RA2L1

Firmware Update over LoRaWAN® Sample Application

Introduction

This application note describes a sample application to update a firmware over LoRaWAN®. The process to update the firmware over LoRaWAN is called as FUOTA (Firmware Update Over The Air) and application layer protocols used for the update are standardized in the LoRa Alliance®.

This sample application supports the FUOTA process, which is intended for the end device in the LoRaWAN network.

Feature

- FUOTA sample application
  - API functions to handle the FUOTA related application layer protocols:
    - Clock synchronization message
    - Remote multicast setup
    - Fragmented data block transport
    - Firmware management (experimental)
    - Multi package access (experimental)
  - Multicast sessions in Class B and Class C operation.
  - User command interface based on AT commands format.
- Firmware update sample application
  - Firmware update of the internal code flash memory.
- Tool
  - Converter tool to generate a firmware image data file from an object file.

Target Device

- MCU: Renesas RA2L1 (R7FA2L1AB)
- Transceiver: Semtech SX1261 or SX1262

Contents

1. Overview

FUOTA (Firmware Update Over The Air) provides a function to remotely update a firmware over the wireless communication. This function is a key feature for IoT applications deployed widely in the field and required long term operation.

The LoRa Alliance standardized the FUOTA process utilizing the application layer protocols on top of the LoRaWAN protocol, such as the clock synchronization message protocol, the remote multicast setup protocol, and the fragmented data block transport protocol. These protocols can realize to deliver a firmware image (thereafter referred to as “F/W image”) to multiple devices at the time specified by an application server.

Figure 1 shows the overview of the FUOTA in the LoRaWAN network architecture. The application server requests the LoRaWAN network server to deliver the F/W image to an end device or a group of end devices with the time of the delivery. The LoRaWAN network server delivers the F/W image to end device(s) via the LoRaWAN wireless network according to the request.

The application layer protocols are utilized for the delivery from the LoRaWAN network server to the end device(s). The fragmented data block transport protocol provides the functions to divide the F/W image into the size less than the maximum size of a message that can be transmitted in the LoRaWAN network and reconstruct them into the F/W image. The remote multicast protocol provides the functions to simultaneously deliver the fragmented F/W image to a group of end devices. The clock synchronization protocol provides the functions to synchronize the end device’s clock to the LoRaWAN network’s GPS clock so that the end devices can prepare for the delivery and receive the fragmented F/W image.

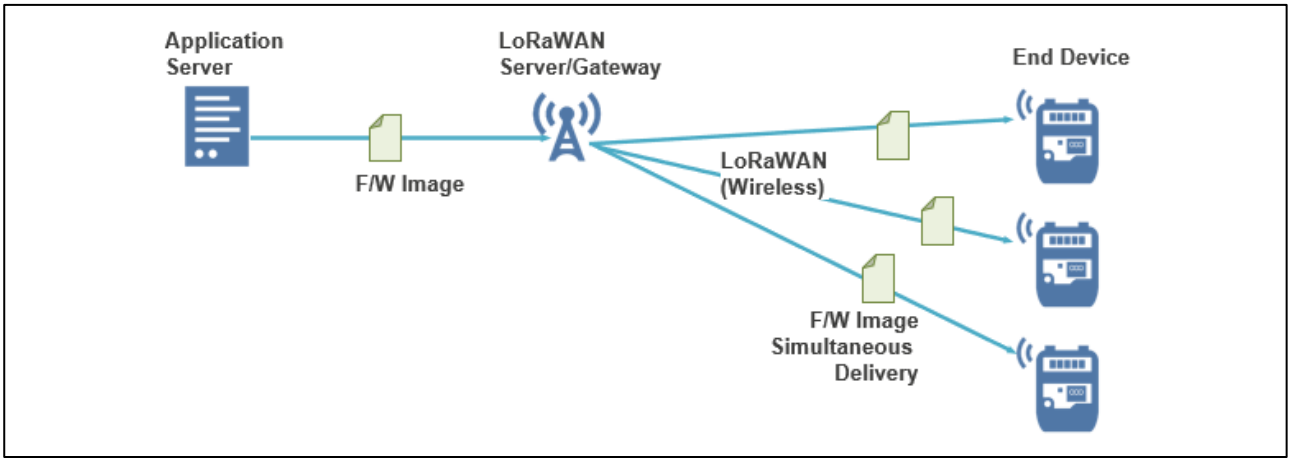


Figure 1 Overview of FUOTA in LoRaWAN Network Architecture

Figure 2 shows the message exchange between LoRaWAN network server and an end device. First, the parameters required for the delivery are set to the end device using the application layer protocols. After that, the F/W image is delivered to the end device via the data fragment message of the fragmented data block transport protocol. The end device reconstructs the fragmented data into the F/W image, updates the internal firmware with the F/W image and reboots itself.

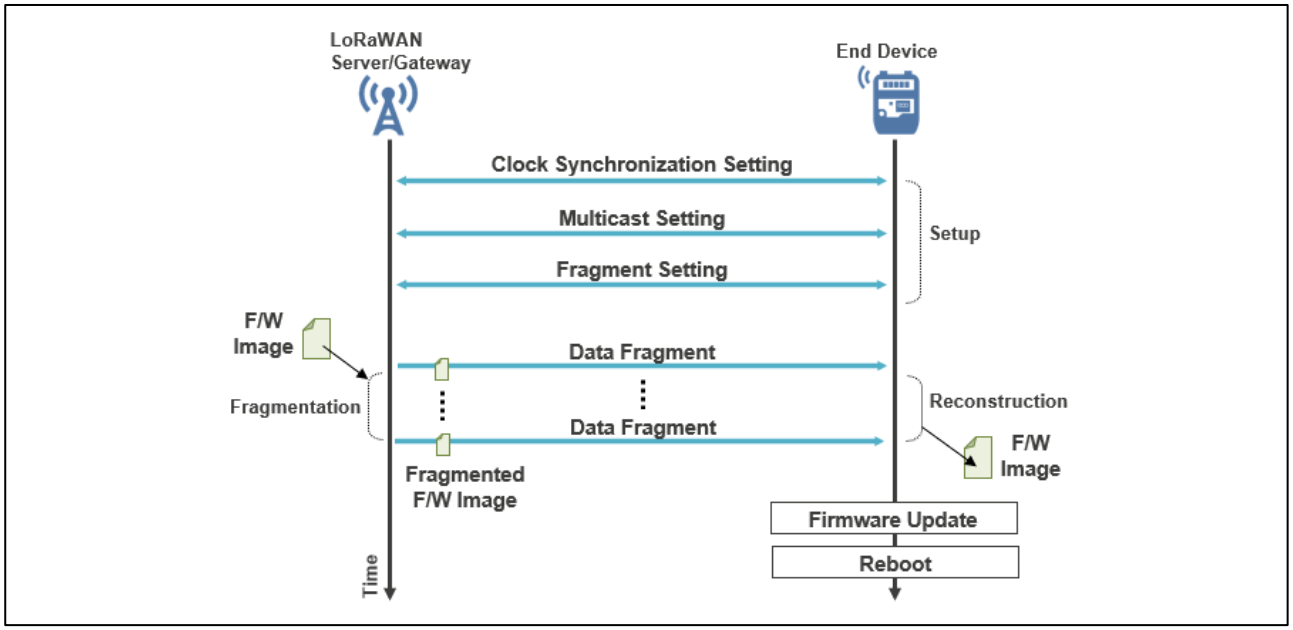


Figure 2 FUOTA Message Exchange between LoRaWAN Network Server and End Device

This application note provides the sample application for FUOTA targeting for the end device based on RA2L1and the Semtech SX1261/62 transceiver for LoRa.

Figure 3 shows overview of the FUOTA process of this sample software for the end device. The FUOTA process can be achieved by two sample applications: The FUOTA sample application and the F/W update sample application.

The FUOTA sample application is to receive a F/W image over the LoRaWAN and its application layer protocols related to FUOTA, and to store it in the internal code flash memory. Once the F/W image is received, it switches to the F/W update sample application by the RA2L1 boot swap function.

The F/W update sample application is to update an end device's firmware using the F/W image. Once the update is completed, it switches to the updated FUOTA sample application by the RA2 boot swap function.

For details, the FUOTA sample application is described in chapter 2, the F/W update sample application is described in chapter 3 and the example operation of the end device is described in chapter 4.

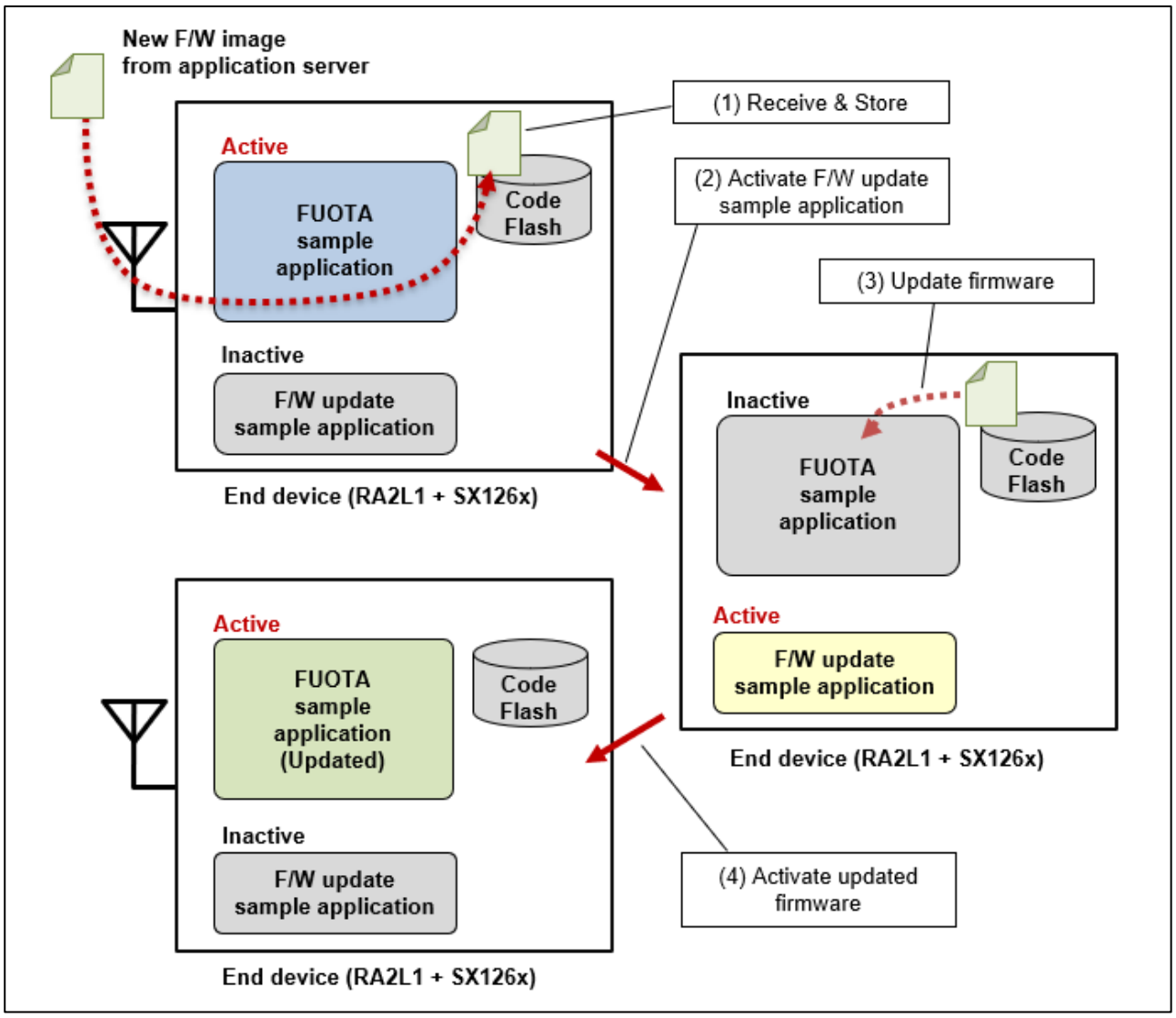


Figure 3 Overview of FUOTA Process of This Sample Application for End Device

In this application note, the FUOTA using the following application packages is described as “FUOTA V1.0.0”.

- Clock Synchronization Message Package v1.0.0
- Remote Multicast Setup Package v1.0.0
- Fragment Data Block Transport Package v1.0.0

In addition, the FUOTA using the following application packages is described as “FUOTA V2.0.0”.

- Clock Synchronization Message Package v2.0.0
- Remote Multicast Setup Package v2.0.0
- Fragment Data Block Transport Package v2.0.0
- Firmware Management Protocol v1.0.0
- Multi Package Access Protocol v1.0.0

The FUOTA sample application supports FUOTA V1.0.0 by default. In addition, the FUOTA sample application experimentally supports FUOTA V2.0.0. Please refer to the chapter 2 to 4 for FUOTA V1.0.0 (hereinafter FUOTA), and Appendix.A for FUOTA V2.0.0 respectively.

1.1 Acronyms and Abbreviations

Table 1. Acronyms and abbreviations

Acronyms	Description
FUOTA	Firmware Update Over-The-Air
FUOTA V1.0.0	In this application note, FUOTA using following application packages is described as “FUOTA V1.0.0” (see Table 3 in 2.1.1). ● Clock Synchronization Message Package v1.0.0 ● Remote Multicast Setup Package v1.0.0 ● Fragmented Data Block Transport Package v1.0.0
FUOTA V2.0.0	In this application note, FUOTA using following application packages is described as “FUOTA V2.0.0” (see Table 16 in A.2.1). ● Clock Synchronization Message Package v2.0.0 ● Remote Multicast Setup Package v2.0.0 ● Fragmented Data Block Transport Package v2.0.0 ● Firmware Management Protocol v1.0.0 ● Multi Package Access Protocol v1.0.0
Boot swap function	Startup area select function. In this application note, this function is described as “boot swap function”.
Boot cluster 0 area	Default area. In this application note, this area is described as “boot cluster 0 area”.
Boot cluster 1 area	Alternate area. In this application note, this area is described as “boot cluster 1 area”.

1.2 Related Documentation

Table 2. Related Documentation

	Document No.	Title	Author	Language
[1]	R11AN0228	LoRaWAN® Stack Reference Guide	Renesas Electronics	English
[2]	R11AN0231	LoRaWAN® Stack Sample Application	Renesas Electronics	English
[3]	R11AN0227	Radio Driver Reference Guide	Renesas Electronics	English
[4]	R11AN0834	Radio Driver Support Functions for Regional Radio Regulations	Renesas Electronics	English
[5]	R30UZ0095	Renesas LPWA Studio	Renesas Electronics	English
[6]	MCP-AA-22-0133-1	RA LoRaWAN® Sensor Demo Tutorial Setup and Operation Method	Renesas Electronics	English
[7]	R11AN0596	RA2E1, RA2L1, RA0E1, RA0E2 LoRa®-based Wireless Software Package	Renesas Electronics	English

2. FUOTA Sample Application

This chapter describes the FUOTA sample application.

2.1 Overview of FUOTA Sample Application

2.1.1 FUOTA Sample Application Block Diagram

Figure 4 shows a block diagram of the FUOTA sample application. This sample application consists of the LoRaWAN stack, FUOTA and application layers. The FUOTA includes the application layer messaging packages over LoRaWAN shown in Table 3.

The clock synchronization package is used to synchronize an end device's clock to the LoRaWAN network's GPS clock. The remote multicast setup package is used to setup Class B or Class C multicast session for a group of end devices. The fragment data block transport package is used to receive fragmented data and reconstruct them into the original data.

These packages can realize an application server simultaneously sends fragments of a firmware image to a group of end devices at the time notified from the server.

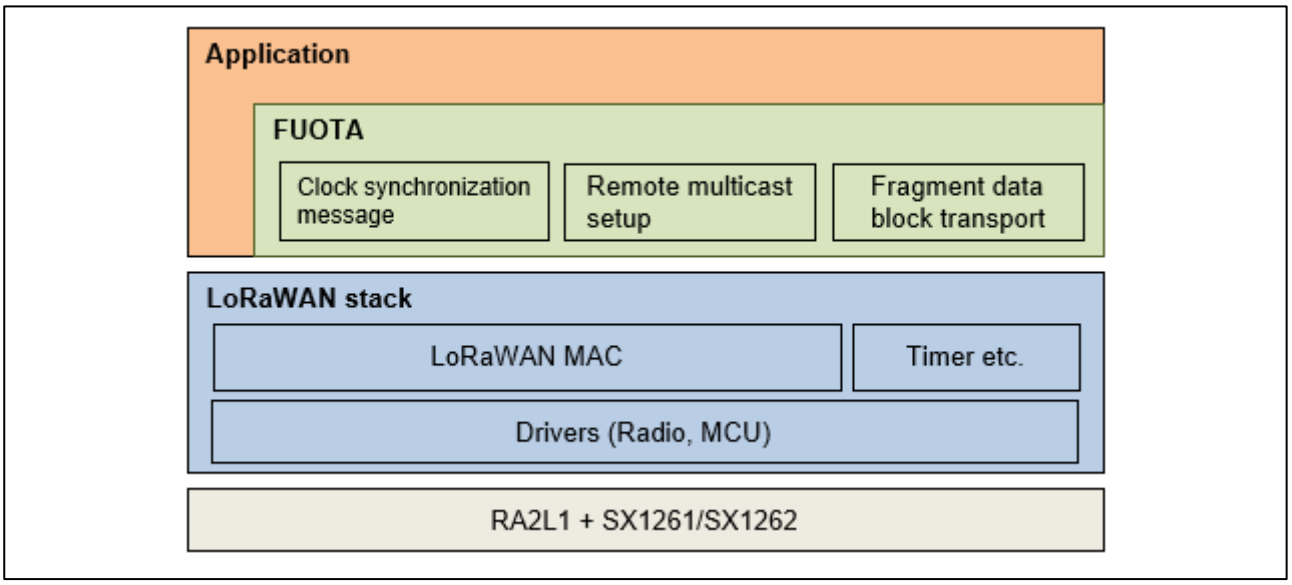


Figure 4 FUOTA Sample Application Block Diagram

Table 3 Application layer messaging package list

Package name	Version	Package ID	Package version	FPort
Clock Synchronization Message Package (*1)	v1.0.0	1	1	202
Remote Multicast Setup Package (*2)	v1.0.0	2	1	200
Fragmented Data Block Transport Package (*3)	v1.0.0	3	1	201

(*1) https://lora-alliance.org/resource_hub/lorawan-application-layer-clock-synchronization-specification-v1-0-0/
(*2) https://lora-alliance.org/resource_hub/lorawan-remote-multicast-setup-specification-v1-0-0/
(*3) https://lora-alliance.org/resource_hub/lorawan-fragmented-data-block-transport-specification-v1-0-0/

2.2 Directories

Table 4 shows a folder structure and what kind of codes are included in each folder.

Table 4 Directories

Directories	Description
src/apps/LoRaFuotaSample	FUOTA sample application
src/apps/FWUpdateSample	FW update sample application codes
src/boards	Board specific codes
src/boards/mcu	MCU drivers
src/mac	LoRaWAN MAC stack
src/radio	Radio driver for LoRa®
src/peripherals	Security related codes
src/system	Utility codes
e2studio/{BOARDS}/LoRaFuotaSample/script/ fsp_LoRaFuotaSample.ld	Linker script file for FUOTA sample application.
e2studio/{BOARDS}/FWUpdateSample/script/ fsp_FWUpdateSample.ld	Linker script file for FW update sample application codes.

* {BOARDS} = ra2l1ek_sx126x

2.2.1 Resource Usage Example

Please refer to [7] in the following folder as for the resource usage such as memory and peripherals.

Folder: `(package top)\documents\`

2.2.2 Software Architecture

Figure 5 shows a software architecture of the FUOTA layer.

The application can request to the FUOTA by the API functions of the FUOTA and receive the notification from the FUOTA by the callback functions of the FUOTA.

The FUOTA processes the received messages of the application layer protocols such as the clock synchronization, the fragment data block and the remote multicast setup according to the frame port number (`FPort`) when those are notified via the callback functions of the LoRaWAN stack. The FUOTA sends back the messages in response to the received messages if necessary via the API functions of the LoRaWAN stack.

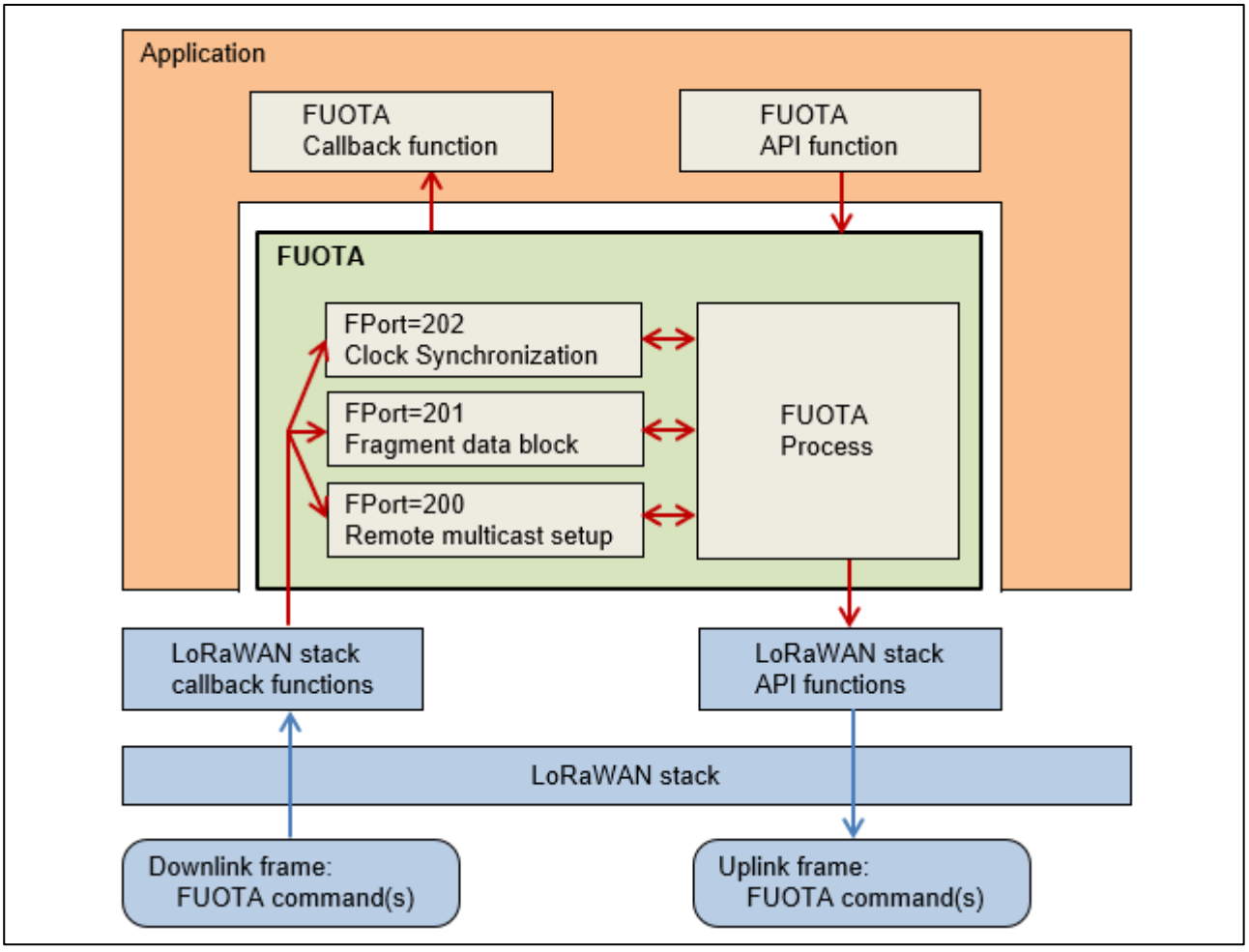


Figure 5 FUOTA Software Architecture

2.3 Macros

2.3.1 FUOTA Setting

Table 5 shows the macros for the FUOTA setting. These macros need to be defined in the project build options.

Table 5 Macros for FUOTA Setting

Macro	Description
FUOTA_ENABLED	Enable FUOTA feature
FUOTA_VERSION_1_0_0	Enable FUOTA V1.0.0. (If omitted, the default version is set to 1.0.0.)

2.3.2 FUOTA Configuration

Table 6 shows the macros for the FUOTA configuration.

These macros need to be specified in the file "`LoRaFuotaConfig.h`".

Table 6 Macros for FUOTA Configuration

Macro	Description *** [ ] indicates related application layer protocol**	
FUOTA_CONFIG_RMTMC_MAX_MC_SESSION	**Type:** uint8_t (1 - 4)	**Default:** 1
	[Remote multicast setup] Maximum number of multicast sessions which can be used simultaneously.	
FUOTA_CONFIG_FRGMNT_MAX_FRAG_SESSION	**Type:** uint8_t (1 - 4)	**Default:** 1
	[Fragment data block] Maximum number of fragment sessions which can be used simultaneously.	
FUOTA_CONFIG_FRGMNT_MAX_DATABLK_SIZE	**Type:** uint32_t (128 or more) (depends on RAM)	**Default:** 4096
	[Fragment data block] Maximum size of a data block to be transported via a fragment session.	
FUOTA_CONFIG_FRGMNT_MAX_NBFRAG	**Type:** uint16_t (1 - 16383) (depends on RAM)	**Default:** 200
	[Fragment data block] Maximum number of fragments of a data block to be transported via a fragment session. Note: Maximum number of fragments (`NbFrag`) can be determined by possible minimum fragment size (`fragSize`): `NbFrag` = (Data block size + `fragSize` - 1) / `fragSize` The `fragSize` is determined by LoRaWAN network server. Please take this into account when setting this configuration.	
FUOTA_CONFIG_FRGMNT_MAX_NBLOST	**Type:** uint16_t (1 - 400) (depends on RAM)	**Default:** 50
	[Fragment data block] Maximum acceptable number of missed fragments of a data block to be transported via a fragment session. If the number of missed fragments exceeds this value, the data block cannot be reassembled.	

2.3.3 Information Base (IB)

Table 7 shows the FUOTA information base (IB).

These are the parameters of FUOTA which can be get or set by the FUOTA API functions. See 2.5.4 and 2.5.5.

Table 7 Macros for FUOTA Information Base (IB)

Macro	Description * [ ] indicate related application layer protocol		
FUOTA_IB_CLKSNC_TIMEREQ_PERIOD_SEC	**Type:** uint32_t (0, 10 - 0x418390)	**Default:** 256	Read/Write
	[Clock synchronization] Periodicity of `AppTimeReq` command transmission in seconds. If it is set to 0, `AppTimeReq` is not transmitted. If it is set to other than 0, `AppTimeReq` is periodically transmitted according to the periodicity. Note: It could be updated when an application server requests to change the periodicity by `DeviceAppTimePeriodicityReq` command.		
FUOTA_IB_CLKSNC_TIMEREQ_MIN_PERIODICITY	**Type:** uint8_t (0 - 15)	**Default:** 0	Read/Write
	[Clock synchronization] Acceptable minimum periodicity of `AppTimeReq` transmission (*1).		
FUOTA_IB_CLKSNC_TIMEREQ_MAX_PERIODICITY	**Type:** uint8_t (0 - 15)	**Default:** 15	Read/Write
	[Clock synchronization] Acceptable maximum periodicity of `AppTimeReq` transmission (*1).		
FUOTA_IB_CLKSNC_FORCESYNC_PERIOD_SEC	**Type:** uint8_t (10 - 255)	**Default:** 60	Read/Write
	[Clock synchronization] Periodicity of `AppTimeReq` transmission in seconds when the specified number of transmissions are requested by `ForceDeviceResyncReq` command.		
FUOTA_IB_CLKSNC_TIMEANS_REQUIRED	**Type:** uint8_t (0, 1)	**Default:** 0	Read/Write
	[Clock synchronization] `AnsRequired` field of `AppTimeReq` to be transmitted from the end device. It indicates whether to request answer in response to `AppTimeReq`. 1: Answer required 0: Answer not required		
FUOTA_IB_PROC_POLLING_PERIOD_SEC	**Type:** uint32_t (0, 10 - 0x418930)	**Default:** 0	Read/Write
	Periodicity of uplink frame transmission to receive downlink frame in seconds. If it is set to 0, uplink frame is not transmitted. If it is set to other than 0, uplink frame is periodically transmitted according to the period. Note: Periodical uplink frame transmission is suspended during multicast session is active.		
FUOTA_IB_PROC_POLLING_FPORT	**Type:** uint8_t (1 - 223, except 200 - 202)	**Default:** 223	Read/Write
	Frame port (`FPort`) value used for the uplink frame transmitted according to the setting of `FUOTA_IB_PROC_POLLING_PERIOD_SEC`. Note: It cannot be set 200, 201, and 202 because application layer packages use them.		

(*1) Actual periodicity of `AppTimeReq` transmission in seconds is 128 * (2 ^ Period) +/- rand(30).

2.4 Enumerations

2.4.1 FuotaStatus_t

This type is an enumeration containing the status of the operation of a FUOTA service.

Table 8 FuotaStatus_t

Enumerator	Description
FUOTA_STATUS_OK	Service processed successfully
FUOTA_STATUS_ERROR	Error - FUOA process was failed
FUOTA_STATUS_BUSY	Error - FUOTA and/or LoRaWAN stack is busy for other operations
FUOTA_STATUS_SERVICE_UNKNOWN	Error - Unknown request
FUOTA_STATUS_PARAMETER_INVALID	Error - Invalid parameter
FUOTA_STATUS_IB_READONLY	Error - IB is read only

2.5 FUOTA APIs

This section describes the API functions of FUOTA shown in Table 9.

Table 9 FUOTA APIs

Function	Description
FuotaInit	Initialize FUOTA.
FuotaStart	Start FUOTA
FuotaStop	Stop FUOTA
FuotaIbGetRequest	Information Base service to get attribute of FUOTA.
FuotaIbSetRequest	Information Base service to set attribute of FUOTA.
FuotaProcess	Process FUOTA interruption.
FuotaMcpsConfirm	Process MCPS-Confirm related to FUOTA.
FuotaMcpsIndication	Process MCPS-Indication related to FUOTA.
FuotaMlmeConfirm	Process MLME-Confirm related to FUOTA.
FuotaMlmeIndication	Process MLME-Indication related to FUOTA.

2.5.1 FuotaInit

FuotaStatus_t FuotaInit(FuotaEventCb_t *p_fuotaEventCb)		
This function initializes FUOTA. Event handler functions shall be specified in '`p_fuotaEventCb`'. Please call it before calling other FUOTA API functions.		
Parameters:		
p_fuotaEventCb	Input	Pointer to the structure to set the FUOTA event handler functions. See 2.6 for details.
Return:		
FUOTA_STATUS_OK		Request is finished successfully.
FUOTA_STATUS_PARAMETER_INVALID		Requested parameter is invalid.

2.5.2 FuotaStart

void FuotaStart(void)
This function starts FUOTA.
Parameters:
None
Return:
None

2.5.3 FuotaStop

void FuotaStop(void)		
This function stops FUOTA and initializes the information inside FUOTA except for the information base. It can be used to prevent RA2L1 boot swap process from being interrupted by FUOTA.		
Parameters:		
None		
Return:		
None		

2.5.4 FuotaIbGetRequest

FuotaStatus_t FuotaIbGetRequest(uint8_t ib, void *vpVal)		
This function is the FUOTA information base (IB) service to get attributes of the FUOTA. See 2.3.3 for the IDs and types of IB.		
Parameters:		
ib	Input	ID of the information base
*vpVal	Output	Destination of the attribute value

Return:	
FUOTA_STATUS_OK	Request is finished successfully.
FUOTA_STATUS_ERROR	Request cannot be accepted.
FUOTA_STATUS_PARAMETER_INVALID	Requested parameter is invalid.
FUOTA_STATUS_SERVICE_UNKNOWN	Requested IB is unknown.

2.5.5 FuotaIbSetRequest

FuotaStatus_t FuotaIbSetRequest(uint8_t ib, void *vpVal)		
This function is the FUOTA information base (IB) service to set attributes of the FUOTA. See 2.3.3 for the IDs and types of IB.		
Parameters:		
ib	Input	ID of the information base
*vpVal	Input	Source of the attribute value

Return:	
FUOTA_STATUS_OK	Request is finished successfully.
FUOTA_STATUS_ERROR	Request cannot be accepted.
FUOTA_STATUS_PARAMETER_INVALID	Requested parameter is invalid.
FUOTA_STATUS_SERVICE_UNKNOWN	Requested IB is unknown.
FUOTA_STATUS_IB_READONLY	Requested IB is read-only.
FUOTA_STATUS_BUSY	MAC is busy. Another service is running.

2.5.6 FuotaProcess

void FuotaProcess(void)		
This function processes pending events of FUOTA. Application shall periodically call this function in its main loop as short an interval as possible. Please call this function right after the `LoRaMacProcess()` function. (`LoRaMacProcess()` is an API function of LoRaWAN stack. See [1])		
Parameters:		
None		
Return:		
None		

2.5.7 FuotaMcpsConfirm

void FuotaMcpsConfirm(McpsConfirm_t *p_mcpsConfirm)		
This function processes the MCPS-Confirm message if it is related to FUOTA. Please call this function at the beginning of the MCPS-Confirm callback function. (See [1] about MCPS-Confirm callback function.)		
Parameters:		
p_mcpsConfirm	Input	Pointer to MCPS-Confirm message, which is an argument of MCPS-Confirm callback function.
Return:		
None		

2.5.8 FuotaMcpsIndication

FuotaStatus_t FuotaMcpsIndication(McpsIndication_t *p_mcpsIndication)		
This function processes the MCPS-Indication message if it is related to FUOTA. Please call this function at the beginning of the MCPS-Indication callback function. (See [1] about MCPS-Indication callback function.)		
Parameters:		
p_mcpsIndication	Input	Pointer to MCPS-Indication message, which is an argument of MCPS-Indication callback function.
Return:		
FUOTA_STATUS_OK		The request is finished successfully.
FUOTA_STATUS_ERROR		Request cannot be accepted.
FUOTA_STATUS_BUSY		FUOTA is busy. Another service is running.

2.5.9 FuotaMlmeConfirm

void FuotaMlmeConfirm(MlmeConfirm_t *p_mlmeConfirm)		
This function processes the MLME-Confirm message if it is related to FUOTA. Please call this function at the beginning of the MLME-Confirm callback function. (See [1] about MLME-Confirm callback function.)		
Parameters:		
p_mlmeConfirm	Input	Pointer to MLME-Confirm message, which is an argument of MLME-Confirm callback function.
Return:		
None		

2.5.10 FuotaMlmeIndication

void FuotaMlmeIndication(MlmeIndication_t *p_mlmeIndication)		
This function processes the MLME-Indication message if it is related to FUOTA. Please call this function at the beginning of the MLME-Indication callback function. (See [1] about MLME-Indication callback function.)		
Parameters:		
p_mlmeIndication	Input	Pointer to MLME-Indication message, which is an argument of MLME-Indication callback function.
Return:		
None		

2.6 Callback Handler Functions (FuotaEventCb_t)

`FuotaEventCb_t` is a structure containing FUOTA event handler functions to notify application layers of the events.

Table 10 FuotaEventCb_t

Member (callback handler functions)	Description
void (*FuotaRmtMcSessionSetupIndication)(DeviceClass_t sessionClass, uint8_t mcGroupId, uint32_t timeToStartSec, uint32_t timeoutSec)	Pointer to callback function to be called when the time to start/end of the multicast session is scheduled.
void (*FuotaRmtMcSessionStartIndication)(DeviceClass_t sessionClass, uint8_t mcGroupId, uint32_t timeoutSec)	Pointer to callback function to be called when a multicast session is started.
void (*FuotaRmtMcSessionEndIndication)(DeviceClass_t sessionClass, uint8_t mcGroupId)	Pointer to callback function to be called when a multicast session end is ended.
FuotaStatus_t (*FuotaFrgmntSessionSetupIndication)(uint8_t fragIndex, uint32_t descriptor)	Pointer to callback function to be called before starting a fragment session.
void (*FuotaFrgmntDataBlockIndication)(uint8_t fragIndex, uint8_t *p_dataBlk, uint32_t dataBlkSize)	Pointer to callback function to be called when a data block is received.
void (*FuotaFrgmntSessionEndIndication)(uint8_t fragIndex)	Pointer to callback function to be called when a fragment session is ended.

2.6.1 FuotaRmtMcSessionSetupIndication

void (*FuotaRmtMcSessionSetupIndication)(DeviceClass_t sessionClass, uint8_t mcGroupId, uint32_t timeToStartSec, uint32_t timeoutSec)

This function will be called when the time to start/end of the multicast session is scheduled on reception of '`MulticastClassCSessionReq`' or '`MulticastClassBSessionReq`' command.
FUOTA will switch the device class to Class B or Class C when the multicast session is started. So, the application needs to prepare especially in case that the device class will be switched to Class B. If the application operates in Class A, it has to request the beacon acquisition to the LoRaWAN stack and start beacon tracking until multicast session is started.

Parameters:

sessionClass	Input	Class of multicast session; CLASS_C or CLASS_B.
mcGroupId	Input	Group ID
timeToStartSec	Input	Time to start multicast session in seconds.
timeoutSec	Input	Timeout of the session from start in seconds.

Return:

None

2.6.2 FuotaRmtMcSessionStartIndication

void (*FuotaRmtMcSessionStartIndication)(DeviceClass_t sessionClass, uint8_t mcGroupId, uint32_t timeoutSec)		
This function notifies when the multicast session is started, and the device class is changed to the class of multicast session.		
Parameters:		
sessionClass	Input	Class of multicast session; CLASS_C or CLASS_B.
mcGroupId	Input	Group ID
timeoutSec	Input	Timeout of the session in seconds.
Return:		
None		

2.6.3 FuotaRmtMcSessionEndIndication

void (*FuotaRmtMcSessionEndIndication)(DeviceClass_t sessionClass, uint8_t mcGroupId)		
This function notifies when multicast session is ended, and the device class is returned to the class before the multicast session is started.		
Parameters:		
sessionClass	Input	Class of multicast session; CLASS_C or CLASS_B.
mcGroupId	Input	Group ID
Return:		
None		

2.6.4 FuotaFrgmntSessionSetupIndication

FuotaStatus_t (*FuotaFrgmntSessionSetupIndication)(uint8_t fragIndex, uint32_t descriptor)		
This function will be called when the parameters used for the fragment session are notified on reception of '`FragSessionSetupReq`' command by FUOTA. Application has to check parameters and decide if the fragment session can be started. The result of the decision is set to the status parameter of '`FragSessionSetupAns`' command.		
Parameters:		
fragIndex	Input	Index of fragment session.
descriptor	Input	Descriptor; parameter in '`FragSessionSetupReq`' command. This parameter is vendor specific. So, please check it if necessary.
Return:		
FUOTA_STATUS_OK		Fragment session can be started.
FUOTA_STATUS_ERROR		Fragment session cannot be started.

2.6.5 FuotaFrgmntDataBlockIndication

void (*FuotaFrgmntDataBlockIndication)(uint8_t fragIndex, uint8_t *p_dataBlk, uint32_t dataBlkSize)		
This function notifies the reception of a data block. Application can store the data block to the internal code flash memory. See 2.8.2 for details.		
Parameters:		
fragIndex	Input	Index of fragment session.
p_dataBlk	Input	Pointer to the received data block.
dataBlkSize	Input	Size of the received data block.
Return:		
None		

2.6.6 FuotaFrgmntSessionEndIndication

void (*FuotaFrgmntSessionEndIndication)(uint8_t fragIndex)		
This function notifies when the fragment session is ended and deleted.		
Parameters:		
fragIndex	Input	Index of fragment session.
Return:		
None		

2.7 FUOTA Related Commands Sequence, Usage of API and Callback Functions

2.7.1 Flow of FUOTA Processing

Figure 6 shows a basic flow diagram of FUOTA process.

After the FUOTA is initialized and started, the FUOTA related commands can be processed by passing the MCPS indication (downlink data) notified from the LoRaWAN stack.

Application needs to call `FuotaProcess()` function periodically for FUOTA to process its events; the FUOTA related command transmissions and timer interruptions.

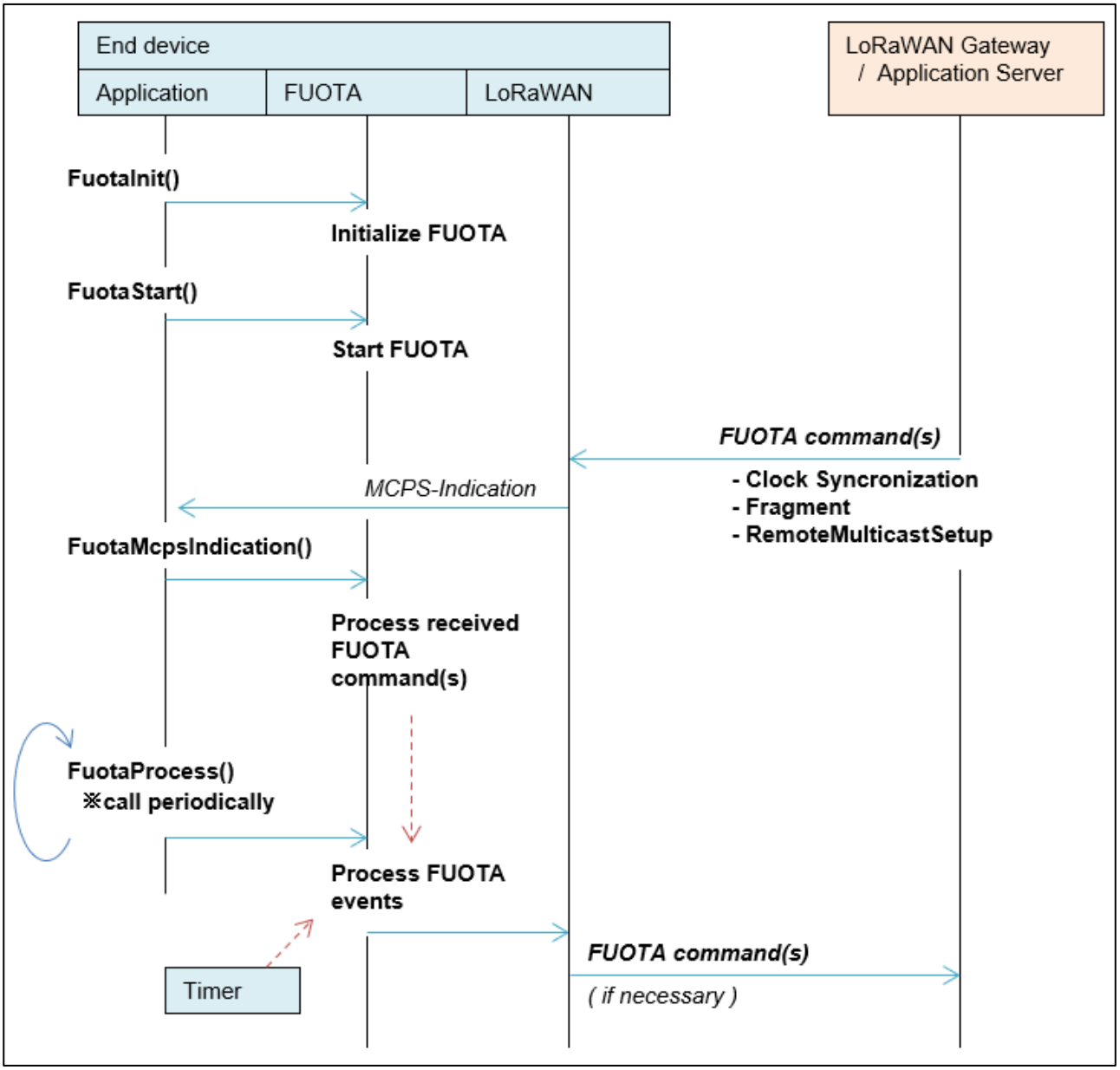


Figure 6 Flow of FUOTA Processing

2.7.2 Clock Synchronization

Figure 7 shows a flow diagram of the clock synchronization between an end-device's clock and the LoRaWAN network's GPS based clock.

When the FUOTA is started, FUOTA starts to send `AppTimeReq` command periodically according to the IB `FUOTA_IB_CLKSNC_TIMEREQ_PERIOD_SEC`. See 2.3.3 for details.

The FUOTA controls the process of the clock synchronization, and no event is notified to the application.

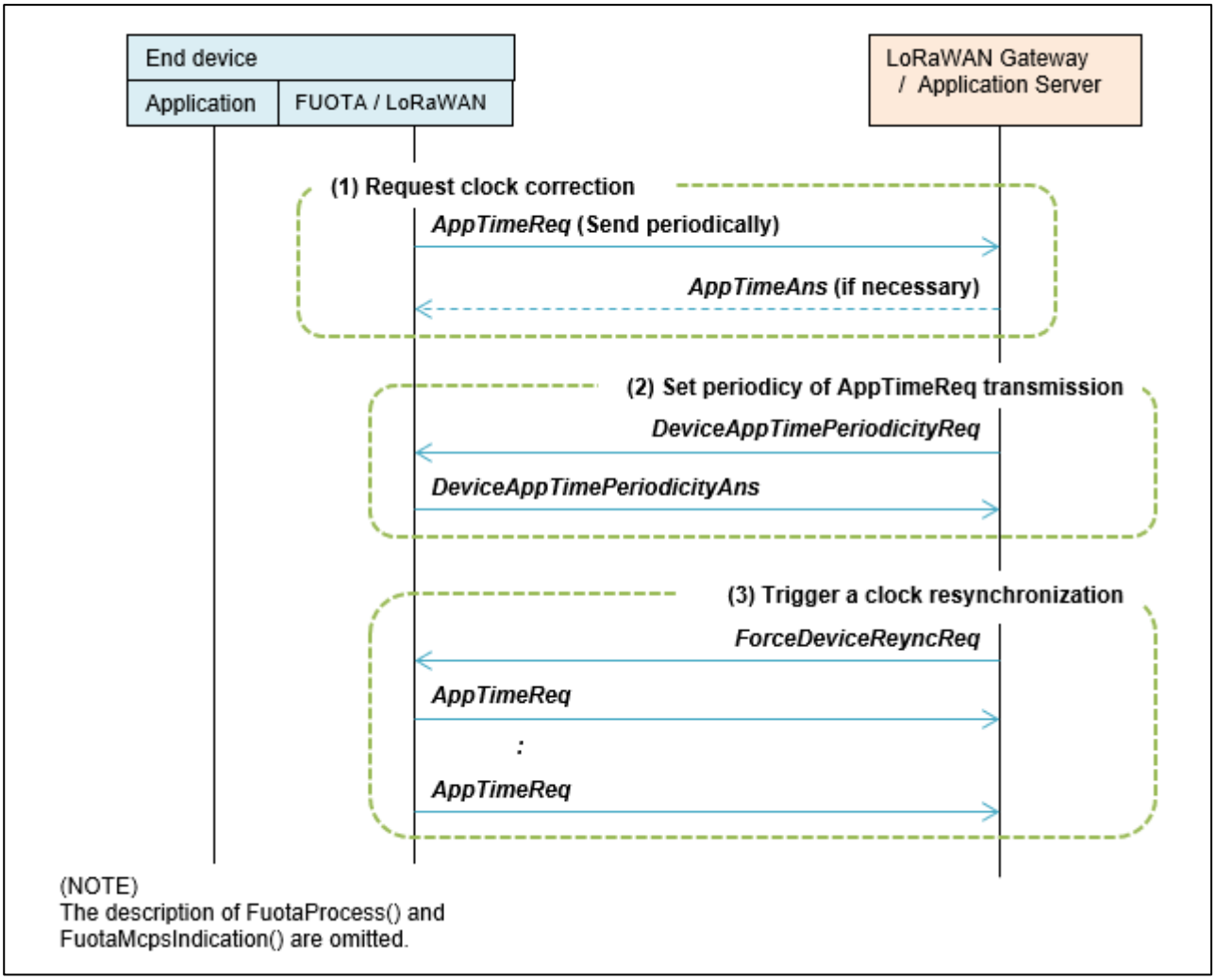


Figure 7 Clock Synchronization

2.7.3 Remote Multicast Setup

Figure 8 shows a flow diagram of the remote multicast setup.

The FUOTA notifies the application of the start and the end time of the multicast session when the application server requests the end device to schedule the start and end of a multicast session.

There are two type of multicast sessions: Class C and Class B. When a multicast session is started, FUOTA switches the device class to Class C or Class B. So, the application needs to prepare to start the multicast session; especially in case of Class B session, application which operates in Class A has to request the beacon acquisition to the LoRaWAN stack and start beacon tracking before the multicast session is started.

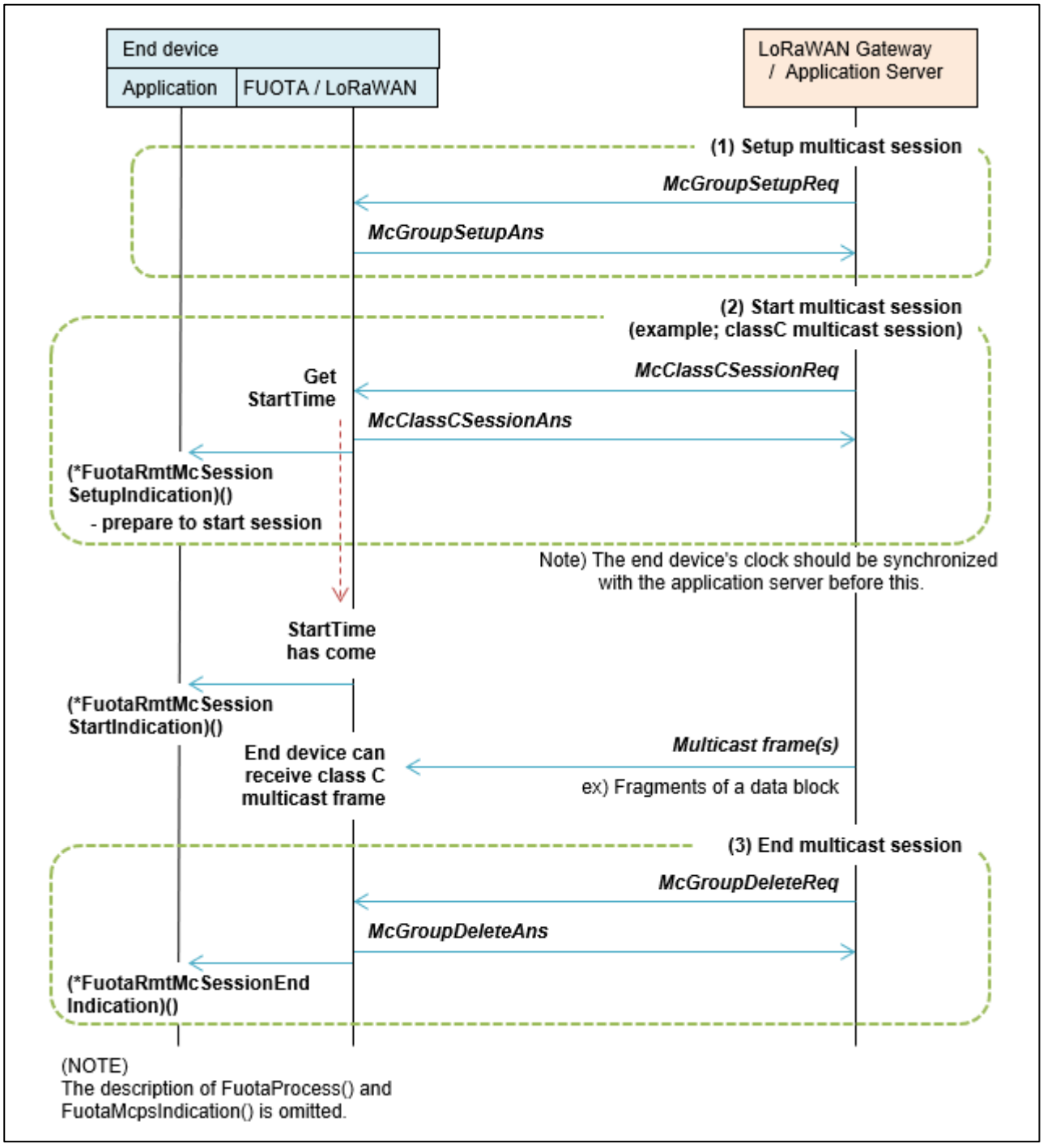


Figure 8 Remote Multicast Setup

2.7.4 Fragment Data Block Transport

Figure 9 shows a flow diagram of the fragment data block transport.

The FUOTA notifies the application of the start and end of the fragment session when the application server requests the end device to start and end a fragment session. Also, FUOTA notifies the application of the reception of a data block when it is reconstructed by the fragments during the fragment session.

The application needs to store the data block to the code flash memory to update the firmware later. See 2.8 for details.

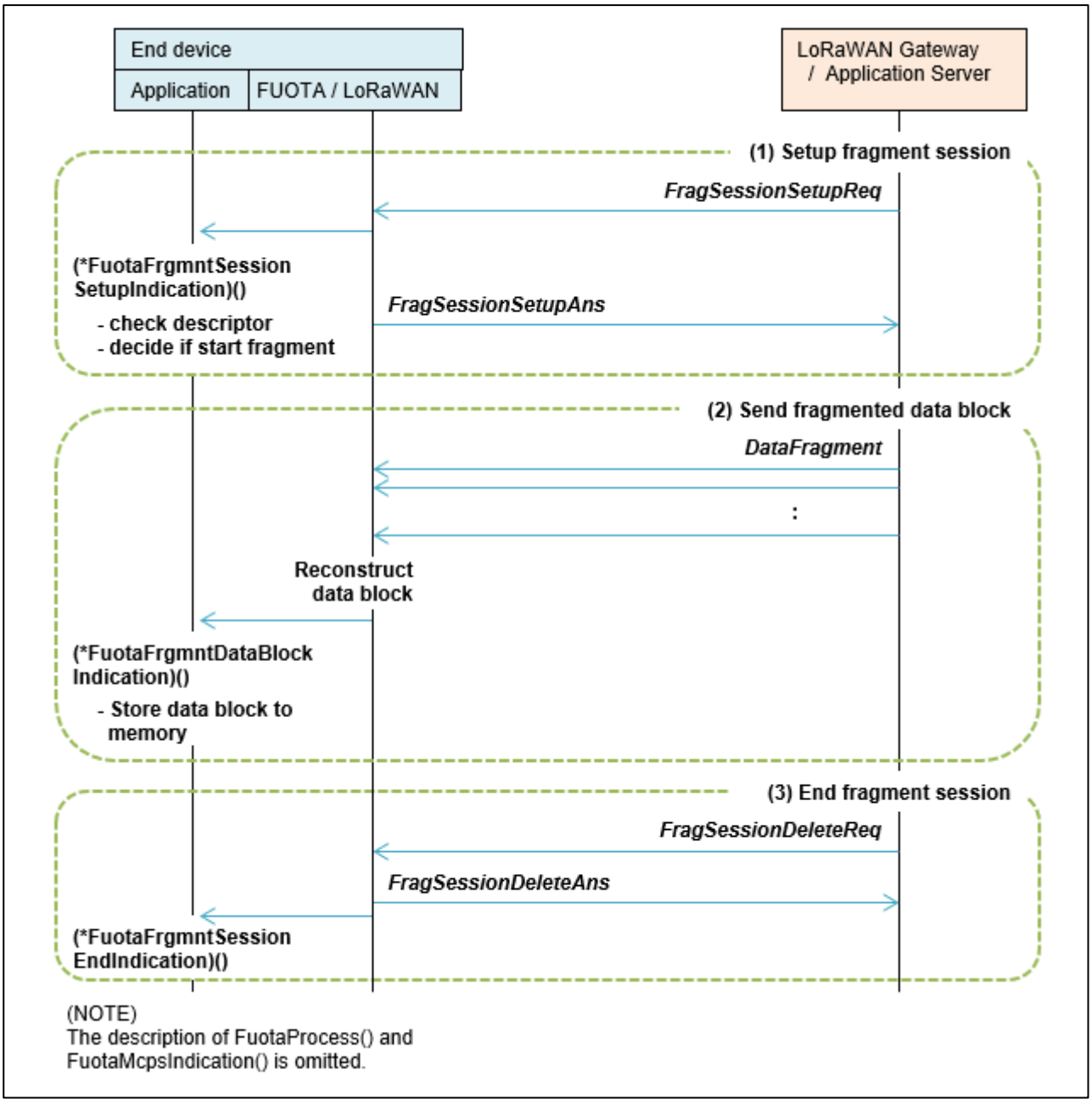


Figure 9 Fragment Data Block

2.8 FUOTA Sample Application

This section describes about the application layer specification of the FUOTA sample application software.

2.8.1 Overview

Figure 10 shows the sequence of the firmware update process according to the following (Step 1) to (Step 4).

The FUOTA sample application supports the following (Step 1) and (Step 2), and the F/W update sample application supports the following (Step 3) and (Step 4). There are functions prepared to write the data blocks to the code flash memory for (Step 1), and to activate the F/W update sample application for (Step 2). Refer to the section 2.8.2 and 2.8.3 respectively.

The FUOTA sample application can be controlled by the AT commands defined in [2] and additional FUOTA related AT commands. Refer to the section 2.8.4 for details.

(Step 1) Receives the new F/W image and stores it to the code flash memory [see 2.8.2]

The FUOTA sample application starts to receive the new F/W image from the application server. The FUOTA layer processes the received F/W image, which could consist of some data blocks.

The application layer stores the data blocks notified from the FUOTA to the internal code flash memory.

(Step 2) Validates the new F/W image and activates the F/W update sample application [see 2.8.3]

After the validation of the stored new F/W image, the FUOTA sample application activates the F/W update sample application by the RA2L1 boot swap function. The F/W update sample application is supposed to be pre-programed in the code flash memory.

(Step 3) Update the firmware using the new F/W image [see chapter 3]

The F/W update sample application validates the stored new F/W image and updates the internal firmware using the F/W image.

(Step 4) Activates the updated firmware [see chapter 3]

After the updates of the firmware, the F/W update sample application activates the updated FUOTA sample application by the RA2L1 boot swap function.

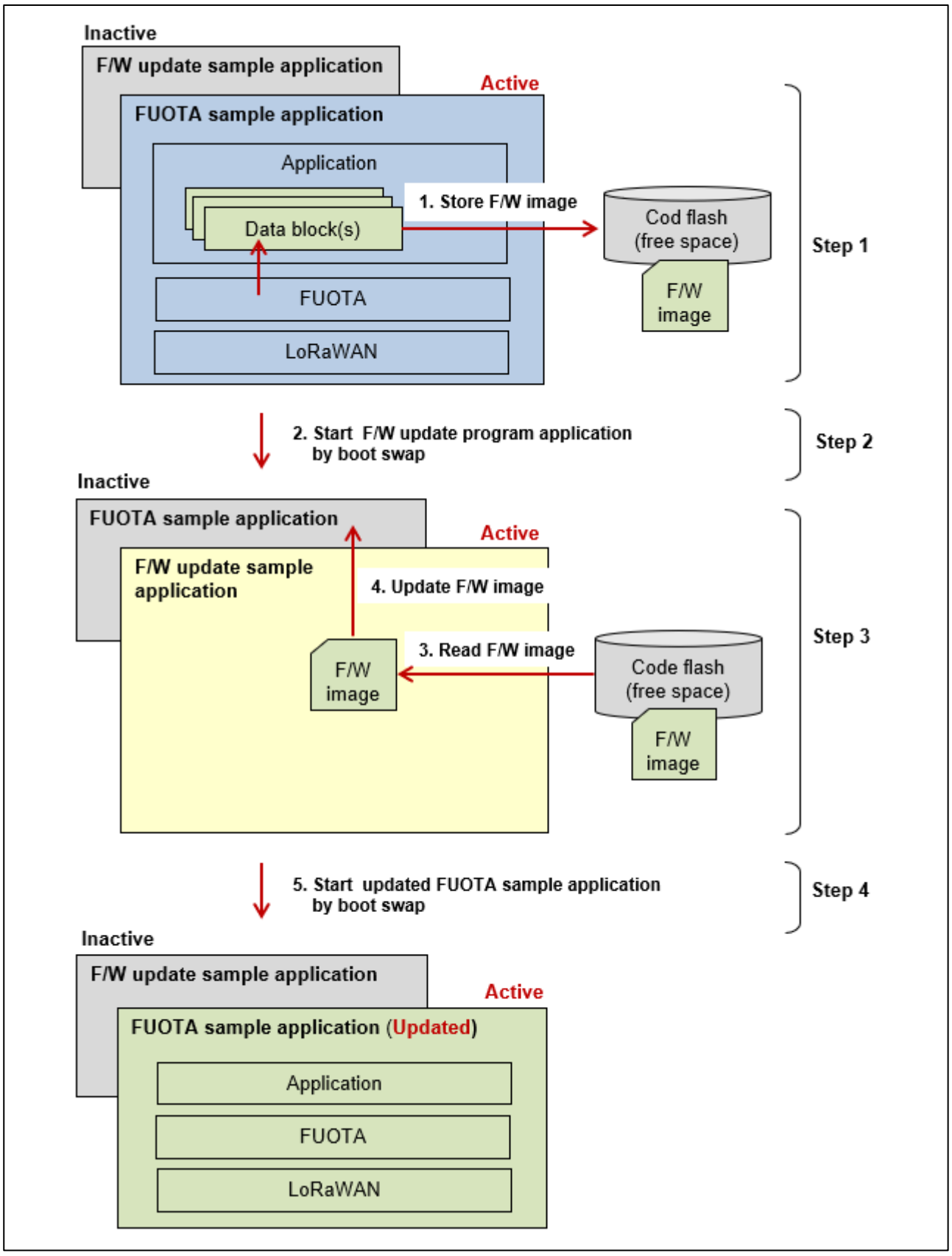


Figure 10 Sequence of Firmware Update

2.8.2 Functions to Write F/W Image to Code Flash Memory

The FUOTA sample application receives the data blocks divided from the new F/W image via the fragment sessions. After that, it needs to write the data blocks to the internal code flash memory.

There is the function prepared for the application to write the data blocks. Figure 11 shows the usage of the functions.

FuotaUpdateStatus_t AppFuotaUpdateStoreFwImage(uint8_t *p_dataBlk, uint16_t dataSize)

This function writes a data block indicated by the callback function `FuotaFrgmntDataBlockIndication()` to the code flash memory.
Please call this function within the callback function. See 2.6.5.

Parameters:

p_dataBlk	Input	Specify the pointer to the received data block, which is the 2nd argument of the callback function `FuotaFrgmntDataBlockIndication()`.
dataSize	Input	Specify the size of the received data block, which is the 3rd argument of the callback function `FuotaFrgmntDataBlockIndication()`.

Return:

FUOTAUPDT_STATUS_OK (= FUOTA_STATUS_OK)	Processed successfully.
FUOTAUPDT_STATUS_ERROR (= FUOTA_STATUS_ERROR)	Process was failed.

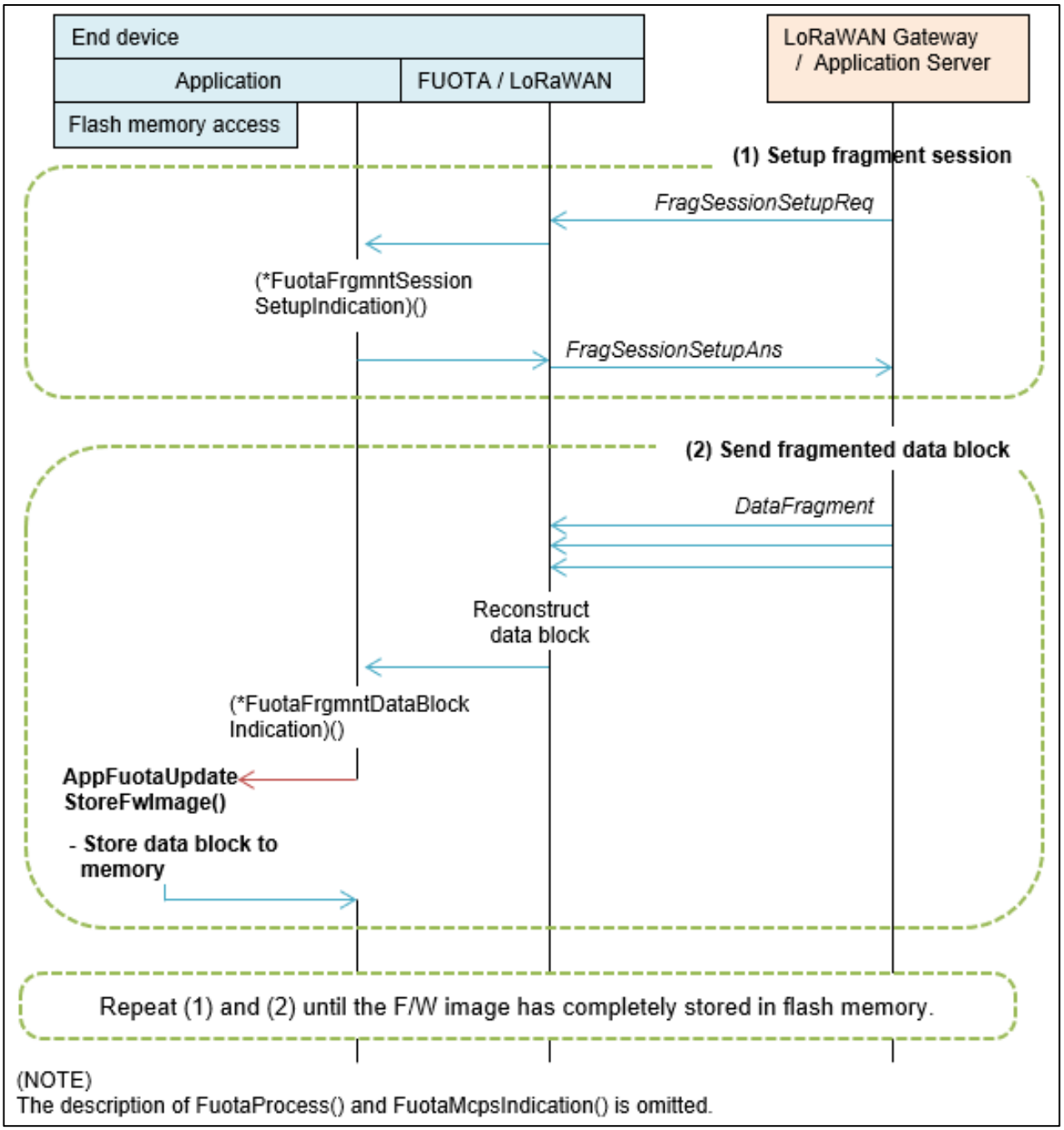


Figure 11 Example Usage of Functions to Write F/W Image to Code Flash Memory

2.8.3 Functions to Activate F/W Update Sample Application

There are two functions for application to get the status of the FUOTA process and activate the F/W update sample application. Figure 12 shows the example of the function usage.

uint8_t AppFuotaUpdateGetStatus(void)	
This function gets the status of storing F/W image	
Parameters:	
None	
Return:	
FUOTAUPDT_STATE_SUCCESS	Complete F/W image is written to the code flash memory.
FUOTAUPDT_STATE_NONE	No F/W image.
FUOTAUPDT_STATE_INITIAL	No F/W image. A fragment session is requested to be setup.
FUOTAUPDT_STATE_RUNNING	F/W image is not yet completed. Some data blocks of F/W image are written to the code flash memory.
FUOTAUPDT_STATE_FAILED	● Failed to write F/W image to the code flash memory. ● Address of the F/W image storage area is invalid. ● Information in F/W image is invalid. ● Divided F/W images were not delivered in order (see 4.2.3).

FuotaUpdateStatus_t AppFuotaUpdateStartFwUpdate(AppFuotaPreUpdate_t p_preUpdateCbFunc)		
This function activates the F/W update sample application if the complete F/W image is stored in the code flash memory, and it is validated successfully. This function is returned only if the RA2L1 boot swap to activate the F/W update sample application could not be performed.		
Parameters:		
p_preUpdateCbFunc	Input	Callback function that is called before starting F/W update sample application by the RA2L1 boot swap. If NULL is specified, the callback function is not called.
Return:		
FUOTAUPDT_STATUS_BUSY (= FUOTA_STATUS_BUSY)		LoRaWAN stack is busy for other processing. Note: Validation of F/W image was successfully.
FUOTAUPDT_STATUS_ERROR (= FUOTA_STATUS_ERROR)		Failed to activate the F/W update sample application. Note: The detail reason for the failure can be retrieved with `AppFuotaUpdateGetStatus()` function. If this return value is FUOTAUPDT_STATE_SUCCESS, it indicates the validation of the F/W image was failed.

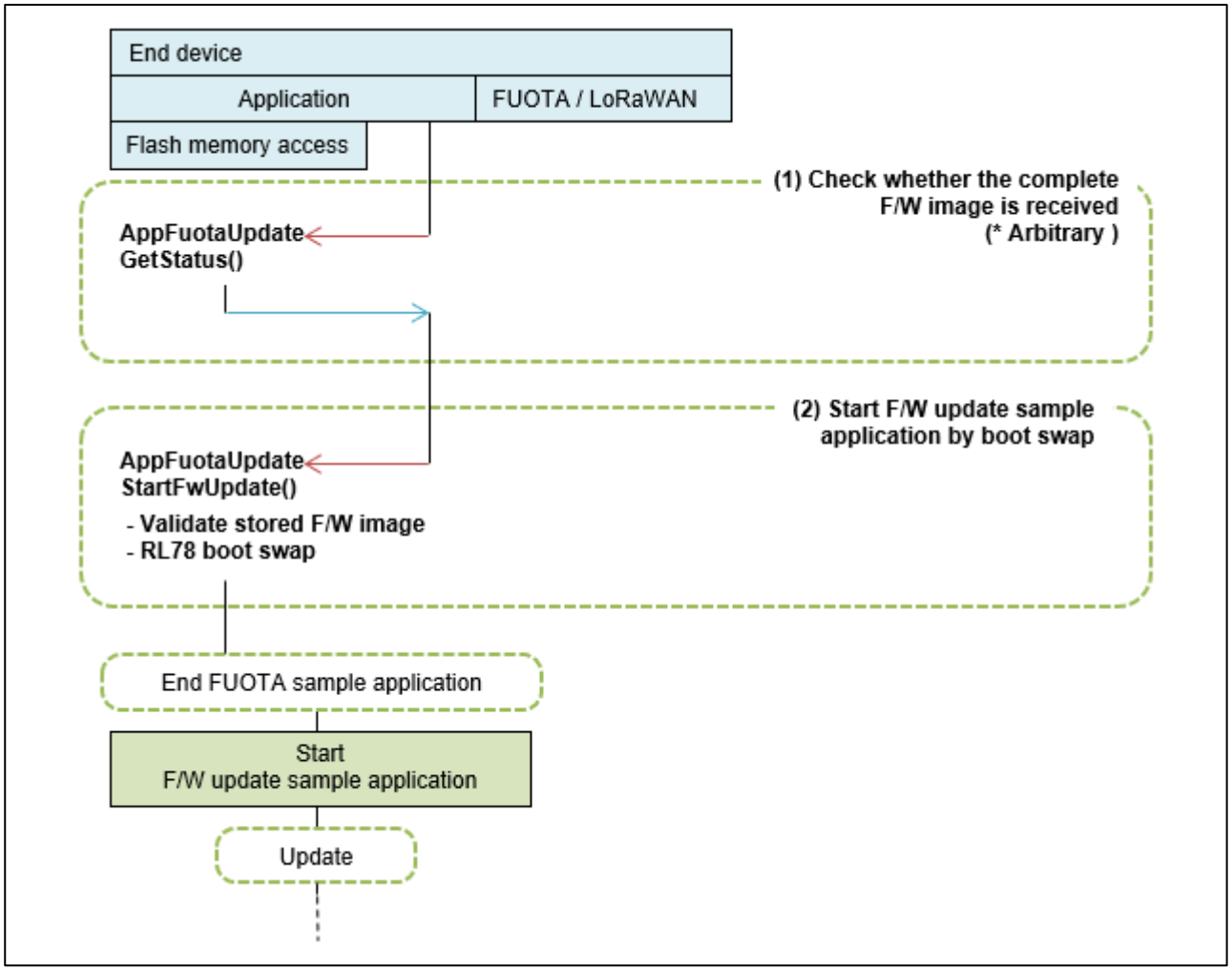


Figure 12 Example Usage of Functions to Activate F/W Update Sample Application

2.8.4 AT Commands for the FUOTA Sample Application

This section describes about AT commands for the FUOTA sample application.

The FUOTA sample application can be controlled by the AT commands defined in [2] and additional FUOTA related AT commands specified as follows.

Table 11 AT commands request for FUOTA

AT Command	Description
AT+FUOTASTART	● Enables FUOTA related application protocols and starts FUOTA process by calling API function `FuotaStart()`. ● This command shall be issued after network join.
AT+FUOTASTOP	● Disables FUOTA related application protocols and stops FUOTA process by calling API function `FuotaStop()`.
AT+FUOTASET=<IB>,<Val> <IB> ID of IB (Hexadecimal without prefix) <Val> Value of IB (Decimal or 16-byte Hexadecimal without prefix)	● Sets a FUOTA related IB by calling API function `FuotaIbSetRequest()`.
AT+FUOTAGET=<IB> <IB> ID of IB (Hexadecimal without prefix)	● Gets a FUOTA related IB by calling API function `FuotaIbGetRequest()`.
AT+FUOTAUPDT	● Activates the F/W update sample application by the RL78 boot swap function and update the FUOTA sample application using the F/W image stored in the code flash memory. If MAC is not idle at this timing, this AT command will return BUSY error. ● After the update, activates the updated FUOTA sample application by the RL78 boot swap function.
AT+GENAPPKEY=<genappkey> <genappkey> GenAppkey in 16 bytes hexadecimal value (32 characters).	● Sets `GenAppkey` to LoRaWAN stack. ● See [2] for more details.

Table 12 ID of IB to set/get by AT+FUOTASET and AT+FUOTAGET

IB	ID	Example
FUOTA_IB_CLKSNC_TIMEREQ_PERIOD_SEC	0x10	`AT+FUOTASET=10,100` Set to 100 seconds as transmission interval of `AppTimeReq`
FUOTA_IB_CLKSNC_TIMEREQ_MIN_PERIODICITY	0x12	`AT+FUOTASET=12,1` Set to 1 as the minimum periodicity of `AppTimeReq` transmission
FUOTA_IB_CLKSNC_TIMEREQ_MAX_PERIODICITY	0x13	`AT+FUOTASET=13,14` Set to 14 as the minimum periodicity of `AppTimeReq` transmission
FUOTA_IB_CLKSNC_FORCESYNC_PERIOD_SEC	0x15	`AT+FUOTASET=15,200` Set 200 seconds as transmission interval of `AppTimeReq` when `ForceDeviceResyncReq` is requested.
FUOTA_IB_PROC_POLLING_PERIOD_SEC	0xF0	`AT+FUOTASET=F0,300` Set 300 seconds as transmission interval of polling uplink to get downlink
FUOTA_IB_PROC_POLLING_FPORT	0xF1	`AT+FUOTASET=F1,55` Set `FPort` to 55 for the polling uplink to get downlink

Table 13 AT command indication from FUOTA

AT command	Description
+FUOTAIND:<indication>,<param1>,<param2>,<param3>,<param4> <indication> FUOTA evet indication type <param1> - <param4> Depends on indication value.	Event indication from FUOTA layer ● When `<indication>` is 0: Indicates an event of the RemoteMulticast session setup. `<param1>` Session Class ID. 1:ClassB / 2:ClassC `<param2>` Multicast Group ID `<param3>` Seconds to start session `<param4>` Seconds to timeout session ● When `<indication>` is 1: Indicates an event of the RemoteMulticast session start. `<param1>` Session Class ID. 1:ClassB / 2:ClassC `<param2>` Multicast Group ID `<param3>` Seconds to timeout session ● When `<indication>` is 2: Indicates an event of the RemoteMulticast session end. `<param1>` Session Class ID. 1:ClassB / 2:ClassC `<param2>` Multicast Group ID ● When `<indication>` is 128: Indicates an event that F/W image is ready. — `<param1>` - `<param4>` are omitted.

3. F/W Update Sample Application

3.1 Overview of F/W Update Sample Application

Figure 13 shows the sequence of the firmware update.

The F/W update sample application is activated after the F/W image is received and stored in the internal flash by the FUOTA sample application.

The F/W update sample application processes the following steps.

(Step 1) Update the firmware of the FUOTA sample application by referring to the information contained in the F/W image. [See 3.2 and 3.3]

(Step 2) Activate the updated firmware by the RA2L1 boot swap.

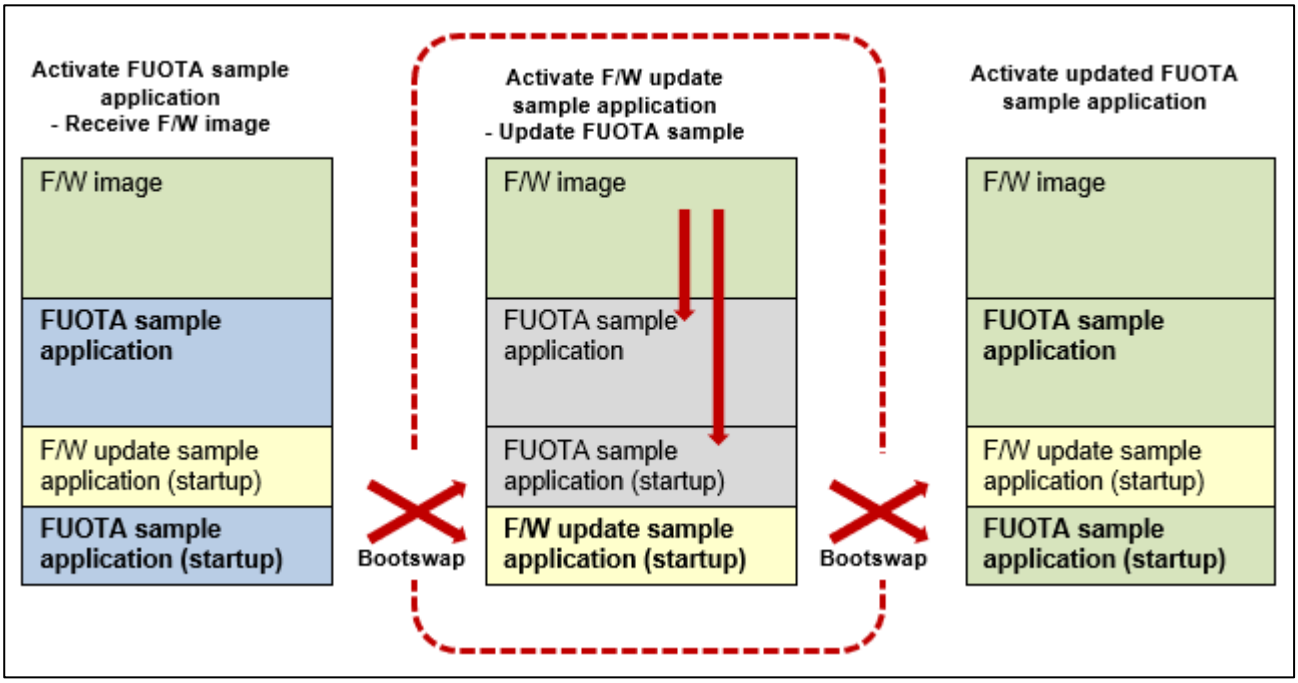


Figure 13 Overview of F/W Update Sample Application

3.2 F/W Image Format

Figure 14 shows the F/W image format that the F/W update sample application supports.

The F/W image is a binary data of the firmware and consists of some image blocks. Each block includes the information such as the start address to write the program code, the code size, and the code data.

Table 14 shows the detail format of the F/W image.

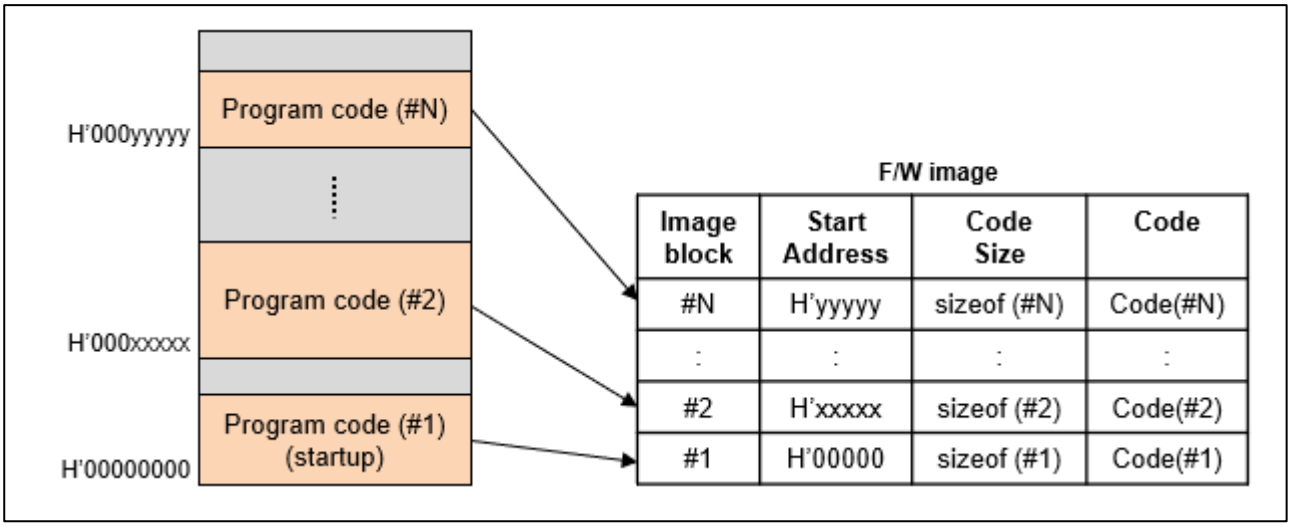


Figure 14 F/W Image

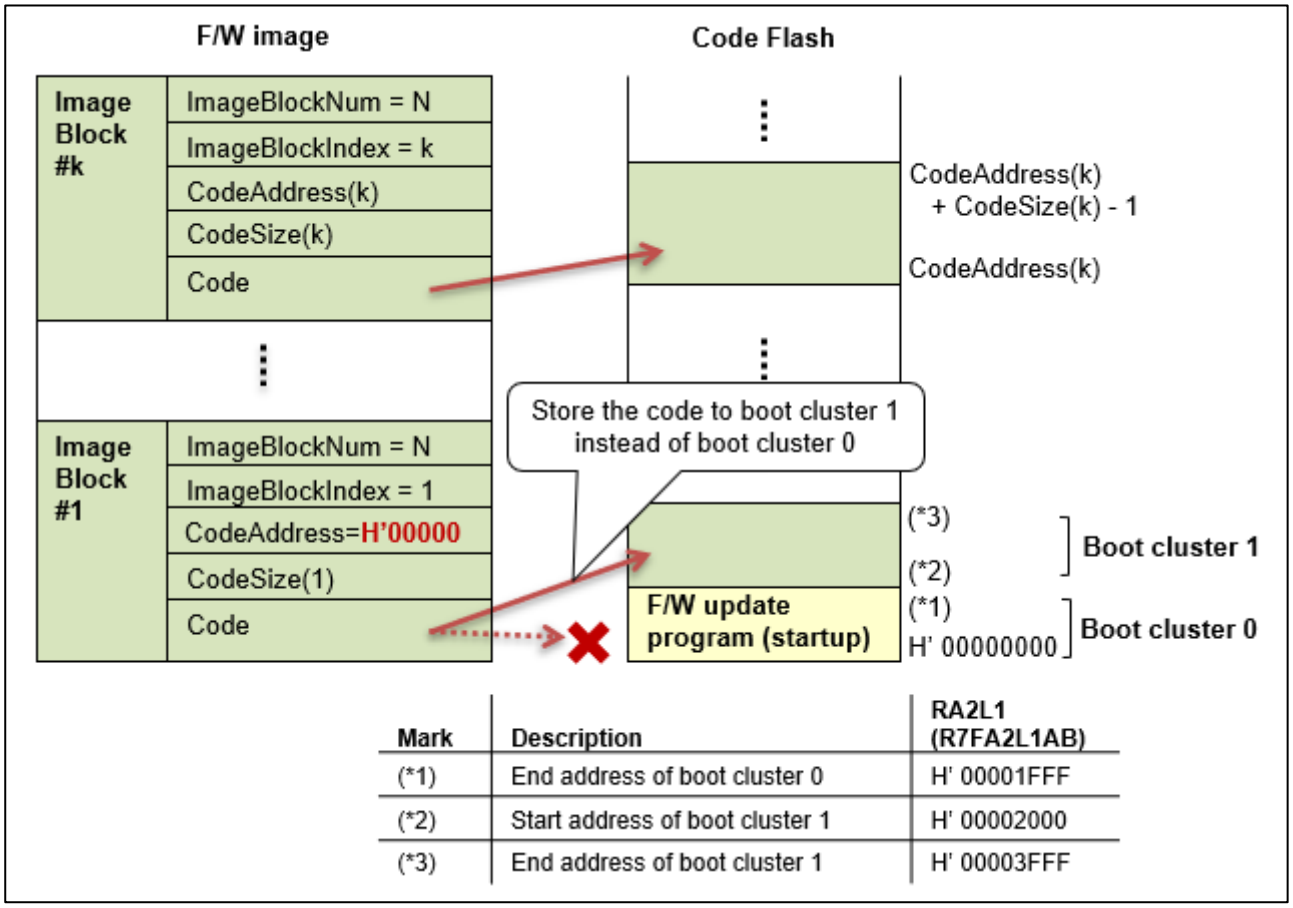
Table 14 Format of F/W Image

Contents		**Size (Byte)**	**Description**
F/W image information	ImageBlockNum	1	Total number of image blocks (=N)
	ImageBlockIndex	1	Index of image bock; here set 0.
	ImageVersion	4	Version of F/W image
	ImageSize	4	Total size of F/W image
	ImagePriority	1	Priority (Arbitrary use)
	_reserved	1	(Reserved to adjust alignment)
	ImageVerify	32	ImageVerify is used to check F/W image validity. Upper 4 byte of ImageVerify is a checksum. Lower 28 byte of ImageVerify is reserved for future extension.
Image block #1	ImageBlockNum	1	Total number of image blocks (=N)
	ImageBlockIndex	1	Index of image bock (=1)
	CodeAddress	4	Address to write code #1.
	CodeSize	4	Size of code #1.
	Code	(CodeSize)	Code #1. Note) If CodeSize is odd number, 0x00 padding for alignment.
:	:	:	:
Image block #N	ImageBlockNum	1	Total number of image blocks (=N)
	ImageBlockIndex	1	Index of image bock (=N)
	CodeAddress	4	Address to write code #N.
	CodeSize	4	Size of code #N
	Code	(CodeSize)	Code #N (see note above).

3.3 Firmware Update Using F/W Image

The F/W update sample application updates the firmware according to the information of the image blocks in the F/W image except the case the code needs to be written in the boot cluster 0 area as shown in Figure 15. The F/W update sample application writes the code to the boot cluster 1 area so that the code is mapped to the boot cluster 0 area when the new firmware is activated by the RA2L1 boot swap. (Figure 15)

After the update is finished, the F/W update sample application activates the new firmware by the RA2L1 boot swap.



Mark	Description	RA2L1 (R7FA2L1AB)
(*1)	End address of boot cluster 0	H' 00001FFF
(*2)	Start address of boot cluster 1	H' 00002000
(*3)	End address of boot cluster 1	H' 00003FFF

Figure 15 Firmware Update using F/W Image

3.4 Memory Mapping

3.4.1 Code Flash Memory Mapping

Figure 16 shows the code flash memory mapping.

The left side of the figure shows the code flash memory mapping in case the F/W update sample application is activated. In the figure, the yellow boxes show the F/W update sample application, the green box shows the storage area of F/W image, and the purple box shows the work area for F/W update sample application. The work area is used to temporarily store the code in case of power interruption. Even if a power interruption occurs during updating the firmware, the F/W update sample application can resume updating the firmware using the work area after rebooting.

The right side of the figure shows the code flash memory mapping in case the FUOTA sample application is activated, and the F/W update sample application is inactivated. In the figure, the blue boxes show the FUOTA sample application. In order for the FUOTA sample application to activate the F/W update sample application by the RA2L1 boot swap, the startup code of the F/W update sample application must be stored at the boot cluster 1. So, the code of the FUOTA sample application cannot be allocated to the boot cluster 1.

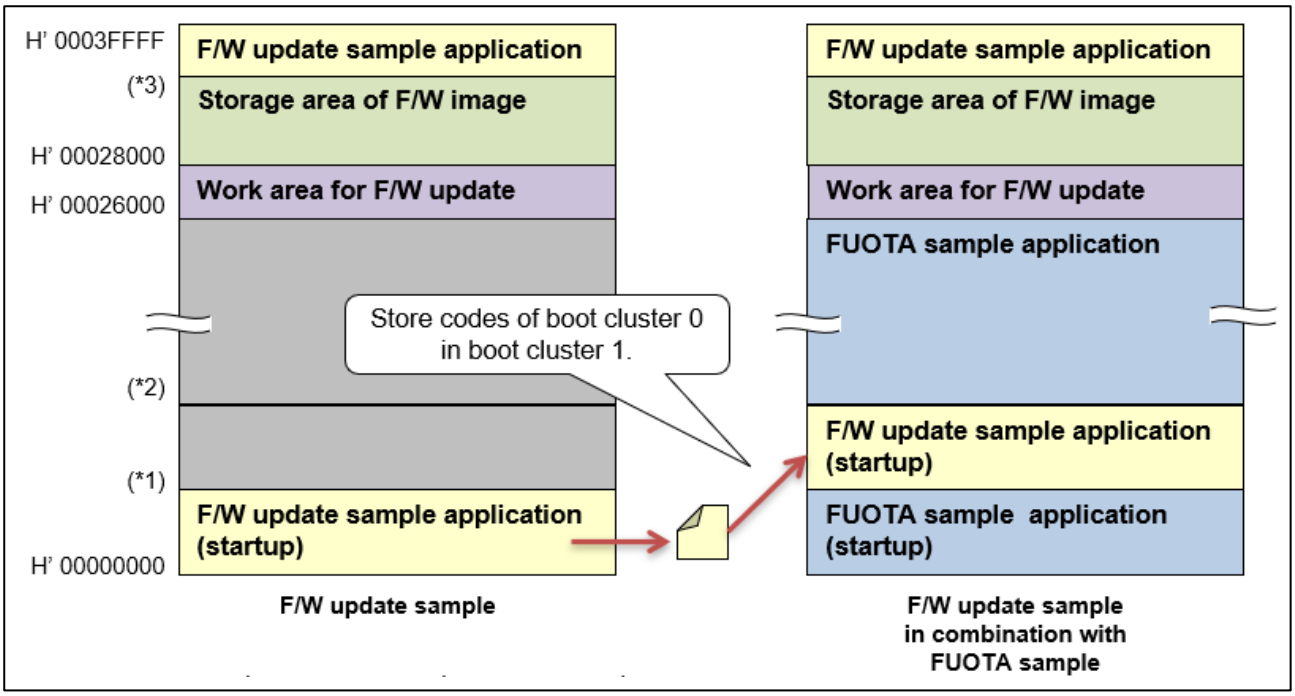


Figure 16 Code Flash Memory Mapping

Figure 17 shows the section definition and S-record of F/W update sample application. The startup code of F/W update sample application is in .text section. The code of .text section needs to be moved to the boot cluster 1 (address H'2000) by Hex/Bin converter (objcopy) with the option "--adjust-section-vma". It can be set in e2studio.

Click **File** on Menu > select **Properties**.
Expand **C/C++ Build** > select **Settings**.
Select **General** in **GNU ARM Cross Create Flash Image**.
Input "**–adjust-section-vma .text=0x2000** " in **Other flags** input box.

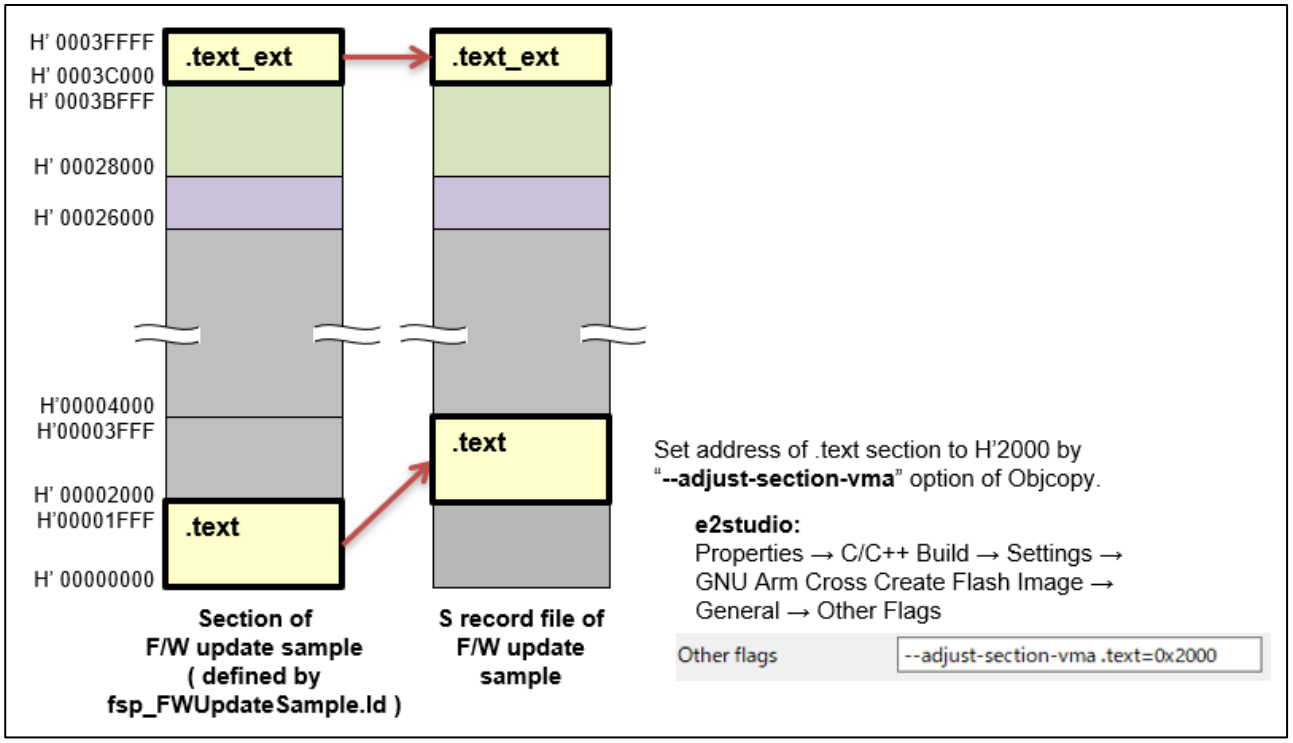


Figure 17 Section definition and S-record of F/W Update Sample

3.4.2 RAM Mapping

Both FUOTA sample application and F/W update sample application execute writing to flash memory and the boot swap. These processes must be executed in RAM. To do so, it is necessary to modify the linker script files “fsp_LoRaFuotaSample.ld” and “fsp_FWUpdateSample.ld”.

The source file to write to flash memory and the boot swap is “r_flash_lp.c” generated by FSP (Flexible Software Package). It is necessary to allocate the code of “r_flash_lp.o” (object of “r_flash_lp.c”) to RAM. To do this, modify the linker script files as shown in Figure 18.

Code Flash Memory (.text section)

```
.text :
{
    __tz_FLASH_S = ABSOLUTE(FLASH_START);
    __ROM_Start = .;

    /* Even though the vector table is not 256 e
```

⋮

```
    /* *(.text*) */
    *(EXCLUDE_FILE(*r_flash_lp.o) .text*)

    KEEP(*( version))
```

Exclude the code of “r_flash_lp.o” from .text section (code flash memory)

RAM (.data section)

```
/* Initialized data section. */
.data :
{
    __data_start__ = .;
    . = ALIGN(4);

    __Code_In_RAM_Start = .;

    KEEP(*(.code_in_ram*))
    *r_flash_lp.o(.text*)
    __Code_In_RAM_End = .;

    *(vtable)
```

Allocate the code of “r_flash_lp.o” to RAM.

(Describe between “__Code_In RAM_Start” and “__Code_In_RAM_End”.)

Figure 18 Allocate the “r_flash_lp.o” to RAM by Editing Linker Script File(s)

4. Example Operations of FUOTA Sample Application

This chapter describes the example operations of FUOTA sample application.

The section 4.1 describes preparation required for the end device. The section 4.2 describes preparation required for the LoRaWAN network server. The section 4.2.3 describes the example operations using the AT commands to setup/run FUOTA operations and update the firmware by the received new F/W image.

4.1 Preparation for End Device

Two sample applications, the FUOTA sample application and the F/W update sample application, needs to be built and programmed to the hardware you use.

4.1.1 Hardware Setup

The sample application supports the Evaluation Kit for RA2L1 MCU Group and Semtech LoRa RFIC shield. As for detail setup, please refer to [7].

<Evaluation Kit >

- Evaluation Kit for RA2L1 MCU Group (EK-RA2L1)

<Semtech LoRa RFIC shield>

- Semtech SX1261/SX1262 Shield

4.1.2 Configuration of Sample Application

Table 15 shows the major macros available for the configuration of the FUOTA sample application. These macros can be specified in the project build option as needed.

Table 15 Macros Available for Configuration of FUOTA Sample Application

Macro	Description	Default
FUOTA_ENABLED	Enables FUOTA features. This macro needs to set for the FUOTA sample application.	Defined
FUOTA_VERSION_1_0_0	Support FUOTA V1.0.0. (If omitted, the default version is set to 1.0.0.)	Defined
LORAWAN_VERSION_1_0_4	Support LoRaWAN protocol version 1.0.4. (`LORAWAN_VERSION_1_0_3` cannot be specified simultaneously.)	Defined
LORAWAN_VERSION_1_0_3	Support LoRaWAN protocol version 1.0.3. (It can be omitted, i.e. default version is 1.0.3.)	Undefined
REGION_AS923	Enable `AS923` feature. [LoRaWAN 1.0.4 only] Enables all groups of `AS923` (`AS923-1`, `AS923-2`, `AS923-3` and `AS923-4`).	Defined
REGION_EU868	Enable `EU868` feature.	Defined
REGION_US915	Enable `US915` feature.	Defined
RP_USE_RADIO_CFG_CHECK	Enable the regulatory function for each region in Radio Driver (see [4]).	Defined
LORAMAC_CLASSB_ENABLED	Enable class B feature.	Defined
APP_AT_KEY_READ_ENABLED	Enables to read keys such as `AppKey`, `AppSKey`, `NwkSKey` and `GenAppKey` by using corresponding AT commands.	Undefined
DEBUG_LORAMAC, DEBUG_RADIO	Enables the debug mode. Both `DEBUG_LORAMAC` and `DEBUG_RADIO` need to be set when to use the debug mode. The debug mode is necessary if MCU cannot wake up from the low power mode by an interrupt before receiving UART data.	Defined
DEBUG_LORAMAC_DEFAULT_MODE=0xXXXXXXXX	Specifies the default debug mode with the ORed value of the following if necessary. For more details, refer to [2]. 0x00000100 — Enables Pseudo MCU low power operation. 0x00000001 — Enables debug log of Tx/Rx data as the sniffer mode format of Renesas LPWA Studio (see [5]). 0x00000002 — Enables debug log of radio Rx. 0x00000004 — Enables debug log of radio TX. 0x00000008 — Enables debug log of radio CCA. This macro can be specified when `DEBUG_LORAMAC` and `DEBUG_RADIO` are defined.	Defined Set to 0x00000100

4.1.3 Building of FUOTA Sample Application

The FUOTA sample application needs to be built using one of the following project files. The object file of the program image will be made as '`LoRaFuotaSample.srec`'. The object file of the program and the symbols will also be made as '`LoRaFuotaSample.elf`' in case of e2studio.

[Project file]

<Evaluation Kit for RA2L1 MCU Group (EK-RA2L1)>

`samples\project\e2studio\ra2l1ek_sx126x\LoRaFuotaSample`

4.1.4 Building of F/W Update Sample Application

The FW update sample application needs to be built using one of the following project files.

In case of RA2L1, the object file of the program image will be made as '`FWUpdateSample.srec`'. The program is mapped from the address of H'00002000 to be programmed to the boot cluster 1 area. Refer to the section 3.4 for details.

[Project file]

<Evaluation Kit for RA2L1 MCU Group (EK-RA2L1)>

`samples\project\e2studio\ra2l1ek_sx126x\FWUpdateSample`

4.1.5 Programing of Object Files to Code Flash Memory

The two object files built in the section 4.1.3 and 4.1.4 need to write to the code flash memory of RA2L1. The operations for the flash programing are shown in the following (1) and (2)

(1) When Renesas Flash Programmer (RFP) is used for the flash programing

If only the flash programing is necessary, RFP can be used. Figure 19 shows the configuration of RFP. This configuration is necessary before writing the object files. After that, the object file of only program image (.srec) of the FUOTA sample application needs to be download first, and the object file of only program image (.srec) of the F/W update sample application needs to be download secondly.

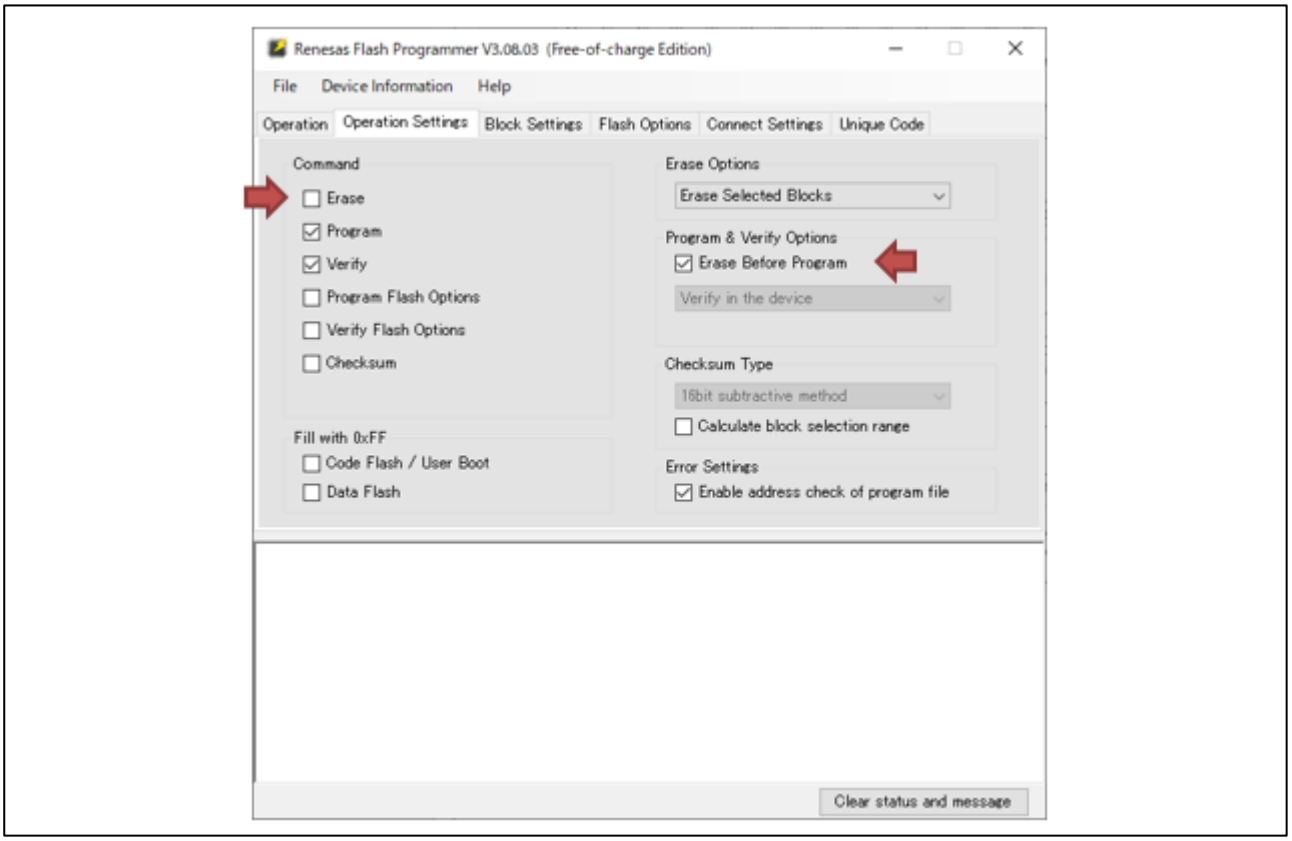


Figure 19 Configuration of RFP to Download Object Files

(2) When e2studio is used for the flash programing and debug

If not only the flash programing but also the debugging is necessary, e2studio needs to be used. Figure 20 show the example configurations of the object files to be download in case of e2studio.

When e2studio is used, the object file with the image and symbols (.elf) of the FUOTA sample application needs to be download first, and the object file with only image (.srec) of the F/W update sample application needs to be download secondly.

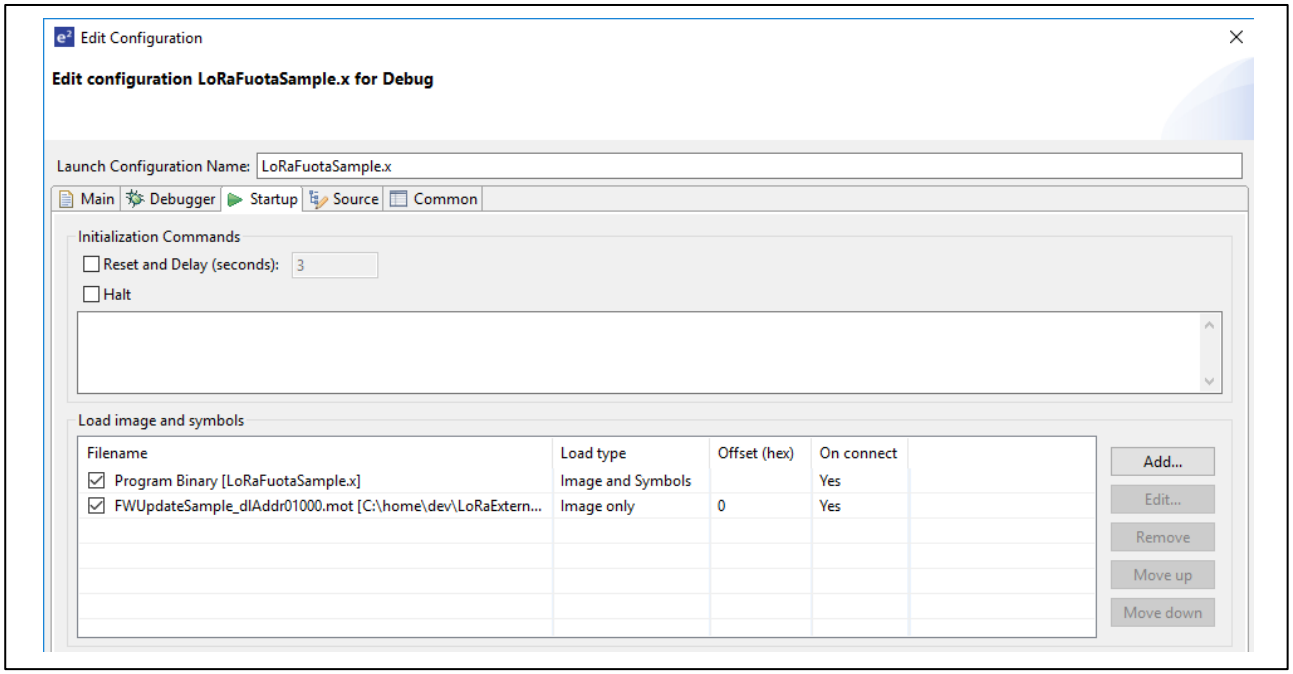


Figure 20 Configuration of Object Files To Be Download In Case of e2studio

4.2 Preparation for LoRaWAN Network Server

4.2.1 Basic Configuration of LoRaWAN Network Server

An end device related information such as the region, channel plan, device class, activation mode (OTAA/ABP), device EUI, and application key needs to be configured in the LoRaWAN network server you use. Refer to [6] for an example of the configuration.

The section 4.2.2 and 4.2.3 describe the additional configuration specific for the FUOTA.

4.2.2 Make F/W Image Files (Binary)

The F/W image files (binary) need to be converted from the object file (.srec) of the new firmware to deliver the files from the LoRaWAN network server. A batch file, 'make_fwimage.bat', is prepared for the conversion.

Name	make_fwimage.bat	
Description	Make the F/W image from an object file (.srec) and output F/W image file(s) from the F/W image, divided by the specified size	
Syntax	make_fwimage.bat [MCU] [MotFile] [FWVersion] [DividingSize] [OutputFile]	
Folder	`samples\tools\FUOTA`	
Argument	MCU	Specify the MCU; RA2
	MotFile	Object file (.srec)
	FWVersion	Version to be set to F/W image Four bytes HEX number without prefix.
	DividingSize	Dividing size of the F/W image. Need to specify the dividing size less than to the data block size that the end device can receive, which can bet set to `FUOTA_CONFIG_FRGMNT_MAX_DATABLK_SIZE` (see 2.3.3). If 0 is specified, the F/W image will not be divided.
	OutputFile	The base file name for the F/W image. See below.
Example	**Argument:** [MCU] = RA2 [MotFile] = Application.srec [FWVersion] = 0x00000100 (4 byte) [DividingSize] = 8192 byte [OutputFile] = FWImage make_fwimage.bat RA2 Application.srec 00000100 8192 FWImage **Output divided F/W image files:** ● FWImage_0x00000000.bin (Image file to be sent first, 8192+4 byte) (*1) ● FWImage_0x00000001.bin (Image file to be sent second, 8192+4 byte) (*1) ● FWImage_0x00000002.bin (Image file to be sent third, 1024+4 byte) (*1) Where the total size of the FW image is 17408 bytes. (*1) The four bytes are for the divided F/W image header. See 4.2.3 for details.	

4.2.3 Setup to Deliver F/W Image

The F/W image file(s) made in the section 4.2.2 need to be specified to the application server so that the files will be delivered from the server via the fragmentation session(s). The file(s) need to be specified in the order of the number appended to the file name. If the file(s) are not delivered in order, firmware update will fail because F/W image cannot be written to the code flash memory correctly (see 2.8.3).

Note that, a header is added to the divided F/W image file(s). It includes 4 bytes of the index number only. (Figure 21)

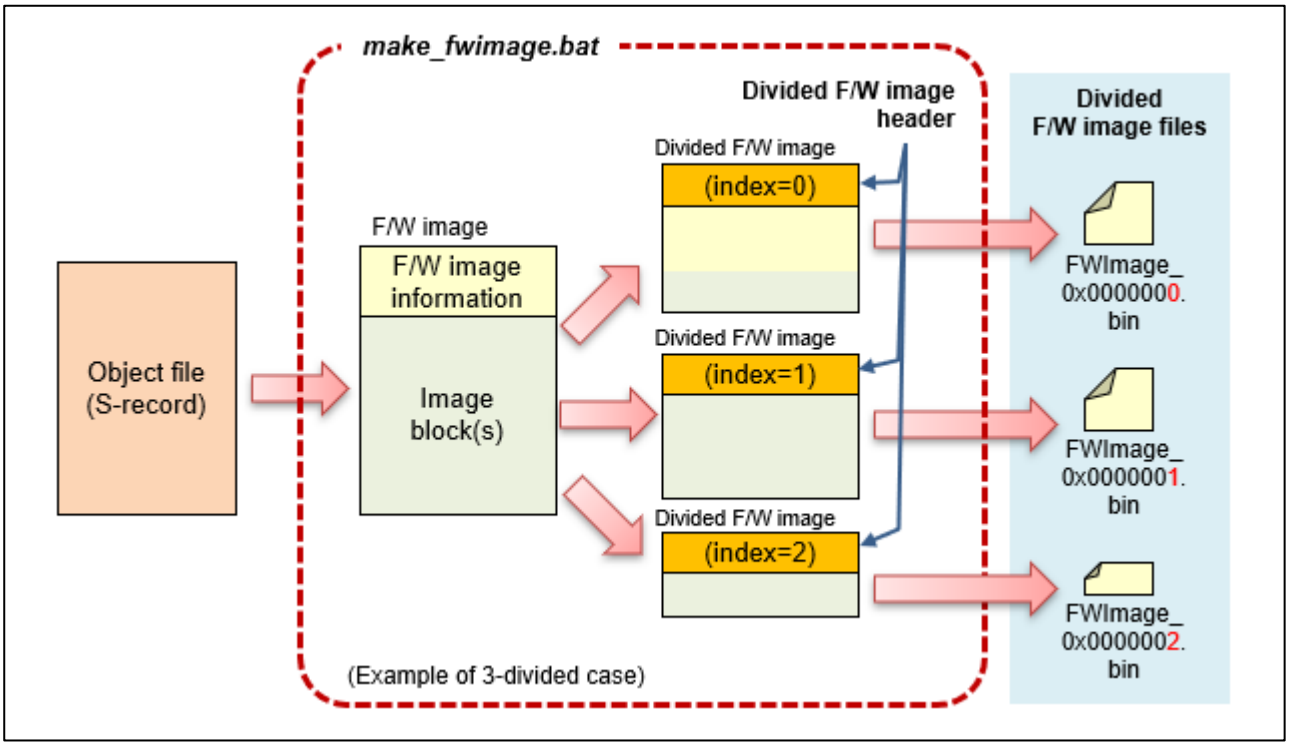


Figure 21 F/W Image Header Added to F/W Image File(s)

In addition, the setup such as the following items might be additionally necessary. For more details, refer to the specification of the LoRaWAN server to be used. See Appendix.B as for the example operations how to setup the FUOTA task when MultiTech Conduit is used as a LoRaWAN network server.

Example of the setup:

- Fragmentation
  - Transmission interval of the fragments to be sent
- Multicast (if multicast is used for the delivery)
  - Device class used for the multicast session
  - `GenAppKey` to share the multicast session key
- Time configuration
  - Time to start/end of the delivery

4.3 Example Operations of End Device

This section describes the example operation of end device. In this example, it is supposed that the sample F/W image file included in the software package is used. The file is in the following folders. The size of the file is 314 bytes, and the size of the code included in the file is 256 bytes.

Please note that the sample F/W image can be used if the source code and the build setting of the FUOTA sample application are not changed from the original ones included in the software package because the sample F/W image updates the code in the specific address area.

[Sample F/W image file]

<Evaluation Kit for RA2L1 MCU Group (EK-RA2L1)>

```
samples\project\e2studio\ra2l1ek_sx126x\LoRaFuotaSample\sample_fwimage\
sample_fwimage_0x00000000.bin
```

When the sample F/W image file is applied via the FUOTA process, the version of the FUOTA sample application is changed from Ver.04.70 to Ver.09.00. The change can be confirmed using the AT command, "`AT+VER?`".

Before the sample F/W image is applied	AT+VER? +VER: LoRa Sample App Ver.04.70 OK
After the sample F/W image is applied	AT+VER? +VER: LoRa Sample App Ver.09.00 OK

The following is the sample operation of the end device for the FUOTA. The value with under line should be change according to the setting of the LoRaWAN server and the information of the end device.

Example operation and notification for end device	Description
	Confirm the version of the current FUOTA sample application is 'Ver.04.20'.
AT+VER? +VER: LoRa Sample App Ver.04.20 OK	Show version
	Set parameters such as region, device class, activation mode, `AppKey`, `AppEUI`, `DevEUI` required in case of OTAA (see [2]).
AT+REGION=6 OK	Set `AS923` for region
AT+CLASS=0 OK	Set Class A for device class
AT+ACTMODE=1 OK	Set OTAA for activation mode
AT+DEVEUI=90F OK	Set 000000000000090F for `DevEUI`
AT+APPEUI=10E OK	Set 000000000000010E for `AppEUI`
AT+APPKEY=F0E OK	Set 00000000000000000000000000000F0E for `AppKey`
AT+JOIN OK +JOIN: JOIN_ACCEPTED	Request to join the network
	Start FUOTA and set related parameters
AT+FUOTASTART OK	Start FUOTA
AT+FUOTASET=10,0 OK	Disable to send `AppTimeReq` periodically
AT+FUOTASET=F0,30 OK	Set 30 seconds to sending interval of uplink messages to receive downlink message

AT+GENAPPKEY=00000000000000000000000000000000 OK	Set 00000000000000000000000000000000 to `GenAppKey` to receive the multicast session key from the server. Need to change the key value according to the server setting/specification.
	Create a FUOTA task and schedule a FUOTA session in the application server.
	Indication of multicast when the multicast is used.
+FUOTAIND: 0,2,0,54,4096	Indication of setup of multicast session, which is indicated when the multicast is used Session class: Class C Multicast group ID: 0 Seconds to start session: 54 seconds Seconds to timeout session: 4096 seconds
+FUOTAIND: 1,2,0,4096	Indication of start of multicast session Session class: Class C Multicast group ID: 0 Seconds to timeout session: 4096 seconds
	Wait until complete F/W image is received and update the FUOTA sample application
+FUOTAIND: 128	Indication of the F/W image is ready If the F/W image consists of multiple data blocks, this command will be indicated when all data blocks are received.
AT+FUOTAUPDT OK	Update the firmware, the FUOTA sample application, using the F/W image after its validation. After the update, the updated FUOTA sample application will be activated.
	Confirm the version of the updated FUOTA sample application is changed to ‘Ver.09.00’. It indicates the firmware update is successful.
AT+VER? +VER: LoRa Sample App Ver.09.00 OK	Show version.

Appendix.A. FUOTA V2.0.0

The FUOTA sample application experimentally supports FUOTA V2.0.0.

This appendix describes FUOTA V2.0.0, focusing on the differences from FUOTA V1.0.0.

A.1 Features of FUOTA V2.0.0

FUOTA V2.0.0 provides the following features in addition to FUOTA V1.0.0 (see chapter 1). Figure 22 shows the FUOTA V2.0.0 message exchange.

- **Fragment Data Block Transport Protocol v2.0.0**

  This protocol supports the MIC (Message Integrity Code). The end device can verify the integrity of the received data block using the MIC.

- **Firmware Management Protocol v1.0.0**

  This protocol is available when FUOTA V2.0.0 is enabled. The application server can query the end device to see if the F/W image is ready. When the end device is ready to update the firmware, the application server can request the end device to reboot. The end device will start to update the firmware and reboot at the specified reboot timing.

- **Multi Package Access Protocol v1.0.0**

  This protocol is available when FUOTA V2.0.0 is enabled. It can be used to send multiple commands of one or more packages in a single uplink and downlink message. This can save the number of downlink and uplink transmissions.

- **Clock Synchronization Message Protocol v2.0.0**

- **Remote Multicast Setup Protocol v2.0.0**

  These protocols include minor updates to address issues in v1.0.0.

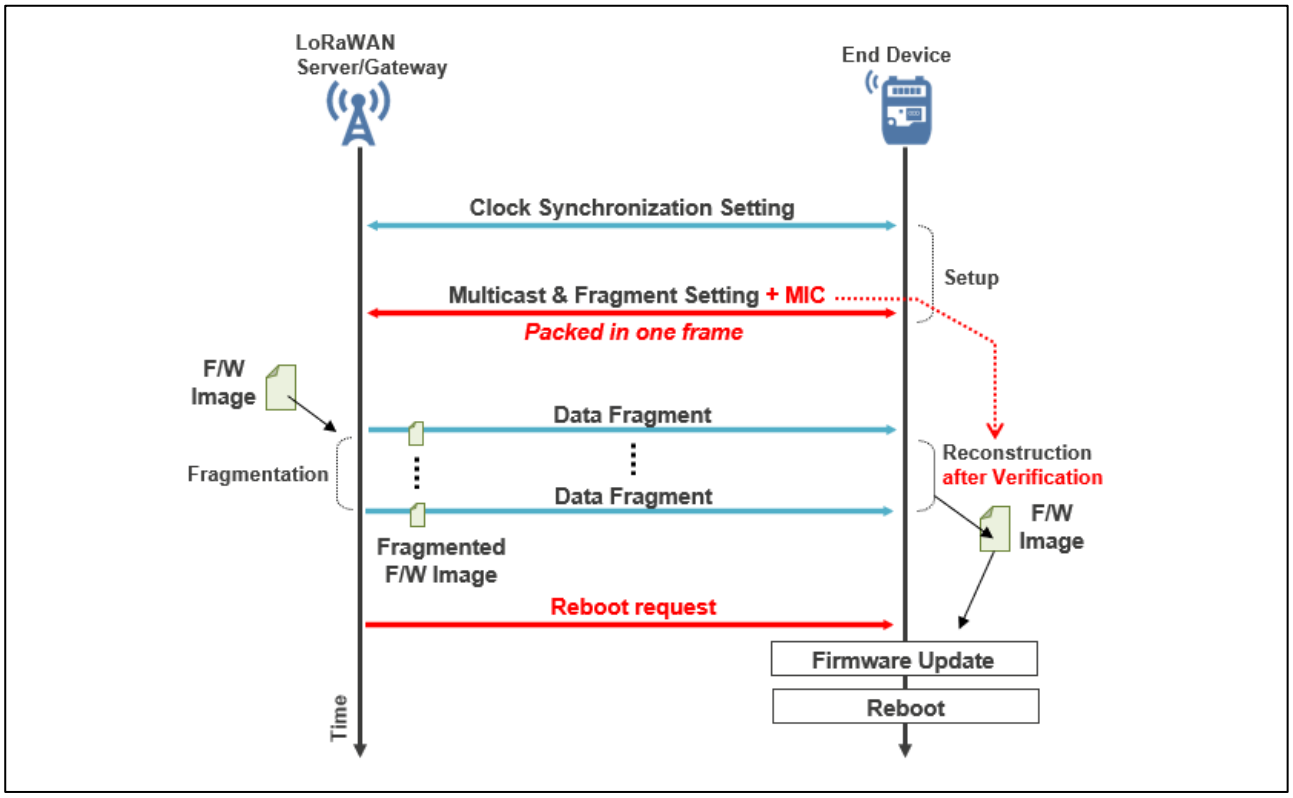


Figure 22 FUOTA V2.0.0 Message Exchange between LoRaWAN Network Server and End Device

A.2 FUOTA V2.0.0 Sample Application

A.2.1 FUOTA V2.0.0 Sample Application Block Diagram

Figure 23 shows a block diagram of the FUOTA V2.0.0 sample application. The FUOTA V2.0.0 includes the application layer message protocols over LoRaWAN shown in Table 16.

When FUOTA V2.0.0 is used, the firmware management protocol and the multi package access protocol are additionally supported compared to FUOTA V1.0.0 (see 2.1.1). The firmware management protocol is used to manage the firmware of the end device. The multi package access protocol is used to send several commands of one or more packages in a single downlink and uplink payload to save the number of transmissions.

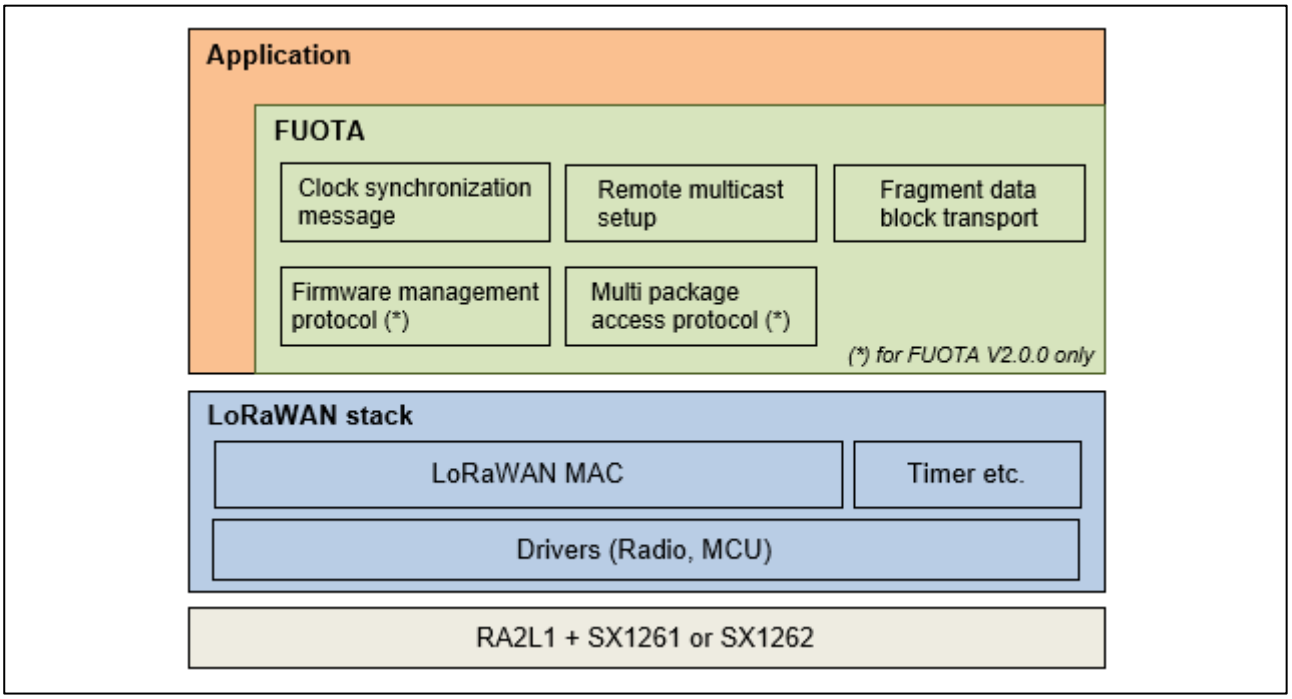


Figure 23 FUOTA V2.0.0 Sample Application Block Diagram

Table 16 Application Layer Messaging Package List for FUOTA V2.0.0

Package name	Version	Package ID	Package version	FPort
Clock Synchronization Message Package (*1)	v2.0.0	1	2	202
Remote Multicast Setup Package (*2)	v2.0.0	2	2	200
Fragmented Data Block Transport Package (*3)	v2.0.0	3	2	201
Firmware Management Protocol (*4)	v1.0.0	4	1	203
Multi Package Access Protocol (*5)	v1.0.0	0	1	225

(*1) https://resources.lora-alliance.org/technical-specifications/ts003-2-0-0-application-layer-clock-synchronization
(*2) https://resources.lora-alliance.org/technical-specifications/ts005-2-0-0-remote-multicast-setup
(*3) https://resources.lora-alliance.org/technical-specifications/ts004-2-0-0-fragmented-data-block-transport
(*4) https://resources.lora-alliance.org/technical-specifications/ts006-1-0-0-firmware-management-protocol
(*5) https://resources.lora-alliance.org/technical-specifications/ts007-1-0-0-multi-package-access

A.2.2 Software Architecture

Figure 24 shows a software architecture of the FUOTA V2.0.0 layer. The firmware management protocol and the multi package access protocol are added to the FUOTA layer described in 2.2.2.

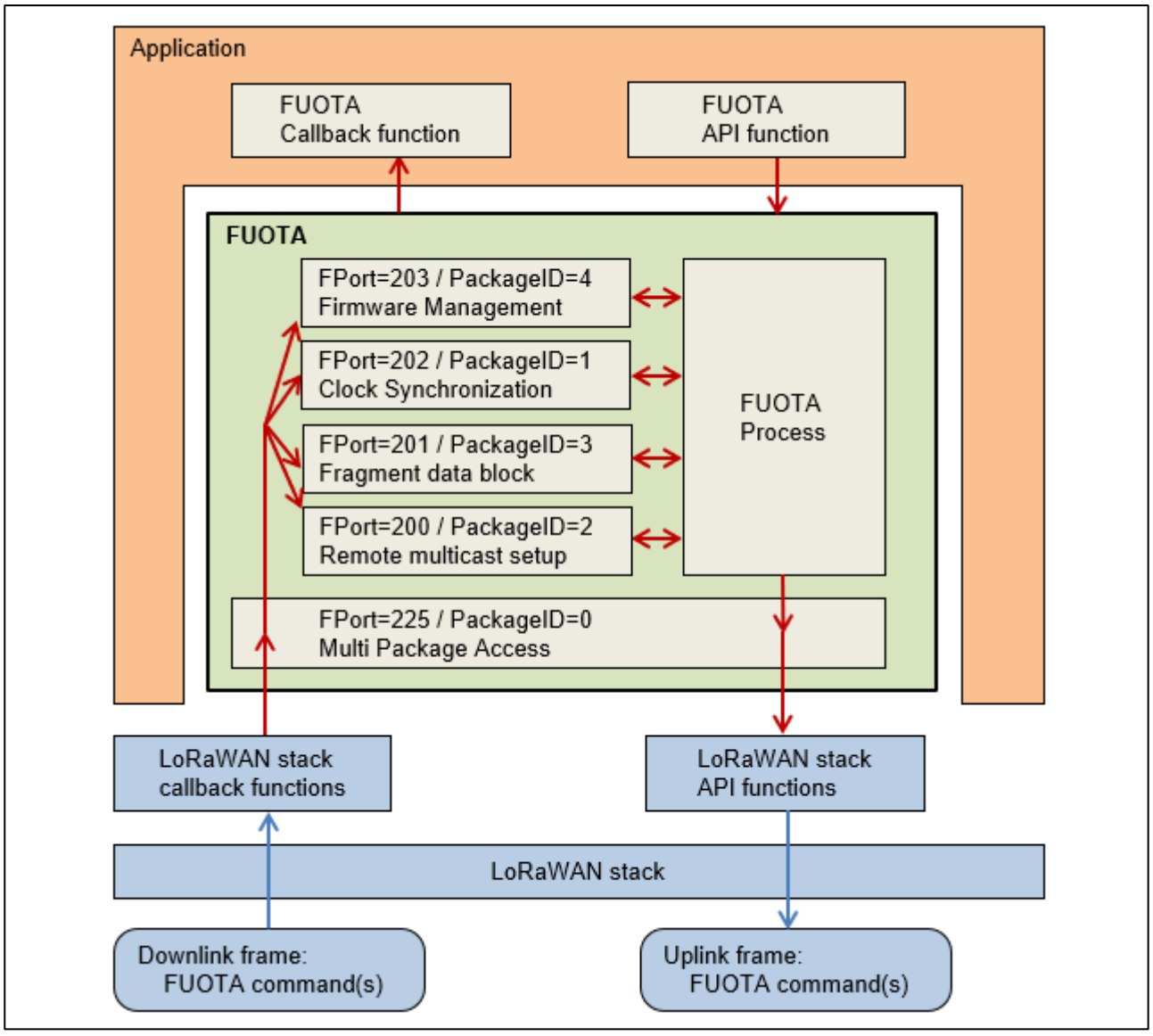


Figure 24 FUOTA V2.0.0 Software Architecture

A.2.3 Macros

A.2.3.1 FUOTA Setting

Table 17 shows the additional macro for the FUOTA setting. FUOTA V2.0.0 functions are available if "`FUOTA_ENABLED`" (see 2.3.1 and Table 5) and "`FUOTA_VERSION_2_0_0`" are defined in the project build options.

Table 17 Additional Macro for FUOTA Setting

Macro	Description
FUOTA_VERSION_2_0_0	Support FUOTA version 2.0.0 (Experimental). (`FUOTA_VERSION_1_0_0` cannot be specified simultaneously.)

A.2.3.2 FUOTA Configuration

Other The FUOTA V2.0.0 configurations are same as the ones of FUOTA V1.0.0 (see 2.3.2 and Table 6).

A.2.4 FUOTA APIs

The APIs for FUOTA V2.0.0 are same as the ones of FUOTA V1.0.0 (see 2.5).

A.2.5 Callback Handler Functions (FuotaEventCb_t)

Table 18 shows the five additional members (pointer to the callback handler functions) for FUOTA V2.0.0. Other members are same as the ones of FUOTA V1.0.0 (see 2.6).

Table 18 FuotaEventCb_t (additional member for FUOTA V2.0.0)

Member (callback handler functions)	Description
FuotaStatus_t (*FuotaFwMngRebootRequestIndication)(uint32_t rebootSec);	Pointer to callback function to be called when the reboot request is received.
void (*FuotaFwMngRebootCanceledIndication)(void);	Pointer to callback function to be called when the previous reboot request is canceled.
void (*FuotaFwMngRebootExecIndication)(void);	Pointer to callback function to be called when the reboot time has come.
uint8_t (*FuotaFwMngUpImageStatusRequest)(uint32_t *p_nextFirmwareVersion);	Pointer to callback function to be called when the application server asks whether the end device has a F/W image.
uint8_t (*FuotaFwMngDeleteImageRequest)(uint32_t fwToDelVersion);	Pointer to callback function to be called when the application server requests the end device to delete the F/W image.
void (*FuotaFwMngVersionInfoRequest)(uint32_t *p_fwVersion, uint32_t *p_hwVersion);	Pointer to callback function to be called when the application server asks the firmware and hardware information.

A.2.5.1 FuotaFwMngRebootRequestIndication

FuotaStatus_t (*FuotaFwMngRebootRequestIndication)(uint32_t rebootSec)		
This function notifies when reboot request has been received from application server.		
Parameters:		
rebootSec	Input	The number of seconds until the end device reboots.
Return:		
FUOTA_STATUS_OK		Accept the reboot request.
FUOTA_STATUS_ERROR		Cannot accept the reboot request at the requested time.

A.2.5.2 FuotaFwMngRebootCanceledIndication

void (*FuotaFwMngRebootCanceledIndication)(void)
This function notifies when the reboot schedule has been canceled.
Parameters:
None
Return:
None

A.2.5.3 FuotaFwMngRebootExecIndication

void (*FuotaFwMngRebootExecIndication)(void)		
This function notifies when the requested reboot time has come. The end device shall reboot as soon as possible. If it has a valid F/W image, it will update its firmware at this timing.		
Parameters:		
None		
Return:		
None		

A.2.5.4 FuotaFwMngUpImageStatusRequest

uint8_t (*FuotaFwMngUpImageStatusRequest)(uint32_t *p_nextFirmwareVersion)		
This function notifies when the application server asks whether F/W image is present. If the end device has a valid F/W image, the end device shall set its version to '`p_nextFirmwareVersion`'.		
Parameters:		
p_nextFirmwareVersion	Output	F/W image version (if an end device has valid F/W image.)

Return:	
0x00: FUOTA_FWIMG_STATUS_NONE	No F/W image
0x01: FUOTA_FWIMG_STATUS_INVALID	Invalid F/W image (for example, F/W image is corrupted)
0x02: FUOTA_FWIMG_STATUS_HW_NONSUPPORT	F/W image is not compatible (for example, F/W image is for other hardware platform)
0x03: FUOTA_FWIMG_STATUS_AVAILABLE	F/W image is valid

A.2.5.5 FuotaFwMngDeleteImageRequest

uint8_t (*FuotaFwMngDeleteImageRequest)(uint32_t fwToDelVersion)		
This function notifies when the application server requests the end device to delete the F/W image. The end device shall delete the F/W image at this timing.		
Parameters:		
fwToDelVersion	Input	Version of F/W image to delete.

Return:	
0x00: FUOTA_FWIMG_DELETEIMG_STATUS_OK	An end device has deleted the F/W image successfully.
0x01: FUOTA_FWIMG_DELETEIMG_STATUS_NO_VALID_IMAGE	An end device does not have valid F/W image.
0x02: FUOTA_FWIMG_DELETEIMG_STATUS_INVALID_VERSION	An end device does not have requested version of F/W image.

A.2.5.6 FuotaFwMngVersionInfoRequest

void (*FuotaFwMngVersionInfoRequest)(uint32_t *p_fwVersion, uint32_t *p_hwVersion)		
This function notifies when the application server asks the firmware and hardware version information. The end device shall set them to `p_fwVersion` and `p_hwVersion`.		
Parameters:		
*p_fwVersion	Output	Firmware version.
*p_hwVersion	Output	Hardware version
Return:		
None		

A.2.6 FUOTA V2.0.0 Related Commands Sequence, Usage of Callback Functions

This section describes the command sequence of the fragment data block transport package for FUOTA V2.0.0, firmware management protocol, multi package protocol. Note that the command sequence of the remote multicast package and the clock synchronization package are same as the ones of FUOTA V1.0.0.

A.2.6.1 Fragment Data Block Transport Package for FUOTA V2.0.0

Figure 25 shows a flow diagram of the fragment data block transport package for FUOTA V2.0.0. Message exchanges between the application server and end device are same as FUOTA V1.0.0 (see 2.7.4) except that the MIC (Message Integrity Code) for the data a block is delivered when the application server requests a setup of fragment session. The end device can verify the received data block using the MIC.

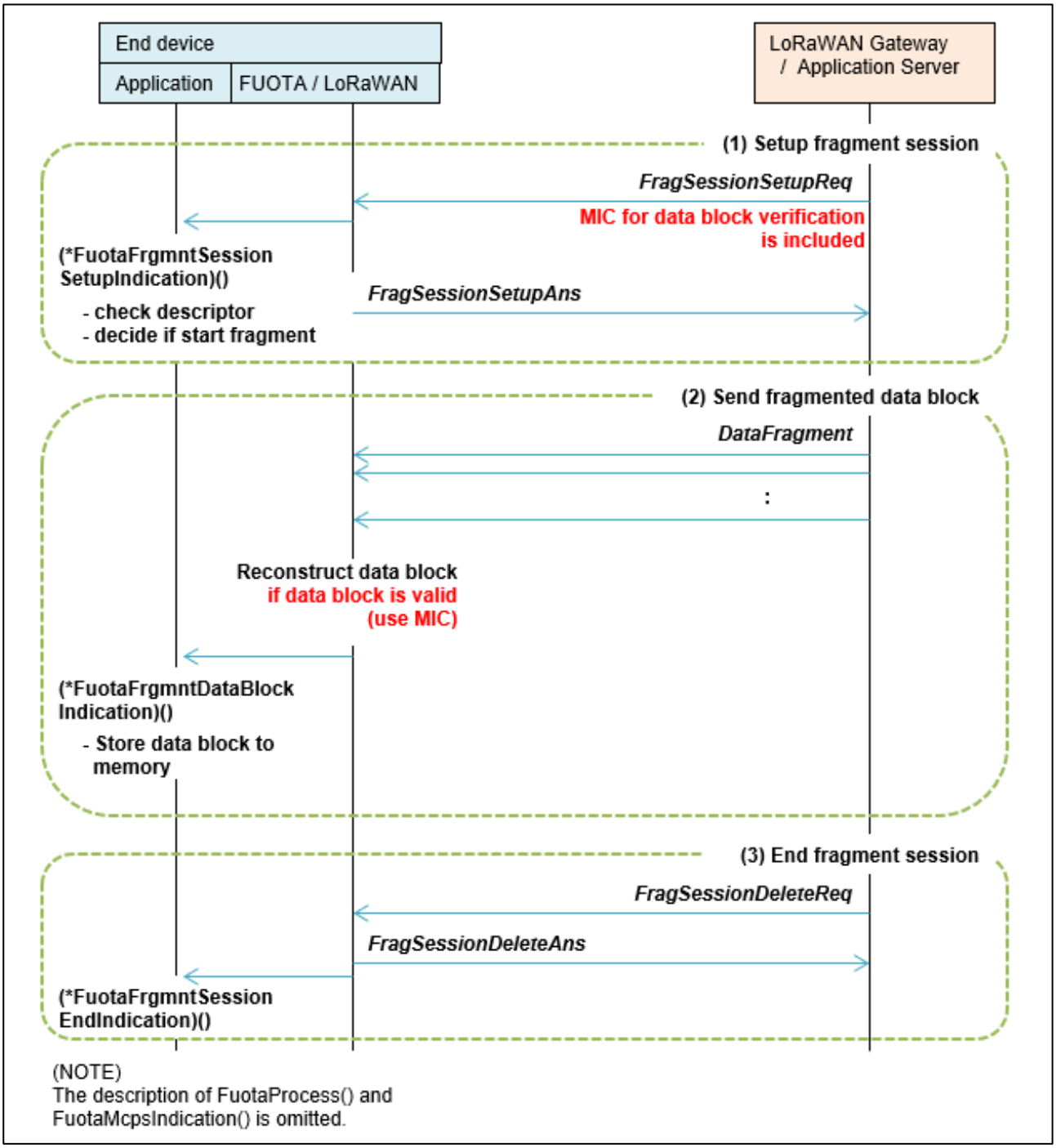


Figure 25 Fragment Data Block Transport Package for FUOTA V2.0.0

A.2.6.2 Firmware Management Protocol

Figure 26 shows a flow diagram of the firmware management protocol.

When the application server asks the end device whether it has a valid F/W image, the FUOTA layer notify the application of the query via the callback function. The application responses the status of the FW image to the application server via the callback function.

If the application server knows that the end device can update its firmware, it can request the end to update the firmware and reboot itself. When the end device receives the reboot request from the application server, the FUOTA notifies the application of the timing of reboot. The application shall reboot at that time. Also, if the F/W image to update is present, the application shall update its firmware and reboot with a new firmware.

The application server can know whether the firmware of the end device has been updated by asking the version information to the end device. The FUOTA layer notifies the application of the query of firmware and hardware version via the callback function. The application responses the version information to the application server via the callback function.

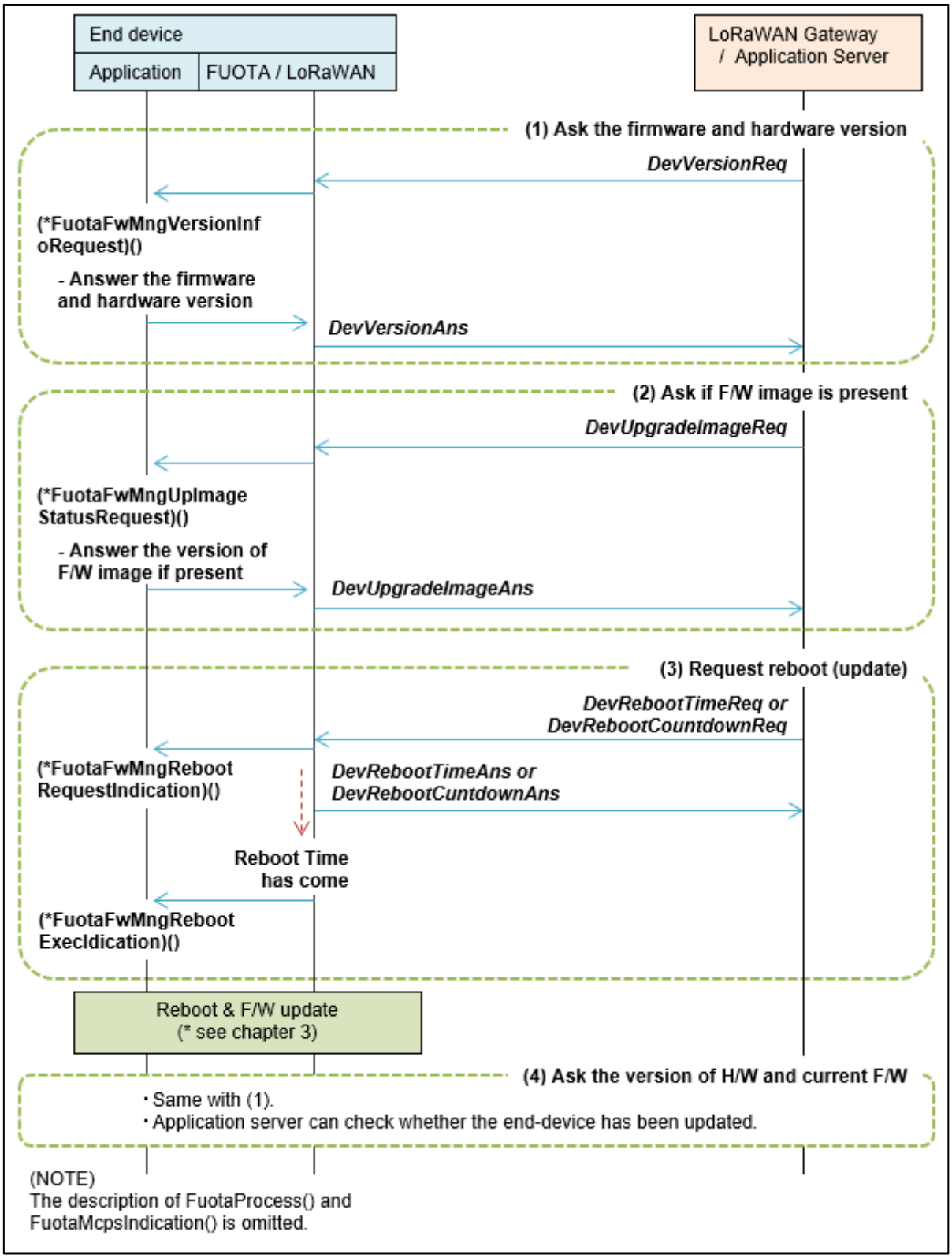


Figure 26 Firmware Management Protocol

A.2.6.3 Multi Package Access Protocol

Figure 27 and Figure 28 show the flow diagrams of the multi package access protocol.

The packages used for the FUOTA process could be increased and updated in the future. The multi package access protocol provides a command that the application server can retrieve the information of the active packages and those versions which the end device supports (see Figure 27).

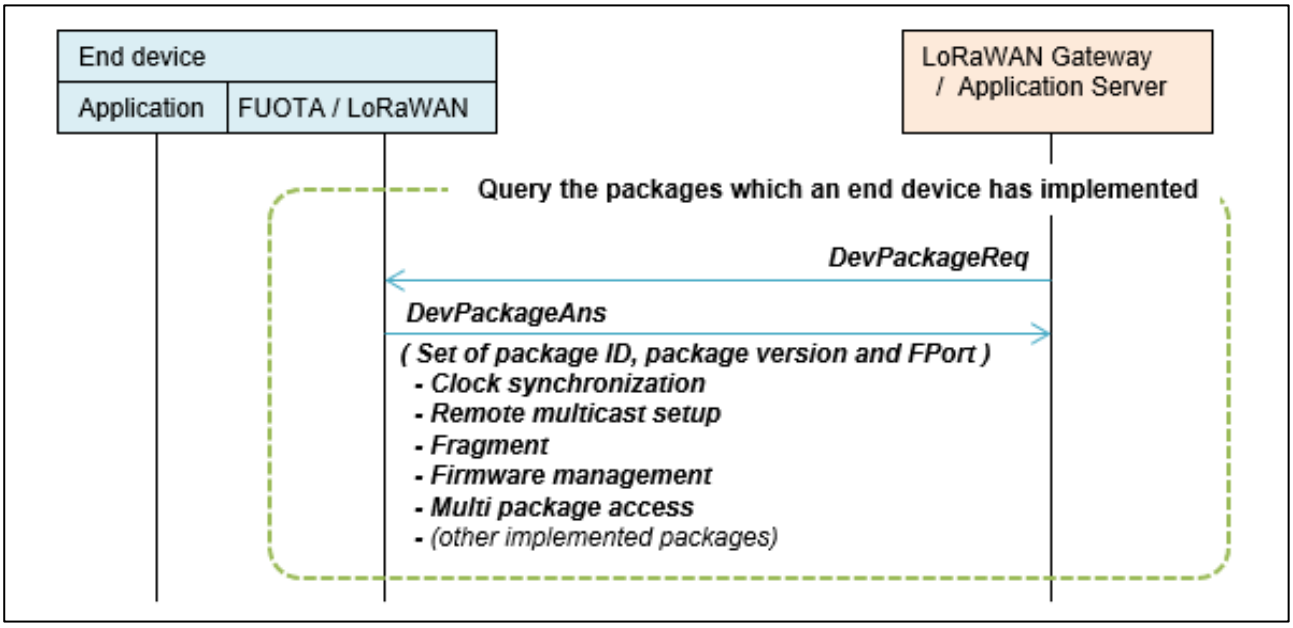


Figure 27 Multi Package Access Protocol (Query Implemented Packages)

The multi package access protocol can be used to send multiple commands of package(s) in a single uplink and downlink.

The application server can pack several commands in a downlink payload. When an end device receives it, the FUOTA will process commands one by one. And the FUOTA pack the answer commands in an uplink payload. The application does not need to care whether the FUOTA received several commands by multi-package access (see Figure 28).

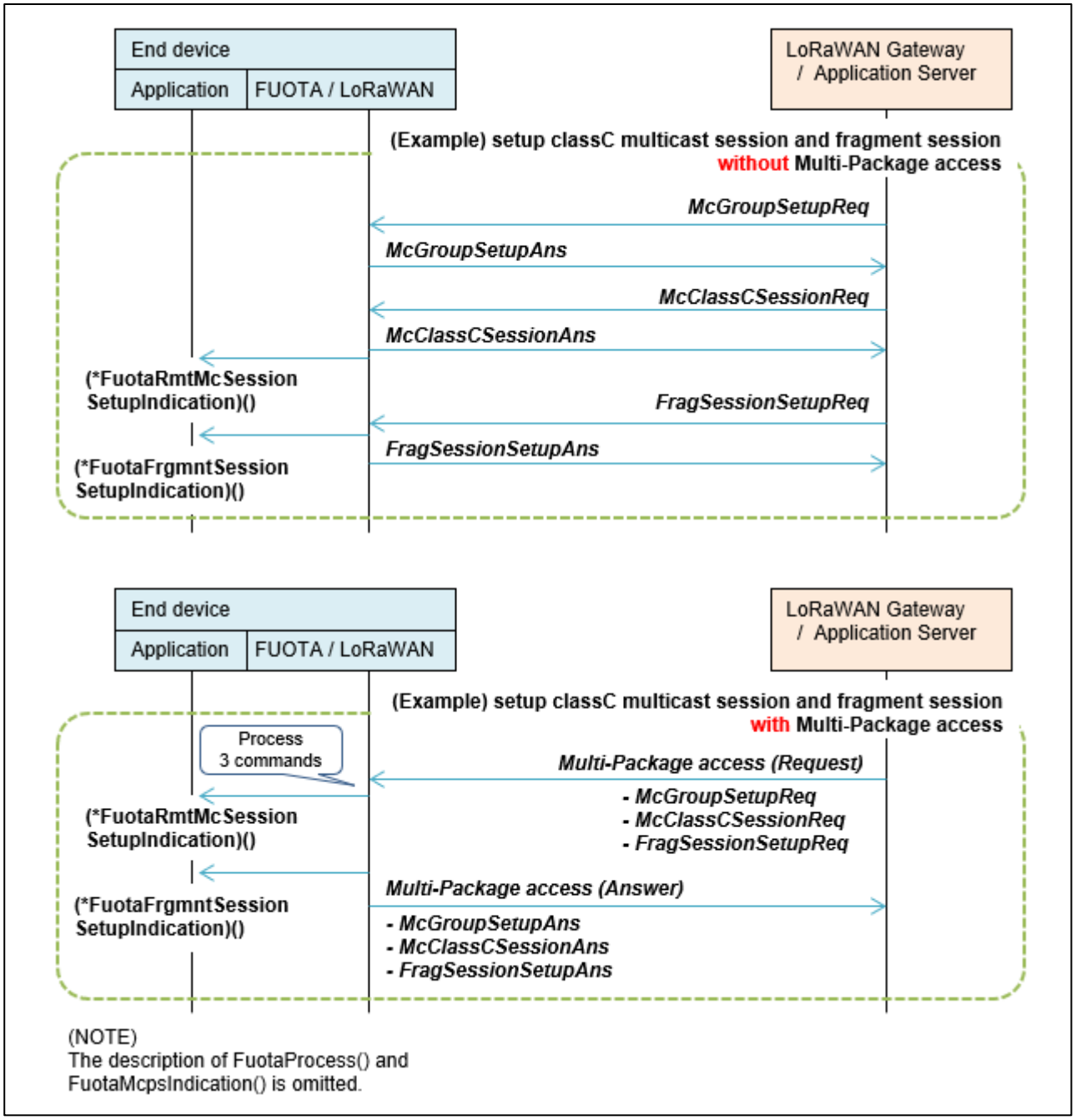


Figure 28 Multi Package Access Protocol (Send Several Commands)

A.2.7 FUOTA Sample Application

A.2.7.1 AT Commands for the FUOTA Sample Application

There are 3 additional indications for FUOTA V2.0.0 (`<indication>=129`, `240`, and `241`). Other indications are same as the ones of FUOTA V1.0.0 (see 2.8.4).

AT command	Description
+FUOTAIND:<indication>,<param1>,<param2>,<param3>,<param4> <indication> FUOTA evet indication type <param1> - <param4> Depends on indication value.	Event indication from FUOTA layer ● When <indication> is 0: Indicate an event of the RemoteMulticast session setup <param1> Session Class ID. 1:ClassB / 2:ClassC <param2> Multicast Group ID <param3> Seconds to start session <param4> Seconds to timeout session ● When <indication> is 1: Indicate an event of the RemoteMulticast session start. <param1> Session Class ID. 1:ClassB / 2:ClassC <param2> Multicast Group ID <param3> Seconds to timeout session ● When <indication> is 2: Indicate an event of the RemoteMulticast session end. <param1> Session Class ID. 1:ClassB / 2:ClassC <param2> Multicast Group ID ● When <indication> is 128: Indicate an event that F/W image is ready. <param1> - <param4> are omitted. **Following are additional indications from FUOTA V2.0.0:** ● When <indication> is 129: Indicate an event that F/W image is removed. <param1> Version of removed F/W image <param2> - <param4> are omitted. ● When <indication> is 240: Indicate an event that a server request to reboot device. <param1> Seconds to reboot device. 0xFFFFFFFF means previous reboot request is canceled. <param2> - <param4> are omitted. ● When <indication> is 241: Indicate an event that reboot time has come. <param1> - <param4> are omitted.

A.2.7.2 Example AT Command Operation of End Device

AT command operation of the end device for the FUOTA V2.0.0 shown in below.

Operations for joining network, setup and start FUOTA are same as the ones for FUOTA V1.0.0. See 4.3 for details.

After F/W image is ready (`+FUOTAIND: 128`), the application server can request the reboot to the end device using firmware management protocol. If requested, the end device will update the firmware at the reboot timing which the application server requests.

Example operation and notification for end device	**Description**
(See 4.3) ● Confirm the current version of the FUOTA sample application. ● Setup parameters for network joining. ● Request to join the network. ● Setup parameters for FUOTA. ● Request to start FUOTA. : *[Receive F/W image]* :	
+FUOTAIND: 128	Indication of the F/W image is ready. Application server can request the reboot to the end device using the firmware management protocol. This example assumes that the application server requests the reboot after 10 seconds.
+FUOTAIND: 240,10	Indication of the request to reboot after 10 seconds. (After 10 seconds)
+FUOTAIND:241	Indication of come the reboot time. Firmware update can be started.
AT+FUOTAUPDT OK	Update the firmware after the FUOTA sample application validates the F/W image. After the update, the updated FUOTA sample application will be activated.
(See 4.3) ● Confirm the version of the FUOTA sample application has been updated.	

Appendix.B. Example Operations of LoRaWAN Server to Perform FUOTA

This appendix describes the example operations on how to create and schedule a FUOTA task with multicast.

In this sample,

- In case that MultiTech Conduit AEP which supports FUOTA V1.0.0 is used for the LoRaWAN server.
- In case that AWS IoT Core for LoRaWAN with Kerlink iFemtoCell LoRaWAN gateway.

B.1 Example operations for FUOTA in case of MultiTech Conduit AEP

Example operations for FUOTA in case of MultiTech Conduit AEP is shown below. Before starting FUOTA task on the network server, the end device needs to join the network and start FUOTA (for example, execute `AT+FUOTASTRT`).

Create Multicast Group

1. Click **LoRaWAN** > **Device Groups** in the left side menu.
2. Click **Add NEW**, and fill out the fields as follows in **ADD GROUP** window.
   - Enter a multicast group name to **Group Name**.
   - Click the check box(es) to the left of **Device EUI** to be added to the multicast group in **End Device Selection**.
   - Click **OK**.

Create and schedule FUOTA task

1. Click **LoRaWAN** > **Operations** in the left side menu.
2. Click **Schedule**.
3. Select **FOTA** in **Operation Type**.
4. Click **Firmware Upgrade File** and select F/W image file.
   For example, sample_fwimage_0x00000000.bin.
5. Select **Countdown To Setup From Setup** in **Setup Time Input** and enter relative time (hours, minutes, and seconds) in **HH:MM:SS** when to start the setup of FUOTA.
6. Select **Countdown To Launch From Setup** in **Launch Time Input** and enter relative time (hours, minutes, and seconds) in **HH:MM:SS** when to start the session of FUOTA.
7. Select **End-device Group** under **Target End-Devices.**
8. Click the check box(es) to the left of **Group Name** to be updated**.**
9. Click **Submit**.
10. Click **Progress** to see the progress of FUOTA session.

B.2 Example operations for FUOTA in case of AWS IoT Core for LoRaWAN

Example operations for FUOTA in case of AWS IoT Core for LoRaWAN is shown below. You need to get login account in advance if you don't have it.

Go to the AWS IoT Core for LoRaWAN

1. Login to the AWS (https://aws.amazon.com/).
2. Select **IoT Core** in **Services**.

B.2.1 Preparing AWS

B.2.1.1 Register the LoRaWAN Gateway

1. Click **Manage** > **LPWAN devices** > **Gateways** in the left side menu.
2. Click **Add gateway**, and fill out the fields as follows in **Add gateway** section.
   - In **Gateway details:**
     - Enter Gateway's EUI to **Gateway's EUI** and **Confirm gateway's EUI**.
     - Select RF region from **Frequency band (RFRegion)**.
     - **Name** and **Description** is optional.
     - No other items need to be changed.
   - Click **Add gateway**.
3. In **Configure your gateway** section:
   - In **Gateway certificate**, click **Create certification** and click **Download certificate files** to download gateway certificate file and private key file. These are used to setup the gateway.
   - In **Provisioning credentials**, copy CUPS and LNS endpoints and save them. And click Download server trust certificates to download the CUPS and LNS server trust certificates. These are also used to setup the gateway.
   - In **Gateway permissions**, select **IotWirelessGatewayCertManagerRole** to make IAM role.
   - Click **Submit**.

B.2.1.2 Add Device Profile(s)

1. Click **Manage** > **LPWAN devices** > **Profiles** in the left side menu.
2. Click **Add device profile**, and fill out the fields as follows in **Add device profile** section.
   - In **Device profile**:
     - Select default profile from **Select a default profile and customize**.
     - No other items need to be changed.
   - Click **Add device profile**.

B.2.1.3 Add Service Profile(s)

1. Click **Manage** > **LPWAN devices** > **Profiles** in the left side menu.
2. Click **Add service profile**, and fill out the fields as follows in **Add service profile** section.
   - In **Service profile**:
     - Enter a profile name to **Service profile name**.
     - No other items need to be changed.
   - Click **Add service profile**.

B.2.1.4 Add Destination(s)

1. Click **Manage** > **LPWAN devices** > **Destinations** in the left side menu.
2. Click **Add destination**, and fill out the fields as follows in **Add destination** section.
   - In **Destination details**:
     - Enter a destination name to **Destination name**.
     - Select **Enter a rule name**, and enter a rule name.
     - No other items need to be changed.
   - In **Permissions**:
     - Select **Create a new service role**.
     - You can enter a custom role name to **Role name**.
   - Click **Add destination**.

B.2.1.5 Add LoRaWAN Device(s)

1. Click **Manage** > **LPWAN devices** > **Devices** in the left side menu.
2. Click **Add wireless device**, and fill out the fields as follows in **Add device** section.
   - In **LoRaWAN specification and wireless device configuration**:
     - Select **OTAA v1.0.x** in **Wireless device specification**.
     - Enter device EUI of the added device in **DevEUI** and **Confirm DevEUI**.
     - Enter AppKey in **AppKey** and **Confirm AppKey**.
     - Enter AppEUI in **AppEUI** and **Confirm AppEUI**.
     - No other items need to be changed.
   - In **FUOTA configuration**:
     - Enter GenAppKey in **GenAppKey** and **Confirm GenAppKey**.
     - No other items need to be changed.
   - In **Profiles**:
     - Select device profile (see B.2.1.2) in **Wireless device profile**.
     - Select service profile (see B.2.1.3) in **Service profile**.
   - In **Choose destination**:
     - Select destination name (see B.2.1.4) in **Destination name**.
   - Click **Next**.
3. No need to change items in **Set device position – optional** section.
   - Click **Add device.**

B.2.2 Preparing LoRaWAN Gateway

Please refer to the manual of your LoRaWAN gateway.

In case of Kerlink iFemtoCell LoRaWAN gateway,

1. Get Login Account for Kerlink Website (https://www.kerlink.com/).
2. Get Installation Manual for iFemtocell from Kerlink Website.
3. Turn on the Kerlink iFemtoCell.
4. Update Kerlink firmware for AWS.
5. Register certificate file, private key file, and the server trust certificate downloaded from AWS (see B.2.1.1) to the gateway.

B.2.3 AWS Operation for FUOTA

Before starting FUOTA task on the network server, the end device needs to join the network and start FUOTA (for example, execute `AT+FUOTASTRT`).

B.2.3.1 Create Multicast Group(s)

If you want to perform FUOTA by multicast, you need to create a multicast group.

1. Click **Manage** > **LPWAN devices** > **Devices** > **Multicast groups** in the left side menu.
2. Click **Create multicast group**, and fill out the fields as follows in **Create multicast group** section.
   - In **Multicast properties**:
     - Enter a desired multicast group name in **Name**.
   - Click **Next**.
3. In **Add devices to multicast group** section:
   - In **Add device**:
     - Select RF region in **RFRegion**.
     - Select **Class C** in **Select multicast device class**.
     - Click **Add individual devices** and enter the device IDs which you registered (see B.2.1.5) in **Enter the device ID of devices you wish to add to your multicast group**.
   - Click **Create**.
4. Click multicast group you made in **Multicast groups**.
5. In **Added devices** section, check if the device status is **Multicast setup ready** (green letters). If not, click reload button to update status.

B.2.3.2 Create FUOTA Task

1. Click **Manage** > **LPWAN devices** > **Devices** > **FUOTA tasks** in the left side menu.
2. Click **Create FUOTA task**, and fill out the fields as follows in **FUOTA properties** section.
   - In **Task properties**:
     - Enter a desired task name in **Name**.
     - Select RF region in **Frequency band (RFRegion)**.
   - Click **Next**.
3. In **FUOTA configuration** section:
   - In **Configure**:
     - Click **Upload a new firmware image**, click **Choose file**, and select F/W image file (for example, sample_fwimage_0x00000000.bin). After that, if no S3 bucket is available, **Create S3 bucket** and enter the bucket name.
       Or, click **Select an existing firmware image** if you want to use the F/W image file which has already uploaded. After that, click **Browse S3** and select bucket.
   - In **Permissions**:
     - Select **Create a new service role**.
     - You can enter a custom role name to **Role name - optional**.
   - Click **Next**.
4. In **Review and create section**, click **Create task**.

B.2.3.3 Schedule FUOTA Task

1. Click **Manage** > **LPWAN devices** > **Devices** > **FUOTA task** in the left side menu.
2. Click **Task ID** in the list of FUOTA tasks which you created (see B.2.3.2).
3. In **Devices**, click **Add device**, and fill out the fields as follows in **Add devices** section.
   - In **Add devices**:
     - Select RF region in **Frequency band (RFRegion)**.
     - If you want to perform FUOTA by multicast, click **Select multicast groups** and select multicast group which you created (see B.2.3.1) in **Multicast groups to update**.
   - Click **Save**.
4. Click **Schedule FUOTA task**, enter start date and time. The date and time must be at least 30 minutes later from the current. Click **Schedule**.

Revision History

Rev.	Date	Description	
		Section	Summary
01.00	Oct.9.20	-	Initial release
03.00	Mar.26.21	4.3	Changed the versions from V03.00 to V09.00, and from V02.10 to V03.00 in the example operations
03.10	Sep.30.21	-	Supported RL78/G23 (R7F100GSN) as a target device
		2.3.2	Modified the range of FUOTA_IB_PROC_POLLING_FPORT
		2.8.4	- Added AT+GENAPPKEY description - Deleted FUOTA_IB_RMTMC_GENAPPKEY - Modified the ID of FUOTA_IB_CLKSNC_FORCESYNC_PERIOD_SEC
		4.2.1	Added the argument 'MCU' for 'make_fwimage.bat'
		4.3	Changed the version from V03.10 to V09.00 in the example operations
		4.1.1, 4.1.2	Added a section for the hardware setup and configuration of sample application
		4.2.1	Added a section for basic configuration of LoRaWAN network server
03.12	Jan.21.22	Table 2-3	Removed 'Real Time Clock (RTC)' for correction. Added I/O ports use in Application layer for correction.
		Table 2-4	Added I/O ports use in Application layer for correction.
		Table 4-1	Added a column for default settings. Added 'LoRaWAN Regional Parameters RP002-1.0.3' for LORAWAN_VERSION_1_0_4. Added 'LoRaWAN 1.0.3 Regional Parameters Revision A' for LORAWAN_VERSION_1_0_3. Added 'AS923-1 for Japan' for REGION_AS923.
04.00	Aug.29.22	1.2	Changed the document number and title of [7]
04.10	Nov.29.22	Table 1	Added "FUOTA V1.0.0" and "FUOTA V2.0.0"
		Table 3	Updated URL.
		2.3.1	Added a section for the macros for FUOTA setting.
		2.6.4	Updated description about descriptor parameter.
		2.8.2 Figure 11	Removed 'AppFuotaUpdateStartFragment()' function.
		2.8.3	Updated description about 'FUOTAUPDT_STATE_FAILED'.
		Table 11	Changed the name from 'F/W image Header' to 'F/W image information'
		Table 12	Added 2 columns for default settings: 'FUOTA_VERSION_V1_0_0' and 'RP_USE_RADIO_CFG_CHECK'
		4.2.2	Removed 'desc' from output file name.
		4.2.3	Updated description.
		4.3	Removed 'desc' from sample F/W image file name. Changed the version from Ver3.12 to Ver4.10. Updated the description of example operation.
		Appendix. A	Added appendix for the FUOTA V2.0.0.
		Appendix. B	Added appendix for the network server operation.
04.20	Mar.31.23	-	Supported RA2L1 (R7FA2L1AB) as a target device.
		2.8.4	Added AT+FUOTAUPDT description.

		A.2.3.2	Removed Table 15 and FUOTA V2.0.0 configuration: ‘FUOTA_CONFIG_FWMNG_FWVERSION’ and ‘FUOTA_CONFIG_FWMNG_HWVERSION’.
		Table 15 A.2.5.6	Added Callback function: FuotaFwMngVersionInfoRequest()
		Figure 24	Changed a flow diagram when the end device receives the requests of firmware and hardware version from application server.
		B.2	Added appendix for AWS operation.
4.40	Dec.22.23	1.2	Replaced related document [4].
		Table 15	Updated description about RP_USE_RADIO_CFG_CHECK.
		4.3	Changed the version from Ver4.20 to Ver4.40.
4.50	May 24.24	4.3	Changed the version from Ver4.40 to Ver4.50.
4.70	Apr 18.25	1.2	Updated document title [7].
		3.4.1	Added work area in code flash memory to prepare for power interruption.
		3.4.2	Added RAM mapping description.
		4.3	Changed the version from Ver4.50 to Ver4.70.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Arm® and Cortex® are registered trademarks of Arm Limited. Semtech, the Semtech logo, LoRa, LoRaWAN and LoRa Alliance are registered trademarks or service marks, or trademarks or service marks, of Semtech Corporation and/or its affiliates. Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.