# How to use FDCAN bootloader protocol on STM32 MCUs

## Introduction

This application note describes the FDCAN protocol used in the STM32 microcontroller bootloader, providing details on each supported command.

This document applies to STM32 products embedding any bootloader version, as specified in the application note *STM32 microcontroller system memory boot mode* (AN2606), available from *www.st.com*. These products are listed in Table 1.

For more information about the FDCAN hardware resources and requirements for their device bootloader, users are advised to refer to the application note AN2606.

**Table 1. Applicable products**

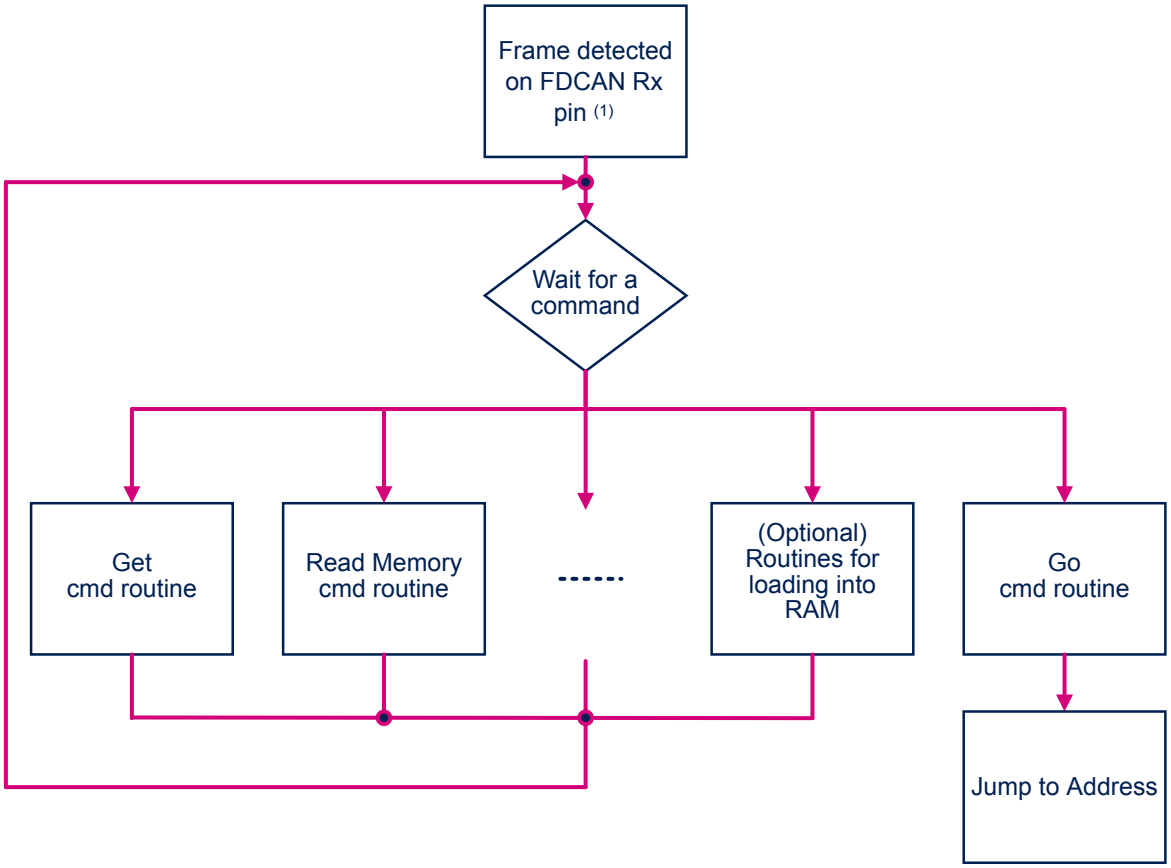
| Type | Product series |
|---|---|
| Microcontrollers | STM32G0 series<br>STM32G4 series<br>STM32H5 series<br>STM32H7 series<br>STM32L5 series<br>STM32U3 series<br>STM32U5 series |

# 1 Bootloader code sequence

The 32-bit microcontrollers listed in Table 1. Applicable products are based on the Arm® Cortex® processor. They are referred to as STM32 in this document.

Figure 1 presents the global sequence for the STM32 bootloader with FDCAN.

**Figure 1. Bootloader for STM32 with FDCAN**




(1) Depending on the FDCAN protocol version, the frame sent to the FDCAN to start communication is different
* When protocol version is <= 2.1, any frame can be sent
* When protocol version is > 2.1, only this frame is accepted "ID = 111h, data length = 1 and data = 5Ah"

Once the system memory boot mode is entered and the STM32 device is configured (refer to the application note AN2606 for additional details), the bootloader code waits for a frame on the FDCANx_Rx pin.

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

# 2 FDCAN settings

In this application, the FDCAN settings are:

- Frame format: FD mode with Bit Rate Switching
- Mode: Normal mode
- AutoRetransmission: Enabled
- TransmitPause: Disabled
- Standard identifier (not extended)
- Time quantum = Nominal Prescaler × (1 / fdcan_ker_ck)
  - fdcan_ker_ck = 20 MHz
  - Nominal Prescaler = 0x1
- Synchronization Jump Width = 0x10
- Nominal Time Segment1 = 0x3F
- Nominal Time Segment2 = 0x10
- Data Prescaler = 0x1
- Data Sync Jump Width = 0x4
- Data Time Segment1 = 0xF
- Data Time Segment2 = 0x4
- Standard Filters Number = 1
- Extended Filters Number = 0

Filter settings are:

- ID Type = 0
- Filter Index = 0
- Filter Type = 2 (classic filter)
- Filter Config = FDCAN_FILTER_TO_RXFIFO0
- Filter ID1 = 0x111
- Filter ID2 = 0x7FF

For protocol version ≤ 2.1 the filters are configured but global filters rejection is not configured. So, any message ID sent to bootloader is not filtered and the message ID is taken as a command opcode.

For protocol versions > 2.1, the filter configuration is changed in order to filter Messages IDs between 00h and FFh:

- Filter Type = FDCAN_FILTER_RANGE;
- Filter Config = FDCAN_FILTER_TO_RXFIFO0;
- Filter ID1 = 0x00
- Filter ID2 = 0xFF

Transmit settings (from the STM32 to the host) are:

- Identifier = 0x111
- Id Type = 0
- Frame Type = FDCAN_DATA_FRAME
- Data Length = FDCAN_DLC_BYTES_64
- ErrorStateIndicator = FDCAN_ESI_ACTIVE
- BitRateSwitch = FDCAN_BRS_ON
- FDFormat = FDCAN_FD_CAN
- TxEventFifoControl = FDCAN_NO_TX_EVENTS
- MessageMarker = 0

*Note:*

1. *The CAN bootloader firmware supports only one node at a time. This means that it does not support CAN Network Management.*
2. *Depending on the FDCAN protocol version, the frame acceptance is different:*
   - *When protocol version is ≤ 2.1, filters are not enabled, so any MessageID and FilterID1 are accepted.*
   - *When protocol version is > 2.1, MessageID and FilterID1 must match exactly.*
3. *In the bootloader, only the 8 LSB bits of the messaged ID are taken into account, so, any value above FFh is ignored.*

# 3 Bootloader command set

Table 2 lists the supported commands, each of them being described in the corresponding subsection.

**Table 2. FDCAN bootloader commands**

| Command | Command code | Subsection | Description |
|---|---|---|---|
| Get(1) | 0x00 | Section 3.1 | Gets the version and allowed commands supported by the current version of the protocol. |
| Get Version(1) | 0x01 | Section 3.2 | Gets the protocol version and the readout protection status of the flash memory. |
| Get ID(1) | 0x02 | Section 3.3 | Gets the chip ID. |
| Read Memory(2) | 0x11 | Section 3.4 | Reads up to 256 bytes of memory starting from an address specified by the application. |
| Go(2) | 0x21 | Section 3.5 | Jumps to user application code located in the internal flash memory or SRAM. |
| Write Memory(2) | 0x31 | Section 3.6 | Writes up to 256 bytes of RAM or flash memory starting from an address specified by the application. |
| Erase Memory(2) | 0x44 | Section 3.7 | Erases from one to all the flash memory sectors. |
| Special | 0x50 | Section 3.12 | Generic command that allows the addition of new features depending on the product constraints without adding a new command for every feature needed. |
| Extended Special | 0x51 | Section 3.13 | Generic command that allows the user to send more data compared to the Special command. |
| Write Protect | 0x63 | Section 3.8 | Enables the write protection of some sectors. |
| Write Unprotect | 0x73 | Section 3.9 | Disables the write protection of all flash memory sectors. |
| Readout Protect(1) | 0x82 | Section 3.10 | Enables readout protection. |
| Readout Unprotect(1) | 0x92 | Section 3.11 | Disables readout protection. |

1. *Protection: When the protection is active, only this limited subset of commands is available. All other commands are NACK-ed and have no effect on the device. Protection depends on product series. Protection is active means*
   - *For the STM32H5 series: Trust zone (TZEN) = 0 and Product state > Provisioning, and HiDeProtection Level (HDPL) = 3*
   - *For the other products listed in Table 1. Applicable products: read protection set.*
2. *Refer to the STM32 product datasheet and application note AN2606 to know about the valid memory spaces for this command.*

### Communication safety

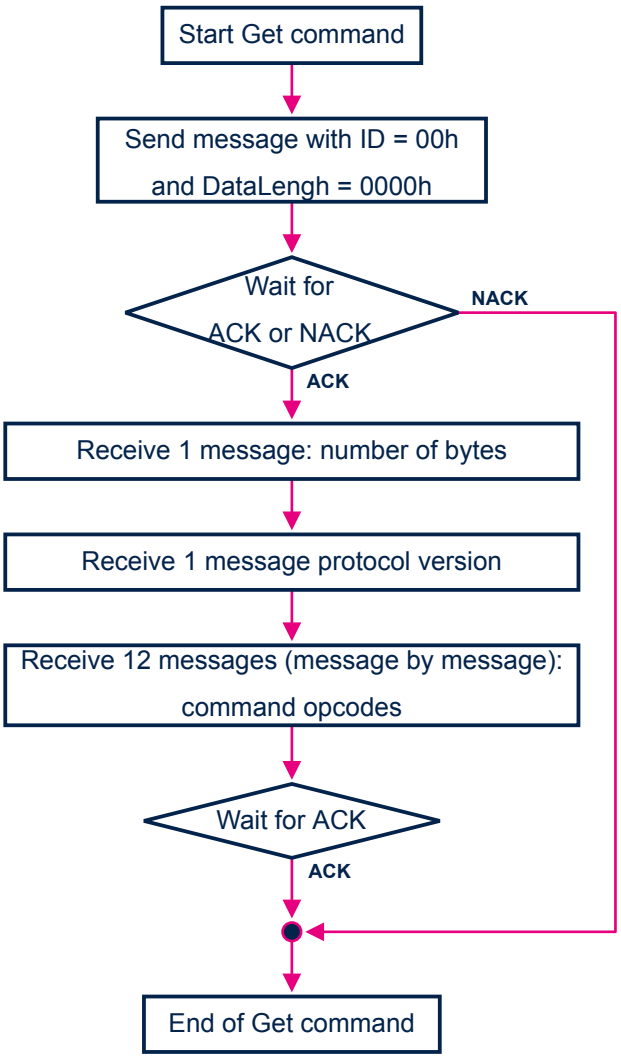
Each packet is either accepted (ACK answer) or discarded (NACK answer):

- ACK message = 0x79
- NACK message = 0x1F

## 3.1 Get command

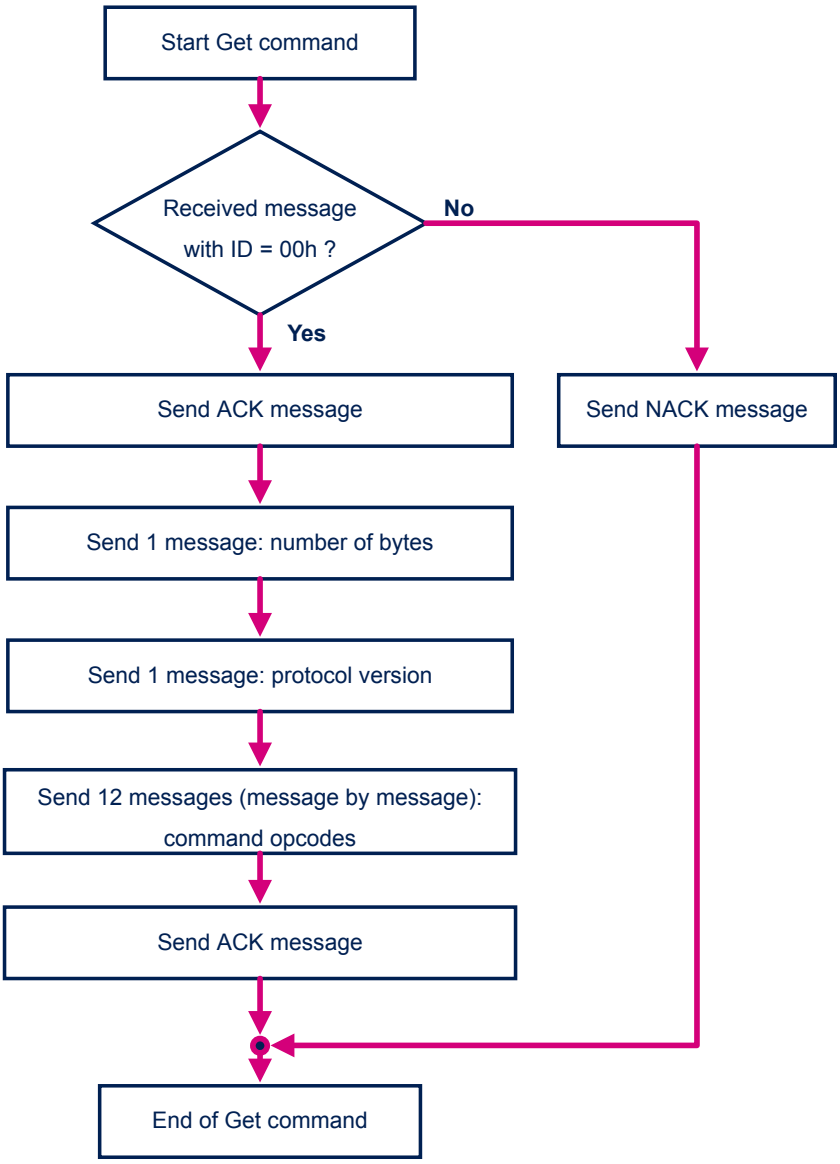
The Get command allows the host to get the version of the protocol and the supported commands. When the bootloader receives the Get command, it transmits the protocol version and the supported command codes to the host.

**Figure 2. Get command (host side)**



Based on the host command, the device answers as shown in Figure 3.

**Figure 3. Get command (device side)**




All data sent from the device on this command are sent using DataLength = 0000h (one byte).

The data are received as per the following order:

1. Number of commands (11)
2. FDCAN protocol version (0x11)
3. Get command opcode (0x00)
4. Get Version command opcode (0x01)
5. Get ID command opcode (0x02)
6. Read Memory command opcode (0x11)
7. Go command opcode (0x21)
8. Write Memory command opcode (0x31)
9. Erase Memory command opcode (0x44)
10. Write Protect command opcode (0x63)
11. Write Unprotect command opcode (0x73)
12. Readout Protect command opcode (0x82)
13. Readout Unprotect command opcode (0x92)

Some bootloader commands depend on the availability of hardware features on the STM32 product.

From the FDCAN BL version V2.0, the number of commands is not fixed anymore. It can change from one product to another.

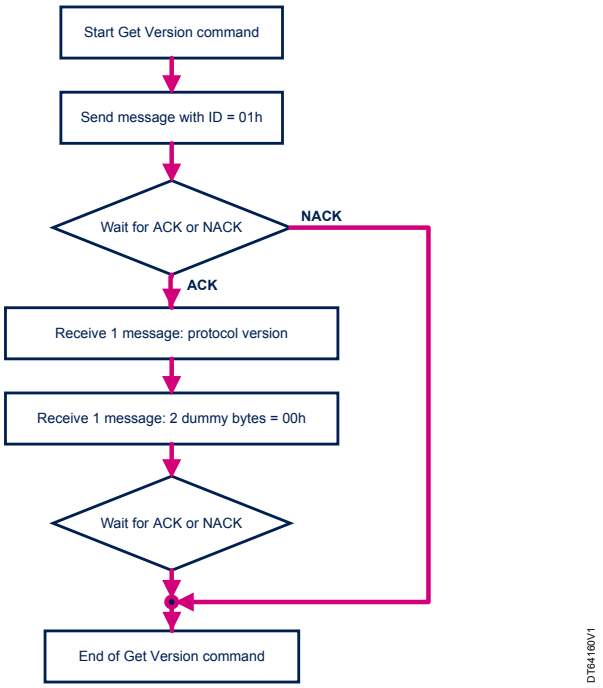
For instance, in the STM32H5 series, the RDP hardware feature is not available. The getcommand is as follows:

1. Number of commands (10)
2. FDCAN protocol version (0x20)
3. Get command opcode (0x00)
4. Get Version command opcode (0x01)
5. Get ID command opcode (0x02)
6. Read Memory command opcode (0x11)
7. Go command opcode (0x21)
8. Write Memory command opcode (0x31)
9. Erase Memory command opcode (0x44)
10. Special command opcode(0x50)
11. Write Protect command opcode (0x63)
12. Write Unprotect command opcode (0x73)

## 3.2 Get Version command

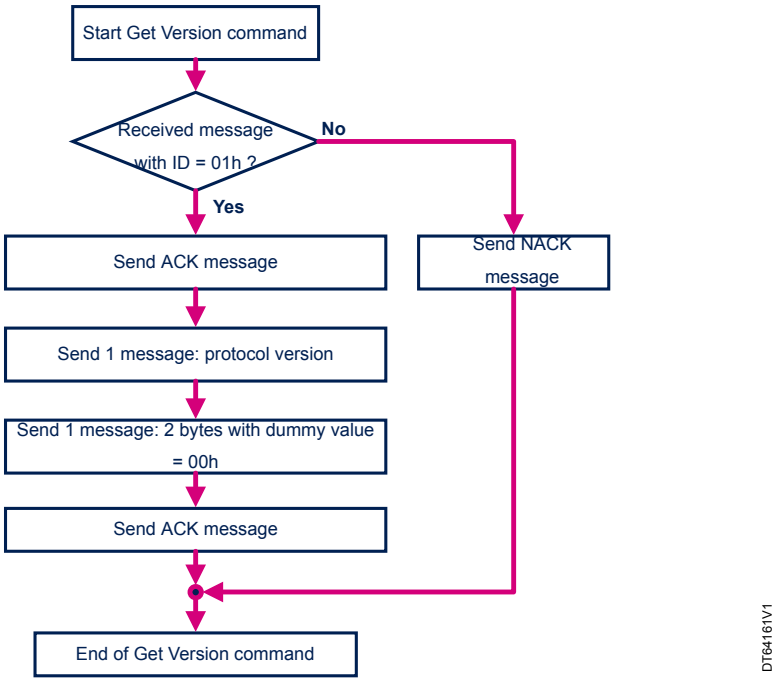
The Get Version command is used to get the protocol version. When the bootloader receives the command, it transmits the information to the host as described in Figure 4 (version and two dummy bytes having the value 0x00).

**Figure 4. Get Version command (host side)**




The host receives first the protocol version (one byte), and then two dummy bytes equal each to 00h.

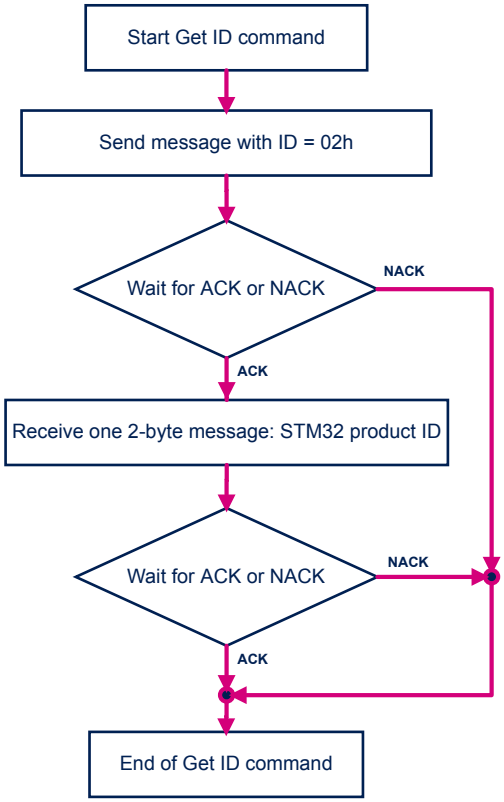
**Figure 5. Get Version command (device side)**



## 3.3 Get ID command

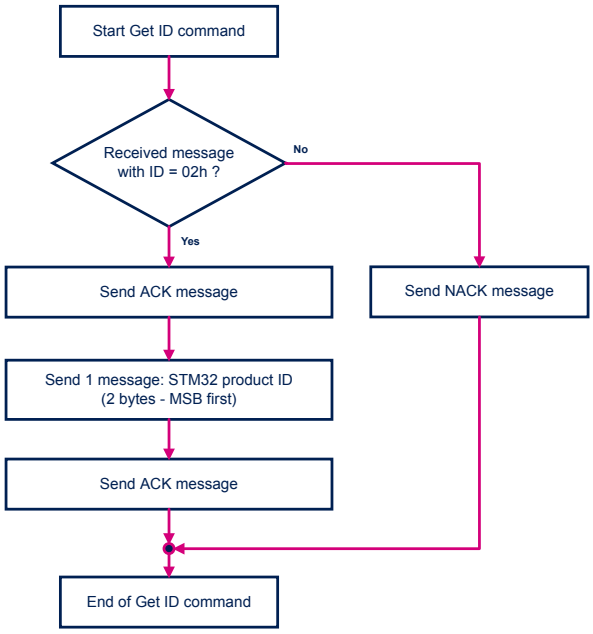
The Get ID command is used to get the version of the STM32 product ID (identification). When the bootloader receives the command, it transmits the product ID to the host.

**Figure 6. Get ID command (host side)**




The host receives the STM32 product ID (two bytes) sent by the device as shown in Figure 7. The LSB (byte) is sent first.

**Figure 7. Get ID command (device side)**



## 3.4 Read Memory command

The Read Memory command is used to read data from any valid memory address: RAM, flash memory, and information block (system memory or option byte areas).

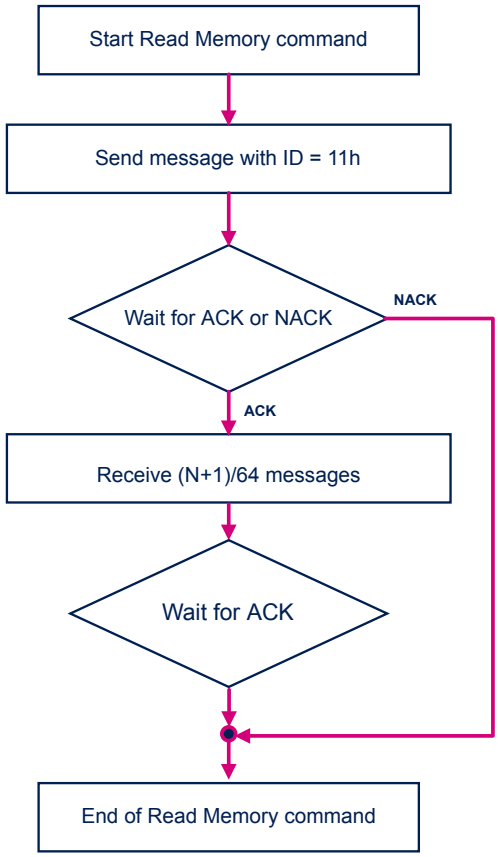
Running the Read Memory command is only possible when the read protection is not set and the address to read from is valid. For this reason, the bootloader software performs the following checks in sequence:

1. Is the ID of the command correct or not?
2. Is the protection disabled or not?
3. Is the address to read from valid or not?

If all checks pass, the Read Memory command proceeds with reading the data, otherwise NACK is sent to the host.

*Note: The bootloader sends the data as 64-byte messages. The host takes only the number of data needed.*

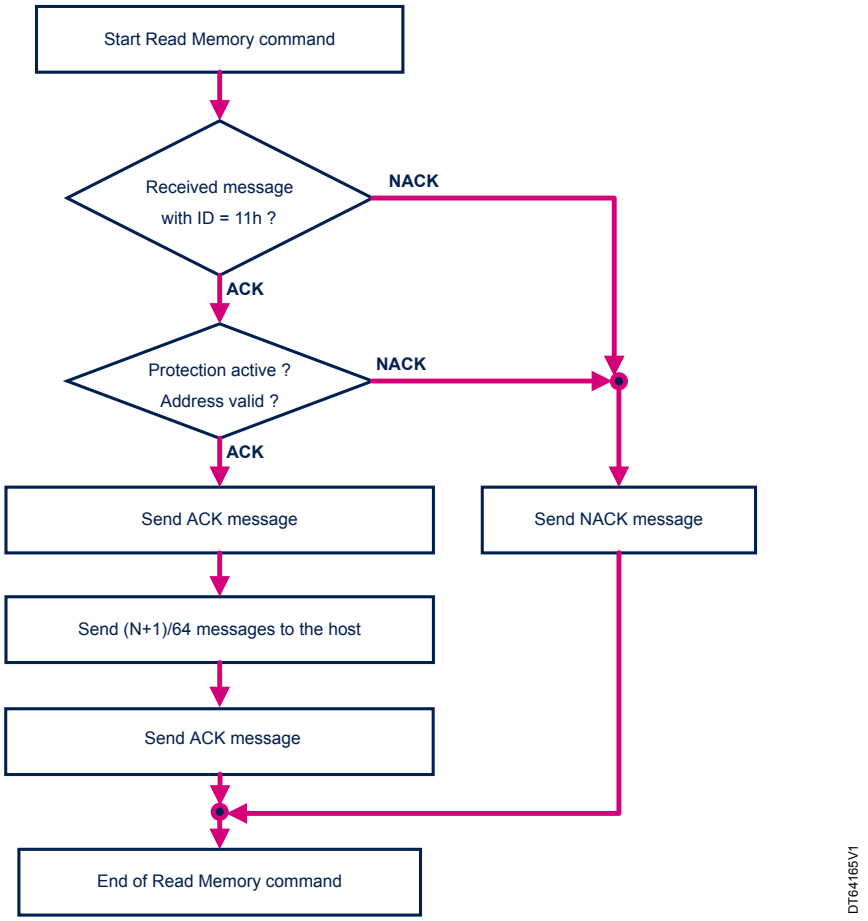
**Figure 8. Read Memory command (host side)**



Together with the message ID for the Read Memory command (11h), the address, and number of bytes are sent to the device. The message content is as follows:

1. ID = 0x11, DLC = 0x05
2. data[0] = 0xXX (MSB of the address)
3. data[1] = 0xYY
4. data[2] = 0xZZ
5. data[3] = 0xTT (LSB of the address)
6. data[4] = N (number of bytes to be read minus one; 0 < N ≤ 255)

**Figure 9. Read Memory command (device side)**




The data are always sent to the host by multiples of 64 bytes. The host must filter the needed data based on the number of data requested.

## 3.5 Go command

The Go command is used to execute downloaded code or any other code by branching to an address specified by the application. When the bootloader receives the Go command, it starts only if the message contains valid information and passes the following checks:

- Is the ID of the command correct?
- Is the protection disabled?
- Is the address to jump to valid?

If the message content is correct, the Go command transmits an ACK message. Otherwise, it transmits a NACK message.

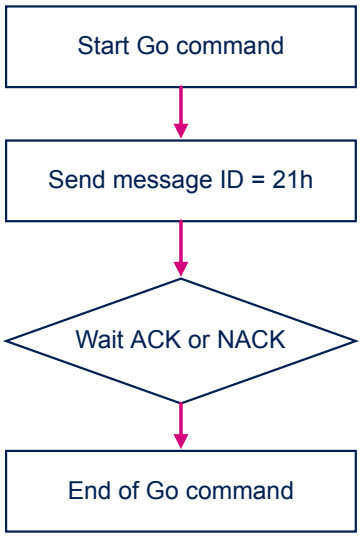
After sending an ACK message to the application, the bootloader firmware:

- Resets the registers of the peripherals used by the bootloader to their default values
- Initializes the main stack pointer of the user application
- Jumps to the memory location programmed in the received *"address + 4"*, which is the address of the application reset handler.
  For example, if the received address is 0x0800 0000, the bootloader jumps to the memory location programmed at address 0x0800 0004. The host must send the base address where the application to jump to is programmed.

*Note:*

1. *The jump to the application works only if the user application sets the vector table correctly to point to the application address.*
2. *The valid addresses for the Go command are in RAM or flash memory. All other addresses are considered not valid and are NACK-ed by the device.*
3. *Not all addresses in the RAM are considered valid. The application to jump must consider an offset to avoid overlapping with the first RAM memory used by the bootloader firmware.*

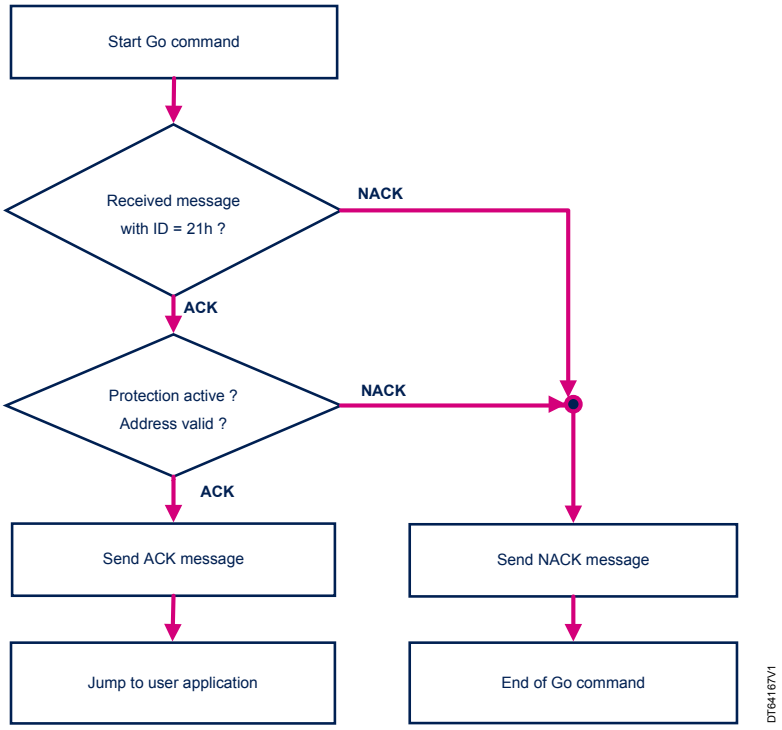
**Figure 10. Go command (host side)**



The message content sent to the device, including the Go command ID (21h), is as follows:

1. ID = 0x21, DLC = 0x04
2. data[0] = 0xXX (MSB of the address)
3. data[1] = 0xYY
4. data[2] = 0xZZ
5. data[3] = 0xTT (LSB of the address)

**Figure 11. Go command (device side)**




## 3.6 Write Memory command

The Write Memory command is used to write data to any valid memory address of the RAM, flash memory, or option byte area. When the bootloader receives the Write Memory command, it starts only if the message contains valid information and passes the following checks:

- Is the ID of the command correct?
- Is the protection disabled?
- Is the address to write to valid?

If the message content is correct, the Write Memory command transmits an ACK message and continues the job. Otherwise, it transmits a NACK message and exits the command.

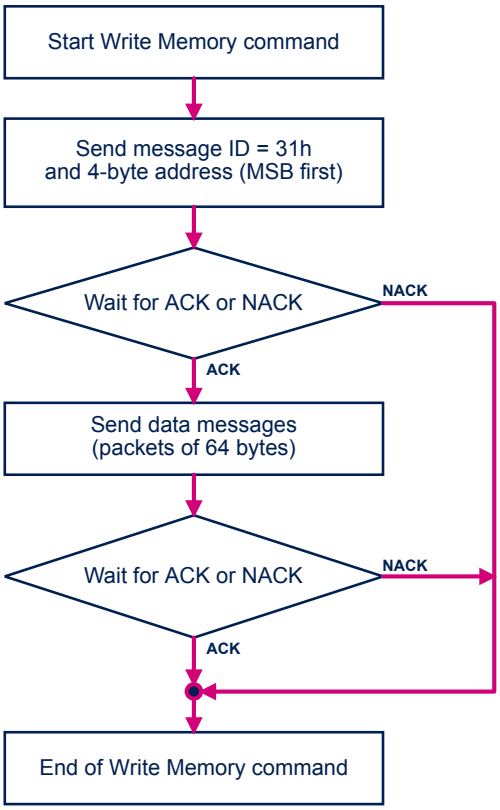
For the option byte area, the start address must be the base address of the option byte area to avoid writing inopportunely in this area.

When the address is valid, the bootloader:

- Receives the user data (N bytes). This means that the device receives N/64 messages, each message being composed of 64 data bytes.
- Programs the user data into the memory starting from the received address
- Transmits the ACK message at the end of the command if the write operation was successful. Otherwise, it transmits a NACK message to the application and aborts the command.

*Note:*
1. *The maximum length of the block to be written for the STM32 is 256 bytes.*
2. *No error is returned when performing write operations on write protected sectors.*

Figure 12. **Write Memory command (host side)**

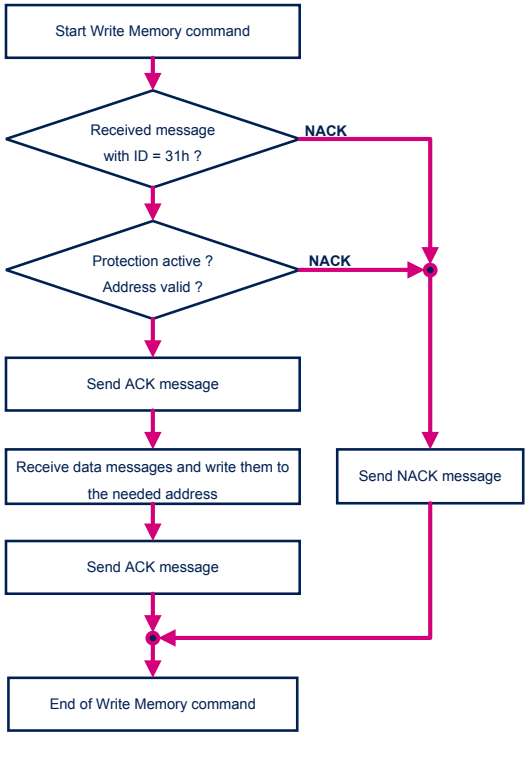


Together with the message ID for the Write Memory command (31h), the address, and number of bytes are sent to the device. The message content is as follows:

1. ID = 0x31, DLC = 0x05
2. data[0] = 0xXX (MSB of the address)
3. data[1] = 0xYY
4. data[2] = 0xZZ
5. data[3] = 0xTT (LSB of the address)
6. data[4] = N (number of bytes to be written minus one; 0 < N ≤ 255)

The host sends N / 64 messages.

**Figure 13. Write Memory command (device side)**




## 3.7 Erase Memory command

The Erase Memory command allows the host to erase flash memory pages. When the bootloader receives the Erase Memory command and the protection is disabled, it transmits the ACK message to the host.

After the transmission of the ACK message, the bootloader checks the first two bytes of data received in an MSB format and constructs a value called *PageNumber*, which controls the execution of the Erase Memory command as follows:

- *PageNumber* = 0xFFFF: mass erase is requested
- *PageNumber* = 0xFFFE: Bank1 erase is requested (if supported)
- *PageNumber* = 0xFFFD: Bank2 erase is requested (if supported)
- Otherwise, *PageNumber* represents the number of pages to erase

If the *PageNumber* value indicates mass or bank erasing, an ACK message is sent upon completion. Otherwise, the bootloader starts erasing the memory pages as defined by the host, and then sends an ACK message when all the requested pages are erased.

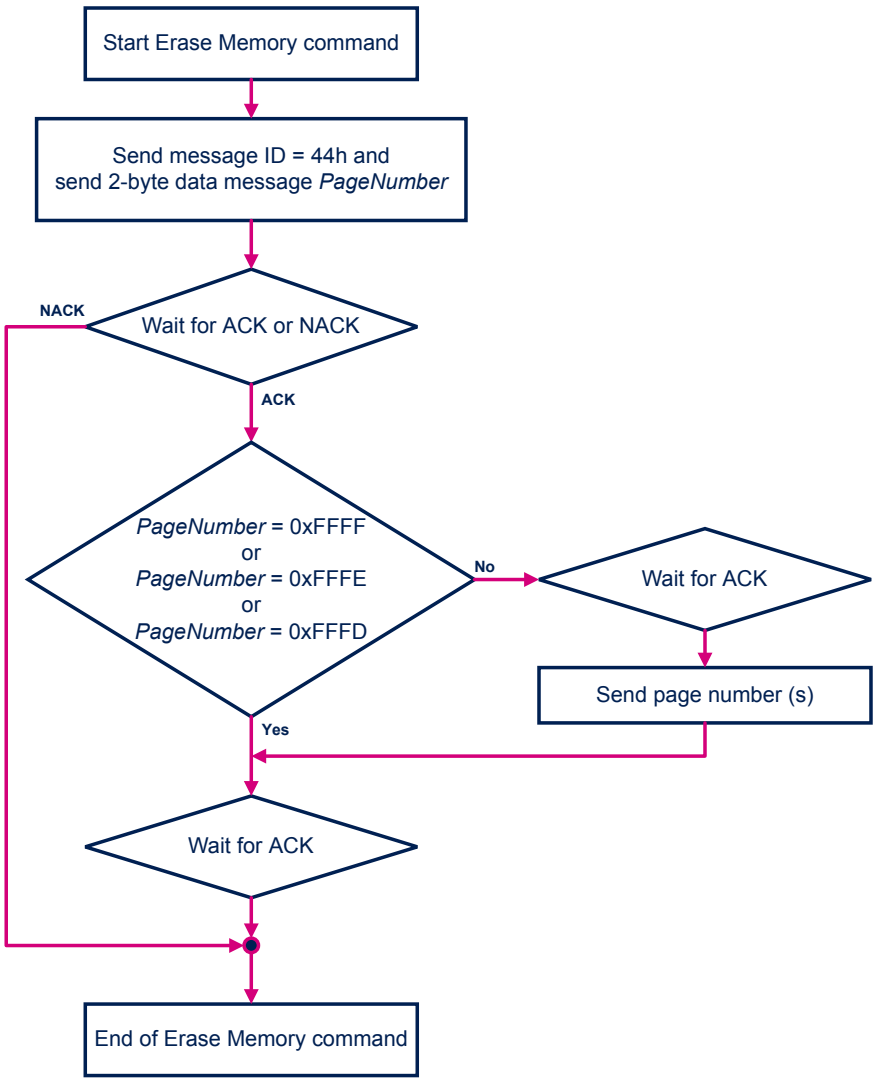
### Erase Memory command specifications

1. The bootloader receives one message that contains N, the number of pages to be erased.
2. The bootloader receives N bytes, every two bytes containing the page number to be erased.

*Note:*

1. *The ACK message sent after the erase operation only indicates that the command is finished. It does not guarantee that the erase operation has succeeded.*
2. *No error is returned when performing erase operations on write-protected sectors.*

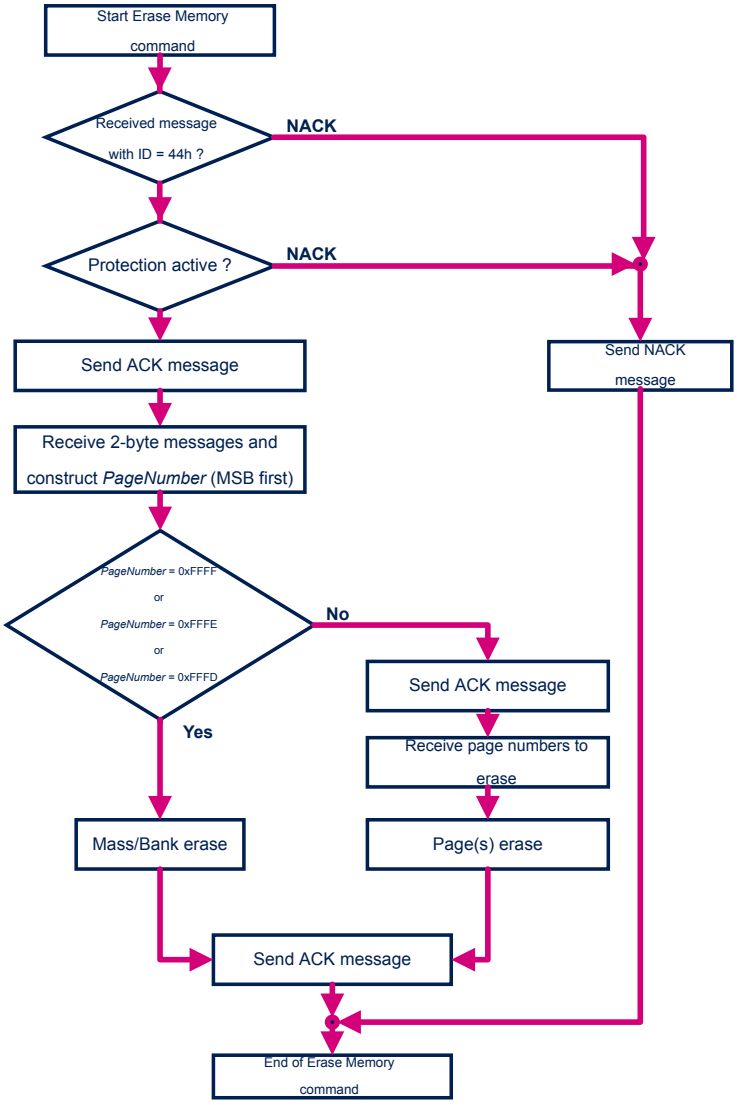
**Figure 14. Erase Memory command (host side)**




The *PageNumber* value sent from the host is MSB first:

- *PageNumber* = (data[0] << 8) | (data[1])
- Page numbers sent for erasing are sent on a 64-byte frame format, of which only *PageNumber* is read by the bootloader (the padding bytes in the 64-byte frame are not read)

**Figure 15. Erase Memory command (device side)**



## 3.8 Write Protect command

The Write Protect command is used to enable the write protection for some or all flash memory sectors. When the bootloader receives the Write Protect command, it transmits an ACK message to the host if the protection is disabled. Otherwise, it transmits a NACK message.

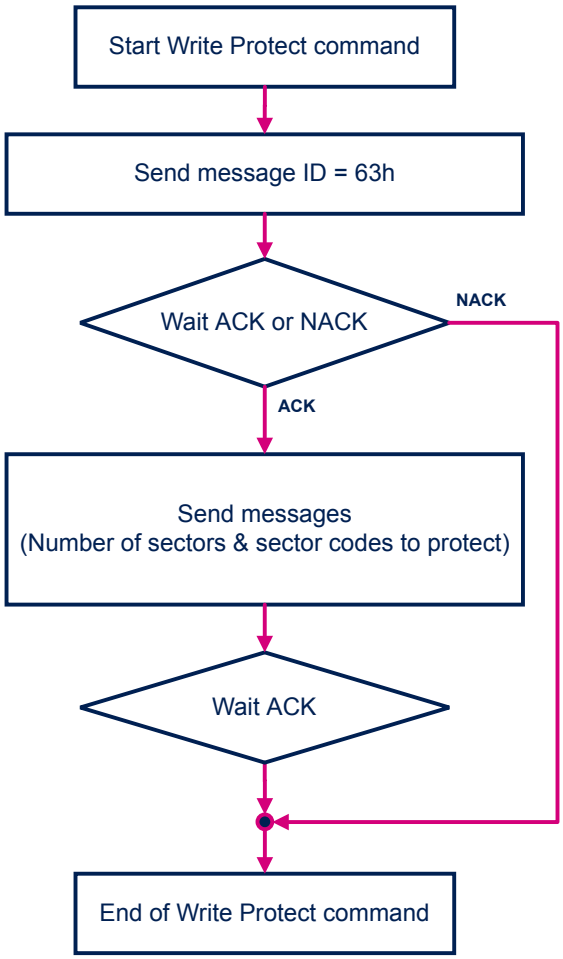
After the transmission of the ACK byte, the bootloader waits to receive the flash memory sector codes from the application.

At the end of the Write Protect command, the bootloader transmits the ACK message.

*Note:*

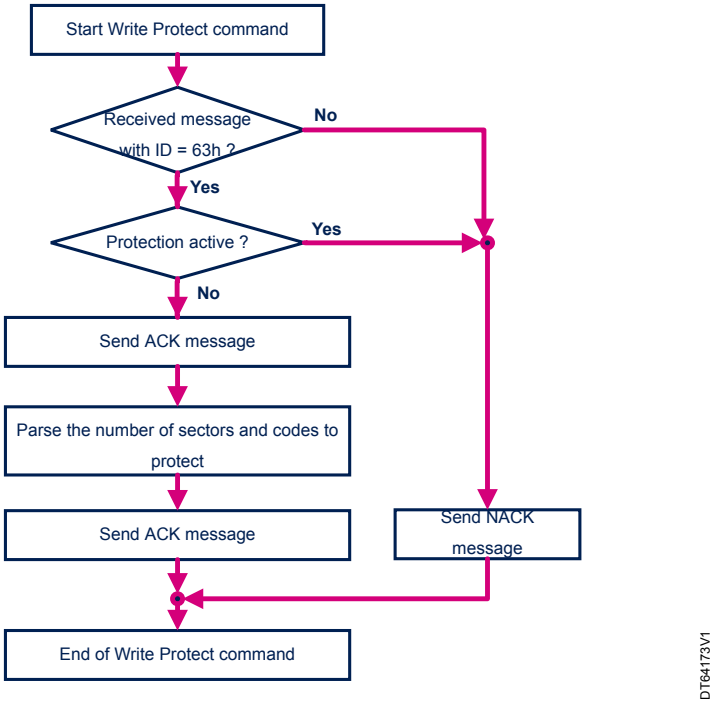
1. *If a second Write Protect command is executed, the flash memory sectors protected by the first command become unprotected, and only the sectors passed within the second Write Protect command become protected.*
2. *The total number of sectors and the sector numbers to be protected are not checked. This means that no error is returned when a command is passed with a wrong number of sectors to be protected, or a wrong sector number.*

**Figure 16. Write Protect command (host side)**




The sector code is sent on one byte. The number of sectors and the codes are sent together with the command ID of the Write Protect command.

Figure 17. **Write Protect command (device side)**




## 3.9 Write Unprotect command

The Write Unprotect command is used to disable the write protection of all the flash memory sectors. When the bootloader receives the Write Unprotect command, it transmits an ACK message to the host if the protection is disabled. Otherwise, it transmits a NACK message. After the transmission of the ACK message, the bootloader disables the write protection of all the flash memory sectors.

Figure 18. **Write Unprotect command (host side)**

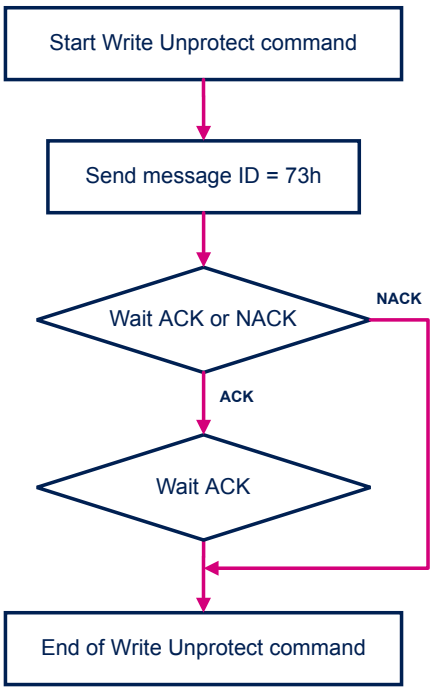
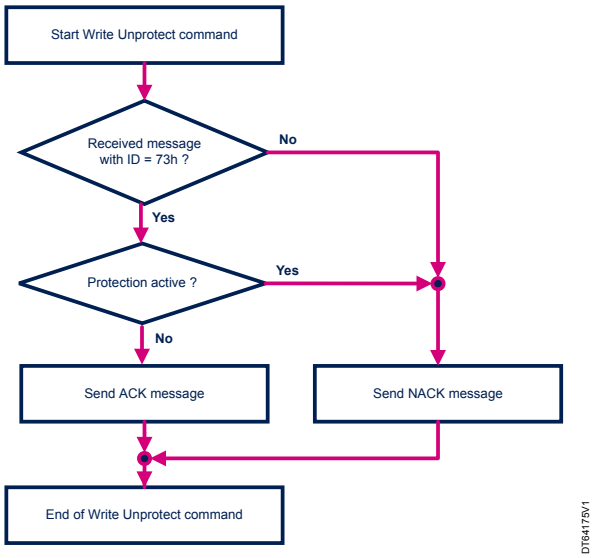


Figure 19. **Write Unprotect command (device side)**




## 3.10 Readout Protect command

The Readout Protect command is used to enable the flash memory readout protection. When the bootloader receives the Readout Protect command, it transmits an ACK message to the host if RDP is disabled. Otherwise, it transmits a NACK message. After the transmission of the ACK message, the bootloader enables the readout protection of the flash memory.

Figure 20. **Readout Protect command (host side)**

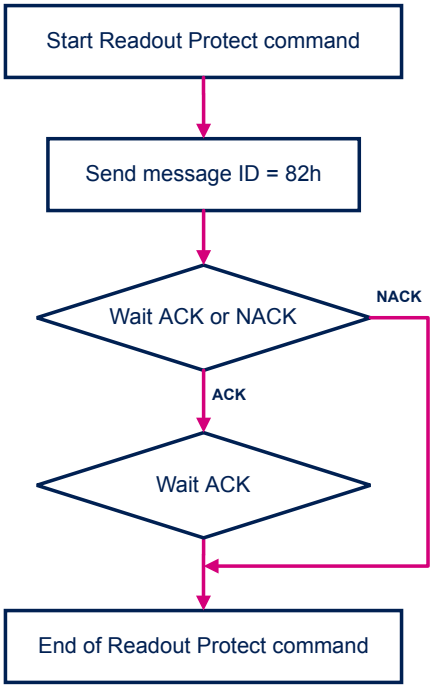
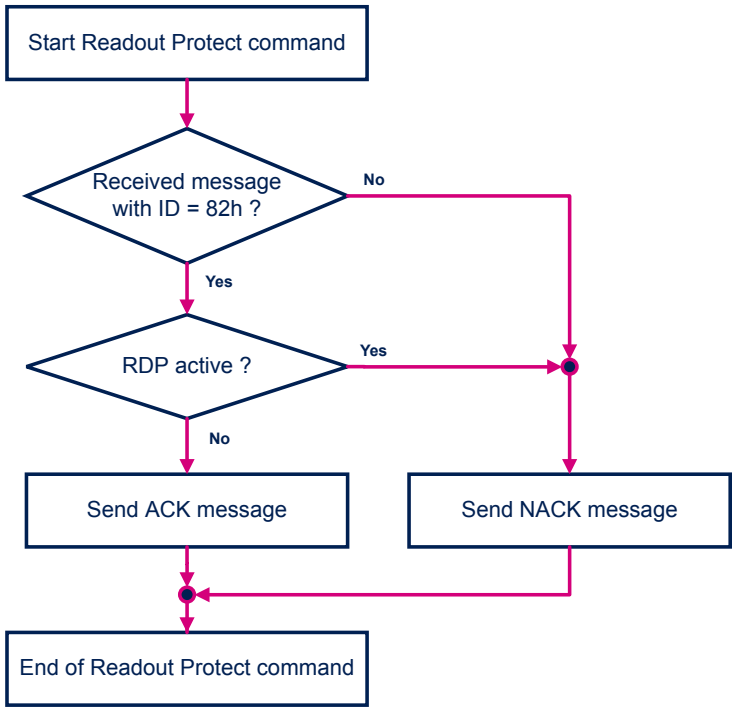


Figure 21. **Readout Protect command (device side)**




## 3.11 Readout Unprotect command

The Readout Unprotect command is used to disable the flash memory readout protection. When the bootloader receives the Readout Unprotect command, it transmits an ACK message to the host. After the transmission of the ACK message, the bootloader deactivates the RDP.

Figure 22. **Readout Unprotect command (host side)**

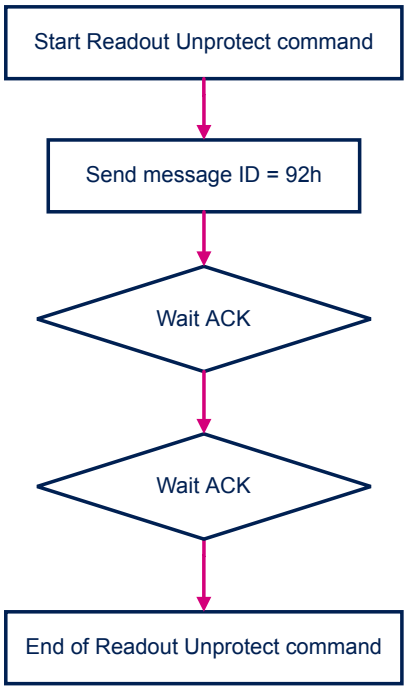
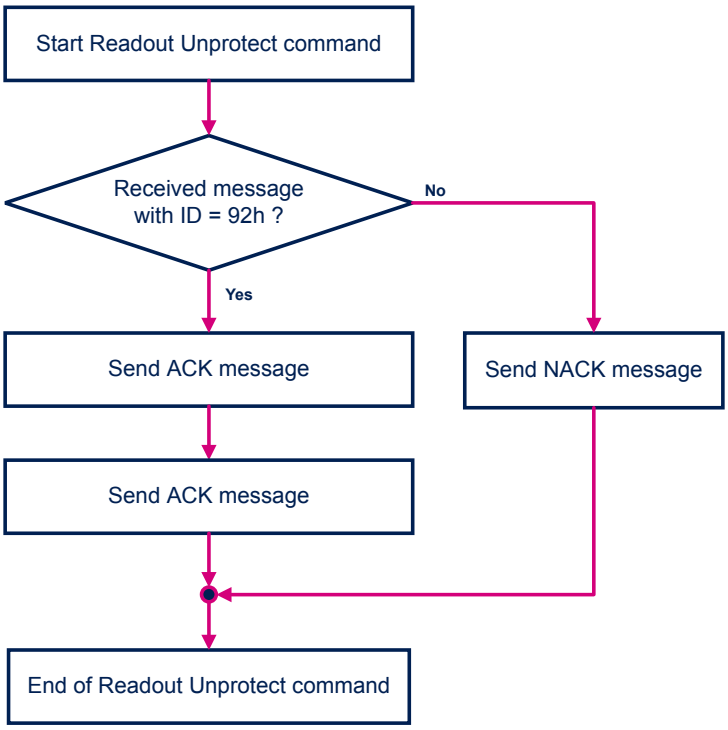


Figure 23. **Readout Unprotect command (device side)**



## 3.12 Special command

Many new bootloader commands are needed to support new features and fulfill users requirements for new applications. To avoid having new specific commands for every project, the generic Special command is created to cover new command requests.

**Figure 24. Special command (host side)**

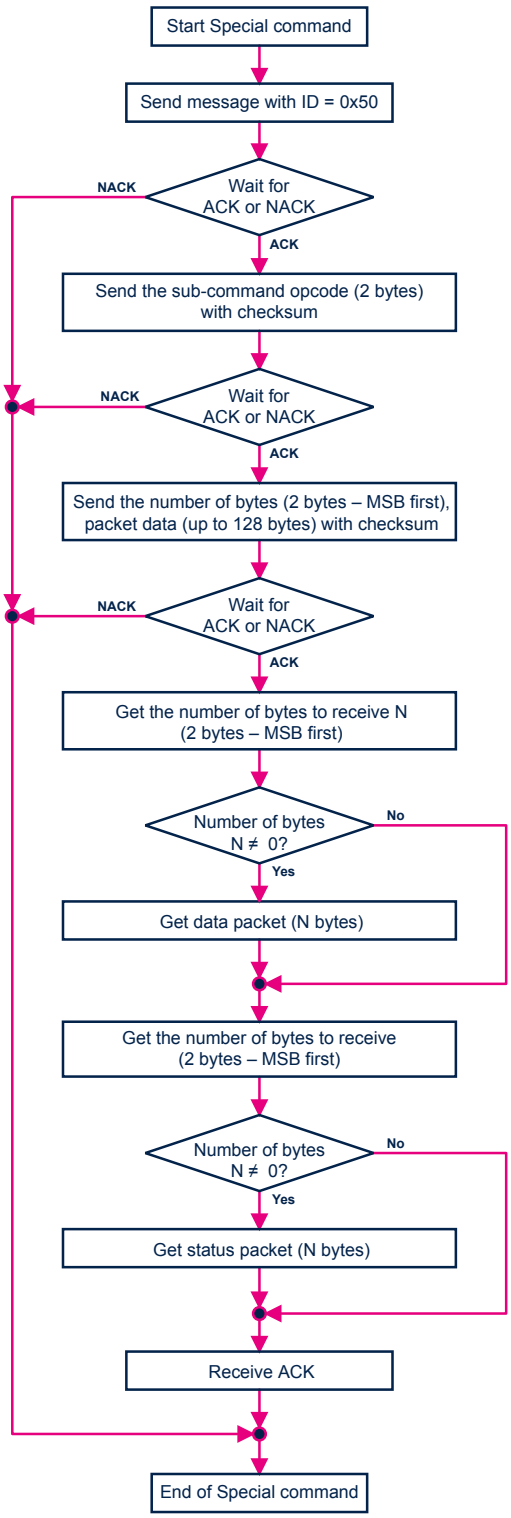
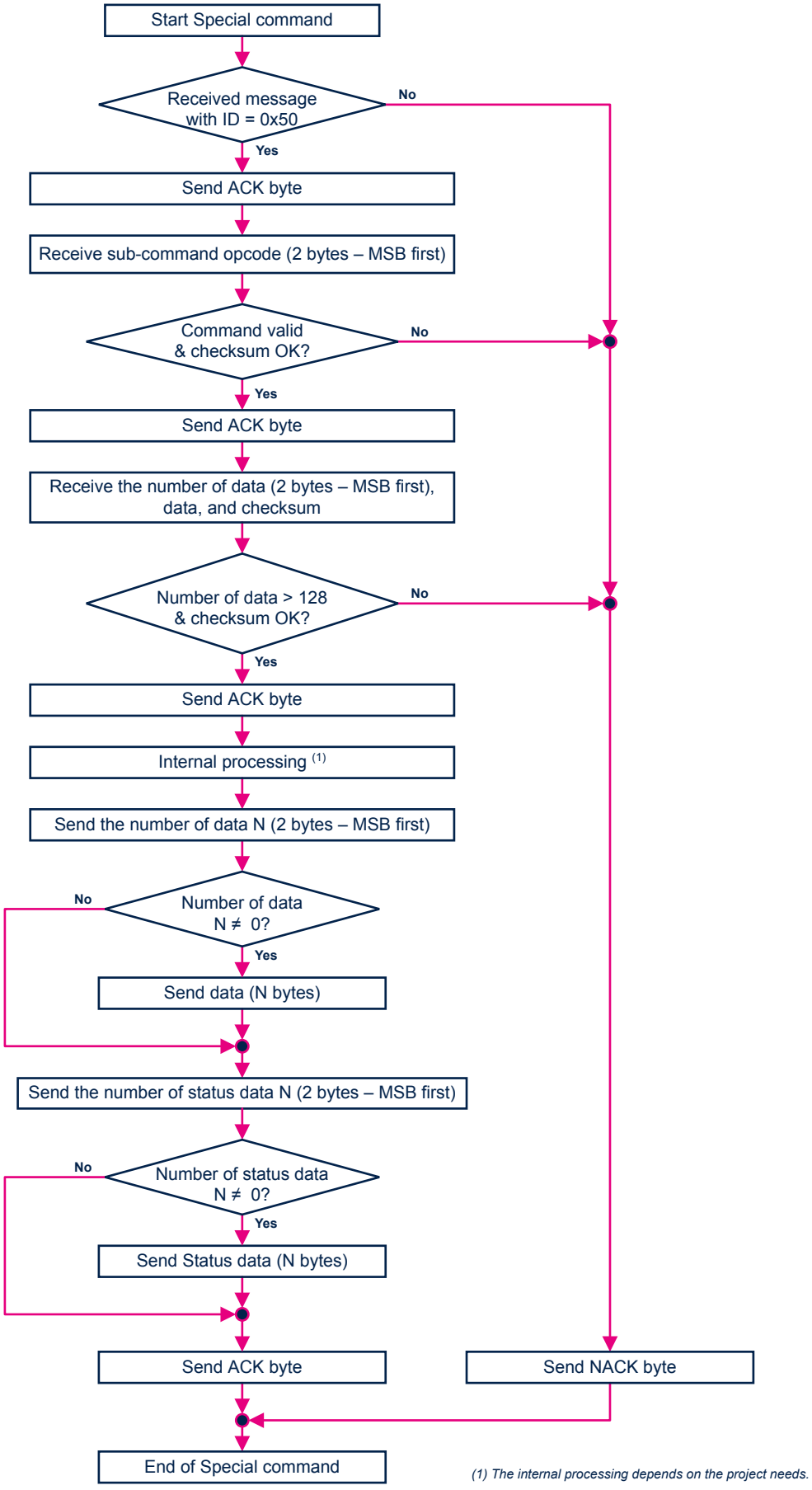


**Figure 25. Special command (device side)**




*(1) The internal processing depends on the project needs.*

When the bootloader receives the Special command, it transmits the ACK byte to the host. Once the ACK is transmitted, the bootloader waits for a subcommand opcode (two bytes, MSB first) and a checksum byte. If the subcommand is supported by the STM32 bootloader and if its checksum is correct, the bootloader transmits an ACK byte, otherwise it transmits a NACK byte and aborts the command.

To keep the Special command generic, the data packet received by the bootloader can have different sizes depending on the subcommand needs.

Therefore, the packet is split in two parts:

- Size of the data (2 bytes, MSB first)
- N bytes of data
  - If N = 0, no data is transmitted
  - N must be less than 128

If all conditions are satisfied (N ≤ 128 and checksum OK), the bootloader transmits an ACK byte. Otherwise, it transmits a NACK byte and aborts the command.

Once the subcommand is executed using the received data, the bootloader sends a response that consists of two consecutive packets:

- Data packet
  - Size of the data (2 bytes, MSB first)
  - N bytes of data
    - If N = 0, no data is transmitted
- Status packet
  - Size of the status data (2 bytes, MSB first)
  - N bytes of data
    - If N = 0, no status data is transmitted

Finally, an ACK byte closes the Special command interaction between the bootloader and the host.

## 3.13 Extended Special command

The Extended Special command is derived from the Special command, with a slightly different behavior. It allows the user to send more data with the addition of a new buffer of 1024 bytes and only returns the commands status as a response.

**Figure 26. Extended Special command (host side)**

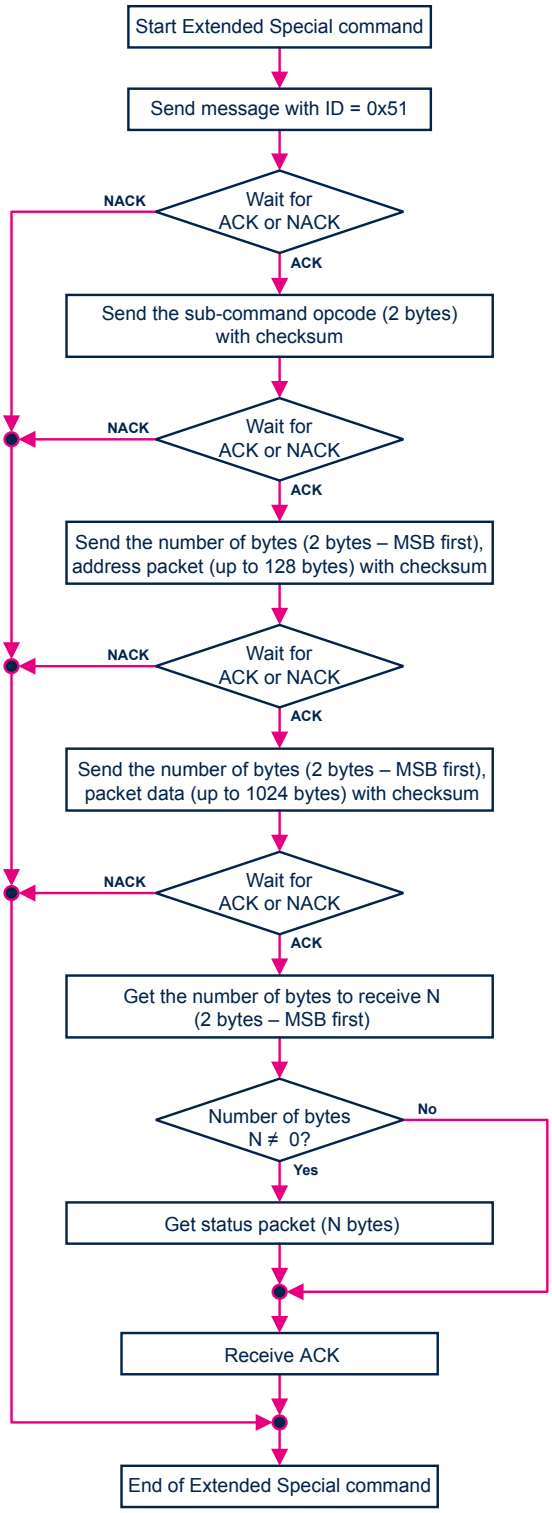
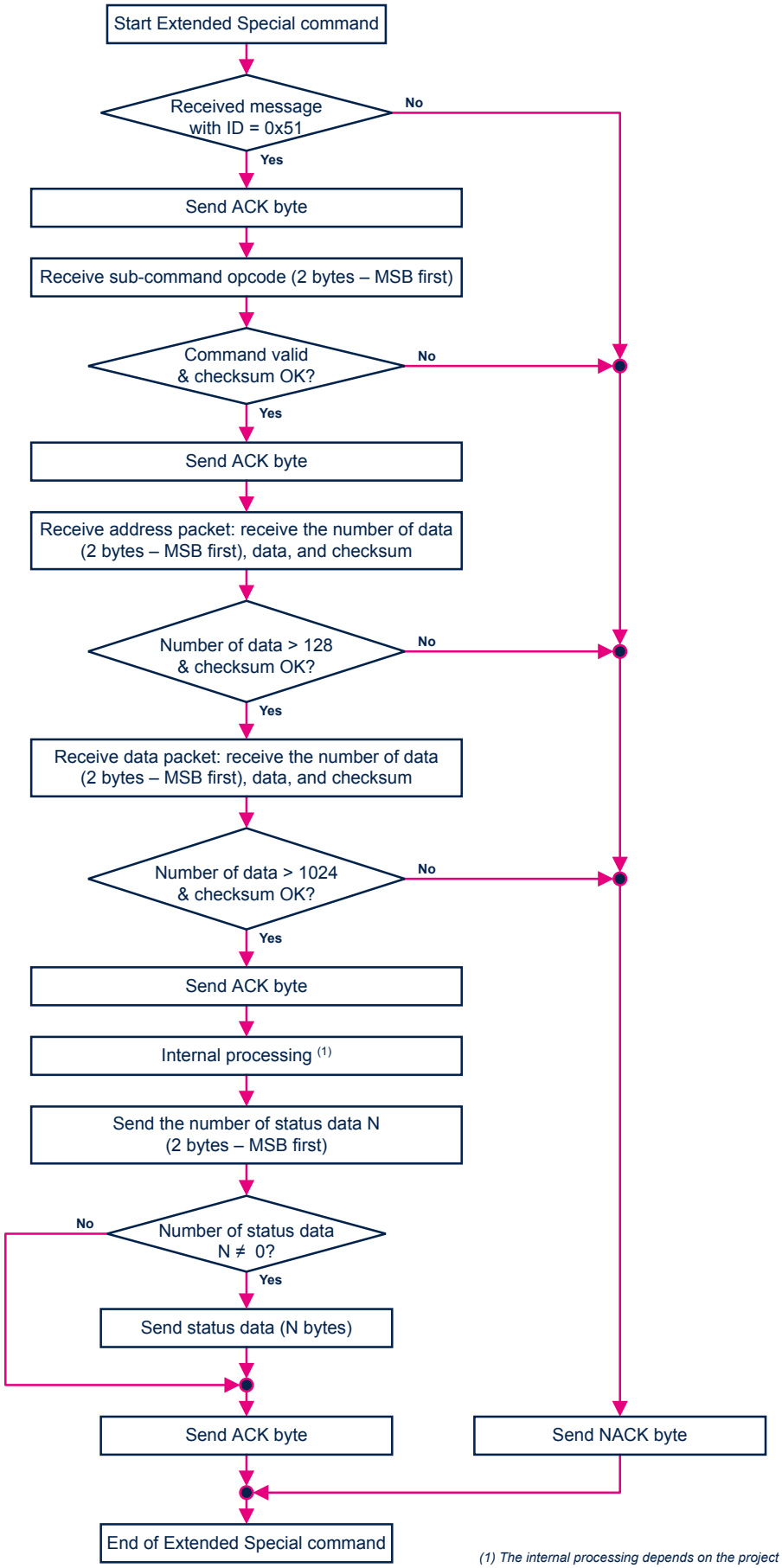


**Figure 27. Extended Special command (device side)**




*(1) The internal processing depends on the project needs.*

When the bootloader receives the Extended Special command, it transmits the ACK byte to the host. Once the ACK is transmitted, the bootloader waits for a subcommand opcode (two bytes, MSB first) and a checksum byte. If the subcommand is supported by the STM32 bootloader and if its checksum is correct, the bootloader transmits an ACK byte. Otherwise, it transmits a NACK byte and aborts the command.

Two packets can then be received depending on the subcommand needs:

- Packet1
  - Data1 packet, where the number of bytes is limited to 128 bytes
- Packet2
  - Data2 packet, where the number of bytes is limited to 1024 bytes

If all conditions are satisfied for packet1 first (N ≤ 128 and checksum OK) and then for packet2 (N ≤ 1024 and checksum OK), the bootloader transmits an ACK byte. Otherwise, it transmits a NACK byte and aborts the command.

Once the subcommand is executed using the received data, the bootloader sends a response that consists of one packet:

- Size of the data (2 bytes, MSB first)
- N bytes of data
  - If N = 0, no data is transmitted

Finally, an ACK byte closes the Extended Special command interaction between the bootloader and the host.

# 4 Bootloader protocol version evolution

Table 3 lists the bootloader versions.

**Table 3. Bootloader protocol versions**

| Date | Revision | Changes |
|---|---|---|
| 16-Aug-2018 | V1.0 | Initial version |
| 27-Feb-2021 | V1.1 | Added support for extended special commands |
| 24-Feb-2023 | V2.0 | The number of commands can vary from device to device with the same protocol version v2.0. To know the supported commands, use the Get command. |
| 15-Mar-2024 | V2.1 | Add check on the received command opcode to not exceed one byte max value. Otherwise, a NACK is sent to the host. |
| 15-Feb-2024 | V2.2 | • Limit detection frame to "ID = 111h, data length =1 and data =5Ah"<br>• Enable global filters rejection |

# Revision history

Table 4. **Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 14-Nov-2019 | 1 | Initial release. |
| 28-Jun-2021 | 2 | Extended the range of applicable products to the STM32G0 Series and STM32G4 Series.<br>Added *Section 3.12 Special command* and *Section 3.13 Extended Special command*. |
| 09-Feb-2022 | 3 | Extended the range of applicable products to the STM32U5 series. |
| 14-Feb-2023 | 4 | Extended the range of applicable products to the STM32H5 series (updated Table 1. Applicable products and Section 1: Bootloader code sequence)<br>Replaced "bootloader version" with "protocol version" throughout the document<br>Updated note 1 to Table 2. FDCAN bootloader commands<br>Updated information below Figure 3. Get command (device side)<br>Replaced "readout protection" or RDP with "protection" in Section 3.4: Read Memory command to Section 3.9: Write Unprotect command<br>Added Section 4: Bootloader protocol version evolution |
| 02-Nov-2023 | 5 | Updated:<br>• Document title.<br>• Table 2. FDCAN bootloader commands.<br>• Erase Memory command specifications |
| 07-Mar-2024 | 6 | Updated: Table 3. Bootloader protocol versions |
| 22-May-2025 | 7 | Updated Table 1. Applicable products.<br>Updated Figure 1. Bootloader for STM32 with FDCAN.<br>Updated Section 2: FDCAN settings.<br>Updated Figure 8. Read Memory command (host side) and Figure 9. Read Memory command (device side). |

# Contents

1 **Bootloader code sequence** . . . . . 2
2 **FDCAN settings** . . . . . 3
3 **Bootloader command set** . . . . . 5
3.1 Get command . . . . . 6
3.2 Get Version command . . . . . 9
3.3 Get ID command . . . . . 10
3.4 Read Memory command . . . . . 11
3.5 Go command . . . . . 13
3.6 Write Memory command . . . . . 14
3.7 Erase Memory command . . . . . 16
3.8 Write Protect command . . . . . 19
3.9 Write Unprotect command . . . . . 20
3.10 Readout Protect command . . . . . 21
3.11 Readout Unprotect command . . . . . 22
3.12 Special command . . . . . 24
3.13 Extended Special command . . . . . 27
4 **Bootloader protocol version evolution** . . . . . 30
**Revision history** . . . . . 31
**List of tables** . . . . . 33
**List of figures** . . . . . 34

# List of tables

Table 1. Applicable products . . . 1
Table 2. FDCAN bootloader commands . . . 5
Table 3. Bootloader protocol versions . . . 30
Table 4. Document revision history . . . 31

# List of figures

Figure 1. Bootloader for STM32 with FDCAN . . . 2
Figure 2. Get command (host side) . . . 6
Figure 3. Get command (device side) . . . 7
Figure 4. Get Version command (host side) . . . 9
Figure 5. Get Version command (device side) . . . 9
Figure 6. Get ID command (host side) . . . 10
Figure 7. Get ID command (device side) . . . 10
Figure 8. Read Memory command (host side) . . . 11
Figure 9. Read Memory command (device side) . . . 12
Figure 10. Go command (host side) . . . 13
Figure 11. Go command (device side) . . . 14
Figure 12. Write Memory command (host side) . . . 15
Figure 13. Write Memory command (device side) . . . 16
Figure 14. Erase Memory command (host side) . . . 17
Figure 15. Erase Memory command (device side) . . . 18
Figure 16. Write Protect command (host side) . . . 19
Figure 17. Write Protect command (device side) . . . 20
Figure 18. Write Unprotect command (host side) . . . 20
Figure 19. Write Unprotect command (device side) . . . 21
Figure 20. Readout Protect command (host side) . . . 21
Figure 21. Readout Protect command (device side) . . . 22
Figure 22. Readout Unprotect command (host side) . . . 22
Figure 23. Readout Unprotect command (device side) . . . 23
Figure 24. Special command (host side) . . . 24
Figure 25. Special command (device side) . . . 25
Figure 26. Extended Special command (host side) . . . 27
Figure 27. Extended Special command (device side) . . . 28