SAMR34 LoRaWAN FUOTA Demo QuickStart Guide

Table of Contents

Introduction

This document explains the steps required to configure and use firmware over-the-air upgrade (FUOTA) in Microchip LoRaWAN Stack (MLS). FUOTA Demo is an example application to demonstrate FUOTA in LoRaWAN. It is provided as Atmel Studio project and contains the necessary HAL drivers, LoRaWAN stack, FUOTA support packages and the demo application logic.

Hardware

- Microchip SAMR34 Xplained PRO evaluation kit[1]

Software

1. Atmel Studio 7.0[2]
2. JRE version 1.6
3. Bootloader PC Tool 1.2.2.233 (available in project)
4. Python 3.x
5. Prepare_image.py (available in project)[3]
6. Actility ThingPark RMC FUOTA Service Account

Development environment

This section provides information on the required tools needed to setup and build the example project, and the platform to run it on.

Atmel Studio 7

Atmel Studio 7 can be used to develop and debug applications for Microchip ARM-based platforms. Atmel Studio 7 is equipped with the GCC compiler and does not require any additional external software tools to compile and debug SAMR34 LoRaWAN applications.

Below are a few reference links to documents that will help you get started with Studio 7.
https://www.microchip.com/mplab/avr-support/atmel-studio-7
http://ww1.microchip.com/downloads/en/DeviceDoc/Getting-Started-with-Atmel-Studio7.pdf
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-Studio-7-User-Guide.pdf

Here are additional links to detailed documents to help you understand the SAMR34/35 LoRaWAN stack.
http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-R34-R35-Microchip-LoRaWAN-Stack-Software-API-Reference-Manual-DS70005382A.pdf
http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-R34-MLS-Getting-Started-Guide-User-Guide-DS50002812A.pdf

SAMR34 Xplained Pro [DM320111]

Full information regarding the SAMR34 Xplained Pro evaluation kit can be found at the links provided:
https://www.microchip.com/DevelopmentTools/ProductDetails/DM320111
https://www.microchip.com/wwwproducts/en/ATSAMR34J18
http://ww1.microchip.com/downloads/en/DeviceDoc/ATSAMR34-Xplained-Pro-User-Guide-DS50002803A.pdf

Device Time – GPS Epoch Maintenance

LoRaWAN end-devices implementing class-B or class-C option, has support for receiving multicast messages. Multicasting can be used to send the new firmware image over-the-air, to multiple devices simultaneously. Such simultaneous multicast, happens over a multicast session and therefore, requires synchronization between end-device and network server in order to properly receive the firmware bytes. The starting time of this session is provided by Update Server in GPS Epoch. Therefore, it is necessary for end-device to know the current GPS Epoch time for synchronizing with the multicast session from network server.

But, SAMR34 Xplained PRO does not have GPS hardware and only maintains a system time using its local hardware timer. This system time is valid only when the system is powered-on. And it will be reset, if a power-on-reset or backup sleep happens – although, standby sleep does not reset the system time.

To overcome the non-availability of GPS hardware, end-device running FUOTA Demo application firmware utilizes DeviceTimeReq/Ans command pair provided by the network server. ED uplinks DeviceTimeReq to network server. For which, ED gets the DeviceTimeAns with GPS epoch time. This time value points to the GPS epoch time at the instance of end-of-transmission that sent the last DeviceTimeReq.

Once the DeviceTimeAns is received, end-device records both the GPS Epoch Time and the system time. It then uses the difference between the recorded system time stamp and the current time stamp to compute the current GPS Epoch time. Also, FUOTA Demo App refreshes its GPS Epoch time stamp by sending DeviceTimeReq command in every uplink.

FUOTA Demo App

FUOTA Demo App is an example application to demonstrate the over-the-air firmware upgrade through Microchip LoRaWAN Stack. LoRaWAN FUOTA update server can update either a single device by unicast, or many end-devices by multicast. The application requires a bootloader to be pre-programmed in SAMR34 Xplained PRO. Then, initial firmware image of the FUOTA Demo project shall be updated using Bootloader PC tool application provided in the project itself. Bootloader PC Tool takes SREC format as input.

NOTE: Bootloader firmware is also provided as ELF file separately in the Studio project. Bootloader PC Tool requires application firmware in SREC format.

Application Structure

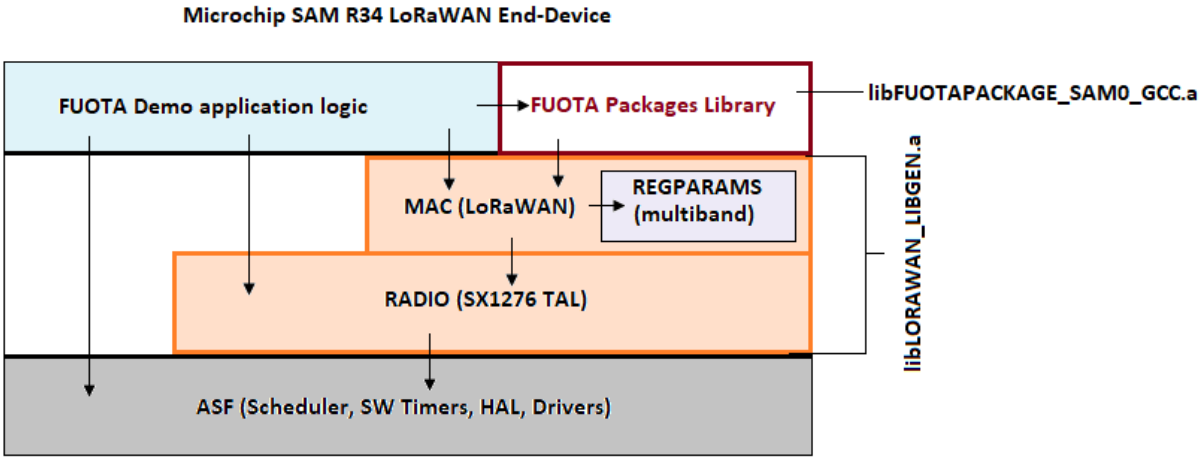


Figure 1FUOTA Demo application structure

Project location	thirdparty/wireless/lorawan/apps/fuota_demo/multiband_bl/as5_arm
Project name	APPS_FUOTA_DEMO.cproj
FUOTA lib	libFUOTAPACKAGE_SAM0_GCC.a
LoRaWAN lib	libLORAWAN_LIBGEN.a
Application logic	thirdparty/wireless/lorawan/apps/fuota_demo/main.c

Memory Layout with Bootloader

Typical application firmware compiles to run from FLASH address 0x00000000. But, FUOTA Demo application runs alongside a bootloader. Hence, its memory layout starts after the bootloader firmware in FLASH and compiled to run from address 0x00004000.

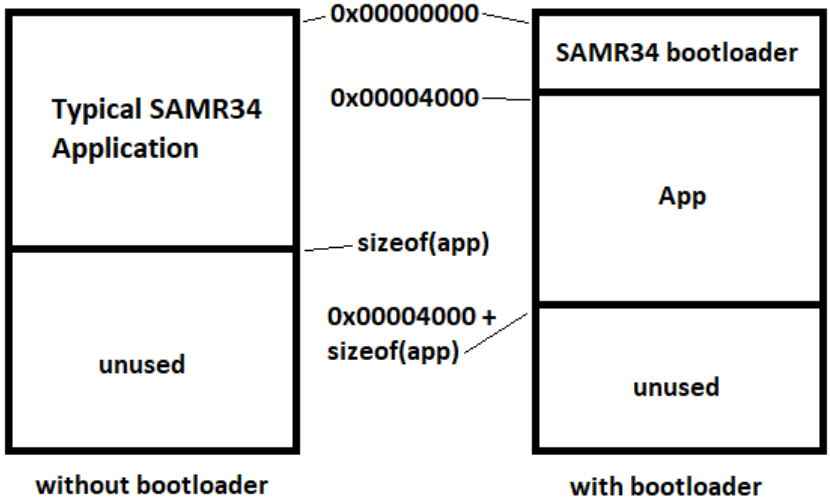


Figure 2Difference in memory layout in terms of Bootloader presence

Bootloader Memory Requirements

SAMR34 bootloader is compiled to run from address 0x00000000 and MUST be flashed before uploading FUOTA Demo app. First 16K of flash memory, is assumed to be reserved for bootloader and application must not use the same. Besides, bootloader requires 512 bytes of scratch space to maintain application information during startup. Therefore, the last 512 bytes of the flash memory is reserved for bootloader.

Bootloader Start Address	0000:0000
Bootloader End Address	0000:3FFF
Bootloader Reserved Size	16K
Bootloader App Info Start Address	0003:FE00
Bootloader App Info End Address	0003:FFFF
Bootloader App Info Reserved Size	512 bytes

FUOTA memory layout

To perform firmware upgrade, SAMR34 needs extra space to store the newly received firmware. But, SAMR34 Xplained PRO does not have external serial flash available. Therefore, newly upgraded firmware is stored in internal flash itself. In order to achieve that, application space i.e., exclusive of bootloader's space, is interleaved into 2 sections say, IMAGE_1 and IMAGE_2 in this case. Where, IMAGE_1 is immediately followed by IMAGE_2 in flash.

IMAGE_1 Start Address	0000:4000
IMAGE_1 End Address	0002:1FFF
IMAGE_1 Maximum Size	122880 bytes
IMAGE_2 Start Address	0002:2000
IMAGE_2 End Address	0003:FDFF
IMAGE_2 Maximum Size	122368 bytes (512 bytes less due to bootloader's app info)

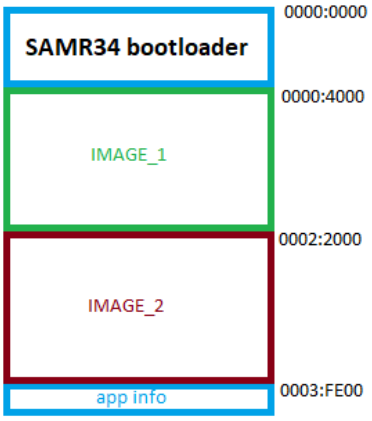


Figure 3FUOTA Demo Application Memory Layout

Bootloader App Info

```
typedef struct {
 uint8_t       crc;              // CRC over the entire app-info (excluding this field)
 /*
 Specifies default application index for the bootloader to jump to,
 if the Active fields in application-related fields are in inconsistent state
 (e.g. all set as non-active).
 */
   /*Application-configurable bit-mask indicating enabled bootloader features and interfaces:
   Bit 0: multi-image support
   Bit 1: RF interface support
   Bit 2: USART 0 interface support
   Bit 3: USART 1 interface support
   Bit 4: SPI interface support
   Bit 5: TWI interface support
   Bit 6: USB interface support
   Bit 7-15: Reserved
 */
 uint8_t reservedbytes;
 uint8_t reservedbytes1;
 featureMask_t   required_features; // Bit-mask for features requested by app (See boot_version.h for bit
positions)
 // Number of valid applications in internal flash. Shall be at least 1.
 uint8_t       reserved1[3];       // Place-holder
 app_table_t    appTable[1];       // App-specific sub-table for each app (upto 2)
} app_info_t;


typedef struct app_table_tag {
 char app_name[16];           // Description string for the app
 uint32_t  app_start_addr;      // Start address of the app image
 uint32_t  app_size;          // Size of the app image
 uint32_t  src_addr;          // Address where the image is downloaded
 uint32_t  dest_addr;          // Address where the image should be placed at the end of upgrade
 uint32_t  skip_pds_start;
 uint32_t  skip_pds_end;
 uint32_t  img_size;          // Size of the downloaded image
 uint8_t   imgcrc;           //CRC of the downloaded image
 ExistingImageInfo_t existingImageInfo;  // 2 Byte flag to indicate information abt existing image
 BootInfo_t bootInfo;         // 2 byte flag
 uint8_t   security_key [16];    // Security key for recovering the encryted image after download
```

```
}app_table_t;

typedef struct
{
  uint16_t  update_action:1;     // Whether an image update is pending
  uint16_t  src_memtype:1;       // Whether the downloaded image is in internal or external
  uint16_t  dest_memtype:1;      // Whether the executable location is in internal or external
  uint16_t  img_type:2;          // Type of the downloaded image: Encrypted app, Plain app, not-an-executable-app
  uint16_t  reserved:11;         // Place-holder
}BootInfo_t;
```

The bootloader app info section will be written by the FUOTA Demo application to indicate to bootloader that a new image is pending for upgrade. Actual upgrade will be done by bootloader by overwriting the IMAGE_1 section with the contents from IMAGE_2. During startup, bootloader firmware will read the app info section and checks for **update_action** field. If **update_action** is set to 1, then bootloader will upgrade to new image.

Application Support Layers

FUOTA Demo application uses the following components to simplify elapsing long timeouts, LoRaWAN messaging and NVM access. They are available in **apps/packages/common/** folder.

Component Interface	Description
msg.h	LoRaWAN messaging support interfaces
fuota_nvm.h	NVM interfaces for firmware storage & access
long_timer.h	Timer to elapse interval > SWTIMER_MAX_TIMEOUT

LoRaWAN messaging interfaces

This layer currently supports buffering only one request. Once request is accepted, it will be buffered. It is provided as source; therefore, it can be extended with other features if required. It provides:

1. Message buffering (currently one)
2. Message retries
    a. Failure due to duty cycle pending
    b. Failure due to MAC or RADIO error

Usage of each interface is documented in msg.h header file itself. Please refer to appInit() function in application logic (main.c).

NOTE: It is important to invoke MSG_Process( ) function every time application task handler is called. MSG_Process() function keeps the messaging support rolling within application context.

Application Flow

OTAU happens as part of downlink processing. Application checks the FPORT and redirects the FUOTA application support commands to corresponding handlers.

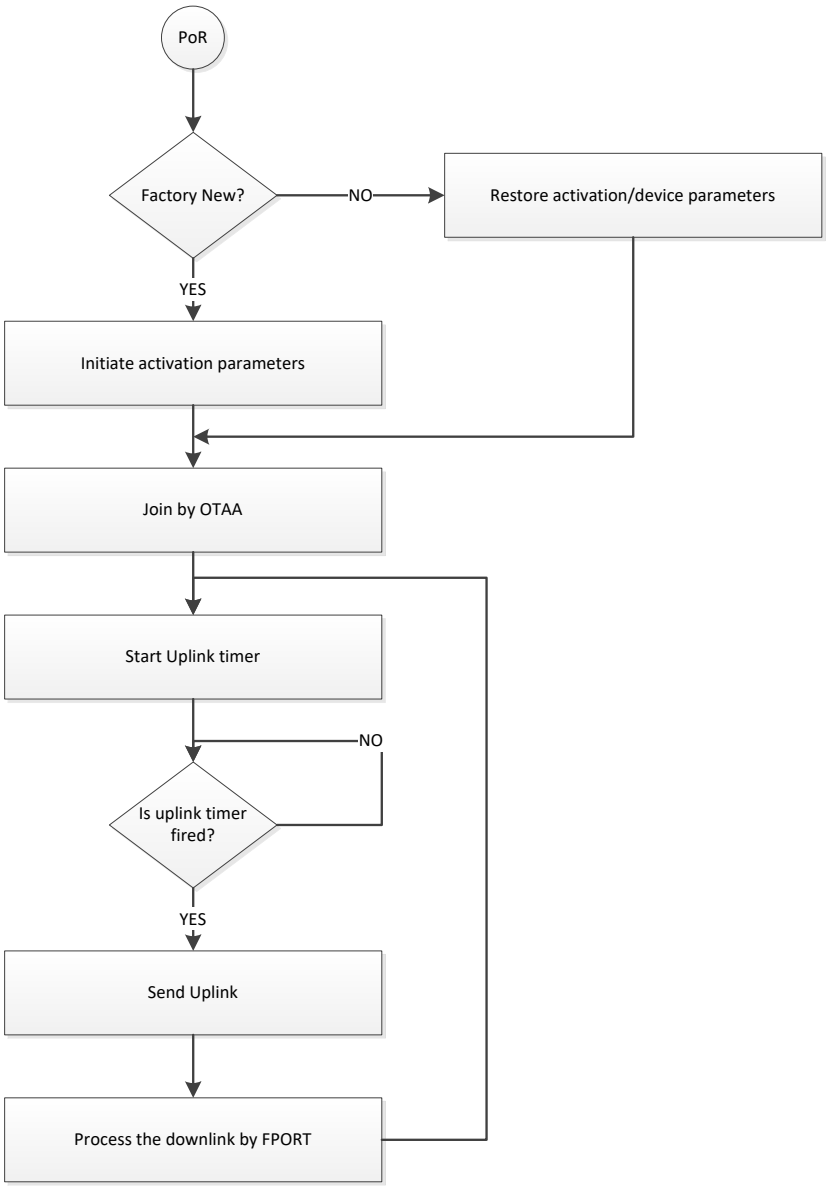


Figure 4FUOTA Demo application flow

Note:

User shall reset the device (either soft or hard) to trigger the bootloader mode in order to upgrade to new firmware. Once, SAMR34 is upgraded to new firmware, it cannot revert to previous version. FUOTA Demo application is currently configured to be manually reset in order to trigger upgrade.

Application Project and Configuration

There are few configurable parameters in application. It is available in conf_app.h header file under config section of the project. Project is available at the following location:

`thirdparty/wireless/lorawan/apps/fuota_demo/multiband_bl`

```
/* Select the band for this app */
#define APP_ISMBAND                             (ISM_EU868)
//#define APP_ISMBAND                             (ISM_NA915)
//#define APP_ISMBAND                             (ISM_NZ923)
//#define APP_ISMBAND                             (ISM_IND865)

/* Select the Type of Transmission - Confirmed(CNF) / Unconfirmed(UNCNF) */
#define APP_TRANSMISSION_TYPE                   LORAWAN_UNCNF
//#define APP_TRANSMISSION_TYPE                   LORAWAN_CNF

#define APP_EDCLASS                             (CLASS_A)
//#define APP_EDCLASS                             (CLASS_C)

/* FPORT Value (1-255) */
#define APP_FPORT                               (1)

/* OTAA Join Parameters */
#define DEVICE_EUI                              {0x00, 0x04, 0x25, 0x19, 0x18, 0x01, 0xFF, 0xFF}
#define APPLICATION_EUI                         {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
#define APPLICATION_KEY                         {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x

/* Other security parameters */
#define APP_GENAPPKEY                           {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x

/* This macro defines the application's uplink timer interval in milliseconds */
#define APP_UPLINKINTERVAL_MS                   (10000)

/* This macro defines the application's default sleep duration in milliseconds */
#define APP_SLEEPTIME_MS                        (PMM_SLEEPTIME_MAX_MS)

/* This macro defines the number of retries to be done for each failed send request */
#define APP_SENDRETRY_COUNT                     (3)

/* This macro defines the timeout in milliseconds to elapse between send retries */
#define APP_SENDRETRYINTERVAL_MS                (6000)

/* This macro defines the custom FUOTA descriptor to be used by FTMPackage */
#define APP_FTM_FUOTA_DESCRIPTOR                (0x2)

#endif /* APP_CONFIG_H_ */
```

Parameter	Description
APP_ISMBAND	Choose the required region
APP_TRANSMISSION_TYPE	LoRaWAN message type to be used in uplinks
APP_EDCLASS	Default LoRaWAN class of the application
APP_FPORT	Frame port number to be used in uplinks
APP_UPLINKINTERVAL_MS	Interval between consecutive uplinks (in seconds)
APP_SLEEPTIME_MS	Device sleep duration (PMM_SLEEP_MIN <= duration < PMM_SLEEP_MAX)
APP_SENDRETRY_COUNT	Number of retries to done for a failed uplink. **Note**: Applies only to uplink that is queued into MAC
APP_SENDRETRYINTERVAL_MS	Interval between consecutive application retries. **Note**: This parameter does not control MAC **NbTrans**
APP_FTM_FUOTA_DESCRIPTOR	32-bit unsigned value to check against Descriptor field in FragSessionSetupReq message. **Note**: Set the same value that will be sent by update server.
APP_GENAPPKEY	GenAppKey to dervice McKey
APPLICATION_EUI	Application EUI 64-bit
APPLICATION_KEY	Application Key 128-bit
DEVICE_EUI	Device EUI 64-bit

Editing Build Parameters

FUOTA Demo app can read the device's unique ID supplied from the hardware itself. This is controlled by a build switch viz. EDBG_EUI_READ. If this is set to '1', then application reads the unique ID from the hardware during initialization and uses it as DevEUI. It the build switch is set to any other value then application must provide a valid DevEUI in 'APP_DEVEUI'.

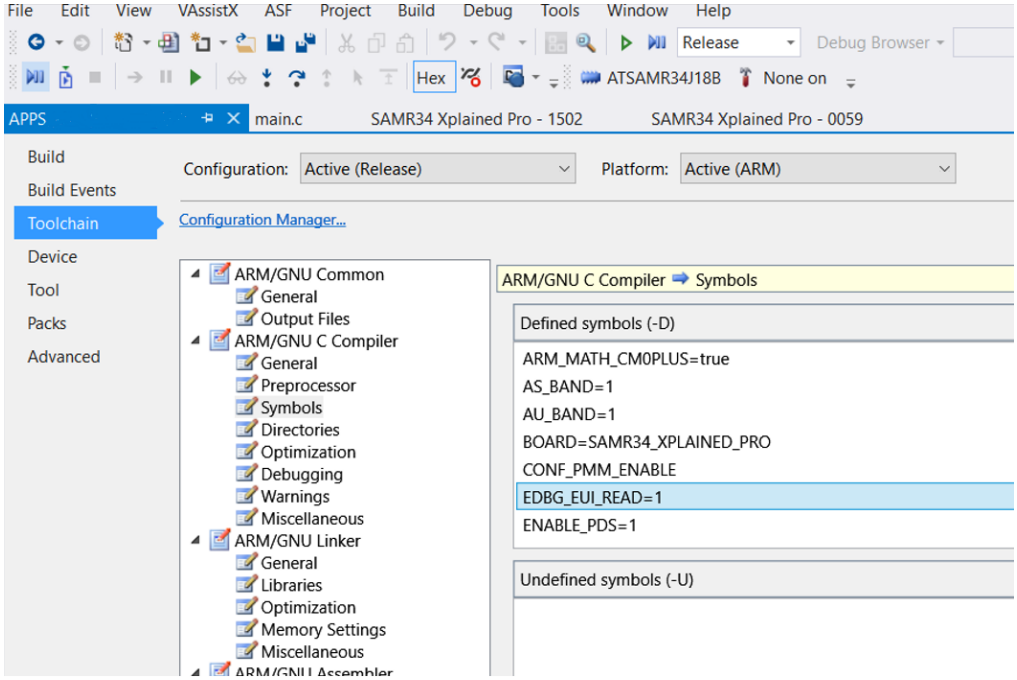


Figure 5Edit Build Symbols

Build configuration can be changed from 'Toolchain properties' dialog in Atmel Studio.

- Press [Alt+F7] or click [Project →APPS_FUOTA_DEMO Properties]
- Select 'Toolchain' tab
- From 'ARM/GNU C Compiler' section, choose 'Symbols'
- It will bring-up the build switches dialog where EDBG_EUI_READ can be modified as per requirement

LEDs

LED0 (AMBER)	ON	New image successfully received and available for upgrade
LED0 (AMBER)	OFF	---
LED1 (GREEN)	ON	Device is in ACTIVE mode
LED1 (GREEN)	OFF	Device is in LOW-POWER (STANDBY) mode

SW0

Scenario	Button State	Description
During startup	Pressed	Application will clear the PDS to factory NEW state
In class-A mode	Pressed	Toggles the uplink timer (to either ON or OFF)
In class-C mode	Pressed	Triggers a uplink

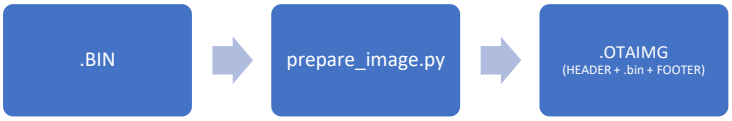
Versioning Information

```
#define APP_FW_VERSION          (0x00000002)
#define APP_HW_VERSION          (0x00000001)
```

Application has these macros shown above in main.c file, to input the hardware and firmware versioning information.

Preparing OTA image

To perform over-the-air upgrade, a firmware image needs to be provided to update server. The over-the-air-image is generated by this script by adding header and footer information to RAW BINARY output (.bin), generated by Atmel Studio during compilation. This script adds a 64-byte header and 1-byte footer to the original raw binary.



Header contents:

```
typedef struct _FotaImageHeader
{
        uint32_t magic;
        uint8_t  fotaImgHdrLen;
        uint8_t  pad1;
        uint8_t  pad2;
        uint8_t  pad3;
        uint32_t stackVersion;
        uint32_t appVersion;
        uint32_t hwVersion;
        uint32_t flags;
        uint32_t fotaImgSize;
        uint8_t  crc8;      // CRC-8 for this header
        uint8_t  pad4;
        uint8_t  pad5;
        uint8_t  pad6;
} FotaImageHeader_t;
```

Scripts appends 8-bit CRC at the end of the image for integrity validation. This CRC is computed only for the actual image bytes and does not include FotaImageHeader_t. In device, once all firmware bytes are received over-the-air, application computes CRC for the image stored in IMAGE_2 section once again (excludes the HEADER and FOOTER). This CRC is compared with the CRC of over-the-air-image (as footer) for validating the integrity.

```
typedef struct {
 uint8_t       crc;              // CRC over the entire app-info (excluding this field)
 /*
 Specifies default application index for the bootloader to jump to,
 if the Active fields in application-related fields are in inconsistent state
 (e.g. all set as non-active).
 */
   /*Application-configurable bit-mask indicating enabled bootloader features and interfaces:
  Bit 0: multi-image support
  Bit 1: RF interface support
  Bit 2: USART 0 interface support
  Bit 3: USART 1 interface support
  Bit 4: SPI interface support
  Bit 5: TWI interface support
  Bit 6: USB interface support
  Bit 7-15: Reserved
 */
```

```
  uint8_t reservedbytes;
  uint8_t reservedbytes1;
  featureMask_t   required_features; // Bit-mask for features requested by app (See boot_version.h for bit positions)
  // Number of valid applications in internal flash. Shall be at least 1.
  uint8_t       reserved1[3];       // Place-holder
  app_table_t     appTable[1];        // App-specific sub-table for each app (upto 2)
} app_info_t;

typedef struct app_table_tag {
  char app_name[16];            // Description string for the app
  uint32_t  app_start_addr;       // Start address of the app image
  uint32_t  app_size;           // Size of the app image
  uint32_t  src_addr;           // Address where the image is downloaded
  uint32_t  dest_addr;          // Address where the image should be placed at the end of upgrade
  uint32_t  skip_pds_start;
  uint32_t  skip_pds_end;
  uint32_t  img_size;           // Size of the downloaded image
  uint8_t   imgcrc;             //CRC of the downloaded image
  ExistingImageInfo_t existingImageInfo;  // 2 Byte flag to indicate information abt existing image
  BootInfo_t bootInfo;          // 2 byte flag
  uint8_t   security_key [16];    // Security key for recovering the encryted image after download
} app_table_t;

typedef struct
{
  uint16_t   update_action:1;      // Whether an image update is pending
  uint16_t   src_memtype:1;        // Whether the downloaded image is in internal or external
  uint16_t   dest_memtype:1;       // Whether the executable location is in internal or external
  uint16_t   img_type:2;          // Type of the downloaded image: Encrypted app, Plain app, not-an-executable-app
  uint16_t   reserved:11;          // Place-holder
}BootInfo_t;
```

If the CRC validation is successful then, FUOTA demo app writes 'application boot information' or appinfo. Boot loader reads the appinfo every time, it starts to check if there is any update_action set in BootInfo_t. If update_action is set for boot loader, then boot loader revalidates the CRC of stored firmware and upgrades to new image on success. That is, it copies the contents from IMAGE_2 to IMAGE_1. Once the upgrade is completed, boot loader resets the update_action. If update_action is not set when boot loader starts, then it will simply start the application stored in IMAGE_1 section.

Syntax:

$ python prepare_image.py APPS_FUOTA_DEMO.bin
$ ls
prepare_image.py APPS_FUOTA_DEMO.bin APPS_FUOTA_DEMO.otaimg

NOTE: Python version 3 is required to run this script.

The OTAIMG file shall be uploaded to Actility ThingPark RMC server for sending over multicast session.

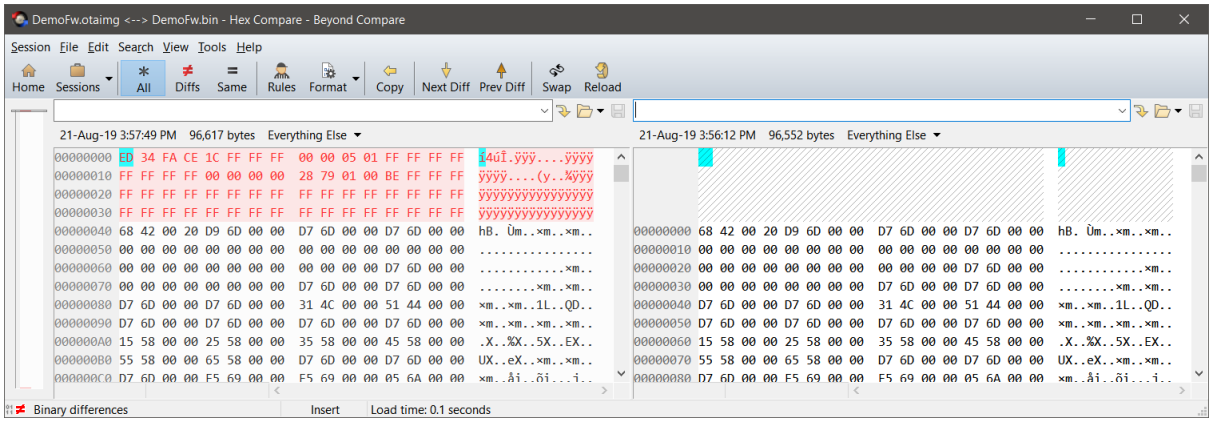


Figure 6: OTAIMG with 64-byte header added

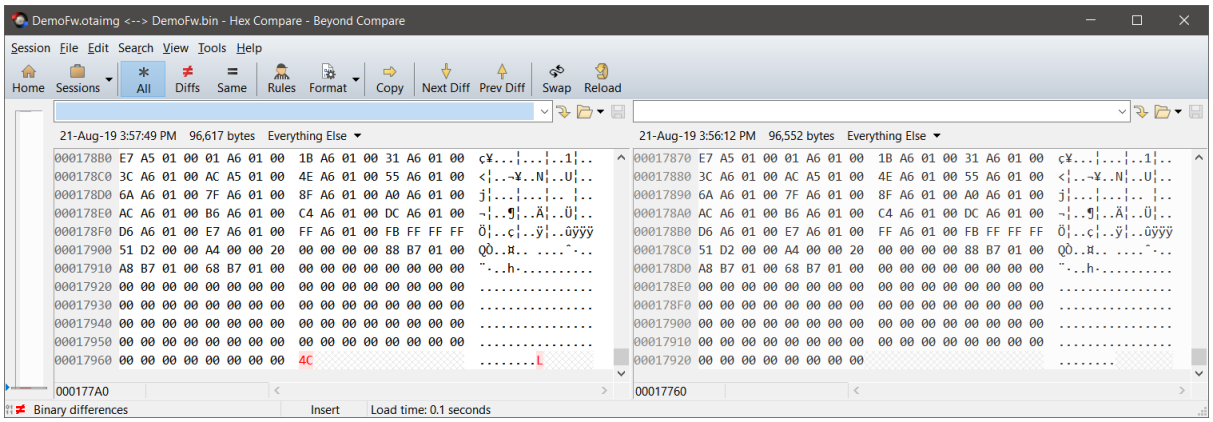


Figure 7: OTAIMG with CRC-8 at the footer

Setting FUOTA Log Level

FUOTA Demo uses log module which supports several levels. They are **NONE, INFO, WARN, ERROR, DEBUG**. FUOTA package library is pre-compiled with highest level of log information i.e., DEBUG. Application also includes this log module to configure the log level.

By default, FUOTA Demo app is configured to compile with LOG_LEVEL=ERROR. To change the log-level, please open the “Symbols” window in project properties dialog and edit the **LOG_LEVEL** build parameter. Refer this figure given above to edit the LOG_LEVEL.

Bootloader PC Tool

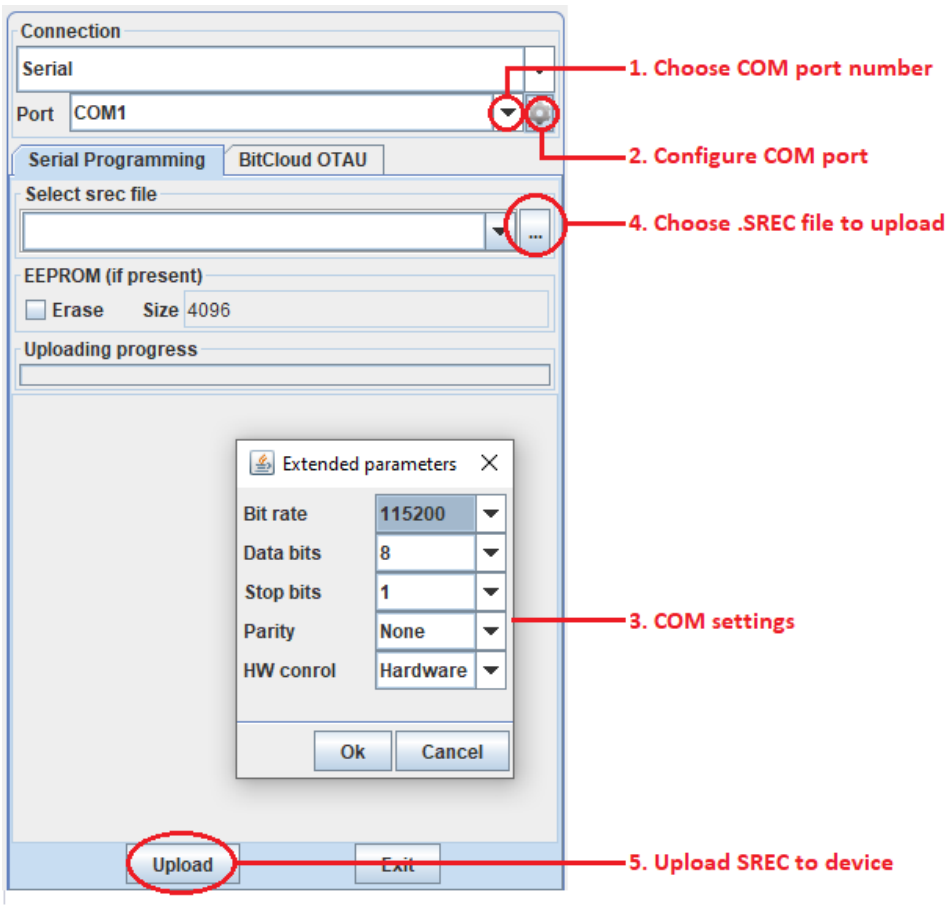


Figure 8Boot Loader PC Tool Usage

Image uploaded using Bootloader PC Tool will be written into IMAGE_1 section.

Over-the-air Upgrade

FUOTA Demo is used with Actility RMC update server for over-the-air upgrade. The steps involved in upgrade are…

Image fragments are transported to end-device(s) via multicast downlink RX2 windows. Multicasting is supported in either class-B or class-C. MLS currently does not support Class-B operation. Multicast happens by mentioning the multicast group address as the *SrcAddr* in MAC downlink frame. Therefore, it is necessary for the end-device to switch to class-C or class-B before OTAU. And, end-device must be added to a multicast group. Update server is a special type of application server that serves new image to a fleet of end-devices that are combined in the same multicast group.

Step 1: Multicast Group Setup

```
UPLINK Channel=0, Frequency=868099840Hz, Status=OK
[DEBUG] Sending... DR5, FPort=1, FcntUP=38, FRMPayloadLen=4
DONE, Status = SUCCESS
[DEBUG] Unicast packet received
[DEBUG] FPort   = 200
[DEBUG] Length  = 30
[DEBUG] Fcnt    = 38
[DEBUG] Payload:
020001000ff0873f1241da7a95b1eb59589c7579f5d92d1a0000212b0000
[DEBUG] MC_GROUP_SETUP_REQ
[DEBUG] mcGroupId=0 mcAddr=0xf00f0001
[DEBUG] minFcnt=6701 maxFcnt=11041
[DEBUG] McAppSKey=DD6EA450F585AAC64B5EA71B860DB55D
[DEBUG] McNwkSKey=94E241C7D5E691E3206F34D93313B6CF
[DEBUG] MC_GROUP_SETUP_ANS idError=0
[DEBUG] Sending... DR5, FPort=200, FcntUP=39, FRMPayloadLen=2
[DEBUG] Dutycycle pending: 3663ms
[DEBUG] Send unsuccessful..! Retries left = 3
[DEBUG] Sending... DR5, FPort=200, FcntUP=39, FRMPayloadLen=2
```

Step 2: Fragmentation Session Setup

```
DONE, Status = SUCCESS
UPLINK Channel=0, Frequency=868099840Hz, Status=OK
[DEBUG] Sending... DR5, FPort=1, FcntUP=41, FRMPayloadLen=4
DONE, Status = SUCCESS
[DEBUG] Unicast packet received
[DEBUG] FPort   = 201
[DEBUG] Length  = 11
[DEBUG] Fcnt    = 41
[DEBUG] Payload:
0201b201d8009702000000
[DEBUG] FRAG_SESSION_SETUP_REQ
[DEBUG] frags=434 fsize=216 padd=151
[DEBUG] fragidx=0 mcgpmask=0x1 desc=0x00000002 blkackdly=7s
[DEBUG] imgSize=93593
[DEBUG] FRAG_SESSION_SETUP_ANS err=0x00
[DEBUG] Sending... DR5, FPort=201, FcntUP=42, FRMPayloadLen=2
[DEBUG] Dutycycle pending: 3766ms
[DEBUG] Send unsuccessful..! Retries left = 3
[DEBUG] Sending... DR5, FPort=201, FcntUP=42, FRMPayloadLen=2
DONE, Status = SUCCESS
```

Step 3: Multicast Class-C Session Setup

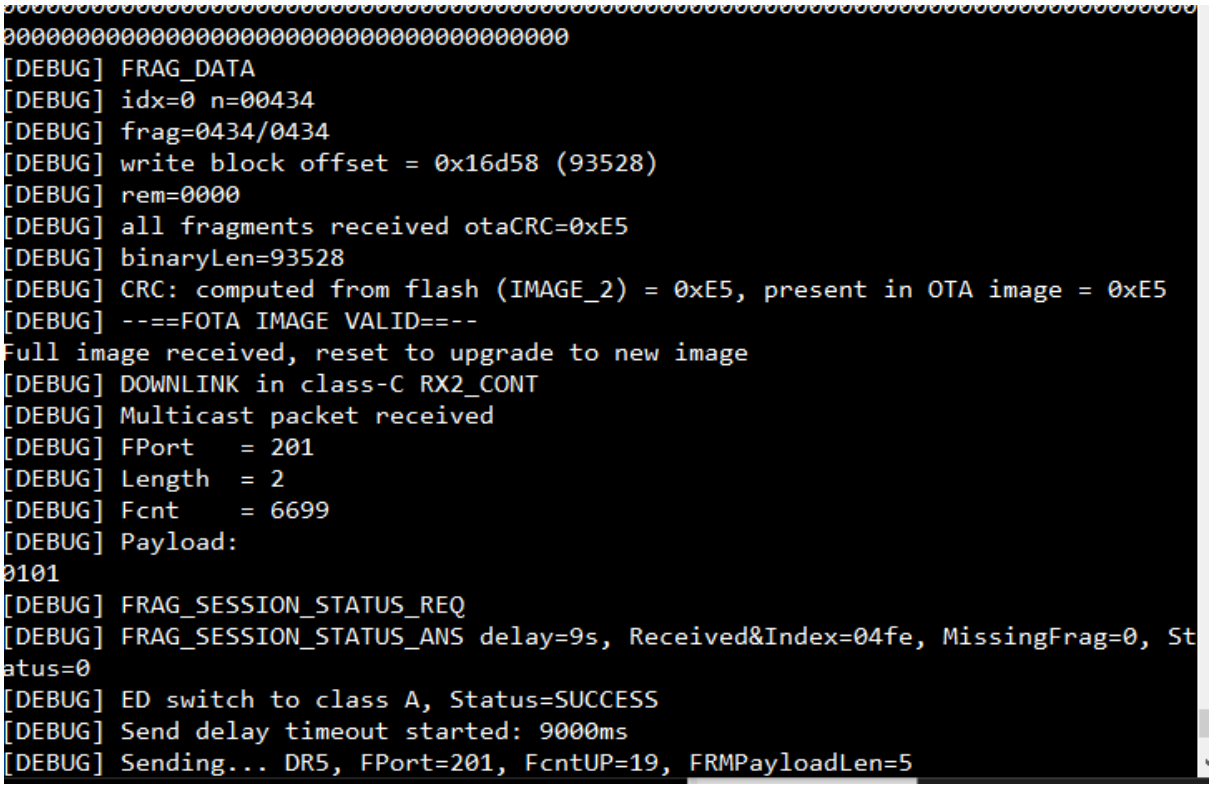
```
DONE, Status = SUCCESS
UPLINK Channel=1, Frequency=868300000Hz, Status=OK
[DEBUG] Sending... DR5, FPort=1, FcntUP=44, FRMPayloadLen=4
DONE, Status = SUCCESS
[DEBUG] Unicast packet received
[DEBUG] FPort   = 200
[DEBUG] Length  = 11
[DEBUG] Fcnt    = 44
[DEBUG] Payload:
04008896514b0fd2ad8404
[DEBUG] MC_CLASSC_SESSION_REQ
[DEBUG] now=1263638101s (gps epoch)
[DEBUG] sst=1263638152s (gps epoch)
[DEBUG] delta=50141ms
[DEBUG] tmo=32768s
[DEBUG] freq=869525000
[DEBUG] datarate=DR4
[DEBUG] periodicity=0
[DEBUG] class C session time set. tmo=40113ms
[DEBUG] MC_CLASSC_SESSION_ANS: 0400320000
[DEBUG] Sending... DR5, FPort=200, FcntUP=45, FRMPayloadLen=5
[DEBUG] Dutycycle pending: 3745ms
[DEBUG] Send unsuccessful..! Retries left = 3
[DEBUG] Sending... DR5, FPort=200, FcntUP=45, FRMPayloadLen=5
DONE, Status = SUCCESS
```

Step 4: Data Fragment Transport

File Edit Setup Control Window Help

```
[DEBUG] FRAG_DATA
[DEBUG] idx=0 n=00244
[DEBUG] frag=0244/0434
[DEBUG] rem=0010
[DEBUG] DOWNLINK in class-C RX2_CONT
[DEBUG] Multicast packet received
[DEBUG] FPort   = 201
[DEBUG] Length  = 219
[DEBUG] Fcnt    = 6076
[DEBUG] Payload:
08f500d4331b78042b33d02300d4331b78012b05d123005c331a780e231a422cd102210120204e25
70b0472300b6331b78002d24d10b20034083421fd123005c331b78db0703d529003030174b984722
0001205c3213786c3483431370042322780b2113432370b0470020104b9847082004e00f4b984708
28c7d0112076bd34220b2013409342f9d104380a4b98470022d2342078094b00920949094ca047e6
e7c046a01b00208dce0000e5cb0000794201008d3a0100410d0100b1fc000050c3000069d90000f0
b50700012489b005930eab049204cb03911d78
[DEBUG] FRAG_DATA
[DEBUG] idx=0 n=00245
[DEBUG] frag=0245/0434
[DEBUG] rem=0010
[DEBUG] DOWNLINK in class-C RX2_CONT
[DEBUG] Multicast packet received
```

Step 5: Image Validation and Frag Session Status



References

1. Microchip SAMR34 Xplained PRO - https://www.microchip.com/DevelopmentTools/ProductDetails/dm320111
2. Microchip Atmel Studio 7 IDE - https://www.microchip.com/mplab/avr-support/atmel-studio-7
3. Microchip LoRaWAN Stack Software API manual - http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-R34-R35-Microchip-LoRaWAN-Stack-Software-API-Reference-Manual-DS70005382A.pdf
4. Microchip LoRaWAN Stack Getting Started User Guide - http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-R34-MLS-Getting-Started-Guide-User-Guide-DS50002812A.pdf