Application Note

WICED™ OTA2

WICED™ OTA2 Update Support

(Over The Air update)

5300 California Avenue • Irvine, California, 92677 • Phone: 949-926-5000 • Fax: 949-926-5203

WICED-OTA2
August 07, 2015

Revision History

Revision	*Date*	*Change Description*
WICED-OTA2-R1.07	November 9, 2016	Added information about Failsafe Support Removed Known Vulnerabilities
WICED-OTA2-R 1.06	October 17, 2016	Added FLASH and RAM requirements.
WICED-OTA2-R 1.05	October 6, 2016	Removed some Vulnerabilities as they have been addressed. Added API documentation.
WICED-OTA2-R 1.04	September 27, 2016	Added Known Vulnerabilities
WICED-OTA2-R 1.03	September 21, 2016	Added known issues
WICED-OTA2-R 1.02	August 9, 2016	More info concerning console commands.
WICED-OTA2-R 1.01	July 28, 2016	ota2_extract description and usage, DCT between SDKs reminders, misc updates
WICED-OTA2-R 1.00	July 8, 2016	OS support description
WICED-OTA2-R 0.99	June 6, 2016	Instructions for adding OTA2 to new platforms.
WICED-OTA2-R 0.98	June 1, 2016	Addition of SoftAP support, updated some descriptions.
WICED-OTA2-R 0.97	March 2, 2016	Minor updates
WICED-OTA2-R 0.96	February 5, 2016	Update API for Background Service Support And External + Internal FLASH support
WICED-OTA2-R 0.95	December 18, 2015	Add Usage notes and bring doc up to date
WICED-OTA2-R 0.94	December 2, 2015	Bring doc up to date with latest code
WICED-OTA2-R 0.93	November 3, 2015	Add more details, update APIs for latest code
WICED-OTA2-R 0.92	October 20, 2015	Changed naming from OTA to OTA2
WICED-OTA-R 0.91	October 6, 2015	Adding more explanations
WICED-OTA-R 0.9	September 23, 2015	Adding details Add CRC of OTA Header + Component Headers Add Internal + External FLASH layout Add download_status and bytes_received updating Adjust comments in API for callback return values
WICED-OTA-R 0.8	September 4, 2015	Add FLASH layout Add structure header definitions Add preliminary APIs
WICED-OTA-R 0.7	August 21, 2015	Fix up table layout due to merge “Push” model compiler option
WICED-OTA-R O.6	August 20, 2015	Insert watermark
WICED-OTA-R 0.5	August 19, 2015	WiFi Firmware is part of apps_lut.
WICED-OTA-R 0.4	August 17, 2015	Incorporate Evan comments, more description of OTA Image, Simplified Flash structure, incorporating existing code/techniques (wiced_apps_lut[] in particular)
WICED-OTA-R 0.3	August 07, 2015	Updates with James/Stephen/Jay/Kelly
WICED-OTA-R 0.2	August 06, 2015	Work on Bootloader/start sequence
WICED-OTA-R 0.1	August 04, 2015	Initial Revision

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Table of Contents

1 About this Document

1.1 Purpose and Scope

This document provides instructions to use the WICED OTA2 Image Packages and samples to provide Over The Air Update capability to your application. Using the sample Applications, API's and WICED SDK utilities, you will be able to add "call home" capability to your IoT device to update system images dynamically with little or no intervention, as well as have full administrative capabilities for software maintenance.

Note: This document applies to **WICED SDK 3.6.0** or higher.

1.2 Audience

This document is for software developers who are using the WICED Development System to create applications for secure embedded wireless networked devices.

2 Terminology

Bootloader	The initial program that is run when power is applied. Initializes hardware and decides which Application to run.
Current Application	The Currently running validated Application.
Current Application Area	Area of FLASH reserved for the Current Application (and data).
Factory Reset	Returning the product to the state it was in when first manufactured.
Factory Reset OTA Image	The OTA Image initially shipped with the product, FLASHed into the device at manufacture, extracted to Current Application Area in production.
HTTP	Hyper-Text Transfer Protocol
HTTPS	Hyper-Text Transfer Protocol (Secure)
LKG Area	Last Known Good Area. If the FLASH on the device is sufficient, an LKG Application Area is set aside. Before an update OTA Image is extracted, the Current Application Area is copied (1:1) to the LKG Area. If the OTA extraction fails, then the LKG Area can copied back to the Current Application Area as a first backup.
LUT	Look Up Table – in the Wiced Multi-Application Framework, this is a simple directory where the system (App, DCT, Resources) is located in FLASH.
OTA2	Over The Air updating of the software in the device over WiFi (OTA2 is to distinguish this support from previous OTA support).
OTA2 Update Image	The OTA2 Image sent to the device to update the current software. The OTA2 Image consists of a Header (size, secure signing, etc.) and Data to update the device (File System, Application, DCT, and Resources).

OTA2 Staging Area	Area of FLASH set aside to save the downloaded OTA2 Update Image before it is extracted to the Current Application Area.
SoftAP	Software based Access Point. The device can become an AP so you can connect your WiFi enabled computer to the device to perform updates or for adjusting settings.
Watchdog Timer	Independently running hardware timer used to determine if software has stopped running (hung or crashed). Time is configurable.

3 Over The Air 2 Product Update Description & Overview

This document describes the system software, utilities, and reference application(s) and snippets which demonstrate OTA2 capability along with how to enable OTA2 update in your WICED application. Using this documentation, the developer will learn how to use the WICED libraries for manual and background over the air product updates.

The update mechanism supports the following components:

- Configuration and build options to define the required storage volumes/file systems for all of the assets required by the update (Application, DCT, Resources, etc). This allows the developer to customize their platform to meet their storage and software life-cycle needs up-front throughout the lifetime of the product.
- A utility which collects all needed binary assets into an OTA2 or “firmware update” unified file format (OTA2 Image). This is stored on the customer’s server for user updates.
- A Bootloader which can determine if a valid OTA2 Image is present and ready for extraction (Staged), if the Current Application File System is valid, attempt to use a backup copy (LKG) if enabled, or defaulting to a Factory Reset OTA2 Image when no valid application is found (e.g. the unit was bricked).
- A SoftAP Webserver Interface for manually updating the product software via the HTTP protocol and browser based file uploads.
- A timed background function that checks the customer’s designated update server for an image and then uses HTTP to retrieve and update a system image. Facility is available for the Application to be notified that an update is available, and allow the Application to retrieve the data in its own fashion and save the data so that the Bootloader can extract the OTA2 Image on the next power cycle.

3.1 OTA2 OS Support

OTA2 is supported under the ThreadX Operating System.

3.2 OTA2 Platform Support

OTA2 is supported on BCM94390x and STM32F4xx platforms under ThreadX. See the individual <Wiced-SDK>/platform/<platform-name> directories for files described in the “Adding OTA2 to a new platform” section below.

4 Memory Layout

Most Systems do not contain enough internal FLASH to accommodate the needs of OTA2, and will require external FLASH to store the code and data needed.

4.1 Memory Requirements

4.1.1 OTA2 Flash Requirements

Size	Description
16k	Bootloader
128k	OTA2 Failsafe Application (snip/ota2_failsafe)
1MB+	OTA2 Image - Factory Restore Area (includes reset DCT + LUT + Filesystem + OTA2 Extractor + Application Size)
16k	OTA2 DCT Save Area
32k	DCT Copies (normal program use)
4k	Application Look Up Table (LUT)
512k	Filesystem (may include WiFi + BT Firmware)
256k	OTA2 Extractor (SoftAP, DHCP server, Web Server, OTA2 extraction code)
256k	Medium Sized Application (WiFi, BT)
1MB+	OTA2 Image - Staging Area (includes new DCT + LUT + Filesystem + OTA2 Extractor + Application Size)
3MB+	TOTAL + Application Size

4.1.2 OTA2 RAM Requirements

Example Application

256k	Application (Code/Data)
16k	DCT copy in RAM (optional, can be read directly from FLASH)
64k	Simple Network Heap (32K for 10 TX/RX buffers)
336k	Sub Total (Application)

Adding OTA2 to Example Application

215k	OTA2 Library
25k	OTA2 Static RAM
64k	Allocated RAM for context and sector buffering
304k	Sub Total (OTA2)
640k	Total (Application (326k) + OTA2 support)

Generic & Configurable System Storage Layout

A means will be provided in the WICED SDK, Makefiles and build infrastructure to define regions for storing system software images and associated metadata assets. The concept of volume/file system will be supported and board level support for external and internal FLASH virtualized by the WICED system software. The following are the basic requirements for System Storage layout:

READ ONLY SECTION

- Bootloader
- Factory Reset OTA2 Image

Read / Write Section

- System and Application DCT
- Copy of Application DCT (copied before any OTA2 Image is extracted)
- Current Application Area
    - File system

    - Current Application
    - Etc.
- OTA2 Extract Application Area
    - SoftAP Application
- Last Known Good Area (optional – not supported)
    - Requires more FLASH
    - 1:1 copy of Current Application Area before new OTA2 Image is extracted
    - First level fallback support
- OTA2 Image Staging Area
    - Storage for new OTA2 Image update

5 Product Application Utilities

5.1 OTA2 Bootloader Application

The OTA2 Bootloader is a small application that initializes any needed system resources and launches the Current Application. It will check for button presses and for the presence of a staged OTA2 Image, to allow for manual factory reset or automatic product software updates. The OTA2 Bootloader resides in a write-protected part of the Storage System.

The OTA2 Bootloader is finalized at shipment and is not updated through the product cycle. The OTA2 Failsafe Application (discussed in the next section) is always built with the OTA2 Bootloader. It is also finalized at shipment and is not updated through the product cycle.

Upon powering up the board, if the reset button is pressed for ~5 seconds:

- Start an application (snip/ota2_extract) that will run a SoftAP with pre-defined Network parameters (system DCT)
- The application will start DHCP and a Webserver so a user can manually connect and load an OTA2 Image via a browser based upload.
- This functionality will be configurable (excluded) if a “Push” model is not required.

Upon powering up the board, if the reset button pressed for ~10 seconds:

- Start an application (snip/ota2_extract) that will perform a Factory Reset, extracting the Factory Reset OTA2 Image to the Current Application Area.

During boot, the OTA2 Bootloader will attempt to load the Current Application. If that fails, if there is a valid OTA2 Image in the Staging Area, the OTA2 Bootloader will start an extraction. If not, the OTA2 Bootloader will start an extraction of the Factory Reset image . See the OTA2 Failsafe Application section below for more information.

5.2 OTA2 Extract Application

This application is used by the Bootloader to:

- Extract a downloaded OTA2 Image from the Staging Area
- Extract the Factory Reset OTA2 Image
- Start the SoftAP, DHCP server, and web server to allow connection from a PC to “PUSH” a new version to the OTA2 Staging Area.

NOTE: The ota2_extract application is updated when an OTA2 Image is extracted, and must be included as part of the build of your updated application. To do this, you need to add the following lines to your <application>.mk file:

```
#OTA SoftAp application
OTA_APPLICATION:= snip.ota2_extract-$(PLATFORM)

OTA_APP     := build/$(OTA_APPLICATION)/binary/$(OTA_APPLICATION).stripped.elf
```

5.3 OTA2 Failsafe Application

There was a vulnerability in OTA2 updates in that if the device was reset or power cycled during an OTA2 extraction, the device could have been left in an unknown state.

The OTA2 Failsafe Application can restore a system to the valid OTA2 image in the Staging Area or the OTA2 Factory Reset Image if the system fails to boot. It is required for all builds, and is automatically built and downloaded with the ota2_bootloader.

NOTE: If the bootloader in the device was built before SDK-4.0.1, it does not have the OTA2 Failsafe Application. Updating using OTA2 will not add the failsafe functionality. Only devices built with SDK-4.0.1 (and later) will have the failsafe application.

5.4 Web Server API

The Web Server API is a small web server library that will allow the user to connect to the product with a standard browser on their PC and upload new software, reset to factory settings, or other functionality as required by the product. The Web Server supports a RESTful architecture, and can support the customer's customization needs. For those devices which are running their own SoftAP (e.g. a PUSH model), the user may connect via a Web-browser to the device and perform web-based uploads. The general architecture is an embedded server which supports browser based uploads and custom CGI/JS on the WICED device side; these custom CGI's will provide server (device) side update of FLASH regions.

This functionality is used in the snip/ota2_extract application and used by the snip/ota2_example program to provide SoftAP support.

5.5 OTA2 Image Packaging Application

The OTA2 Packaging Application is a utility application which will take the various product Components (DCT, Application, File system, etc.) and create an OTA2 Image. The OTA2 Image consists of a Header and all components placed in one linear flat file (suitable for TCP transport). The Header includes a length, size and Secure Signing over all of the assets.

OTA2 Image Package includes:

- OTA2 Image header
  - Software Version
  - CRC of the OTA2 Image Header
  - Size of OTA2 Image
  - Secure signature for the enclosed package
- OTA2 Image Component Headers and Components, including:
  - DCT
  - Application
  - OTA2 Extractor
  - File system
  - Etc.

To build an OTA2 Image file, see **OTA2 Image Packager Application** section below:

5.6 OTA2 Image Extraction API

The OTA2 Image Extraction API is a utility library which will extract the data from the downloaded OTA2 Image. The OTA2 Image Extraction API has function calls to verify the OTA2 Image (secure signing) and the individual components included in the OTA2 image (see OTA2 Image Packaging Application above).

The OTA2 Image Extraction API is demonstrated in both snip/ota2_example and in snip/ota2_extract.

5.7 OTA2 Service for background Updates

The product may utilize a timed process to connect to the Customer's website to check for updates periodically through the User's home Wi-Fi network. The OTA2 Service will connect to a specified web server to check for updates at a regular time interval. The connection can be a secure connection, and updates loaded only if there is a version newer than the current version, that also works with the product board version. This is equivalent to an HTTP pull of the image file (e.g. a wget-like utility which fetches the OTA2 Image and updates the system images).

5.8 OTA2 Updates and DCTs

There is a facility for the newly updated Application to access the previously run Application's DCT and copy any pertinent information, to keep any settings the User has made intact across the Update.

When performing an OTA2 Image extraction (update), the sequence will be:

1) Copy the Current Application DCT to a DCT Save location.
2) Extract the OTA2 Image
3) The new updated Application can now access the previously run Application's DCT copy and use fields as appropriate.

6 OTA2 Bootloader Logic

The state of the Current Application Area, LKG (if enabled) and Staged OTA2 Image is stored in the OTA2 DCT. The OTA2 Bootloader will run the snip/ota2_extract application if OTA2 Image Extraction is needed.

The OTA2 Image Staging area will have a status value (incorporated into the OTA2 Image Header) so that a background OTA2 Service running from the Current Application (or the SoftAP in snip/ota2_extract) can save an OTA2 Image and the download progress so that the OTA2 Bootloader can determine if there is a valid OTA2 Image to be extracted on a reboot (it will then launch the snip/ota2_extract application to do the extraction).

NOTES:

- The FLASH writing code can check for battery level before starting writing.
- The WICED software CRC32 calculation can be replaced at compile time.

6.1 OTA2 Bootloader sequence

6.1.1 Check for Watchdog Reset

If the OTA2 Bootloader detects that a System Watchdog caused a system reset, the OTA2 Bootloader will set the boot_type to be OTA2_BOOT_FAILSAFE_UPDATE (if the OTA2 Staged Area has a valid image) or OTA2_BOOT_FAILSAFE_FACTORY_RESET (if there is no valid image in the OTA2 Staged Area).

6.1.2 Check for Failsafe Recovery

Just before an OTA2 Image Extraction call, the boot_type is set to OTA2_BOOT_FAILSAFE_UPDATE (for Staged Area Image Extraction) or OTA2_BOOT_FAILSAFE_FACTORY_RESET (for Factory Reset Image Extraction). If the current boot_type is one of these two types, immediately load the OTA2 Failsafe Application and extract the minimal number of components to allow for extraction of an OTA2 Image. The two components are the Application Look Up Table (Apps LUT) and the OTA2 Extraction Application (snip.ota2_extract). With these two components, the OTA2 Bootloader can extract the remaining parts of an OTA2 Image.

- For OTA2_BOOT_FAILSAFE_UPDATE, try to extract the LUT and ota2_extract from the OTA2 Image in the Staging Area.
  - If this is interrupted the boot_type is still set to OTA2_BOOT_FAILSAFE_UPDATE, and when the system reboots, it will try again.
  - After a successful extraction of these 2 components, set the boot_type to OTA2_BOOT_EXTRACT_UPDATE
- For OTA2_BOOT_FAILSAFE_FACTORY_RESET, extract the LUT and ota2_extract from the OTA2 Image in the Factory Reset Area.
  - If this is interrupted the boot_type is still set to OTA2_BOOT_FAILSAFE_FACTORY_RESET, and when the system reboots, it will try again.
  - After a successful extraction of these 2 components, set the boot_type to OTA2_BOOT_EXTRACT_FACTORY_RESET

6.1.3 Is the OTA2 Image status in the Staging Area is “Extract on Boot”?

If boot_type is **not** OTA2_BOOT_EXTRACT_FACTORY_RESET or OTA2_BOOT_EXTRACT_UPDATE, check the status of the OTA2 Image in the Staged Area Image. If the status of WICED_OTA2_IMAGE_EXTRACT_ON_NEXT_BOOT, set the boot_type to OTA2_BOOT_EXTRACT_UPDATE.

6.1.4 Check for Button Press to Start SoftAP

This support is configurable (can be turned off) at compile time.

If the button is pressed for 5 seconds, `set the boot_type to` OTA2_BOOT_SOFTAP_UPDATE.

6.1.5 Check for Button Press to Start Factory Reset

This time is configurable at compile time. If no SoftAP is supported, the default time will be ~5 seconds. If button is pressed for 10 seconds, `set the boot_type to` OTA2_BOOT_EXTRACT_FACTORY_RESET.

6.1.6 Act on current boot_type

If boot type is OTA2_BOOT_EXTRACT_FACTORY_RESET, run the OTA2 Extractor program and extract the OTA2 Image in the Factory Reset Area.

If boot_type is OTA2_BOOT_EXTRACT_UPDATE, run the OTA2 Extractor program and extract the OTA2 Image in the Staging Area.

If boot_type is OTA2_BOOT_SOFTAP_UPDATE, run the OTA2 Extractor program and start the Softap, DHCP Server and Web Server.

If boot_type is OTA2_BOOT_NORMAL, run the application.

7 FLASH layout and application/DCT regions

Flash offsets and sizes are defined in <platform>/ota2_image_defines.mk.

7.1 External FLASH

External FLASH holds all components. Code is loaded into RAM for execution.

Currently supported on BCM943907WAE_1 and BCM943909WCD1_3 platforms.

External only Flash File System
Bootloader & Factory Reset Area Read Only Section (offset = 0x00, address increases down this table)

Bootloader	Starts from power on to check for these events to run the ota2_extract application: - OTA2 Image in Staging Area - 5 second button press to start SoftAP - 10 second button press to start Factory Reset - force_factory_reset flag
Factory Reset OTA2 Image	Factory Reset OTA2 Image. (This is ***extracted*** to Current Application Area before being used. This is the OTA2 Image used for the initial production and for catastrophic failure, as in a bricked board).
Failsafe Application	OTA2 Failsafe Application. (Used to restore a system that was reset or had a power cycle during the extraction process.)
Application DCT Copy Area (Application DCT copy to save DCT during update) Read / Write	
DCT Copy Area	Current Application DCT is copied here before OTA2 Upgrade Image extraction. (Allows User settings to remain intact during an OTA2 update.)
Current Application Area Read / Write	
Current Application Area	wiced_apps_lut[] DCT Resources file system (WiFi firmware, images, audio, web pages, etc.) Application(s) such as the SoftAP.
OTA2 Extract Application Area Read/Write	
OTA2 Extraction Application	Application used to carry out extraction duties or SoftAP/DHCP/web server when bootloader deems them necessary.

Last Known Good Area Read / Write	
Last Known Good Area – Optional	1:1 copy of Current Application Area before a new OTA2 Image extraction. (Optional – requires more FLASH. If the extraction fails, the LKG will be copied back to the Current Application Area to restore Last Known Good version.)
OTA Image Staging Area (storage for downloading a new OTA Image)	
OTA2 Image Staging Area	New OTA2 Upgrade Image. Individual components may be compressed for size. (This is ***extracted*** to Current Application Area before being used).

NOTES: All major section sizes are known at compile time and will not change over the life of the product.

7.2 Internal + External FLASH (Wiced-SDK-3.6.0 and higher)

Internal FLASH has Bootloader and Application; they execute in place. External FLASH holds all other components. Support on platforms is ongoing.

Internal Flash File System	
Bootloader Read Only Section (offset = 0x00, address increases down this table)	
Bootloader	Starts from power on to check for these events to run the ota2_extract application: - OTA2 Image in Staging Area - 5 second button press to start SoftAP - 10 second button press to start Factory Reset - force_factory_reset flag
Current Application Area Read / Write	
Current Application Area	Application(s) only

External Flash File System	
Factory Reset Area Read Only Section (offset = 0x00, address increases down this table)	
Factory Reset OTA2 Image	Factory Reset OTA2 Image. (This is ***extracted*** to Current Application Area before being used. This is the OTA2 Image used for the initial production and for catastrophic failure, as in a bricked board).

Failsafe Application	OTA2 Failsafe Application. (Used to restore a system that was reset or had a power cycle during the extraction process.)
Application DCT Copy Area (Application DCT copy to save DCT during update) Read / Write (Items can be in Internal or External Flash)	
DCT Copy Area	Current Application DCT is copied here before OT2A Upgrade Image extraction. (Allows User settings to remain intact during an OTA2 update.)
Current Application Area Read / Write	
Current Application Area	wiced_apps_lut[] DCT Resources file system (WiFi firmware, images, audio, web pages, etc.) Application(s) such as the SoftAP.
OTA2 Extract Application Area Read/Write	
OTA2 Extraction Application	Application used to carry out extraction duties or SoftAP/DHCP/web server when bootloader deems them necessary.
Last Known Good Area (optional, requires more FLASH) Read / Write	
Last Known Good Area – Optional	1:1 copy of Current Application Area before a new OTA2 Image extraction. (Optional – requires more FLASH. If the extraction fails, the LKG will be copied back to the Current Application Area to restore Last Known Good version. For future expansion, not currently implemented)

OTA Image Staging Area (storage for downloading a new OTA Image)	
OTA2 Image Staging Area	New OTA2 Upgrade Image. Individual components may be compressed for size. (This is ***extracted*** to Current Application Area before being used).

NOTES: All major section sizes are known at compile time and will not change over the life of the product.

8 Adding OTA2 to a new platform:

8.1 All platform additions:

- If the platform has buttons, add these defines for button control of Factory Reset and starting the SoftAP (manual updating):

```
<Wiced-SDK>/platforms/$(PLATFORM)/platform.h

 /* Bootloader OTA and OTA2 factory reset during settings */
#define PLATFORM_FACTORY_RESET_BUTTON_GPIO      ( WICED_BUTTON_BACK )
#define PLATFORM_FACTORY_RESET_PRESSED_STATE    (   0  )
#define PLATFORM_FACTORY_RESET_CHECK_PERIOD     (  100 )
#define PLATFORM_FACTORY_RESET_TIMEOUT          ( 5000 )
#define PLATFORM_FACTORY_RESET_LED_GPIO         ( WICED_LED1 )
#define PLATFORM_FACTORY_RESET_LED_ON_STATE     (WICED_ACTIVE_HIGH )
```

- Add this function (copy from another platform is ok):

```
<Wiced-SDK>/platforms/$(PLATFORM)/platform.c,

uint32_t  platform_get_factory_reset_button_time ( uint32_t max_time )
```

- Create this include file (copy from another platform is ok to start):

```
<Wiced-SDK>/platforms/$(PLATFORM)/platform_ota2_image.h
```

- Add SFLASH layout defines:
    - The defines needed for laying out the External FLASH are located in this file.
    - You may need to adjust for the size of your FLASH, application, and Filesystem

```
<Wiced-SDK>/platforms/$(PLATFORM)/<B0 | B1>/platform_ota2_defines.mk
```

- Add new platform name to VALID_PLATFORMS in these makefiles:

apps/snip/ota2_example/ota2_example.mk
apps/snip/ota2_extract/ota2_extract.mk
apps/waf/ota2_bootloader/ota2_bootloader.mk
apps/waf/ota2_failsafe/ota2_failsafe.mk

- Add new Linker Files

This is specific per platform, you will need to determine RAM usage and adjust for OTA2 example:
WICED/platform/MCU/BCM4390x/BCM43907/GCC_ota2_app_without_rom_memory.ld
WICED/platform/MCU/BCM4390x/BCM43907/GCC_ota2_app_with_rom_memory.ld
WICED/platform/MCU/BCM4390x/BCM43907/GCC_ota2_bootloader_memory.ld
WICED/platform/MCU/BCM4390x/BCM43907/GCC_ota2_tiny_bootloader_memory.ld

8.2 Internal + External SFLASH

- You must define this in your Application make file:

```
GLOBAL_DEFINES      += CURRENT_APPLICATION_USES_INTERNAL_FLASH
```

- Add this to the platform's target.mk file in order to build the correct bootloader.

Add/adjust these parts of WICED/platform/MCU/<PLATFORM>/<PLATFORM>.mk

```
ifeq (1, $(OTA2_SUPPORT))
EXTERNAL_DCT := 1
endif

# DCT linker script
ifeq (1, $(OTA2_SUPPORT))
DCT_LINK_SCRIPT += $(TOOLCHAIN_NAME)/$(HOST_MCU_VARIANT)/ota2_dct$(LINK_SCRIPT_SUFFIX)
else
DCT_LINK_SCRIPT += $(TOOLCHAIN_NAME)/$(HOST_MCU_VARIANT)/dct$(LINK_SCRIPT_SUFFIX)
endif

################################################################################
# Building bootloader
################################################################################ifeq (1,
$(OTA2_SUPPORT))
NO_TINY_BOOTLOADER_REQUIRED:=1
BOOTLOADER_TARGET := waf.ota2_bootloader-NoOS-NoNS-$(PLATFORM)-$(BUS)
else
BOOTLOADER_TARGET := waf.bootloader-NoOS-NoNS-$(PLATFORM)-$(BUS)
endif

################################################################################
# Building standard application to run with bootloader
################################################################################
ifeq (1, $(OTA2_SUPPORT))
DEFAULT_LINK_SCRIPT := $(TOOLCHAIN_NAME)/ota2_app_with_bootloader$(LINK_SCRIPT_SUFFIX)
else
PRE_APP_BUILDS      += bootloader
DEFAULT_LINK_SCRIPT := $(TOOLCHAIN_NAME)/app_with_bootloader$(LINK_SCRIPT_SUFFIX)
GLOBAL_INCLUDES     += WAF ../../../../../apps/waf/bootloader/
endif
```

9 OTA2 Image Header

The first fields of the OTA2 Image Header (download status, bytes_received) are used during the download process. The Bootloader (or OTA2 Background Process) will update these fields as the download continues and completes.

OTA2 Image File Structure (file offset 0x0)
OTA2 Image File Header (OTA version, Application Version, Component count, etc.)

Component Header(s) (Component size, offset in OTA2 Image file, etc.)
Component(s)

OTA Image Header (file offset 0x00)		
OTA2 Image Version	16 bit	Version of the OTA2 Image header
Major Version	16 bit	Software Major Version, defined during build (add to make command, see below)
Minor Version	16 bit	Software Minor Version, defined during build (add to make command, see below)
Platform Name	32 Byte	String defining the Hardware
OTA2 Image download status	16 bit	When an OTA2 Update Image is built, this is always 0x00. When an OTA2 Factory Reset Image is built, this is always IMAGE_VALID. The download process updates the value during download.
OTA2 Image bytes_received	32 bit	Amount of data written to Staging Area during download. Compare to OTA2 Image Size to determine if download complete. NOTE: OTA2 Update Image Packaging Application sets this to 0x00. Download process updates during download. In OTA2 Factory Reset Images, this value matches image_size
Magic String	8 Byte	Further validates the Image (Currently "OTAImage")
OTA2 Image Header CRC	32 bit	CRC of OTA2 Header and Component Headers (excludes download status and bytes_received for OTA2 Update Images, OTA2 Factory Reset Images include these bytes in the CRC)
OTA2 Secure Signature Type	16 bits	SHA/CRC/other Type (TBD)

OTA2 Secure Signature Value	64 Byte	SHA/CRC/other (excluding this header) (256 bits)
OTA2 Image Size	32 bit	Size of entire OTA2 Image File (includes header)
OTA2 Image Component Count	16 bits	Number of Components in this OTA2 Image. Component Headers are stored as an array that directly follows this header.
Data Start	32 bits	Offset from start of Image to where the data starts (typically sector start after headers)

Component Header (One per component, as an array)		
Component Type	8 bits	Component Type (see `wiced_ota2_component_type_t`)
Compression	8 bits	Compression Type (see `wiced_ota2_compression_type_t`)
CRC	32 bits	CRC of Component (after decompression, if any)
Source Offset	32 bits	Offset from start of Component Data (In OTA2 Image File, after Component Headers)
Source Size	32 bits	Size of data in OTA2 Image File (compressed size, if compressed)
Destination	32 bits	Destination offset in SFLASH
Destination Size	32 bits	Destination Size (after decompression, if compressed)
Component Name	32 Byte	Component Name String

9.1 OTA2 Header Update Status during Download

The OTA2 Image will always be stored in FLASH.

9.1.1 FLASH Based Storage of download_status and bytes_received

The First section of the download will always be the OTA2 Header. In the header there are fields for download_status and bytes_received. When the OTA2 Update Image is constructed, these values will be 0x00, which indicates that the image is not downloaded (download_status = INVALID and bytes_received = 0). As data is received and written to the FLASH, these fields will be updated. The update sequence will be:

- Copy the sector to RAM (first sector of the download, which contains the OTA2 Header)
- Change the bytes_received and download_status in the RAM Copy
- Erase the sector
- Write the RAM Copy to the sector

If there is a power loss during the erase and/or writing the new values, the OTA2 Image will be considered invalid (the erased sector will not have the Magic String).
If there is a power loss during the re-writing step in the middle or at the end of the download, the bytes_received will not match the OTA2 image_size, and the OTA2 Image Header CRC will not match, and will be considered invalid.

10 OTA2 Image Packager Application

The OTA2 Image Packager Application will combine all of the assets into a linear file which has a length and CRC for easy retrieval via the HTTP protocol. This will include Executable Application, DCT, and File system (resources, WiFi Firmware, etc.). Adding “ota2_image” to the build line will automatically create the OTA2 Update Image configuration file, and call the tool to build an OTA2 Image file for placing on a server for download. The OTA2 Image Packager Application is written in C, and works in concert with the Wiced build environment.

10.1Build Examples

Make Target examples:

Build an OTA2 Update Image suitable for upgrade server:
```
<application>-<platform> ota2_image
```

Build an OTA2 Update Image suitable for upgrade server + download to FLASH OTA2 Staging Area at end of build for testing purposes (using sflash_write as part of build process):
```
<application>-<platform> ota2_download
```

Build AN OTA2 Factory Reset Image suitable for manufacturing FLASHing of the device:
```
<application>-<platform> ota2_factory_image
```

Build AN OTA2 Factory Reset Image + download to FLASH OTA2 Factory Reset Area at end of build (using sflash_write as part of build process):
```
<application>-<platform> ota2_factory_download
```

10.2Defining Application Version for OTA2

Adding these arguments to the build line will set the Software versions in the OTA2 Image header. Leaving them off the build line will have default values of 0.

```
APP_VERSION_FOR_OTA2_MAJOR=X
APP_VERSION_FOR_OTA2_MINOR=Y
```

For an initial product release:

```
<application>-<platform> ota2_factory_download download download_apps run APP_VERSION_FOR_OTA2_MAJOR=1
APP_VERSION_FOR_OTA2_MINOR=0
```

For an update OTA2 Image suitable for placing on an update server:

```
<application>-<platform> ota2_image APP_VERSION_FOR_OTA2_MAJOR=1 APP_VERSION_FOR_OTA2_MINOR=2
```

10.3Checking OTA2 versions during an update

We save the original build major and minor version values in the application DCT (see snip/ota2_example/ota2_test_dct.c, .h):

```
uint16_t        ota2_major_version;         /* define APP_VERSION_FOR_OTA2_MAJOR as make arg */
uint16_t        ota2_minor_version;         /* define APP_VERSION_FOR_OTA2_MINOR as make arg */
```

Set the compile flag for testing the update version versus the current version (see snip/ota2_example/ota2_example.mk):

```
# undefine to skip check for Version checking of update while developing updates
# define (and provide version #) for production version testing
# You can provide version # during compilation (default is 0) by adding to build compile args
# <application>-<platform> ota2_image APP_VERSION_FOR_OTA2_MAJOR=x APP_VERSION_FOR_OTA2_MINOR=y
#CHECK_OTA2_UPDATE_VERSION := 1
```

We can then test the version during an upgrate (see snip/ota2_example/ota2_test.c in the ota2_callback function):

```
    OTA2_APP_PRINT(OTA2_LOG_NOTIFY, ("Current Version %d.%d\r\n", player->dct_app->ota2_major_version,
                                                                  player->dct_app->ota2_minor_version));
    OTA2_APP_PRINT(OTA2_LOG_NOTIFY, ("   OTA2 Version %d.%d\r\n", ota2_header->major_version,
                                                                  ota2_header->minor_version));

#ifdef (CHECK_OTA2_UPDATE_VERSION)
        if ((player->dct_app->ota2_major_version > ota2_header->major_version) ||
            ((player->dct_app->ota2_major_version == ota2_header->major_version) &&
             (player->dct_app->ota2_major_version > ota2_header->minor_version)) )
        {
            OTA2_APP_PRINT(OTA2_LOG_NOTIFY, ("OTA2 Update Version Fail - return ERROR, do not
update!\r\n"));
            return WICED_ERROR;
        }
```

```
#endif
```

After an update (or Factory Reset) we update the application DCT with the new version # (see snip/ota2_example/ota2_test.c:: over_the_air_2_app_restore_settings_after_update()):

```
    /* update version number based on boot type */
    switch (boot_type)
    {
        case OTA2_BOOT_NEVER_RUN_BEFORE:
        case OTA2_BOOT_NORMAL:
        case OTA2_BOOT_LAST_KNOWN_GOOD: /* unsupported */
            break;
        case OTA2_BOOT_FACTORY_RESET:
            if (wiced_ota2_image_get_version( WICED_OTA2_IMAGE_TYPE_FACTORY_RESET_APP, &major, &minor) ==
WICED_SUCCESS)
            {
                player->dct_app->ota2_major_version = major;
                player->dct_app->ota2_minor_version = minor;
            }
            break;
        case OTA2_SOFTAP_UPDATE:
        case OTA2_BOOT_UPDATE:
            if (wiced_ota2_image_get_version( WICED_OTA2_IMAGE_TYPE_STAGED, &major, &minor) ==
WICED_SUCCESS)
            {
                player->dct_app->ota2_major_version = major;
                player->dct_app->ota2_minor_version = minor;
            }
            break;

    }
```

11 OTA2 Image Extraction Library API

The OTA2 Extraction Library has functions to write data to the download staging area, check download status, verify, and extract downloaded OTA2 Images.

```
/*****************************************************
 *                    Constants
 *****************************************************/

#define WICED_OTA2_IMAGE_VERSION             0x01

#define WICED_OTA2_PLATFORM_NAME_LEN        32

#define WICED_OTA2_IMAGE_MAGIC_STRING        "OTAimage"
#define WICED_OTA2_IMAGE_MAGIC_STR_LEN       8

#define WICED_OTA2_IMAGE_COMPONENT_NAME_LEN  32
#define WICED_OTA2_IMAGE_SECURE_SIGN_LEN     64


/* this is the same as the SFLASH sector size */
#ifndef SECTOR_SIZE
#define SECTOR_SIZE                     (4096)
#endif

/* let's get the FLASH base address - These are system addresses, not offsets! */
#define OTA_FLASH_CHIP_BASE              SI_SFLASH

/*****************************************************
 *                   Enumerations
 *****************************************************/
```

```
typedef enum
{
    OTA2_BOOT_NEVER_RUN_BEFORE = 0,
    OTA2_BOOT_NORMAL,
    OTA2_BOOT_FACTORY_RESET,
    OTA2_BOOT_UPDATE,
    OTA2_SOFTAP_UPDATE,
    OTA2_BOOT_LAST_KNOWN_GOOD,
} ota2_boot_type_t;


typedef enum
{
    WICED_OTA2_IMAGE_TYPE_NONE  = 0,
    WICED_OTA2_IMAGE_TYPE_FACTORY_RESET_APP,
    WICED_OTA2_IMAGE_TYPE_CURRENT_APP,
    WICED_OTA2_IMAGE_TYPE_LAST_KNOWN_GOOD,
    WICED_OTA2_IMAGE_TYPE_STAGED

}wiced_ota2_image_type_t;

typedef enum {
    WICED_OTA2_IMAGE_INVALID             = 0,
    WICED_OTA2_IMAGE_DOWNLOAD_IN_PROGRESS,
    WICED_OTA2_IMAGE_DOWNLOAD_FAILED,
    WICED_OTA2_IMAGE_DOWNLOAD_UNSUPPORTED,
    WICED_OTA2_IMAGE_DOWNLOAD_COMPLETE,
    WICED_OTA2_IMAGE_VALID,
    WICED_OTA2_IMAGE_EXTRACT_ON_NEXT_BOOT,
    WICED_OTA2_IMAGE_DOWNLOAD_EXTRACTED,
} wiced_ota2_image_status_t;

typedef enum
{
    WICED_OTA2_IMAGE_SWAP_HOST_TO_NETWORK = 0,
    WICED_OTA2_IMAGE_SWAP_NETWORK_TO_HOST,
} ota2_image_swap_type_t;

typedef enum {
    WICED_OTA2_IMAGE_SIGN_NONE       = 0,
    WICED_OTA2_IMAGE_SIGN_CRC,
    WICED_OTA2_IMAGE_SIGN_SHA,

} wiced_ota2_image_sign_type_t;

typedef enum {
    WICED_OTA2_IMAGE_COMPONENT_LUT       = 0,
    WICED_OTA2_IMAGE_COMPONENT_FR_APP,
    WICED_OTA2_IMAGE_COMPONENT_DCT,
    WICED_OTA2_IMAGE_COMPONENT_OTA_APP,
    WICED_OTA2_IMAGE_COMPONENT_FILESYSTEM,
    WICED_OTA2_IMAGE_COMPONENT_WIFI_FW,
    WICED_OTA2_IMAGE_COMPONENT_APP0,
    WICED_OTA2_IMAGE_COMPONENT_APP1,
    WICED_OTA2_IMAGE_COMPONENT_APP2
} wiced_ota2_image_component_type_t;

typedef enum {
    WICED_OTA2_IMAGE_COMPONENT_COMPRESSION_NONE       = 0,
    WICED_OTA2_IMAGE_COMPONENT_COMPRESSION_LZW,
    WICED_OTA2_IMAGE_COMPONENT_COMPRESSION_GZIP,
    WICED_OTA2_IMAGE_COMPONENT_COMPRESSION_BZ2,
} wiced_ota2_image_component_compress_t;

/******************************************************
 *                Type Definitions
 ******************************************************/

/******************************************************
 *                    Structures
 ******************************************************/
```

```
#pragma pack(1)

typedef struct wiced_ota2_component_s {
        uint8_t         type;               /* wiced_ota2_image_component_type_t
*/
        uint8_t         compression;        /* wiced_ota2_image_component_compress_t
*/
        OTA2_CRC_VAR    crc;                /* crc on uncompressed component data
*/
        uint32_t        source_offset;      /* offset within OTA Image Component Data section
*/
        uint32_t        source_size;        /* size of data in OTA Image
*/
        uint32_t        destination;        /* absolute offset of destination in FLASH        */
        uint32_t        destination_size;   /* size of data
*/
        uint8_t         name[WICED_OTA2_IMAGE_COMPONENT_NAME_LEN];   /* component name
*/
} wiced_ota2_image_component_t;

typedef struct wiced_ota2_header_s {
        uint16_t        ota2_version;               /* OTA2 Image Version  (version of this
format)                         */
        uint16_t        major_version;              /* Software Version Major (version of
software contained in image)      */
        uint16_t        minor_version;              /* Software Version Minor (version of
software contained in image)      */
        uint8_t         platform_name[WICED_OTA2_PLATFORM_NAME_LEN];    /* Platform name (31 char
+ NULL)                   */
        uint16_t        download_status;            /* Status of image download
*/
        uint32_t        bytes_received;             /* bytes received (valid for
WICED_OTA2_DOWNLOAD_IN_PROGRESS and

WICED_OTA2_DOWNLOAD_COMPLETE)                   */
        uint8_t         magic_string[WICED_OTA2_IMAGE_MAGIC_STR_LEN];    /* Magic string
"OTAImage"                        */

        OTA2_CRC_VAR    header_crc;                 /* CRC of OTA header and component headers,
                                                    excluding header_crc, download_status and
bytes_received                */
        uint16_t        secure_sign_type;           /* Secure signature type
*/
        uint8_t         secure_signature[WICED_OTA2_IMAGE_SECURE_SIGN_LEN]; /* depends on
secure_sign_type  up to 256 bit   */

        uint32_t        image_size;                 /* total size of OTA image (including
headers)                        */

        uint16_t        component_count;            /* number of components in the component list
                                                   (component list directly follows this
structure)                       */
        uint32_t        data_start;                 /* offset in this file to start of data
*/
} wiced_ota2_image_header_t;

#pragma pack()

/*****************************************************
 *                  Global Variables
 *****************************************************/

/*****************************************************
 *               Function Declarations
 *****************************************************/
static inline void wiced_ota2_image_header_swap_network_order(wiced_ota2_image_header_t
*ota2_header,
                                                              ota2_image_swap_type_t
host_to_network )
{
```

```
    if (host_to_network == WICED_OTA2_IMAGE_SWAP_HOST_TO_NETWORK)
    {
        /* convert 16 & 32 bit values to network order */
        ota2_header->ota2_version     = htons(ota2_header->ota2_version);
        ota2_header->major_version    = htons(ota2_header->major_version);
        ota2_header->minor_version    = htons(ota2_header->minor_version);
        ota2_header->download_status  = htons(ota2_header->download_status);
        ota2_header->bytes_received   = htonl(ota2_header->bytes_received);
        ota2_header->header_crc       = htonl(ota2_header->header_crc);
        ota2_header->secure_sign_type = htons(ota2_header->secure_sign_type);
        ota2_header->image_size       = htonl(ota2_header->image_size);
        ota2_header->component_count  = htons(ota2_header->component_count);
        ota2_header->data_start       = htonl(ota2_header->data_start);
    }
    else
    {
        /* convert 16 & 32 bit values to host order */
        ota2_header->ota2_version     = ntohs(ota2_header->ota2_version);
        ota2_header->major_version    = ntohs(ota2_header->major_version);
        ota2_header->minor_version    = ntohs(ota2_header->minor_version);
        ota2_header->download_status  = ntohs(ota2_header->download_status);
        ota2_header->bytes_received   = ntohl(ota2_header->bytes_received);
        ota2_header->header_crc       = ntohl(ota2_header->header_crc);
        ota2_header->secure_sign_type = ntohs(ota2_header->secure_sign_type);
        ota2_header->image_size       = ntohl(ota2_header->image_size);
        ota2_header->component_count  = ntohs(ota2_header->component_count);
        ota2_header->data_start       = ntohl(ota2_header->data_start);
    }
}
static inline void
wiced_ota2_image_component_header_swap_network_order(wiced_ota2_image_component_t
*component_header,
                                                     ota2_image_swap_type_t
host_to_network )
{
    if (host_to_network == WICED_OTA2_IMAGE_SWAP_HOST_TO_NETWORK)
    {
        /* convert 16 & 32 bit values to network order */
        component_header->crc              = OTA2_CRC_HTON(component_header->crc);
        component_header->source_offset    = htonl(component_header->source_offset);
        component_header->source_size      = htonl(component_header->source_size);
        component_header->destination      = htonl(component_header->destination);
        component_header->destination_size = htonl(component_header->destination_size);

    }
    else
    {
        /* convert 16 & 32 bit values to host order */
        component_header->crc              = OTA2_CRC_NTOH(component_header->crc);
        component_header->source_offset    = ntohl(component_header->source_offset);
        component_header->source_size      = ntohl(component_header->source_size);
        component_header->destination      = ntohl(component_header->destination);
        component_header->destination_size = ntohl(component_header->destination_size);
    }
}

/**
 * Simple validation of the OTA Image
 *
 * Checks header version, magic string, size, # components
 *
 * @param[in]  ota_type     - OTA Image type
 *
 * @return WICED_SUCCESS
 *         WICED_ERROR      - Bad OTA Image
 *         WICED_BADARG     - NULL pointer passed in or bad size
 */
wiced_result_t wiced_ota2_image_validate ( wiced_ota2_image_type_t ota_type );

/**
 * Get status of OTA Image at download location
```

```
 *
 * @param[in]  ota_type     - OTA Image type
 * @param[out] status       - Receives the OTA Image status.
 *
 * @return WICED_SUCCESS
 *         WICED_ERROR      - Bad OTA Image
 *         WICED_BADARG     - NULL pointer passed in or bad size
 */
wiced_result_t wiced_ota2_image_get_status ( wiced_ota2_image_type_t ota_type,
wiced_ota2_image_status_t *status );

/**
 * Extract OTA2 extractor (or Apps LUT) from the OTA2 Image to the current area
 *
 * NOTE: This is used by the OTA2 Failsafe code
 *
 * @param[in]  ota_type     - OTA Image type
 * @param[in]  component    - OTA Component to extract
 *                            (ONLY WICED_OTA2_IMAGE_COMPONENT_LUT or WICED_OTA2_IMAGE_COMPONENT_OTA_APP)
 * @param[out] destination  - address extracted to in FLASH
 * @param[out] destination  - extracted size
 *
 * @return WICED_SUCCESS
 *         WICED_ERROR      - Bad OTA Image, not fully downloaded
 *         WICED_BADARG     - NULL pointer passed in or bad size
 */
wiced_result_t wiced_ota2_image_extract_uncompressed_component( wiced_ota2_image_type_t ota_type,
                                        wiced_ota2_image_component_type_t component,
                                        uint32_t* destination, uint32_t* destination_size );
/**
 * Extract OTA Image to the current area
 * NOTE: All information regarding destination of data in the system is part of the OTA Image.
 *
 * @param[in]  ota_type     - OTA Image type
 * @param[in]  image_size   - Size of the OTA Image
 *
 * @return WICED_SUCCESS
 *         WICED_ERROR      - Bad OTA Image, not fully downloaded
 *         WICED_BADARG     - NULL pointer passed in or bad size
 */
wiced_result_t wiced_ota2_image_extract ( wiced_ota2_image_type_t ota_type );

/**
 * Write OTA Image to the Staging area (WICED_OTA2_IMAGE_TYPE_STAGED)
 * NOTE: The total size of the OTA image is included in a valid OTA image header.
 *       This function will update the status in the OTA image header by calling
 *       wiced_ota2_update_header() TODO: make this platform-specific
 *
 * @param[in]  data     - pointer to part or all of an OTA image to be stored in the staging
area
 * @param[in]  offset   - offset from start of staging area to store this data
 * @param[in]  size     - size of the data to store
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG
 */
wiced_result_t  wiced_ota2_image_write_data(uint8_t* data, uint32_t offset, uint32_t size);


/** Get the OTA2 application software version from the header
 *
 * NOTE: Only updates the major & minor values on WICED_SUCCESS
 *
 * @param image_type [in]   : OTA2 Image type to get software app version
 * @param major [out]       : ptr to store software major version
 * @param minor [out]       : ptr to store software minor version
 *
 * @return  WICED_SUCCESS
 *          WICED_BADARG
```

```
 *          WICED_ERROR
 */
wiced_result_t wiced_ota2_image_get_version(wiced_ota2_image_type_t image_type, uint16_t* major, uint16_t*
minor)

/** Update the OTA image header after writing (parts of) the downloaded OTA image to FLASH  TODO:
make this platform-specific
 *
 * @param delta_written - number of bytes written to the image
 *
 * @return  WICED_SUCCESS
 *          WICED_BADARG
 *          WICED_ERROR
 */
wiced_result_t wiced_ota2_image_update_staged_header(uint32_t delta_written);

/** Update the OTA image header status
 *
 * @param new status - one of wiced ota2 image status t
 *
 * @return  WICED_SUCCESS
 *          WICED_BADARG
 *          WICED_ERROR
 */
wiced_result_t wiced_ota2_image_update_staged_status(wiced_ota2_image_status_t new_status);

/** Call this to set the flag to force a facory reset on reboot
 *  NOTE: This is equal to holding the "factory reset button" for 5 seconds.
 *          Use this if there is no button on the system.
 *
 * @param   N/A
 *
 * @return  WICED_SUCCESS
 *          WICED_ERROR
 */
wiced_result_t wiced_ota2_force_factory_reset_on_reboot( void );

/** Get the last boot type - did we update or have a factory reset?
 *
 * @param   N/A
 *
 * @return  ota2_boot_type_t
 */
ota2_boot_type_t wiced_ota2_get_boot_type( void );

/* debugging only */
wiced_result_t wiced_ota2_image_fakery(wiced_ota2_image_status_t new_status);
```

12 OTA2 Update Service Library API

The OTA2 Background Update Library will automatically download and update the product based on pre-determined or user modified (using the SoftAP OTA2 Web Server capability) settings. The use of this library is determined by the Application Developer, and is included in the Application (not the Bootloader). This code is partially written, not tested.

- OTA2 Service will be the Client and will connect to your OTA2 Upgrade Server.
- Saves package to the Flash.
    - After first chunk written
        - sets download_status to “Download in progress”.
        - Runs validation check. Will stop download if OTA2 header is not valid.
    - After each chunk, sets bytes_received value.
- Upon Completion

- Sets download_status to “Download complete”.
- If User / Application has set Automatic Update flag
  - Sets download_status to “Extract on Reboot”.
  - Upon reboot, Bootloader will extract the OTA2 Image
- Provides a callback so that Application can take over the download process.

```
/*****************************************************
 *                    Enumerations
 *****************************************************/
typedef enum
{
    OTA2_SERVICE_NOMINAL = 0,       /* Used internally, does not produce a callback        */

    OTA2_SERVICE_STARTED,           /* Background timer thread has started
                                     * return - None - informational                       */

    OTA2_SERVICE_AP_CONNECT_ERROR,  /* Background timer thread failed to connect to supplied OTA2 AP
                                     * return - None - informational                       */

    OTA2_SERVICE_SERVER_CONNECT_ERROR,  /* Background timer thread failed to TCP connect to update server
                                         * return - None - informational                   */

    OTA2_SERVICE_AP_CONNECTED,      /* Background timer thread connected to OTA2 AP
                                       * return - None - informational                    */
    OTA2_SERVICE_SERVER_CONNECTED,  /* Background timer thread TCP connected to update server
                                       * return - None - informational                    */

    OTA2_SERVICE_CHECK_FOR_UPDATE,  /* Time to check for updates.
                                     * return - WICED_SUCCESS = Service will check for update availability
                                     *        - WICED_ERROR   = Application will check for update
                                     *                                                availability   */

    OTA2_SERVICE_UPDATE_AVAILABLE,  /* Service has contacted server, update is available
                                     * value   - pointer to the wiced_ota2_image_header_t struct
                                     *           from the file on the server.
                                     *
                                     * return - WICED_SUCCESS = Application indicating that it wants the
                                     *                          OTA Service to perform the download
                                     *        - WICED_ERROR   = Application indicating that it will
                                     *                          perform the download, the OTA Service
                                     *                          will do nothing.
                                     *                          If Application is going to ignore the
                                     *                          update, return WICED_ERROR */

    OTA2_SERVICE_DOWNLOAD_STATUS,   /* Download status - value has % complete (0-100)
                                     *   NOTE: This will only occur when Service is performing download
                                     * return - WICED_SUCCESS = Service will continue download
                                     *        - WICED_ERROR   = Service will STOP download and service
                                     *                          will issue OTA2_SERVICE_UPDATE_ERROR */

    OTA2_SERVICE_PERFORM_UPDATE,    /* Download is complete
                                     * return - WICED_SUCCESS = Service will inform Bootloader to extract
                                     *                          and update on next power cycle
                                     *        - WICED_ERROR   = Service will inform Bootloader that
                                     *                          download is complete
                                     *                          - Bootloader will NOT extract       */

    OTA2_SERVICE_UPDATE_ERROR,      /* There was an error in transmission
                                     * This will only occur if Error during Service performing data
                                     * transfer.
                                     * return - WICED_SUCCESS = Service will retry using retry_timer
                                     *        - WICED_ERROR   = Service will retry on next check_interval
                                     *           Application can call
                                     *           wiced_ota2_service_check_for_updates()
                                     *           to run another check earlier                 */

    OTA2_SERVICE_UPDATE_ENDED,     /* All update actions for this check are complete.
```

```
                                 * This callback is to allow the application to take any actions when
                                 * the service is done checking / downloading an update
                                 * (succesful, unavailable, or error)
                                 * return - None  - informational                        */

    OTA2_SERVICE_STOPPED,       /* Background service has stopped
                                 * return - None - informational                        */
} wiced_ota2_service_status_t;

/*****************************************************
 *               Callback Function Definition
 *****************************************************/

/**
 *  Application callback for OTA service
 *  NOTE: This callback is called rather than the
 *          default checking for an update. Return value tells
 *          service how to handle the notification, or if the
 *          Application will handle the downloads - see .
 *
 * @param[in]  session - value returned from wiced_ota2_service_init()
 * @param[in]  status  - current status of service (wiced_ota2_service_status_t)
 * @param[in]  value   - value associated with status
 * @param[in]  opaque  - user supplied opaque pointer
 *
 * @return - WICED_SUCCESS  - Service will perform default action
 *           WICED_ERROR    - Application will perform action
 */
typedef wiced_result_t (*ota2_service_callback)(void* session_id,
                                                wiced_ota2_service_status_t status, int value,
                                                void* opaque );

/*****************************************************
 *                      Structures
 *****************************************************/

typedef struct
{
    char*           host_name;              /* host to get updates from                     */
    char*           file_path;              /* filename to get                              */
    uint16_t        port;                   /* port on host machine (default: HTTP_PORT = 80) */

    uint32_t        initial_check_interval; /* seconds before first update check            */
    uint32_t        check_interval;         /* seconds between checks                       */
    uint32_t        retry_check_interval;   /* seconds between re-try if initial contact to
                                             * server for update info fails
                                             * 0 = wait until next check_interval           */

    uint8_t         auto_update;            /* Callback return value over-rides this parameter
                                             * Auto-update behavior if no callback registered.
                                             *   1 = Service will inform Bootloader to extract
                                             *       and update on next power cycle after download
                                             *   0 = Service will inform Bootloader that download
                                             *       is complete - Bootloader will NOT extract/update
                                             *       until user / application requests        */

    wiced_config_ap_entry_t*    ota2_ap_info;       /* Alternate AP to use to connect to the
                                                     *   OTA2 update server
                                                     * - This is optional. If the default AP has access
                                                     *   to the OTA2 update server, this can be NULL
                                                     * - Use this or ota2_ap_list, not both.
                                                     *   The list over-rides this.
                                                     */

    wiced_config_ap_entry_t*    default_ap_info;    /* Default AP to connect to after the OTA2 update
                                                     *   is complete
                                                     * This is optional. If the default AP has access
                                                     * to the OTA2 update server, this will be NULL
                                                     * - If the application needs a special access point
                                                     *   connection, and does not wish the OTA2 code
```

```
                                                *   re-connect, set this to NULL.
                                                */
    uint8_t                     ota2_ap_list_count; /* Number of APs in the ota2_ap_list */
    wiced_config_ap_entry_t*    ota2_ap_list;       /* Alternate AP list to use to connect to the
                                                *   OTA2 update server
                                                * - This is optional, this can be NULL
                                                * - Use this or ota2_ap_info, not both.
                                                *   This over-rides ota2_ap_info.
                                                */
} wiced_ota2_backround_service_params_t;

/******************************************************
 *               Variables Definitions
 ******************************************************/

/******************************************************
 *               Function Definitions
 ******************************************************/

/**
 * Initialize a timed backgound service to check for updates
 *
 * @param[in]  params - ptr to wiced_ota2_backround_service_params_t structure
 * @param[in]  opaque - application value passed to application in callback
 *
 * @return - session pointer
 *           NULL indicates error
 */
void*  wiced_ota2_service_init(wiced_ota2_backround_service_params_t *params, void* opaque);

/**
 * De-initialize the service
 *
 * @param[in]  session_id - value returned from wiced_ota2_service_init()
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG
 */
wiced_result_t  wiced_ota2_service_deinit(void* session_id);

/**
 * Start the service
 *
 * NOTE: This is a non-blocking call (process is async)
 * NOTE: Register callbacks *before* calling start
 *
 * @param[in]  session_id - value returned from wiced_ota2_service_init()
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG
 */
wiced_result_t  wiced_ota2_service_start(void* session_id);

/**
 * Stop the service
 *
 * @param[in]  session_id - value returned from wiced_ota2_service_init()
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG

 */
wiced_result_t  wiced_ota2_service_stop(void* session_id);

/**
 * Register or Un-register a callback function to handle the actual update check
 *
 * @param[in]  session_id  - value returned from wiced_ota2_service_init()
```

```
 * @param[in]  callback - callback function pointer (NULL to disable)
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG

 */
wiced_result_t  wiced_ota2_service_register_callback(void* session_id, ota2_service_callback
update_callback);

/**
 * Force an update check now
 * NOTE: This is a separate call from the scheduled update checking
 *      If a scheduled update is in progress, this call will fail
 *      If a scheduled update is waiting (not active), this call will
 *             - pause the update scheduler
 *             - do the update call (will appropriate callbacks to the application)
 *             - un-pause the update scheduler
 *
 * This will:
 *  - check for an update
 *  - If update is available
 *      if callback registered, will callback app
 *      if (no callback) or (callback returns WICED_SUCCESS)
 *         - download update
 *
 * @param[in]  session_id - value returned from wiced_ota2_service_init()
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG
 */
wiced_result_t  wiced_ota2_service_check_for_updates(void* session_id);

/**
 * Split a URI into host and file_path parts
 *
 * @param[in]  uri           - the URI of the file desired
 * @param[in]  host_buff     - pointer to where the host part of the URI will be stored
 * @param[in]  host_buff_len - length of host_buff
 * @param[in]  path_buff     - pointer to where the path part of the URI will be stored
 * @param[in]  path_buff_len - length of path_buff
 * @param[in]  port          - pointer to store the port number
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG
*/
wiced_result_t wiced_ota2_service_uri_split(const char* uri, char* host_buff, uint16_t host_buff_len,
char* path_buff, uint16_t path_buff_len, uint16_t* port);

/** Set Debug Logging level
 *
 * @param[in]  session_id - value returned from wiced_ota2_service_init()
 * @param[in]  level      - debug level
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG
 */
wiced_result_t wiced_ota2_service_set_debug_log_level(void* session_id, wiced_ota2_log_level_t
new_log_level);

/** Output status to console
 *
 * @param[in]  session_id - value returned from wiced_ota2_service_init()
 *
 * @return - WICED_SUCCESS
 *           WICED_ERROR
 *           WICED_BADARG
```

```
 */
wiced_result_t wiced_ota2_service_status(void* session_id);
```

13 OTA2 Save and Restore DCT API

The OTA2 Save and Restore DCT functions allow the application to save the current DCT settings to a saved copy in the FLASH. Upon boot of the Application, the application can determine if an extraction just occurred, and can then restore the previous user settings.

See snip/ota2_example/ota2_test.c.

```
/**
 *  Used by OTA2, lives in wiced_dct_external_ota2.c  and  wiced_dct_internal_ota2.c
 *
 * Checks if saved DCT is from a FACTORY_RESET or UPDATE. If so, Don't copy over.
 * This is so that if we get multiple resets during an extract, we don't trash the user settings that
 * were initially saved on the first extraction.
 *
 * NOTE: Application MUST save to saved DCT copy with boot type as "OTA2_BOOT_NORMAL" after extraction for
this to work.
 *       see snip/ota2_example/ota2_test.c
 *
 *       // Restore application settings from the before the update/factory reset
 *       over_the_air_2_app_restore_settings_after_update(app_context); // NOTE: this is application
specific
 *
 *       //Set the reboot type back to normal so we don't think we updated on the next reboot
 *       wiced_dct_ota2_save_copy( OTA2_BOOT_NORMAL );
 *
 *  @param      boot_type: Type of OTA2 boot we want to mark in the saved area
 *
 *  return:     WICED_SUCCESS
 *              WICED_ERROR
 *
 */
wiced_result_t wiced_dct_ota2_save_copy         ( uint8_t boot_type );

/**
 *  Used by OTA2, lives in wiced_dct_external_ota2.c  and  wiced_dct_internal_ota2.c
 *
 *  Read data from the saved area DCT - saved last time wiced_dct_ota2_save_copy() was called,
 *  wiced_dct_ota2_save_copy() may have been called by the application OR just before an OTA2 extraction.
 *
 *  Behaves the same as wiced_dct_read_with_copy()
 *
 *  @param [out] info_ptr : destination for the data read
 *  @param       section  : section of the DCT to read
 *  @param       offset   : offset within the section
 *  @param       size     : size of data to read
 *
 *  @return     WICED_SUCCESS
 *              WICED_ERROR
 */
wiced_result_t wiced_dct_ota2_read_saved_copy (void* info_ptr, dct_section_t section, uint32_t offset,
                                               uint32_t size);

/**
 * Used by OTA2, lives in wiced_dct_external_common.c  and  wiced_dct_internal_common.c
 *
 * Erase the destination DCT area, copy the current DCT to the area directly (exact copy).
 *
```

```
 * @param   dst_dct : Destination location for DCT (for OTA2 saved area, use
OTA2_IMAGE_APP_DCT_SAVE_AREA_BASE)
 *
 * @return  WICED_SUCCESS
 *          WICED_ERROR
 *
 */
wiced_result_t wiced_dct_ota2_erase_save_area_and_copy_dct( uint32_t dct_dct );
```

14 Snip.ota2_example Application

14.1 <Wiced-SDK>/apps/snip/ota2_example

- Uses new layout for FLASH (see <Wiced-SDK>/platform/<platform>/ota2_image_defines.mk)
  - adjustments were made to include this file in these make files:
    - <Wiced-SDK>/tools/makefiles/wiced_apps.mk
    - <Wiced-SDK>/tools/makefiles/wiced_config.mk
    - <Wiced-SDK>/tools/makefiles/wiced_elf.mk
    - Platform-specific _common.mk and _targets.mk
- Uses snip/ota2_extract, waf/ota2_bootloader and waf/ota2_failsafe
- Shows how to extract Factory Reset or Staged Update independent of the code in ota2_extract
- Shows calls to determine boot type (normal, factory reset, update)
- Some code in place (not tested) for :
  - Checking Battery level before starting extraction
  - Using a different CRC function
- Example of how to download an image using **`wiced_ota2_service_check_for_updates`**().
- Example of how to extract an image from the Factory Reset Area
- Example of how to extract an image from the Update Staging Area
- Functions for saving / reading Application Save DCT area to keep data after an upgrade
- Shows code to restore user settings after update or factory reset
  - **`wiced_dct_ota2_read_saved_copy()`**
- Shows how to use the wiced_ota2_service API to download a file.
- Shows how to use the wiced_ota2_service API to set up a timed/recurring download of a file.

NOTE: For Internal FLASH models (code executes from internal FLASH), extracting images directly from snip.ota2_example will not work, as the program would be erasing and re-writing the code it is executing from. For these devices, extraction is only available from the ota2_extract application.

14.2 How to build snip.ota2_example

Normal OTA2 example program build:

Starting with SDK-3.7.0, you must build the snip/ota_extract application (includes the SoftAP functionality). This only needs to be done once (or after `./make clean`):

```
snip.ota2_extract-<platform_name>
```

Creates the ota2_bootloader and all the components, and the application.

```
snip.ota2_example-<platform_name> ota2_image download run
```

Build a different application OTA2_image_file.bin suitable for upgrade server:

Add these lines to the <application>.mk file before building so that the ota2_extract application is included in the OTA2_image_file.bin. This is very important so that when the update occurs, the ota2_extract application is available for the next update (and Factory Reset).

```
#OTA SoftAp application
OTA_APPLICATION:= snip.ota2_extract-$(PLATFORM)

OTA_APP     := build/$(OTA_APPLICATION)/binary/$(OTA_APPLICATION).stripped.elf
```

Build the application to create the OTA2_image_file.bin:

```
snip.mini_printf_test-<platform_name> ota2_image download run
```

- This does NOT add any OTA2 support into the Application!
- This builds the application with different FLASH addressing from a normal (non-OTA2 supported) build.
- Adding the OTA_APP directive to the <application>.mk file directs the build to add the snip/ota2_extract application to the OTA2_image_file.bin.

OTA2 Image suitable for upgrade server + download to FLASH OTA2 Staging Area at end of build:

```
snip.ota2_example-<platform_name> ota2_download APP_VERSION_FOR_OTA2_MAJOR=1
APP_VERSION_FOR_OTA2_MINOR=3
```

Build OTA2 Factory Reset Image + download to FLASH OTA2 Factory Reset Area at end of build:

```
snip.ota2_example-<platform_name> ota2_factory_download
```

14.3 snip.ota2_example console commands

factory_status	Shows status of OTA2 Image in OTA2 Factory Reset Area
factory_now	Extracts OTA2 Image in OTA2 Factory Reset Area (for debugging purposes) • Prints message when done - need to reboot to see changes • factory_status will always show OTA2 Image in Factory Reset Area as "valid" • Holding the designated Factory Reset Button for 10 seconds will do this during a reboot • This command will fail on an Internal-Flash system
update_status	Shows update status of OTA2 Image in OTA2 Staging Area
update_now	Fakes the download status of a valid OTA2 Image in the Staging Area (when downloaded by the IDE at compile time) as COMPLETE and sets bytes received. • Extracts the OTA2 Image in OTA2 Staging Area right NOW. • Prints a message when done extracting: "Reboot to see updated program"

- update_status will now show OTA2 Image in Staging Area as "extracted"
- This command will fail on an Internal-Flash system

update_reboot — Fakes download status as EXTRACT_ON_REBOOT and sets bytes received of OTA2 Image in OTA2 Staging Area

- update_status will now show OTA2 Image in Staging Area as "extract on reboot"
- Reboot to see the extraction happen after reboot.
- After reboot/extraction, update_status shows OTA2 Image in Staging Area as "extracted".

get_update <host></file> — This will connect to the OTA2 AP (if defined) or use the current AP to find the <host>, and will try to download the <file>

- Depending on how the background service was initialized, see `wiced_ota2_backround_service_params_t`
    - update_status may show OTA2 Image in Staging Area as "extract on reboot" or “completed download”

timed_update <host></file> — This will connect to the OTA2 AP (if defined) or use the current AP to find the <host>, and will try to download the <file>

- Timing of when this happens is determined by the initialization of the background service. See `wiced_ota2_backround_service_params_t`
- Depending on how the background service was initialized:
    - update_status may show OTA2 Image in Staging Area as "extract on reboot" or “completed download”

stop_update — This will stop the timed update scheduler and exit the scheduler thread

15 Testing an OTA2 update and Factory Reset

Build the scan application for showing that the upgrade has changed the current application.

Starting with SDK-3.7.0, you must build the ota_extract application (includes the SoftAP functionality). This only needs to be done once (or after `./make clean`):

```
snip.ota2_extract-<platform_name>
```

Add these lines to apps/snip/scan.mk file before building so that the ota2_extract application is included in the OTA2_image_file.bin. This is very important so that when the update occurs, the ota2_extract application is available for the next update (and Factory Reset).

```
#OTA SoftAp application
OTA_APPLICATION:= snip.ota2_extract-$(PLATFORM)

OTA_APP     := build/$(OTA_APPLICATION)/binary/$(OTA_APPLICATION).stripped.elf
```

```
snip.scan-<platform_name> ota2_image
```

Copy the resulting OTA2 image file onto a server accessible through your AP. The server must be open (no password).

```
<Wiced-SDK>/build/snip.scan-<platform_name><OS_name>/OTA2_image_file.bin
```

Build the snip.ota2_example application using this command line:

```
snip.ota2_example-<platform_name> ota2_factory_download download run
```

Change the WiFi configuration parameters to connect to your AP. You need to reboot the system after the “config save” for the changes to take effect.

Previous to SDK-3.7.1:

```
> config ssid <your_AP_ssid>
> config pass <your_AP_password>
> config save
```

After Version 3.7.1:

The method of changing the configuration parameters has changed. The values are saved directly to the DCT, so you do not to issue a “save” command. You do need to reboot the system for the changes to take effect.

```
> dct_wifi_ap_list 0 ssid <your_AP_SSID>
> dct_wifi_ap_list 0 pass <your_AP_passphrase>
```

Reboot the device and let it join your AP

```
> get_update <your_test_server>/<path_to_OTA_image_file.bin>
```

The device will now download the file, and show you a completion bar. The center vertical bar will show the progress.

```
|---|-------------------------|
```

When successfully completed, the device will print a message with how to proceed. Re-boot the device and the extraction will take place, and the new application will be running.

After SDK-3.7.1, the ota2_extraction code will run. When it is finished extracting the OTA2 image file, you will need to reboot the system again.

To get back to the Factory Reset Application (in this case ota2_example), reboot the device holding down the “Factory Reset button” for about ten (10) seconds. The device will then extract the Factory Reset OTA Image (flashed to the device when you had the IDE build the ota2_example with “ota2_factory_download” on the command line).

16 OTA2 and DCT changes

When building an updated application, it is important to designate the SDK that the original application was built under. This is due to layout changes in the system DCT section. Please see <Wiced-SDK>/doc/WICED-DCT.pdf for more depth information.

Starting with SDK-3.7.0 and forward, if the software programmed into your device was originally built under an earlier SDK, you must add an argument to instruct the new build that the DCT is to be treated as if it were built under the earlier SDK.

```
<application_name>-<platform_name> UPDATE_FROM_SDK=<bootloader_sdk>
```

Where `<bootloader_sdk>` is one of:

```
3_5_2
3_6_0
3_6_1
3_6_2
3_6_3
3_7_0
```

Example for building an upgrade application using SDK-3.7.1 for software that was originally built using SDK-3.6.0:

```
<application_name>-<platform_name> ota2_image UPDATE_FROM_SDK=3_6_0
```

There were optional structures in the System DCT that are now always included. They are the Bluetooth (BT), Peer to Peer (P2P) and Over The Air 2 (OTA2) sub-structures. This information must also be designated so that the code knows which (if any) of the optional structures were used in the original application build DCT. Add them to the make command line:

```
<application_name>-<platform_name> <optional_struct> <optional_struct> <optional_struct>
```

Where `<optional_struct>` is:

```
APP_USED_BT=1
APP_USED_P2P=1
APP_USED_OTA2=1
```

See <Wiced-SDK>/doc/WICED-DCT.pdf for more information.

BROADCOM CORPORATION
5300 California Avenue
Irvine, California, 92677

Phone : +1-949-926-5000
Fax: +1-949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com

WICED-OTA2 August 07, 2015