WICED™ OTA2 Update Support
(Over The Air update)

WICED™ by BROADCOM.

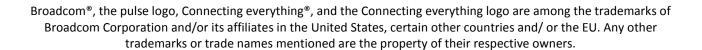# Revision History

Revision	*Date*	*Change Description*
WICED-OT2A-R 0.95	December 18, 2015	Add Usage notes and bring doc up to date
WICED-OT2A-R 0.94	December 2, 2015	Bring doc up to date with latest code
WICED-OT2A-R 0.93	November 3, 2015	Add more details, update APIs for latest code
WICED-OT2A-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 3.3.x** 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 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.

4 Memory Layout

4.1 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
- Bootloader DCT
- Copy of Application DCT (copied before any OTA2 Image is extracted)
- Current Application Area
  - DCT
  - File system
  - Current Application
  - Etc.
- Last Known Good Area (optional)
  - 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 Bootloader Application

The Bootloader is a small application that loads 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 or automatic product software updates. The Bootloader resides in a write-protected part of the Storage System.

The Bootloader is 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 a SoftAP with pre-defined Network parameters (Bootloader DCT)
- Start DHCP and a Webserver so a user can manually connect and load an OTA2 Image via a browser based upload.
- This functionality is configurable (excluded) if a "Push" model is not required.

Upon powering up the board, if the reset button pressed for ~10 seconds:
- Perform a Factory Reset, extracting the Factory Reset OTA2 Image to the Current Application Area.

During boot, the Bootloader will attempt to load the Current Application. If that fails and an optional LKG Application is present, it will attempt to load the LKG Application. If that fails, the Factory Reset is loaded. The failure condition is any of the below:
- The Security check fails on the image load
- The application has not loaded and cleared the watchdog timer within some configurable number of seconds.

5.2 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.

5.3 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
  - File system
  - Etc.

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

5.4 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).

5.5 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).

5.6 OTA2 Updates and DCTs

There will be 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 second 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 Bootloader Logic

The state of the Current Application Area, LKG (if enabled) and Staged OTA2 Image is stored in the Bootloader DCT, and is not accessible from an Application. The Bootloader will handle the OTA2 Image Extraction process.

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) can save an OTA2 Image and the download progress so that the Bootloader can verify/extract the OTA2 Image on a reboot (depending on pre-determined and/or user settings). Note

NOTES:
- The FLASH writing code can check for battery level before starting writing.
- Storage for the download_status and bytes_received can be located in a place other than the OTA Image Staging Area. This is adjustable at compile time.

6.1 Bootloader sequence

6.1.1 Check for Downloaded OTA2 Image

If OTA2 Image Staging Area has a downloaded OTA2 Image
- Check for settings regarding auto-updating (download_status == EXTRACT ON REBOOT)
    - o Possibly ask user to allow the update
- Run OTA2 Image Verification (secure sign, etc.)
    - o If Image verified, start update process:
        - ▪ If using LKG build, copy Current Application Area to LKG File Area
        - ▪ Preserve current DCT data (TBD)
        - ▪ Extract OTA2 Image from Staging Area to Current Application Area
        - ▪ If Extraction Fails and using LKG build
            - • Restore Current Application Area from LKG File System
    - o Mark OTA2 Image as "extracted" so we do not use it again
    - o If OTA2 Image Extraction (or LKG) good
        - ▪ Start Application (Reboot not necessary as Application was not running)

6.1.2 Check for Button press ~5 seconds

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

Is button pressed for 5 seconds?
- Is Current Application valid?
    - o If OK (no Security error or Watchdog reset)
        - ▪ Current Application runs with signal to run Application's SoftAP / Webserver.
    - o else
        - ▪ Bootloader will run a simple SoftAP / Webserver with only OTA2 update.

6.1.3 Check for Button press ~10 seconds

This time is configurable at compile time. If no SoftAP is supported, the default time will be ~5 seconds.

Is button pressed for 10 seconds?
- If Pressed, run Factory Reset program
  - Extract OTA2 Factory Reset Image to Current Area
  - Start Application (Reboot not necessary as Application was not running)

6.1.4 Check Current Application (File System)

Is Current Application valid?
- If OK (no Security error or Watchdog reset)
  - Current Application runs
- If Fail
  - If LKG support and LKG Area valid
    - Copy LKG Area to Current Application Area
    - Start Application (Reboot not necessary as Application was not running)
  - else if Current Application Version different from Factory Reset OTA2 Image version
    - Extract Factory Reset OTA2 Image to Current Application Area
    - Start Application (Reboot not necessary as Application was not running)
  - else
    - If a "Push" model is being used:
      - Run SoftAP and Webserver code within Bootloader (this is the default, minimal support to load an OTA2 image – no specific Application settings)
    - Else
      - Flash error LEDs

7 FLASH layout and application/DCT regions

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

7.1 External FLASH

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: - Possibly extract new OTA2 Image - Check for button presses - Validate Current Application - Possibly restore LKG (if enabled) - Possibly extract Factory Reset OTA2 Image Includes OTA2 Image Package Extractor and Default Minimal Web Server.
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).
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)
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 proposal (future support)

Internal Flash File System	
Bootloader Read Only Section (offset = 0x00, address increases down this table)	
Bootloader	Starts from power on to: - Possibly extract new OTA2 Image - Check for button presses - Validate Current Application - Possibly restore LKG (if enabled) - Possibly extract Factory Reset OTA2 Image Includes OTA2 Image Package Extractor and Default Minimal Web Server.

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).
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.) NOTE: Shown in the internal FLASH as it can be configured to reside there – this will be decided at compile time, and remain the same through the life of the product.
Current Application Area Read / Write	

Current Application Area	wiced_apps_lut[] DCT Resources file system (WiFi firmware, images, audio, web pages, etc.)
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.)
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   OTA2 Image Header

The first fields of the OTA 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 (TBD)
Minor Version	16 bit	Software Minor Version (TBD)
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	32 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

8.1 OTA2 Header Update Status during Download

The OTA2 Image will always be stored in FLASH. Where the download_status and bytes_recieved are stored can be changed for platform-specific requirements.

8.1.1 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.

8.1.2 Non-FLASH Based storage of download_status and bytes_received

If the platform supports a different storage location, there will be a Make file define available so that download_status and bytes_received can be stored elsewhere.

9 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.

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 (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 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.

```c
/********************************************************
 *                    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_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_DCT,
    WICED_OTA2_IMAGE_COMPONENT_FILESYSTEM,
    WICED_OTA2_IMAGE_COMPONENT_APPLICATION,
} 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;
```

```c
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 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);

/** 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 total_bytes_received - number of bytes written to the image  TODO: make this platform-
specific
 *
 * @return  WICED_SUCCESS
 *          WICED_BADARG
 *          WICED_ERROR
 */
wiced_result_t wiced_ota2_image_update_staged_status(wiced_ota2_image_status_t new_status);

/** 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);
```

11 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.
-       o   After each chunk, sets bytes_received value.
- Upon Completion
  - o   Sets download_status to "Download complete".
- If User / Application has set Automatic Update flag
  - o   Sets download_status to "Extract on Reboot".
  - o   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_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
                                     * 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.   */

    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_TIME_TO_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 immediately
                                     *        - WICED_ERROR   = Service will retry on next
check_interval
                                     *          Application can call
                                     *          wiced_ota2_service_check_for_updates()
                                     *          to run another check earlier
*/

} 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*           url;                    /* url to "get" updates from
*/
    char*           file_name;              /* filename to "get"
*/
    uint32_t        check_interval;         /* seconds between update 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_ota2_backround_service_params_t;

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

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

/**
 * Initialize a timed backgound service to check for updates
 *
 * @param[in]  session - value returned from wiced_ota2_service_init()
 *
 * @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
 *
 * @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: does not affect the timed checks - this is separate
 *
 * @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);
```

12 Snip.ota2_example Application

12.1 <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 ota2_bootloader
- Shows how to extract Factory Reset or Staged Update independent of the code in ota2_bootloader
- Shows calls to determine boot type (normal, factory reset, update)
- Some code in place (not tested) for :
    - Last Known Good Support
    - Checking Battery level before starting extraction
    - Using a different CRC function
- Example of how to download an image using HTTP GET.
- 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 preserve user settings during update or factory reset

12.2 How to build snip.ota2_example

This example is for BCM943907WAE_1.B0 or BCM943909WCD1_3.B0

Normal OTA2 example program build, creates the ota2_bootloader and all the components, and the application.

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

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

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

- 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.
- This will build ota2_bootloader, not the normal bootloader.

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

```
snip.ota2_example-<platform_name> ota2_download
```

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

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

12.3 snip.ota2_example console commands

connect <url>   Connect to a server (no security at the moment)
- get_update <filename>Use HTTP GET to download the OTA2 Image file from the server we are connected to and save to OTA2 Staging Area
- Default filename is "/brcmtest/OTA_image_file.ota_image" (get_update concatenates "<url>/" and <filename> for the HTTP GET request)

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 5 seconds will do this during a reboot

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"

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".

13 Testing an OTA2 update and Factory Reset

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

```
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
```

Change the WiFi configuration parameters to connect to your AP

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

Reboot the device and let it join your AP

```
> connect <your_test_server>
> get_update <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. Simply re-boot the device and the extraction will take place, and the new application will be running.

To get back to the Factory Reset Application (in this case ota2_example), reboot the device holding down the "Factory Reset button" for about five (5) 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).

14 Future Roadmap

These are planned features, not shown in any order of precedence.

14.1 Use of Internal + External Flash

- Allow mixture of Internal and External Flash for storage of various components (Application, Factory Reset OTA2 Image, Filesystem, etc.)

14.2 OTA2 Background Service

- Complete ota2_service background timer service for periodically checking for OTA2 Image update availability.
- Handle downloading piecemeal (in chunks) so that we can re-start if interrupted during a download.
- Handle re-start after interruption (like when user power cycles during a download).

14.3 Watchdog Timer trigger determination

- To determine the Current Application failed
  - o If so, extract Factory Reset Image

14.4 APPS LUT usage

- Add CRC to wiced_apps_lut[] structure
- Verify the current Application's CRC before running it

14.5 Add SoftAP support to the OTA2 Bootloader

- Add SoftAP Server & Web pages to manually upload an OTA2 Image
- Add Web page(s) for SoftAP
- Allow manual loading of OTA2_image to Staging Area
- Allow setting of EXTRACT_ON_REBOOT for downloaded OTA2_images

Connecting
e v e r y t h i n g ®

BROADCOM.

BROADCOM CORPORATION

5300 California Avenue
Irvine, California, 92677
© 2015 by BROADCOM CORPORATION. All rights reserved.

WICED-OTA2   August 07, 2015

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