AN5185

Application note

ST firmware upgrade services for STM32WB series

Introduction

This document describes the firmware upgrade services (FUS) available on STM32WB series microcontrollers. These services are provided by STMicroelectronics code located in a secure portion of the embedded flash memory, and are used by any code running on Cortex®-M4 with a user flash memory, or through embedded bootloader commands (also running on Cortex®-M4).

1 General information

This document applies to STM32WB series Arm®-based devices.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 Firmware upgrade services definition

FUS (firmware upgrade services) is a firmware running on STM32WB Cortex®-M0+ and offering following features:

1. Install, upgrade or delete STM32WB Cortex®-M0+ wireless stack:
   - Only encrypted and signed by STMicroelectronics
   - Optionally, additionally double signed by customer if needed
2. FUS self-upgrade:
   - Only encrypted and signed by STMicroelectronics
   - Optionally, additionally double signed by customer if needed
3. Customer authentication key management:
   - Used for images double signature
   - Install, update and lock the customer authentication key
4. User key management:
   - Store customer keys
     - Master key
     - Simple clear key
     - Encrypted key (by master key)
     - In secure area accessible only by Cortex®-M0+ code.
   - Write stored key (simple or encrypted) into AES1 (advanced encryption standard) in secure mode (the Cortex®-M4 cannot access the key)
   - Lock a stored key to prevent its usage until next system reset
   - Unload a previously loaded key from AES to prevent its usage by other applications
   - Key width: 128 or 256 bits
   - Up to 100 user keys (encrypted by master key or clear) and one user master key
5. Communication with Cortex®-M4 (user code or bootloader):
   - Through IPCC commands and response model (same as wireless stack model)
   - Commands already supported by STM32WB bootloader (in ROM)

1.2 FUS versioning and identification

The user needs to read the shared table memory in SRAM2a to identify the FUS version, as explained in Section 1.6 Shared tables memory usage and in Section 6.1 Shared tables usage.

The first word in SRAM2a pointed by IPCCDBA Option Bytes is the "Device info table" address. This table (described in Table 7) contains the FUS version at offset 0xC which is encoded on four bytes. Typically, if IPCCDBA = 0x0000 and @0x20030000 contains 0x20030024, then the FUS version is @0x20030030.

Installation of a FUS image must follow the conditions stated in the image binary release notes.

Note: When using the SWD interface with the STM32CubeProgrammer (STM32CubeProg) older than V2.7.0, the address of the device information table is located at 0x20030890. For STM32CubeProgrammer V2.7.0 and higher, the device information table is located at 0x20030024.

Table 1. FUS versions

Version	Description
V0.5.3	Default version programmed in production for all STM32WB5xx devices. Must be upgraded to V1.0.1 on STM32WB5xG devices or to V1.0.2 on STM32WB5xE/5xC devices. This version is not available for download on www.st.com and cannot be installed by users.
V1.0.1	First official release available on www.st.com and dedicated to STM32WB5xG devices only (1 Mbyte flash memory size) This version must not be installed on STM32WB5xE/5xC devices, otherwise the device enters a locked state and no further updates are possible.
V1.0.2	First official release available on www.st.com and dedicated to STM32WB5xE/5xC devices (512 Kbytes and 256 Kbytes flash memory size) Use the V1.0.2 on the STM32WB5xG devices if the devices present FUS V0.5.3. If an STM32WB5xG device has FUS V1.0.1, then there is no need to upgrade to V1.0.2, as it does not bring any new feature/change vs. V1.0.1. In case FUS V1.0.2 installation is started by user on an STM32WB5xG device with FUS V1.0.1, FUS returns FUS_STATE_IMG_NOT_AUTHENTIC error and discard the upgrade.
V1.1.0	FUS update to support following features: • Add FUS_ACTIVATE_ANTIROLLBACK command that allows activating Anti-rollback on wireless stack by user. User can activate this feature to prevent any installation of older wireless stack. • Replace Safeboot by V1.1.0 version (replace full chip lock by factory reset) • Add factory reset in case of Flash ECC, corruption or Option Bytes corruption error. Factory reset means erase of wireless stack if present and reboot on FUS and full erase of other user sectors. FUS V1.1.0 can be installed only on devices containing V1.0.1 or V1.0.2 FUS. In case a device has V0.5.3 installed, user must first install V1.0.2 then install V1.1.0. When installing FUS V1.1.0 over an FUS V0.5.3 results in FUS_STATE_IMAGE_NOT_AUTHENTIC error and discarding the upgrade.
V1.1.1	FUS update to support STM32WB5xx 640 KB products. This version is not available on www.st.com, and cannot be used for upgrade. This version is fully compatible with V1.1.0, and does not present any difference, except management of the new 640 KB products.
V1.1.2	FUS update to: • Optimize flash usage: makes possible to install a stack, maintaining a one sector separation below a previously installed stack (instead of stack size space constraint explained in Section 2 Wireless stack image operations) • Security enhancements To upgrade from FUS V1.1.0 to FUS V1.1.2 , the Anti-rollback must first be activated. Before activating Anti-rollback, a wireless stack installed must be present. Upgrading from V1.1.0 to V1.2.0 is possible without constraints nor additional operations from the user.

Version	Description
V1.2.0	FUS update to: • Includes V1.1.2 FUS updates in production • Allows direct update from FUS V1.1.0 to FUS V1.2.0 without activating the Anti-rollback. • Allows direct update from FUS V0.5.3 to FUS V1.2.0 (without installing intermediate FUS versions) • Security updates Upgrading from FUS V1.1.0 or any other FUS version, to FUS V1.2.0 is possible without constraints and no interaction from the user.

The table below details the FUS versions compatibility options (when it is possible to upgrade from a version to another). FUS V1.2.0 is the version that allows the upgrade from any previous version. It is released in two binaries:

- `stm32wb5x_fus_fw_V1.2.0.bin` : for upgrades from any FUS version V1.x.y
- `stm32wb5x_fus_fw_V1.2.0_for_V0.5.3.bin`: upgrades from FUS version V0.5.3

Note: STM32WB10xx and STM32WB15xx have only FUS V1.2.0, which is fully compatible with STM32WB5xxx FUS V1.2.0, but does not provide user key services.

Table 2. FUS versions compatibility

Upgrade		To					
		V0.5.3	V1.0.2	V1.1.0	V1.1.1	V1.1.2	V1.2.0
From	V0.5.3	X	√	X	X	X	√
	V1.0.2	X	X	√	X	√	√
	V1.1.0	X	X	**X**	X	(1)	√
	V1.1.1	X	X	X	X	√	√
	V1.1.2	X	X	X	X	√	√
	V1.2.0	X	X	X	X	X	√

Legend:
- X: Cannot upgrade
- √: Upgradable
- **X**: Must not upgrade, otherwise encryption keys are lost
- (1): Upgradable but a BLE stack needs to be installed first and enable Anti-rollback

A FUS version is available from two different sources:

- Programmed directly in the STM32WB series devices by STMicroelectronics during the production phase.
- Available from www.st.com. This method is used mainly for the FUS version upgrade process.

The following table details the availability of each version at production and on www.st.com.

Table 3. FUS versions availability

Version	Production	Binary on www.st.com
V0.5.3	√	X
V1.0.2	√	√
V1.1.0	√	√
V1.1.1	√	X
V1.1.2	X	√
V1.2.0	√	√
Legend: • X: Not available • √: Available		

1.2.1 Known limitations

This section details the known limitations on the latest version of the FUS (V1.2.0).

- Upgrade error
  Possible upgrade error in the case of external power loss or reset events. This is in the event of external power loss, or forced reset during the firmware upgrade operation. The ongoing operation might be corrupted. In that case FUS abort the operation and return an error message. Workaround: When an error message is returned, repeat the firmware upgrade operation from the beginning. Special care must be taken in the case of OTA (on the air upgrade) where the wireless stack might be needed to download again the image.
- Wireless stack
  If a wireless stack "A" is installed, then another wireless stack "B", larger by exactly one sector than the wireless stack "A" must be installed. Then, FUS rejects the operation unless the wireless stack "B" is loaded at an address "add" `< 0x080F4000 - 3x SizeOf("B")`. The workaround is to add padding to the wireless stack to avoid the condition of one sector difference in size. This limitation can be seen when using some wireless stacks present on STM32CubeWB V1.14.0. For more details, refer to the STM32CubeWB release notes. From STM32CubeWB v1.14.1 release, the size of the wireless firmware binaries is controlled to guarantee at least two sectors size difference between all generated binaries to workaround the limitation.

1.3 How to activate FUS

The FUS runs on Cortex®-M0+ and on the protected flash memory zone dedicated for FUS and wireless stack. There are two possible situations:

Table 4. FUS activation cases

Situation	How to activate FUS
No wireless stack is running (for example the first time the STM32WB series device is running or the wireless stack has been removed)	Ensure Cortex®-M0+ is activated by setting C2BOOT bit in PWR_CR4 register Ensure IPCCDBA (Option Bytes) points to a valid shared table information structure in SRAM2a (enter the correct pointers to device information table and system table) *Note: Both conditions are performed automatically by system bootloader. So if device boot is configured on system memory, the FUS must be activated, with no need for further user actions.* Otherwise, these actions must be performed by user code running on Cortex®-M4 CPU.
Wireless stack is installed and running	Perform same steps as above Request wireless stack to launch FUS by sending two consecutive FUS_GET_STATE commands. The first one must return FUS_STATE_NOT_RUNNING state, the second causes FUS to start.

To check if FUS is running or not, the following options are available:

- Send a single FUS_GET_STATE command and check the return status. If it is FUS_STATE_NOT_RUNNING then FUS is not running.
- Check the SBRV Option Bytes value:
  – if it is 0x3D800 (for FUS V0.5.3) or 0x3D000 (for FUS V1.x.z) then FUS must be running
  – If it is different from 0x3D800 (for FUS V0.5.3) and from 0x3D000 (for FUS V1.x.z) then FUS is not running
- Send a wireless stack command:
  – If it is acknowledged, then FUS is not running
  – If it is not acknowledged, then FUS is running
- Read the shared table information:
  – Read IPCCDBA (in Option Bytes) to get the shared tables start address in SRAM2a
  – Get the device information table address
  – Read the field “Last FUS active state”
    ◦ 0x04 means that stack must be running
    ◦ Other values mean that FUS must be running
  – Read the "Async Ready" event that is sent by FUS at startup. For more information about this event and content, refer to Section 6.3.2 Event packet.

1.4 Memory mapping

The FUS has a dedicated space in flash memory that depends on the FUS size. It also uses a dedicated space in SRAM2a and SRAM2b, and a shared space in SRAM2a (shared tables). The size of the dedicated space in flash memory, SRAM2a and SRAM2b is defined by Option Bytes. For more information, refer to the product reference manual.

The dedicated flash memory and SRAM areas are shared with the wireless stack if it is installed. But at a given time, either FUS or wireless stack is running on Cortex®-M0+.

Figure 1. Flash memory mapping

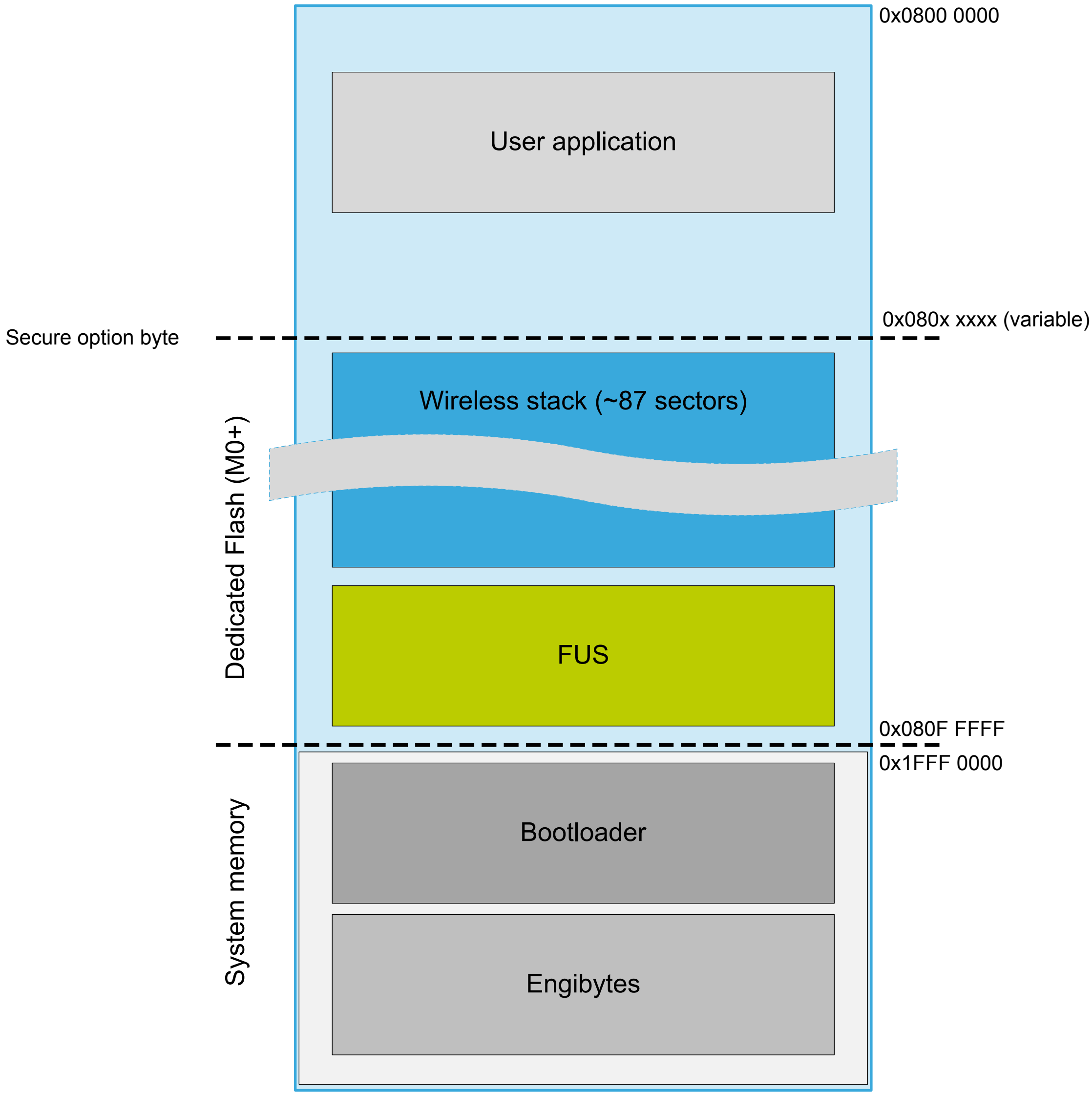
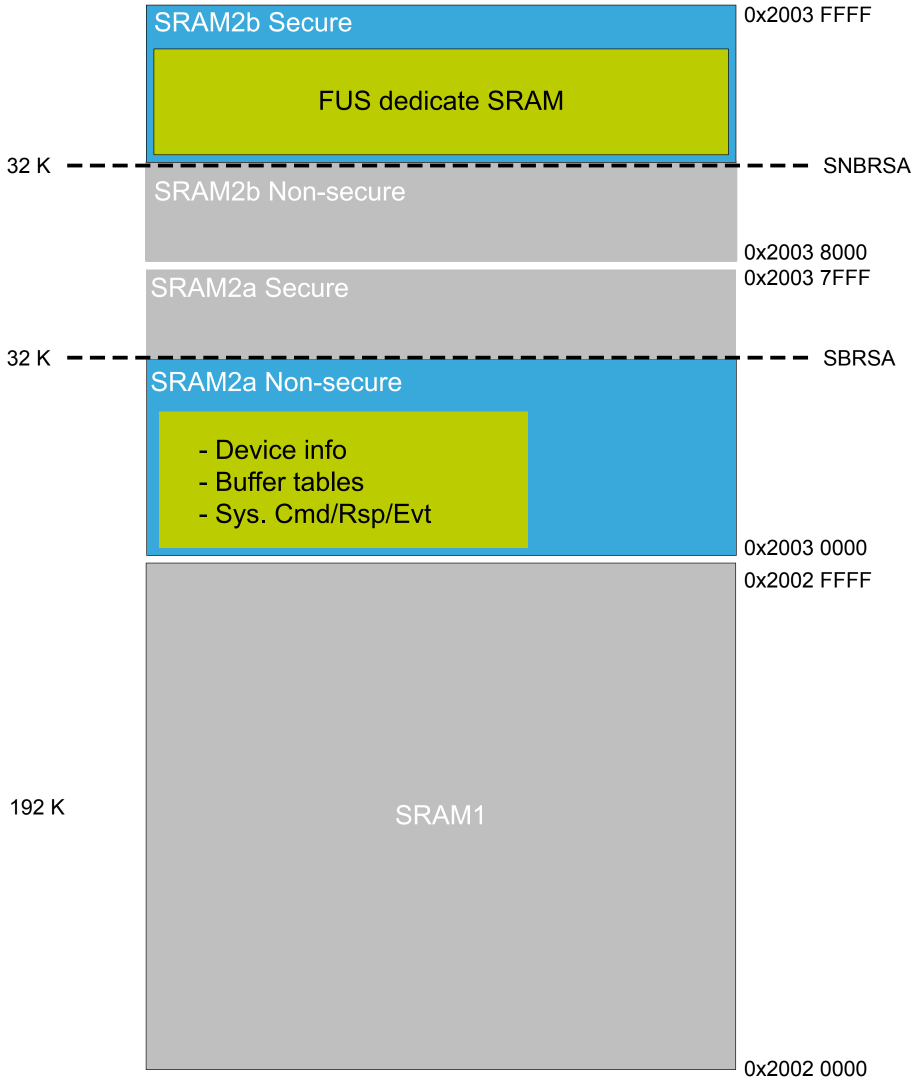


Figure 2. SRAM memory mapping



1.5 FUS resources usage

The FUS only configures/uses the resources listed in Table 5.

The RCC (reset and clock control), flash memory, PWR (power control) and all necessary components for the STM32WB series microcontroller normal operation must be configured by Cortex®-M4 application prior to enabling Cortex®-M0+ (it is automatically done by system bootloader when started).

Table 5. FUS resources usage

Resource	Case	Configuration
Flash	Always	Dedicated flash is used by FUS depending on its size and on the size of the current wireless stack and the image requested to be installed. Parts of the dedicated flash memory may be written and/or erased during FUS operations. **Caution:** Take care of operations performing write/erase cycles on the flash memory while FUS is running.
SRAM2b	Always	SRAM2b secure area is used by FUS depending on its version.
SRAM2a	Always	SRAM2a secure area is used by FUS depending on its version. SRAM2a public area is used by FUS to write into shared tables for information table and commands table.
IPCC	Always	IPCC is used by FUS for mail boxing between Cortex®-M0+ and Cortex®-M4 user application or bootloader or JTAG. Two channels are used: P1CH2 (command/response channel) and P1CH4 (trace channel).
PKA	When install is required	PKA is enabled, configured and used for signature verification.
AES1	When key service is required	AES1 is configured in secure mode (key register is accessible only by Cortex®-M0+) AES1 key register is written by FUS with the key requested by user. Once AES1 is configured in secure mode, it remains in secure mode until next system reset, or FUS_UNLOAD_USR_KEY command is executed.
Option Bytes	When install/ delete is required	Option Bytes are programmed by FUS using Cortex®-M0+ registers: only SFR and SBRR registers are modified.
CRC	When install is required	CRC is used for authentication and it is not initialized by FUS. If CRC is used by Cortex®-M4 user application, it has to be reset before starting FUS or wireless stack install operations.
System Reset	When install/ delete is required	FUS forces the System Reset when loading Option Bytes or after critical errors detection.
NVIC	Always	Following handlers are used: • NMI • SysTick • IPCC_C2_RX_C2_TX_HSEM

Important: *During FUS or wireless stack upgrade/delete operations, Cortex®-M4 and SWD shall not:*

- *Perform any write/erase operation on Flash*
- *Perform any write on Option Bytes*
- *Change PWR and RCC configuration.*

If any of the above operations are performed during FUS or wireless stack upgrade/delete, there is a risk of corrupting the flash and losing data.

Important: In case power supply failure occurs during a FUS operation (install/delete), one of the following 3 cases may occur:

- *Power failure without impact: If the flash content is not corrupted, FUS recovers the failure and continues operating without the need for any user intervention.*
- *Power failure with flash corruption: the flash content is corrupted, the image is not installed by FUS (rejected as non-integer). FUS erases the image and generates an error (FUS_ERR_IMG_CORRUPT). User must restart the whole operation by re-loading the binary and send an upgrade command to FUS.*
- *Power failure with option bytes corruption: Safeboot is started by hardware and all the flash is locked by hardware. In this case, if FUS V1.1.0 or higher version is running, then a factory reset is triggered (user shall activate CM0 by writing the value 0x00008000 at the address @0x5800040C). If a FUS version lower than V1.1.0 is running, then, no recovery is possible at this point.*

Note: If there are user keys stored by FUS, when FUS is upgraded, these keys are erased.

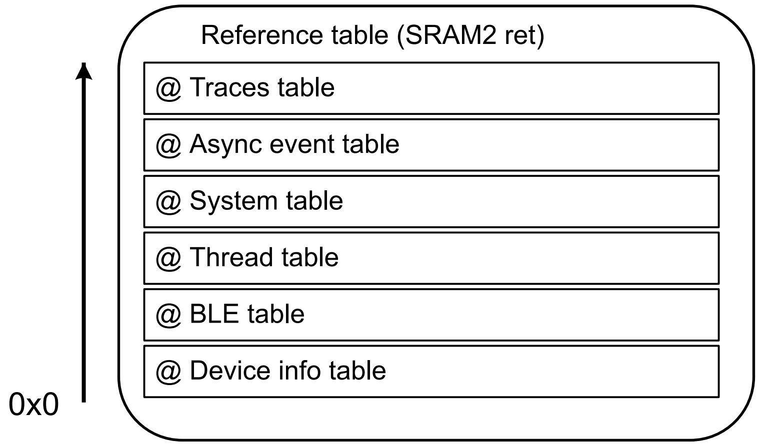
1.6 Shared tables memory usage

Communication data buffers are pointed to by lookup table for which the address is determined by an Option Byte: IPCCDBA (IPCC mailbox data buffer base address). This address provides the base address of the buffer tables pointers as detailed in *Building a wireless application* (AN5289).

If IPCCDBA points to an address that does not fit all table pointers such as (SRAM2a_END_ADDRESS - SharedTable_BaseAddress) < SizeOf(SharedTable), then the FUS must discard usage of shared table completely and thus no communication or commands are possible with FUS.

User application has to setup the shared table base address correctly, otherwise it must stop the FUS services initialization.

Figure 3. Shared table architecture



FUS uses only two tables:

- Device information table: this table provides useful information from FUS to the Cortex®-M4 user application (or JTAG) at startup (content written by FUS at startup).
- System table: this table allows the exchange commands and responses between FUS and Cortex®-M4 user application.

2 Wireless stack image operations

The FUS allows the user to install, upgrade and delete the wireless stack.

The wireless stack has to be provided by STMicroelectronics (encrypted and signed) in order to be installed by FUS. The user may add a custom signature to the wireless stack image binary using the ST tools and as detailed in Section 4 User authentication (if the user authentication key has already been loaded by FUS).

The wireless stack install, upgrade and delete operations are performed through the bootloader, JTAG, or user application. STM32CubeProgrammer provides the tools to perform this operation through the bootloader interfaces: USART and USB-DFU, and also directly through SWD interface.

2.1 Wireless stack install and upgrade

Here are two definitions that are useful to remember:

- Wireless stack install: means the first installation on a chip where there is no wireless stack already installed.
- Wireless stack upgrade: means the installation on a chip where a wireless stack is already installed (may be running or not).

Operation instructions

In order to perform a wireless stack install or upgrade, follow the procedure below:

1. Download the wireless stack image from www.st.com or from the STM32CubeMX repository.
2. Write the wireless stack image in the user flash memory at the address: DownloadAddress = 0x08000000 + (SFSA x SectorSize) - ImageSize - 1 x SectorSize, where:
   – DownloadAddress is the address where the new wireless stack is loaded, aligned to sector size
   – SFSA is the Secure Option register value indicating the current boundary of flash memory secure area
   – SectorSize is 4 KB for STM32WB5xxx, and 2 KB for STM32WB1xxx
   – ImageSize is the size, in bytes, of the binary to install

   When using STM32CubeProgrammer, the best placement is suggested based on SFSA and binary size.
   A FUS_FW_DELETE operation is preferred before starting a new wireless stack installation, but it is not mandatory. Selecting the option "-firstinstall=0" or unchecking the box "First Install" on STM32CubeProgrammer forces the delete.
   In the case where the new wireless stack size is larger than the already installed stack, read "Memory instructions" in Section 2.2 Wireless stack delete.
3. Ensure FUS is running (follow steps in Section 1.3 How to activate FUS).
4. Send FUS_FW_UPGRADE command through IPCC mechanism (explained in sections below).
5. Send FUS_GET_STATE until a state equal to FUS_STATE_NOT_RUNNING is reached (this means that the wireless stack has been installed and is now running).

During the installation process, expect multiple system resets to occur. These system resets are performed by FUS and are necessary for the modification of dedicated memory parameters and to make Cortex®-M0+ run the installed wireless stack. The number of system resets depends on the configuration and the location of new and old images.

Table 6 explains possible errors when the install/upgrade operation is requested and their respective results.

Table 6. FUS upgrade returned errors

Error	Reason	Result
Not enough space	Space between current installed wireless stack and the address of loaded image is too small.	Installation request is rejected. FUS returns error state, and goes back to an Idle state.
Image signature not found	Incorrect or corrupted signature header or body.	FUS returns an authentication error, then goes back to idle state. The image is not installed and no changes on flash memory/SRAM.
Image customer signature not found	Incorrect or corrupted signature header or body.	FUS returns an authentication error, then goes back to an idle state. The image is not installed and no changes on flash memory/ SRAM.

Error	Reason	Result
Image corrupted	Incorrect image header or corrupted image.	FUS returns an image corruption error, then goes back to an idle state. The image is not installed and it is erased by FUS.
No state is returned by FUS	A reset performed by FUS has occurred before receiving the command response.	Command resending must result in receiving a FUS response.
Other failures	External power interruption or external reset during FUS operation	FUS must be able to recover and delete the corrupted image, and go back to default state. It can perform several system resets to complete the recovery operation.

Memory considerations

At first install or when no wireless stack is installed, the FUS does not make any optimization on the address where the wireless stack is installed. The wireless stack image must be installed at the same address where it has been loaded by user.

At wireless stack upgrade (a wireless stack is already installed), the FUS may move the upgraded stack after upgrade and before running it.

The remaining space in this case is left free for Cortex®-M4 user application usage.

After the install/upgrade operation is successfully completed, the SRAM2a, SRAM2b, flash memory secure boundaries and SBRV values are changed according to requirements of the installed wireless stack.

2.2 Wireless stack delete

Wireless stack delete means removing the wireless stack already installed on a chip (whether it is running or not).

Operation instructions

To perform a wireless stack delete go through the following steps:

1. Ensure FUS is running (follow steps in Section 1.3 How to activate FUS)
2. Send FUS_FW_DELETE command through IPCC mechanism (explained in following sections)
3. Send FUS_GET_STATE until FUS_STATE_IDLE is reached with error code FUS_STATE_NO_ERROR. During the delete process, expect multiple system resets to occur. These system resets are performed by FUS and are necessary for the modification of dedicated memory parameters. The number of system resets depends upon the configuration and the location of the wireless stack.

If no wireless stack is installed and a delete request is sent, then the FUS returns an error state informing that no wireless stack was found (FUS_STATE_IMG_NOT_FOUND).

Memory considerations

After the delete operation is done successfully, all the space used by wireless stack becomes free for usage by Cortex®-M4 user applications, or for further wireless stack install operations.

Image start address must be aligned to sector start (this is a multiple of 4-kbytes) and the image size must be a multiple of 4 bytes, otherwise FUS rejects the installation procedure.

If a new wireless stack "B" must be installed while a stack "A" is already installed, and size of B is larger than size of A, there are two possible options for the address where "B" can be loaded:

- Condition1 (back-to-back option): (both conditions C1 and C2 below must be met)
  - C1. AddressOf(B) > (FUS_ADD - (2 x SizeOf(B)) (aligned to sector size)
  - C2. AddressOf(B) < AddressOf(A) - SizeOf(B) (aligned to sector size)
- Condition2 (non back-to-back option): (only condition C3 must be met)
  - C3. AddressOf(B) < (FUS_ADD - (3 x SizeOf(B) (address must be aligned to sector size)

Where FUS_ADD is the FUS address in the flash (0x080F4000 for STM32WB5xxx and 0x08046000 for STM32WB1xxx)

In all cases, the most optimized download address is:

DownloadAddress = 0x08000000 + (SFSA x SectorSize) - SizeOf(B) - 1xSectorSize

2.3 Wireless stack start

It is possible that a wireless stack is installed but not currently running. The resulting situation is: installation is done, the wireless stack is running and then user application sends two consecutive FUS_GET_STATE, making the FUS to start again.

In that case, it is possible to launch the wireless stack execution by sending FUS_START_WS command. This command switches from the Cortex®-M0+ execution to the wireless stack, and results in at least one system reset.

The command is completed when FUS_GET_STATE returns FUS_STATE_NOT_RUNNING value. On receiving this value, no other FUS_GET_STATE must be issued, otherwise the FUS is executed again.

2.4 Anti-rollback activation

When FUS supports Anti-rollback, it is possible to activate this feature by sending a command to the FUS.

When this command is executed by FUS, it is no longer possible to deactivate it.

This feature is executed through FUS_ACTIVATE_ANTIROLLBACK command.

After sending this command it is possible to check its status by sending FUS_GET_STATE command. The FUS then returns the state FUS_STATE_IDLE.

This command is not reversible. This command does not apply to FUS, as no rollback is possible on FUS anyway.

Important: *Before activating Anti-rollback, ensure that a wireless stack is correctly installed, and that it has not been deleted. If it is activated without any wireless stack installed, the FUS registers 0xFFFFFFFF as new version, and it is not possible to install a wireless stack.*

When Anti-rollback is activated, it locks the version of wireless stack that can be installed.

It is impossible to install any wireless stack with a version lower than the current one. For example, if wireless stack V1.9.0 is installed, when Anti-rollback is activated, only wireless stacks with versions V1.9.0 or higher can be installed (it is no longer possible to install, for example, V1.8.0).

3 FUS upgrade

The FUS is capable of self-upgrade in the same way as wireless stack upgrade. Deleting FUS is not possible.

3.1 Operation instructions

In order to perform a FUS upgrade, perform the following steps:

1. Download the FUS image from www.st.com or from the STM32CubeMx repository.
2. Write the FUS image in the user flash memory at the address indicated in the FUS image directory Release_Notes.html file.
3. Ensure FUS is running (follow steps in Section 1.3 How to activate FUS).
4. Send FUS_FW_UPGRADE command through IPCC mechanism (explained in sections below).
5. Send FUS_GET_STATE till getting state equal to FUS_STATE_NOT_RUNNING (this means that the wireless stack has been installed and is now running).

During the installation process, expect multiple system resets, performed by FUS, needed for the modification of dedicated memory parameters and to make Cortex®-M0+ run the installed wireless tack. The number of system resets depends upon the configuration and the location of new and old images.

FUS identifies the image as FUS upgrade image and launches the FUS upgrade accordingly. This operation can result in a relocation of the firmware stack if it is already installed, and if the size of the new FUS is larger than the size of the current FUS. This information and any relative constraints are detailed in the FUS image release note.

3.2 Considerations on memory

The FUS upgrade requires no specific memory conditions. But if the new FUS image size is larger than existing FUS size, the upgrade can move the wireless stack lower in flash memory, to ensure enough space for FUS upgrade.

This means that:

- Less flash memory is available for the Cortex®-M4 user application.
- The wireless stack is moved from its current address to another one, defined by FUS.
- If a user code is written in the sectors close to the wireless stack start sector, there is a risk of it being erased during this operation.

The size of the FUS and the results of its upgrade are detailed in its Release_Notes.html file.

The image start address must be aligned to a sector start (a multiple of 4 Kbytes), and the image size must be a multiple of 4 bytes, otherwise FUS rejects the installation procedure.

4 User authentication

The FUS services allows the user to add a customized signature to any image (wireless stack or FUS image) provided by STMicroelectronics (encrypted and signed by STMicroelectronics).

The instruction to sign a binary with a user authentication key are provided in STM32CubeProgrammer user manual.

FUS checks on the user signature only if a user authentication key has already been installed.

The signature is a 64 bytes data buffer based on RSA ECC Prime256v1 (NIST P-256) and HASH-256. It is generated by STM32CubeProgrammer tool.

4.1 Install user authentication key

FUS allows storing a user authentication key through following steps:

1. Ensure FUS is running (follow steps in Section 1.3 How to activate FUS).
2. Send FUS_UPDATE_AUTH_KEY command through IPCC mechanism (explained in sections below)
3. Send FUS_GET_STATE till getting state equal to FUS_STATE_IDLE.
   This operation does not generate any system resets.

Once the user authentication key is installed, it is changed (unless lock user authentication key operation is done) using the same flow as above. But it cannot be removed.

Once it is installed, FUS must systematically check on the binary user signature before performing the installation or upgrade. If the signature is not present or if it is not authentic, the install or upgrade is rejected with error equal to FUS_STATE_IMG_NOT_AUTHENTIC.

4.2 Lock user authentication key

FUS allows the user authentication key to be locked. It means that this key can no longer be changed for the entire product life cycle. There is no way to undo this operation once it is performed.

To lock user authentication key:

1. Ensure FUS is running (follow steps in Section 1.3 How to activate FUS).
2. Send FUS_LOCK_AUTH_KEY command through IPCC mechanism (explained in sections below).
3. Send FUS_GET_STATE till getting state equal to FUS_STATE_IDLE.
   This operation does not generate any system resets.

Once this operation is done, the user authentication key is locked.

5 Customer key storage

The FUS allows customer keys to be stored in the dedicated FUS flash memory area and then to load the stored key to the AES1 in secure mode (the AES1 key register is only accessed by Cortex®-M0+ and data registers accessible by Cortex®-M4 user application).

5.1 Key types and structure

FUS supports the storage of 101 keys (1 master key and 100 clear/encrypted keys)

Key size can be 128 bits or 256 bits. The key size and structure is the same for all type of keys. Any stored key cannot be changed or removed.

FUS supports three key types:

- **Clear key**: a key sent to FUS, unencrypted.
- **Master key**: a key sent to FUS, unencrypted and used to decrypt other keys to be sent to FUS later. The storage of this key must be done in a trusted environment (where the key cannot be extracted on the communication path). It allows the user to share encrypted keys in untrusted environments without exposing the content. A master key cannot be written in AES1 key register. It is exclusively used for decryption and cannot be changed or removed. The Master key is written only once and is never updated afterwards. Once the master key is written, any request to write master key again is rejected with error message. Writing more than 100 keys, result in the command being rejected.
- **Encrypted key**: a key that is sent to FUS in encrypted format. It is then decrypted by FUS using the master key before using it. This key must be accompanied by an IV (initialization vector) allowing its decryption by FUS. 16-bit IV is sent in the same command packet as the key itself.

The user key encryption must be based on AES-128 GCM mode. The FUS decrypts the key without using the AES hardware.

The key type must be communicated to FUS in the command packet where the key is sent (more details in commands description).

Keys are managed through their index.

When a key is sent to FUS, FUS acknowledges its reception and responds with the key allocated index. This index is assigned by FUS and cannot be changed by user application.

To store a key, the user application must send FUS_STORE_USR_KEY to FUS (with key type and the associated IV if any) and then receive key index.

To use the stored key, the user application must:

- Configure AES1 initialization registers and IV register.
- Send FUS_LOAD_USR_KEY to FUS and wait for the response to be received which means the key has been written in AES1 key register.
- Write in AES1 data register to decrypt/encrypt data using the stored key (the key register remains protected and cannot be accessed by Cortex®-M4 user application). If more than 100 keys are written, it results in that command being rejected.

There are two additional services provided by FUS for user keys management. These two services are intended for use by the Cortex®-M4 user application in the context of a secure application and they are not exposed by bootloader or STM32CubeProgrammer.

User key lock

This service ensures a key can no longer be used by any application (cannot be loaded into AES) until the next device reset. It is possible to use this service by sending FUS_LOCK_USR_KEY command containing the index of the key to be locked (Master key index is always 0 and it cannot be locked neither loaded).

When FUS_LOCK_USR_KEY command is sent, FUS stores the state of the requested key as locked and issuing any FUS_LOAD_USR_KEY for that key index results in operation fail (0x01 returned by the command response).

User key unload

This service is used to unload the currently key loaded in AES (if FUS_LOAD_USR_KEY has been used) and prevent any further operation using the loaded key by user application.

It is possible to use this service by sending FUS_UNLOAD_USR_KEY command containing the index of the key to be unloaded (Master key index is always 0 and it cannot be loaded neither unloaded).

When FUS_UNLOAD_USR_KEY is sent, FUS writes zeros into the key registers of the AES and thus the loaded key cannot be used anymore.

6 Communication with FUS

Communication with FUS is performed through the IPCC channels and by Cortex®-M4 user application or by bootloader or by JTAG. In all cases, the communication principles are exactly the same.

Using STM32 system bootloader to communicate with FUS provides abstraction of all the low layer by directly using bootloader interfaces (USART or USB-DFU).

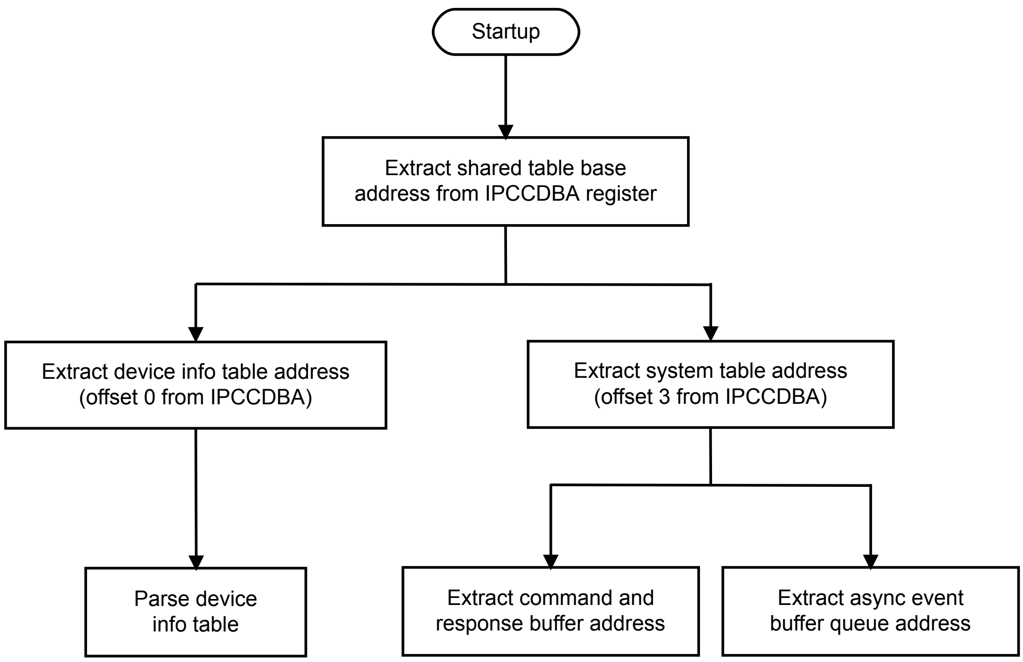
To communicate with the FUS, there are two elements to be used:

- Shared tables: used to store FUS information and to get the command/response packets.
- IPCC: used to exchange message notifications (message content is located in the shared tables).

6.1 Shared tables usage

Shared tables are an information structure located in SRAM2a public area, whose structure is explained in the following subsections. FUS uses two shared tables, namely the Device information table, and the System table. Both of them must be parsed by the Cortex®-M4 user application (or JTAG application), to correctly communicate with FUS.

Figure 4. Shared table usage process



6.1.1 Device information table

This is a 42-byte buffer used to update current status of the device. The table may be updated either by FUS code, or wireless stack code at startup, or before a programmed system reset.

Table 7. **Device information table**

Field	Size (bytes)	Values
Device info table state	4	0xA94656B9: Device info table valid Any other value: Device info table not valid
Reserved	1	Reserved
Last FUS active state	1	• 0x00: FUS idle • 0x01: Wireless stack firmware upgrade • 0x02: FUS firmware upgrade • 0x03: FUS service • 0x04: Wireless stack running • 0x05-0xFE: Not used • 0xFF: Error
Last wireless stack state	1	0x00: Not Started 0x01: Running 0x08-0xFE: Not used 0xFF: Error
Current wireless stack type	1	0x00 : None 0x01 : BLE 0x02 : Thread type1 0x03 : Thread type2 More details available in wireless stack documentation.
Safe boot version	4	Firmware version: [31:24]: Major (updated when backward compatibility is broken) [23:16]: Minor (updated when a major feature is added) [15:8]: Sub-version (updated for minor changes) [7:4]: Branch (specific build) [3:0]: Build (build version)
FUS version	4	Firmware version [31:24]: Major (updated when backward compatibility is broken) [23:16]: Minor (updated when a major feature is added) [15:8]: Sub-version (updated for minor changes) [7:4]: Branch (specific build) [3:0]: Build (build version)
FUS memory size	4	Current FUS stack memory usage: [31:24]: SRAM2b number of 1 K sectors used [23:16]: SRAM2a number of 1 K sectors used [15:8]: Reserved [14:0]: flash memory number of 4 K sectors used
Wireless stack version	4	Firmware version: [31:24]: Major (updated when backward compatibility is broken) [23:16]: Minor (updated when a major feature is added) [15:8]: Sub-version (updated for minor changes) [7:4]: Branch (specific build) [3:0]: Build (build version) When no stack present, all data is 0xFFFF FFFF
Wireless stack memory size	4	Current wireless stack memory usage: [32:24]: SRAM2b number of 1 K sectors used

Field	Size (bytes)	Values
		[23:16]: SRAM2a number of 1 K sectors used [15:8]: Reserved [14:0]: flash memory number of 4 K sectors used When no stack present, all data is 0xFFFF FFFF
Wireless FW-BLE info	4	[31:0]: Reserved for wireless stack usage When no stack present, all data is 0xFFFF FFFF
Wireless FW-thread info	4	[31:0]: Reserved for wireless stack usage When no stack present, all data is 0xFFFF FFFF
Reserved	4	0x00000000
UID64	8	STM32 device unique ID 64-bit
Device ID	2	STM32 generic device ID

6.1.2 System table

System table is an 8-byte table containing two buffer pointers, described in table below.

Table 8. System table content

Address	Size (bytes)	Content	Description
0x00	4	Address of system command/response buffer	A single buffer is used at any given time, only a command or its response must be written. Response overwrites the command. The new command overwrites any previous command response.
0x04	4	Address of system events queue buffer (address of first event)	FUS code has to parse and fill the queue when necessary. Events messages are managed as a chained list and are freed once Cortex®-M4 has read them (notification through IPCC). Parsing of the event is done through their size only. (not chained list structure),

In order to get useful information to communicate with FUS, the Cortex®-M4 code (application or bootloader) perform parsing as described in Figure 4.

6.2 IPCC usage

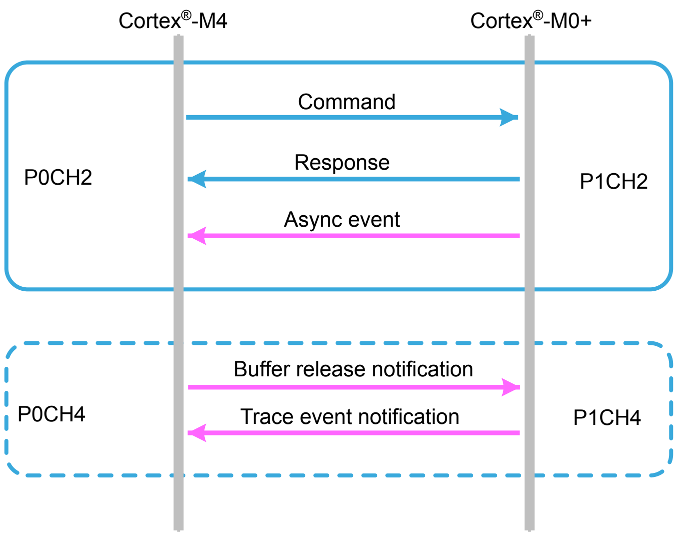
FUS uses system IPCC allocated channels: P0CH2 (on Cortex®-M4 side) and P1CH2 (on Cortex®-M0+ side). These channels offer three communication ways:

- Cmd: Command request from Cortex®-M4 to Cortex®-M0+. This route is used to send a command to Cortex®-M0+.
- Rsp: Response to command from Cortex®-M0+ to Cortex®-M4. This route is used only to answer a command requested by Cortex®-M4.
- Asynch Evt: Asynchronous event from Cortex®-M0+ to Cortex®-M4. This route is used to inform Cortex®-M4 about an asynchronous event, without requiring an answer from Cortex®-M4 on this event.

There are optional channels that may be used by FUS:

- P1CH4 may be used by FUS (Cortex®-M0+) to output trace events
- P0CH4 may be used by Cortex®-M4 in order to notify Cortex®-M0+ about buffer release events.

Figure 5. IPCC channels used by FUS

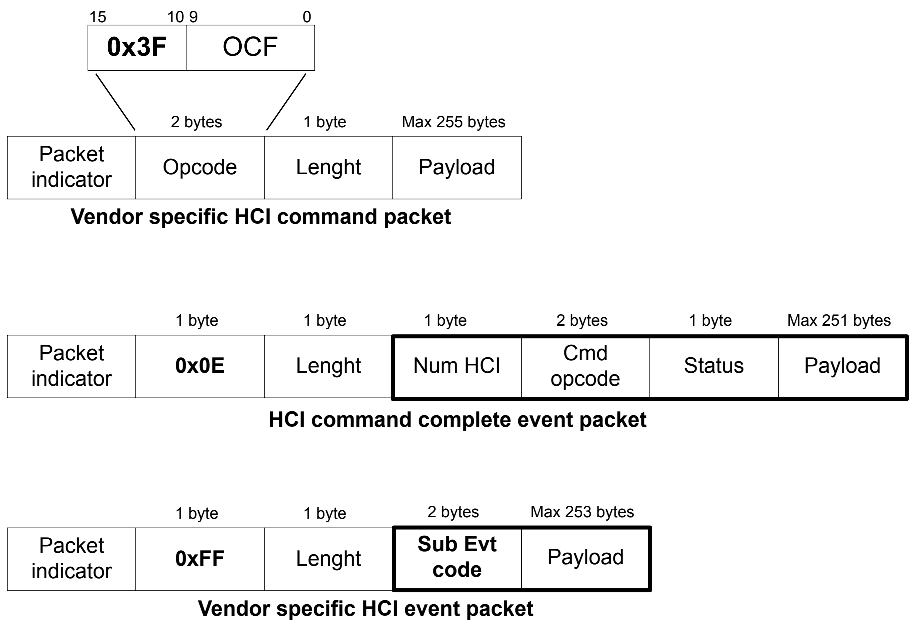


6.3 FUS commands

FUS uses the same command/response structure as wireless stacks and based on HCI model. FUS uses a subset of the HCI commands, namely:

- Vendor specific HCI command packet: used to send command from the Cortex®-M4 to the Cortex®-M0+.
- HCI command complete event packet: used to send a response from the Cortex®-M0+ to the Cortex®-M4
- Vendor specific HCI event packet: used to send asynchronous events from the Cortex®-M0+ to the Cortex®-M4.

Figure 6. FUS HCI subset



6.3.1 Packet indicators

Packet indicator is one byte and its value depends on the packet type.

Table 9. Packet indicator values

Packet type	Packet indicator value
Command packet	0x10
Response packet	0x11
Event packet	0x12

6.3.2 Event packet

Only one asynchronous event is sent by FUS. It is sent only at startup of the FUS.

The length field represents the length of SubEvtCode+Payload.

Table 10. FUS asynch event (vendor specific HCI event)

Length	SubEvtCode	Payload	Meaning
3	0x9200	Error code: • 0x00: Wireless stack running • 0x01: FUS running • 0x02: SW Error • 0x03 to 0xFF: Not used	FUS initialization phase done and the error code presented in payload byte.

6.3.3 Command packet

The table below details all commands supported by FUS and their HCI format values.

Table 11. FUS commands (vendor specific HCI command packet)

Command	Opcode	Length (bytes)	Payload
Reserved	0xFC00	N/A	N/A
FUS_GET_STATE	0xFC52	0	None
Reserved	0xFC53	N/A	N/A
FUS_FW_UPGRADE	0xFC54	0 / 4(1) / 8 (2)	None (optional 4 bytes) address of the firmware image location (optional 8 bytes) address of the firmware destination
FUS_FW_DELETE	0xFC55	0	None
FUS_UPDATE_AUTH_KEY	0xFC56	Up to 65	Byte0: authentication key size N in bytes Byte1 to ByteN-1: authentication key data
FUS_LOCK_AUTH_KEY	0xFC57	0	None
FUS_STORE_USR_KEY	0xFC58	N + 2	Byte0: key type: • 0x00:None • 0x01:Simple key • 0x02: Master key • 0x03: Encrypted key Byte1: key size N in bytes Byte2 to ByteN-1: key data (key value + IV if any)
FUS_LOAD_USR_KEY	0xFC59	1	Byte0: key index (from 0 to 124)
FUS_START_WS	0xFC5A	0	None

Command	Opcode	Length (bytes)	Payload
FUS_LOCK_USR_KEY	0xFC5D	1	One byte, index of the key to be locked
FUS_UNLOAD_USR_KEY	0xFC5E	1	One byte, index of the key to be unloaded
FUS_ACTIVATE_ANTIROLLBACK	0xFC5F	0	None
Reserved	0xFC60 to 0xFCFF	N/A	N/A

1. 4 bytes, not used in current version.

2. 8 bytes, not used in current version.

6.3.4 Response packet

For each command packet, a response packet is sent by FUS containing information detailed in table below. The NumHCI field value is always set to 0xFF.

The length field indicates the length of NumHCI+CmdOpcode+Status+Payload. So if there is no payload, the length value is four.

Table 12. FUS responses (HCI command complete packet)

<table>
<tr><th>Status</th><th>Length</th><th>Cmd Opcode value</th><th>Status value</th><th>Payload</th></tr>
<tr><td>FUS_STATE</td><td>5</td><td>0xFC52</td><td>Values in table FUS state values</td><td>Values in table FUS state error values.</td></tr>
<tr><td>FW_UPGRADE_STATE</td><td>4</td><td>0xFC54</td><td rowspan="2">• 0x00: Operation started
• 0x01: Fail
• 0x02-0xFF: Not used</td><td>None</td></tr>
<tr><td>FW_DELETE_STATE</td><td>4</td><td>0xFC55</td><td>None</td></tr>
<tr><td>UPDATE_AUTH_KEY_STATE</td><td>4</td><td>0xFC56</td><td rowspan="4">• 0x00: Operation done
• 0x01: Fail
• 0x02-0xFF: Not used</td><td>None</td></tr>
<tr><td>LOCK_AUTH_KEY_STATE</td><td>4</td><td>0xFC57</td><td>None</td></tr>
<tr><td>STORE_USR_KEY_STATE</td><td>5</td><td>0xFC58</td><td>One byte: Stored key index (from 0 to 100)</td></tr>
<tr><td>LOAD_USR_KEY_STATE</td><td>4</td><td>0xFC59</td><td>None</td></tr>
<tr><td>FUS_START_WS_START</td><td>4</td><td>0xFC5A</td><td>• 0x00: Operation started
• 0x01: Fail
• 0x02-0xFF: Not Used</td><td>None</td></tr>
<tr><td>FUS_LOCK_USR_KEY</td><td>4</td><td>0xFC5D</td><td>• 0x00: Operation done
• 0x01: Fail
• 0x02-0xFF: Not used</td><td>None</td></tr>
<tr><td>FUS_UNLOAD_USR_KEY</td><td>4</td><td>0xFC5E</td><td>• 0x00: Operation done
• 0x01: Fail
• 0x02-0xFF: Not used</td><td>None</td></tr>
<tr><td>FUS_ACTIVATE_ANTIROLLBACK</td><td>4</td><td>0xFC5F</td><td>• 0x00: Operation done
• 0x01: Fail
• 0x02-0xFF: Not used</td><td>None</td></tr>
</table>

FUS response state values are detailed in table below. Some values are represented as a range (for example 0x10 to 0x1F), which means all values from that range provide same state meaning (for example 0x12 or 0x1E both mean FUS_STATE_FW_UPGRD_ONGOING). This range of values is reserved for future extensions of the protocol.

Table 13. **FUS state values**

Value	Name	Meaning
0x00	FUS_STATE_IDLE	FUS is in idle state. Last operation done successfully and returned its state. No operation is ongoing.
0x01..0x0F	Not used	These values are reserved for future use.
0x10..0x1F	FUS_STATE_FW_UPGRD_ONGOING	The firmware upgrade operation is ongoing.
0x20..0x2F	FUS_STATE_FUS_UPGRD_ONGOING	The FUS upgrade operation is ongoing.
0x30..0x3F	FUS_STATE_SERVICE_ONGOING	A service is ongoing: Authentication key service (update/lock) or user key service (store/load).
0x40..0xFE	Not Used	These values are reserved for future use.
0xFF	FUS_STATE_ERROR	An error occurred. For more details about the error origin, refer to the response payload.

Table 14. **FUS state error values**

Value	Name	Meaning
0x00	FUS_STATE_NO_ERROR	No error occurred.
0x01	FUS_STATE_IMG_NOT_FOUND	Firmware/FUS upgrade requested but no image found. (such as image header corrupted or flash memory corrupted)
0x02	FUS_SATE_IMC_CORRUPT	Firmware/FUS upgrade requested, image found, authentic but not integer (corruption on the data)
0x03	FUS_STATE_IMG_NOT_AUTHENTIC	Firmware/FUS upgrade requested, image found, but its signature is not valid (wrong signature, wrong signature header)
0x04	FUS_SATE_NO_ENOUGH_SPACE	Firmware/FUS upgrade requested, image found and authentic, but there is no enough space to install it due to already installed image. Install the stack in a lower location then try again.
0x05	FUS_IMAGE_USRABORT	Operation aborted by user or power off occurred
0x06	FUS_IMAGE_ERSERROR	Flash Erase Error
0x07	FUS_IMAGE_WRTERROR	Flash Write Error
0x08	FUS_AUTH_TAG_ST_NOTFOUND	STMicroelectronics Authentication tag not found error in the image
0x09	FUS_AUTH_TAG_CUST_NOTFOUND	Customer Authentication tag not found in the image
0x0A	FUS_AUTH_KEY_LOCKED	The key that user tries to load is currently locked
0x11	FUS_FW_ROLLBACK_ERROR	Rollback to older version of FW detected and not allowed
0x12..0xFD	N/A	Reserved for future use.
0xFE	FUS_STATE_NOT_RUNNING	FUS is not currently running. wireless stack is running and returned this state.
0xFF	FUS_STATE_ERR_UNKOWN	Unknown error

6.4 Image footers

Each element of the image upgrade has its own footer:

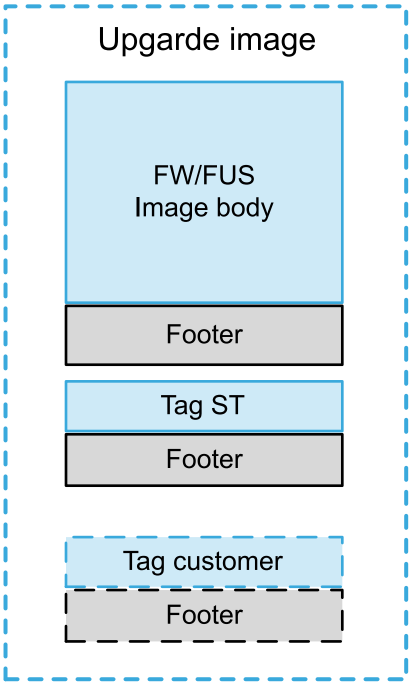
- The image body
- The STMicroelectronics signature (mandatory element)
- The customer signature (optional element)

The footers must follow on directly from the end of their relative element as a footer (for example the image body header address must be contiguous to the image body address)

The authentication tags do not have this continuity obligation, they do not need to be located next to the image. They are located anywhere in the user flash memory. FUS looks for them independently of the image location.

All images, footers addresses, and sizes must be four bytes multiples and four bytes aligned, otherwise, they are not recognized by FUS.

Figure 7. Image footers placement



Each footer contains an identification value allowing FUS to recognize it.

Figure 8. **FW/FUS upgrade image footer structure**

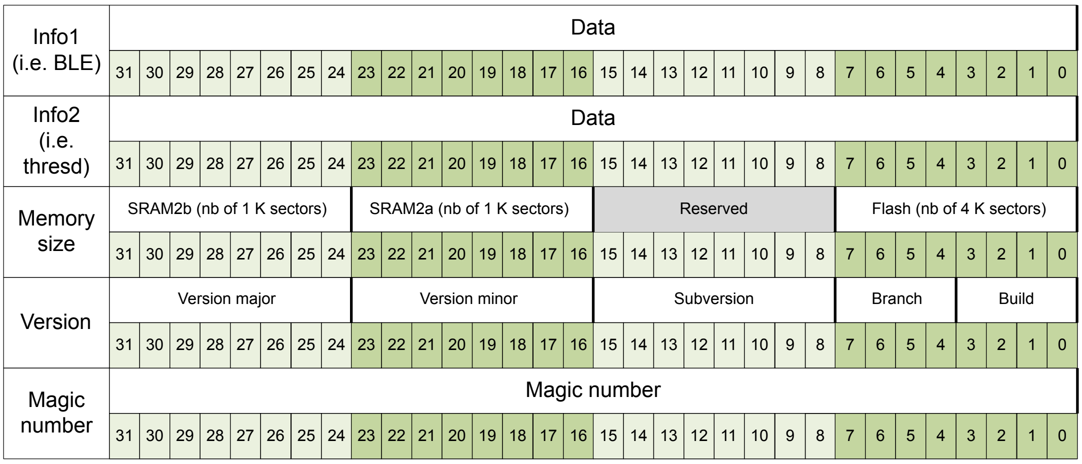


Table 15. **Parsing of image footer structure**

Field	Meaning
Info1	Specific to wireless stack / FUS image
Info2	Specific to wireless stack / FUS image
Flash memory	Image total size expressed as multiple of 4 Kbytes
SRAM2a	Image total required space in SRAM2a secure area
SRAM2b	Image total required spec in SRAM2b secure area
Build	Version build number
Branch	Version branch number
SubVersion	Version subversion number
VersionMinor	Version minor number
VersionMajor	Version major number
Magic Number	Specific value allowing to identify the nature of the image.

Note: FUS V1.2.0 version is written in the binary as 0xFFFFFFFF in order to be able to upgrade from all versions of FUS.

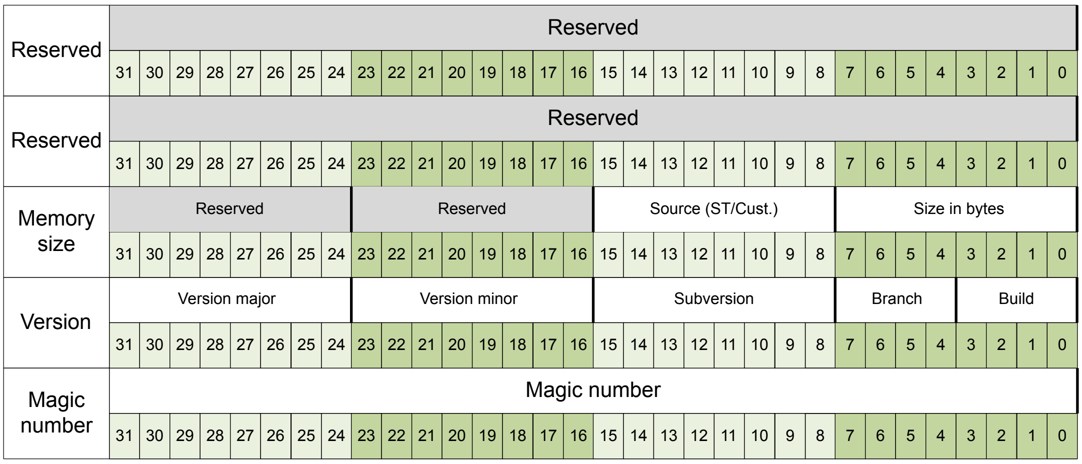


Figure 9. Signature (tag) footer structure

Table 16. Parsing of signature footer

Field	Meaning
Reserved	Not used in this version
Size	Signature total size in bytes (without footer)
Source	Signature nature: 0x00: ST signature 0x01: Customer signature
Build	Version build number
Branch	Version branch number
SubVersion	Version subversion number
VersionMinor	Version minor number
VersionMajor	Version major number
Magic Number	Specific value allowing to identify the nature of the image.

The magic number values allowing to identify the image nature are detailed in table below:

Table 17. Magic number values

Value	Nature
0x23372991	Wireless stack image
0x32279221	FUS Image
0xD3A12C5E	STMicroelectronics signature
0xE2B51D4A	Customer signature
0x42769811	Other firmware image

7 STM32 system bootloader extension for FUS

A command set extension has been added to STM32WB system bootloader in order to support FUS operation. These commands are implemented on USART and USB-DFU interfaces and follow the same rules as existing standard bootloader commands.

In order to help to understand this section, a prior reading of *STM32 microcontroller system memory boot mode* (AN2606) and *USART protocol used in the STM32 bootloader* (AN3155) and *USB DFU protocol used in the STM32 bootloader* (AN3156) documentation is required.

7.1 USART extension

Two commands have been added to bootloader USART standard protocol in order to support the FUS extension. All FUS commands are passed through these two special commands: one for writing (used for all FUS commands from host to FUS) and one for reading (used for all FUS commands from FUS to host).

Table 18. **Bootloader USART commands extension**

Command	Opcode	Usage
Special read command	0x50 (complement 0xAF)	Get data from FUS
Special write command	0x51 (complement 0xAE)	Send data to FUS

Note: For bootloader, the following commands added in FUS are not supported (neither on UART nor USB DFU)

- *FUS_LOCK_USR_KEY*
- *FUS_UNLOAD_USR_KEY*
- *FUS_ACTIVATE_ANTIROLLBACK*

Lock and Unload user key are two commands that are meant for use by Cortex®-M4 user application only.

Activate anti-rollback can be used either by implementing it in Cortex®-M4 user application code, or by using STM32CubeProgrammer features or by using STM32 open bootloader example code.

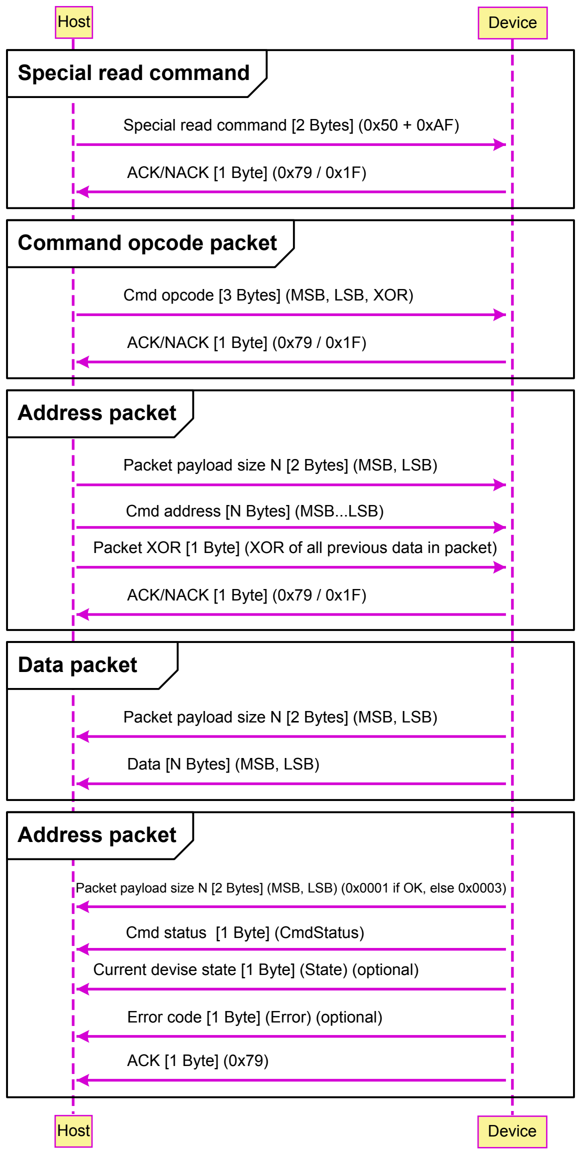
7.1.1 USART special read

Special read command is used to perform FUS command sending requesting data from device. It is divided into five separate packets:

- Special read command packet:
  - Host sends the special read command code and complement (0x50, 0xAF) and waits for ACK/NACK byte. In case of NACK, it means the command is not supported.
- Command opcode packet
  - Host sends the command packet containing:
    - FUS command opcode (2 bytes)
    - XOR of the FUS command opcode (2 bytes)
  - Device sends ACK if opcode is supported. NACK otherwise.
- Address packet:
  - Host sends address packet payload size on two bytes (MSB first).
  - Host sends address payload bytes (MSB first).
  - Host sends packet XOR value (checksum of all previous bytes in current packet, 1 byte).
  - Device sends ACK if data is correct and supported. NACK otherwise.
- Response data packet: (optional)
  - Device sends packet data payload size in bytes on 2 bytes (MSB first).
  - Device sends data payload bytes (MSB first). Some commands require not data payload.

- Response status packet:
  - Device sends packet payload size in bytes on 2 bytes (MSB first).
  - Device sends command status on one byte (status of current command requested by host).
  - Device sends current device state on 1 byte (optional, if payload size > 3).
  - Device sends current command error code (or key index) on 1 byte.
  - Device sends ACK to signal end of response packet.

Figure 10. USART special read command



7.1.2 USART special write

Special write command is used to perform FUS command sending requesting data from device. It is divided into four separate packets:

- Special write command packet: the host sends the special write command code and complement (0x51, 0xAE), and waits for ACK/NACK byte. In case of NACK, it means the command is not supported.
- Command opcode packet:
  - Host sends the command packet containing:
    - FUS command opcode (2 bytes)
    - XOR of the FUS command opcode (2 bytes)
  - Device sends ACK if opcode is supported. NACK otherwise.
- Address packet:
  - Host sends address packet payload size on two bytes (MSB first).
  - Host sends address payload bytes (MSB first).
  - Host sends packet XOR value (checksum of all previous bytes in current packet, 1 byte).
  - Device sends ACK if data is correct and supported. NACK otherwise.
- Data packet:
  - Host sends packet data payload size in bytes on 2 bytes (MSB first). This number may be zero when no data is needed for the command.
  - Host sends data payload bytes (MSB first). No data is sent if payload size is zero.
  - Host sends packet XOR value (checksum of all previous bytes in current packet, 1 byte).
  - Device sends ACK if the data is correct and supported. NACK otherwise.
- Response packet:
  - Device sends packet payload size in bytes on 2 bytes (MSB first).
  - Device sends command status on one byte (status of current command requested by host).
  - Device may send current device state on 1 byte (optional, if payload size > 1).
  - Device sends current command error code on 1 byte (optional, if payload size > 1).
  - Device sends ACK to signal end of response packet.

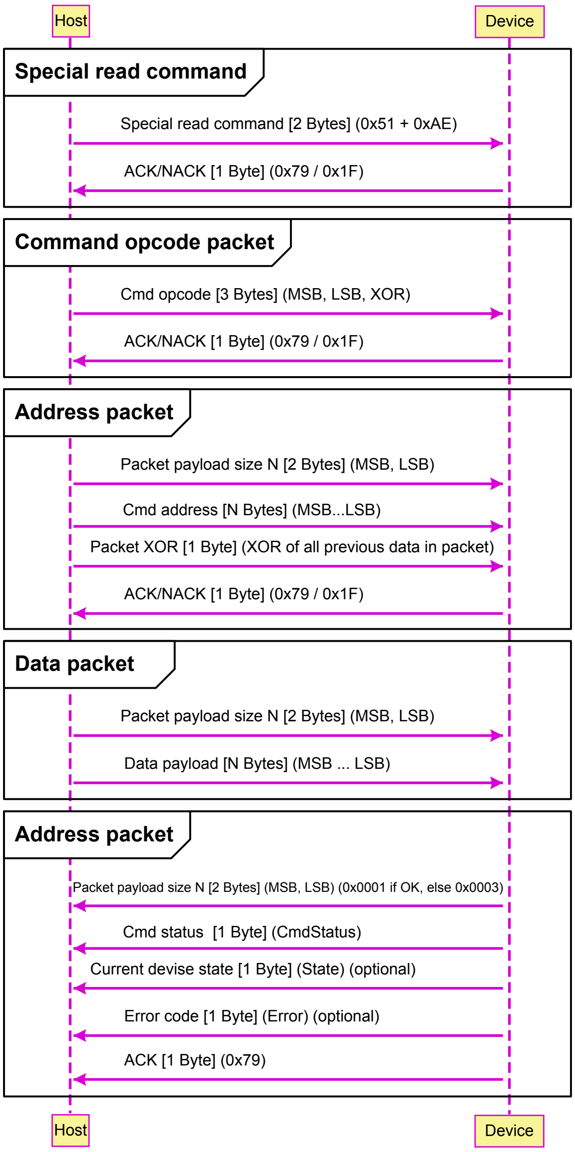


Figure 11. USART special write command

7.1.3 USART FUS command mapping

There is only one FUS command mapped on special read command.

Table 19. USART FUS command mapping on read command

Command	Opcode	Address packet	Data packet	Cmd status packet
FUS_GET_STATE	0x54	Size = 0x0000 Data = None	Size = 0x0003 Data = [0x00, FUS_STATE, ErrorCode]	Size = 0x0001 or 0x0003 Data = [0x00] if OK or [0x01, state, error] if KO

There are seven FUS commands mapped on special write command.

Table 20. USART FUS command mapping on write command

Command	Opcode	Address packet	Data packet	Cmd status packet
FUS_FW_DELETE	0x52	Size = 0x0000 Data = None	Size = 0x0000 Data = None	Size = 0x0001 or 0x0003 Data = [0x00] if OK or [0x01, state, error] if KO
FUS_FW_UPGRADE	0x53	Size = 0x0000 Data = None	Size = 0x0000 Data = None	
FUS_UPDATE_AUTH_KEY	0x56	Size = 0x0000 Data = None	Size = up to 65 Data = Key (1 byte key size + 64 bytes key data)	
FUS_LOCK_AUTH_KEY	0x57	Size = 0x0000 Data = None	Size = 0x0000 Data = None	
FUS_STORE_USR_KEY	0x58	Size = 0x0000 Data = None	Size = up to 34 Data = [KeyType (1byte), KeySize(1byte), KeyData (16/32bytes)]	Size = 0x0003 Data = [0x00, state, KeyIndex]
FUS_LOAD_USR_KEY	0x59	Size = 0x0000 Data = None	Size = 0x0001 Data = [KeyIndex]	Size = 0x0001 or 0x0003 Data = [0x00] if OK or [0x01, state, error] if KO
FUS_START_WS	0x5A	Size = 0x0000 Data = None	Size = 0x0000 Data = None	

7.2 USB-DFU extension

FUS commands are processed over bootloader USB-DFU standard download and upload commands.

7.2.1 USB-DFU download FUS extension

Bootloader USB-DFU download FUS extension is managed in the same way as SET_ADDRESS_POINTER and ERASE standard commands: Value = 0 and following bytes are command data MSB first.

Exception is made for FUS_STORE_USR_KEY which is split over two steps:

1. Download command, only allows to send the key data (up to 34 bytes)
2. Upload command, must be done after download step and allows to get the key index (1 byte)

Table 21. USB-DFU download extension

Command	Opcode	Data
FUS_FW_DELETE	0x52	None
FUS_FW_UPGRADE	0x53	None
FUS_UPDATE_AUTH_KEY	0x56	Key Buffer = [KeySize (1byte), KeyData (64bytes MSB first)]
FUS_LOCK_AUTH_KEY	0x57	None
FUS_STORE_USR_KEY	0x58	Key Buffer = [KeyType (1byte), KeySize(1byte), KeyData (16/32bytes)]
FUS_LOAD_USR_KEY	0x59	Key Index (1byte)
FUS_START_WS	0x5A	None

7.2.2 USB-DFU upload FUS extension

Bootloader USB-DFU upload FUS extension is managed in same ways as regular upload command for reading physical address (wBlockNum > 1). But in this case, a virtual memory address mask is used: 0xFFFF0000. So the FUS read command is managed through a read to virtual address 0xFFFF00YY where YY is the FUS command opcode.

Upload command allows to perform the second step of FUS_STORE_USR_KEY which is getting the key index.

Table 22. USB-DFU upload extension

Command	Address	Returned data
FUS_GET_STATE	0xFFFF0054	State buffer = [FUS state (1byte), FUS error code (1byte)]
FUS_STORE_USR_KEY	0xFFFF0058	Key index (1byte)

8 FAQ and troubleshooting

Table 23. Frequently ask and answer

Question/troubleshooting	Answer
When I receive a virgin STM32WB device from ST, what does it contain exactly?	All STM32WB devices delivered by STMicroelectronics contain by default the FUS and the bootloader. They do not contain the pre-installed wireless stack.
I cannot read the FUS version	Accessing device information table is possible when following conditions are met : 1. Device info table address is written in location pointed by the IPCCDBA option byte. 2. Cortex®-M0+ is enabled 3. FUS is running on Cortex®-M0+ (and not wireless stack) (If the wireless stack is running, it is possible to force FUS to run by sending 2 FUS_GET_STATE commands). So when accessing device via SWD, it is normal to not find device info table valid because it has not yet been written or Cortex®-M0+ has not been enabled yet. That's why it is more convenient to read device info table when bootloader is running because it performs the actions (1) and (2) above. *Note: It is possible to connect through SWD and disable the hardware reset option (hot plug) and keep boot on bootloader which allows user to read the device info table.*
I want to upgrade FUS image and I already have a wireless stack installed. Do I need to delete the wireless stack prior to upgrading FUS?	It is advised to delete the wireless stack before performing the FUS upgrade in general and especially when upgrading from FUS V0.5.3. If the existing FUS version is higher than V0.5.3, then, it is not mandatory to perform the wireless stack deletion.
How do I know quickly if my device is running FUS or wireless stack?	There are multiple ways to check it: • Read the Option Bytes and check the value of SBRV. If FUS is running it is 0x3D000 (or 0x3D800 if FUS V0.5.3 is running) • Read the device information table @0x20030030, if it is different for the FUS version, then the wireless stack is running or Cortex®-CM0+ is not enabled. • Send FUS_GET_STATE command, if FUS_STATE_NOT_RUNNING is received, then the wireless stack is running or Cortex®-CM0+ is not enabled.
What is IPCCDBA Option Byte used for?	IPCCDBA is used to change the offset where to read/write the device information table.
After an upgrade operation, I cannot access flash memory anymore and cannot communicate with FUS.	First check if SFSA=0x00. If it is the case, then it means safeboot has been triggered. Safeboot is triggered when an Option Bytes corruption occurs. This may occur during a FUS upgrade operation or during any user application operation dealing with Option Bytes. When safeboot is triggered it locks the device by setting SFSA=0x00 (all flash memory secure) and so no user application/debugger can access the user flash memory anymore. This operation is not reversible. Starting from FUS V1.1.0, the safeboot is modified to perform a factory reset instead of locking the device.
Is it possible to downgrade FUS version (for example when current FUS running version is V1.0.2, is it possible to install FUS V1.0.1?)	FUS downgrade is not possible in any combination. It can be installed only forward. In case of downgrade tentative, FUS simply rejects the upgrade and returns an error message.

<table>
<tr><th>Question/troubleshooting</th><th>Answer</th></tr>
<tr><td>What is a typical STM32CubeProgrammer command to perform an upgrade using FUS?</td><td>1. First check that FUS is running by sending FUS_GET_STATE commands until receiving FUS_STATE_IDLE state response:
– STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate
– STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate
– STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate
Sending 3 times FUS_GET_STATE command ensures that FUS is running and idle in most cases.
2. Delete the existing wireless stack and install the new one (case of wireless stack upgrade):
– STM32_Programmer_CLI.exe -c port=usb1 -fwupgrade stm32wb5x_BLE_Stack_fw.bin 0x080CB000 firstinstall=0
– STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate
– STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate
– ... (keep sending -fusgetstate till received state is FUS_STATE_NOT_RUNNING)
Setting "firstinstall=0" ensures that the previous stack is deleted before the new one is installed.
Even if there is no previously installed stack, setting "firstinstall=0" would not cause any problem.
Alternately proceed to FUS image installation (case of FUS upgrade):
• STM32_Programmer_CLI.exe -c port=usb1 -fwupgrade stm32wb5x_FUS_fw.bin 0x080EC000 firstinstall=0
• STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate
• STM32_Programmer_CLI.exe -c port=usb1 -fusgetstate
• ... (keep sending -fusgetstate till received state is FUS_STATE_IDLE)
"firstinstall=0" means existing wireless stack is deleted prior to upgrading FUS.
It is possible to use "firstinstall=1" if upgrading from FUS version different from FUS V0.5.3.</td></tr>
<tr><td>What is safeboot and how can it be used?</td><td>Safeboot is an independent part of the FUS that manages specifically one case: option bytes corruption.
When option bytes are corrupted, the STM32WB hardware forces the boot to safeboot whatever the running firmware.
The safeboot then either:
• Locks the device in full secure mode (on FUS versions lower than V1.1.0) which means all the device flash memory cannot be accessed and this operation is not reversible (there is no mean to cancel it and the device cannot be used anymore).
• or performs a factory reset (on FUS versions V1.1.0 and higher) which means the wireless stack is removed if any and the Cortex®-M4 code is erased and boot is reset to FUS (virgin part state). This operation is also not reversible. In order to activate the Safeboot, the user must activate Cortex®-M0+ by writing the value 0x00008000 at the address 0x5800040C using the SWD interface.</td></tr>
<tr><td>Does FUS erase the shadow of encrypted firmware after installation?</td><td>Yes, FUS does erase the shadow remaining sectors of the encrypted firmware after it has been installed and moved to upper address.</td></tr>
<tr><td>Is there a restriction on firmware image sizes that can be installed?
Is it necessary to delete the installed firmware image before installing a new image?</td><td>When using FUS version older than V1.2.0:
• If a wireless firmware image B is installed while another wireless firmware image A is already installed/running. if B size is larger than A size, and if B is loaded at an address too close to A (no enough free space between start address of A and end address of B, as explained in Section 2 Wireless stack image operations) then the device might be blocked with SBRV value pointing to the firmware image A (which is then corrupted) instead of pointing to firmware B, and the recovery might not be possible in that case.
• For this, it is advised to delete the firmware image A before installing the firmware image B (in case of FOTA, this might be non feasible) or to make sure enough space is available before performing the installation. This known limitation is fixed in FUS V1.2.0.</td></tr>
</table>

Question/troubleshooting	Answer
After upgrade of the FUS V1.2.0, I get an error message on STM32CubeProgrammer (or other programming interface) with error code: "FUS_UFB_CORRUPT. What does it means and what to do in this case?	When upgrading from some older versions of FUS to FUS V1.2.0, it is normal to have the error message FUS_UFB_Corrupt. It means that UFB area (used for storing FUS configuration information) has been erased and needs to be configured to reset values. FUS then write the reset values and provoke a system reset and then the error is cleared. FUS returns FUS_IDLE state with no errors. No operation is required from user side.

Revision history

Table 24. Document revision history

Date	Version	Changes
21-Mar-2019	1	Initial release.
17-Jun-2019	2	Added Section 1.2 FUS versioning and identification Updated: • Section 2 Wireless stack image operations, Section 5.1 Key types and structure, Section 5.1 Key types and structure • Table 20. USART FUS command mapping on write command, Table 22. USB-DFU upload extension
10-Jul-2019	3	Updated Table 7. Device information table
31-Mar-2020	4	Updated: • Section 1.1 Firmware upgrade services definition, Section 2 Wireless stack image operations, Section 2.1 Wireless stack install and upgrade, Section 2.2 Wireless stack delete,Section 5.1 Key types and structure, Section 7.1 USART extension • Table 1. FUS versions, Table 11. FUS commands (vendor specific HCI command packet), Table 12. FUS responses (HCI command complete packet), Table 14. FUS state error values Added: Section 2.4 Anti-rollback activation and Section 8 FAQ and troubleshooting
06-May-2021	5	Updated: • Section 1.2 FUS versioning and identification • Table 1. FUS versions • Section 1.3 How to activate FUS • Figure 1. Flash memory mapping • Section 1.5 FUS resources usage • Table 5. FUS resources usage • Section 2.4 Anti-rollback activation • Section 8 FAQ and troubleshooting Added: • Table 1. FUS versions • Table 2. FUS Versions Compatibility
22-Oct-2021	6	Updated: • Section 1.5 FUS resources usage • Figure 8. FW/FUS upgrade image footer structure • Figure 9. Signature (tag) footer structure • Section 8 FAQ and troubleshooting with new limitation
2-Aug-2022	7	Added Section 1.2.1 Known limitations Updated: • Section 2.1 Wireless stack install and upgrade • Section 2.2 Wireless stack delete • Section 8 FAQ and troubleshooting
16-Jan-2023	8	Updated Table 5. FUS resources usage. Updated Section 2.2 Wireless stack delete. Minor text edits across the whole document.

Contents

List of tables

List of figures