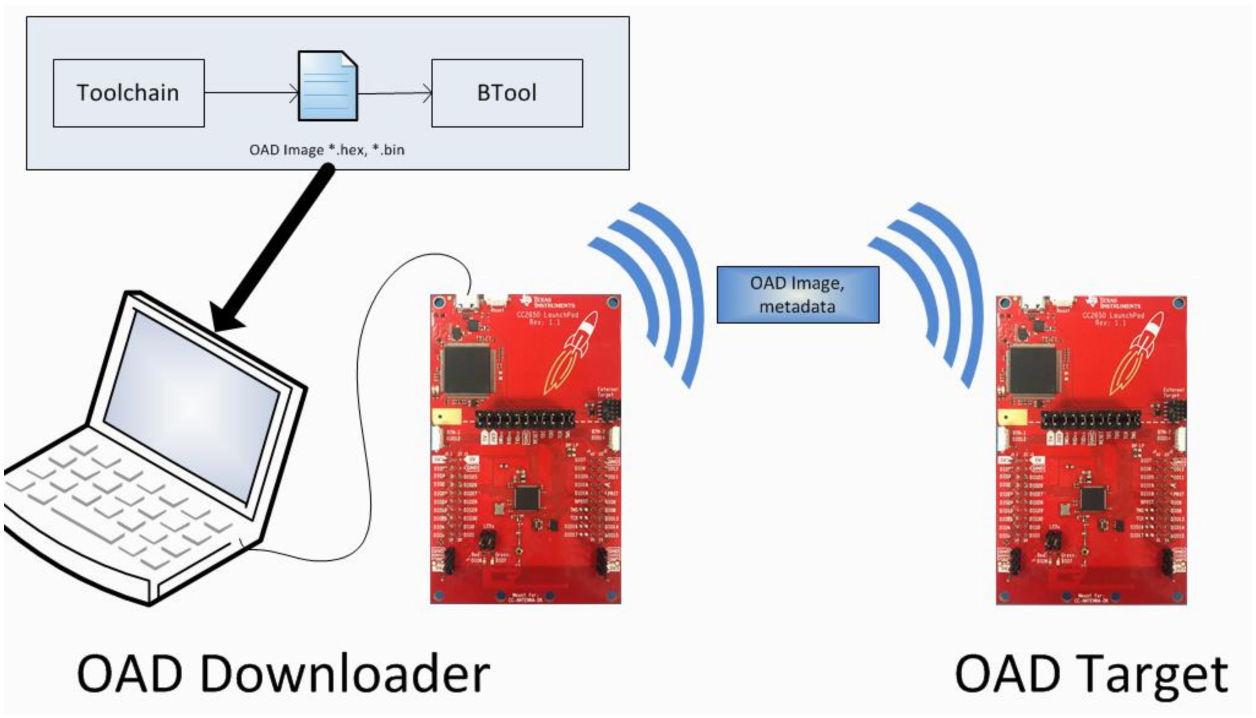


Figure 2. OAD Downloader and Target

All provided TI example applications (BTool, mobile applications, etc.) are implemented such that the OAD Target is a peripheral device, and the OAD Downloader is a central device. For this reason, other configurations are outside of the scope of this document.

2.3 OAD Image Metadata

There are many points in the OAD process where information must be gathered about images. This information can be used by the OAD service to determine whether or not an image is acceptable for download or by the bootloader to determine which image should be run. In order to prevent this information from being calculated multiple times and to assist in the linking process, all TI OAD images use a standard 16-byte metadata vector. This metadata vector is embedded at the beginning of the image, occupying the first 16 bytes before the application code. This section aims to explain the various fields within the metadata vector and what they mean. The following sections will describe how the fields are used specifically for On-chip and Off-chip OAD.

Figure 3 below shows a description of the metadata vector.

Field	Size (in bytes)	Description
CRC	2	Cyclic Redundancy Check
CRC Shadow	2	Place holder for CRC
Version	2	Version
Length	2	Length of the image in words*
UID	4	User Identification
Start Address	2	The destination address of the image in words*
Image Type	1	The type of image to be downloaded
State	1	The status of this image

Figure 3. Metadata description

Note that the above fields that are marked with an asterisk * are measured in 32-bit words. For example, an image length of 0x100 describes an image that is 1024 bytes in size. This OAD word size is defined by EFL_OAD_ADDR_RESOLUTION for off-chip OAD and by HAL_FLASH_WORD_SIZE for on-chip OAD.

2.3.1 CRC and CRC Shadow

The cyclic redundancy check (CRC) is a means to check an image to ensure that it has not become corrupted. This must be done in two steps. First the CRC must be calculated when the image is generated from the toolchain, this will be stored in the CRC field in the metadata vector. This initial CRC will be sent over the air via the OAD service (see section 2.4). Later, the target will need to ensure that the image has not been corrupted during transfer. The target will then re-calculate the CRC of the downloaded image; this will be stored in the CRC shadow field of the metadata vector.

If the CRC and CRC shadow are equivalent, the target can assume that the image was not corrupted while sending over the air.

The algorithm selected for CRC calculations is the CRC-16-CCITT, it is a 16 bit CRC calculation that boasts a 99.9984% error detection rate in the worst case.

2.3.2 Version

The image version field is used to track revisions of images and ensure upgrade compatibility. Customers may implement their own versioning scheme. See the appendix for more information about OAD version checks

2.3.3 Length

The length field is the length of the image in words, where the word size is defined by EFL_OAD_ADDR_RESOLUTION and HAL_FLASH_WORD_SIZE for On-chip and Off-chip OAD respectively. Off-chip OAD customers who are using different external flash parts may need to modify EFL_OAD_ADDR_RESOLUTION to match the word size of their part. For On-chip OAD, the word size of the CC26X0 is fixed.

2.3.4 User Identification (UID)

This field is un-used by the TI OAD profile, but the hooks are in place for a customer to add their own implementation of verifying images based on UID.

For on-chip OAD, the convention is that Image A will embed ‘A’, ‘A’, ‘A’, ‘A’ and Image B will embed ‘B’, ‘B’, ‘B’, ‘B’. Off-chip images use ‘E’, ‘E’, ‘E’, ‘E’ by default. See On-chip OAD section for explanation of image A vs Image B.

2.3.5 Start Address

The start address is the first address where the proposed image is to be stored in internal flash. Similar to the length field, this is calculated in words. Off-chip OAD solutions put restrictions on the start address based on image type (more on this in the next section).

Note that for On-chip OAD solutions, this field is reserved in the metadata as they use a fixed start address that is based on the internal flash memory map (OAD_IMG_D_PAGE).

2.3.6 Image Type

In Off-chip OAD systems with external flash, there are multiple types of images that can be uploaded. These image types include: App + Stack, App only, Network Processor, or Stack only.

Note: While BLE Stack only upgrades are possible, the user must be sure that the App/Stack boundary has not changed between the resident OAD image and the proposed OAD image. Since there are no runtime checks on the App/Stack boundary, a Stack only OAD will overwrite the resident application if the boundary has grown. Users should exercise care when using this option.

If a boundary change is required (i.e. BLE stack is growing or shrinking), it is recommended that a user perform a merged update (App+Stack) to ensure that the OAD image is ready to run.

Note that for on-chip OAD solutions this field is reserved in the metadata as they determine image type based on the least significant bit of the image version field as discussed above.

The supported image types are listed below:

Image Type	Value	Description
EFL_OAD_IMG_TYPE_APP	1	An application or application + stack merged update
EFL_OAD_IMG_TYPE_STACK	2	A stack only update
EFL_OAD_IMG_TYPE_NP	3	A network processor update. This only applies to the SimpleAP + SimpleNP demo
EFL_OAD_IMG_TYPE_FACTORY	4	Describes the permanently resident production image that runs on the device before any OTA updates.

Figure 4. Off-chip OAD Image Types

2.3.7 Image State

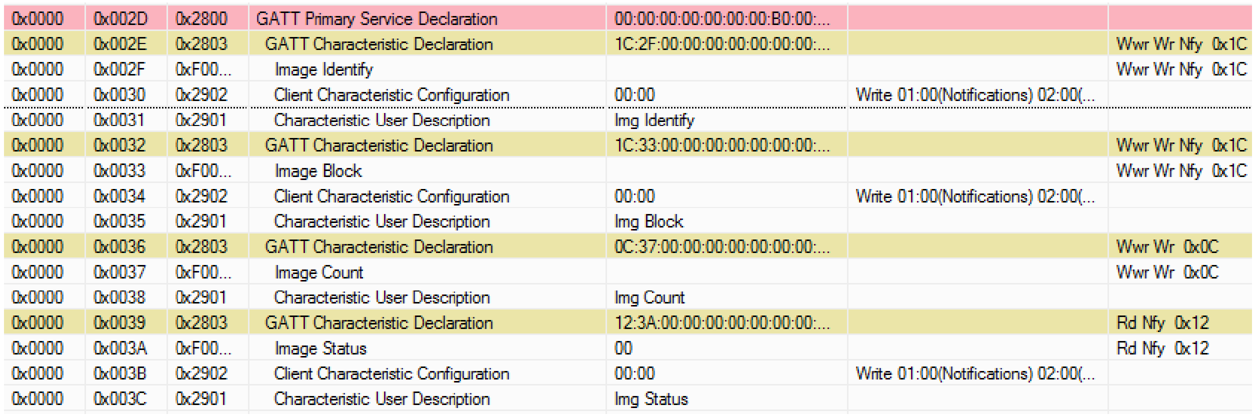
The image state is a one byte metadata field that is used only by Off-chip OAD solutions. The state informs the bootloader whether or not the image is ready to run or currently running. This prevents the bootloader from copying the same image from external to internal flash on every boot.

Note that for On-chip OAD solutions this field is reserved in the metadata as the OAD reset service handles switching between images in the bootloader.

2.4 OAD Service Description

The OAD service has been designed to provide a simple and customizable implementation for the customer. In its most rudimentary form, this service is responsible for accepting/rejecting an OAD interaction based on image header criteria, storing the image in its appropriate location, and causing a device reset if the download is successful so that the downloaded application image is run by the BIM.

A screenshot of BTool displaying the OAD service is shown below.



0x0000	0x002D	0x2800	GATT Primary Service Declaration	00:00:00:00:00:00:00:B0:00:...		
0x0000	0x002E	0x2803	GATT Characteristic Declaration	1C:2F:00:00:00:00:00:00:00:...		Wwr Wr Nfy 0x1C
0x0000	0x002F	0xF00...	Image Identify			Wwr Wr Nfy 0x1C
0x0000	0x0030	0x2902	Client Characteristic Configuration	00:00	Write 01:00(Notifications) 02:00(...	
0x0000	0x0031	0x2901	Characteristic User Description	Img Identify		
0x0000	0x0032	0x2803	GATT Characteristic Declaration	1C:33:00:00:00:00:00:00:00:...		Wwr Wr Nfy 0x1C
0x0000	0x0033	0xF00...	Image Block			Wwr Wr Nfy 0x1C
0x0000	0x0034	0x2902	Client Characteristic Configuration	00:00	Write 01:00(Notifications) 02:00(...	
0x0000	0x0035	0x2901	Characteristic User Description	Img Block		
0x0000	0x0036	0x2803	GATT Characteristic Declaration	0C:37:00:00:00:00:00:00:00:...		Wwr Wr 0x0C
0x0000	0x0037	0xF00...	Image Count			Wwr Wr 0x0C
0x0000	0x0038	0x2901	Characteristic User Description	Img Count		
0x0000	0x0039	0x2803	GATT Characteristic Declaration	12:3A:00:00:00:00:00:00:00:...		Rd Nfy 0x12
0x0000	0x003A	0xF00...	Image Status	00		Rd Nfy 0x12
0x0000	0x003B	0x2902	Client Characteristic Configuration	00:00	Write 01:00(Notifications) 02:00(...	
0x0000	0x003C	0x2901	Characteristic User Description	Img Status		

Figure 5. OAD Service Overview

The OAD service is a primary service with four characteristics. The characteristics of the OAD service, their UUIDs, and descriptions are listed in Figure 7.

Note that the characteristics use the 128-bit TI base UUID of the format F000XXXX-0451-4000-B000-000000000000 where XXXX is their shortened 16bit UUID. For brevity, this document will refer to the characteristics by their 16-bit short UUID.

UUID	Name	Description
0xFFC0	OAD Service	OAD service declaration
0xFFC1	Image Identify	Used to send image properties (metadata) over the air so that the OAD target device can determine if it should accept or reject the proposed image
0xFFC2	Image Block	Actual block of image data along with offset into the image.
0xFFC3	Image Count	Number of complete images to be downloaded in the OAD session
0xFFC4	Image Status	Status of current OAD download

Figure 6. OAD Service Description

The primary method for sending data from the OAD downloader to the OAD target is the GATT writes with no response message. GATT notifications are the primary method used to send data from the target to the downloader. This communication scheme was selected to prevent the target device from having to include the GATT client code required in order to receive notifications from the downloader. The downloader shall register for notifications from any characteristic with a CCCD (by writing 01:00 to the CCCD).

Note that both GATT notifications and GATT write with no response are non-acknowledged message types. This means that in poor RF conditions, the OAD process may not be successful. There is an inherent tradeoff between the speed of the OAD process and its reliability. Implementing a reliable OAD communication protocol (with retries, acknowledgments, etc.) is outside the scope of this document.

For a message sequence chart describing the OAD process in terms OAD service messages exchanged between the target and downloader please see Figure 11.

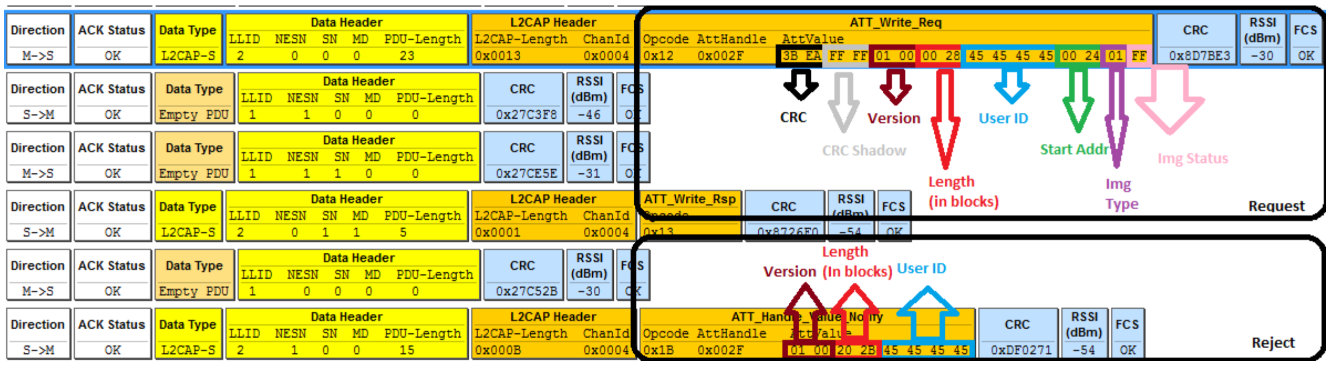
2.4.1 OAD Image Identify (0xFFC1)

The Image Identify characteristic is used to exchange image metadata between downloader and target. The OAD process begins when the downloader sends the 16 byte metadata of the proposed OAD image to the target. Upon receiving the candidate metadata, the target will do some calculations to determine whether or not the proposed image should be downloaded. “01:00” shall be written to the CCCD of this characteristic so that notification for metadata rejection is enabled.

Note that the conditions under which an OAD is accepted vary slightly between the on and off-chip methods. Please see the respective sections for more information about image reject conditions.

If the target accepts the image it will continue the OAD process by sending a notification on the Image Block characteristic requesting the first block. Otherwise the target will reject the image by sending back a portion the currently resident image’s metadata. The reject metadata contains the Image Version, Image version, and User ID fields. For more information about these fields, please refer to the metadata section.

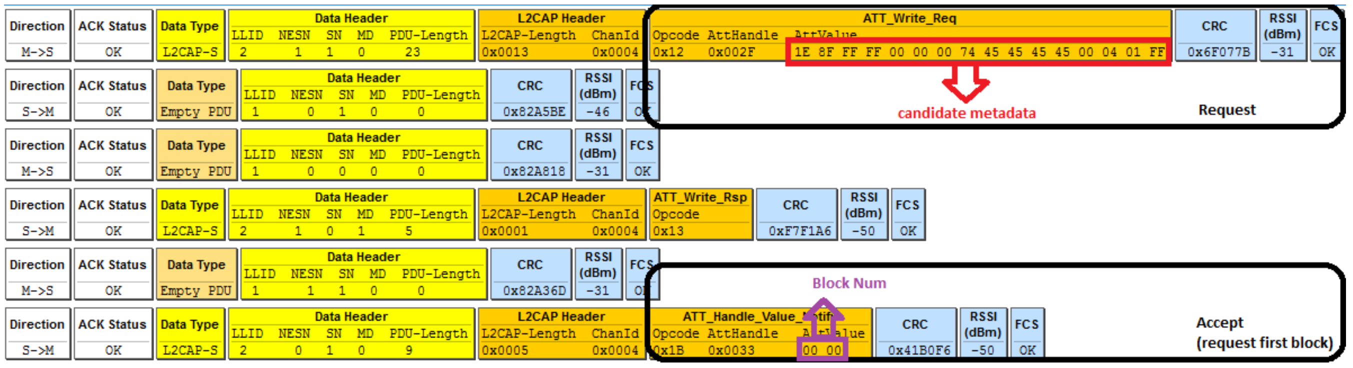
A sniffer capture of the image identify characteristic being used to reject a candidate OAD image is shown below. Note that only image version, length, and user ID are contained in the reject notification.



```
Direction  ACK Status  Data Type  Data Header                         L2CAP Header           ATT_Write_Req                                                                   CRC       RSSI   FCS
                                  LLID NESN SN MD PDU-Length          L2CAP-Length ChanId    Opcode AttHandle AttValue                                                                (dBm)
M->S       OK          L2CAP-S    2    0    0  0  23                  0x0013       0x0004    0x12   0x002F    3B EA FF FF 01 00 00 28 45 45 45 45 00 24 01 FF             0x8D7BE3  -30    OK

Direction  ACK Status  Data Type  Data Header                         CRC       RSSI   FCS
                                  LLID NESN SN MD PDU-Length                    (dBm)
S->M       OK          Empty PDU  1    1    0  0  0                   0x27C3F8  -46    OK

Direction  ACK Status  Data Type  Data Header                         CRC       RSSI   FCS
                                  LLID NESN SN MD PDU-Length                    (dBm)
M->S       OK          Empty PDU  1    1    1  0  0                   0x27CE5E  -31    OK

Direction  ACK Status  Data Type  Data Header                         L2CAP Header           ATT_Write_Rsp  CRC       RSSI   FCS
                                  LLID NESN SN MD PDU-Length          L2CAP-Length ChanId    Opcode                   (dBm)
S->M       OK          L2CAP-S    2    0    1  1  5                   0x0001       0x0004    0x13           0x8726F0  -54    OK

Direction  ACK Status  Data Type  Data Header                         CRC       RSSI   FCS
                                  LLID NESN SN MD PDU-Length                    (dBm)
M->S       OK          Empty PDU  1    0    0  0  0                   0x27C52B  -30    OK

Direction  ACK Status  Data Type  Data Header                         L2CAP Header           ATT_Handle_Value_Notify                 CRC       RSSI   FCS
                                  LLID NESN SN MD PDU-Length          L2CAP-Length ChanId    Opcode AttHandle AttValue                         (dBm)
S->M       OK          L2CAP-S    2    1    0  0  15                  0x000B       0x0004    0x1B   0x002F    01 00 20 2B 45 45 45 45   0xDF0271  -54    OK
```

Figure 7. Reject Notification in Sniffer Capture

Alternatively, a successful OAD initiation is shown in Figure 8.

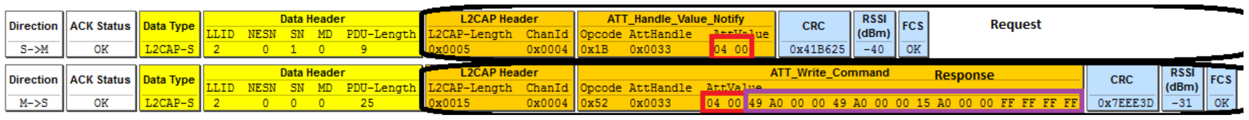


```
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | L2CAP Header: L2CAP-Length ChanId | ATT_Write_Req: Opcode AttHandle AttValue                                              | CRC       RSSI (dBm)  FCS
M->S       OK          L2CAP-S    | 2    1    1  0  23                      | 0x0013        0x0004              | 0x12   0x002F    1E 8F FF FF 00 00 00 74 45 45 45 45 00 04 01 FF                      | 0x6F077B  -31         OK
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | CRC       RSSI (dBm)  FCS
S->M       OK          Empty PDU  | 1    0    1  0  0                       | 0x82A5BE  -46         OK
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | CRC       RSSI (dBm)  FCS
M->S       OK          Empty PDU  | 1    0    0  0  0                       | 0x82A818  -31         OK
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | L2CAP Header: L2CAP-Length ChanId | ATT_Write_Rsp: Opcode | CRC       RSSI (dBm)  FCS
S->M       OK          L2CAP-S    | 2    1    0  1  5                       | 0x0001        0x0004              | 0x13                  | 0xF7F1A6  -50         OK
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | CRC       RSSI (dBm)  FCS
M->S       OK          Empty PDU  | 1    1    1  0  0                       | 0x82A36D  -31         OK
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | L2CAP Header: L2CAP-Length ChanId | ATT_Handle_Value_Notify: Opcode AttHandle AttValue | CRC       RSSI (dBm)  FCS
S->M       OK          L2CAP-S    | 2    0    1  0  9                       | 0x0005        0x0004              | 0x1B   0x0033    00 00                             | 0x41B0F6  -50         OK
```

Figure 8. Successful OAD Initiation Sniffer Capture

2.4.2 OAD Image Block Characteristic (0xFFC2)

The OAD Image Block characteristic is used to request and transfer a block of the OAD image. “01:00” shall be written to the CCCD of this characteristic so that notification for block request is enabled. The target requests the next block of the image by sending a GATT notification to the downloader with the requested block number. The downloader will respond (GATT write no response) with the block number and a 16 byte OAD image block. The image block contains the actual binary data from OAD image offset by the block number. Figure 11 shows a block request/response sniffer capture.



```
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | L2CAP Header: L2CAP-Length ChanId | ATT_Handle_Value_Notify: Opcode AttHandle AttValue | CRC       RSSI (dBm)  FCS
S->M       OK          L2CAP-S    | 2    0    1  0  9                       | 0x0005        0x0004              | 0x1B   0x0033    04 00                             | 0x41B625  -40         OK
Direction  ACK Status  Data Type  | Data Header: LLID NESN SN MD PDU-Length | L2CAP Header: L2CAP-Length ChanId | ATT_Write_Command: Opcode AttHandle AttValue                              | CRC       RSSI (dBm)  FCS
M->S       OK          L2CAP-S    | 2    0    0  0  25                      | 0x0015        0x0004              | 0x52   0x0033    04 00 49 A0 00 00 49 A0 00 00 15 A0 00 00 FF FF FF FF  | 0x7EEE3D  -31         OK
```

Figure 9. Block Request/Response Sniffer Capture

In Figure 9 above, the block number field is 2 bytes (little endian) and highlighted in red. The OAD image block is 16 bytes and highlighted in purple.

2.4.3 OAD Image Count Characteristic (0xFFC3)

The OAD Image Count characteristic is used to set the number of OAD images to be downloaded. This is used for only Off-chip OAD and the default value of the characteristic is 1. Note On-chip OAD only supports one image download per session. See On-chip OAD 4.2 for details.

2.4.4 OAD Image Status (0xFFC4)

The OAD image status characteristic is used to report various failures that may occur during the OAD process. The downloader may use this information to determine why an OAD failed, so that it may correct for the errors and try again. “01:00” shall be written to the CCCD of this characteristic so that notification for status update is enabled. There are four OAD status messages that are defined by default. The OAD status codes are listed in the table below:

OAD Status Code	Value	Description
OAD_SUCCESS	0	OAD succeeded
OAD_CRC_ERR	1	The downloaded image’s CRC doesn’t match the one

		expected from the metadata
OAD_FLASH_ERR	2	The external flash cannot be opened
OAD_BUFFER_OFL	3	The block number of the received packet doesn't match the one requested. An overflow has occurred.

Figure 10. OAD Status Codes

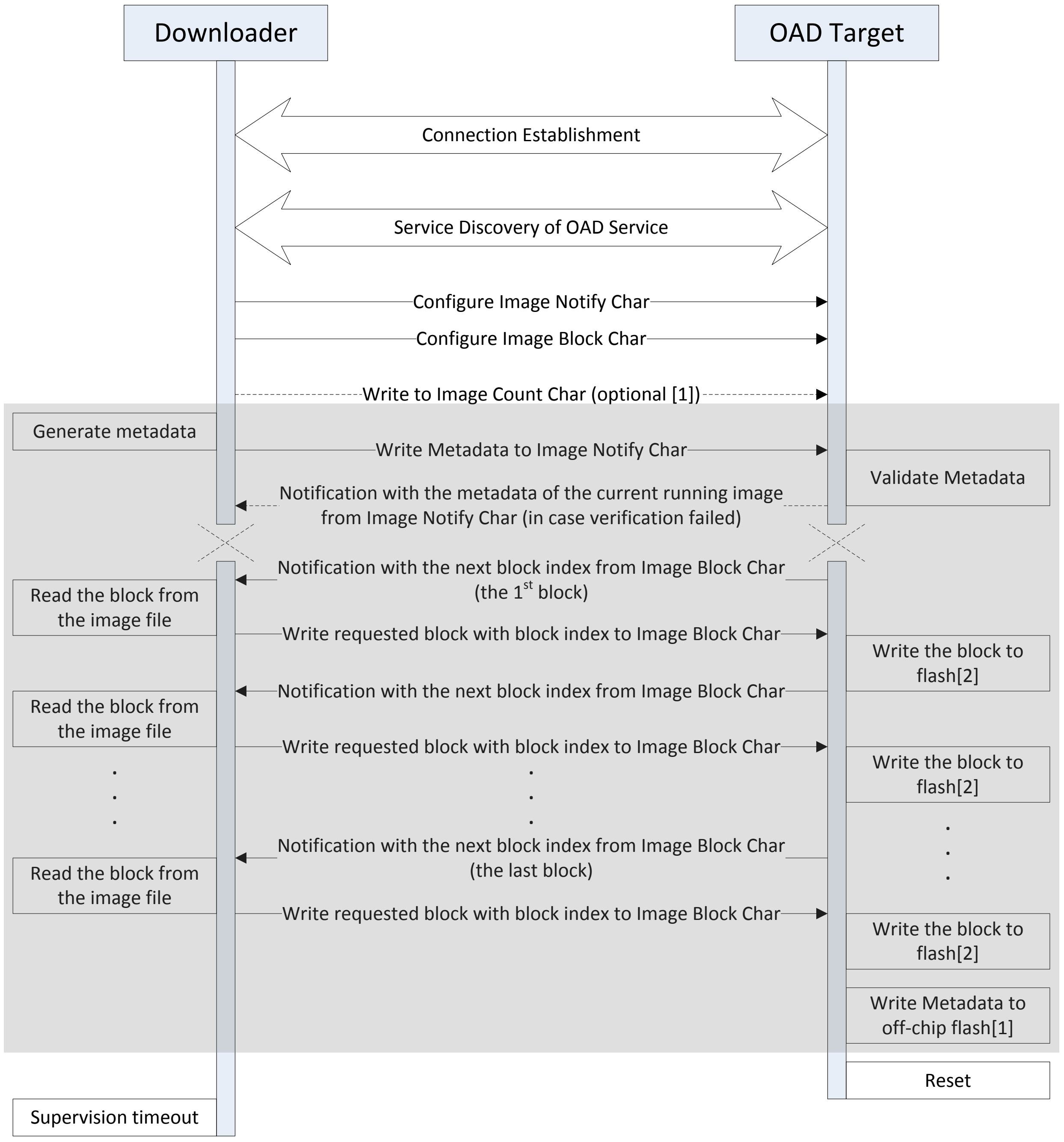
The customer may extend these values as needed, and use the OAD_sendStatus() function to send updates to the downloader.

2.5 OAD Process

This Profile has been designed to provide a simple and customizable OAD Profile for the customer. In its most rudimentary form, for both On-chip and Off-chip OAD, this profile is responsible for accepting an OAD interaction based on image header criteria, storing the image onto the flash and causing a device reset if the download is successful so that the downloaded application image is run by the BIM. Downloader and OAD Target perform Client role and Server role respectively.

2.5.1 Initiation of the OAD Process

After establishing a new connection, updating the connection interval for a faster OAD and enabling notifications of OAD Image Identify and OAD Image Block characteristics on the OAD Target, the Downloader shall write to the Image Identify characteristic of the OAD Target. The message data will be the header retrieved from the OAD Image available for OAD.


Downloader
OAD Target
Connection Establishment
Service Discovery of OAD Service
Configure Image Notify Char
Configure Image Block Char
Write to Image Count Char (optional [1])
Generate metadata
Write Metadata to Image Notify Char
Validate Metadata
Notification with the metadata of the current running image from Image Notify Char (in case verification failed)
Notification with the next block index from Image Block Char (the 1st block)
Read the block from the image file
Write requested block with block index to Image Block Char
Write the block to flash[2]
Notification with the next block index from Image Block Char
Read the block from the image file
Write requested block with block index to Image Block Char
Write the block to flash[2]
Notification with the next block index from Image Block Char (the last block)
Read the block from the image file
Write requested block with block index to Image Block Char
Write the block to flash[2]
Write Metadata to off-chip flash[1]
Reset
Supervision timeout

[1] Applies only to Off-chip OAD
[2] Writes to on-chip flash if using On-chip OAD and off-chip flash if using Off-chip OAD

Figure 11. OAD Sequence Diagram

Upon receiving the write request to the Image Identify characteristic, the OAD Target will compare the image available for OAD to its own running image. By default, only the image size and version number, which implies whether the image is of type A or B, are checked to determine if the new image is acceptable to download.

If the OAD Target determines that the image available for OAD is acceptable, the OAD Target will initiate the OAD process by notifying the Image Block Transfer characteristic to the Downloader requesting the first block of the new image. Otherwise, if the OAD Target finds that the new image does not meet its criteria to begin the OAD process, it shall respond by notifying the Image Identify characteristic with its own Image Header data as sign of rejection. In that case, the OAD procedure will end at the moment where dotted 'X's are placed as depicted in Figure 11.

2.5.2 Image Block Transfers

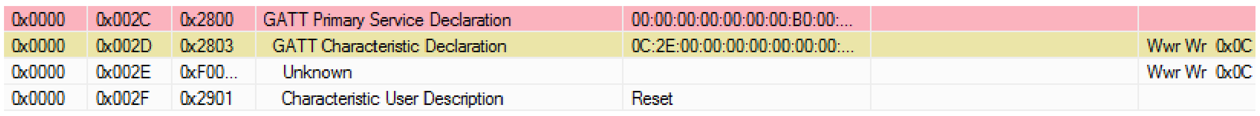
The Image Block Transfer characteristic allows the two devices to request and respond with the OAD image, one block at a time. The image block size is defined to be 16 bytes – see OAD_BLOCK_SIZE in oad.h. The OAD Target will request an image block from the Downloader by notifying the OAD Image Block characteristic with the correct block index. The Downloader shall then respond by writing to the OAD Image Block characteristic. The message's data will be the requested block's index followed by the corresponding 16-byte block of the image. Whenever the OAD Target is ready to digest another block of the OAD image, it will notify the Image Block Transfer characteristic with the index of the desired image block. The Downloader will then respond.

2.5.3 Completion of the OAD Process

After the OAD Target has received the final image block, it will verify that the image is correctly received and stored by calculating the CRC over the stored OAD image. The OAD Target will then invalidate its own image and reset so that the BIM can run the new image in-place. The burden is then on the Downloader, which will suffer a lost BLE connection to the OAD Target during this verification and instantiation process, to restart scanning and the to reestablish a connection and verify that the new image is indeed running.

2.6 OAD Reset Service

The OAD reset service is only used by on-chip OAD solutions. It implements a method for invalidating the currently running image and resetting the device. This must occur because in on-chip solutions the currently running image cannot update itself. More information about the on-chip OAD process will be covered in the on-chip OAD chapter. Figure 12 shows an overview of the OAD reset service and it's characteristic. Like the OAD service, the reset service uses the 128 bit TI base UUID with a 16 bit short UUID of 0xFFD0.



0x0000	0x002C	0x2800	GATT Primary Service Declaration	00:00:00:00:00:00:00:B0:00:...		
0x0000	0x002D	0x2803	GATT Characteristic Declaration	0C:2E:00:00:00:00:00:00:00:...		Wwr Wr 0x0C
0x0000	0x002E	0xF00...	Unknown			Wwr Wr 0x0C
0x0000	0x002F	0x2901	Characteristic User Description	Reset		

Figure 12. OAD Reset Service

2.6.1 OAD Reset (0xFFD1)

The OAD reset is accomplished by invalidating Image B, which forces the bootloader to revert to Image A until another successful OAD of Image B has occurred. Image B is invalidated by corrupting its CRC. After the corruption, the reset service immediately invokes a HAL reset to jump to the bootloader. Note that a GATT write of any value to the reset service will trigger a reset of the device/invalidation of Image B.

2.7 Bootloader

Since a running image cannot update itself, both On-chip and Off-chip OAD methods must employ a bootloader. A bootloader is a lightweight section of code that is designed to run every time the device resets, check the validity of newly downloaded images, and if necessary, load the new image into internal flash. TI's bootloader implementation is called the Boot Image Manager (BIM). BIM's implementation varies slightly for On-chip and Off-chip OAD solutions, thus there is a separate BIM project for each.

- /util/bim – This project implements an On-chip OAD bootloader
- /util/bim_extflash – This project implements an Off-chip OAD bootloader

Note: Some BIM projects contain project configurations for a baseloader. This a legacy implementation of BIM that is no longer used. Please see section 1.4 for more details.

BIM is always linked to page 0 and page 31 of internal flash, and will always link the CCFG section with it.

See the on-chip and off-chip OAD sections of the guide for the details on their respective BIM implementations.

3. Off-Chip OAD

Off-chip OAD maximizes the available user flash space by storing the incoming OAD image in external flash as it is received over the air since a running application cannot update itself. The external flash component adds additional value such as factory image support and having multiple copies of images in external flash. The off-chip OAD solution is made of the following three images

- BIM: Iterates over external flash metadata table and determines which image is ready to run, and if necessary copying that image from external to internal flash. Jumps to the active image
- BLE-Stack Image: Implements the BLE protocol stack functionality.
- User application: This is the application that implements the user's desired functionality. This image is upgradable via OAD.

Off-chip OAD also has the ability to update both the BLE protocol stack and application image when they are combined as a single merged image. This section aims to explain the concepts and tradeoffs involved in an off-chip OAD system.

3.1 Off-chip OAD Memory Map

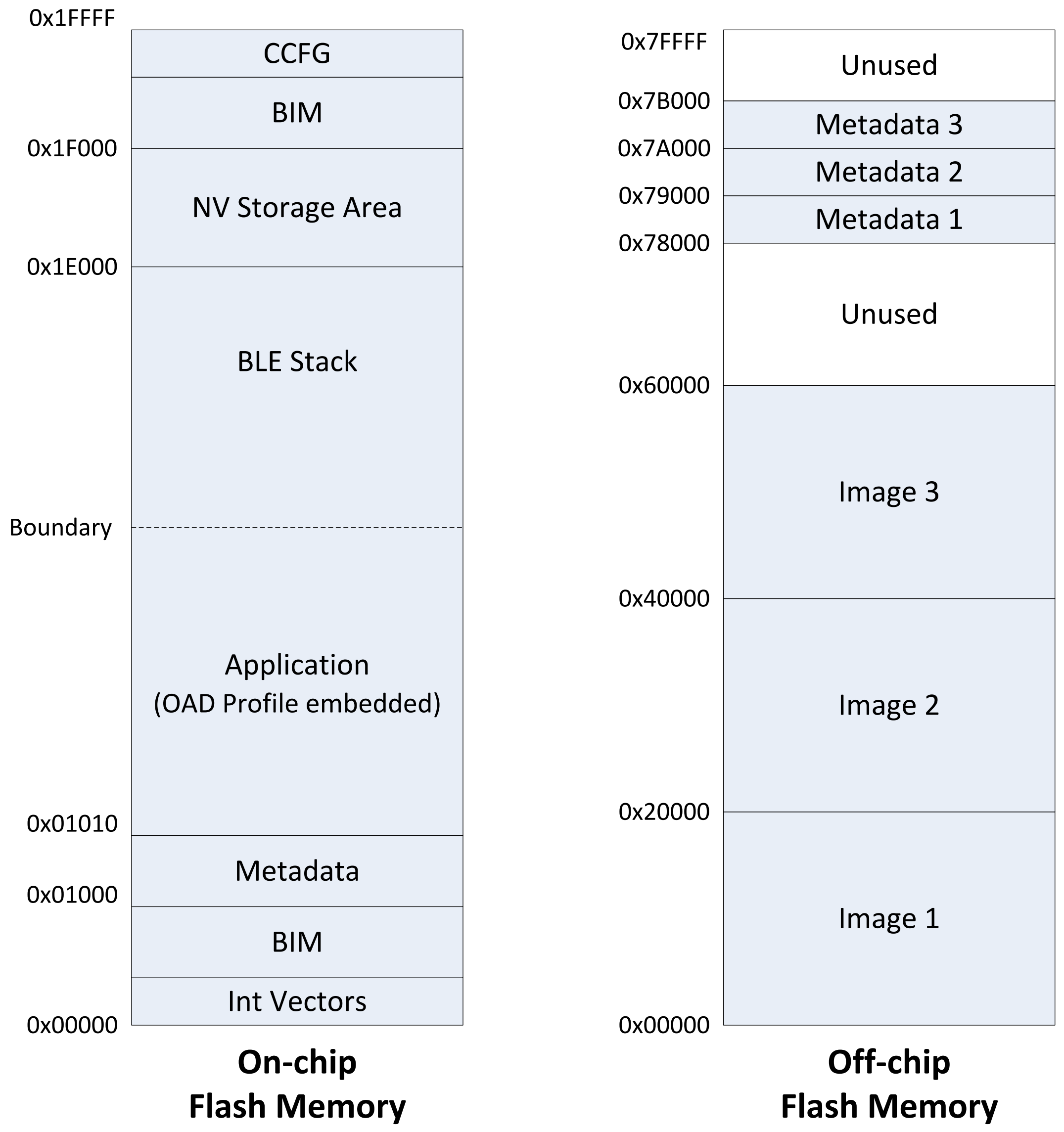


Figure 13. Off-chip OAD Target Memory Partition

The Off-chip OAD Target has both on-chip flash memory and off-chip flash memory device. The on-chip flash memory contains the Interrupt Vectors, the BLE Stack, the Application where OAD Profile is embedded, and the BLE stack image, the NV Storage Area, the BIM and the CCFG.

The off-chip flash memory contains up to 3 OAD Images and up to 3 metadata corresponding to the OAD Images. The size of each OAD Image placeholder is 128kB. The memory partition of the OAD Target for off-chip OAD is depicted in Figure 13. Each OAD image, if it's of either App only or App+Stack, must support OAD Profile so that further OAD is enabled after it is downloaded to the off-chip memory, copied to the on-chip memory and executed.

3.2 Constraints and Requirements for Off-chip OAD

Using the internal flash of CC26X0F128, the first page and the last page, or 8kB in total, of flash are reserved for the flash interrupt vectors and the BIM. The flash page 31 or the last page starting at address 0x1F000 where BIM is located is shared with the CCFG. Neither the first page nor the BIM is designed to be upgraded by Off-chip OAD.

An off-chip flash component of at least 120kB plus space for a 16 byte image metadata block is required for a full flash update. A SPI connection is used to communicate with the off-chip flash component.

The OAD image to be downloaded to the off-chip flash memory can be an application image, a stack image, a hex merge of application plus stack, an image intended for the upgrade of a network processor, or any type of image as far as it is supposed to eventually replace any part of the on-chip 120kB area between the first and the last pages. More than one image can be downloaded before the system reset followed by BIM's copying the downloaded images from the off-chip flash memory to the on-chip flash memory.

Since page 0 cannot be updated in Off-chip OAD, an application must include its own TI-RTOS instance in flash without dependency on the TI-RTOS ROM implementation (see section 8 for more info). Also, it must include OAD profile so that further OAD upgrades are available when it runs on the on-chip flash memory since Off-chip OAD doesn't require any OAD-dedicated application like Image A for On-chip OAD.

The first and last flash pages must never be attempted to update because a power cycle during an update of either page could render the device unresponsive until physically reprogrammed.

While On-chip OAD Target receives only one application OAD image, Off-chip OAD Target can receive up to 3 OAD images. The metadata is inserted into the beginning of the Off-chip OAD Image when transferred and is also stored separately in the off-chip flash.

3.3 Conditions for rejecting Metadata

Off-chip OAD Target checks that the new image's version is greater than the current image's version. However, as a bypass mechanism, any image of version 0 will be accepted. See OADTarget_validateNewImage() in oad_target_external_flash.c for more details.

3.4 BIM for Off-chip OAD

The OAD solution requires that permanently resident boot code, the BIM, exists in order to provide a fail-safe mechanism for determining whether to run the existing application image or to copy a new image or images from off-chip flash to on-chip flash. It is assumed that a valid image exists either in off-chip flash ready to be copied or already placed in on-chip flash at any given time. Given this assumption, the initial image placed in internal flash which does not exist in external flash will have invalid external image metadata, and so the bootloader will choose to jump to the existing image's entry point.

At startup, BIM checks if the application image metadata in off-chip flash has a status indicating that the image is to be copied to the on-chip flash. If the status is 0xFF, copies the image if a valid CRC and CRC Shadow are found. If the status is anything other than 0xFF, assumes the application in the on-chip flash is valid to run. If a 2 byte value is found that is neither 0x0000 nor 0xFFFF, but a 0xFFFF shadow checksum is found, the BIM computes the CRC over the image. Image length is determined by the metadata that is also stored contiguous with the CRC in on-chip flash that was copied over during the original write of the image from the off-chip flash.

If off-chip flash contains a “bad” image to be downloaded, but this image is undesirable, BIM can be programmed with symbol NO_COPY to skip image checking and jump directly into the image already placed in on-chip flash; at which point the on-chip flash image could invalidate the bad image’s metadata or OAD a new image in its place. BIM will not be able to load any new images while NO_COPY is defined in the build.

BIM is only responsible for making an application image failsafe upon entry. This could mean an app and stack image, or just the application. BIM has exactly one entrance to the application image.

The BIM occupies the last flash page with CCFG and uses the interrupt vectors at the start of flash where the Reset Interrupt Vector calls the BIM startup routine to ensure its control of the system upon a device reset.

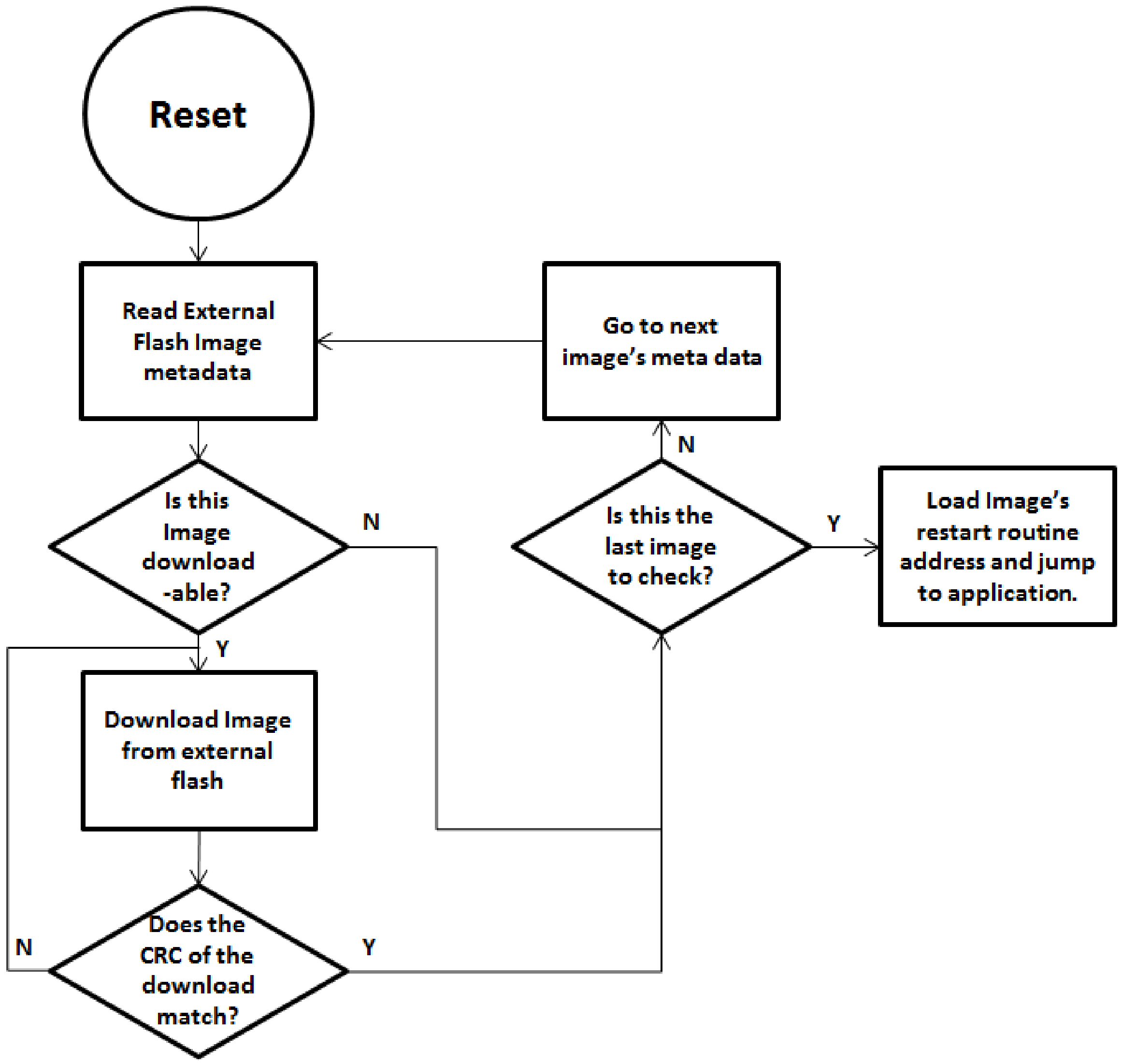

Reset
Read External Flash Image metadata
Go to next image’s meta data
Is this Image download -able?
N
Y
Is this the last image to check?
N
Y
Load Image’s restart routine address and jump to application.
Download Image from external flash
Does the CRC of the download match?
N
Y

Figure 14. Functional Overview of Off-chip BIM

4. On-Chip OAD

Note: BLE On-chip OAD is only supported on CC26x0 and CC13x0 R1 devices by the IAR toolchain. These projects also have very little available flash for application development, so it is important for customers to understand clearly the limitations before starting a design with on-chip OAD.

On-chip OAD solutions only require a single chip and upgrade the user's application by means of a permanently resident application named the oad_target_app. (Not to be confused with the physical OAD Target device). On-chip OAD solutions employ four separate images to accomplish the OAD process

- BIM: Responsible for determine which image to boot, see BIM for On-chip OAD. The BIM is not upgradable via OAD.
- BLE-Stack Image: This image implements the BLE protocol stack functionality. This image is not upgradable via OAD.
- OAD Target (image A): The permanently resident OAD application that is responsible for upgrading the user application. Since a running application cannot update itself, the OAD target application is responsible for implementing the OAD profile, and storing the incoming user application in flash. This image is not upgradable via OAD.
- User application (Image B): This is the application that implements the user's desired functionality. This image is upgradable via OAD.

This section aims to explain the concepts and tradeoffs involved in an on-chip OAD system.

4.1 On-chip OAD Memory Map

The flash memory of OAD Target for On-chip OAD contains the Interrupt Vectors, RCFG, the permanently resident OAD Target App which is also called Image A, the Image B which is initially empty and the place holder for the downloaded OAD Image, the BLE stack, the NV Storage Area, the BIM and the CCFG as shown in Figure 15.

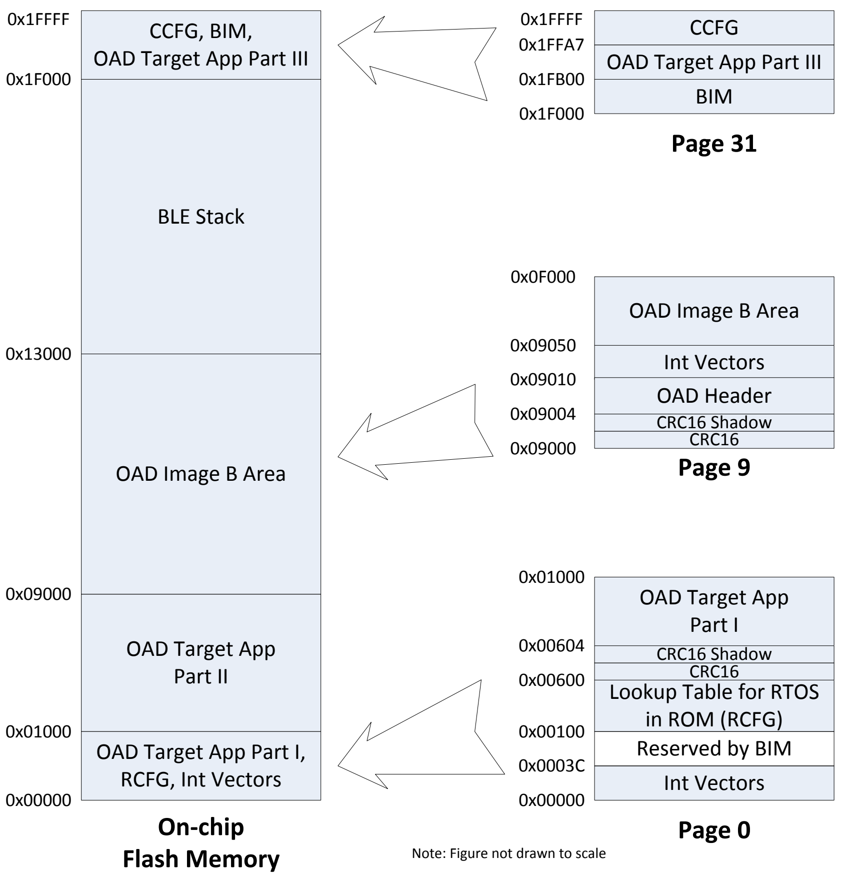


Figure 15. On-chip OAD Target Memory Partition

BIM's design offers the flexibility of having two valid images ready to run; the choice as to which image will run is decided in the BIM. Only the OAD Image B can be downloaded. The OAD Target application, Image A, is a permanent resident which relies on code in the first and last flash page – which if erased during a power down would break the device. The advantage of a permanently resident Image A whose sole purpose is to implement the BLE Stack and OAD Profile is that it increases the amount of available flash for Image B. The developer of a custom Image B does not have to include the OAD Profile implementation. The only reference to OAD feature that Image B needs is a valid image header described in Figure 3. The reference to the valid image header is necessary to use OAD Reset Service described in 2.6. Both Image A and Image B must be developed using exactly the same BLE Stack build, linked at the same location in memory.

4.2 Constraints and Requirements for On-chip OAD

Using the internal flash of CC26X0F128, the first 9 pages, or 36KB, of flash are, by default, reserved for the flash interrupt vectors, the BIM and the permanently resident OAD Target App using an instance of TI-RTOS partially implemented in ROM. BIM and the OAD Target App also use the remaining space on flash page 31, starting at address 0x1F000, shared with the CCFG. Neither BIM nor the OAD Target App is designed to be upgraded by On-chip OAD.

The OAD Image to be downloaded is, by default, allocated 10 flash pages, or 40KB, from address 0x9000 to 0x12FFF. Because page 0 cannot be updated, an application must include its own TI-RTOS instance in flash without dependence on the TI-RTOS ROM implementation. This image also shares the CCFG referenced in the above paragraph. It is not possible to update the CCFG parameters via an OAD.

The OAD Target App and the OAD image should share the same RAM range as only one is used per device reset. The OAD Image must be a complete application image, capable of running independently of the permanently resident OAD Target App.

The BLE protocol stack defaults to a range of 12 flash pages, or 48kB, ranging from address 0x13000 to 0x1EFFF and no SNV pages are used by default. If the OAD Image is too large to fit in its allocated space, consider removing some features of the BLE stack to reduce its size. The OAD Target App, or the Image A, and the Image B shall share the same BLE stack. It is not possible to perform an On-chip OAD of the BLE Stack image.

The first and last flash pages must never be erased as doing so puts the device in an unsafe state and a reset at this time will “brick” the chip and prevent it from restarting without the help of a JTAG or serial boot loader.

4.3 Conditions for Rejecting Metadata

The LSB of the new image’s version must not be equal to the LSB of the current image’s version. This is to prevent redundant OAD sessions. The LSB is checked using the OAD_IMG_ID() macro. See OADTarget_validateNewImage() in oad_target_internal_flash.c for more details.

4.4 BIM for On-chip OAD

The OAD solution requires that permanently resident boot code, the BIM, exists in order to provide a fail-safe mechanism for determining (in preferential order) the image which is ready to run. When a valid image is found, the BIM jumps to that image at which point the image takes over execution. Either Image A or Image B must implement the proprietary TI OAD Profile. By default, this is Image A’s role. When an image with the OAD Profile downloads a new image, a system reset can be executed to return to BIM to verify the correctness of the download and begin execution.

The BIM co-occupies the last flash page with CCFG and additional OAD Target application code. The OAD Target application code is linked into a specific section of the last flash page as defined in the linker file. BIM uses the interrupt vectors at the start of flash where the Reset Interrupt Vector calls the BIM startup routine to ensure its control of the system upon a device reset.

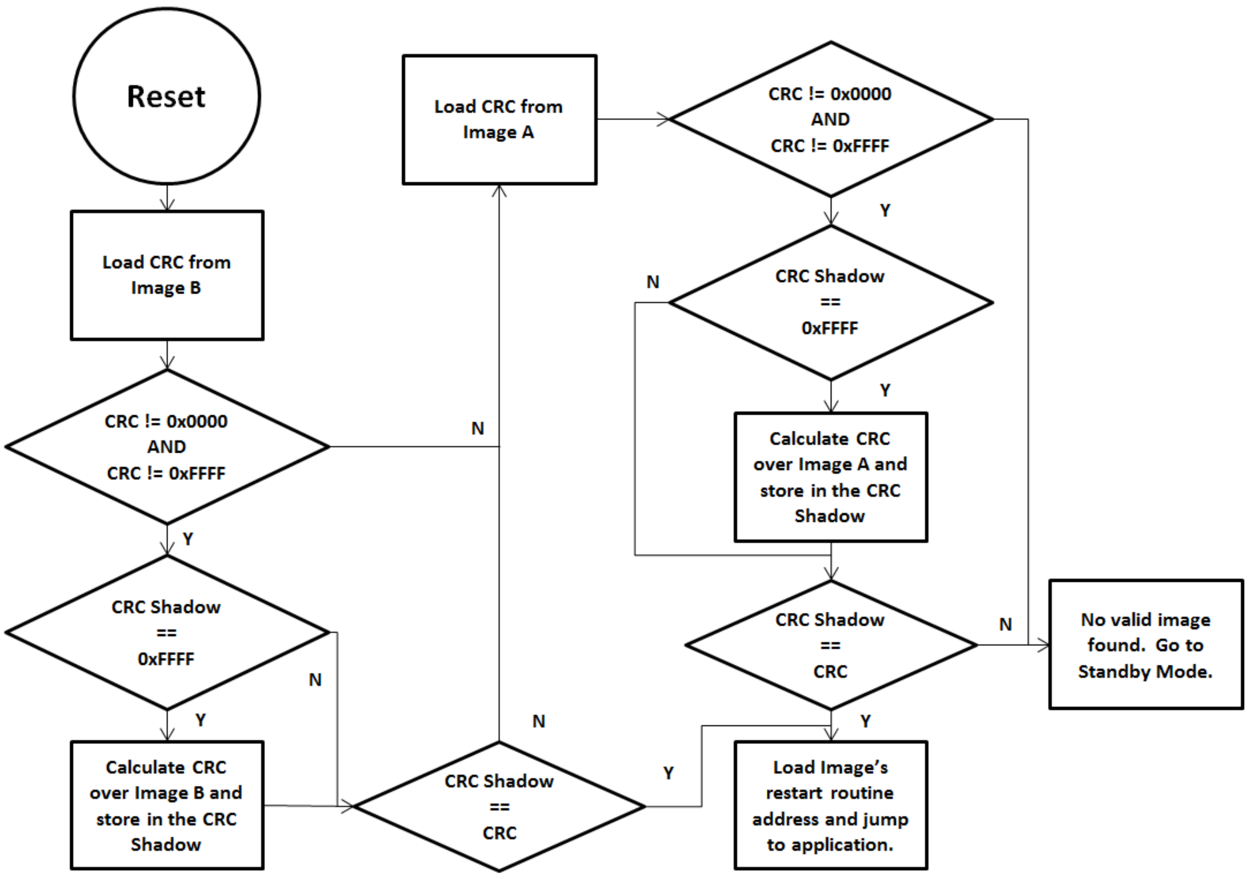

Reset
Load CRC from Image B
CRC != 0x0000 AND CRC != 0xFFFF
N
Y
CRC Shadow == 0xFFFF
N
Y
Calculate CRC over Image B and store in the CRC Shadow
CRC Shadow == CRC
N
Y
Load CRC from Image A
CRC != 0x0000 AND CRC != 0xFFFF
Y
N
CRC Shadow == 0xFFFF
Y
Calculate CRC over Image A and store in the CRC Shadow
CRC Shadow == CRC
N
Y
No valid image found. Go to Standby Mode.
Load Image's restart routine address and jump to application.

Figure 16. Functional Overview of On-chip BIM

As the permanent owner of the flash interrupt vectors, BIM provides a fail-safe mechanism for intercepting the reset vector, putting the hardware into a safe state, and taking the most appropriate action by reading the headers of Image A and Image B.

By default, BIM gives precedence to Image B, as Image A is only expected to be run when a newer instance of Image B is ready to OAD or no valid Image B exists. If the preferred image is not ready to run, then the other image is checked. If neither image is ready to run – an unlikely scenario because Image A, the OAD Target App, need not ever be erased – then BIM puts the device into a low power Standby mode. Also by default, a CRC check is not performed on Image A because it is expected that the OAD Target App will be used as a fixed image. The check on Image A will only read the checksum placed by IAR to see if an image exists; it will not calculate the CRC shadow.

In order to verify that an image is valid, a fixed 4-byte area known as the CRC and CRC-shadow will be queried. If the 2-byte CRC16 output calculated at build time matches the 2-byte CRC16 shadow calculated by BIM, then the image is commissioned to run immediately. If the CRC is not zero and not the erased-flash value of 0xFFFF and the CRC-shadow is the erased-flash value of 0xFFFF, then the CRC can be calculated over the entire image (not including this 4-byte area) and the result can be compared to the valid CRC to determine whether the image should be commissioned as ready to run.

5. Running the Out of the Box Demos

TI provides a suite of software that is useful for evaluating the TI OAD solution. This section will detail how to setup and run the out of the box demos. In order to evaluate the TI OAD solution, the OAD Downloader and Target must be setup and configured.

5.1 Required Hardware

The demo for both on-chip and off-chip OAD requires the same hardware; the following steps assume that you have this hardware available. Other hardware configurations may be possible such as running on other LaunchPads, but are out of the scope of this document.

- 2x CC2650 LaunchPad
- Windows PC

5.2 Required Software

In order to properly setup and configure the hardware, the following software is required

- TI BLE-Stack 2.2.x
- Smart RF Flash Programmer 2
- BTool v1.41.17 from the CC2640R2 SDK Download Link
    - **Note: The other software in the CC2640R2 SDK is not compatible with the CC26x0/ CC13x0 device, only use this SDK for BTool v 1.41.17**

5.3 OAD Downloader

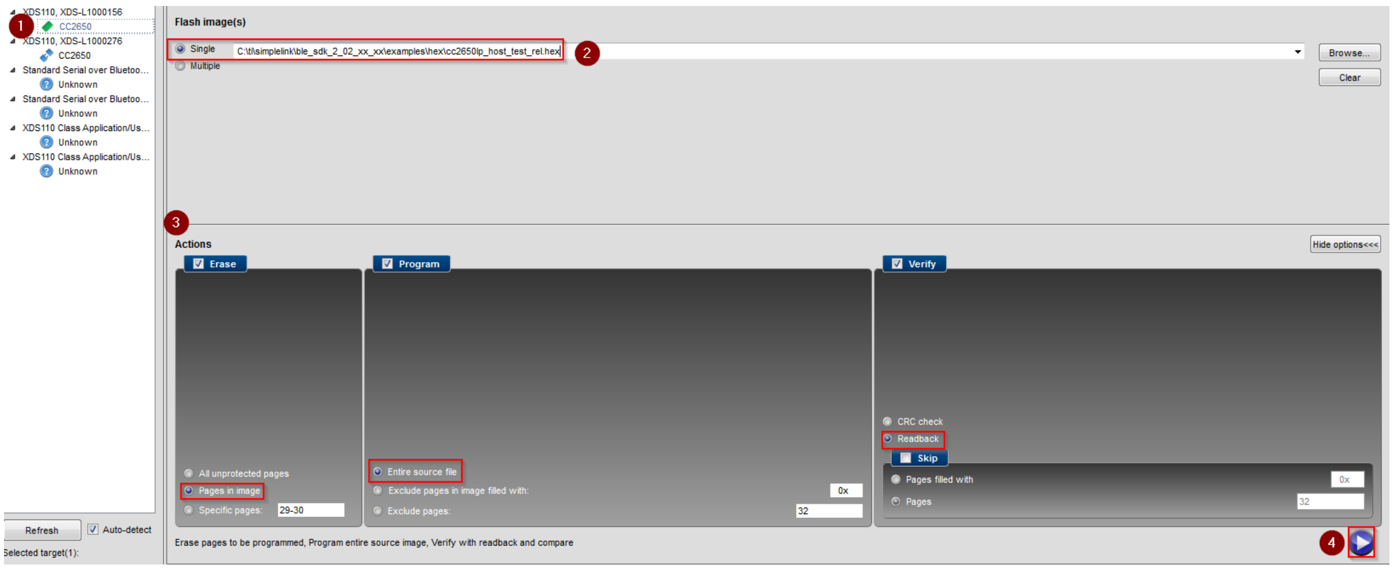
This section will cover setting up and using the OAD Downloader. The supported OAD Downloader is BTool v1.41.17. TI recommends using BTool as the downloader for evaluating both on and off-chip OAD. BTool is a PC application created to interface with a CC26xx device in a network processor configuration. The required platform is a CC2650 LaunchPad. Other devices may be used, but are out of the scope of this document.

5.3.1 Setting Up the CC2650 LaunchPad

In order to work with BTool the LaunchPad, the host_test application must be flashed on to it. Host Test is the network processor configuration supporting the HCI interface. BTool uses this interface to implement the BLE central role and various other configurations that are out of the scope of this document. The following steps will detail how to setup the CC2650 LaunchPad for use with BTool.

1. Open Smart RF Flash Programmer 2
2. Connect a CC2650 LaunchPad to the computer via USB
3. The LaunchPad should appear in the left pane of Smart RF Flash Programmer 2
4. Right click on the LaunchPad and select connect
5. Under the Flash Image(s) pane select single and provide the path to the prebuilt host test hex file (i.e. /examples/hex/CC26x0lp_host_test_rel.hex)
6. Ensure that Erase, Program, and Verify boxes are checked with the following options
    a. Erase: Pages in image
    b. Program: Entire source file
    c. Verify: Readback

7. Hit the play button to program the image



Flash image(s): [✓ Single] C:\ti\simplelink\ble_sdk_2_02_xx_xx\examples\hex\cc2650lp_host_test_rel.hex / Multiple

Erase: All unprotected pages / [✓ Pages in image] / Specific pages: 29-30

Program: [✓ Entire source file] / Exclude pages in image filled with: 0x / Exclude pages: 32

Verify: CRC check / [✓ Readback] — Skip: Pages filled with 0x / Pages 32

Erase pages to be programmed, Program entire source image, Verify with readback and compare

Figure 17. OAD Downloader HostTest Setup

At this point the CC2650 LaunchPad is ready to accept commands from BTool.

5.3.2 Setting up and connecting to BTool

The following steps will cover how to get BTool up and running on your PC. BTool is a very feature rich program, and its many features are out of the scope of this document. The following section assumes that you have familiarized yourself with the BTool User's Guide, particularly the sections on Starting the Application and Creating a Connection. The following steps will detail how to get BTool setup to perform an OAD.

Note: Many times in the BTool User's Guide a CC26r0R2F LaunchPad is mentioned, in this case it is safe to assume that these steps also apply to the CC2650 LaunchPad for the OAD use case as discussed in this document.

1. Download BTool from CC2640R2 SDK Download Link
2. Open BTool (located in the tools/blestack/btool folder)
3. Attach BTool to the use/UART backchannel as per the "Starting the Application Section"
    a. After attaching to the CC2650 LaunchPad running host_test, you should see a few messages printed to the Message Log, indicating that the host_test device was initialized properly.

At this point the OAD Downloader is properly configured and ready to connect to BLE Peers

5.4 OAD Target

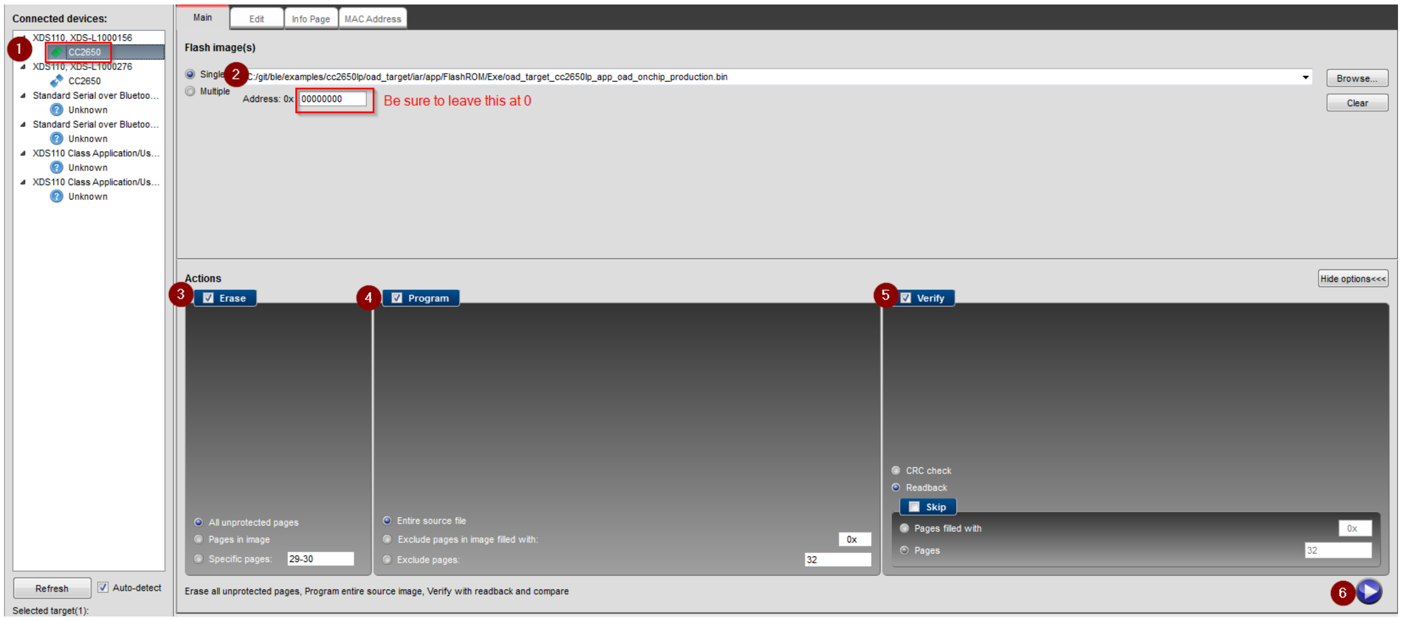
This section will cover how to get an OAD target image setup for both on-chip and off-chip OAD images. By the end of the section, you will have an OAD enabled system on the OAD Target that is ready to accept an incoming OAD image. Since on-chip and off-chip OAD vary slightly in their setup, they will be broken up into their own respective sections. As with the OAD Downloader, the supported OAD Target device is the CC2650 LaunchPad. OAD may be possible on other platforms, but this guide will only cover the CC2650 LaunchPad.

5.4.1 On-Chip OAD

Note: Please refer to the warnings/limitations in the On-Chip OAD section before starting an on-chip OAD design.

As discussed in the On-Chip OAD section, there are four separate applications that make up an on-chip OAD image. Three of these applications (BIM, Image A, and BLE-Stack) make up the OAD Target project. Because BIM and Image A share portions of the same flash pages, the merged image must be used to setup the OAD Target device, **if the images are flashed straight out of the debugger, the device will not boot.** The following steps will detail how to get an OAD target project up and running.

1. Navigate to the oad_target project and import in IAR (examples\CC26x0lp\oad_target)
    a. This workspace will contain three sub projects, BLE-Stack, OAD Target App, and the on-chip BIM
2. Build the projects in the following order (order matters for setting BLE-Stack boundary)
    a. BIM
        i. Be sure to use the correct configuration, for CC2650 LaunchPad use the **FlashOnly_LP configuration**
    b. BLE-Stack
    c. OAD Target app
3. After building the target app, the oad_image_tool will run and produce a merged binary image of the BIM, BLE-Stack, and Image A.
    a. The output binary will contain the proper metadata for Image A.
    b. The default output location for the tool is in the same location as the .out file produced by the toolchain. A shortcut for navigating to this file is to view the .out file in the IDE and right click and view in system explorer.
    c. The output file is of .bin type and contains "_production"
4. Flash the merged binary image onto the device using Smart RF Flash Programmer 2 using the following steps
    a. Right click on the OAD Target device in the left pane and select connect
    b. Select the single image configuration and point the path to the _production image mentioned in step C above
    c. Use the following settings
        i. Erase: All unprotected pages (this is important for clearing out the Image B region)
            1. Be sure to remove the "all unprotected pages" option after download as this may inadvertently erase flash in future download
        ii. Program: Entire source file
        iii. Verify: Readback



Flash image(s)

(●) Single: C:/git/ble/examples/cc2650lp/oad_target/iar/app/FlashROM/Exe/oad_target_cc2650lp_app_oad_onchip_production.bin

( ) Multiple

Address: 0x 00000000 Be sure to leave this at 0

Actions

☑ Erase: (●) All unprotected pages; ( ) Pages in image; ( ) Specific pages: 29-30

☑ Program: (●) Entire source file; ( ) Exclude pages in image filled with: 0x; ( ) Exclude pages: 32

☑ Verify: ( ) CRC check; (●) Readback; ☐ Skip: ( ) Pages filled with 0x; ( ) Pages 32

Erase all unprotected pages, Program entire source image, Verify with readback and compare

Figure 18. OAD On-chip Target Setup

5. After this step, the OAD Target App (Image A) should be advertising with the address 0A:D0:AD:0A:D0:AD. Verify this via BTool (refer to the BTool User's Guide for more info).
6. Connect to the OAD Target Device via BTool
7. Switch the active project to simple_peripheral, change build configuration to **FlashOnly_OAD_ImgB.**
    a. Note that for on-chip OAD the stack project contained in the simple_peripheral workspace is not used, instead the ImgB configuration will link to the stack boundary in the OAD Target workspace
8. Build the Image B configuration of simple_peripheral. The oad_image_tool will run and produce an OAD ready binary. The default location of the binary is in the same folder as the .out file produced by the IDE.

At this time you are ready to perform an on-chip OAD, please refer to the Performing an OAD with BTool section for the steps on how to load the image into BTool to perform an OAD.

5.4.2 Off-Chip OAD

As discussed in the Off-Chip OAD section, there are four separate applications that make up an off-chip OAD image. The following steps will detail how to get an off-chip OAD project up and running.

1. Open the Off-chip BIM project (examples/util/bim_extflash)
2. Enable the correct BIM build configuration; this is **FlashOnly_LP** for the CC2650 LaunchPad.
3. Build and flash the off-chip BIM on the device
4. Open the simple_peripheral project
5. Enable the correct off-chip OAD build configuration: **FlashOnly_OAD_ExtFlash**
6. Rebuild and flash the BLE-stack project
7. Rebuild and flash the Application project

At this time you are ready to perform an on-chip OAD, please refer to the Performing an OAD with BTool section for the steps on how to load the image into BTool to perform an OAD.

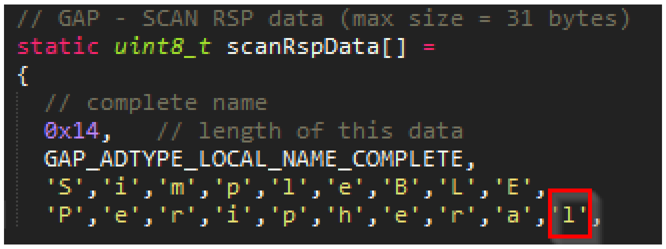
5.5 Changing the Device Name to Validate OAD

It is helpful to be able to differentiate between the image that is permanently resident on the device and the candidate image that is incoming via OAD. This section will detail one easy way to change the device's name and advertising data to validate that the image on the OAD Target device has in fact been updated. In this section we will detail how to change this data before performing an OAD.

The advertisement and scan response data is broadcast by the device while in the advertising state before it has established a connection. (more information about advertising and scanning in the Texas Instruments CC2640 Bluetooth® low energy Software Developer's Guide http://www.ti.com/lit/pdf/swru393). The following steps will detail how to change the scan response data

The ATT Device Name is a field that can be read when the connected state, it is most likely the easiest to see from BTool We will change this field as well.

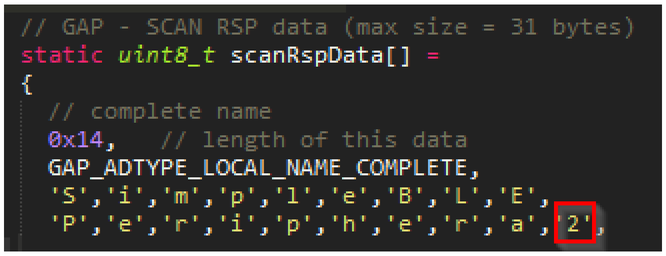
1. Navigate to simple_peripheral.c , find the scanRspData[] structure
2. Append a number to the scan response data, for example change the data from this



```
// GAP - SCAN RSP data (max size = 31 bytes)
static uint8_t scanRspData[] =
{
  // complete name
  0x14,   // length of this data
  GAP_ADTYPE_LOCAL_NAME_COMPLETE,
  'S','i','m','p','l','e','B','L','E',
  'P','e','r','i','p','h','e','r','a','1',
```

Figure 19. OAD Advertisement Data before change

To this

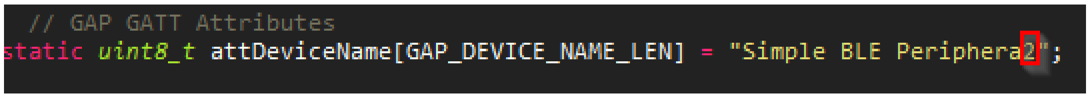


```
// GAP - SCAN RSP data (max size = 31 bytes)
static uint8_t scanRspData[] =
{
  // complete name
  0x14,   // length of this data
  GAP_ADTYPE_LOCAL_NAME_COMPLETE,
  'S','i','m','p','l','e','B','L','E',
  'P','e','r','i','p','h','e','r','a','2',
```

Figure 20. OAD Advertisement Data after Change

Note: Be careful not change this length of this field without changing the length token above

3. Find the attDeviceName array in simple_peripheral.c
4. Change the device name in a similar fashion to below; again be careful to not overflow the GAP_DEVICE_NAME_LEN field.



```
 // GAP GATT Attributes
static uint8_t attDeviceName[GAP_DEVICE_NAME_LEN] = "Simple BLE Periphera2";
```

Figure 21. OAD Device Name field modified

5. Rebuild the project; this will produce a “v2” of the image to be sent over the air.

This image can be labeled as v2 and should be used to send over the air. Sending an image over the air via BTool is detailed in Performing an OAD with BTool.

5.6 Performing an OAD with BTool

This section will detail the steps required to use BTool to perform an OAD. It assumes that you have already followed the steps in this section for setting up the OAD Target and OAD Downloader devices, and that you have created a “v2” image as detailed in Changing the Device Name to Validate OAD. This section also assumes a basic working knowledge of BTool as detailed in the BTool User’s Guide. This guide will detail the OAD specifics of the process.

5.6.1 (On-chip OAD only) Reset the device via the OAD Reset Service

If the on-chip OAD Target device is currently running the user application (Image B), it needs to be reset in order to jump to the OAD target image (Image A). This is done via the OAD reset service. The following section details how to switch an on-chip OAD device to the target image (Image A).

1. Connect to the device via BTool
2. Right click on the device and select Read Values, see screenshot below

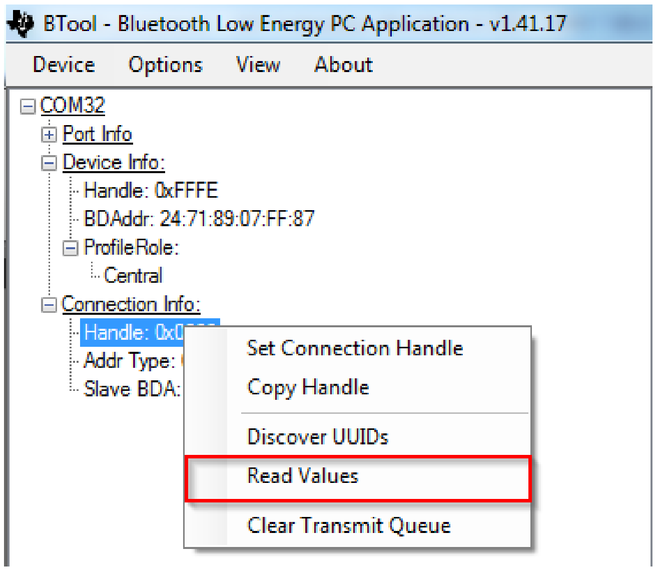
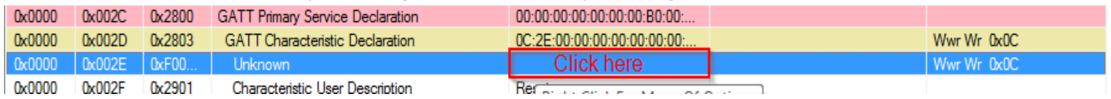


Figure 22. Read all characteristic data in BTool

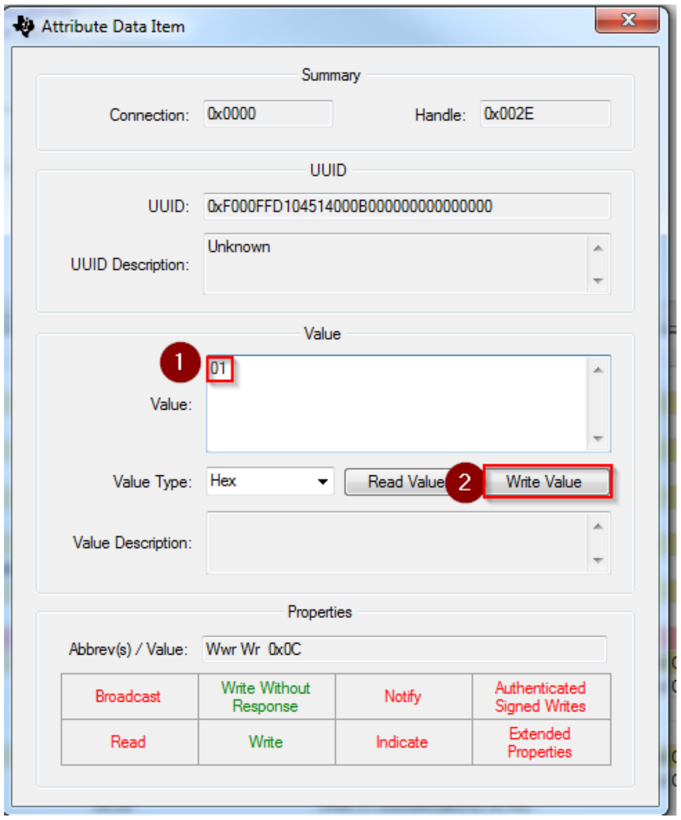
3. Write to the reset service by clicking on box corresponding to the characteristic value



0x0000	0x002C	0x2800	GATT Primary Service Declaration	00:00:00:00:00:00:00:B0:00:...		
0x0000	0x002D	0x2803	GATT Characteristic Declaration	0C:2E:00:00:00:00:00:00:00:...		Wwr Wr 0x0C
0x0000	0x002E	0xF00...	Unknown	Click here		Wwr Wr 0x0C
0x0000	0x002F	0x2901	Characteristic User Description	Re		

Figure 23. Writing to OAD Reset pt 1.

4. A box should pop up, write any non-zero value to the reset service


Attribute Data Item

Summary

Connection: 0x0000 Handle: 0x002E

UUID

UUID: 0xF000FFD104514000B000000000000000

UUID Description: Unknown

Value

Value: 01

Value Type: Hex

Value Description:

Properties

Abbrev(s) / Value: Wwr Wr 0x0C

Broadcast	Write Without Response	Notify	Authenticated Signed Writes
Read	Write	Indicate	Extended Properties

Figure 24. Writing to OAD Reset pt 2

At this time the device should reset and it should boot into the OAD Target App (Image A). This means the connection will drop. Any errors regarding timeout can be ignored. From this point, you can follow the steps in the next section to perform an OAD of the user application image.

Note: If the device not appear to advertise after writing to the reset service (and a valid OAD target image is present) then the device maybe in the Halt In Boot state, see the appendix for more information.

5.6.2 Performing an OAD

This section will describe how to perform an OAD of an application image and verify that the download succeeded, these steps apply to on and off-chip OAD.

1. Connect to the OAD Target device
   a. Note that for on-chip OAD the device address will be 0A:D0:AD:0A:D0:AD
   b. For off-chip the device address will default BLE address used by the chip
2. Change to the OAD pane on the far right panel
3. Load the application image created in the previous section into the Image File box
4. Press the send button, see below

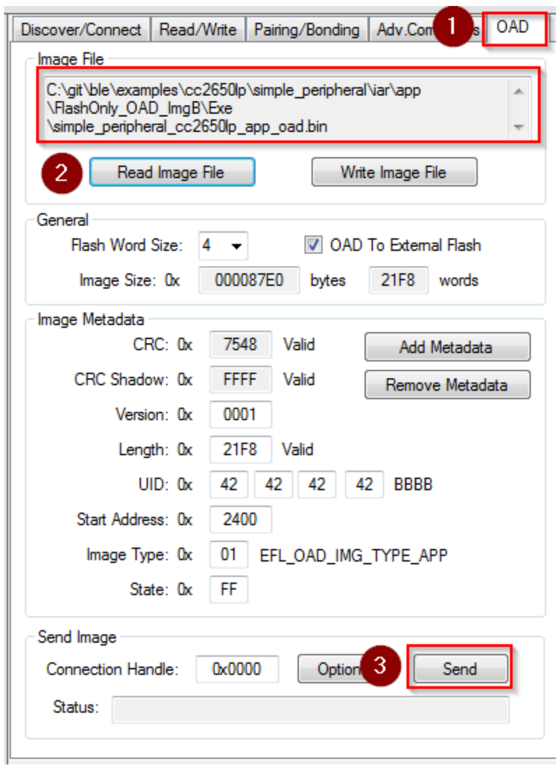


Image File

C:\git\ble\examples\cc2650lp\simple_peripheral\iar\app
\FlashOnly_OAD_ImgB\Exe
\simple_peripheral_cc2650lp_app_oad.bin

General

Flash Word Size: 4 ☑ OAD To External Flash

Image Size: 0x 000087E0 bytes 21F8 words

Image Metadata

CRC: 0x 7548 Valid

CRC Shadow: 0x FFFF Valid

Version: 0x 0001

Length: 0x 21F8 Valid

UID: 0x 42 42 42 42 BBBB

Start Address: 0x 2400

Image Type: 0x 01 EFL_OAD_IMG_TYPE_APP

State: 0x FF

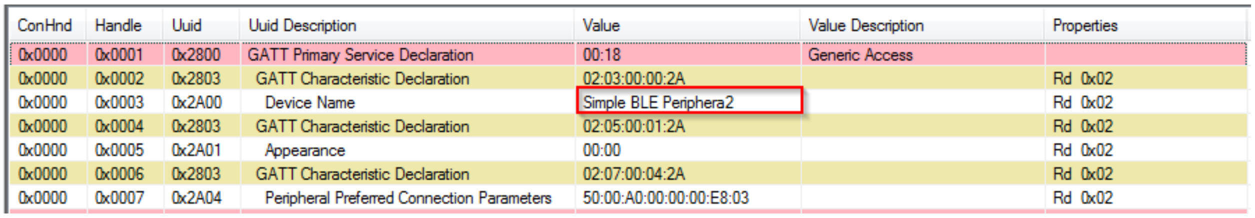
Send Image

Connection Handle: 0x0000

Status:

Figure 25. Sending an OAD Image over the air

5. As the OAD process progresses, the status will be reported by BTool in the status field.
6. At the end of the process OAD_SUCCESS will be reported, and the device will disconnect
    a. Note: If the device does not appear to advertise after OAD then the device may be in Halt in Boot state, see the appendix for more information.
7. Re-connect to the device
    a. Note using on-chip OAD, note the address will have changed from 0A:D0:AD:0A:D0:AD to the default.
8. Read the values and validate that the device name has changed (should contain new data).



ConHnd	Handle	Uuid	Uuid Description	Value	Value Description	Properties
0x0000	0x0001	0x2800	GATT Primary Service Declaration	00:18	Generic Access	
0x0000	0x0002	0x2803	GATT Characteristic Declaration	02:03:00:00:2A		Rd 0x02
0x0000	0x0003	0x2A00	Device Name	Simple BLE Periphera2		Rd 0x02
0x0000	0x0004	0x2803	GATT Characteristic Declaration	02:05:00:01:2A		Rd 0x02
0x0000	0x0005	0x2A01	Appearance	00:00		Rd 0x02
0x0000	0x0006	0x2803	GATT Characteristic Declaration	02:07:00:04:2A		Rd 0x02
0x0000	0x0007	0x2A04	Peripheral Preferred Connection Parameters	50:00:A0:00:00:00:E8:03		Rd 0x02

Figure 26. Verifying an OAD completed

At this time the OAD is considered successful and the new version of the image is running on the device.

6. Adding OAD to an Existing Application

This section details the steps to add OAD to an existing application, for evaluation purposes the default sample applications should be used as detailed in the previous chapter.

6.1 Off-chip OAD

The simple_peripheral project contains a configuration of 'FlashOnly_OAD_ExtFlash' designed for the application to run on CC2650 LaunchPad hardware platform and utilize the external flash component. The Stack project should be used as is.

The simple_peripheral with FlashOnly_OAD_ExtFlash configuration is made through the following procedures. The procedures can be applied to convert any existing application to the downloadable off-chip OAD Image. In addition to the following steps, registration and callbacks for the OAD service should be added to the application. Changes to be made for those are found under FEATURE_OAD in simple_peripheral.c.

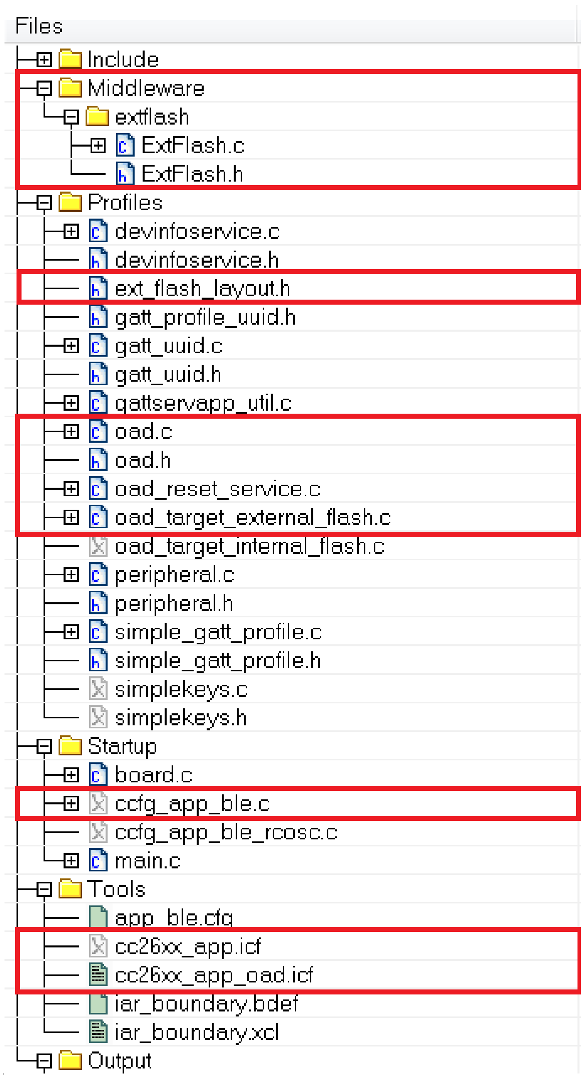
6.1.1 Project Changes For IAR

Using IAR, the Application Image can be built through the following procedure.

I. Select *Project→Options→C/C++ Compiler→Preprocessor* and add the following new definitions to *Defined symbols*:
```
FEATURE_OAD
HAL_IMAGE_E
```
Add the following lines to *Additional include directories*:
```
$SRC_EX$/profiles/oad/cc26xx
$TI_RTOS_DRIVERS_BASE$/ti/mw/extflash
```

II. Select *Project→Options→Linker→Config→Linker configuration file* and paste the following line:
```
$SRC_EX$/common/cc26xx/iar/cc26xx_app_oad.icf
```
And add the following symbols to *Configurable file symbol definitions*:
```
APP_IMAGE_START=0x1000
```
Append the following in Build Actions under *Pre-build command line*:
```
--cfgArgs NO_ROM=1,OAD_IMG_E=1
```

III. Under Tools, include cc26xx_app_oad.icf and exclude cc26xx_app.icf. Also exclude ccfg_app_ble.c. Add the OAD profile modules to the PROFILES folder of the workspace. These files are located here: `$BLE_INSTALL$/src/profiles/oad/cc26xx`.



```
Files
├─⊞ Include
├─⊟ Middleware
│  └─⊟ extflash
│     ├─⊞ ExtFlash.c
│     └── ExtFlash.h
├─⊟ Profiles
│  ├─⊞ devinfoservice.c
│  ├── devinfoservice.h
│  ├── ext_flash_layout.h
│  ├── gatt_profile_uuid.h
│  ├─⊞ gatt_uuid.c
│  ├── gatt_uuid.h
│  ├─⊞ gattservapp_util.c
│  ├─⊞ oad.c
│  ├── oad.h
│  ├─⊞ oad_reset_service.c
│  ├─⊞ oad_target_external_flash.c
│  ├── oad_target_internal_flash.c
│  ├─⊞ peripheral.c
│  ├── peripheral.h
│  ├─⊞ simple_gatt_profile.c
│  ├── simple_gatt_profile.h
│  ├── simplekeys.c
│  └── simplekeys.h
├─⊟ Startup
│  ├─⊞ board.c
│  ├─⊞ ccfg_app_ble.c
│  ├── ccfg_app_ble_rcosc.c
│  └─⊞ main.c
├─⊟ Tools
│  ├── app_ble.cfg
│  ├── cc26xx_app.icf
│  ├── cc26xx_app_oad.icf
│  ├── iar_boundary.bdef
│  └── iar_boundary.xcl
└─⊟ Output
```

Figure 27. OAD Off-Chip Files in IAR

6.1.2 Project Changes for CCS

Using CCS, the Application Image can be built through the following procedure.

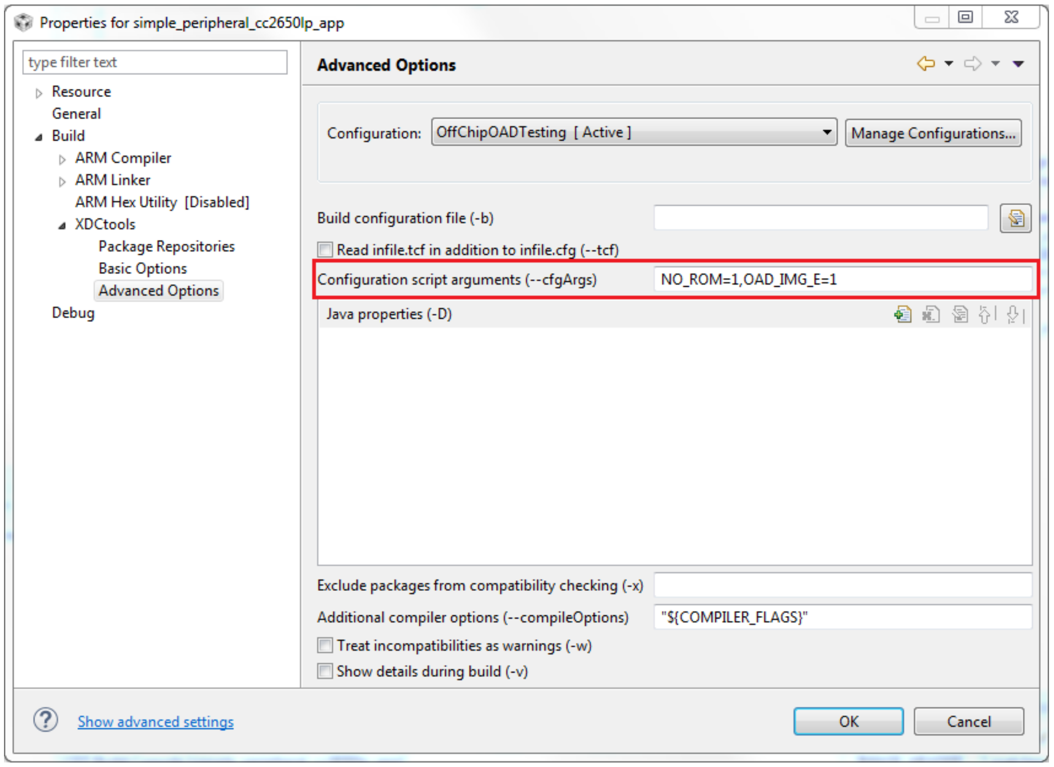
I. Select *Project→Properties→Build→ARM Compiler→Advanced Options→Predefined Symbols* and add the following to *Pre-define NAME*:
   `FEATURE_OAD`

II. Select *Project→Properties→Build→ARM Compiler→Include Options* and add the following lines to *Add dir to #include search path*:
   `"${SRC_EX}/profiles/oad/cc26xx"`

   Select *Project→Properties→Build* and verify the lines below in *Steps→Post-build steps*:

```
${CG_TOOL_HEX} -order MS --memwidth=8 --romwidth=8 --intel -o
${ProjName}.hex ${ProjName}.out
```

Define “NO_ROM=1,OAD_IMG_E=1" in *Project→Properties→Build→XDCtools→Advanced Options* as shown below:


Properties for simple_peripheral_cc2650lp_app

Advanced Options

Configuration: OffChipOADTesting [Active]

Build configuration file (-b)

Read infile.tcf in addition to infile.cfg (--tcf)

Configuration script arguments (--cfgArgs): NO_ROM=1,OAD_IMG_E=1

Java properties (-D)

Exclude packages from compatibility checking (-x)

Additional compiler options (--compileOptions): "${COMPILER_FLAGS}"

Treat incompatibilities as warnings (-w)

Show details during build (-v)

III. Add the OAD profile modules to the PROFILES folder of the workspace. Use cc26xx_ble_app_oad.cmd as a linker command file instead of cc26xx_ble_app.cmd (Right-click on the file →Exclude from Build). These files are located here: `$BLE_INSTALL$/src/profiles/oad/cc26xx`.

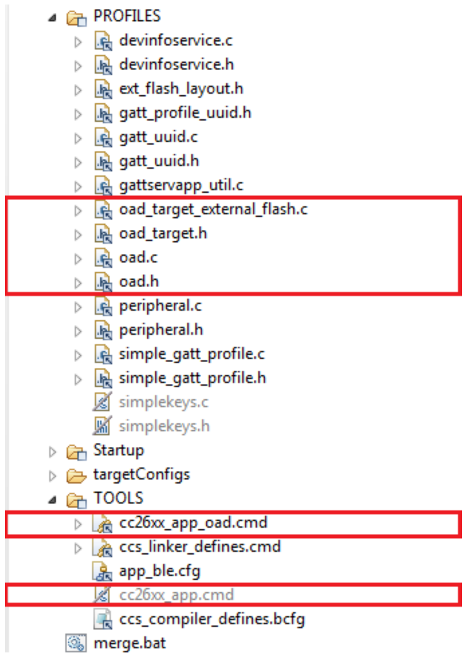


Figure 28. OAD Off-chip files in CCS

6.1.3 Source Code Changes

These changes are required to added OAD to an existing application are independent of toolchain or IDE. Note that these changes should be added to the high level ICall aware application task.

1. Include the proper OAD files

```
#include "oad_target.h"
#include "oad.h"
```

2. [Optional] Set default connection interval for high performance

```
// Minimum connection interval (units of 1.25ms, 8=10ms) if automatic
// parameter update request is enabled
#define DEFAULT_DESIRED_MIN_CONN_INTERVAL     8

// Maximum connection interval (units of 1.25ms, 8=10ms) if automatic
// parameter update request is enabled
#define DEFAULT_DESIRED_MAX_CONN_INTERVAL     8
```

3. Define OAD packet size

```
// The size of an OAD packet.
#define OAD_PACKET_SIZE                    ((OAD_BLOCK_SIZE) + 2)
```

4. Define OAD write callback

```
void SimpleBLEPeripheral_processOadWriteCB(uint8_t event, uint16_t connHandle,
uint8_t *pData);
```

5. Define OAD Write CB structure

```
static oadTargetCBs_t simpleBLEPeripheral_oadCBs =
{
  SimpleBLEPeripheral_processOadWriteCB // Write Callback.
};
```

6. Define OAD Queue structure

```
// Event data from OAD profile.
static Queue_Struct oadQ;
static Queue_Handle hOadQ;
```

7. Inside the application's _init function do the following

```
VOID OAD_addService();              // OAD Profile
OAD_register((oadTargetCBs_t *)&simpleBLEPeripheral_oadCBs);
hOadQ = Util_constructQueue(&oadQ);
```

8. Add the following block of code to the applications' main task loop **after ICall/Stack message processing**

```
while (!Queue_empty(hOadQ))
{
  oadTargetWrite_t *oadWriteEvt = Queue_get(hOadQ);

  // Identify new image.
  if (oadWriteEvt->event == OAD_WRITE_IDENTIFY_REQ)
  {
    OAD_imgIdentifyWrite(oadWriteEvt->connHandle, oadWriteEvt->pData);
  }
```

```
    // Write a next block request.
    else if (oadWriteEvt->event == OAD_WRITE_BLOCK_REQ)
   {
     OAD_imgBlockWrite(oadWriteEvt->connHandle, oadWriteEvt->pData);
    }

    // Free buffer.
    ICall_free(oadWriteEvt);
  }
```

9. Implement the OAD write callback

```
void SimpleBLEPeripheral_processOadWriteCB(uint8_t event, uint16_t connHandle,
                                           uint8_t *pData)
{
  oadTargetWrite_t *oadWriteEvt = ICall_malloc( sizeof(oadTargetWrite_t) + \
                                           sizeof(uint8_t) * OAD_PACKET_SIZE);

  if ( oadWriteEvt != NULL )
  {
    oadWriteEvt->event = event;
    oadWriteEvt->connHandle = connHandle;

    oadWriteEvt->pData = (uint8_t *)(&oadWriteEvt->pData + 1);
    memcpy(oadWriteEvt->pData, pData, OAD_PACKET_SIZE);

    Queue_put(hOadQ, (Queue_Elem *)oadWriteEvt);

    // Post the application's semaphore.
    Semaphore_post(sem);
  }
}
```

6.1.4 [Optional] Changing the pins of the external flash device

It is easiest to use the default pinout when interfacing to the external flash, but if the pins must be changed, the following steps are recommended. Note that the pins must be changed in the BIM image as well as any application images, and they must be in sync. If the images are out of sync in terms of the pins used for external flash, the system may be bricked from future OADs.

1.1.1.1 Updating the BIM's external flash pins

BIM will use the pin definitions in bsp.h to define the external flash pins. BIM will interact with the external flash using a Driverlib only based bare metal interface.

```
// Board external flash defines
#define BSP_IOID_FLASH_CS      IOID_20
#define BSP_SPI_MOSI           IOID_9
#define BSP_SPI_MISO           IOID_8
#define BSP_SPI_CLK_FLASH      IOID_10
```

1.1.1.2 Updating the Application's external flash pins

The application will interface to external flash via the TI-RTOS SPI driver using Board_SPI0 by default. The SPI0's pins are defined in the device's board file (i.e. CC26X0_LAUNCHXL.h):

```
/* SPI Board */
#define Board_SPI0_MISO          IOID_8       /* RF1.20 */
#define Board_SPI0_MOSI          IOID_9       /* RF1.18 */
#define Board_SPI0_CLK           IOID_10      /* RF1.16 */
#define Board_SPI0_CSN           PIN_UNASSIGNED

/* SPI */
#define Board_SPI_FLASH_CS       IOID_20
#define Board_FLASH_CS_ON        0
```

6.2 On-chip OAD

Although the OAD-enabled application image is built and linked separately from the supporting BIM, it must forever adhere to the constraints of the image boundaries and relative locations of external interfaces (e.g. CRC and Image Header) that are expected by BIM. The OAD Target App image is also dependent on BIM existing on the device when it is downloaded as only BIM places its interrupt vectors at the start of flash. Without interrupt vectors at this location, the device will break and become unusable. The example project for building an Image B included in simple_peripheral workspace as 'FlashOnly_OAD_ImgB' configuration can be reproduced through following procedures. The procedures can be applied to convert any existing application to downloadable On-chip OAD Image in IAR.

Note: Only IAR is supported for BLE on-chip OAD on CC26x0/CC13x0 devices.

6.2.1 Steps for IAR

I. Select *Project→Options→C/C++ Compiler→Preprocessor→Defined symbols* and add the following new definitions:

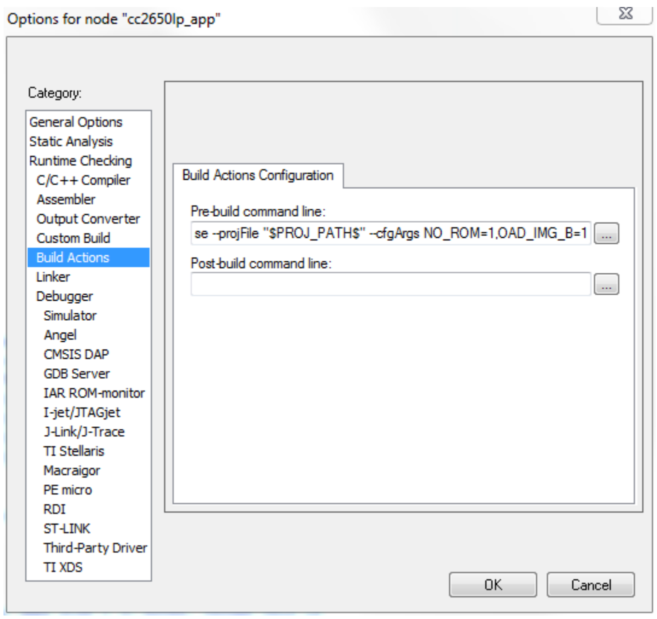
```
FEATURE_OAD
FEATURE_OAD_ONCHIP
IMAGE_INVALIDATE
HAL_IMAGE_B
```

Add the following line to the "Additional include directories":

```
$SRC_EX$/profiles/oad/cc26xx
```

Append the following in Build Actions under *Pre-build command line*:

--cfgArgs NO_ROM=1,OAD_IMG_B=1



Options for node "cc2650lp_app"

Category: Build Actions

Build Actions Configuration

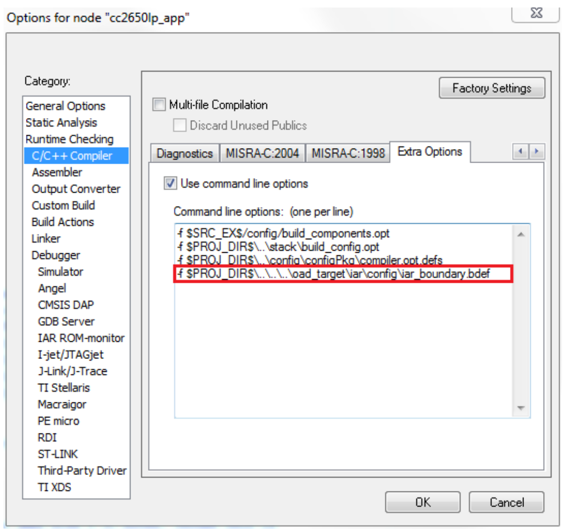
Pre-build command line:

```
se --projFile "$PROJ_PATH$" --cfgArgs NO_ROM=1,OAD_IMG_B=1
```

Post-build command line:

Figure 29. OAD configuro prebuild step (IAR)

Verify in *Project→Options→C/C++ Compiler→Extra Options*, the correct iar_boundary.bdef file is included:



Options for node "cc2650lp_app"

Category: C/C++ Compiler

Multi-file Compilation [ ]
Discard Unused Publics [ ]

Extra Options

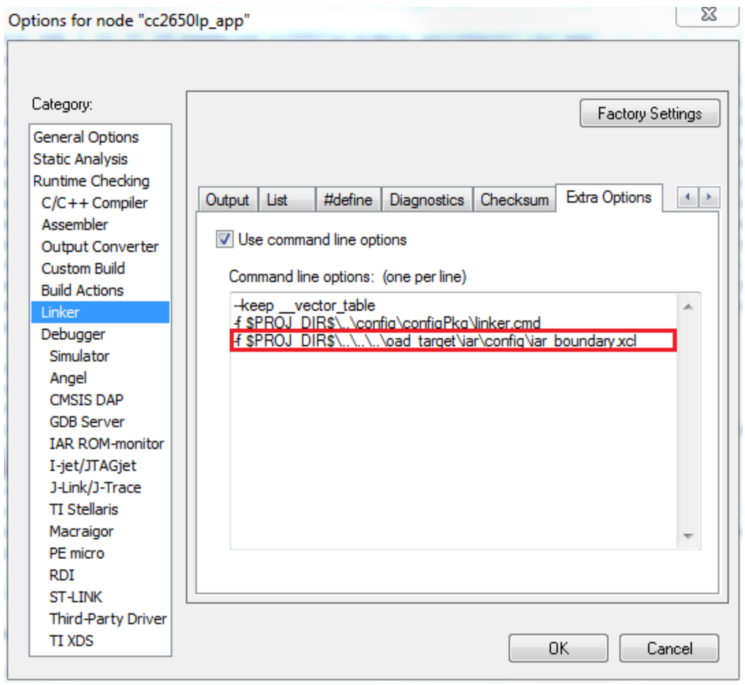
Use command line options [✓]

Command line options: (one per line)

```
-f $SRC_EX$/config/build_components.opt
-f $PROJ_DIR$\..\stack\build_config.opt
-f $PROJ_DIR$\..\config\configPkg\compiler.opt.defs
-f $PROJ_DIR$\..\..\..\oad_target\iar\config\iar_boundary.bdef
```

Figure 30. On-Chip OAD Boundary (compiler)

Verify in Linker -> Extra Options, the correct iar_boundary.xcl file is included:



[✓] Use command line options

Command line options: (one per line)

```
--keep __vector_table
-f $PROJ_DIR$\..\config\configPkg\linker.cmd
-f $PROJ_DIR$\..\..\..\oad_target\iar\config\iar_boundary.xcl
```

Figure 31. On-Chip OAD Boundary (linker)

II. Select *Project→Options→Linker→Config*. Paste the following line to 'Linker configuration file': `$SRC_EX$/common/cc26xx/iar/cc26xx_app_oad.icf`
And add the following symbol to 'Configuration file symbol definitions': `FLASH_ONLY_BUILD=1`

III. Setup the Linker for an image's flash and RAM usage. By default the linker guarantees 10 flash pages, or 40KB, to the OAD image starting at 0x9000 to Image B. It is generally recommended that the values for Image B starting address are not changed from the default settings unless OAD Target App needs to be modified in its size.

IV. Under Tools, include cc26xx_app_oad.icf and exclude cc26xx_app.icf. Also exclude ccfg_app_ble.c. Add the OAD profile modules to the PROFILES folder of the workspace. These files are located here: `$BLE_INSTALL$/src/profiles/oad/cc26xx`.

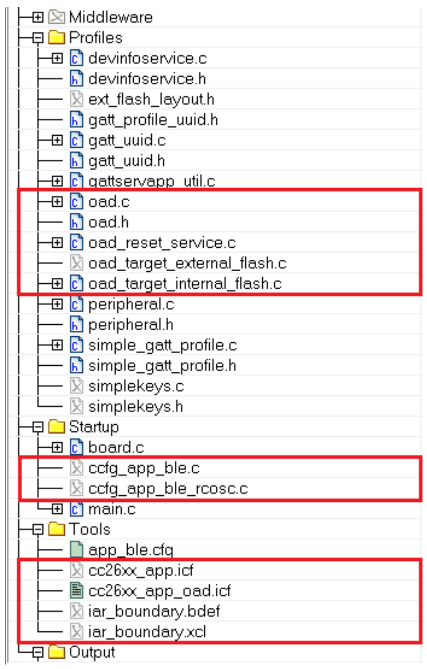


Figure 32. On-Chip OAD Workspace (IAR)

6.2.2 Source Code Changes

These changes are required to added OAD to an existing application are independent of toolchain or IDE. Note that these changes should be added to the high level ICall aware application task. However, since the OAD Target application (Image A) implements most of the OAD functionality, all the needs to be added to the user application is the reset service.

1. Include the proper OAD files

```
#include "oad_target.h"
#include "oad.h"
```

2. In the application's _init() function, call the following routines

```
Reset_addService();
```

7. TI OAD Image Tool

In order to accelerate the process of converting the compiler's hex file output to an OAD ready binary file with embedded metadata, TI has created a Python based OAD image tool. The tool supports On-chip and Off-chip OAD implementations as well as creating super hex merges that should be flashed on the device at production time. OAD Image Tool is provided in source and binary format.

By default the OAD Image Tool will be invoked by OAD sample applications as part of the post build step process. This is recommended method for use.

7.1 Dependencies

The tool requires Python 2.7.10+ to run, as well as the following modules:

```
# Needs Python 2.7.10
from __future__ import print_function
```

```
import __builtin__
import argparse
import crcmod # pip -[--proxy <addr>] install crcmod
from intelhex import IntelHex # pip [--proxy <addr>] install intelhex, needs latest version (v2.1+)
import struct
import textwrap
import sys
import math
import ntpath
from collections import namedtuple
```

When running from source, pip or another package manager should be used to import the required modules.

Running from binary does not require any installation. This is the recommended method.

7.2 Using the tool

The purpose of the tool is to create OTA ready OAD images and also production images. An OTA ready image is defined as an image that has already been processed, merged, and is ready to send to the target device using an OAD downloader. OTA ready images have metadata embedded where necessary.

Production images are images that are intended to be flashed on the device at production time. They contain an entire internal flash image including code that is never updated via OAD such as the BIM.

The tool is configured using a range of arguments that will allow the customer to configure the output of the tool dynamically. The tool's arguments are documented below in **Figure 33**. You may also invoke the help menu of the script by typing:

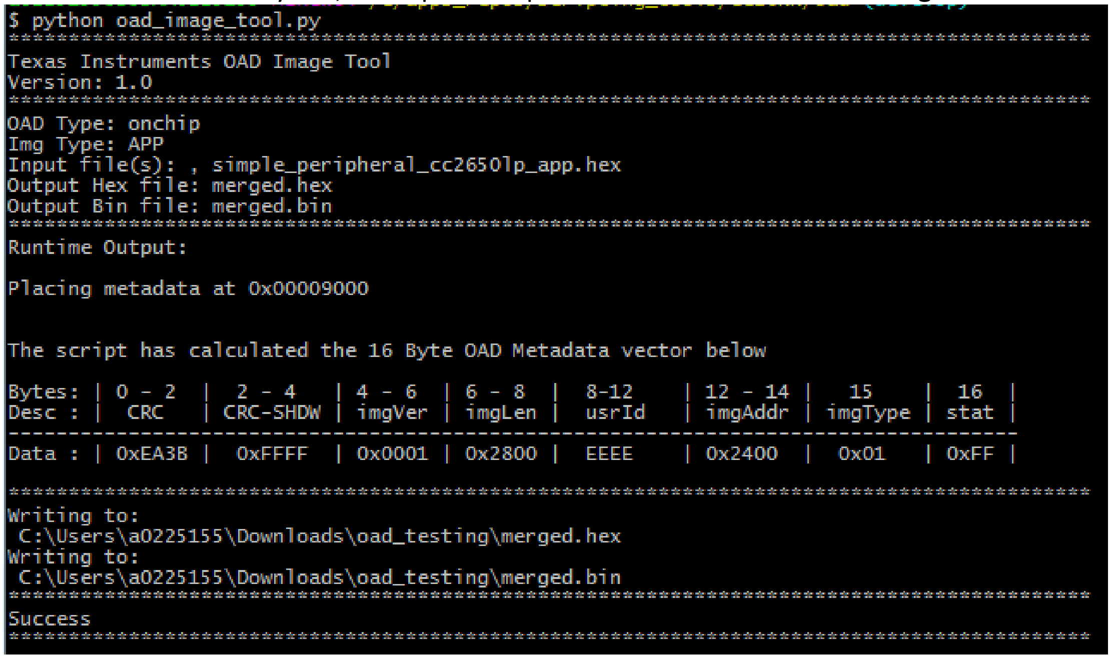
```
python oad_image_tool.py -help
```

Argument	Acceptable input	Description
-h **--help**	None	Display the help menu
-t **--oadtype**	{onchip, offchip}	Whether the generated image is for on-chip or off-chip OAD
-i **--imtype**	{app, stack, np, production}	The type of image to be generated. This argument is used to set the metadata and also enforce some imgType based rules
-v **--imgver**	Any	The version of the image. This is used to populate the metadata
-o	Valid system path	Used to specify where the script

--out		should place the output hex file
-ob **--outbin**	Valid system path	Used to specify where the script should place the output binary file
-f **--fill**	One byte hex value	Value to fill empty addresses within the output image with. Default is 0xFF
-m **--metta**	Any valid internal flash address	Address where the metadata header should be placed
-r **--range**	Any valid internal flash address	Ranges of addresses to be included in the output file
-n **--dry-run**	N/A	Do no produce output files, only print information
-q **--quiet**	N/A	Only produce output files, do not print to the console
--round	Valid sector size	Round sectors up to the nearest internal flash sector size.
--version	N/A	Print the version info of this tool

Figure 33. OAD Image Tool Arguments

Once the tool has successfully run, it will print output similar to what is shown in **Figure 34**.



```
$ python oad_image_tool.py
******************************************************************************************
Texas Instruments OAD Image Tool
Version: 1.0
******************************************************************************************
OAD Type: onchip
Img Type: APP
Input file(s): , simple_peripheral_cc2650lp_app.hex
Output Hex file: merged.hex
Output Bin file: merged.bin
******************************************************************************************
Runtime Output:

Placing metadata at 0x00009000


The script has calculated the 16 Byte OAD Metadata vector below

Bytes: | 0 - 2  |  2 - 4   | 4 - 6  | 6 - 8  |  8-12     | 12 - 14 |   15    |  16  |
Desc : |  CRC   | CRC-SHDW | imgVer | imgLen |  usrId    | imgAddr | imgType | stat |
-------------------------------------------------------------------------------------
Data : | 0xEA3B |  0xFFFF  | 0x0001 | 0x2800 |  EEEE     | 0x2400  |  0x01   | 0xFF |

******************************************************************************************
Writing to:
 C:\Users\a0225155\Downloads\oad_testing\merged.hex
Writing to:
 C:\Users\a0225155\Downloads\oad_testing\merged.bin
******************************************************************************************
Success
******************************************************************************************
```

Figure 34. OAD Image Tool Output

7.3 Building a Production Image

The tool supports two types of production images depending on how your application will be supporting OAD; On-Chip or Off-Chip. As previously described, On-Chip will require metadata of the image that will be initially flashed onto the device. Use the –production flag for creating an image that is merged with BIM.

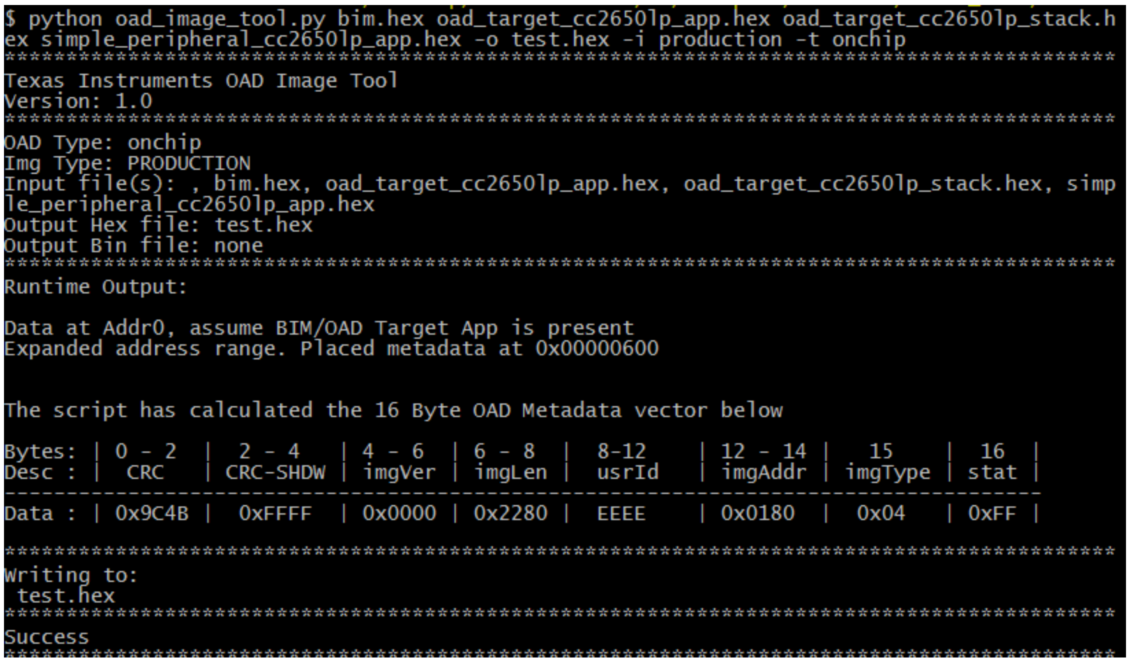
1.1.1.3 On-Chip OAD Production Image Example Usage

On-Chip OAD's Production image is a hex merge of BIM for On-Chip OAD, Image A (the resident Application that implements the TI OAD Profile + Uses the BLE Stack), Image B (the initial Application) and the BLE Stack.

Once all the hex files are correctly generated– invoke the TI OAD Image Tool with:

<python> <oad_image_tool.py> <BIM hexfile> <Image A hexfile> <Stack hexfile> <Image B hexfile> -o <Output hexfile> -i production –t onchip

Where '<*item*>' indicate location of *item* on the system. The flags used in the tool are highlighted. The output should look similar to **Figure 35**.



```
$ python oad_image_tool.py bim.hex oad_target_cc2650lp_app.hex oad_target_cc2650lp_stack.h
ex simple_peripheral_cc2650lp_app.hex -o test.hex -i production -t onchip
*******************************************************************************************
Texas Instruments OAD Image Tool
Version: 1.0
*******************************************************************************************
OAD Type: onchip
Img Type: PRODUCTION
Input file(s): , bim.hex, oad_target_cc2650lp_app.hex, oad_target_cc2650lp_stack.hex, simp
le_peripheral_cc2650lp_app.hex
Output Hex file: test.hex
Output Bin file: none
*******************************************************************************************
Runtime Output:

Data at Addr0, assume BIM/OAD Target App is present
Expanded address range. Placed metadata at 0x00000600


The script has calculated the 16 Byte OAD Metadata vector below

Bytes: | 0 - 2  |  2 - 4   | 4 - 6  | 6 - 8  |  8-12    | 12 - 14 |   15    |  16  |
Desc : |  CRC   | CRC-SHDW | imgVer | imgLen |  usrId   | imgAddr | imgType | stat |
-------------------------------------------------------------------------------------
Data : | 0x9C4B |  0xFFFF  | 0x0000 | 0x2280 |  EEEE    | 0x0180  |  0x04   | 0xFF |

*******************************************************************************************
Writing to:
 test.hex
*******************************************************************************************
Success
*******************************************************************************************
```

Figure 35. OnChip OAD Production Image Example invocation.

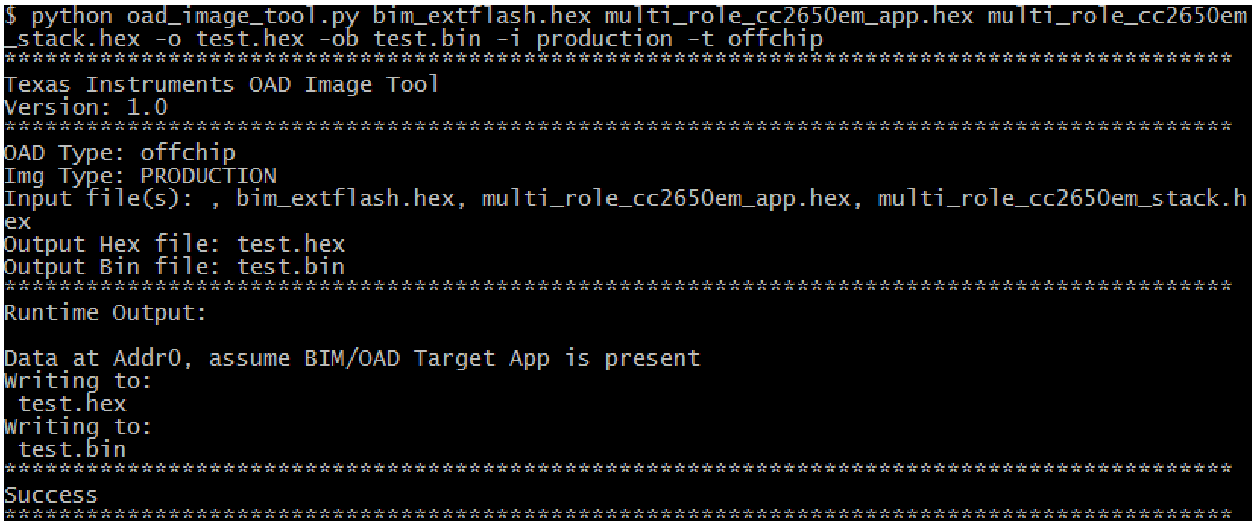
1.1.1.4 Off-Chip OAD Production Image Example Usage

Off-Chip OAD's Production image is simply a hex merge of BIM for External Flash + Initial Application Image + Initial Stack Image.

Once all the hex files are correctly generated – invoke the TI OAD Image Tool with:

<python> <oad_image_tool.py> <BIM hexfile> <App hexfile> <Stack hexfile> -o <Output hexfile> -i production –t offchip

Where '<*item*>' indicate location of *item* on the system. The flags used in the tool are highlighted. The output should look similar to **Figure 36**.



```
$ python oad_image_tool.py bim_extflash.hex multi_role_cc2650em_app.hex multi_role_cc2650em
_stack.hex -o test.hex -ob test.bin -i production -t offchip
*******************************************************************************************
Texas Instruments OAD Image Tool
Version: 1.0
*******************************************************************************************
OAD Type: offchip
Img Type: PRODUCTION
Input file(s): , bim_extflash.hex, multi_role_cc2650em_app.hex, multi_role_cc2650em_stack.h
ex
Output Hex file: test.hex
Output Bin file: test.bin
*******************************************************************************************
Runtime Output:

Data at Addr0, assume BIM/OAD Target App is present
Writing to:
 test.hex
Writing to:
 test.bin
*******************************************************************************************
Success
*******************************************************************************************
```

Figure 36. Off-Chip OAD Production Image Example invocation

8. Appendix

The appendix covers various topics related to OAD

8.1 References

1. Texas Instruments CC2640 Bluetooth® low energy Software Developer's Guide
   http://www.ti.com/lit/pdf/swru393
2. OAD User's Guide Wiki Page
   http://processors.wiki.ti.com/index.php/CC2640_OAD_User%27s_Guide
3. CC2640R2 SDK Download Link
   http://www.ti.com/tool/download/SIMPLELINK-CC2640R2-SDK/1.30.00.25
4. SimpleLink Academy
   http://software-dl.ti.com/lprf/simplelink_academy/overview.html
5. CC2650 LaunchPad
   http://www.ti.com/tool/launchxl-CC2650
6. TI BLE-Stack 2.2.x
   http://www.ti.com/ble-stack
7. Smart RF Flash Programmer 2
   http://www.ti.com/tool/flash-programmer
8. BTool User's Guide
   http://software-dl.ti.com/lprf/simplelink_cc2640r2_sdk/1.30.00.25/exports/docs/blestack/btool_user_guide/BTool_Users_Guide/index.html

8.2 Definitions, Abbreviations, Acronyms

Term	Definition
BIM	Boot Image Manager, the software bootloader
BLE	Bluetooth low energy wireless protocol
CCCD	Client Characteristic Configuration Descriptor
CCFG	Customer Configuration Area, contains lock-bits on flash page 31
CCS	Code Composer Studio
MCU	Microcontroller Unit
OAD	Over the Air Download
RCFG	RTOS in ROM Configuration Table
ROM	Read Only Memory
RTOS	Real Time Operating System
SNV	Simple Non-Volatile storage
TI	Texas Instruments
TI-RTOS	Texas Instruments Real Time Operating System

8.3 RTOS RCFG Section

In order to save space, the CC26xx contains portions of the TI-RTOS kernel in its ROM image. These ROM functions have dependencies on a data structure called the RCFG within flash page 0 of internal flash. The RCFG section is essentially a table of function pointers (and some other kernel data) that allows the kernel ROM functions to access certain data structures in the internal flash memory.

When building a new RTOS in ROM image, the RCFG must be updated. However, since the RCFG lies within page 0 (which contains the vector table and thus is write protected) it cannot be updated via OAD.

This means that all images intended to be sent over the air must use an RTOS in flash configuration that includes their own instance of the RCFG.

8.4 Prevent On-chip BIM from checking the CRC

By default on-chip BIM will not check the CRC of Image A because it is intended to be flashed during production and not updated. If it is necessary to check the CRC of Image A, the FEATURE_FIXED_IMAGE can be undefined.

9. Troubleshooting Guide

9.1 General Troubleshooting Guide

There are various places where OAD can fail; use the following steps to determine where the issue is occurring during the interaction:

- Use a BLE Packet Sniffer and Record the OAD Transaction
  This will verify that the profile is implemented correctly and a valid image was transferred over the air.
    - Look for a OAD Initiation
      Notifications from OAD Image Notify should be requested – and the appropriate response from the OAD Target after a GATT Write of the Candidate metadata.
    - Look for OAD Image Status Characteristic
      This will contain the status of the image prior to BIM launching the image.

Solutions –

- Verify that the OAD Downloader supports the TI OAD Profile as a GATT Client
- Verify that the OAD Downloader is sending the correct image with updated meta data
- Verify that the OAD Target supports the TI OAD Profile as a GATT Server

- Read external/internal flash to ensure CRC Shadow is valid and matches CRC field
  This will verify that the image was received fully without errors by the OAD Target
    - For Internal, various tools can be used. SmartRF Programmer 2 or using a jtag debugger and connecting to running target and utilizing a memory viewer are possible options.
    - For External, interface another MCU or another serial device to dump out the Flash.

Solutions –

- Power cycle and Retry Downloading the Image
- Verify that the Flash Pages, on-chip or off are not corrupt

9.2 Downloaded OAD Image Isn't Starting

Assuming that the device has been flash correctly and is verified to be working prior to an OAD, then a non-responsive device is likely stuck in Halt in Boot. The Halt in Boot scenario and how to work around it is covered in detail on the OAD User's Guide Wiki Page under the section "Device Does not restart after successful OAD"

9.3 Should External Flash be used during OAD?

No, External flash should not be utilized while OAD Profile is active; the Profile is designed to have uninterrupted access to Flash.

9.4 Individually Flashing Hex Files for On-Chip OAD - App doesn't work!

The issue is due to the tools being designed to work on pages instead of on words. When two applications utilize the same page this problem manifests. For example, in On-Chip OAD, BIM and Part of the Image A (the resident application image) share Page 0. Depending on which application is flashed second, the information from the first image will not be preserved.

Solution –

Generate a Merged Hex File instead – this can be done with the TI OAD Image Tool. Look at Section 1 for more information on this tool. Then Smart RF Flash Programmer 2 to flash the merged hexfile.

Utilize the .out files for debugging information and connect to the running target using IAR to verify functionality if needed.

9.5 Debugging BIM/ Pre application

In the case where the expected image is not being selected or loaded, it may be necessary to debug the BIM. The BIM can be debugged by loading its symbols into the application before starting a debugging session. This will trap the device in the reset_isr() routine and allow the user to debug through the BIM main and into the application's entry point in main(). Refer to your specific toolchain for how to load symbols from an image in a debugging session.