

Introduction

STM32CubeProgrammer (STM32CubeProg) provides an all-in-one software tool to program STM32 devices in any environment: multi-OS, graphical user interface or command line interface, support for a large choice of connections (JTAG, SWD, USB, UART, SPI, CAN, I2C), with manual operation or automation through scripting.

This document details the hardware and software environment prerequisites, as well as the available STM32CubeProgrammer software features.



Contents

1	Getting started	10
1.1	System requirements	10
1.2	Installing STM32CubeProgrammer	10
1.2.1	Linux install	10
1.2.2	Windows install	11
1.2.3	macOS install	11
1.2.4	DFU driver	11
1.2.5	ST-LINK driver	13
2	STM32CubeProgrammer user interface for MCUs	14
2.1	Main window	14
2.1.1	Main menu	14
2.1.2	Log panel	15
2.1.3	Progress bar	15
2.1.4	Target configuration panel	16
2.2	Memory and file edition	24
2.2.1	Reading and displaying target memory	24
2.2.2	Reading and displaying a file	25
2.3	Memory programming and erasing	26
2.3.1	Internal Flash memory programming	26
2.3.2	External Flash memory programming	27
2.3.3	Developing customized loaders for external memory	29
2.4	Option bytes	31
2.5	Automatic mode	32
2.6	In application programming (IAP/USBx)	35
2.7	Flash the co-processor binary using graphical interface	36
2.7.1	FUS / Stack upgrade	36
2.7.2	Key provisioning	39
2.8	Serial wire viewer (SWV)	41
2.9	STM32CubeProgrammer Script Manager platform for MCUs	42
2.9.1	Introduction for the usage scenarios of Script Manager	42
2.9.2	Script Manager usage	42
2.10	DFU IAP/USBx with custom PID and VID	47

2.11	SigFox™ credentials	48
2.12	Register Viewer	49
2.13	Hard Fault analyzer	50
2.13.1	Description	50
2.13.2	Example	52
2.13.3	Fault Analyzer note	52
2.13.4	Secure Fault Analyzer for Cortex-M33	53
2.14	Fill memory command	54
2.15	Fill memory operation	56
2.16	Blank check command	57
2.17	Blank check operation	58
2.18	Compare Flash memory with file	62
2.19	Comparison between two files	67
2.20	LiveUpdate feature	72
3	STM32CubeProgrammer command line interface (CLI) for MCUs	73
3.1	Command line usage	73
3.2	Generic commands	75
3.2.1	Connect command	75
3.2.2	Erase command	83
3.2.3	Download command	84
3.2.4	Download 32-bit data command	84
3.2.5	Download 64-bit data command	85
3.2.6	Read command	85
3.2.7	Start command	86
3.2.8	Debug commands	86
3.2.9	List command	87
3.2.10	QuietMode command	88
3.2.11	Verbosity command	88
3.2.12	Log command	89
3.2.13	External loader command	90
3.2.14	External loader command with bootloader interface	91
3.2.15	Read unprotect command	92
3.2.16	TZ regression command	92
3.2.17	Option bytes command	92
3.2.18	Safety lib command	92

	3.2.19	Secure programming SFI specific commands	96
	3.2.20	Secure programming SFIx specific commands	96
	3.2.21	HSM related commands	98
	3.2.22	STM32WB specific commands	99
	3.2.23	Serial wire viewer (SWV) command	101
	3.2.24	Specific commands for STM32WL	102
	3.2.25	SigFox credential commands	104
	3.2.26	Register Viewer	106
	3.2.27	Hard fault analyzer	107
	3.2.28	RDP regression with password	108
4		STM32CubeProgrammer user interface for MPUs	110
	4.1	Main window	110
	4.2	Programming windows	111
5		STM32CubeProgrammer CLI for MPUs	112
	5.1	Available commands for STM32MP1	112
	5.1.1	Connect command	112
	5.1.2	GetPhase command	113
	5.1.3	Download command	113
	5.1.4	Flashing service	114
	5.1.5	Start command	114
	5.1.6	Read partition command	115
	5.1.7	List command	115
	5.1.8	QuietMode command	116
	5.1.9	Verbosity command	116
	5.1.10	Log command	116
	5.1.11	OTP programming	118
	5.1.12	Programming SAFMEM command	118
	5.1.13	Detach command	119
	5.1.14	GetCertif command	119
	5.1.15	Write blob command	119
	5.1.16	Display command	119
	5.2	Secure programming SSP specific commands	119
6		STM32CubeProgrammer C++ API	122

7	Revision history	123
---	------------------------	-----

List of tables

Table 1. Operations supported by Script Manager 42

Table 2. Document revision history 123



List of figures

Figure 1.	Deleting the old driver software	12
Figure 2.	STM32 DFU device with DfuSe driver	12
Figure 3.	STM32 DFU device with STM32CubeProgrammer driver	12
Figure 4.	STM32CubeProgrammer main window	14
Figure 5.	Expanded main menu	15
Figure 6.	ST-LINK configuration panel	16
Figure 7.	UART configuration panel	18
Figure 8.	USB configuration panel	19
Figure 9.	Target information panel	20
Figure 10.	SPI configuration panel	21
Figure 11.	CAN configuration panel	22
Figure 12.	I2C configuration panel	23
Figure 13.	Memory and file edition: Device memory tab	24
Figure 14.	Memory and file edition: Contextual menu	25
Figure 15.	Memory and file edition: File display	25
Figure 16.	Flash memory programming and erasing (internal memory)	26
Figure 17.	Flash memory programming (external memory)	28
Figure 18.	Flash memory erasing (external memory)	28
Figure 19.	Option bytes panel	31
Figure 20.	Automatic mode in Erasing & Programming window	32
Figure 21.	Automatic mode Log traces	33
Figure 22.	Algorithm	35
Figure 23.	STM32Cube Programmer in IAP mode	36
Figure 24.	STM32CubeProgrammer API SWD connection	37
Figure 25.	Steps for firmware upgrade	37
Figure 26.	Pop-up confirming successful firmware delete	38
Figure 27.	Pop-up confirming successful firmware upgrade	38
Figure 28.	Update authentication key	39
Figure 29.	Pop-up requesting to lock authentication key	40
Figure 30.	Store customer key	40
Figure 31.	SWV window	41
Figure 32.	Output of Script Manager example 1	45
Figure 33.	Output of Script Manager example 2	46
Figure 34.	Connect via USB DFU panel	47
Figure 35.	Main window after the connection	48
Figure 36.	SigFox credentials	49
Figure 37.	Register Viewer window	49
Figure 38.	Fault Analyzer window	51
Figure 39.	Fault analyzer GUI view when Hard Fault detected	52
Figure 40.	CCR bits	53
Figure 41.	Example 1	55
Figure 42.	Example 2	55
Figure 43.	Sub-menu displayed from "Read" combo-box	56
Figure 44.	Sub-menu displayed with right click on "Device memory" tab	56
Figure 45.	Sub-menu displayed with right click on the cell of grid	56
Figure 46.	Parameters initialization	57
Figure 47.	Example 1: Flash memory is not blank at address 0x08000014	57
Figure 48.	Example 1: Flash memory is blank	58

Figure 49.	Sub-menu displayed from “Read” combo-box	58
Figure 50.	Sub-menu displayed with right click on “Device memory” tab	59
Figure 51.	Sub-menu displayed with right click on the cell of grid	59
Figure 52.	First address with data	60
Figure 53.	Example 1: Flash memory is blank	61
Figure 54.	Example 2: Flash memory is not blank	62
Figure 55.	Sub-menu displayed from “Read” combo-box	62
Figure 56.	Sub-menu displayed with right click on “Device memory” tab	63
Figure 57.	Sub-menu displayed with right click on the cell of grid	63
Figure 58.	Sub-menu displayed with add tab button	63
Figure 59.	Sub-menu displayed with right click on the opened file tab	64
Figure 60.	Sub-menu displayed from “Download” combo-box displayed in file tab	64
Figure 61.	Data width: 32 bits	65
Figure 62.	Data width: 16 bits	65
Figure 63.	Data width: 8 bits	65
Figure 64.	Data width: 32 bits	66
Figure 65.	Data width: 16 bits	66
Figure 66.	Data width: 8 bits	66
Figure 67.	Before editing the Flash memory	67
Figure 68.	After editing the Flash memory	67
Figure 69.	Multiple comparisons	67
Figure 70.	Sub-menu displayed from “Read” combo-box in device memory tab	68
Figure 71.	Sub-menu displayed with right click on “Device memory” tab	68
Figure 72.	Sub-menu displayed with right click on the cell of grid	68
Figure 73.	Sub-menu displayed with add tab button	69
Figure 74.	Sub-menu displayed with right click on the opened file tab	69
Figure 75.	Sub-menu displayed from “Download” combo-box displayed in file tab	69
Figure 76.	Data width: 32 bits	70
Figure 77.	Data width: 16 bits	70
Figure 78.	Data width: 8 bits	71
Figure 79.	Multiple comparisons	71
Figure 80.	Live update of data	72
Figure 81.	STM32CubeProgrammer: available commands	74
Figure 82.	Connect operation using RS232	77
Figure 83.	Enabling COM DTR pin	78
Figure 84.	Connect operation using USB	79
Figure 85.	Connect operation using USB DFU options	80
Figure 86.	Connect operation using SWD debug port	80
Figure 87.	Connect operation using SPI port	82
Figure 88.	Connect operation using CAN port	82
Figure 89.	Connect operation using I2C port	83
Figure 90.	Download operation	84
Figure 91.	Read 32-bit operation	86
Figure 92.	The available serial ports list	88
Figure 93.	Verbosity command	89
Figure 94.	Log command	89
Figure 95.	Log file content	90
Figure 96.	Safety lib command	93
Figure 97.	Flash memory mapping	94
Figure 98.	Flash memory mapping example	95
Figure 99.	SWV command	102
Figure 100.	Disable security	103

Figure 101. Configure option bytes to their default values	104
Figure 102. Example of -ssigfoxc command	105
Figure 103. Example (1) of -wsigfoxc command	105
Figure 104. Example (2) of -wsigfoxc command	106
Figure 105. Read core and MCU registers	107
Figure 106. Fault Analyzer CLI view when Hard Fault detected	108
Figure 107. STM32CubeProgrammer main window	110
Figure 108. TSV programming window	111
Figure 109. Connect operation using RS232	113
Figure 110. Download operation	114
Figure 111. TSV file format	114
Figure 112. Log file content	117
Figure 113. SSP successfully installed	121

1 Getting started

This section describes the requirements and procedures to install the STM32CubeProgrammer software tool.

STM32CubeProgrammer supports STM32 32-bit MCUs based on Arm^{®(a)} Cortex[®]-M processors and STM32 32-bit MPUs based on Arm[®] Cortex[®]-A processors.

1.1 System requirements

Supported operating systems and architectures:

- Linux[®] 64-bit
- Windows[®] 7/8/10 32-bit and 64-bit
- macOS[®] (minimum version OS X[®] Yosemite)

There is no need to install any Java[™] SE Run Time Environment since version 2.6.0. The STM32CubeProgrammer runs with a bundled JRE available within the downloaded package and no longer with the one installed on your machine.

Note: The bundled JRE is Liberica 8.0.265.

For macOS software minimum requirements are

- Xcode[®] must be installed on macOS computers
- both Xcode[®] and Rosetta[®] must be installed on macOS computers embedding Apple[®] M1 processor

The minimal supported screen resolution is 1024x768.

1.2 Installing STM32CubeProgrammer

This section describes the requirements and the procedure for the use of the STM32CubeProgrammer software. The setup also offers optional installation of the “STM32 trusted package creator” tool, used to create secure firmware files for secure firmware install and update. For more information, check *STM32 Trusted Package Creator tool software description* (UM2238), available on www.st.com.

1.2.1 Linux install

If you are using a USB port to connect to the STM32 device, install the libusb1.0 package by typing the following command:

```
sudo apt-get install libusb-1.0.0-dev
```

To use ST-LINK probe or USB DFU to connect to a target, copy the rules files located under *Driver/rules* folder in */etc/udev/rules.d/* on Ubuntu (*"sudo cp *.* /etc/udev/rules.d"*).

Note: libusb1.0.12 version or higher is required to run STM32CubeProgrammer.

arm

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

To install the STM32CubeProgrammer tool, download and extract the zip package on your Linux machine from STM32CubeProg-Linux part number on the website and execute *SetupSTM32CubeProgrammer-vx.y.z.linux*, which guides you through the installation process. In Ubuntu 20 STM32CubeProgrammer icon is not enabled by default. To enable it right click on the icon and choose “Allow launching”.

1.2.2 Windows install

To install the STM32CubeProgrammer tool, download and extract the zip package from STM32CubeProg-Win-32bits or STM32CubeProg-Win-64bits for, respectively, Windows 32 bits and Windows 64 bits, and execute *SetupSTM32CubeProgrammer-vx.y.z.exe*, which guides you through the installation process.

1.2.3 macOS install

To install the STM32CubeProgrammer tool, download and extract the zip package from STM32CubeProg-Mac part number on the website and execute *SetupSTM32CubeProgrammer-vx.y.z.app*, which guides you through the installation process.

Note: If the installation fails, launch it in CLI mode using the command
`./SetupSTM32CubeProgrammer-x.y.z.app/Contents/MacOs/SetupSTM32CubeProgrammer-x_y_z_macos.`

Make sure you have administrator rights, then double-click *SetupSTM32CubeProgrammer-macos* application file to launch the installation wizard.

In case of error, try one of the following fixes:

- `$sudo xattr -cr ~/SetupSTM32CubeProgrammer-macos.app`
- launch the .exe file with the command
`sudo java -jar SetupSTM32CubeProgrammer-2.7.0.exe.`

1.2.4 DFU driver

If you are using the STM32 device in USB DFU mode, install the STM32CubeProgrammer's DFU driver by running the “*STM32 Bootloader.bat*” file. This driver is provided with the release package, it can be found in the DFU driver folder.

If you have the DFUSE driver installed on your machine, first uninstall it, then reboot the machine and run the previously mentioned “.bat” file. You must check the ‘Delete the driver software for this device’ option to avoid reinstalling the old driver when, later, a board is plugged in.

Figure 1. Deleting the old driver software

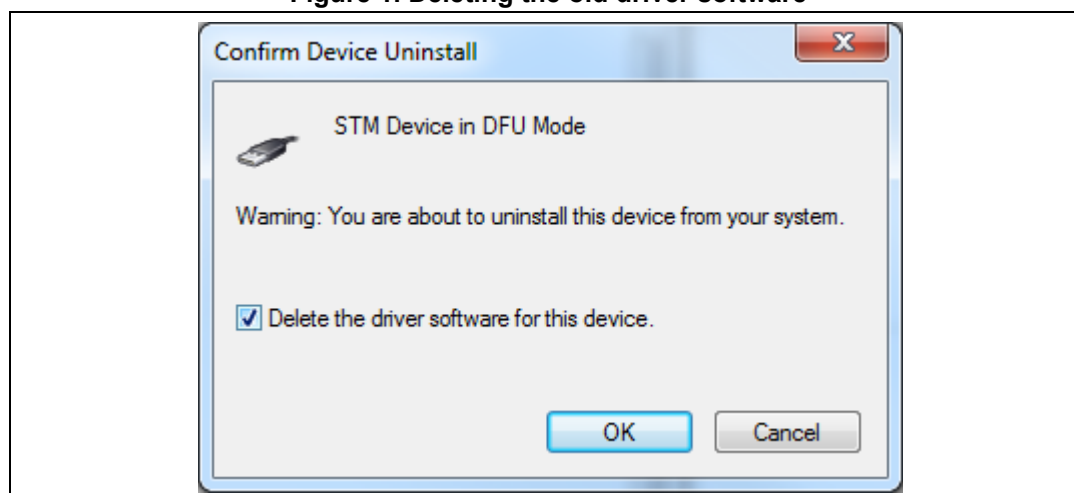
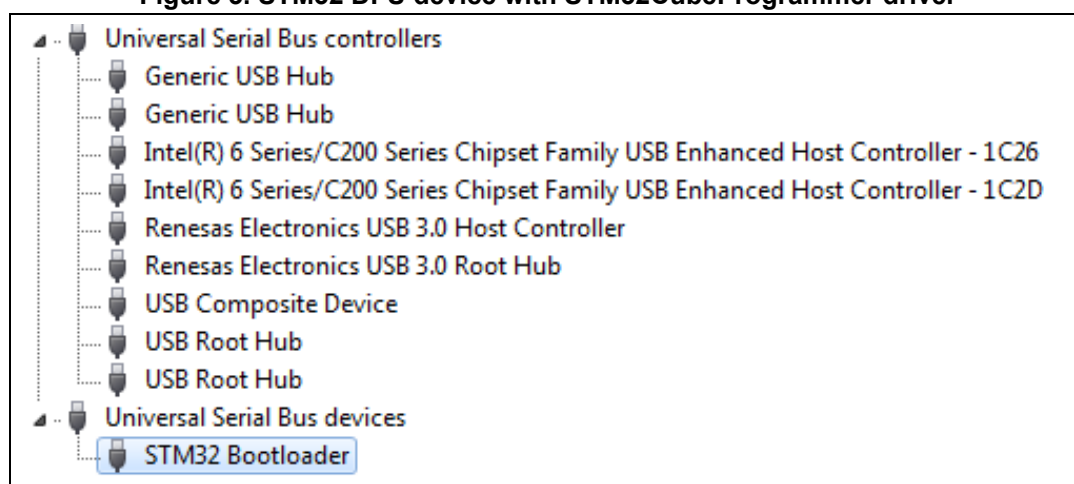


Figure 2. STM32 DFU device with DfuSe driver



Figure 3. STM32 DFU device with STM32CubeProgrammer driver



Note: When using USB DFU interface or STLink interface on a Windows 7 PC, ensure that all the drivers of the USB 3.0 controller drivers are updated. Older versions of the drivers may have bugs that prevent access or cause connection problems with USB devices.

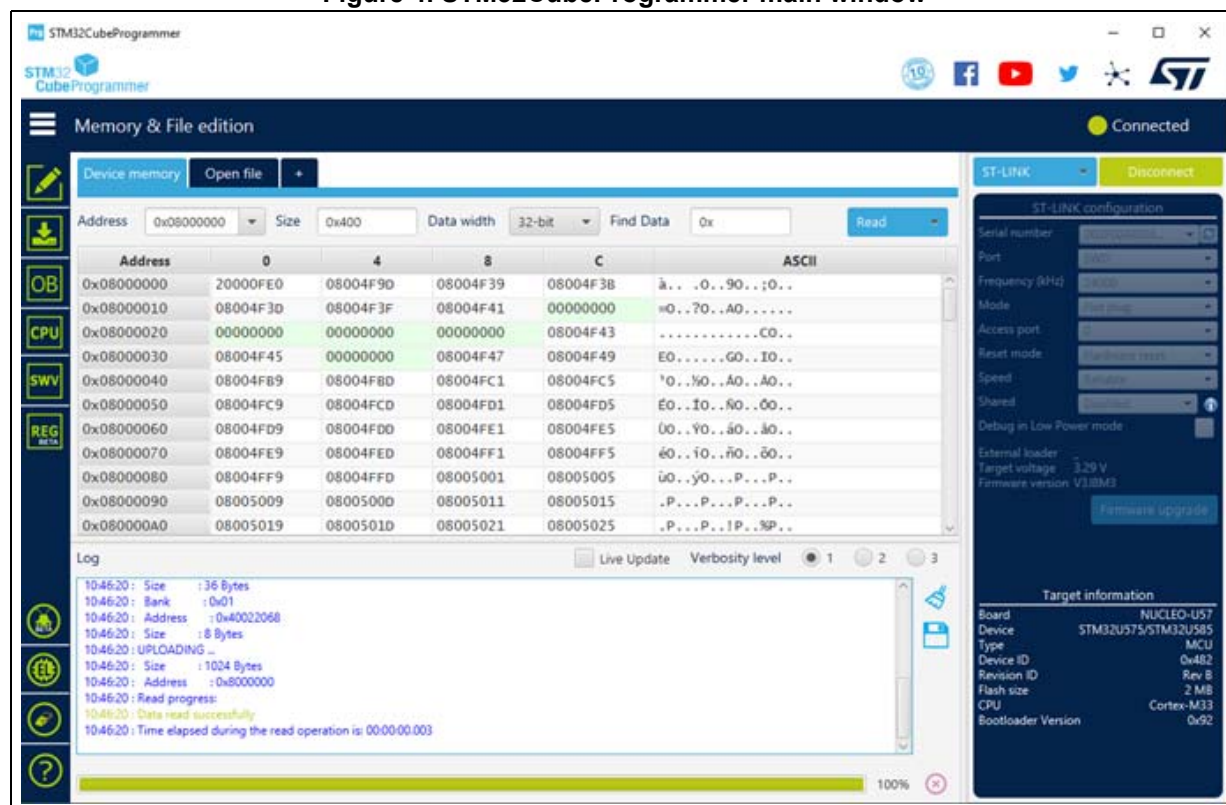
1.2.5 ST-LINK driver

To connect to an STM32 device through a debug interface using ST-LINK/V2, ST-LINKV2-1 or ST-LINK-V3, install the ST-LINK driver by running the *“stlink_winusb_install.bat”* file. This driver is provided with the release package, it can be found under the *“Driver/stsw-link009_v3”* folder.

2 STM32CubeProgrammer user interface for MCUs

2.1 Main window

Figure 4. STM32CubeProgrammer main window

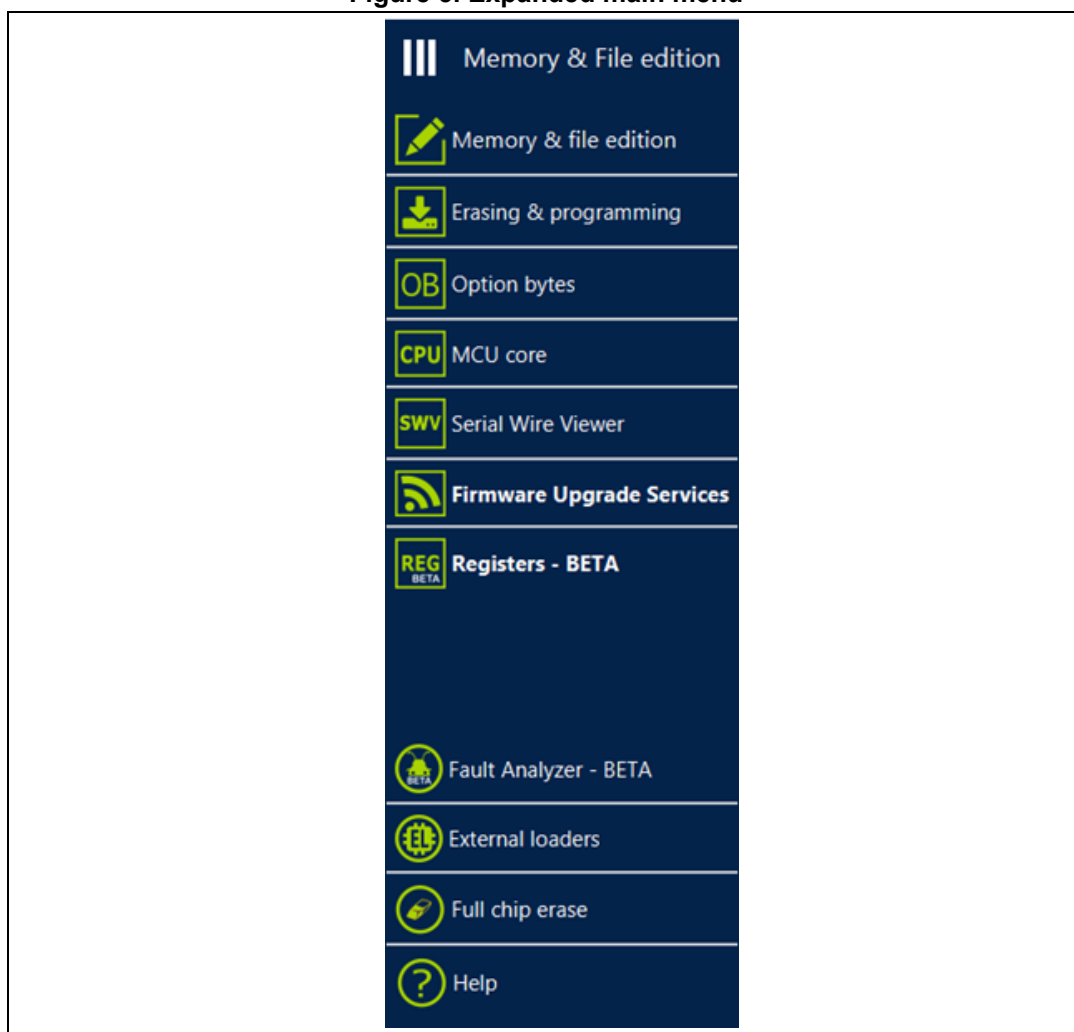


The main window is composed of the parts described in the following sections.

2.1.1 Main menu

The Main menu allows the user to switch between the three main panels of the Memory and file edition, Memory programming and erasing, and Option bytes tools. The other panels will be displayed basing on the used device. By clicking on the Hamburger menu (the three-line button) on the top left corner, the menu expands and displays the textual description shown in [Figure 5](#).

Figure 5. Expanded main menu



2.1.2 Log panel

Displays errors, warnings, and informational events related to the operations executed by the tool. The verbosity of the displayed messages can be refined using the verbosity radio buttons above the log text zone. The minimum verbosity level is 1, and the maximum is 3, in which all transactions via the selected interface are logged. All displayed messages are time stamped with the format “hh:mm:ss:ms” where “hh” is for hours, “mm” for minutes, “ss” for seconds and “ms” for milliseconds (in three digits).

On the right of the log panel there are two buttons, the first to clean the log, and the second to save it to a log file.

2.1.3 Progress bar

The progress bar visualizes the progress of any operation or transaction done by the tool (e.g. Read, Write, Erase). You can abort any ongoing operation by clicking on the ‘Stop’ button in front of the progress bar.

2.1.4 Target configuration panel

This is the first panel to look at before connecting to a target. It allows the user to select the target interface; either the debug interface using ST-LINK debug probe or the bootloader interface over UART, USB, SPI, CAN or I2C.

The refresh button allows you to check the available interfaces connected to the PC. When this button is pressed while the ST-LINK interface is selected, the tool checks the connected ST-LINK probes and lists them in the Serial numbers combo box. If the UART interface is selected, it checks the available com ports of the PC, and lists them in the Port combo box. If the USB interface is selected, it checks the USB devices in DFU mode connected to the PC and lists them also in the Port combo box. Each interface has its own settings, to be set before connection.

ST-LINK settings

Figure 6. ST-LINK configuration panel



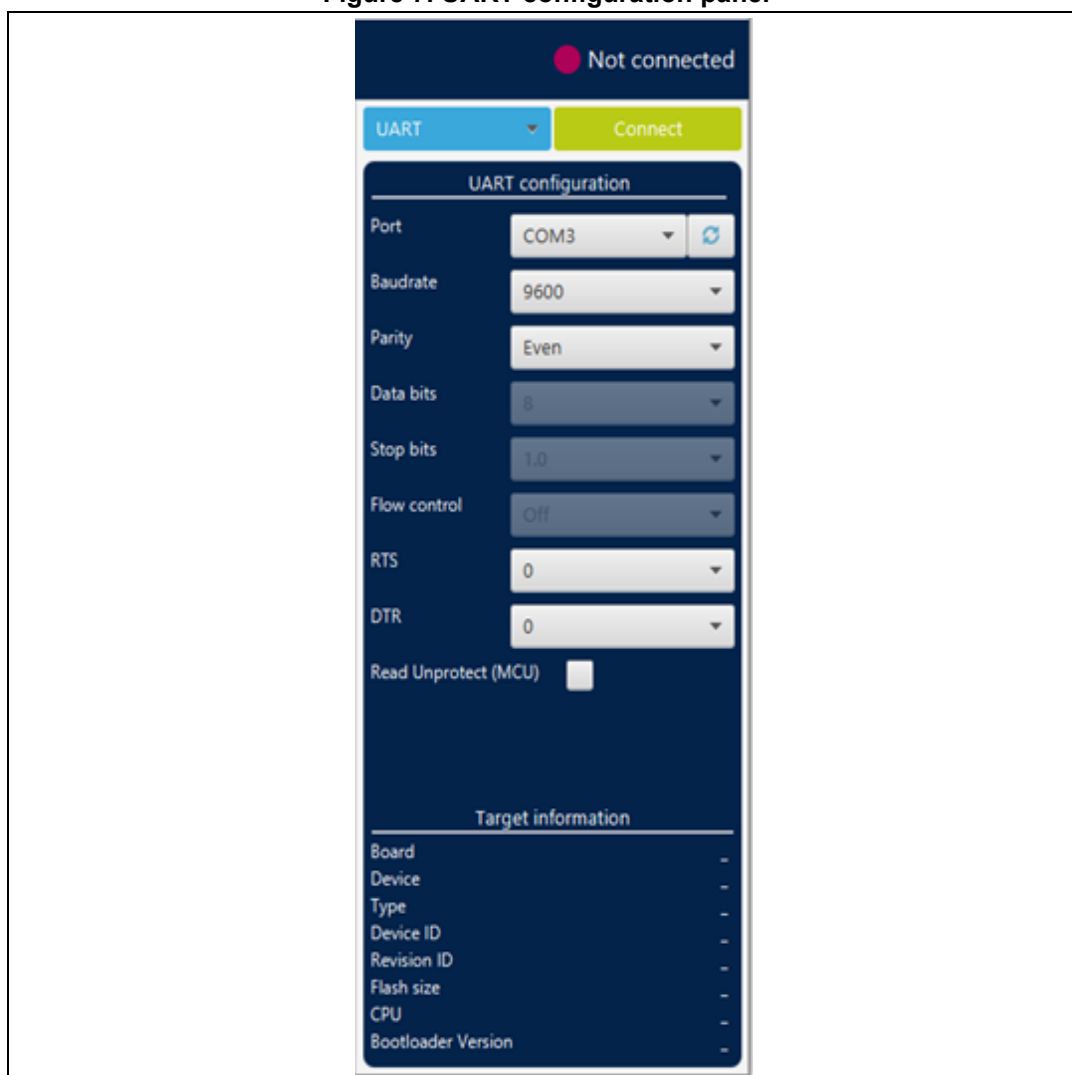
- **Serial number:** This field contains the serial numbers of all connected ST-LINK probes. The user can choose one of them, based on its serial number.
- **Port:** ST-LINK probe supports two debug protocols, JTAG and SWD.

Note: JTAG is not available on all embedded ST-LINK in the STM32 Nucleo or Discovery boards.

- **Frequency:** The JTAG or SWD clock frequency
- **Access port:** Selects the access port to connect to. Most of the STM32 devices have only one access port, which is Access port 0.
- **Mode:**
 - **Normal:** With 'Normal' connection mode, the target is reset then halted. The type of reset is selected using the 'Reset Mode' option.
 - **Connect Under Reset:** This mode enables connection to the target using a reset vector catch before executing any instructions. This is useful in many cases, for example when the target contains a code that disables the JTAG/SWD pins.
 - **Hot plug:** Enables connection to the target without a halt or reset. This is useful for updating the RAM addresses or the IP registers while the application is running.
- **Reset mode:**
 - **Software system reset:** Resets all STM32 components except the Debug via the Cortex-M application interrupt and reset control register (AIRCR).
 - **Hardware reset:** Resets the STM32 device via the nRST pin. The RESET pin of the JTAG connector (pin 15) must be connected to the device reset pin.
 - **Core reset:** Resets only the core Cortex-M via the AIRCR.
- **Speed (Cortex- M33 only):**
 - **Reliable:** allows the user to connect with a slow mode.
 - **Fast:** allows the user to connect with a fast mode.
- **Shared:** Enables shared mode allowing connection of two or more instances of STM32CubeProgrammer or other debugger to the same ST-LINK probe.
- **Debug in Low Power mode (STM32U5/WB/L4 Series only):** Set the bits in DBGMCU_CR to 1.
- **External loader:** Displays the name of the external memory loader selected in the "External loaders" panel accessible from the main menu (Hamburger menu).
- **Target voltage:** The target voltage is measured and displayed here.
- **Firmware version:** Displays the ST-LINK firmware version. The Firmware upgrade button allows you to upgrade the ST-LINK firmware.

UART settings

Figure 7. UART configuration panel



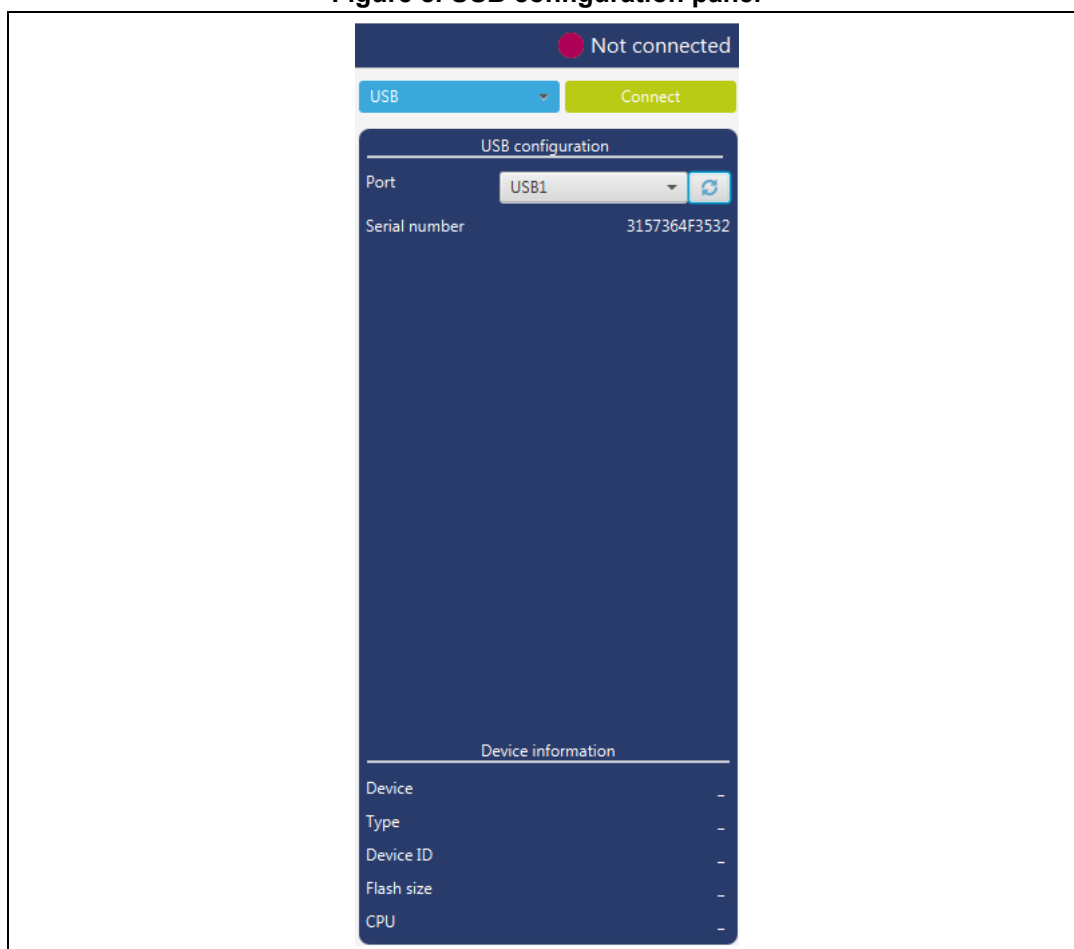
- **Port:** Selects the com port to which the target STM32 is connected. Use the refresh button to recheck the available com port on the PC.

Note: The STM32 must boot in bootloader mode using boot pins and/or the option bits. Check “STM32 microcontroller system memory boot mode” (AN2606), available on www.st.com, for more information on the STM32 bootloader.

- **Baudrate:** Selects the UART baud rate.
- **Parity:** Selects the parity (even, odd, none). Must be ‘even’ for all STM32 devices.
- **Data bits:** Must be always 8. Only 8-bit data is supported by the STM32.
- **Stop bits:** Must be always 1. Only 1-bit stop bit is supported by the STM32.
- **Flow control:** Must be always off
- **RTS** (Request To Send): Sets the COM RTS pin to either high or low level.
- **DTR** (Data Terminal Ready): Sets the COM DTR pin to either high or low level.

USB settings

Figure 8. USB configuration panel



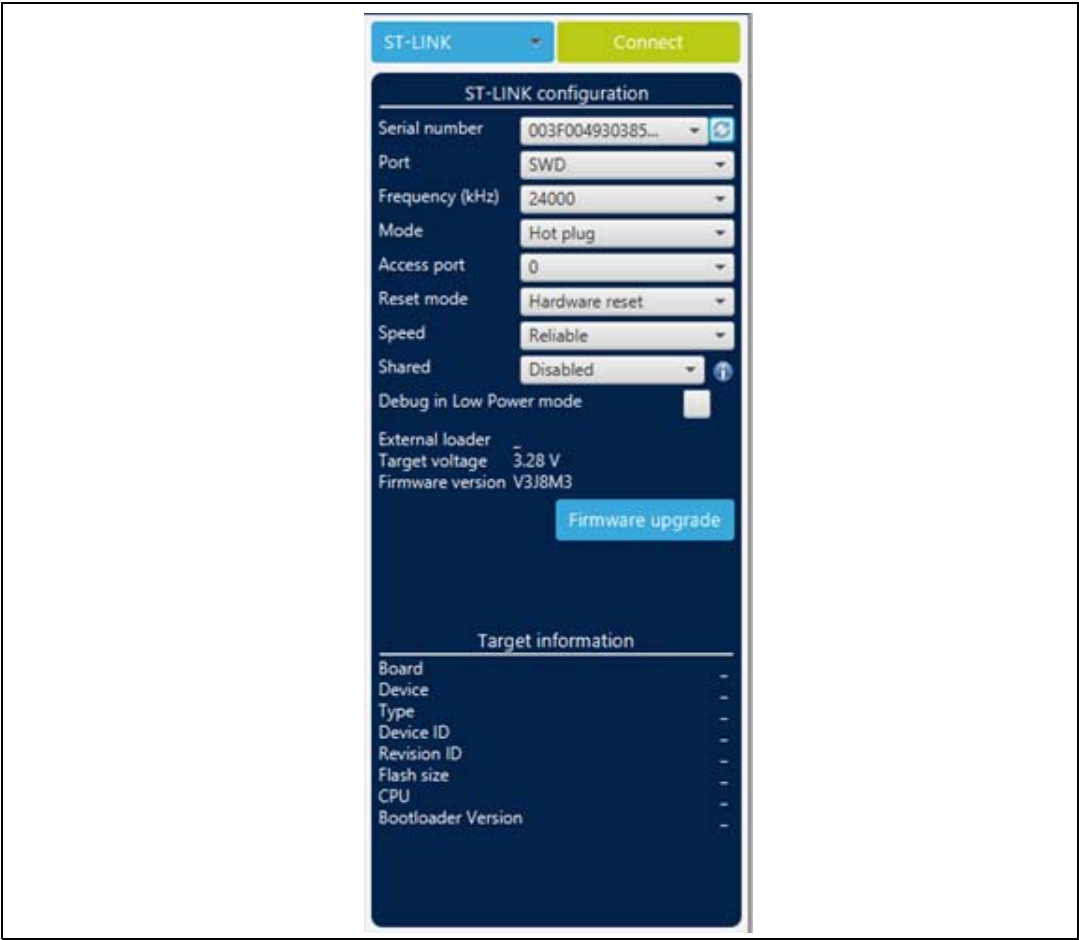
- **Port:** Selects the USB devices in DFU mode connected to the PC. You can use the refresh button to recheck the available devices.

Note: The STM32 must boot in bootloader mode using boot pins and/or the option bits. Check AN2606, available on www.st.com, for more information on the STM32 bootloader.

Once the correct interface settings are set, click on the 'Connect' button to connect to the target interface. If the connection succeeds, it is shown in the indicator above the button, which turns to green.

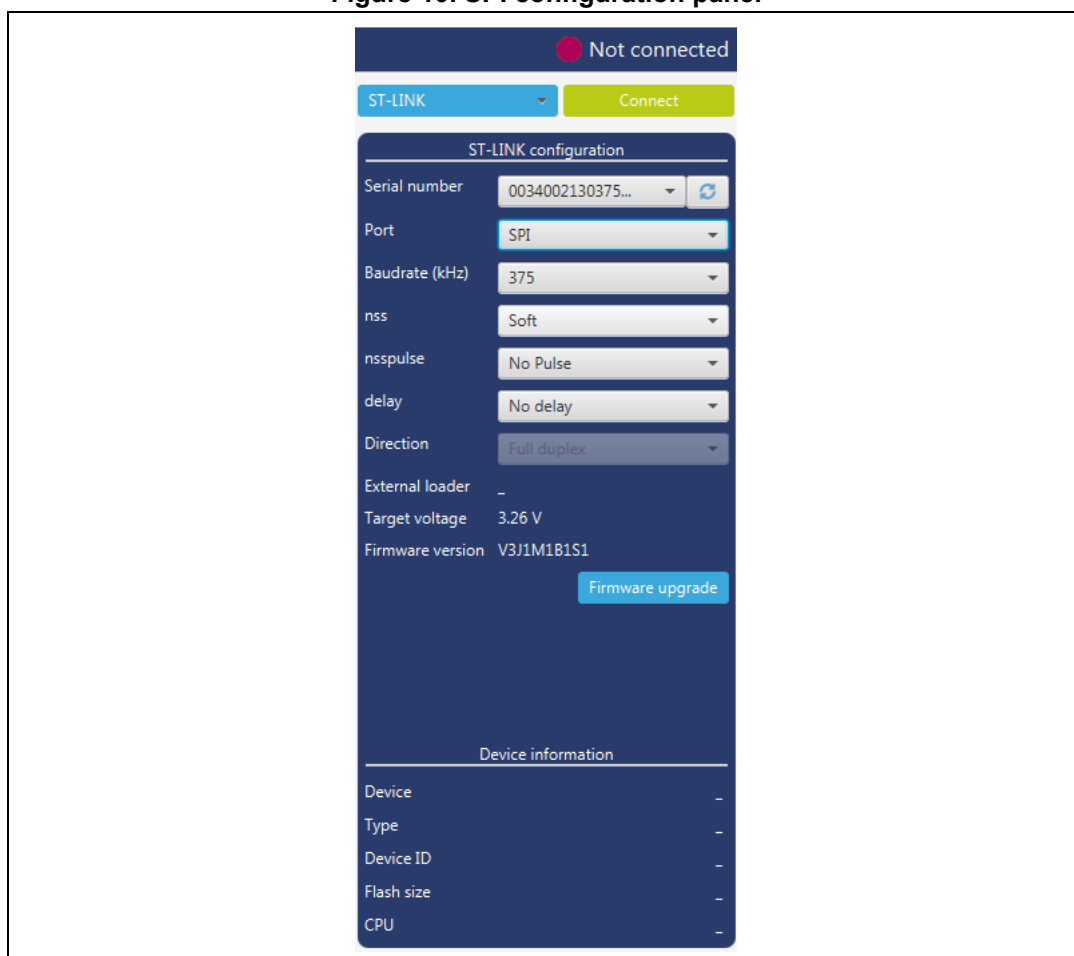
Once connected, the target information is displayed in the device information section below the settings section, which is then disabled as in [Figure 9](#).

Figure 9. Target information panel



SPI settings

Figure 10. SPI configuration panel



- **Serial number:** This field contains the serial numbers of all connected ST-LINK-V3 probes in case of use of SPI bootloader.
- **Port:** Selects the SPI devices connected to the PC. You can use the refresh button to recheck the available devices.
- **Baudrate:** Selects the SPI baud rate.
- **nss:** Slave Select software or hardware.
- **nsspulse:** the Slave Selection signal can operate in a pulse mode where the master generates pulses on nss output signal between data frames for a duration of one SPI clock period when there is a continuous transfer period.
- **Delay:** used to insert a delay of several microseconds between data.
- **Direction:** Must be always Full-duplex, both data lines are used and synchronous data flows in both directions.

CAN settings

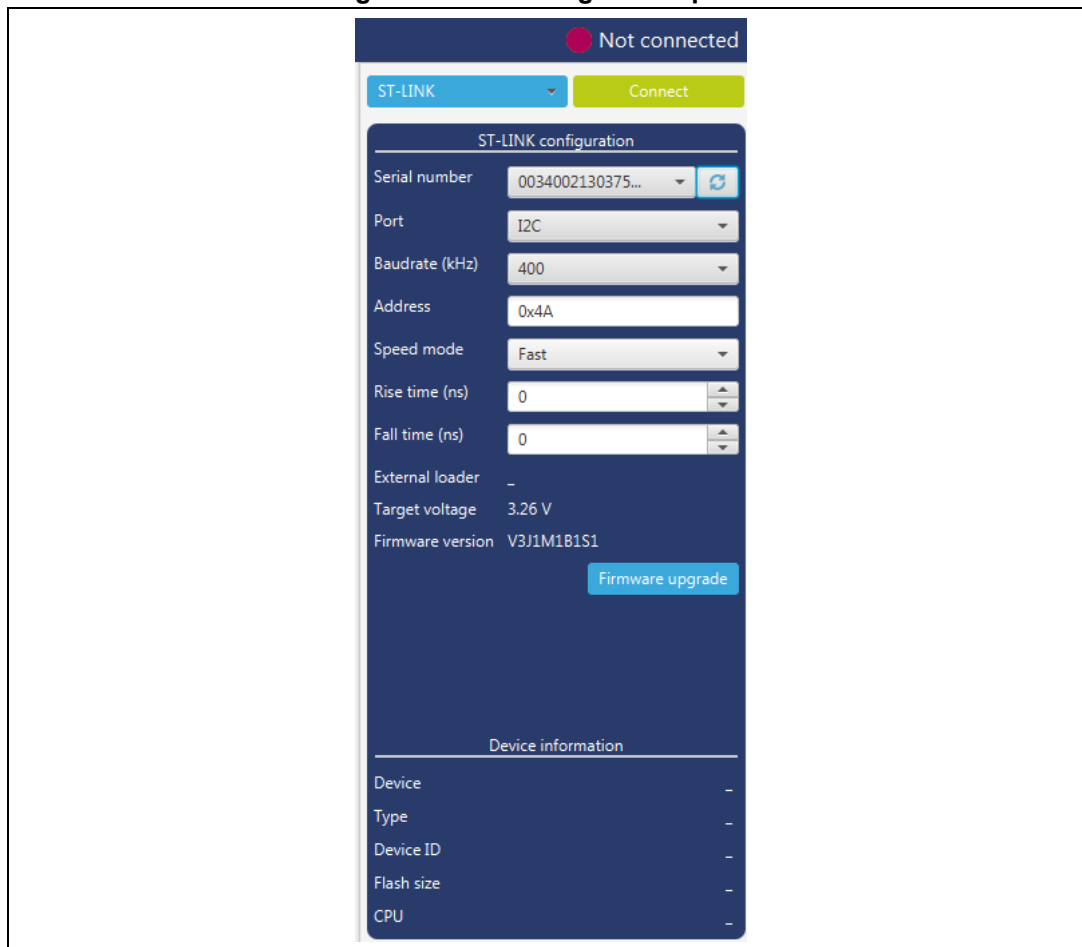
Figure 11. CAN configuration panel



- **Serial number:** This field contains the serial numbers of all connected ST-LINK-V3 probes in case to use CAN bootloader.
- **Port:** Selects the CAN devices connected to the PC. You can use the refresh button to recheck the available devices.
- **Baudrate:** Selects the CAN baud rate.
- **Assigned FIFO:** Selects the receive FIFO memory to store incoming messages.
- **Filter mode:** Selects the type of the filter, MASK or LIST.
- **Filter scale:** Selects the width of the filter bank, 16 or 32 bits.
- **Filter bank:** Values between 0 and 13, to choose the filter bank number.

I2C settings

Figure 12. I2C configuration panel



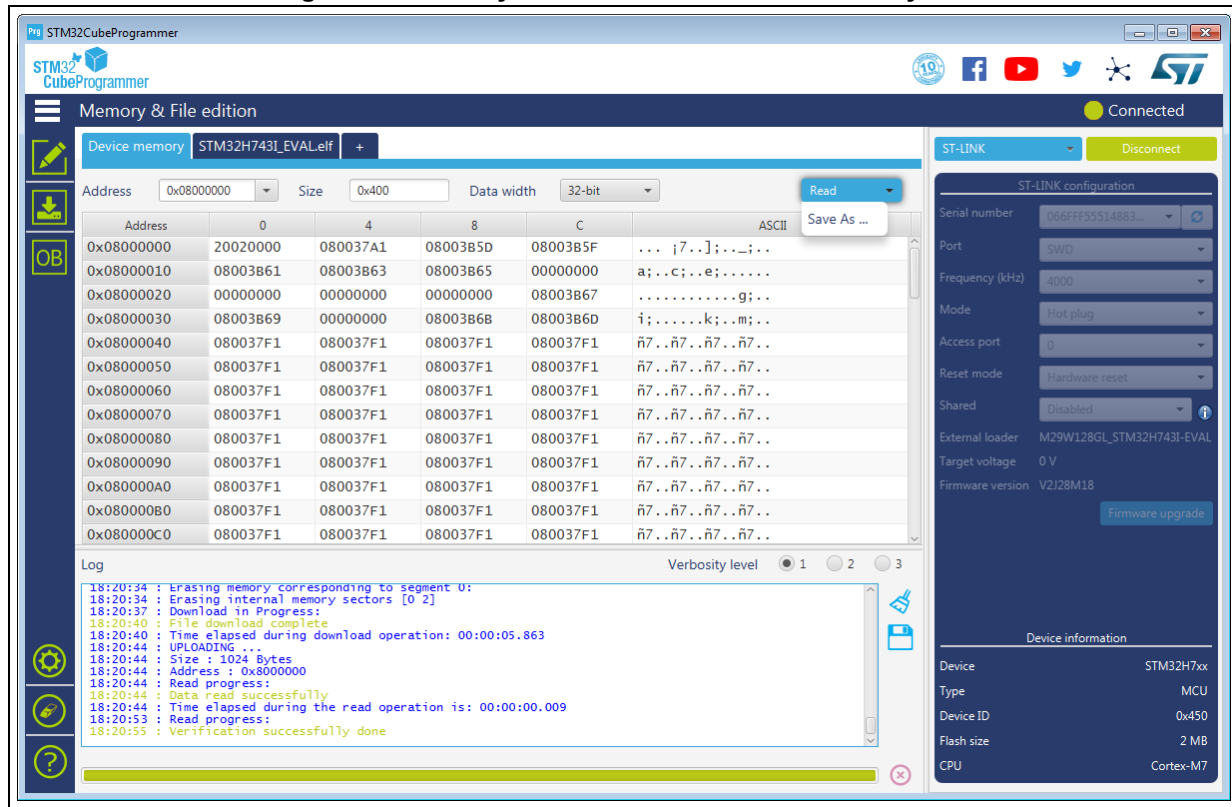
- **Serial number:** This field contains the serial numbers of all connected ST-LINK-V3 probes in case to use I2C bootloader.
- **Port:** Selects the I2C devices connected to the PC. You can use the refresh button to recheck the available devices.
- **Baudrate:** Selects the I2C baud rate.
- **Address:** Adds the address of the slave bootloader in hex format.
- **Speed mode:** Selects the speed mode of the transmission Standard or Fast.
- **Rise time:** Chooses values according to Speed mode, 0-1000 (STANDARD), 0-300 (FAST).
- **Fall time:** Chooses values according to Speed mode, 0-300 (STANDARD), 0-300 (FAST).

2.2 Memory and file edition

The Memory and file edition panel allows the user to read and display target memory and file contents.

2.2.1 Reading and displaying target memory

Figure 13. Memory and file edition: Device memory tab

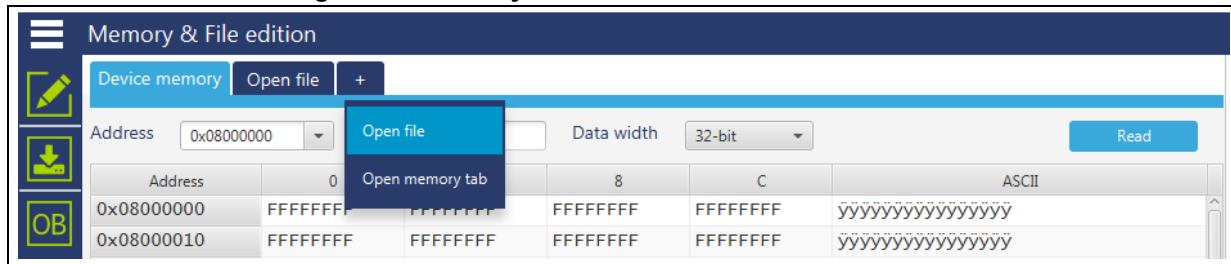


After target connection, you can read the STM32 target memory using this panel. To do this, specify the address and the size of the data to be read, then click on the Read button in the top-left corner. Data can be displayed in different formats (8-, 16- and 32-bit) using the 'Data width' combo box.

You can also save the device memory content in .bin, .hex or .srec file using the "Save As..." menu from the tab contextual menu or the action button.

You can open multiple device memory tabs to display different locations of the target memory. To do this, just click on the "+" tab to display a contextual menu that allows you to add a new "Device memory" tab, or to open a file and display it in a "File" tab:

Figure 14. Memory and file edition: Contextual menu



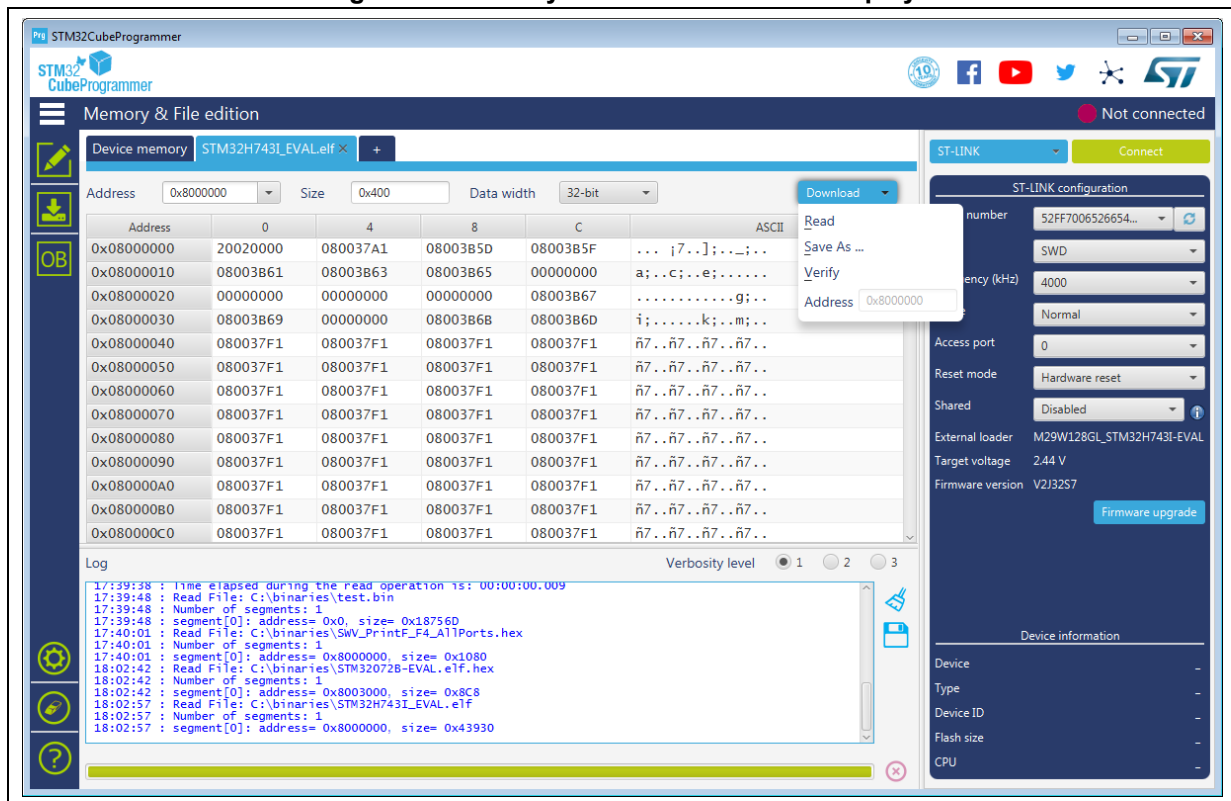
2.2.2 Reading and displaying a file

To open and display a file, just click on the “+” and select ‘Open File’ menu, as illustrated in [Figure 14](#).

The file formats supported are binary files (.bin), ELF files (.elf, .axf, .out), Intel hex files (.hex) and Motorola S-record files (.Srec).

Once the file is opened and parsed, it is displayed in a dedicated tab with its name, as illustrated in [Figure 15](#). The file size is displayed in the ‘Size’ field, and the start address of hex, srec or ELF files, is displayed in the ‘Address’ field, for a binary file it is 0.

Figure 15. Memory and file edition: File display



The address field can be modified to display the file content starting from an offset. Using the tab contextual menu or the action button, you can download the file using “Download” button/menu. For a binary file you need to specify the download address in the “Address” menu. The user can verify if the file is already downloaded using the “Verify” menu, and also save it in another format (.bin, .hex or .srec).

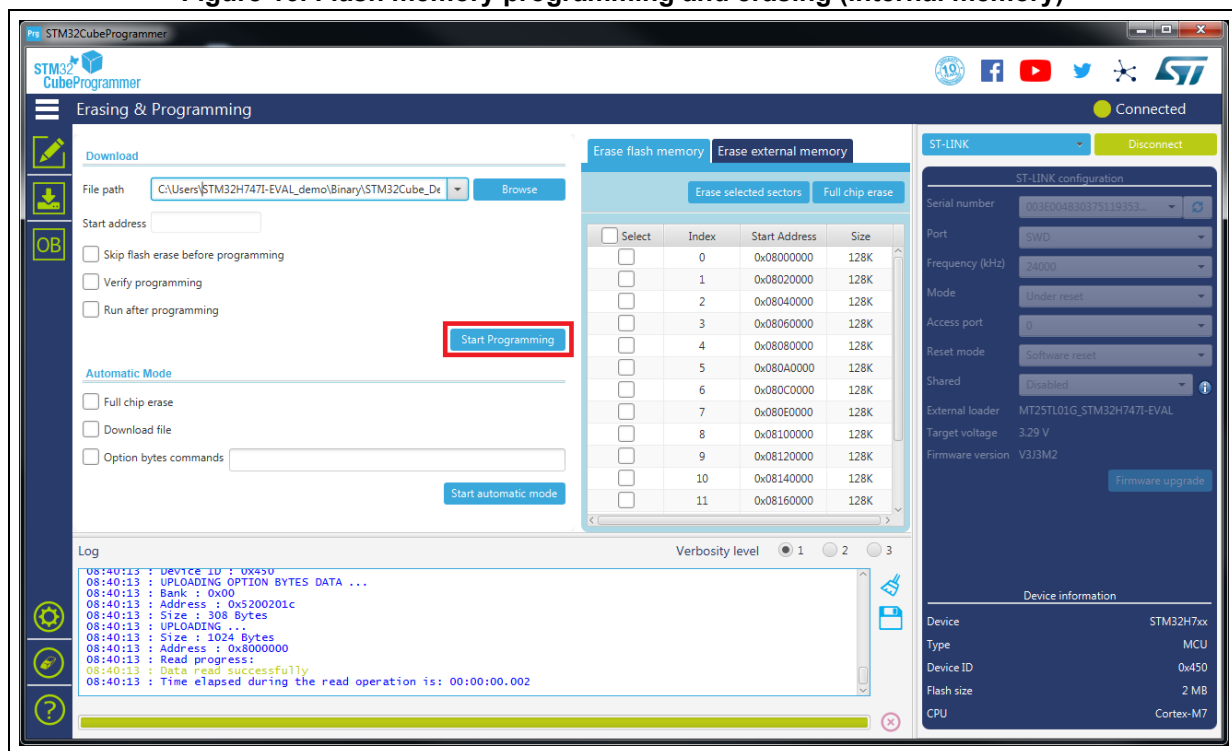
As for the ‘Device memory’ tab, user can display the file memory content in different formats (8-, 16- and 32-bit) using the ‘Data width’ combo box.

2.3 Memory programming and erasing

This panel is dedicated to Flash memory programming and erasing operations.

2.3.1 Internal Flash memory programming

Figure 16. Flash memory programming and erasing (internal memory)



Memory erasing

Once connected to a target, the memory sectors are displayed in the right-hand panel showing the start address and the size of each sector. To erase one or more sectors, select them in the first column and then click on the “Erase selected sectors” button.

The ‘Full chip erase’ button erases the whole Flash memory.

Memory programming

To program a memory execute the following steps:

1. Click on the browse button and select the file to be programmed. The file format supported are binary files (.bin), ELF files (.elf, .axf, .out), Intel hex files (.hex) and Motorola S-record files (.Srec).
2. In case of programming a binary file, the address must be set.
3. Select the programming options:
 - Verify after programming: read back the programmed memory and compare it byte per byte with the file.
 - Skip Flash erase before programming: if checked, the memory is not erased before programming. This option must be checked only when you are sure that the target memory is already erased.
 - Run after programming: start the application just after programming.
4. Click on the 'Start programming' button to start programming.

The progress bar on the bottom of the window shows the progress of the erase and programming operations.

2.3.2 External Flash memory programming

To program an external memory connected to the microcontroller via any of the available interfaces (e.g. SPI, FMC, FSMC, QSPI, OCTOSPI) you need an external loader.

STM32CubeProgrammer is delivered with external loaders for most available STM32 Evaluation and Discovery boards available under the "bin/ExternalLoader" directory. If you need to create a new external loader, see [Section 2.3.3](#) for more details.

To program an external memory, select one or more external loaders from the "ExternalLoader" panel to be used by the tool to read, program, or erase external memories as shown in [Figure 17](#). Once selected, the external loader(s) is (are) used for any memory operation in its (their) memory range.

The "External flash erasing" tab on the right of the "Erasing and Programming" panel displays the memory sectors for each selected loader, and enables sector or full-chip erase, as shown in [Figure 18](#).

Figure 17. Flash memory programming (external memory)

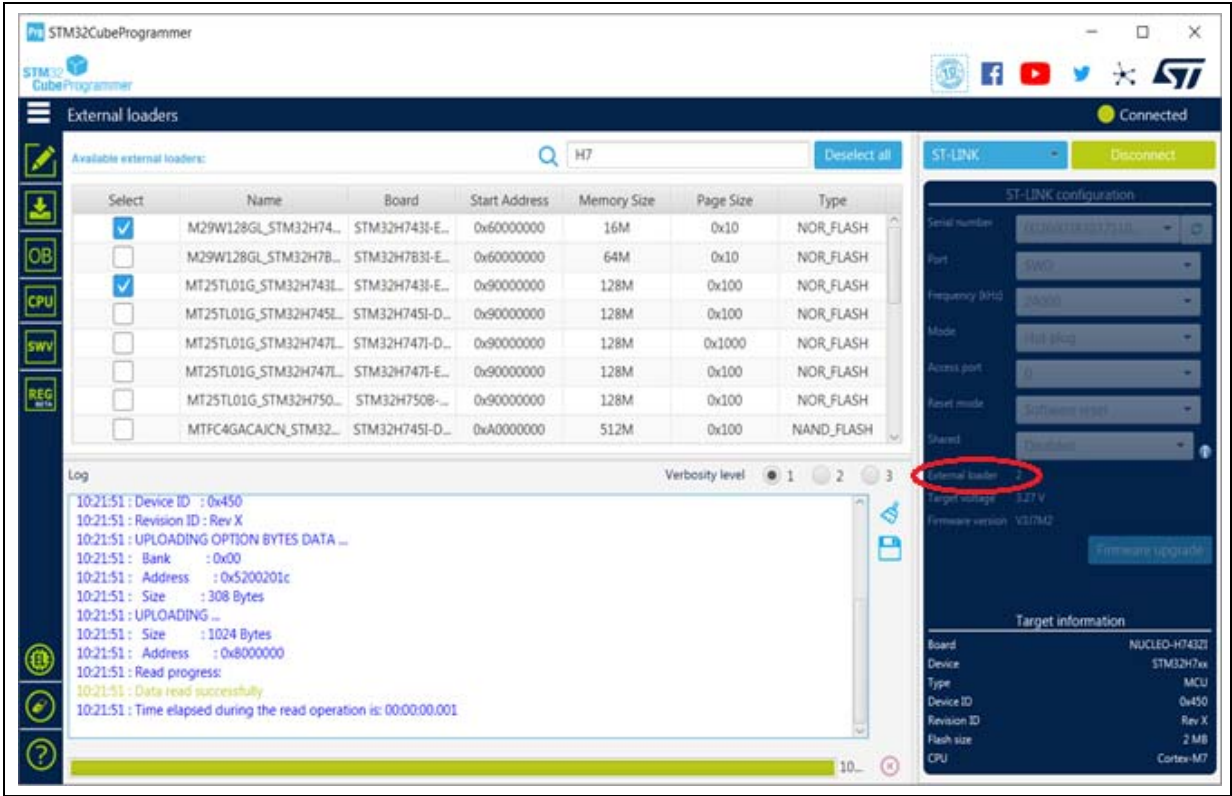
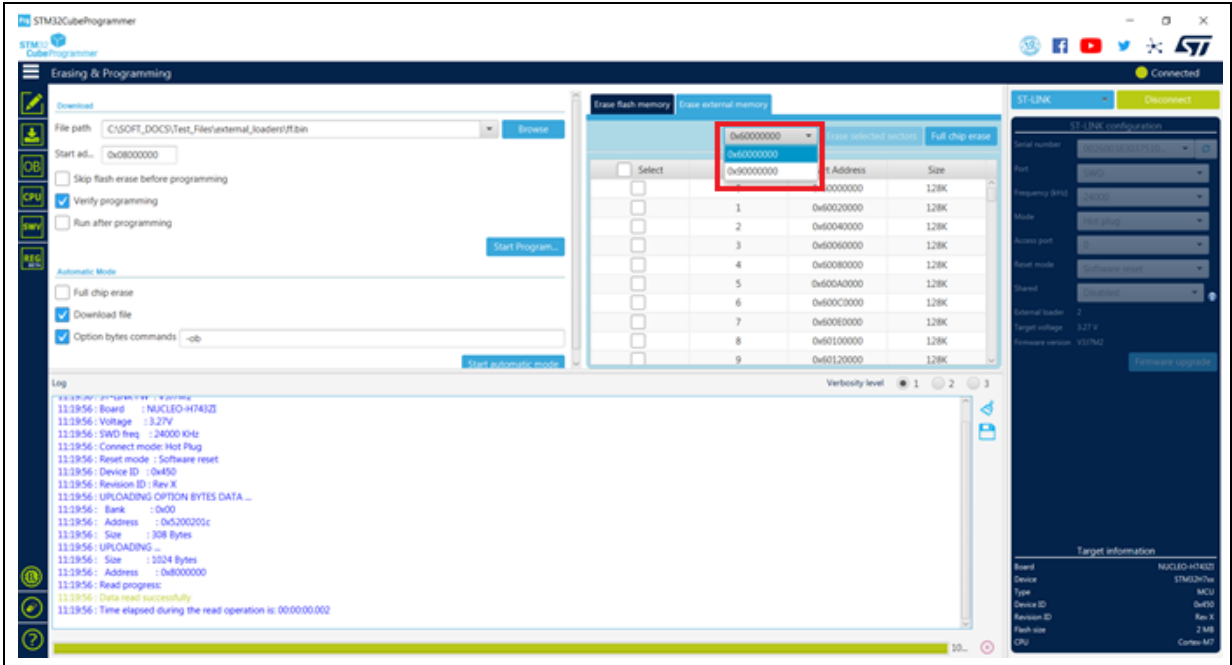


Figure 18. Flash memory erasing (external memory)



2.3.3 Developing customized loaders for external memory

Based on the examples available under the “*bin/ExternalLoader*” directory, users can develop their custom loaders for a given external memory. These examples are available for three toolchains: Keil® MDK, EWARM and TrueSTUDIO®. The development of custom loaders can be performed using one of these toolchains, keeping the same compiler/linker configurations, as in the examples.

The external Flash programming mechanism is the same used by the STM32 ST-LINK utility tool. Any Flash loader developed to be used with the ST-LINK utility is compatible with the STM32CubeProgrammer tool, and can be used without any modification.

To create a new external memory loader, follow the steps below:

1. Update the device information in *StorageInfo* structure in the *Dev_Inf.c* file with the correct information concerning the external memory.
2. Rewrite the corresponding functions code in the *Loader_Src.c* file.
3. Change the output file name.

Note: Some functions are mandatory and cannot be omitted (see the functions description in the *Loader_Src.c* file).

Linker or scatter files must not be modified.

After building the external loader project, an ELF file is generated. The extension of the ELF file depends upon the used toolchain (.axf for Keil, .out for EWARM and .elf for TrueSTUDIO or any gcc-based toolchain).

The extension of the ELF file must be changed to ‘.sldr’ and the file must be copied under the “*bin/ExternalLoader*” directory.

Loader_Src.c file

Developing an external loader for a memory, based on a specific IP requires the following functions:

- **Init** function
The **Init** function defines the used GPIO pins connecting the external memory to the device, and initializes the clock of the used IPs.
Returns 1 if success, and 0 if failure.

```
int Init (void)
```
- **Write** function
The **Write** function programs a buffer defined by an address in the RAM range.
Returns 1 if success, and 0 if failure.

```
int Write (uint32_t Address, uint32_t Size, uint8_t* buffer)
```
- **SectorErase** function

The **SectorErase** function erases the memory specified sectors.

Returns 1 if success, and 0 if failure.

```
int SectorErase (uint32_t StartAddress, uint32_t EndAddress)
```

Where “**StartAddress**” equals the address of the first sector to be erased and “**EndAddress**” equals the address of the end sector to be erased.

Note: This function is not used in case of an external SRAM loader.

It is imperative to define the functions mentioned above in an external loader. They are used by the tool to erase and program the external memory. For instance, if the user clicks on the program button from the external loader menu, the tool performs the following actions:

- Automatically calls the **Init** function to initialize the interface (QSPI, FMC ...) and the Flash memory
- Calls **SectorErase()** to erase the needed Flash memory sectors
- Calls the **Write()** function to program the memory

In addition to these functions, you can also define the functions below:

- **Read** function

The **Read** function is used to read a specific range of memory, and returns the reading in a buffer in the RAM.

Returns 1 if success, and 0 if failure.

```
int Read (uint32_t Address, uint32_t Size, uint16_t* buffer)
```

Where "**Address**" = start address of read operation, "**Size**" is the size of the read operation and "**buffer**" is the pointer to data read.

Note: For QSPI / OSPI (Quad-SPI / Octo-SPI) memories, the memory mapped mode can be defined in the Init function; in that case the Read function is useless since the data can be read directly from JTAG/SWD interface.

- **Verify** function

The **Verify** function is called when selecting the "verify while programming" mode. This function checks if the programmed memory corresponds to the buffer defined in the RAM. It returns an uint64 defined as follows:

```
Return value = ((checksum<<32) + AddressFirstError)
```

where "**AddressFirstError**" is the address of the first mismatch, and "**checksum**" is the checksum value of the programmed buffer

```
uint64_t Verify (uint32_t FlashAddr, uint32_t RAMBufferAddr,
uint32_t Size)
```

- **MassErase** function

The **MassErase** function erases the full memory.

Returns 1 if success, and 0 if failure.

```
int MassErase (void)
```

- A Checksum function

All the functions described return 1 in case of a successful operation, and 0 in case of a fail.

Dev_Inf.c file

The StorageInfo structure defined in this file provides information on the external memory. An example of the type of information that this structure defines is given below:

```
#if defined (__ICCARM__)
    __root struct StorageInfo const StorageInfo = {
#else
    struct StorageInfo const StorageInfo = {
#endif
    "External_Loader_Name", // Device Name + version number
    MCU_FLASH, // Device Type
    0x08000000, // Device Start Address
```

```

0x00100000, // Device Size in Bytes (1MBytes/8Mbits)
0x00004000, // Programming Page Size 16KBytes
0xFF, // Initial Content of Erased Memory
// Specify Size and Address of Sectors (view example below)
0x00000004, 0x00004000, // Sector Num : 4 ,Sector Size: 16KBytes
0x00000001, 0x00010000, // Sector Num : 1 ,Sector Size: 64KBytes
0x00000007, 0x00020000, // Sector Num : 7 ,Sector Size: 128KBytes
0x00000000, 0x00000000,
};

```

2.4 Option bytes

The option bytes panel allows the user to read and display target option bytes grouped by categories. The option bits are displayed in tables with three columns containing the bit(s) name, value and a description of the impact on the device.

The user can modify the values of these option bytes by updating the value fields, then clicking on the apply button, which programs and then verifies that the modified option bytes are correctly programmed. The user can click at any time on the read button, to read and refresh the displayed option bytes.

Figure 19. Option bytes panel

Name	Value	Description
RDP	AA	Read protection option byte. The read protection is used to protect the software code stored in Flash memory. AA : Level 0, no protection BB : Level 1, read protection of memories CC : Level 2, chip protection

Name	Value	Description
PROT_AREA_START1	0xff 0x8001fe0	Flash Bank 1 PCROP start address
PROT_AREA_END1	0x0 0x8000000	Flash Bank 1 PCROP End address. Deactivation of PCROP can be done by enabling DMEP1 bit and changing RDP from level 1 to level 0 while putting
DMEP1	☒	Unchecked : Flash Bank 1 PCROP zone is kept when RDP level regression (change from level 1 to 0) occurs Checked : Flash Bank 1 PCROP zone is erased when RDP level regression (change from level 1 to 0) occurs

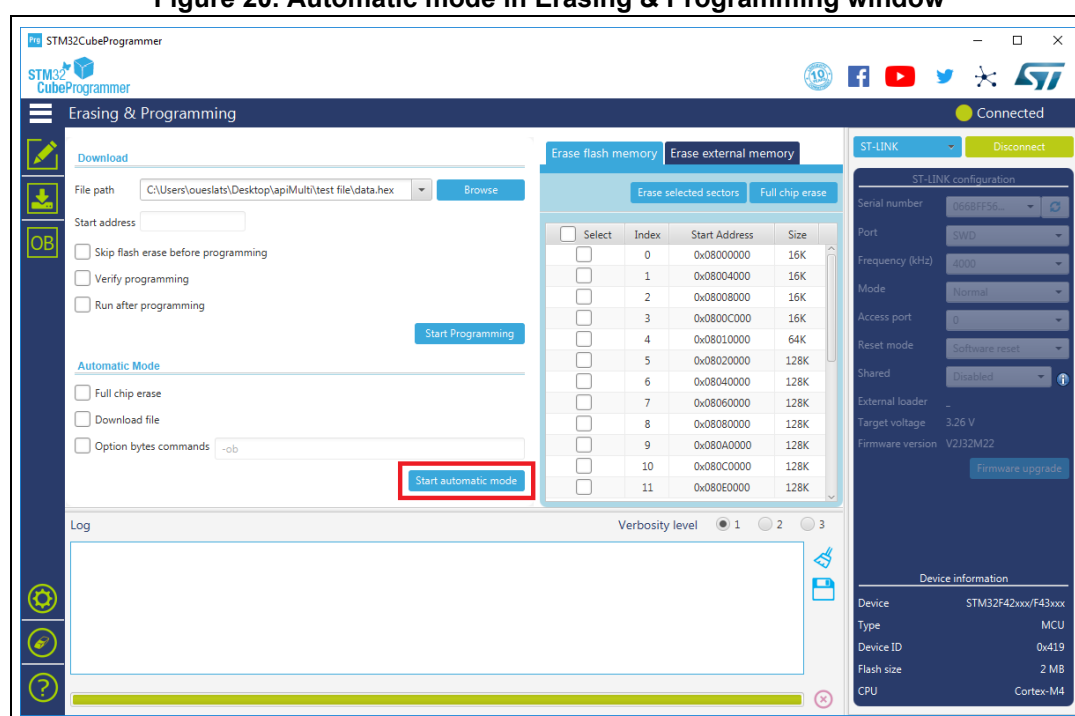
For more details refer to the option bytes section in the Flash memory programming manual and reference manual available from www.st.com.

2.5 Automatic mode

The Automatic mode feature shown in Erasing & Programming window (see [Figure 20](#)) allows the user to program and configure STM32 devices in loop. Allowed actions:

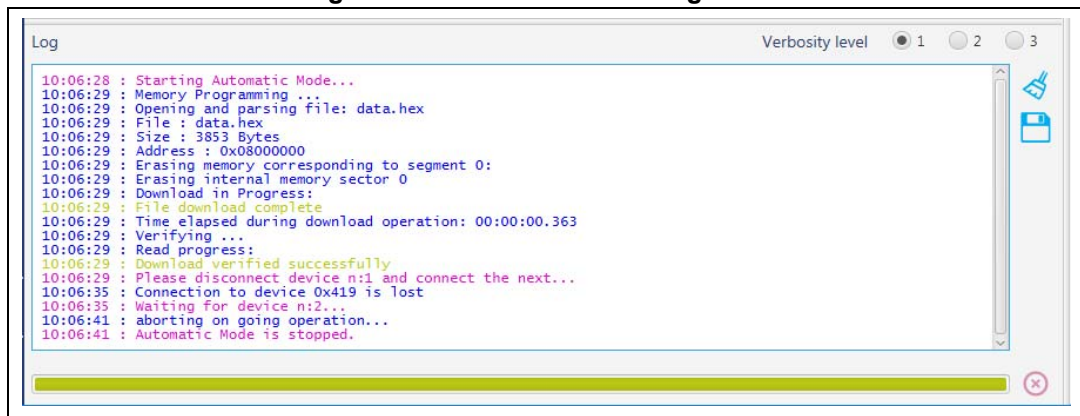
- Full chip erase: erase all the Flash memory
- Download file: activate and set programming options from Download section:
 - File path
 - Start address
 - Skip erase before programming
 - Verify programming
 - Run after programming
- Option bytes commands: configure the device by setting option bytes command line

Figure 20. Automatic mode in Erasing & Programming window



All automatic mode traces are indicated in the Log panel (see [Figure 21](#)) to show the process evolution and user intervention messages.

Figure 21. Automatic mode Log traces



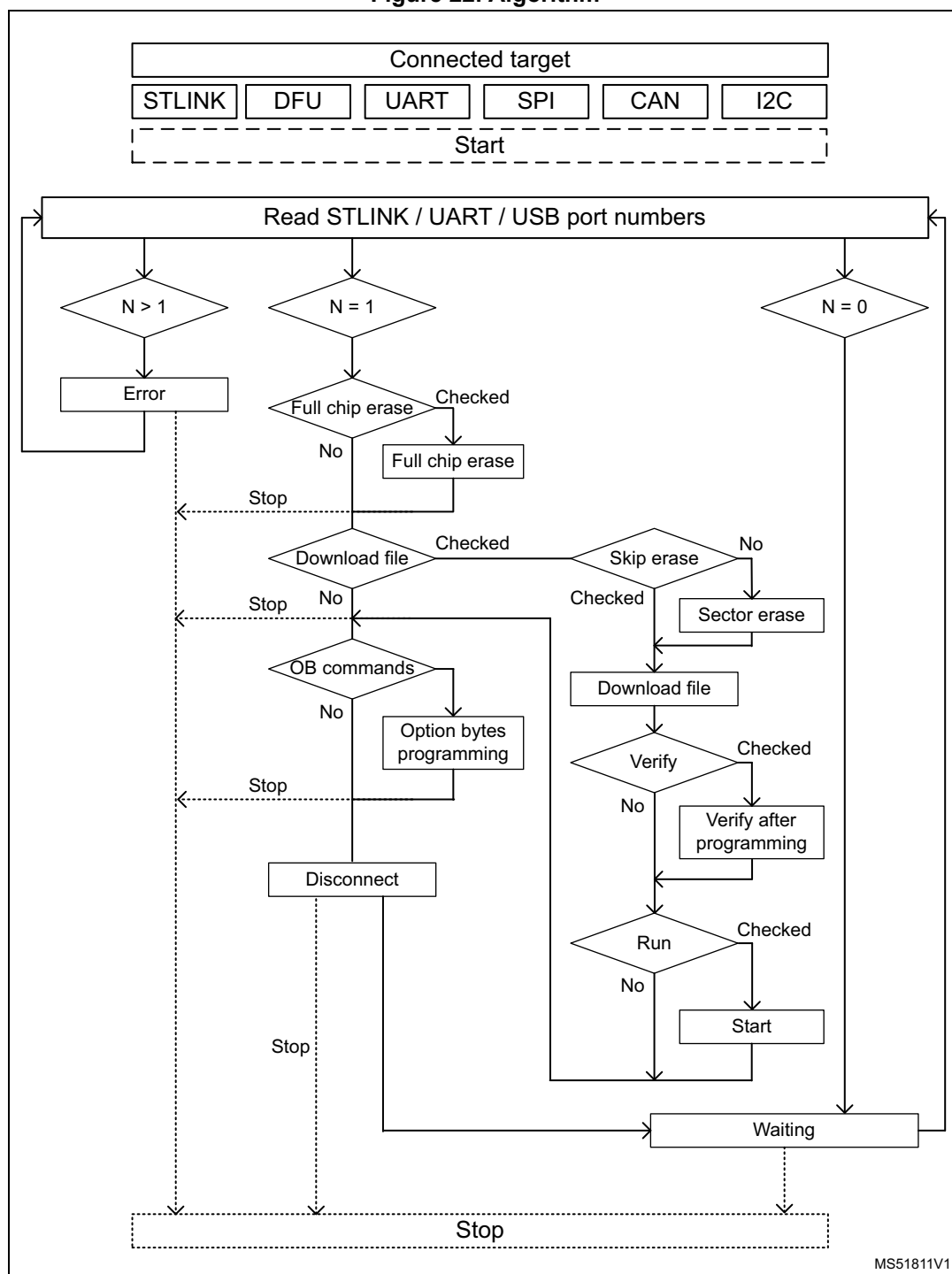
Graphical guide

- Connection to a first target must be established before performing automatic mode to collect connection parameters values associated to all next devices.
- If the Download file is checked, the system takes all Download file options in consideration, otherwise any Download option is performed.
- If the Option bytes commands is checked, the text field is activated, then the user can insert option bytes commands (like CLI commands), and make sure that there are no white spaces at the beginning:
-ob [OptionByte=value] [OptionByte=value] [OptionByte=value] ...
- Example of Option bytes command: **"-ob BOR_LEV=0 nBOOT0=1"**
- If the Start automatic mode button is pressed, the system enters in a loop, until a system stop is called.
- While the automatic mode is in execution state, all graphical objects are disabled.
- The user can stop the process at any preferred time by pressing cancel button or stop automatic mode button.

Log messages

- “Starting Automatic Mode...”
Indicates that the system entered successfully in automatic process.
- “More than one ST-LINK probe detected! Keep only one ST-LINK probe!”
The automatic mode cannot be used if more than one ST-LINK probe is connected to the computer when using JTAG/SWD interfaces. A message is displayed to prevent the user and ask him to keep only one ST-LINK probe connected to continue using this mode.
- “More than one ST-LINK Bridge detected! Keep only one ST-LINK Bridge!”
The automatic mode cannot be used if more than one ST-LINK bridge is connected to the computer when using bootloader interface SPI/CAN/I²C interfaces. A message is displayed to prevent the user and ask him to keep only one ST-LINK bridge connected to continue using this mode.
- “More than one ST-LINK USB DFU detected! Keep only one USB DFU!”
The automatic mode cannot be used if more than one USB DFU is connected to the computer when using USB bootloader interface. A message is displayed to prevent the user and ask him to keep only one USB DFU connected to continue using this mode.
- “More UART ports detected than last connection!”
In the first connection time the automatic mode calculates the number of the available Serial ports and put it as a reference to detect correctly that we use only one port UART for STM32 device.
- “Please disconnect device and connect the next...”
If the system finishes the first process, and whatever the result, disconnect the current device to prepare the second device connection.
- “Waiting for device...”
Once the connection to the previous device is correctly lost, the system keeps searching for a new device.
- “Automatic Mode is stopped.”
Indicates that there is a required cancel and the system stops the process.

Figure 22. Algorithm



MS51811V1

2.6 In application programming (IAP/USBx)

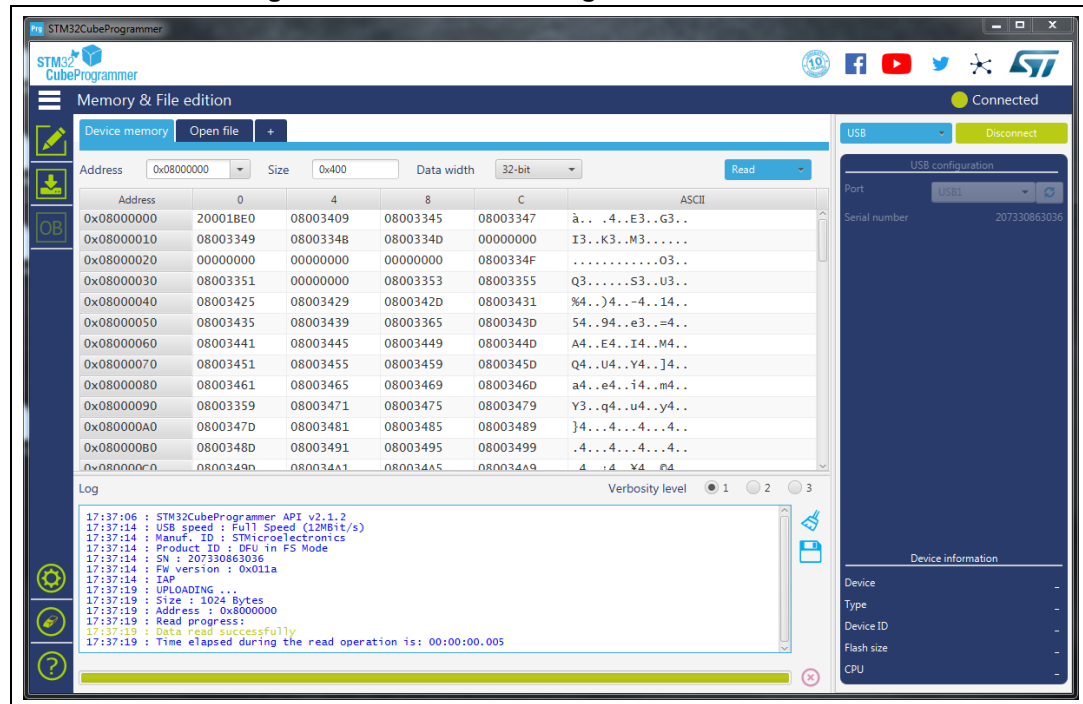
STM32CubeProgrammer supports IAP/ USBx only with USB DFU connection mode. When USB connection is chosen and the boot is from Flash memory, STM32CubeProgrammer

detects the IAP/ USBx like DFU bootloader and after connection an IAP/ USBx message appears in the log panel.

Note: *Option byte and sector erase are not available with IAP/USBx.*

Sample IAPs/ USBx are available in CubeFW/ CubeAzure on www.st.com.

Figure 23. STM32Cube Programmer in IAP mode



2.7 Flash the co-processor binary using graphical interface

2.7.1 FUS / Stack upgrade

1. Use STM32CubeProgrammer (version 2.4 or higher), see [Figure 24](#)
2. Access the SWD/ bootloader USB interface, see [Figure 25](#)
3. Delete the current wireless stack, see [Figure 26](#)
4. Upgrade the FUS version the same way you would download the stack when there is not an updated FUS version
5. Download the new FUS
6. Download the new wireless stack (pop-up must appear to ensure successful upgrade), see [Figure 27](#)

Note: *STM32CubeProgrammer (version 2.7 or higher) allows the user to install only new firmware (Stack v1.11.0 or higher). To install the old firmware, use STM32CubeProgrammer v2.6.0.*

Figure 24. STM32CubeProgrammer API SWD connection

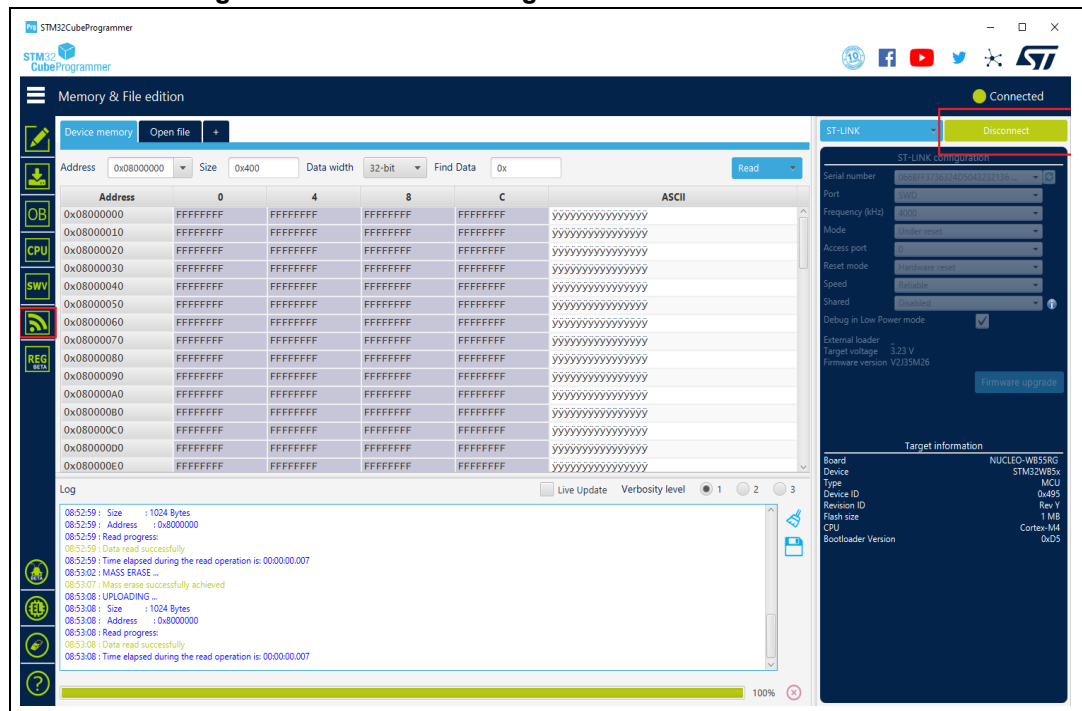


Figure 25. Steps for firmware upgrade

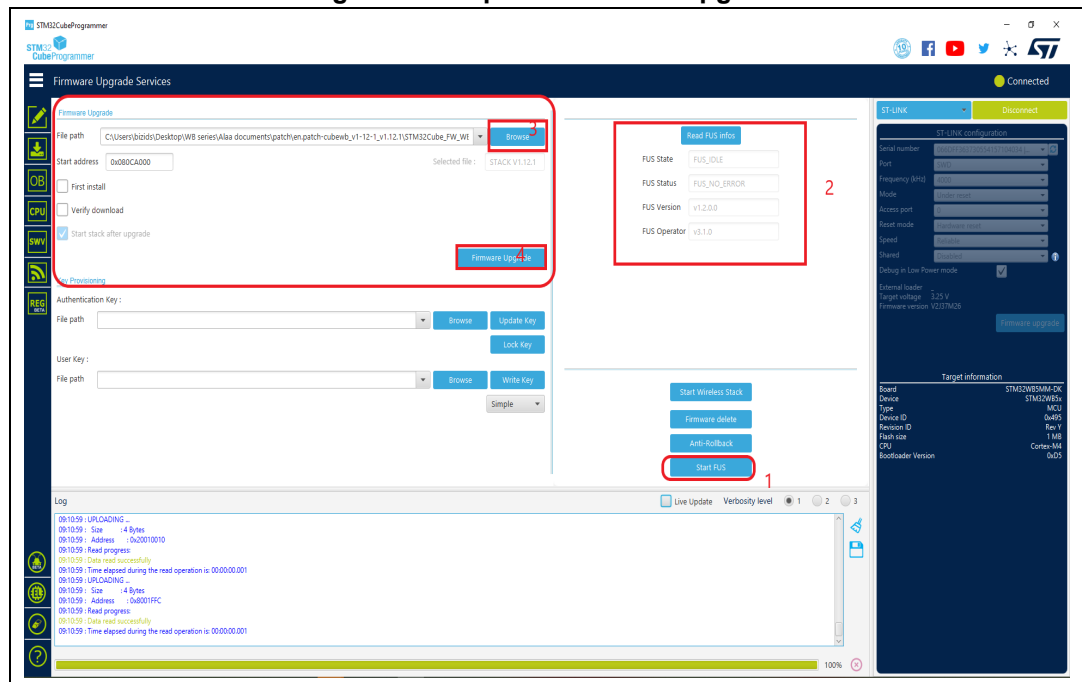


Figure 26. Pop-up confirming successful firmware delete

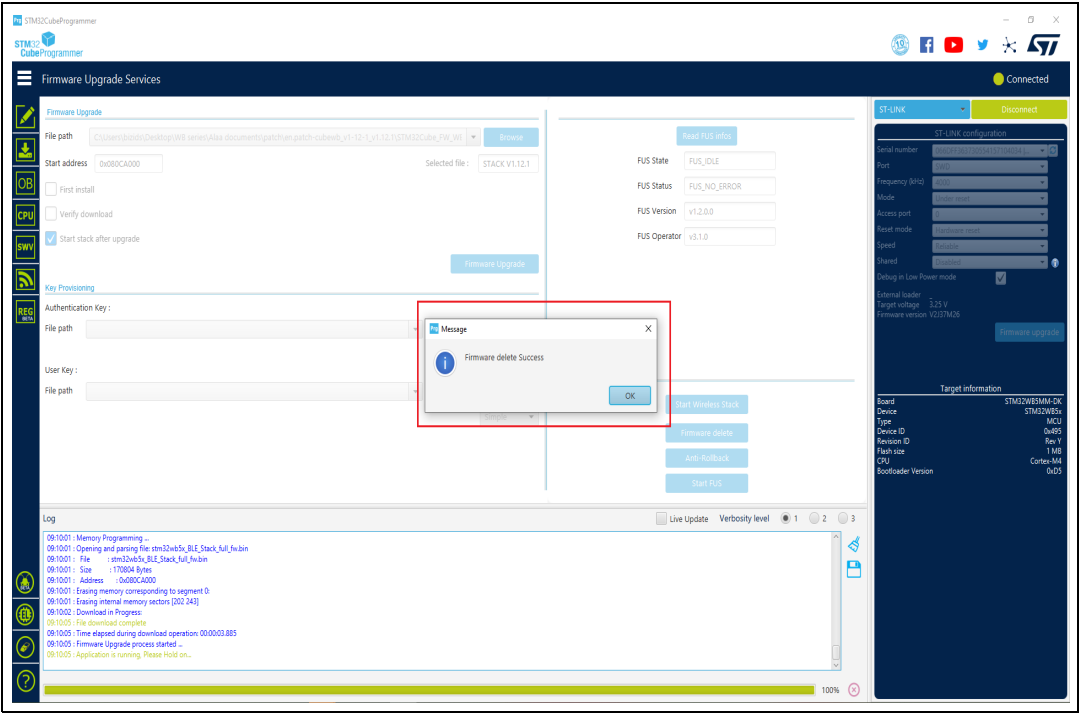
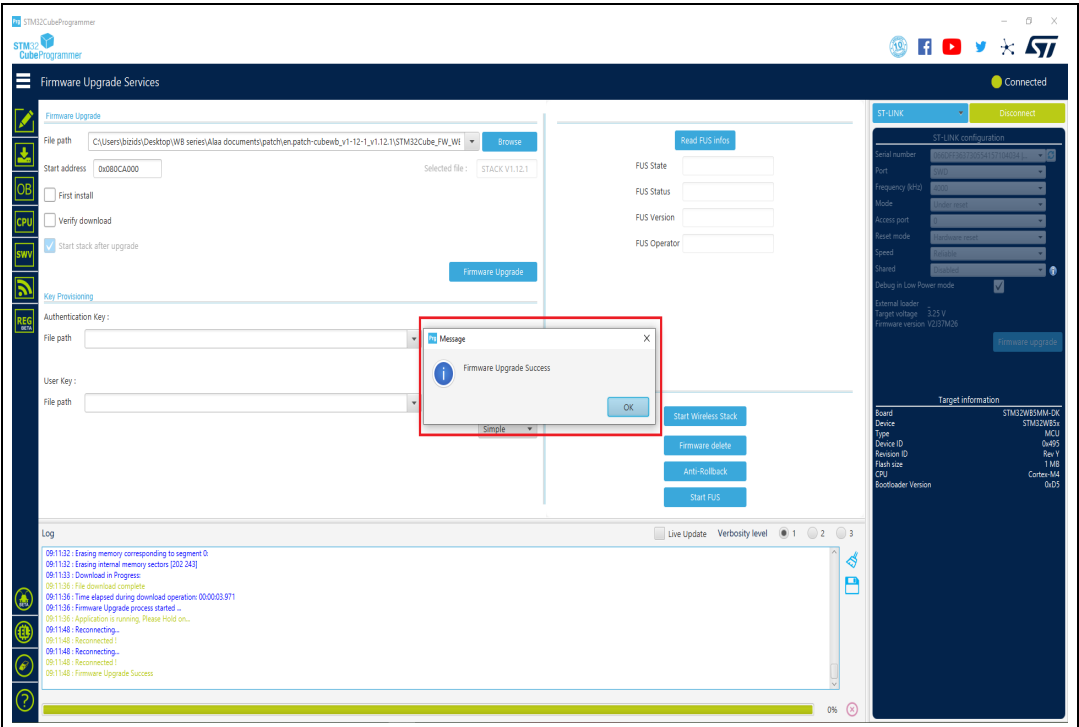


Figure 27. Pop-up confirming successful firmware upgrade



2.7.2 Key provisioning

STM32CubeProgrammer allows the user to add a customized signature (encrypted and signed by STMicroelectronics) to any image.

User authentication

FUS window allows a user authentication key to be stored through the update key button (Figure 28).

Once the user authentication key is installed, it can be changed unless lock user authentication key button is selected (see Figure 29). Once the authentication key is installed, the install or upgrade services must be done with the double signed FUS/ Stack, or it will be rejected.

Figure 28. Update authentication key

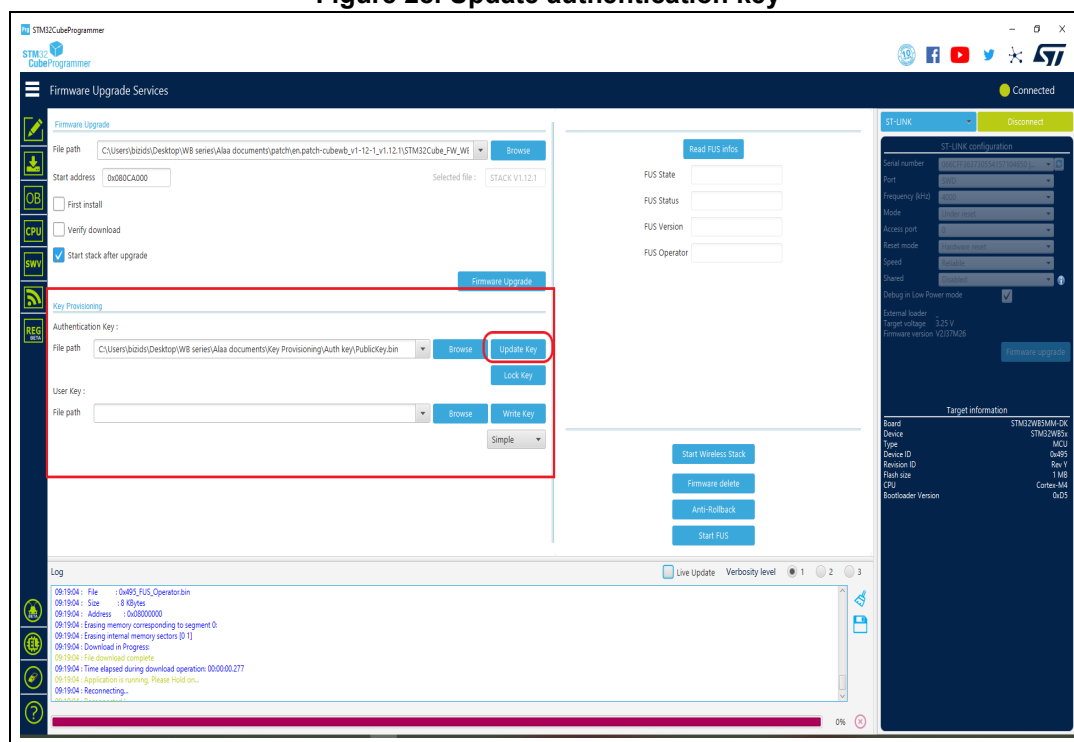
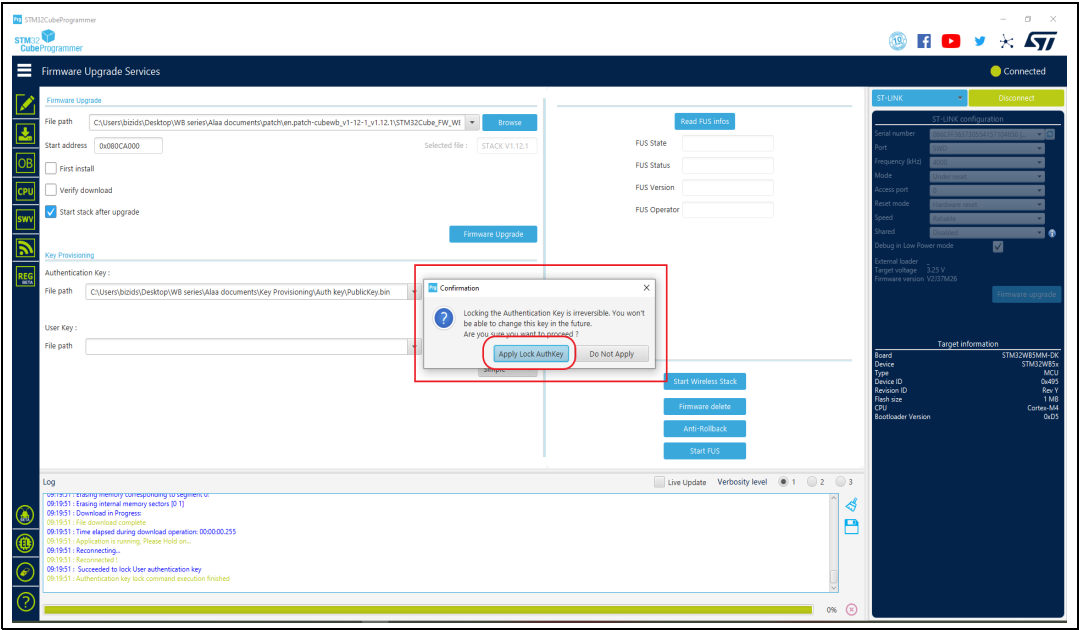


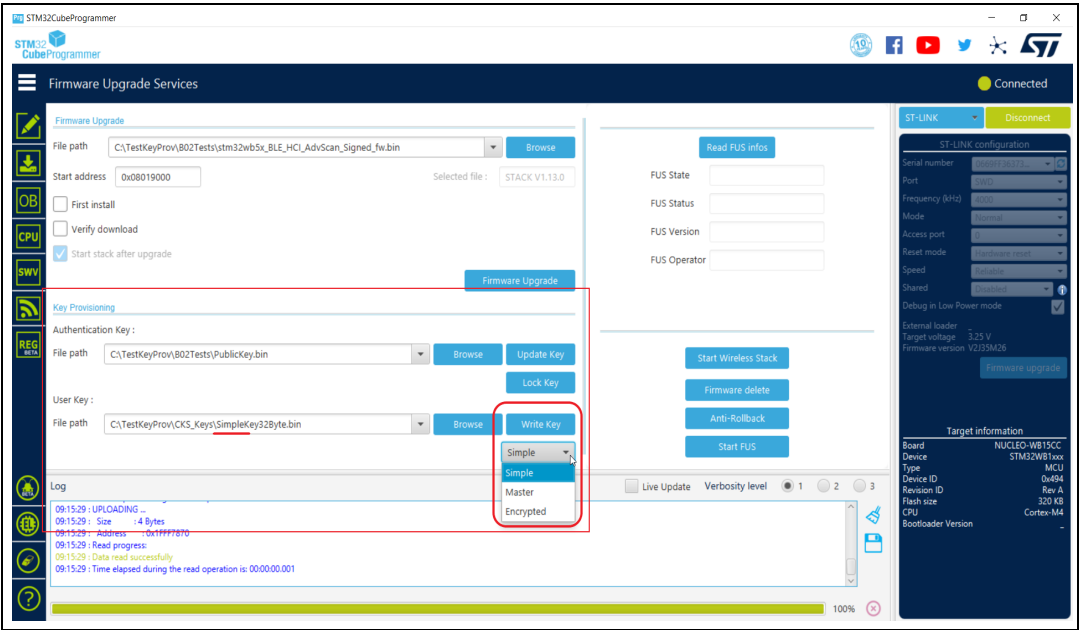
Figure 29. Pop-up requesting to lock authentication key



Customer key storage

STM32CubeProgrammer allows customer keys to be stored in the dedicated FUS Flash memory area in binary format (user key types: simple, master or encrypted), see [Figure 30](#).

Figure 30. Store customer key



For complete documentation on STM32WBxx products visit the dedicated pages on www.st.com.

2.8 Serial wire viewer (SWV)

The serial wire viewer window (see [Figure 31](#)) displays the printf data sent from the target through SWO. It displays useful information on the running firmware.

Note: The serial wire viewer is only available through SWD interface.

Before starting to receive SWO data, the user has to specify the exact target System clock frequency (in MHz) to allow the tool to correctly configure the ST-LINK and the target for the correct SWO frequency. The “Stimulus port” combo box allows the user to choose either a given ITM Stimulus port (from port 0 to 31) or receive data simultaneously from all ITM Stimulus ports.

The user can optionally specify a “.log” file to save the SWV trace log by using the “Browse” button, the default is

“\$USER_HOME/STMicroelectronics/STM32CubeProgrammer/SWV_Log/swv.log”.

The user can optionally check the “Activate colors” checkbox to enable colored traces output. This feature requires the original traces to contain the color codes listed below:

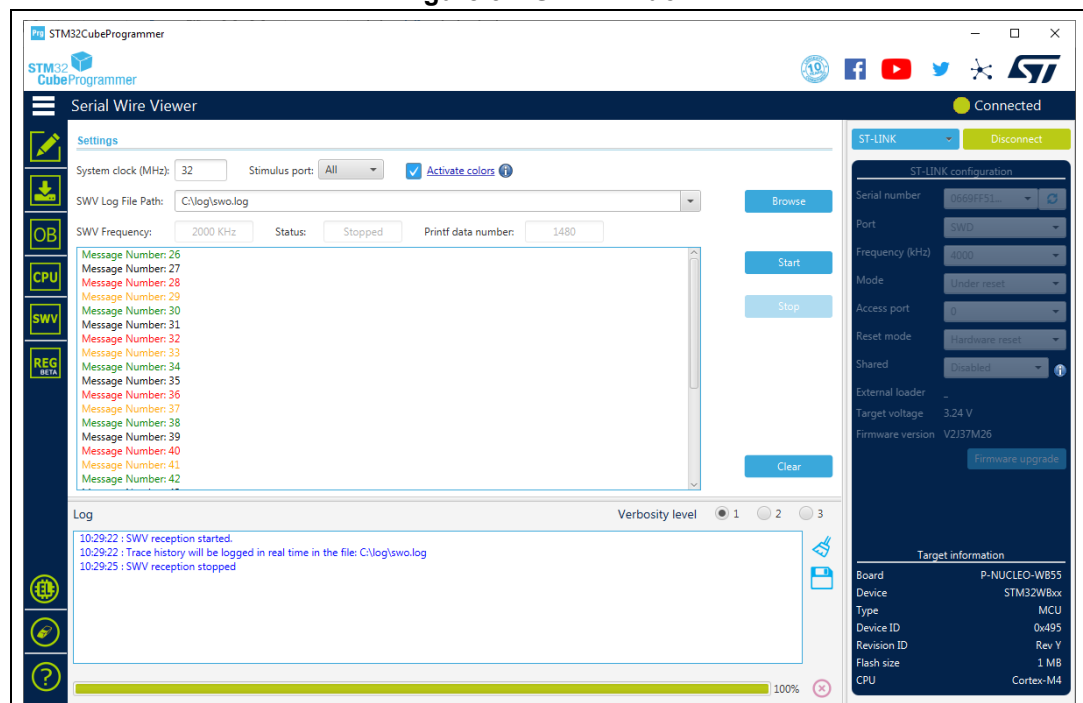
- #GRN# for green color
- #RED# for red color
- #ORG# for orange color

Example:

```
printf("#GRN#This outputs a green message!");
```

A help window that demonstrates the feature and shows how to use it can be accessed by clicking on the “Info icon” button next to the “Activate colors” checkbox.

Figure 31. SWV window



After specifying the SWV configuration, SWV reception can be started or stopped using the “Start” and “Stop” buttons. The SWO data is displayed in the dedicated area, which can be cleared by using the “Clear” button.

The SWV information bar displays useful information on the current SWV transfer, such as the SWO frequency (deduced from the system clock frequency), and the received printf data number (expressed in bytes).

Note: Some SWV bytes can be lost during transfer due to ST-LINK hardware buffer size limitation.

2.9 STM32CubeProgrammer Script Manager platform for MCUs

2.9.1 Introduction for the usage scenarios of Script Manager

The Script Manager platform allows to automate STM32CubeProgrammer CLI commands and adds macros to manipulate data read from STM32 MCU.

2.9.2 Script Manager usage

Create a file with a prg extension, then start writing the command line interface (CLI) supported by all STM32 MCUs and the specific script macros. Once you have finished filling the script, connect the STM32 board and start execution with the `-script` command in CLI mode.

Usage example: `STM32_Programmer_CLI -script myScript.prg`

The Script Manager can apply mathematical and logical operations, see [Table 1](#).

Table 1. Operations supported by Script Manager

Mathematical	Logical
– + (addition) – - (subtraction) – * (multiplication) – / (division)	– && (logical AND) – (logical OR) – & (bitwise AND) – (bitwise OR) – ^ (XOR) – << >> (left and right shift)

Using command line interface (CLI): in this script we can use all CLI supported by STM32 MCUs (see [Section 3](#)).

Using specific Script Manager macros, to analyze, display and modify data, each macro starts with #. Supported macros are described below.

#Write macro:

#Write32 (Address, data)

#Write16 (Address, data)

#Write8 (Address, data)

#WriteX (Address, #var) (where X is 8/16/32)

Description: Downloads the specified (32/16/8)-bit data into Flash memory starting from a specified address.

#Read macro:

#Read(Address)

#variable=#Read(Address)

Description: Reads 32-bit data memory from a specified address or read 32-bit data memory from a specified address and put it in the variable used by the user.

#Display macro:

#Display("message")

#Display(#errorLevel)

#Display(#variable)

Description: Displays any message, data, error level and the content of variables already used in the script.

#Delay macro:

#Delay(Time)

Description: Allows user to put the system in standby for a period in (ms).

Calculate macro:

#variable=[var1] op [var2]

#variable=var1 shift (number of bits to shifted)

Description: Calculates with mathematical and logical operations in script manager.

Disconnection command

--scriptdisconnect

Description: Allows user to disconnect the device and reconnect to another port in the same script.

Note: You can add comments in the Script Manager by using "//", as shown in the examples.

Script Manager example 1 (CLI and Script macro), see [Figure 32](#)

-c port=swd

-e 0 1

#Write32(0x08000000, 0xAAAABBBB)

#var0=#Read(0x08000000)

#Display(#var0)

Script Manager example 2, see [Figure 33](#)

```
-c port=swd
#Write32(0x08000000,0xAAAABBBB)
--scriptdisconnect
#Delay(5000)
-c port=COM17
#Write16(0x08000004,0xCCCC)
```

Script Manager example 3

```
-c port=swd
#Display ("Hello World!")
-e 0 1
#Write32(0x08000000,0xAAAABBBB)
#Read(0x08000000)
-r32 0x08000000 0x50
#var0=#Read(0x08000000)
#Display(#errorLevel)
#Display(#var0)
#Write32(0x08000004,#var0)
#Delay(3000)
#Write16(0x08000008,0xCCCC)
#Read(0x08000004)
#Display(#errorLevel)
#var1=#Read(0x08000008)
#Display(#var1)
#Write8(0x08000010,0xDD)
#Delay(5000)
#var2=#Read(0x08000010)
#Display(#var2)
#var3=((0xbb*1)+(1-1))/1)
#Display(#var3)
#Write8(0x08000014,#var3)
#var4=((0xbb & 0xaa) | 0xbb )
#Display(#var4)
#var5=((0xbb && 0xaa) | | 0xbb )
#Display(#var5)
#var6=(0xbb >>1)
#Display(#var6)
-e 0 1
-w32 0x08000000 0xAAAAAAAA
-r32 0x08000000 0x50
```

Figure 32. Output of Script Manager example 1

```
--- Script Manager BEGIN ---  
  
Operation [1]: -c port=swd  
  
ST-LINK SN : 066BFF565251887067053951  
ST-LINK FW : V2J33M25  
Board : NUCLEO-F429ZI  
Voltage : 3.27V  
SWD freq : 4000 KHz  
Connect mode: Normal  
Reset mode : Software reset  
Device ID : 0x419  
Revision ID : Rev 3  
Device name : STM32F42xxx/F43xxx  
Flash size : 2 MBytes  
Device type : MCU  
Device CPU : Cortex-M4  
BL Version : --  
  
Operation [2]: -e 0 1  
  
Erase sector(s) ...  
  
Existing specified sectors are erased successfully  
Protected sectors are not erased  
  
Operation [3]: #Write32(0x08000000,0xAAAABBBB)  
  
DOWNLOADING ...  
Size : 4 Bytes  
Address : 0x08000000  
  
Data downloaded successfully  
  
Operation [4]: #var0=#Read(0x08000000)  
  
UPLOADING ...  
Size : 4 Bytes  
Address : 0x08000000  
Read progress:  100%  
Data read successfully  
Time elapsed during the read operation is: 00:00:00.001  
  
Operation [5]: #Display(#var0)  
  
#var0 = 0xAAAABBBB  
Device is disconnected  
  
--- Script Manager END ---
```

Figure 33. Output of Script Manager example 2

```

=== Script Manager BEGIN ===
Operation [1]: -c port=swd
ST-LINK SN : 066BFF565251887067053951
ST LINK FW : V2J33M25
Board : NUCLEO-F429ZI
Voltage : 3.27V
SWD freq : 4000 KHz
Connect mode: Normal
Reset mode : Software reset
Device ID : 0x419
Revision ID : Rev 3
Device name : STM32F42xxx/F43xxx
Flash size : 2 MBytes
Device type : MCU
Device CPU : Cortex-M4
BL Version : --

Operation [2]: #Write32(0x08000000,0xAAAABBBB)

DOWNLOADING ...
Size : 4 Bytes
Address : 0x08000000

Erasing internal memory sector 0
Data downloaded successfully

Operation [3]: #Delay(5000)

The system go to sleep for 5000 ms.

Operation [4]: -c port=COM17

Serial Port COM17 is successfully opened.
Port configuration: parity = even, baudrate = 115200, data-bit = 8,
stop-bit = 1.0, flow-control = off

Timeout error occurred while waiting for acknowledgement.
Activating device: OK
Board : --
Chip ID: 0x419
BootLoader protocol version: 3.1
Device name : STM32F42xxx/F43xxx
Flash size : 2 MBytes (default)
Device type : MCU
Revision ID : --
Device CPU : Cortex-M4

Operation [5]: #Write16(0x08000004,0xCCCC)

DOWNLOADING ...
Size : 2 Bytes
Address : 0x08000004

Erasing internal memory sector 0
Data downloaded successfully
Device is disconnected
=== Script Manager END ===

```

2.10 DFU IAP/USBx with custom PID and VID

STM32CubeProgrammer DFU IAP/USBx supports not only ST product IDs while connecting via DFU IAP.

Before starting the DFU connection using a new product ID, sign your USB driver (for more info visit <http://woshub.com>).

When USB connection with a new product ID is chosen and the boot is from Flash memory, STM32CubeProgrammer detects the IAP/USBx like DFU bootloader and after connection an IAP message appears in the log panel.

To connect via the new USB DFU follow this sequence:

1. modify the default product ID
2. modify the default vendor ID
3. click on refresh button then on the connect button

Note: *If user does not enter a PID or VID value STM32CubeProgrammer takes the default PID and VID of ST products (PID=0XDF11, VID=0X0483).*

[Figure 34](#) shows the steps to connect via the new USB DFU panel, and [Figure 35](#) the main window of STM32CubeProgrammer after connection.

Figure 34. Connect via USB DFU panel

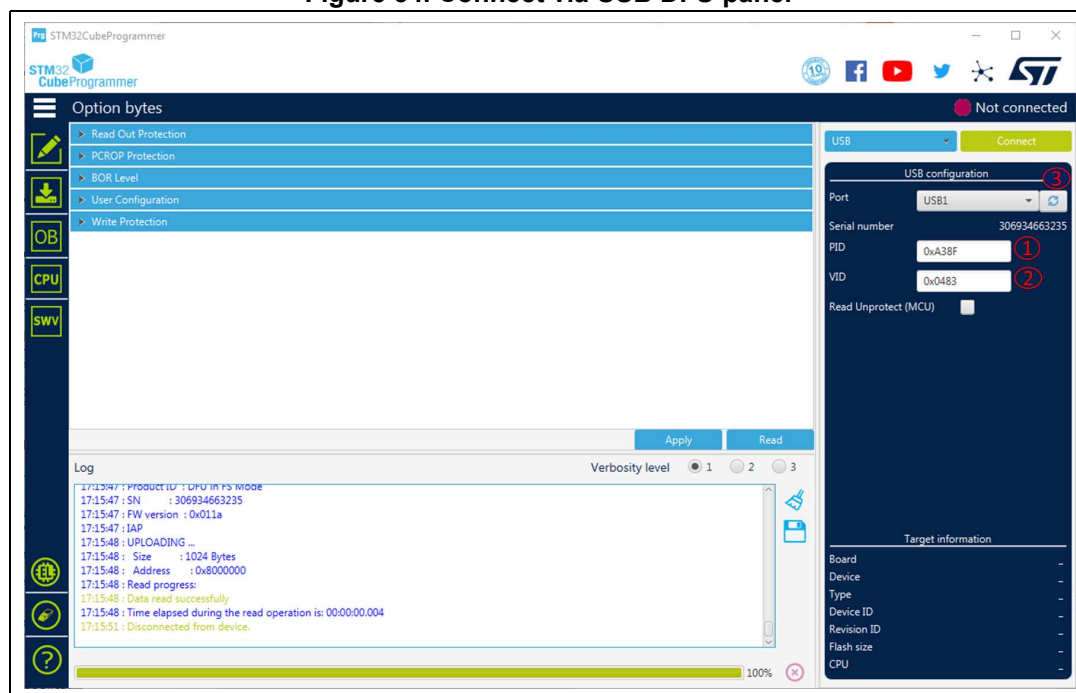
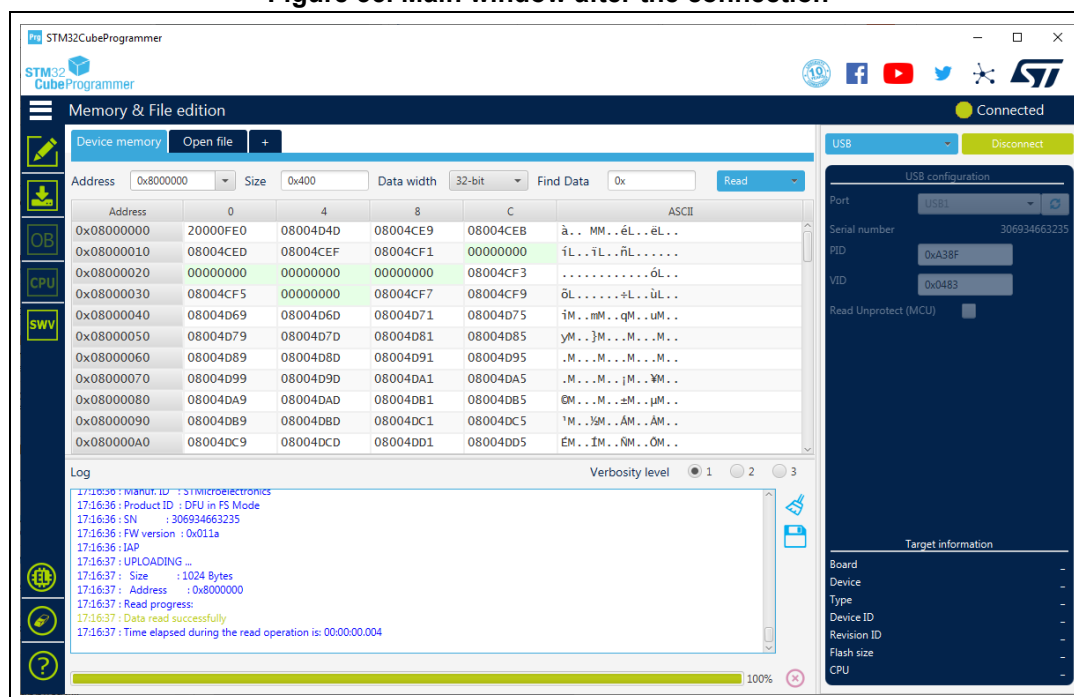


Figure 35. Main window after the connection



Note: For CLI mode check the [Section 3.2.1: Connect command](#).

2.11 SigFox™ credentials

As soon as an STM32WL device is connected, the window shown in [Figure 36](#) is displayed.

This window displays the chip certificate having the size of 136 bytes. The user can save it in binary file and copy the data to the clipboard.

After extracting the chip certificate, a back-end web-service verifies the data and returns two SigFox credentials: binary and header files.

Case 1: Binary-Raw

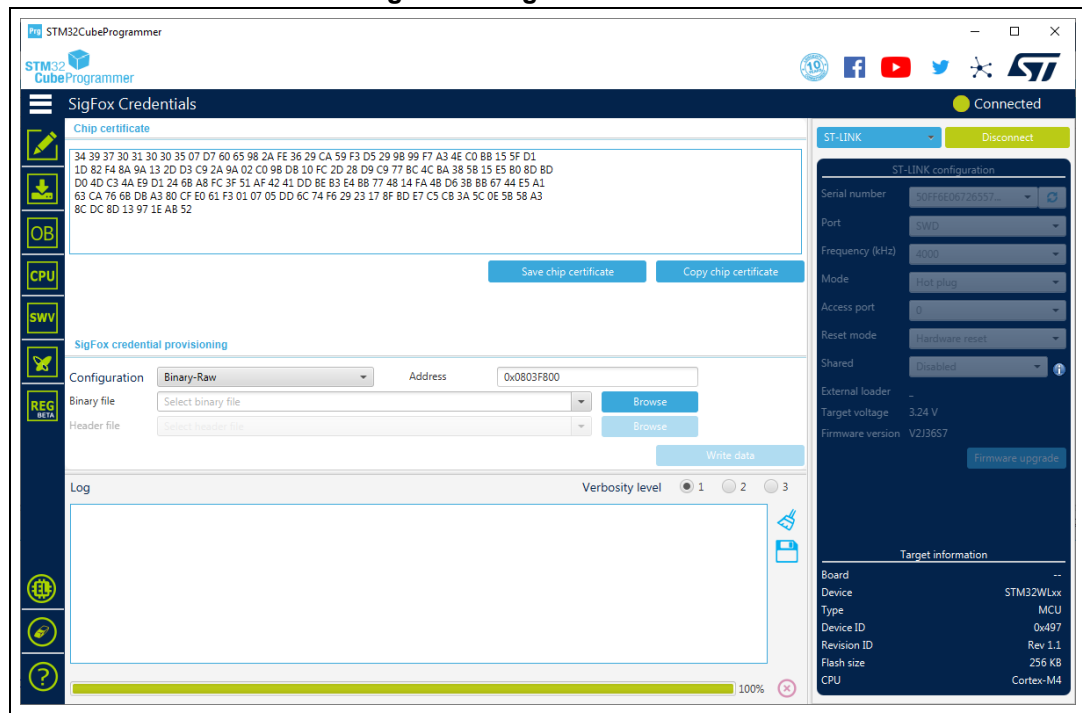
Use the binary file returned by the back-end web-service. The size of this file must be equal to 48 bytes, it is written at the default address 0x0803E500.

Case 2: Binary KMS

Use the header file returned by the back-end web-service. It is written at the default address 0x0803E500.

Note: To access ST SigFox server using STM32CubeProgrammer, user must click on “Open Sigfox page”. A web page opens, the user must manually copy the certificate and then generate the SigFox credentials (binary and header files).

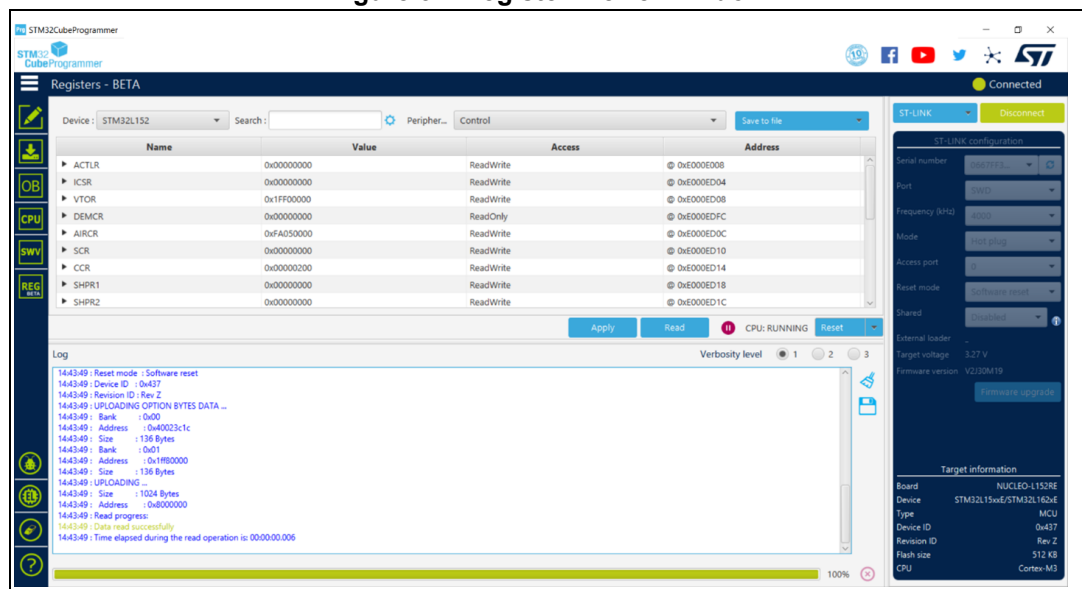
Figure 36. SigFox credentials



2.12 Register Viewer

STM32CubeProgrammer supports the Register Viewer feature (see [Figure 37](#)), allowing the user to visualize all the MCU and core registers in real time while running the application. It also allows the modification of MCU registers values or saving them into a log file.

Figure 37. Register Viewer window



Note: The register viewer is only available through SWD/JTAG interfaces.

Register Viewer has as input a list of files containing the data describing the mapping of the core and STM32 registers ("svd" files).

2.13 Hard Fault analyzer

2.13.1 Description

The STM32CubeProgrammer Fault Analyzer feature interprets information extracted from the Cortex-M based device to identify the reasons that caused a fault.

This information is visualized in the Fault Analyzer window in GUI mode or in CLI mode. It helps to identify system faults occurring when the CPU is driven into a fault condition by the application software.

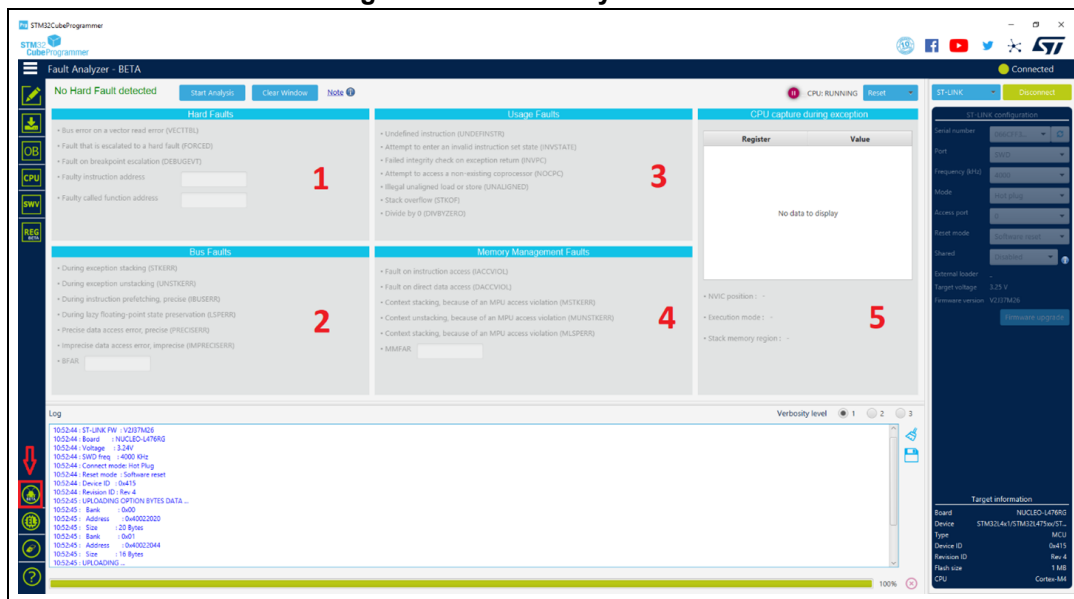
Possible detected fault exceptions:

- Hard Fault: default exception, can be triggered by an error during exception processing by Bus Fault, Memory Management Fault, or Usage Fault if their handler cannot be executed.
- Memory Management Fault: detects memory access violations to regions defined in the memory management unit (MPU), such as code execution from a memory region with read/write access only.
- Bus Fault: detects memory access errors on instruction fetch, data read/write, interrupt vector fetch, and register stacking (save/restore) on interrupt (entry/exit).
- Usage Fault: detects execution of undefined instructions, unaligned memory access for load/store multiple. When enabled, divide-by-zero and other unaligned memory accesses are detected.
- Secure Fault: provides information about security related faults for Cortex-M33 based devices.

Note: Fault Analyzer is available only for ST-LINK interfaces.

As shown in [Figure 38](#) the Fault Analyzer Window has five main sections.

Figure 38. Fault Analyzer window



1. **Hard Faults details:** indicates the type of occurred fault, locates the instruction and the called function addresses.
2. **Bus Faults details:** shows the status of bus errors resulting from instruction fetches and data accesses and indicates memory access faults detected during a bus operation. An address should be displayed on the BFAR text field.
3. **Usage Faults details:** contains the status for some instruction execution faults, and for data access.
4. **Memory Management Faults details:** indicates a memory access violation detected by the MPU. If this fault was triggered by a faulty address, access is displayed on the MMFAR text field.
5. **CPU capture during exception:** shows the CPU state when an exception was generated to have an overview for CPU registers and some helpful information.
 - a) **NVIC position:** indicates the number of the interrupt imposing the error, if it is “-” the interrupt/exception vector has no specific position.
 - b) **Execution mode:** indicates the operation mode Handler/Thread.
 - c) **Stack memory region:** indicates the used stack memory during the fault, Main or Process stack.

2.13.2 Example

Develop a simple application that generates a Usage fault, set an instruction making a divide by zero (a non-permitted operation) in the main program function.

```
int a = 4, b = 0, c = 0;
```

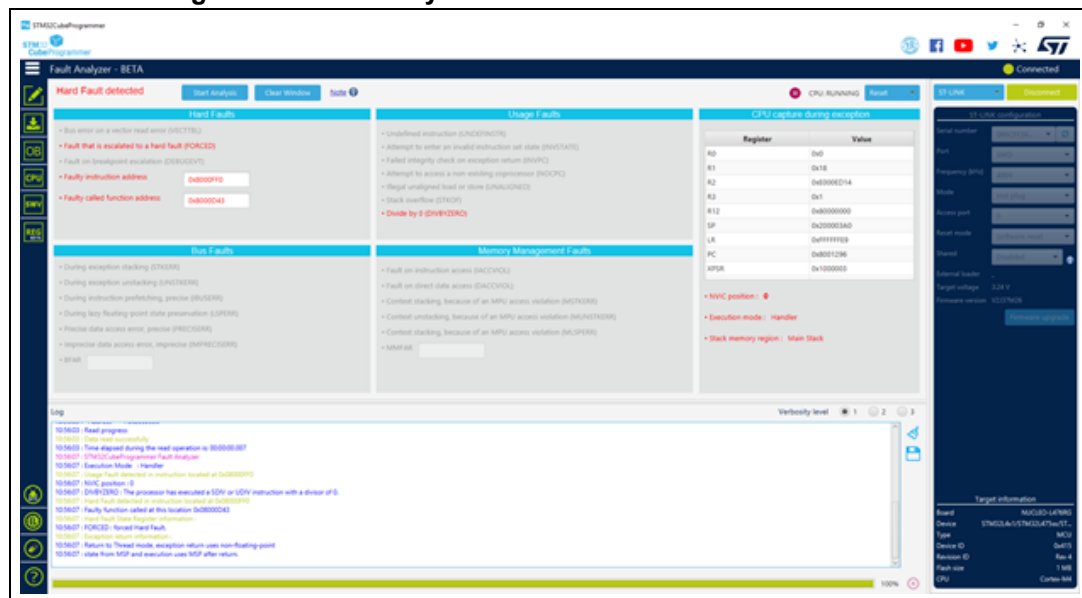
```
c = a / b;
```

Open the Fault Analyzer window, press the “Start Analysis” button to start the fault detection algorithm, the reason of the error is displayed.

In this example, it displays “Hard Fault Detected”, and the label “divide by zero (DIVBYZERO)” is highlighted with additional informations:

- Faulty instruction address: 0x8000FF0
- Faulty called function address: 0x8000D40, indicates the address calling the faulty instruction.
- NVIC position: 0, window watchdog interrupt
- Execution mode: Handler
- Stack memory region: main stack

Figure 39. Fault analyzer GUI view when Hard Fault detected



2.13.3 Fault Analyzer note

Fault Analyzer may be unable to detect untracked faults since they were not enabled by software.

The configuration and control register (CCR) controls the behavior of the Usage Fault for divide-by-zero and unaligned memory accesses and it is used mainly to control customizable fault exceptions.

The following bits of the CCR control the behavior of the Usage Fault:

Figure 40. CCR bits

31	...	10	9	8	7	6	5	4	3	2	1	0
Reserved			STKALIGN	BHFNMMIGN	Reserved			DIV_0_TRP	UNALIGN_TRP	Reserved	USERSETMPEND	NONBASETHRDENA

- **DIV_0_TRP:** Enable Usage Fault when the processor executes an SDIV or UDIV instruction with a 0 divider.
 - 0 = do not trap divide by 0; a divide by 0 returns a quotient of 0.
 - 1 = trap divide by 0.
- **UNALIGN_TRP:** enable usage fault when a memory access to unaligned addresses is performed.
 - 0 = do not trap unaligned half-word and word accesses
 - 1 = trap unaligned half-word and word accesses; an unaligned access generates a usage fault.

Note that unaligned accesses with LDM, STM, LDRD, and STRD instructions always generate a usage fault, even when UNALIGN_TRP is set to 0.

STM32CubeProgrammer enables the required bits at the analysis startup, if no fault detected an informative popup is displayed to indicate that you must reproduce the scenario and restart the fault Analysis.

2.13.4 Secure Fault Analyzer for Cortex-M33

STM32CubeProgrammer provides information about security related faults for Cortex-M33 based devices for both CLI and GUI interfaces.

A new field named “Secure Faults” is added to Fault Analyzer window when you connect a device based on Cortex-M33 (such as MCUs of the STM32L5 Series).

The result analysis is based on Secure Fault Status Register (SFSR) settings and a fault is triggered if an error occurs:

- **INVEP:** this bit is set if a function call from the Non-secure state or exception targets a non-SG instruction in the Secure state. This bit is also set if the target address is a SG instruction, but there is no matching SAU/IDAU region with the NSC flag set.
- **INVIS:** this bit is set if the integrity signature in an exception stack frame is found to be invalid during the unstacking operation.
- **INVER:** set to 1 when returning from an exception in the Non-secure state.
- **AUVIOL:** attempt was made to access parts of the address space that are marked as Secure with NS-Req for the transaction set to Non-secure. This bit is not set if the violation occurred during lazy state preservation.
- **INVTRAN:** indicates that an exception was raised due to a branch not flagged as being domain crossing causing a transition from Secure to Non-secure memory.

- LSPERR: Indicates that an SAU or IDAU violation occurred during the lazy preservation of floating-point state.
- SFARVALID: this bit is set when the SFAR register contains a valid value.
- LSERR: indicates that an error occurred during lazy state activation or deactivation.
- SFAR: indicates the address value when a secure fault is raised.

2.14 Fill memory command

-fillmemory

Description: This command allows the user to fill memory with a given pattern from the chosen address.

Syntax: `-fillmemory <start_address> [size=<value>] [pattern=<value>] [datawidth=8|16|32]`

<start_address>: Start address for write.
The address 0x08000000 is used by default.

[size=<value>]: Size of the data to write.

[pattern=<value>]: The pattern value to write.

[datawidth=8|16|32]: Filling data size, can be 8, 16 or 32 bits.
The selected value by default is 8 bits.

- Example 1: `STM32_Programmer_CLI.exe -c port=swd -fillmemory 0x08000000 size=0x10 pattern=0XAA datawidth=16` ([Figure 41](#))
- Example 2: `STM32_Programmer_CLI.exe -c port=swd -fillmemory 0x08000000 size= 0x10 pattern=0XCC datawidth=32` ([Figure 42](#))

Figure 41. Example 1

```

ST-LINK SN      : 0670FF554949677067035117
ST-LINK FW      : V2J30M20
Board           : STM32H743 -EV
Voltage         : 3.23V
SWD freq        : 4000 KHz
Connect mode     : Normal
Reset mode      : Software reset
Device ID       : 0x450
Revision ID     : Rev Z
Device name     : STM32H7xx
Flash size      : 2 MBytes
Device type     : MCU
Device CPU      : Cortex-M7

Filling memory operation:
  Start address      : 0x08000000
  Size(Bytes)       : 0x00000010
  Data value        : 0x000000AA
  Filling data size(Bytes): 16 bits

Erasing internal memory sector 0
100%

Downloading 16-bit data achieved successfully
The filling memory operation achieved successfully

```

Figure 42. Example 2

```
ST-LINK SN      : 0670FF554949677067035117
ST-LINK FW      : V2J30M20
Board           : STM32H743 -EV
Voltage         : 3.22V
SWD freq        : 4000 KHz
Connect mode    : Normal
Reset mode      : Software reset
Device ID       : 0x450
Revision ID     : Rev Z
Device name     : STM32H7xx
Flash size      : 2 MBytes
Device type     : MCU
Device CPU      : Cortex-M7

Filling memory operation:
  Start address           : 0x08000000
  Size(Bytes)             : 0x00000010
  Data value              : 0x000000CC
  Filling data size(Bytes): 32 bits

Erasing internal memory sector 0
100%

Downloading 32-bit data achieved successfully
The filling memory operation achieved successfully
```

2.15 Fill memory operation

The user can open the Fill memory window from different sub-menus.

Figure 43. Sub-menu displayed from “Read” combo-box

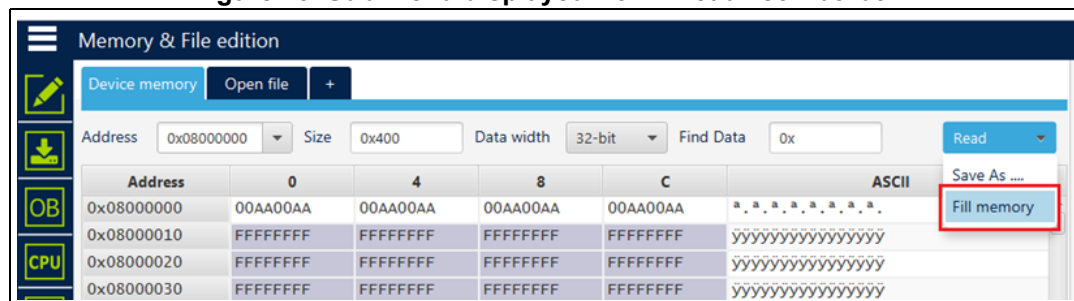


Figure 44. Sub-menu displayed with right click on “Device memory” tab

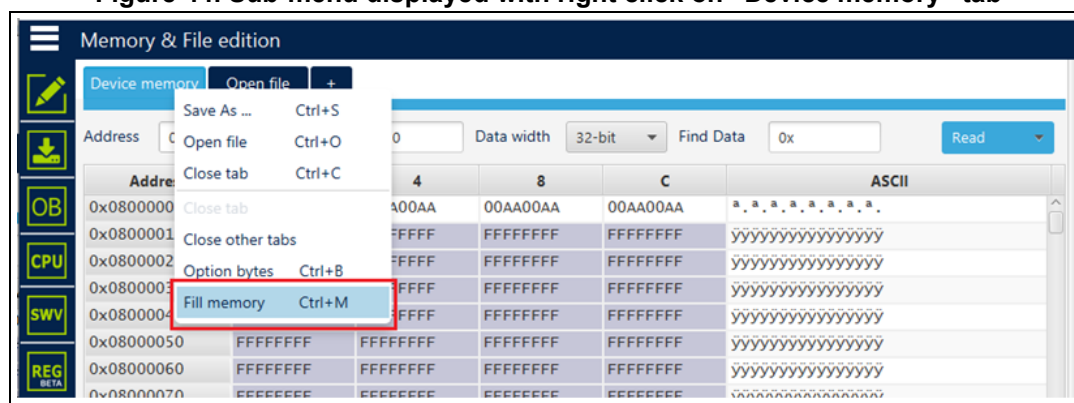
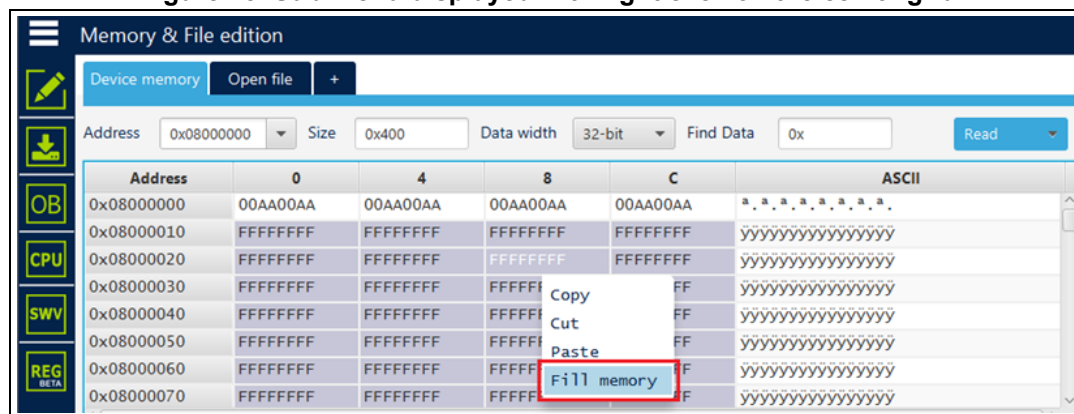


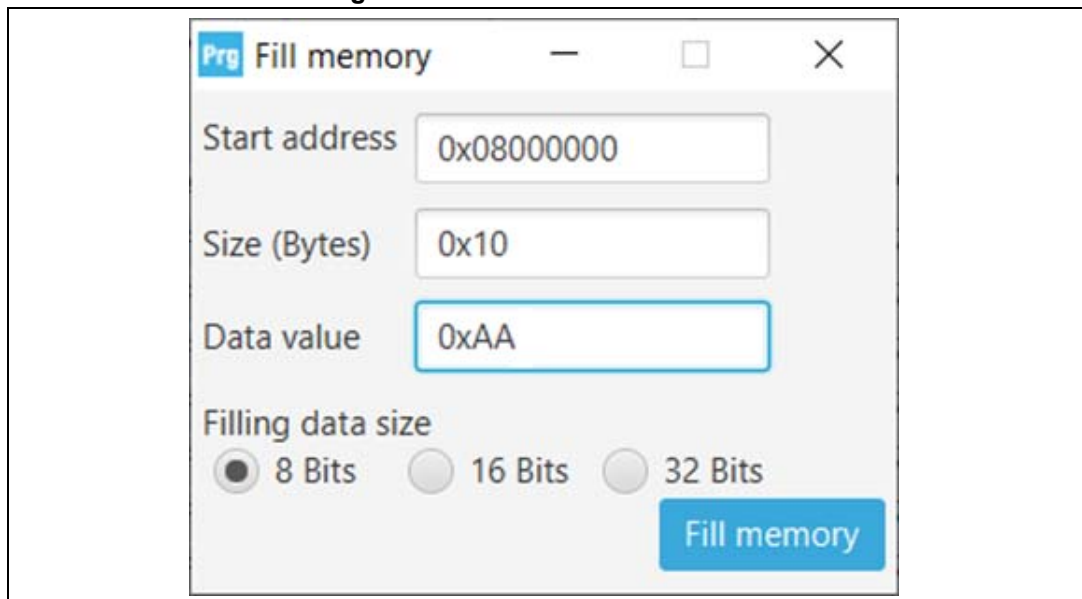
Figure 45. Sub-menu displayed with right click on the cell of grid



Note: In addition to sub-menus to display this window, user can open it directly by using the key combination “Ctrl+M”.

After clicking on “Fill memory” option, a window is displayed so that the user can initialize the parameters of the operation (see [Figure 46](#)).

Figure 46. Parameters initialization



2.16 Blank check command

-blankcheck

Description: This command allows the user to verify that the STM32 Flash memory is blank. If this is not the case, the first address with data is highlighted in a message.

Syntax: `-blankcheck`

Examples: `STM32_Programmer_CLI.exe -c port=swd -blankcheck`

Figure 47. Example 1: Flash memory is not blank at address 0x08000014

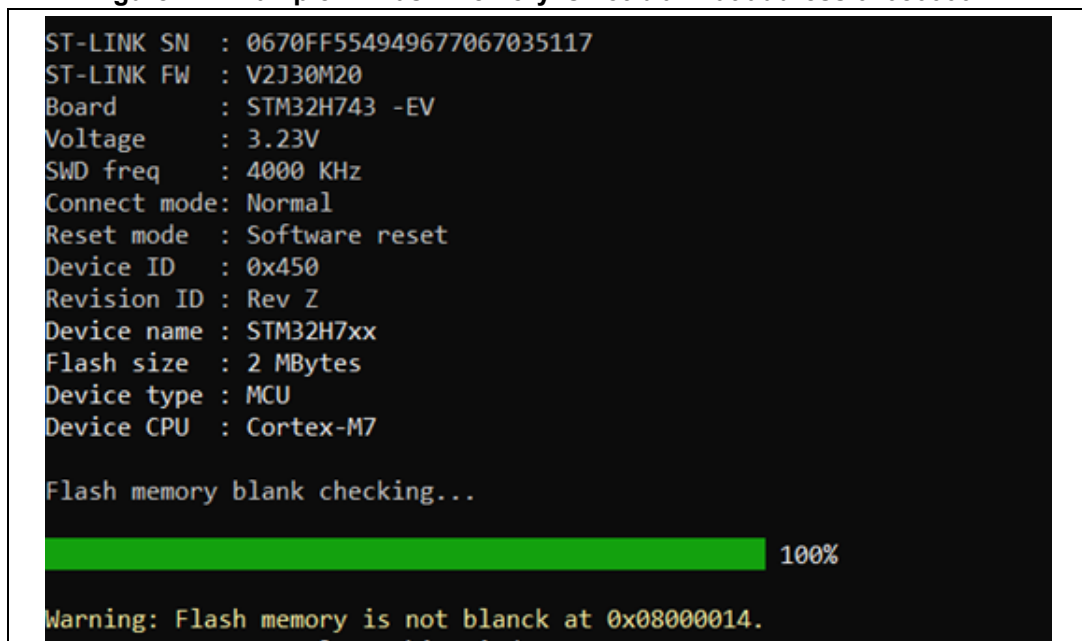
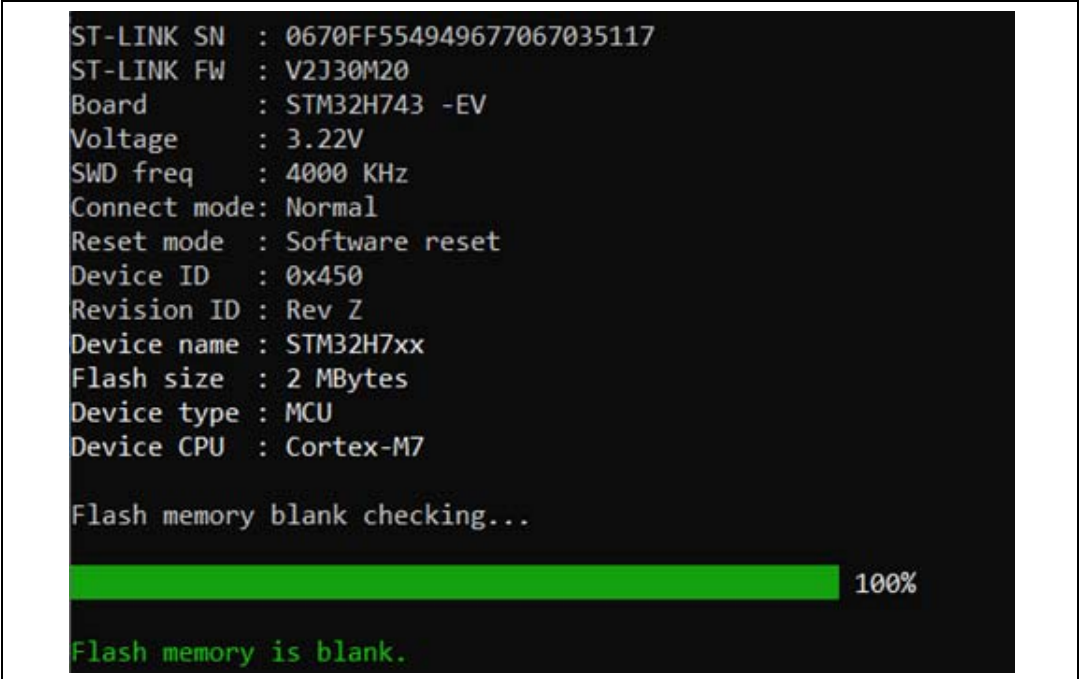


Figure 48. Example 1: Flash memory is blank



2.17 Blank check operation

The user can open the Fill memory window from different sub-menus.

Figure 49. Sub-menu displayed from “Read” combo-box

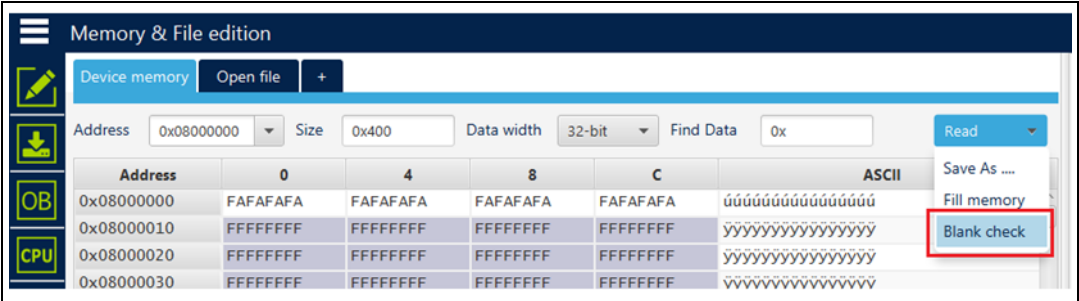


Figure 50. Sub-menu displayed with right click on “Device memory” tab

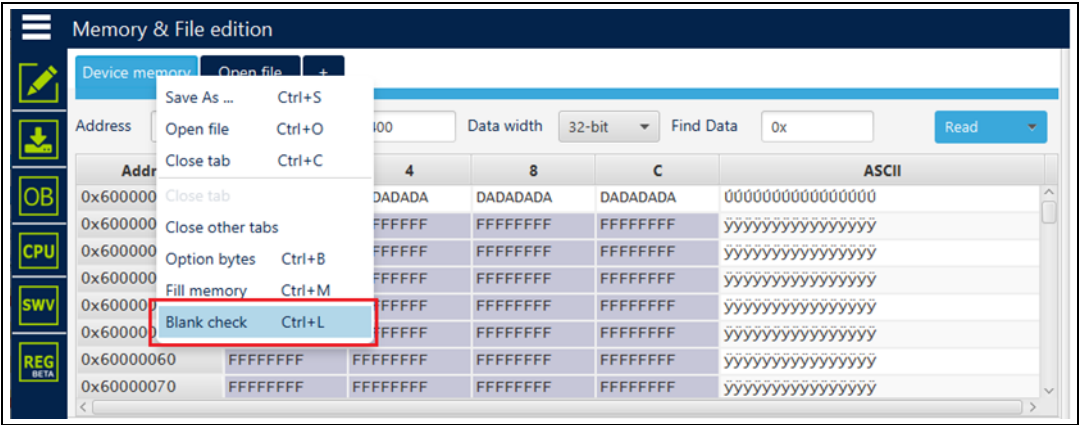
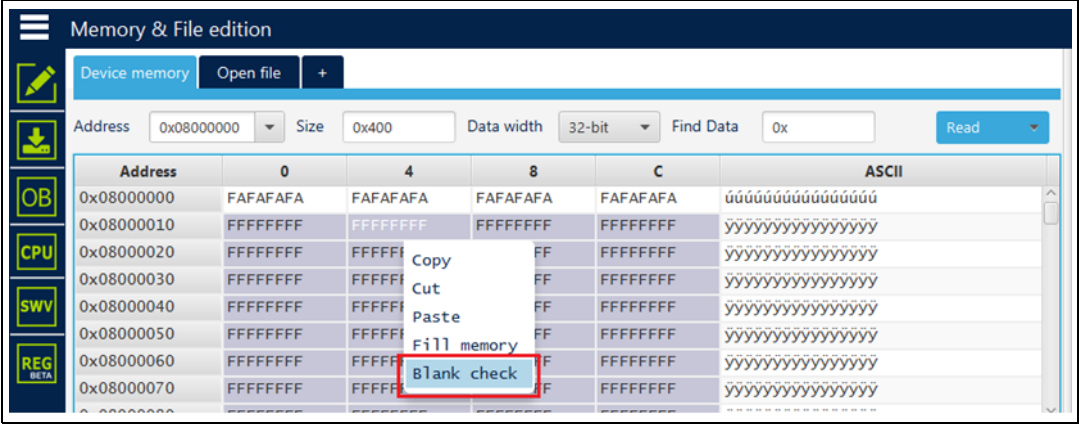


Figure 51. Sub-menu displayed with right click on the cell of grid



Note: In addition to sub-menus to display this window, user can launch the operation directly by using the key combination **Ctrl+L**.

After clicking on “Blank check” sub-menu, the process starts to verify that the STM32 Flash memory is blank. If the Flash memory is not blank, the first address with data is highlighted in a message, as shown in [Figure 52](#).

The expected results are shown in [Figure 53](#) and [Figure 54](#).

Figure 52. First address with data

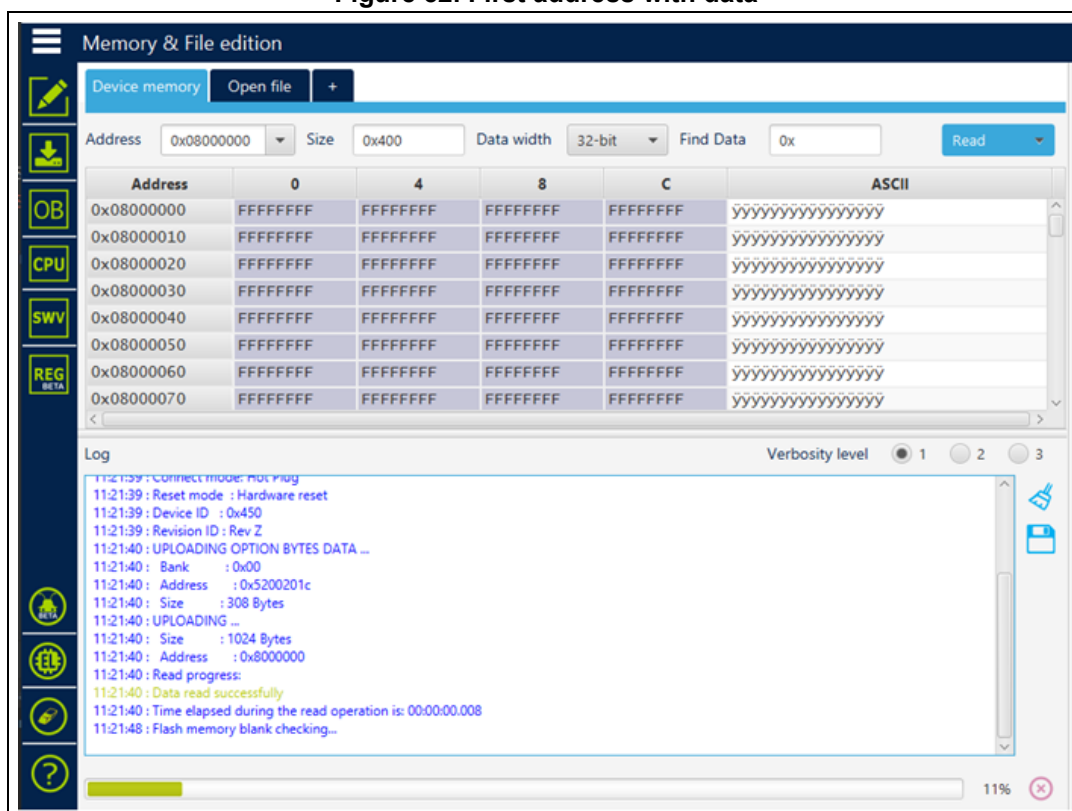


Figure 53. Example 1: Flash memory is blank

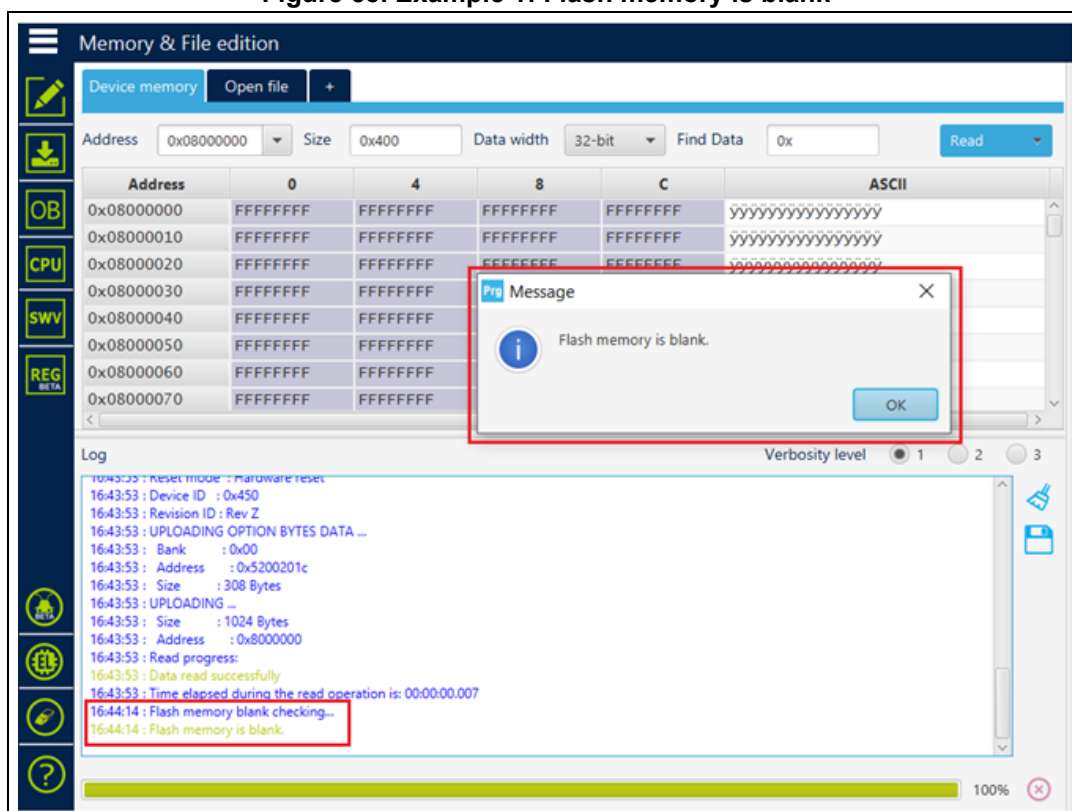
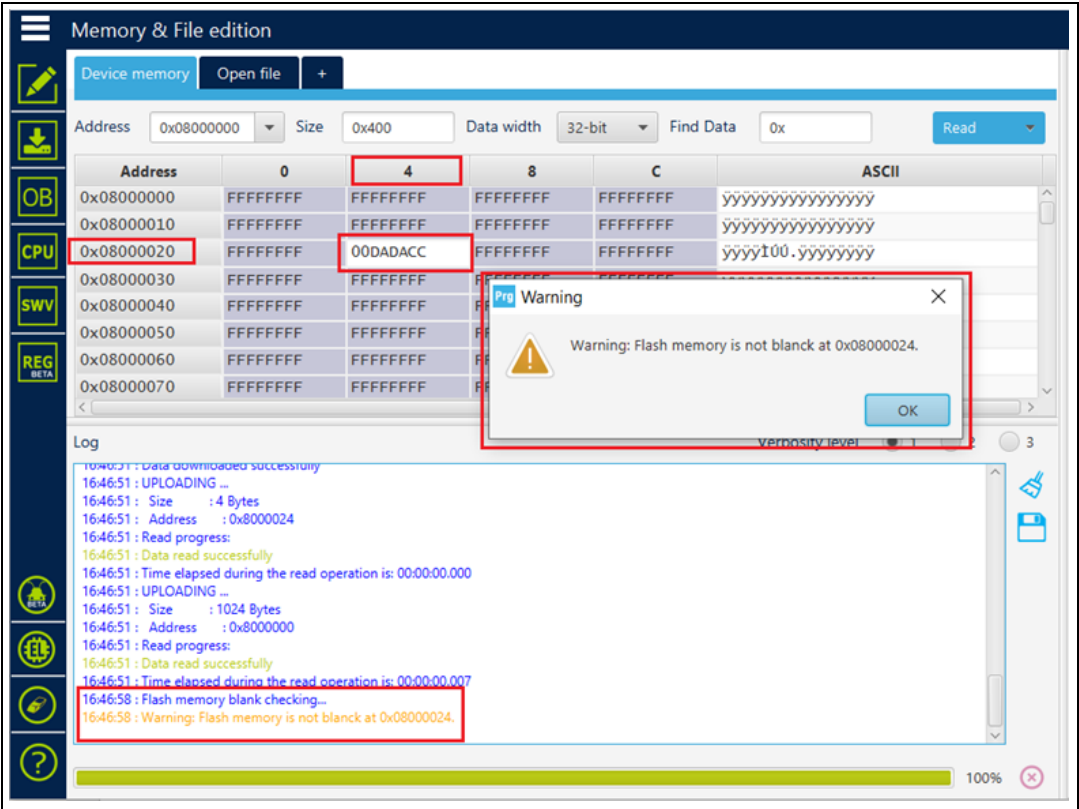


Figure 54. Example 2: Flash memory is not blank



2.18 Compare Flash memory with file

Description: Compares the MCU device memory content with a binary, hex, sec, elf, out and axf file. The difference is shown in red in the file and in the Flash memory panel.

The user can open the comparison window from different sub-menus.

Figure 55. Sub-menu displayed from “Read” combo-box

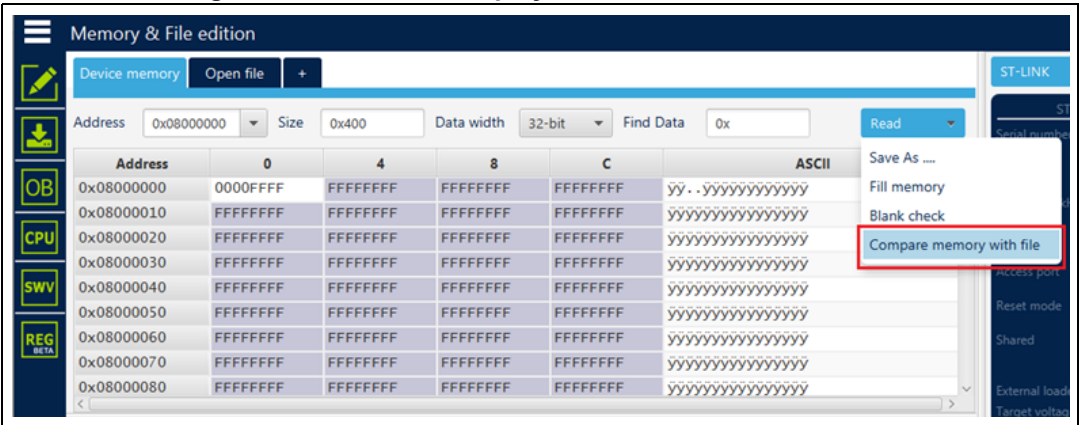


Figure 56. Sub-menu displayed with right click on “Device memory” tab

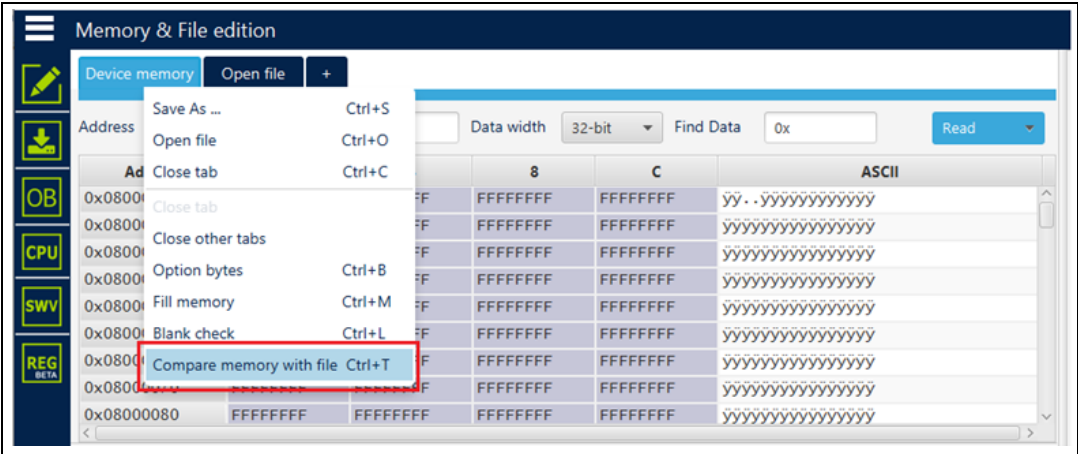


Figure 57. Sub-menu displayed with right click on the cell of grid

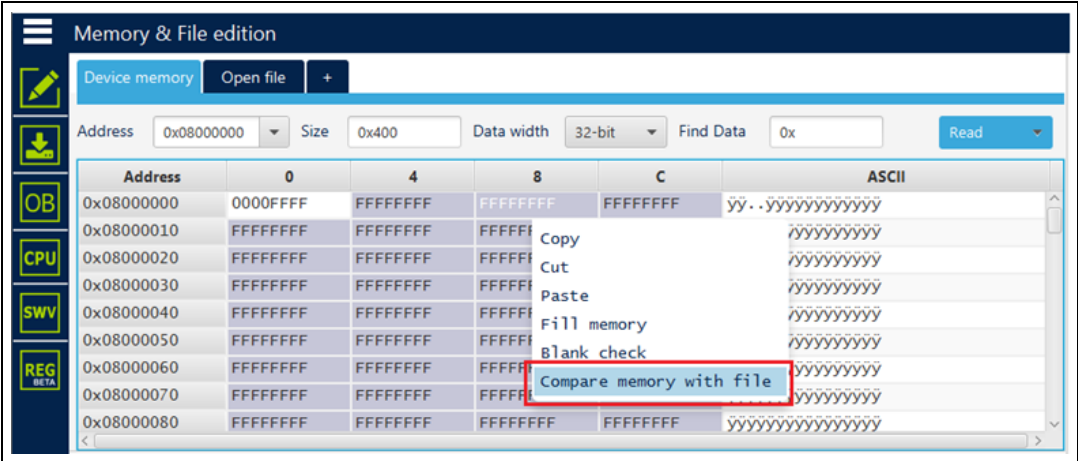


Figure 58. Sub-menu displayed with add tab button

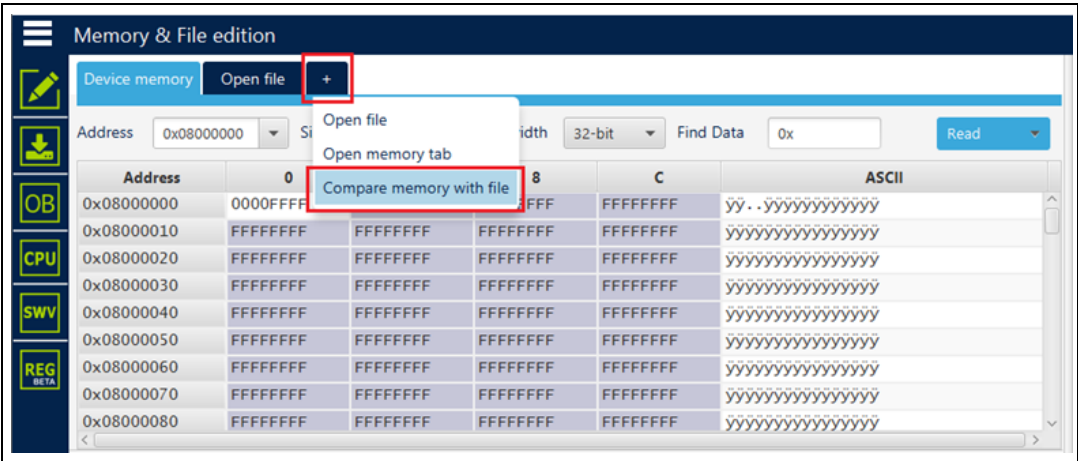


Figure 59. Sub-menu displayed with right click on the opened file tab

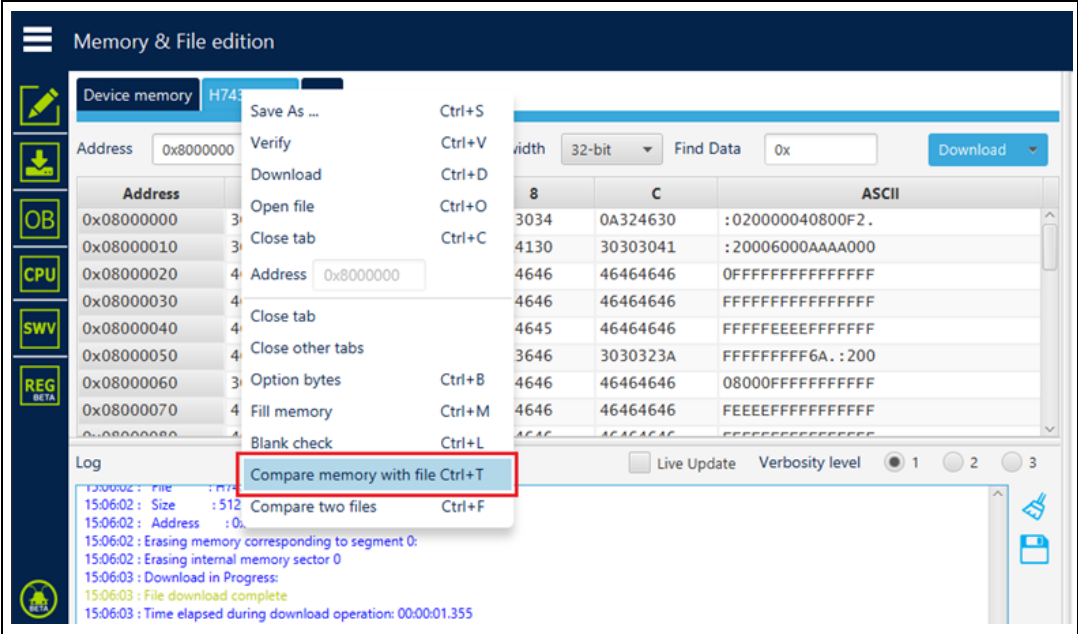
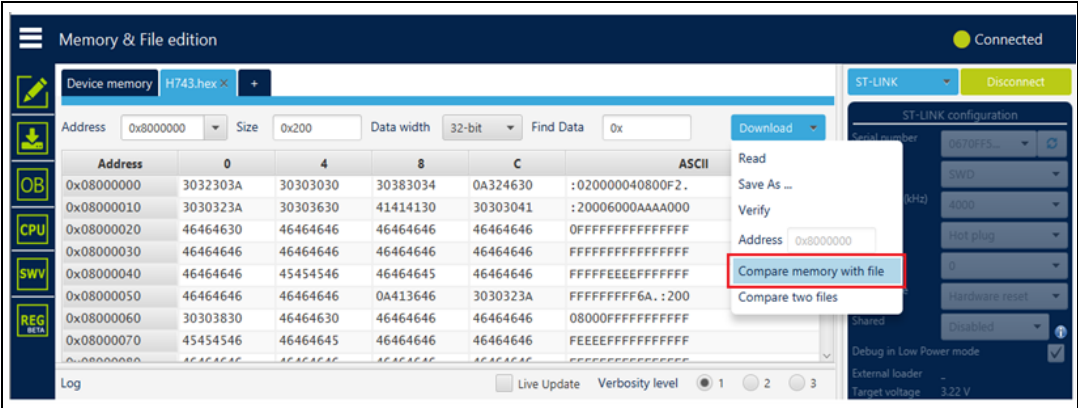


Figure 60. Sub-menu displayed from “Download” combo-box displayed in file tab



Note: In addition to sub-menus to display this window, the user can launch the operation directly by using the key combination **Ctrl+T**.

Example 1: Difference between internal Flash memory and binary file

Figure 61. Data width: 32 bits

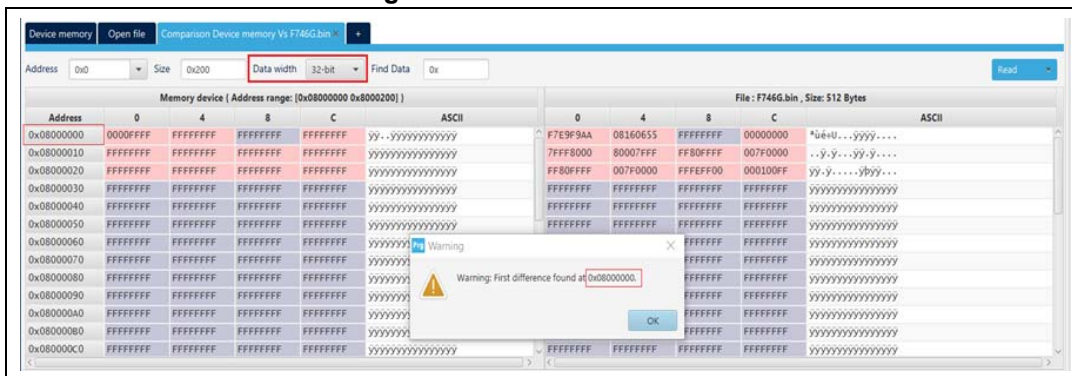


Figure 62. Data width: 16 bits

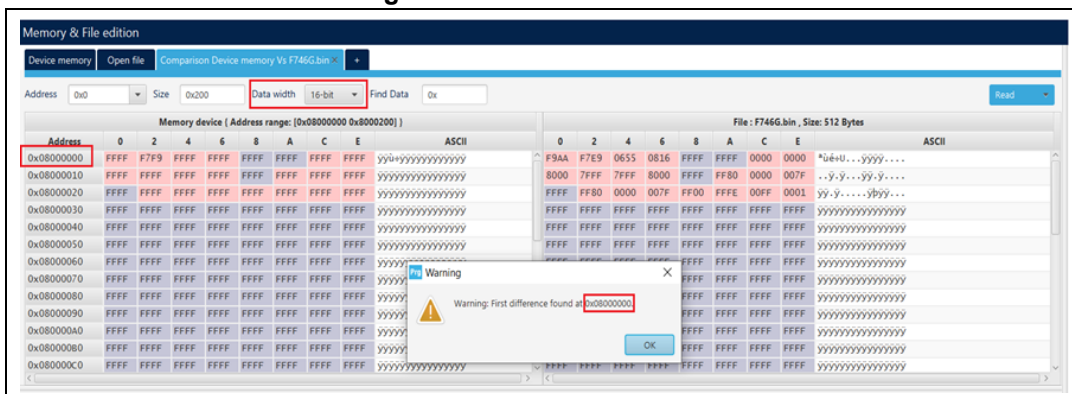
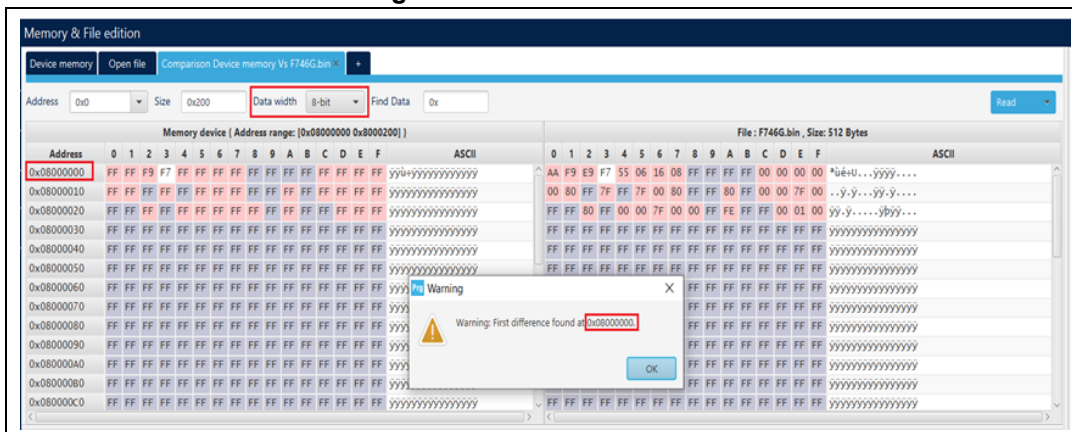


Figure 63. Data width: 8 bits



Example 2: Difference between external Flash memory and hex file

Figure 64. Data width: 32 bits

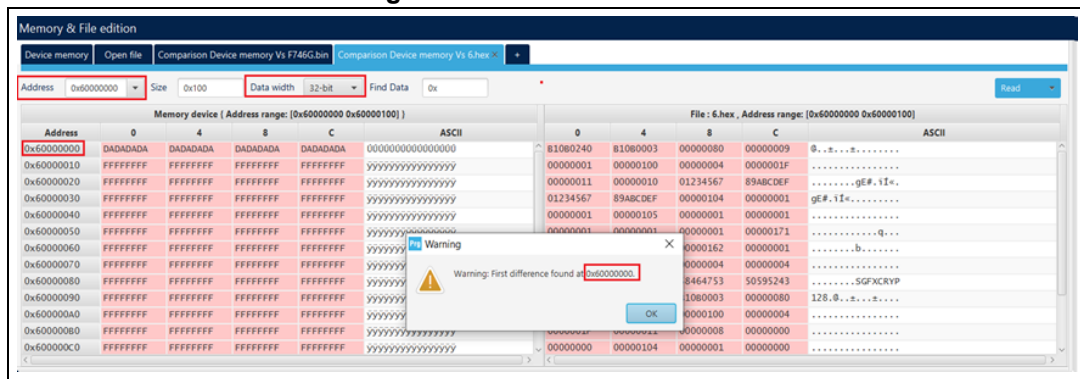


Figure 65. Data width: 16 bits

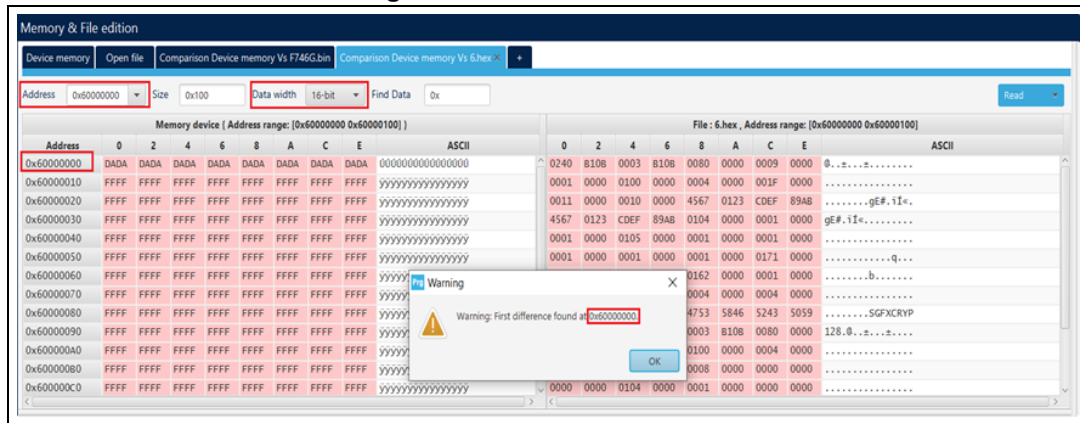
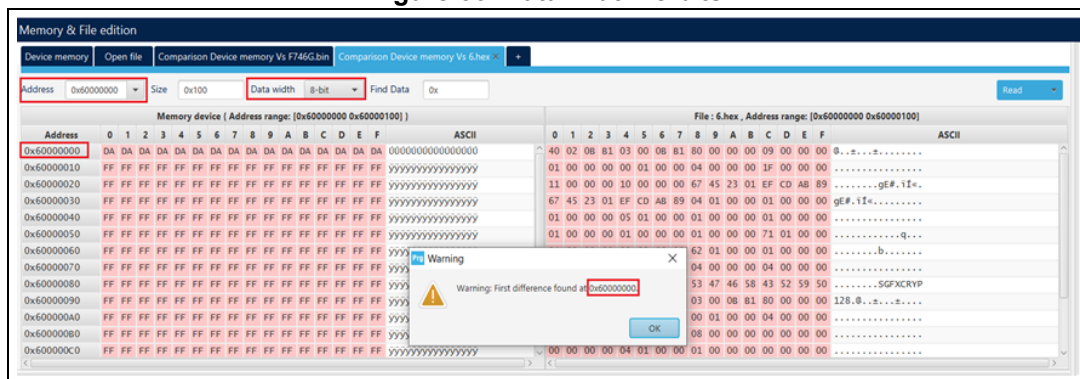


Figure 66. Data width: 8 bits



After launching the comparison between the Flash memory and file, and the edit of data in the memory, the user must make an update in the comparison tab using the read button.

Example 3: Update comparison between Flash memory and file after editing

Figure 67. Before editing the Flash memory

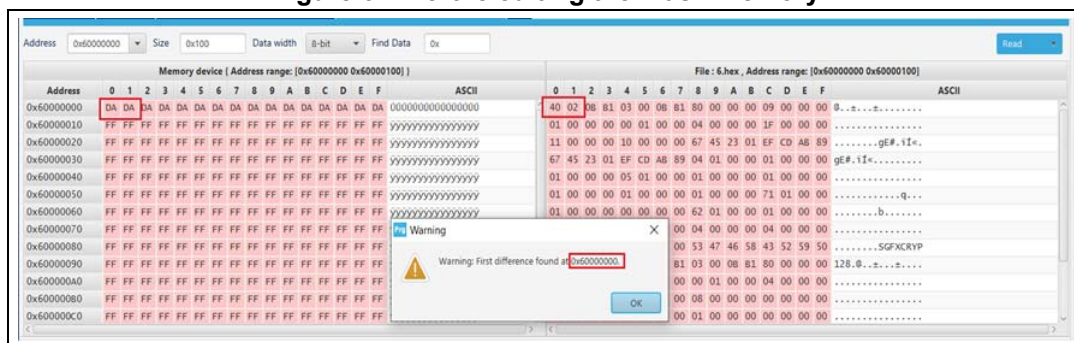
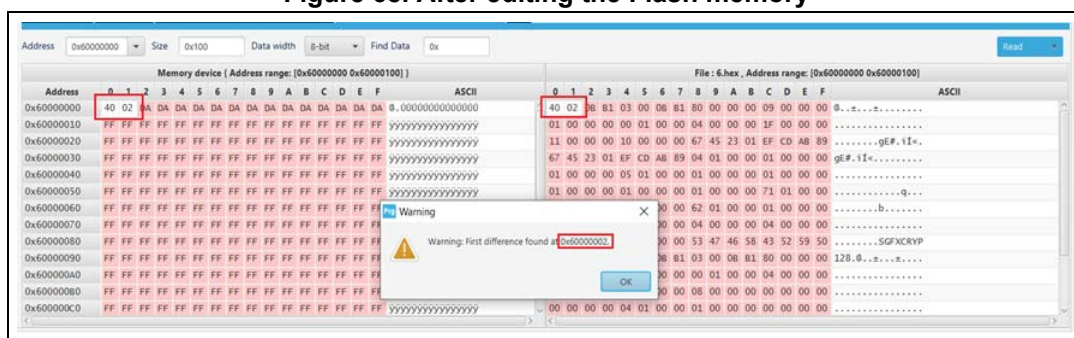


Figure 68. After editing the Flash memory



Note: The user can make multiple comparisons between Flash memory and files.

Figure 69. Multiple comparisons



2.19 Comparison between two files

Description: Compares the content of two different files (binary, hex, srec, elf, out and axf). The difference is colored in red in the grid panel of each file.

This operation does not need a connected board.

The used files can be of different sizes and types.

The user can open the comparison window from different sub-menus.

Figure 70. Sub-menu displayed from “Read” combo-box in device memory tab

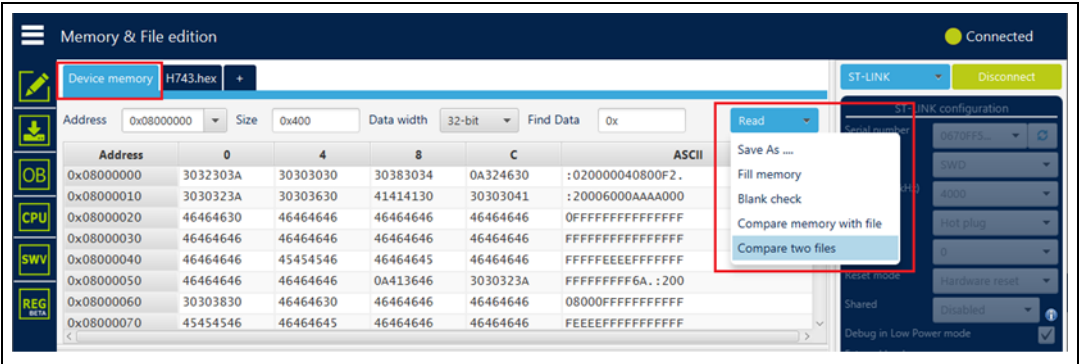


Figure 71. Sub-menu displayed with right click on “Device memory” tab

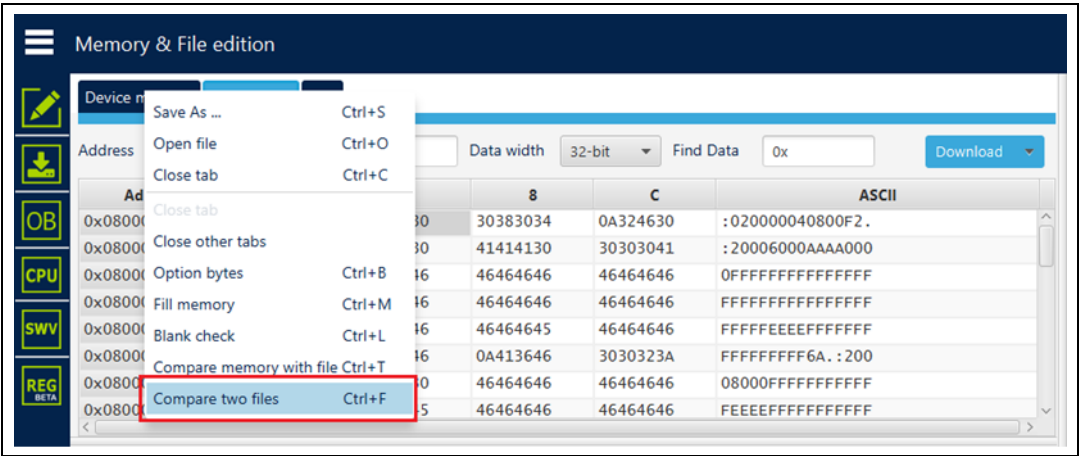


Figure 72. Sub-menu displayed with right click on the cell of grid

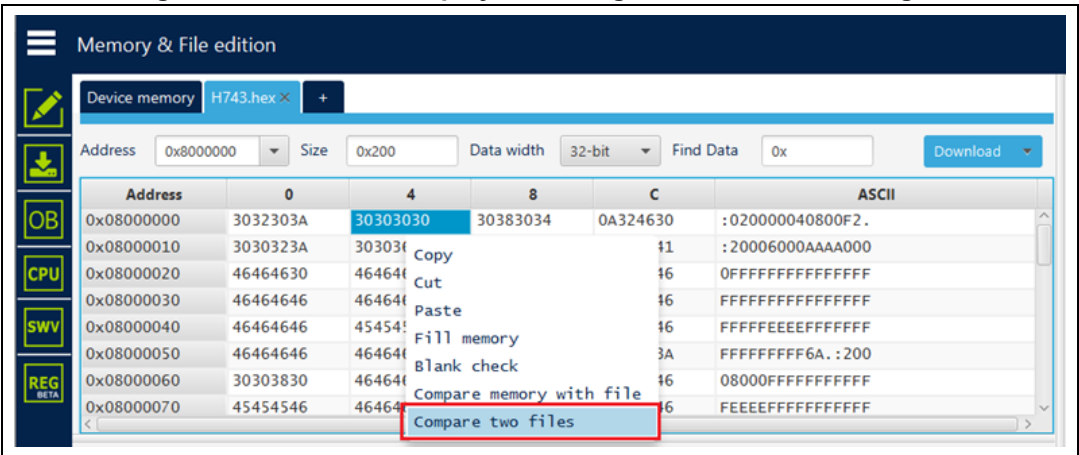


Figure 73. Sub-menu displayed with add tab button

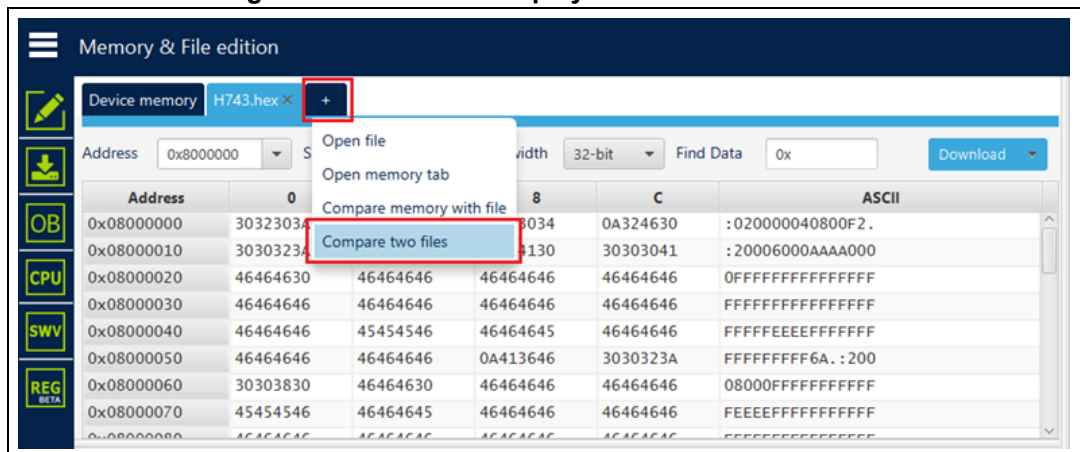


Figure 74. Sub-menu displayed with right click on the opened file tab

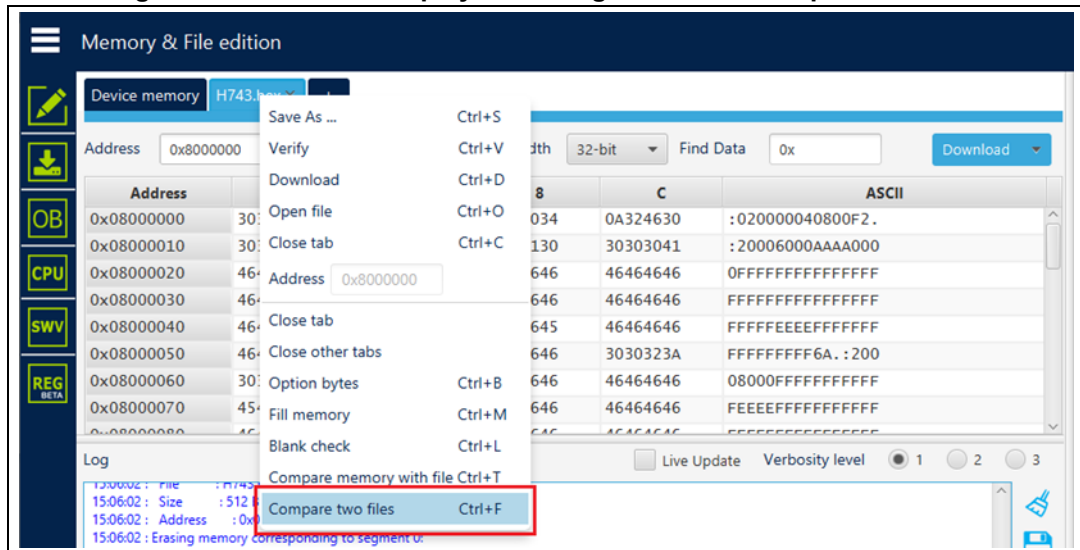
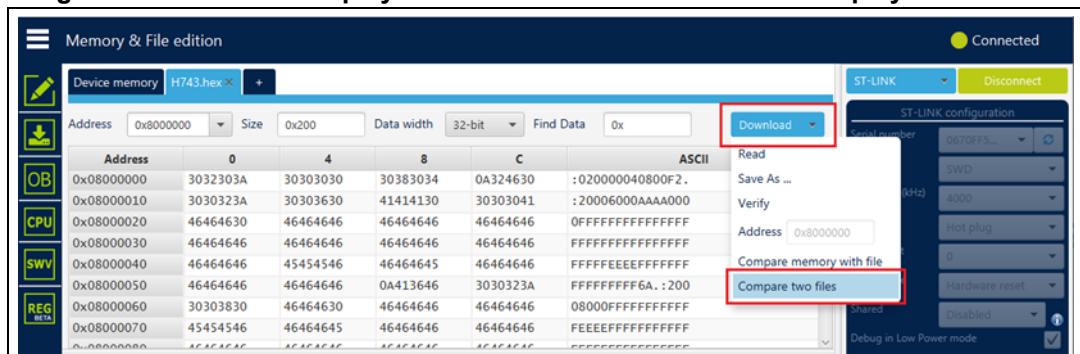


Figure 75. Sub-menu displayed from “Download” combo-box displayed in file tab



Note: In addition to sub-menus to display this window, the user can open it directly by using the key combination **Ctrl+F**.

Example: Difference between two files with same type and different sizes

Figure 76. Data width: 32 bits

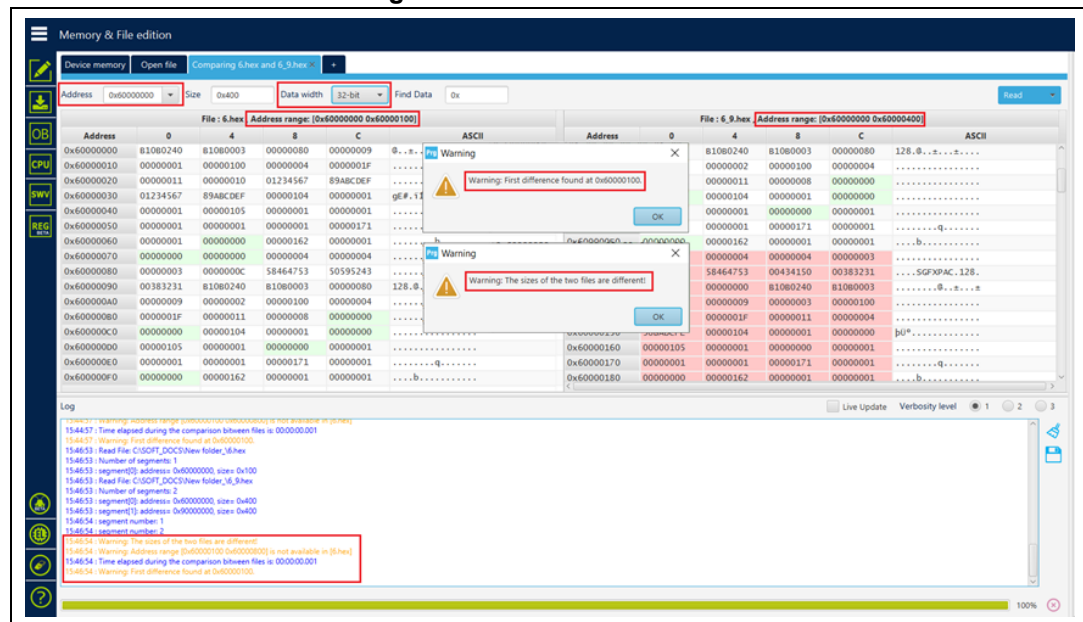


Figure 77. Data width: 16 bits

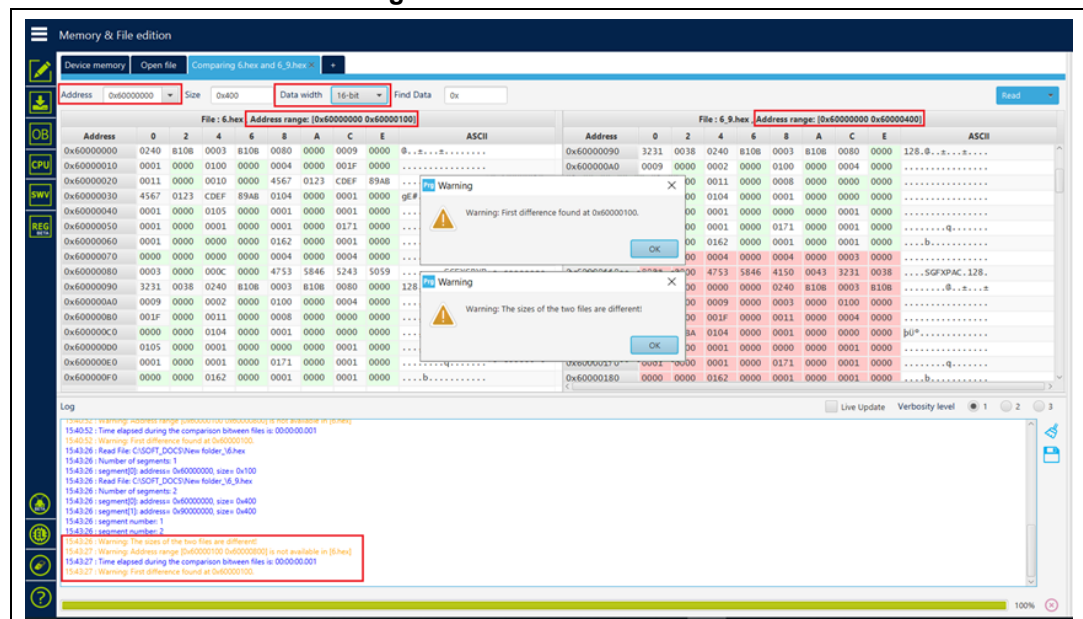
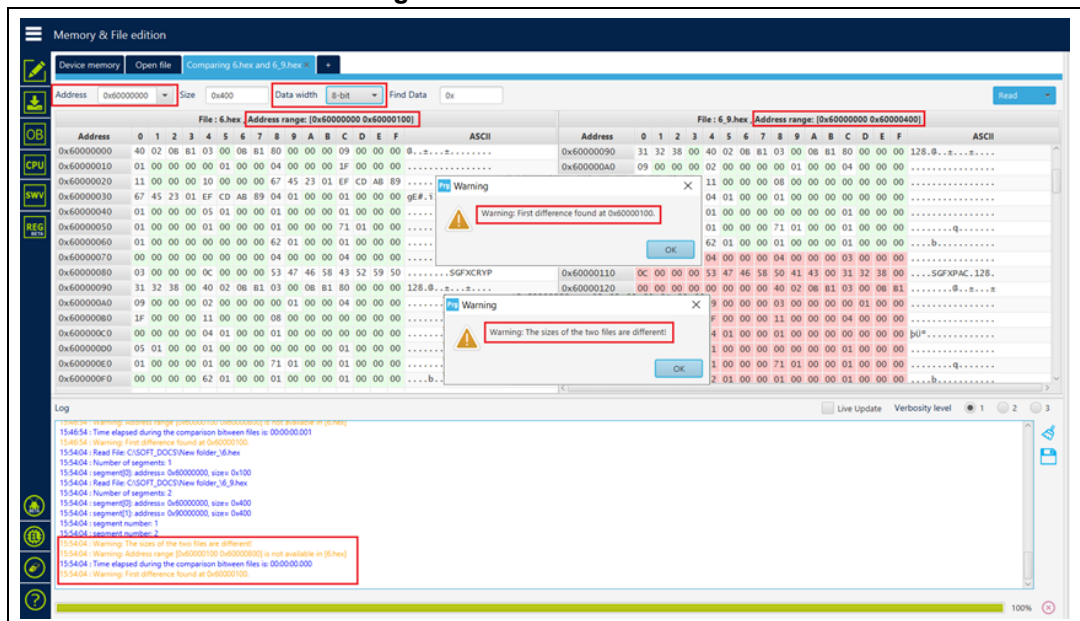
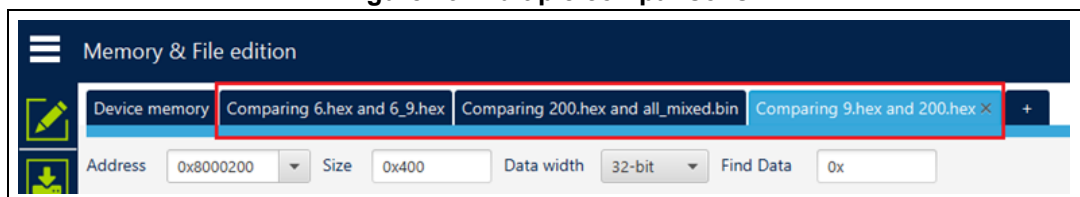


Figure 78. Data width: 8 bits



Note: The user can make multiple comparisons between files.

Figure 79. Multiple comparisons



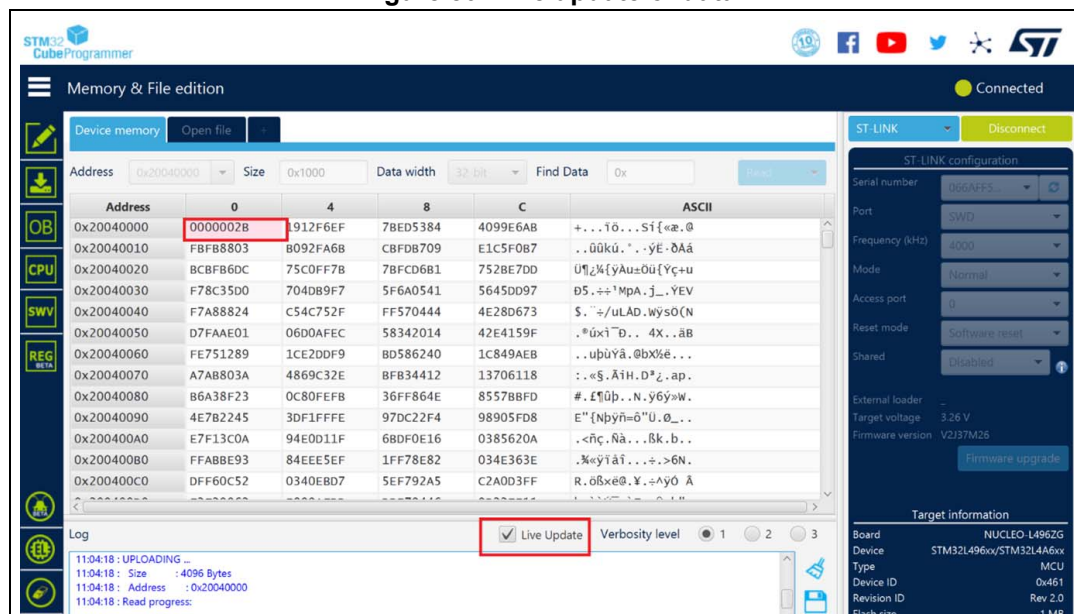
2.20 LiveUpdate feature

-liveUpdate checkbox

Description: When this feature is used the device memory grid is updated in real time and the modified data are colored in pink.

Once the device is connected, user can check the liveUpdate checkbox, memory data are updated in real time.

Figure 80. Live update of data



3 STM32CubeProgrammer command line interface (CLI) for MCUs

3.1 Command line usage

The following sections describe how to use the STM32CubeProgrammer from the command line. Available commands are shown in [Figure 81](#).

Note: To launch command line interface on macOS, call
`STM32CubeProgrammer.app/Contents/MacOs/bin/STM32_Programmer_CLI`.

Figure 81. STM32CubeProgrammer: available commands

```

Usage :
STM32_Programmer_CLI.exe [command_1] [arguments_1][[command_2] [arguments_2]...]

Generic commands:
-?, -h, --help           : Show this help
--version               : Displays the tool's version
-l, --list               : List all available communication interfaces
  <uart>                 : UART interface
  <usb>                   : USB interface
--quietMode              : Enable quiet mode. No progress bar displayed
--log                   : Store the detailed output in log file
  <[file_Path.log]>       : Path of the log file,
                          : default path = %HOME%\STM32Programmer\trace.log
-vb, --verbosity         : Specify verbosity level
  <Level>                 : Verbosity level, value in {1, 2, 3}

Available commands for STM32 MCU

--skipErase              : Skip sector erase before programming
-sl, --safelib            : Add a segment into a firmware file (elf, bin,
                          : hex, srec) containing computed CRC values
                          : to use only with the safety lib component
  <file_path>             : File path to be modified
  <start_address>         : Flash memory start address
  <end_address>           : Flash memory end address
  <slice_size>            : Size of data per CRC value
-ms, --mergesbfu          : Add a binary header and a sbfu segment to an elf file

  <elf_file_path>         : File path to be modified
  <header_file_path>      : Header file path
  <sbfu_file_path>        : SBFU file path
-e, --connect             : Establish connection to the device
  <port> <PortName>       : Interface identifier, ex COM1, /dev/ttyS0, usbl,
                          : JTAG, SWD...

UART port optional parameters:
  <br> <baudrate>            : Baudrate, ex: 115200, 9600, etc, default 115200
  <br> <parity>              : Parity bit, value in {NONE, ODD, EVEN}, default EVEN
  <br> <db> <data_bits>       : Data bit, value in {6, 7, 8}, default 8
  <br> <sb> <stop_bits>       : Stop bit, value in {1, 1.5, 2}, default 1
  <br> <fc> <flowControl>     : Flow control
                          : Value in {OFF, Hardware Software}, default OFF
  <br> <ininit> <noinit_bit> : Set No Init bits, value in {0,1}, default 0
  <br> <console>             : Enter UART console mode

JTAG/SWD debug port optional parameters:
  <br> <freq> <frequency>     : Frequency in KHz. Default frequencies:
                          : 4000 SWD 9000 JTAG with STM32MP
                          : 24000 SWD 21323 with STM32MP
  <br> <index> <index>       : Index of the debug probe, default index 0
  <br> <sn> <serialNumber>    : Serial Number of the debug probe
  <br> <ap> <accessPort>       : Access Port index to connect to, default ap 0
  <br> <mode> <mode>          : Connection mode, Value in {UR/HOTPLUG/NORMAL},
                          : default mode: NORMAL
  <br> <reset> <mode>         : Reset modes: Svrst/HVrst/Crst. Default mode: Svrst

SPI port optional parameters:
  <br> <br> <baudrate>           : Baudrate
  <br> <cpol> <cpol_val>      : Edge or 2Edge, default 1Edge
  <br> <cpol> <cpol_val>      : low or high
  <br> <crc> <crc_val>        : enable or disable {0/1}
  <br> <crcpol> <crc_pol>     : crc polynom value
  <br> <dataize> <dataize>   : 8bit/16bit
  <br> <direction> <val>     : Direction: 2LFullDuplex/2LRxOnly/1LRx/1LTX
  <br> <firstbit> <val>       : First Bit: MSB/LSB
  <br> <frameformat> <val>    : Frame Format: Motorola/TI
  <br> <mode> <val>           : Mode: master/slave
  <br> <inss> <val>           : NSS: soft/hard
  <br> <inspulse> <val>       : NSS pulse: Pulse/NoPulse
  <br> <delay> <val>         : Delay: Delay/NoDelay, delay of few microseconds
  <br> <ininit> <noinit_bit> : Set No Init bits, value in {0,1}, default 0

CAN port optional parameters:
  <br> <br> <baudrate>           : Baudrate : 125, 250, 500, 1000 Kbps, default 125
  <br> <mode> <canmode>       : CAN Mode : NORMAL, LOOPBACK..., default NORMAL
  <br> <type> <type>          : CAN type : STANDARD or EXTENDED, default STANDARD
  <br> <frameformat> <val>    : Frame Format: DATA or REMOTE, default DATA
  <br> <fifo> <afifo>         : Msg Receive : FIFO0 or FIFO1, default FIFO0
  <br> <filter> <val>        : Filter Mode : MASK or LIST, default MASK
  <br> <fe> <enable>         : Filter Scale: 16 or 32, default 32
  <br> <fe> <enable>         : Filter Activation : ENABLE or DISABLE, default ENABLE
  <br> <fbn> <fbanknb>       : Filter Bank Number : 0 to 13, default 0
  <br> <ininit> <noinit_bit> : Set No Init bits, value in {0,1}, default 0

I2C port optional parameters:
  <br> <add> <canadd>        : Slave address : address in hex format
  <br> <br> <baudrate>           : Baudrate : 100 or 400 Kbps, default 400
  <br> <mode> <mode>          : Speed Mode : STANDARD or FAST, default FAST
  <br> <analogfilter> <val>  : Address Mode : 7 or 10 bits, default 7
  <br> <analogfilter> <val>  : Analog filter : ENABLE or DISABLE, default ENABLE
  <br> <digitalfilter> <val> : Digital filter : ENABLE or DISABLE, default DISABLE
  <br> <dnf> <dnfilter>       : Digital noise filter : 0 to 15, default 0
  <br> <rt> <rtime>           : Rise time : 0-1000<STANDARD>, 0-300<FAST>, default 0
  <br> <ft> <ftime>           : Fall time : 0-300<STANDARD>, 0-300<FAST>, default 0
  <br> <ininit> <noinit_bit> : Set No Init bits, value in {0,1}, default 0

-e, --erase              : Erase memory pages/sectors devices:
                          : Not supported for STM32MP
  <br> <all>                  : Erase all sectors
  <br> <sectorsCodes>         : Erase the specified sectors identified by sectors
                          : codes, ex: 0, 1, 2 to erase sectors 0, 1 and 2
  <br> <[start end]>           : Erase the specified sectors starting from
                          : start code to end code, ex: -e 15 101

-w, --write               : Download the content of a file into device memory
-d, --download            : File path name to be downloaded: <bin, hex, srec,
  <file_path>             : elf, stm32 or tv file>
  <br> <[address]>             : Start address of download
-w32, --w32               : Write a 32-bits data into device memory
  <br> <[address]>             : Start address of download
  <br> <[32-bit_data]>        : 32-bit data to be downloaded
                          : values should be separated by space
-v, --verify              : Verify if the programming operation is achieved
                          : successfully
-r32, --r32               : Read a 32-bit data from device memory
  <br> <[address]>             : Read start address
  <br> <[size]>                : Size of data
-rst, --rst               : Reset system
-hardRst, --hardRst       : Hardware reset
                          : Available only with JTAG/SWD debug port
-halt, --halt             : Halt core
-stop, --stop             : Stop core
                          : Available only with JTAG/SWD debug port
-score, --score           : Get core status
                          : Available only with JTAG/SWD debug port
-coreReg, --coreReg        : Read/Write core registers
  <br> <[core_register]>       :
  <br> <[core_reg=<value>]>   : value in case of write operation
                          : Note: multiple registers can be handled at once
                          : Available only with JTAG/SWD debug port
-r, --read                : Read device memory content to a .bin file
-u, --upload              : Upload the device memory content to a .bin file
  <br> <[address]>             : Start address of read and upload
  <br> <[size]>                : Size of memory content to be read
  <br> <[file_path]>          : Binary file path
-el, --extload            : Select a custom external memory-loader
  <br> <[file_path]>          : External memory-loader file path
-s, --start               : Run the code at the specified address.
-g, --go                  : Start address
-rdu, --readunprotect      : Remove memory's Read Protection by shifting the RDP
                          : level from level 1 to level 0.
-ob, --optionbytes         : This command allows the user to manipulate the device
  <br> <[displ]>               : OptionBytes by displaying or modifying them.
                          : This option allows the user to display the whole set
                          : of Option Bytes.
  <br> <[OptByte=<value>]>    : This option allows the user to program the given
                          : Option Byte.

```

3.2 Generic commands

This section presents the set of commands supported by all STM32 MCUs.

3.2.1 Connect command

-c, --connect

Description: Establishes the connection to the device. This command allows the host to open the chosen device port (UART/USB/JTAG/SWD/SPI/CAN/I2C).

Syntax: `-c port=<Portname> [noinit=<noinit_bit>] [options]`

port=<Portname> Interface identifier, ex COMx (for Windows), /dev/ttySx for Linux), usbx for USB interface, SPI, I2C and CAN for, respectively, SPI, I2C and CAN interfaces.

[noinit=<noinit_bit>] Set No Init bits, value in {0, 1} ..., default 0. Noinit = 1 can be used if a previous connection is usually active.

- ST-LINK options

[freq=<frequency>] Frequency in kHz used in connection. Default value is 4000 kHz for SWD port, and 9000 kHz for JTAG port

Note: The entered frequency values are rounded to correspond to those supported by ST-LINK probe.

[index=<index>] Index of the debug probe. Default index value is 0.

[sn=<serialNumber>] Serial number of the debug probe. Use this option if you need to connect to a specific ST-LINK probe of which you know the serial number. Do not use this option with Index option in the same connect command.

[mode=<mode>] Connection mode. Value in {NORMAL/UR/HOTPLUG}. Default value is NORMAL.

Normal With 'Normal' connection mode, the target is reset then halted. The type of reset is selected using the 'Reset Mode' option.

UR The 'Connect Under Reset' mode enables connection to the target using a reset vector catch before executing any instructions. This is useful in many cases, for example when the target contains a code that disables the JTAG/SWD pins.

HOTPLUG The 'Hot Plug' mode enables connection to the target without a halt or reset. This is useful for updating the RAM addresses or the IP registers while the application is running.

[ap=<accessPort>] Access port index. Default access port value is 0.

[speed=] Connection speed. Default is Reliable. Available only for Cortex-M33.

Reliable Allows the user to connect with a slow mode.

Fast	Allows the user to connect with a fast mode.
[shared]	Enables shared mode allowing connection of two or more instances of STM32CubeProgrammer or other debugger to the same ST-LINK probe.
[tcpport=<Port>]	Selects the TCP Port to connect to an ST-Link Server. Shared option must be selected. Default value is 7184.
[dLPM / LPM]	Disable / enable the debug in Low Power mode (default configuration is enabled for the supported devices (STM32U5/ WB/L4 Series).

Note: Shared mode is supported only on Windows.

- USB options

The connection under the DFU interface supports two options, namely product and vendor ID (default values PID=0xDF11, VID=0x0483).

- SPI options

[br=<baudrate>] Baudrate (e.g. 187, 375, 750), default 375

Note: To use SPI on high speed, an infrastructure hardware must be respected to ensure the proper connection on the bus.

[cpha=<cpha_val>] 1Edge or 2Edge, default 1Edge
[cpol=<cpol_val>] Low or high, default low
[crc=<crc_val>] Enable or disable (0/1), default 0
[crcpol=<crc_pol>] CRC polynomial value
[datasize=<size>] 8- or 16-bit, default 8-bit
[direction=<val>] 2LFullDuplex/2LRxOnly/1LRx/1LTx
[firstbit=<val>] MSB/LSB, default MSB
[frameformat=<val>] Motorola/TI, default Motorola
[mode=<val>] Master/slave, default master
[nss=<val>] Soft/hard, default hard
[nsspulse=<val>] Pulse/NoPulse, default Pulse
[delay=<val>] Delay/NoDelay, default Delay

- I2C options

[add=<ownadd>] Slave address: address in hex format

Note: I2C address option must be always inserted, otherwise the connection is not established.

[br=<sbaudrate>] Baudrate: 100 or 400 Kbps, default 400 Kbps.
[sm=<smode>] Speed Mode, STANDARD or FAST, default FAST.
[am=<addmode>] Address Mode: 7 or 10 bits, default 7.
[af=<afilter>] Analog filter: ENABLE or DISABLE, default ENABLE.

[df=<dfilter>] Digital filter: ENABLE or DISABLE, default DISABLE.
 [dnf=<dnfilter>] Digital noise filter: 0 to 15, default 0.
 [rt=<rtime>] Rise time: 0-1000 (STANDARD), 0-300 (FAST), default 0.
 [ft=<ftime>] Fall time: 0-300 (STANDARD), 0-300 (FAST), default 0.

- CAN options

[br=<rbaudrate>] Baudrate: 125, 250..., default 125.
 [mode=<canmode>] Mode: NORMAL, LOOPBACK..., default NORMAL.

Note: The software must request the hardware to enter Normal mode to synchronize on the CAN bus and start reception and transmission between the Host and the CAN device. Normal mode is recommended.

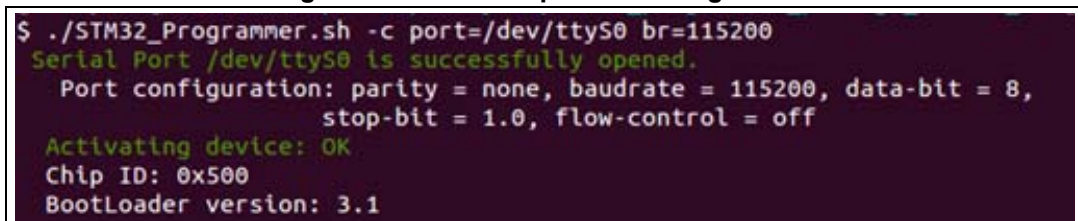
[ide=<type>] Type: STANDARD or EXTENDED, default STANDARD
 [rtr=<format>] Frame format: DATA or REMOTE, default DATA
 [fifo=<afifo>] Assigned FIFO: FIFO0 or FIFO1, default FIFO0
 [fm=<fmode>] Filter mode: MASK or LIST, default MASK
 [fs=<fscale>] Filter scale: 16 or 32, default 32
 [fe=<fenable>] Activation: ENABLE or DISABLE, default ENABLE
 [fbn=<fbanknb>] Filter bank number: 0 to 13, default 0

- Using UART

./STM32_Programmer.sh -c port=/dev/ttyS0 br=115200

The result of this example is shown in [Figure 82](#).

Figure 82. Connect operation using RS232



```
$ ./STM32_Programmer.sh -c port=/dev/ttyS0 br=115200
Serial Port /dev/ttyS0 is successfully opened.
Port configuration: parity = none, baudrate = 115200, data-bit = 8,
                    stop-bit = 1.0, flow-control = off
Activating device: OK
Chip ID: 0x500
BootLoader version: 3.1
```

STM32CubeProgrammer provides the possibility to configure RTS and DTR pins:

- RTS, used as follows: rts=low
- DTR, used as follows: dtr=high

Example: STM32_Programmer_CLI.exe -c port=COM27 dtr=high (see [Figure 83](#)).

Figure 83. Enabling COM DTR pin

```
Serial Port COM27 is successfully opened.  
Port configuration: parity = even, baudrate = 115200, data-bit = 8,  
                    stop-bit = 1.0, flow-control = off  
  
Timeout error occurred while waiting for acknowledgement.  
Activating device: OK  
Chip ID: 0x421  
BootLoader protocol version: 3.1  
Device name : STM32F446xx  
Flash size  : 512 KBytes (default)  
Device type  : MCU  
Device CPU   : Cortex-M4  
Dtr: High
```

Example using USB

./STM32_Programmer.sh -c port=usb1

The result of this example is shown in [Figure 84](#).

Figure 84. Connect operation using USB

Establishing connection with the target device					
USB speed	:	FULL_SPEED(12MBit/s)			
Manufacturer ID	:	STMicroelectronics			
Product ID	:	STM32_BOOTLOADER			
Serial number	:	326F37603234			
Firmware version	:	1.1a			
Device ID	:	0x0419			
AREA NAME	SECT.NBR	ADDRESS	SIZE	TYPE	
Internal Flash	0000	0x08000000	0016 KB	REW	
	0001	0x08004000	0016 KB	REW	
	0002	0x08008000	0016 KB	REW	
	0003	0x0800c000	0016 KB	REW	
	0004	0x08010000	0064 KB	REW	
	0005	0x08020000	0128 KB	REW	
	0006	0x08040000	0128 KB	REW	
	0007	0x08060000	0128 KB	REW	
	0008	0x08080000	0128 KB	REW	
	0009	0x080a0000	0128 KB	REW	
	0010	0x080c0000	0128 KB	REW	
	0011	0x080e0000	0128 KB	REW	
	0012	0x08100000	0016 KB	REW	
	0013	0x08104000	0016 KB	REW	
	0014	0x08108000	0016 KB	REW	
	0015	0x0810c000	0016 KB	REW	
	0016	0x08110000	0064 KB	REW	
	0017	0x08120000	0128 KB	REW	
	0018	0x08140000	0128 KB	REW	
	0019	0x08160000	0128 KB	REW	
	0020	0x08180000	0128 KB	REW	
	0021	0x081a0000	0128 KB	REW	
	0022	0x081c0000	0128 KB	REW	
	0023	0x081e0000	0128 KB	REW	
Option Bytes	0000	0x1fffc000	0016 B	RW	
	0001	0x1ffec000	0016 B	RW	
OTP Memory	0000	0x1fff7800	0512 B	RW	
	0001	0x1fff7a00	0016 B	RW	
Device Feature	0000	0xffff0000	0004 B	RW	

Note:

When using a USB interface, all the configuration parameters (e.g. baud rate, parity, data-bits, frequency, index) are ignored. To connect using a UART interface the port configuration (baudrate, parity, data-bits, stopbits and flow-control) must have a valid combination, depending on the used device.

Example using DFU IAP/USBx options

`/STM32_Programmer.sh -c port=usb1 pid=0xA38F vid=0x0438`

The result of this example is shown in [Figure 85](#).

Figure 85. Connect operation using USB DFU options

```

C:\Windows\System32\cmd.exe
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0\bin>STM32_Programmer_CLI.exe -c port=usb1
pid=0xA38F vid=0x0483
-----
STM32CubeProgrammer v2.6.0
-----
USB speed : Full Speed (12MBit/s)
Manuf. ID : STMicroelectronics
Product ID : DFU in FS Mode
SN : 306034663235
FW version : 0x011a
IAP
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0-A05\bin>

```

Note: The default value of product ID and vendor ID are ST products values (PID=0xDF11, VID=0x0483).

Example using JTAG/SWD debug port

To connect using port connection mode with ST-LINK probe it is necessary to mention the port name with at least the connect command (for example: `-c port=JTAG`).

Note: Make sure that the device being used contains a JTAG debug port when trying to connect through the JTAG.

There are other parameters used in connection with JTAG/SWD debug ports that have default values (see the Help menu of the tool for more information about default values).

The example below shows a connection example with an STM32 with device ID 0x415.

Figure 86. Connect operation using SWD debug port

```

ST-LINK SN : 066BFF574857847167114941
ST-LINK FW : U2J30M20
Voltage : 3.25V
SWD freq : 4000 KHz
Connect mode: Normal
Reset mode : Software reset
Device ID : 0x415
Device name : STM32L4x1/STM32L475xx/STM32L476xx/STM32L486xx
Device type : MCU
Device CPU : Cortex-M4

```

The corresponding command line for this example is `-c port=SWD freq=3900 ap=0`

In the connect command (`-c port=SWD freq=3900 ap=0`)

- The <port> parameter is mandatory.
- The index is not mentioned in the command line. The Index parameter takes the default value 0.
- The frequency entered is 3900 kHz, however the connection is established with 4000 kHz. This is due to the fact that ST-LINK probe has fixed values with SWD and JTAG debug ports.
- ST-LINK v2/v2.1
 - SWD (4000, 1800, 950, 480, 240, 125, 100, 50, 25, 15, 5) kHz
 - JTAG (9000, 4500, 2250, 1125, 562, 281, 140) kHz
- ST-LINK v3
 - SWD (24000, 8000, 3300, 1000, 200, 50, 5)
 - JTAG (21333, 16000, 12000, 8000, 1777, 750)

If the value entered does not correspond to any of these values, the next highest one is considered. Default frequency values are:

- SWD: STLinkV2: 4000 kHz, STLinkV3: 24000 kHz
- JTAG: STLinkV2: 9000 kHz, STLinkV3: 21333 kHz

Note: JTAG frequency selection is only supported with ST-LINK firmware versions from V2J23 onward.

To connect to access port 0 the ap parameter is used in this example, so any command used after the connect command is established through the selected access port.

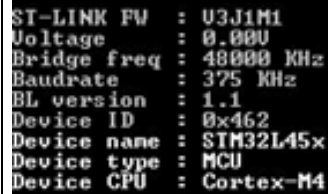
Note: The ST-LINK probe firmware version is shown when connecting to the device. Make sure that you have the latest version of ST-LINK firmware V2J28M17 (STSW-LINK007), available on www.st.com.

Example using SPI

```
STM32_Programmer_CLI -c port=SPI br=375 cpha=1edge cpol=low
```

The result of this example is shown in [Figure 87](#).

Figure 87. Connect operation using SPI port



```
ST-LINK FW : V3J1M1
Voltage    : 0.000
Bridge freq : 48000 KHz
Baudrate   : 375 KHz
BL version  : 1.1
Device ID  : 0x462
Device name : STM32L45x
Device type : MCU
Device CPU  : Cortex-M4
```

Note: Make sure that the device being used supports a SPI bootloader when trying to connect through the SPI.

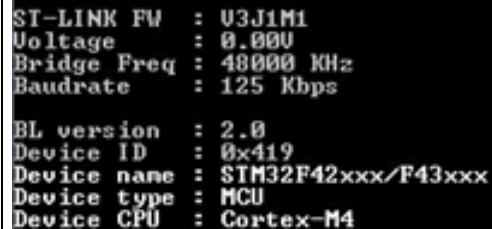
There are other parameters used in connection with SPI port that have default values, and some others must have specific values (see the help menu of the tool for more information).

Example using CAN

```
STM32_Programmer_CLI -c port=CAN br=125 fifo=fifo0 fm=mask fs=32
fe=enable fbn=2
```

The result of this example is shown in [Figure 88](#).

Figure 88. Connect operation using CAN port



```
ST-LINK FW : V3J1M1
Voltage    : 0.000
Bridge Freq : 48000 KHz
Baudrate   : 125 Kbps
BL version  : 2.0
Device ID   : 0x419
Device name : STM32F42xxx/F43xxx
Device type : MCU
Device CPU  : Cortex-M4
```

Note: Not all devices implement this feature, make sure the one you are using supports a CAN bootloader.

There are other parameters used in connection with CAN port that have default values and some others must have specific values (see the help menu of the tool for more information).

Example using I2C

STM32_Programmer_CLI -c port=I2C add=0x38 br=400 sm=fast

In the connect command:

- The parameter <add> changes from a device to another, refer to AN2606 to extract the correct one. In this case, the STM32F42xxx has a bootloader address equal to 0x38.
- The baudrate parameter
 depends directly on the speed mode parameter <sm>, for example, if sm=standard then the baudrate does not support the value 400.

The result of this example is shown in [Figure 89](#).

Figure 89. Connect operation using I2C port

```

ST-LINK FW : U3J1M1
Voltage    : 0.00V
Bridge freq : 192000 KHz
Baudrate   : 400 KHz
BL version  : 1.1
Device ID   : 0x419
Device name : STM32F42xxx/F43xxx
Device type : MCU
Device CPU  : Cortex-M4
  
```

Note: For each I2C connection operation the address parameter is mandatory.

Note: Not all devices implement this feature, make sure that the device supports an I2C bootloader.

There are other parameters used in connection with I2C port that have default values and some others must have specific values (see the help menu of the tool for more information).

Note: For the parallel programming of more than one STM32 device using multiple instances of STM32CubeProgrammer, it is mandatory to add the serial number of each device in the suitable instance, as shown in the following example:

- “-c port=swd/usb sn=SN1” (instance 1 of STM32CubeProgrammer)
- “-c port=swd/usb sn=SN2” (instance 2 of STM32CubeProgrammer)
- “-c port=swd/usb sn=SN3” (instance 3 of STM32CubeProgrammer)

3.2.2 Erase command

-e, --erase

Description: According to the given arguments, this command can be used to erase specific sectors or the whole Flash memory. This operation can take a second or more to complete, depending on the involved size.

Syntax:

[all]	Erase all sectors. EEPROM area is excluded.
[<sectorsCodes>]	Erase the sectors identified by codes (e.g. 0, 1, 2 to erase sectors 0, 1 and 2). For EEPROM: ed1 & ed2 .
[<[start end]>]	Erase the specified sectors starting from start code to end code, e.g. -e [5 10] .

Example

```
./STM32_Programmer.sh --connect port=/dev/ttyS0 -e 2 4
```

This command erases only sectors 2 and 4.

Note: *In the case of multiplicity of external loaders, the first selected is the one that will be taken into account during erasing of the external memory.*

3.2.3 Download command

-w, --write, -d, --download

Description: Downloads the content of the specified binary file into device memory. The download operation is preceded by the erase operation before the Flash memory is downloaded. A write address is only needed to download binary files.

Syntax: `-w <file_path> [start_address]`

`[file_path]` Path of the file to be downloaded

`[start_address]` Start address of download

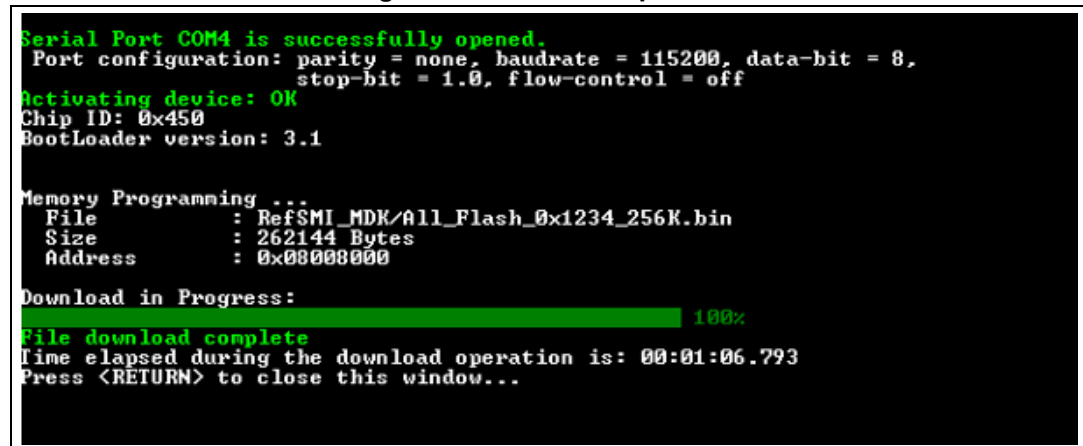
Example

```
-c port=COM4 -w RefSMI_MDK/All_Flash_0x1234_256K.bin 0x08008000
```

This command programs the binary file "All_Flash_0x1234_256K.bin" at address 0x08008000.

The result of this example is shown in [Figure 90](#).

Figure 90. Download operation



```
Serial Port COM4 is successfully opened.
Port configuration: parity = none, baudrate = 115200, data-bit = 8,
                  stop-bit = 1.0, flow-control = off
Activating device: OK
Chip ID: 0x450
BootLoader version: 3.1

Memory Programming ...
File       : RefSMI_MDK/All_Flash_0x1234_256K.bin
Size      : 262144 Bytes
Address   : 0x08008000

Download in Progress:
[Progress Bar] 100%
File download complete
Time elapsed during the download operation is: 00:01:06.793
Press <RETURN> to close this window...
```

Note: *To verify that the download has been successful, call the verify option (-v or --verify) just after the write command, otherwise the verify option is ignored.*

3.2.4 Download 32-bit data command

-w32

Description: Downloads the specified 32-bit data into Flash memory starting from a specified address.

Syntax: `-w32 <start_address> <32_data_bits>`

<start_address> Start address of download.

<32_data_Bits> 32 data bits to be downloaded. Data must be separated by escape.

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 br=9600 -w32 0x08000000
0x12345678 0xAABBCCFF 0x12AB34CD -verify
```

Note: This command makes it possible to write the 32 data bits (0x12345678, 0xAABBCCFF, 0x12AB34CD) into the Flash memory starting from address 0x08000000.

3.2.5 Download 64-bit data command

-w64

Description: Downloads the specified 64-bit data into a destination address.

Syntax: **-w64 <start_address> <64-bit_data>**

<start_address> Start address of download.

<64_data_Bits> 64-bit data to be downloaded. Data must be separated by escape.

Example:

```
/STM32_Programmer_CLI.exe -c port=swd -w64 0x08000000 0x12345678AABBCCFF
```

3.2.6 Read command

-r, --read, -u, --upload

Description: Reads and uploads the device memory content into a specified binary file starting from a specified address.

Syntax: **--upload <start_address> <size> <file_path>**

<start_address> Start address of read.

<size> Size of memory content to be read.

<file_path> Binary file path to upload the memory content.

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 br=9600 --upload
0x20007000 2000 "/local/benayedh/Binaries/read2000.bin"
```

This command makes it possible to read 2000 bytes, starting from address 0x20007000, and uploads the content to a binary file `"/local/benayedh/Binaries/read2000.bin"`

-r32

Description: Read 32-bit data memory.

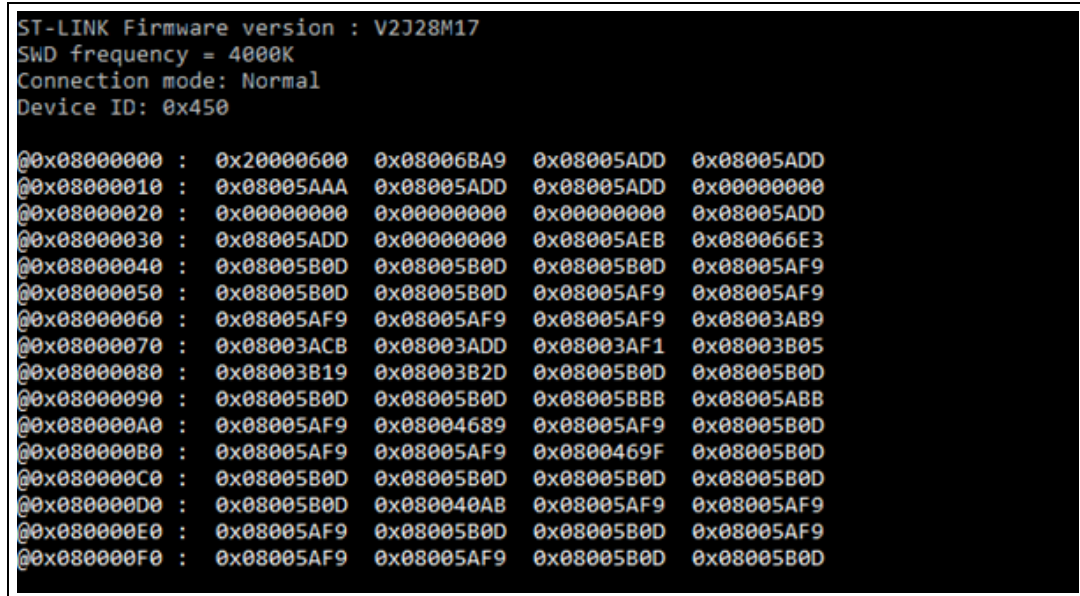
Syntax: **-r32 <start_address> <size>**

<start_address> Start address of read.
<size> Size of memory content to be read.

Example

```
./STM32_Programmer.sh -c port=SWD -r32 0x08000000 0x100
```

Figure 91. Read 32-bit operation



```
ST-LINK Firmware version : V2J28M17
SWD frequency = 4000K
Connection mode: Normal
Device ID: 0x450

0x08000000 : 0x20000600 0x08006BA9 0x08005ADD 0x08005ADD
0x08000010 : 0x08005AAA 0x08005ADD 0x08005ADD 0x00000000
0x08000020 : 0x00000000 0x00000000 0x00000000 0x08005ADD
0x08000030 : 0x08005ADD 0x00000000 0x08005AEB 0x080066E3
0x08000040 : 0x08005B0D 0x08005B0D 0x08005B0D 0x08005AF9
0x08000050 : 0x08005B0D 0x08005B0D 0x08005AF9 0x08005AF9
0x08000060 : 0x08005AF9 0x08005AF9 0x08005AF9 0x08003AB9
0x08000070 : 0x08003ACB 0x08003ADD 0x08003AF1 0x08003B05
0x08000080 : 0x08003B19 0x08003B2D 0x08005B0D 0x08005B0D
0x08000090 : 0x08005B0D 0x08005B0D 0x08005B8B 0x08005ABB
0x080000A0 : 0x08005AF9 0x08004689 0x08005AF9 0x08005B0D
0x080000B0 : 0x08005AF9 0x08005AF9 0x0800469F 0x08005B0D
0x080000C0 : 0x08005B0D 0x08005B0D 0x08005B0D 0x08005B0D
0x080000D0 : 0x08005B0D 0x080040AB 0x08005AF9 0x08005AF9
0x080000E0 : 0x08005AF9 0x08005B0D 0x08005B0D 0x08005AF9
0x080000F0 : 0x08005AF9 0x08005AF9 0x08005B0D 0x08005B0D
```

Note: The maximum size allowed with the `-r32` command is 32 Kbytes.

3.2.7 Start command

-g, --go, -s, --start

Description: This command enables execution of the device memory starting from the specified address.

Syntax: `--start [start_address]`

[start_address] Start address of application to be executed.

Example

```
./STM32_Programmer.sh --connect port=/dev/ttyS0 br=9600 --start 0x08000000
```

This command runs the code specified at 0x08000000.

3.2.8 Debug commands

The following commands are available only with the JTAG/SWD debug port.

-rst

Description: Executes a software system reset;

Syntax: `-rst`

-hardRst

Description: Generates a hardware reset through the RESET pin in the debug connector. The RESET pin of the JTAG connector (pin 15) must be connected to the device reset pin.

Syntax: `-hardRst`

-halt

Description: Halts the core.

Syntax: `-halt`

-step

Description: Executes one instruction.

Syntax: `-step`

-score

Description: Displays the Cortex-M core status.

The core status can be one of the following: 'Running', 'Halted', 'Locked up', 'Reset', 'Locked up or Kept under reset'

Syntax: `-score`

-coreReg

Description: Read/write Cortex-M core registers. The core is halted before a read/write operation.

Syntax: `-coreReg [<core_register>]`

`R0/. . /R15/PC/LR/PSP/MSP/XPSR/APSR/IPSR/EPSR/PRIMASK/BASEPRI / FAULTMASK/CONTROL`

`[core_reg=<value>]`: The value to write in the core register for a write operation. Multiple registers can be handled at once.

Example

`-coreReg` This command displays the current values of the core registers.

`-coreReg R0 R8` This command displays the current values of R0 and R8.

`-coreReg R0=5 R8=10` This command modifies the values of R0 and R8.

3.2.9 List command

-l, -list

Description: This command lists all available RS232 serial ports.

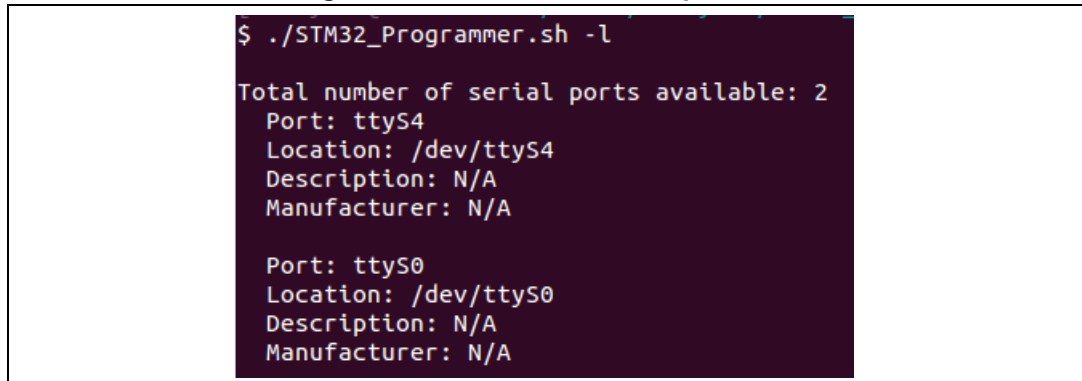
Syntax: `-l, --list`

Example

```
./STM32_Programmer.sh --list
```

The result of this example is shown in [Figure 92](#).

Figure 92. The available serial ports list



```
$ ./STM32_Programmer.sh -l

Total number of serial ports available: 2
Port: ttyS4
Location: /dev/ttyS4
Description: N/A
Manufacturer: N/A

Port: ttyS0
Location: /dev/ttyS0
Description: N/A
Manufacturer: N/A
```

Note: This command is not supported with JTAG/SWD debug port.

3.2.10 QuietMode command

-q, --quietMode

Description: This command disables the progress bar display during download and read commands.

Syntax: -q, --quietMode

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 br=115200 -quietMode -w
binaryPath.bin 0x08000000
```

3.2.11 Verbosity command

-vb, --verbosity

Description: This command makes it possible to display more messages, to be more verbose.

Syntax: -vb <level>

<level> : Verbosity level, value in {1, 2, 3} default value vb=1

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 br=115200 -vb 3
```

The result of this example is shown in [Figure 93](#).

Figure 93. Verbosity command

```
$ ./STM32_Programmer.sh -c port=/dev/ttyS0 br=115200 -vb 3
Serial Port /dev/ttyS0 is successfully opened.
  Port configuration: parity = none, baudrate = 115200, data-bit = 8,
                    stop-bit = 1.0, flow-control = off
Sending init command:
byte 0x7F sent successfully to target
Received response from target: 0x79
Activating device: OK
Sending GetID command and its XOR:
byte 0x02 sent successfully to target
byte 0xFD sent successfully to target
Received response from target: 0x79
Received response from target: 0x01050079
Chip ID: 0x500
Sending Get command and its XOR:
byte 0x00 sent successfully to target
byte 0xFF sent successfully to target
Received response from target: 0x79
Received response from target: 0x07
Received response from target: 0x07310001020311213179
BootLoader version: 3.1
```

3.2.12 Log command

-log, --log

Description: This traceability command makes it possible to store the whole traffic (with maximum verbosity level) into a log file.

Syntax: `-log [filePath.log]`

`[filePath.log]` Path of log file, default is `$HOME/.STM32CubeProgrammer/trace.log`.

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 br=115200 -log trace.log
```

The result of this example is shown in [Figure 94](#).

Figure 94. Log command

```
$ ./STM32_Programmer.sh -c port=/dev/ttyS0 br=115200 -log trace.log
Log output file:  trace.log
Serial Port /dev/ttyS0 is successfully opened.
  Port configuration: parity = none, baudrate = 115200, data-bit = 8,
                    stop-bit = 1.0, flow-control = off
Activating device: OK
Chip ID: 0x500
BootLoader version: 3.1
```

The log file trace.log contains verbose messages, as shown in [Figure 95](#).

Figure 95. Log file content

```
16:41:19:345
Log output file: trace.log
16:41:19:368 Serial Port /dev/ttyS0 is successfully opened.
16:41:19:368 Port configuration: parity = none, baudrate = 115200, data-bit = 8,
|stop-bit = 1.0, flow-control = off
16:41:19:368 Sending init command:
16:41:19:368 byte 0x7F sent successfully to target
16:41:19:369 Received response from target: 0x79
16:41:19:369 Activating device: OK
16:41:19:369 Sending GetID command and its XOR:
16:41:19:369 byte 0x02 sent successfully to target
16:41:19:369 byte 0xFD sent successfully to target
16:41:19:370 Received response from target: 0x79
16:41:19:370 Received response from target: 0x01050079
16:41:19:370 Chip ID: 0x500
16:41:19:370 Sending Get command and its XOR:
16:41:19:370 byte 0x00 sent successfully to target
16:41:19:370 byte 0xFF sent successfully to target
16:41:19:371 Received response from target: 0x79
16:41:19:371 Received response from target: 0x07
16:41:19:371 Received response from target: 0x07310001020311213179
16:41:19:371 BootLoader version: 3.1
```

3.2.13 External loader command

-el

Description: This command allows the path of one or more external memory loaders to be entered, to perform programming, write, erase and read operations with an external memory.

Syntax: `-el [externalLoaderFilePath1.stldr]` Absolute path of external loader file.

`-el [externalLoaderFilePath1.stldr]... -el [externalLoaderFilePath10.stldr]` Absolute path of external loader files.

Example 1:

```
./STM32_Programmer.sh -c port=swd -w "file.bin" 0x90000000 -v -el
"/local/user/externalLoaderPath.stldr"
```

Example 2:

```
./STM32_Programmer.sh -c port=swd -e all -el
"/local/user/externalLoaderPath.stldr"
```

Example 3:

```
./STM32_Programmer.sh -c port=swd -w "file.bin" 0x90000000 -v -el
"/local/user/externalLoaderPath1.stldr"
"/local/user/externalLoaderPath2.stldr"
```

Note: This command is only supported with SWD/JTAG ports.

Note: A maximum of ten external loaders can be used.

3.2.14 External loader command with bootloader interface

-elbl

Description: This command allows to provide the path of an external memory loader used to perform programming, write, erase and read operations with an external memory using bootloader interface (only in RSS/RSSe context). This command is used only when performing SFlx process.

Syntax: `-elbl [externalLoaderFilePath.stldr]` Absolute path of external loader file.

Example 1:

```
>STM32_Programmer_CLI.exe -c port=usb1 -elbl MX25LM51245G_STM32L552E-EVAL-SFIX-BL.stldr -sfi out.sfix hsm=0 license.bin -rsse
RSSe\L5\enc_signed_RSSe_sfi_jtag.bin
```

Note: This command is only supported with bootloader interface (UART/I2C/SPI/USB).

External loader for SFlx

The external loader for SFlx operation is aligned with the RSSe_SFI_CallNsFunction, as a result, all the functions used inside the external loader must have the same signature of this function.

```
rsse_sfi_ns_call_t
```

```
rsse_sfi_ns_call_t description in C coding language :
```

```
typedef uint32_t (*rsse_sfi_ns_call_t)(void * input_param);
```

As a consequence the implementation of these function inside the external loader must be slightly modified to be synchronized with input parameters.

Example of Sector erase function after modification:

```
KeepInCompilation int SectorErase (uint32_t *params)
{
    int result = 0;
    uint32_t BlockAddr;
    uint32_t EraseStartAddress = params[0];
    uint32_t EraseEndAddress = params[1];
```

3.2.15 Read unprotect command

-rdu, --readunprotect

Description: This command removes the memory read protection by changing the RDP level from level 1 to level 0.

Syntax: `--readunprotect`

Example

```
./STM32_Programmer.sh -c port=swd -rdu
```

3.2.16 TZ regression command

-tzenreg, --tzenregression

Description: This command removes TrustZone protection by disabling TZEN from 1 to 0.

Syntax: `--tzenregression`

Example

```
./STM32_Programmer.sh -c port=usb1 -tzenreg
```

Note: This command is only supported for bootloader interface and MCUs with trusted zone.

3.2.17 Option bytes command

-ob, --optionbytes

Description: This command allows the user to manipulate the device option bytes by displaying or modifying them.

Syntax: `-ob [displ] / -ob [OptByte=<value>]`

[displ]: Allows the user to display the whole set of option bytes.

[OptByte=<value>]: Allows the user to program the given option byte.

Example

```
./STM32_Programmer.sh -c port=swd -ob rdp=0x0 -ob displ
```

Note: For more information about the device option bytes, refer to the dedicated section in the programming manual and reference manual, both available on www.st.com.

3.2.18 Safety lib command

-sl, --safelib

Description: This command allows a firmware file to be modified by adding a load area (segment) containing the computed CRC values of the user program.

Supported formats: bin, elf, hex and Srec.

Syntax: `-sl <file_path> <start_address> <end_address> <slice_size>`

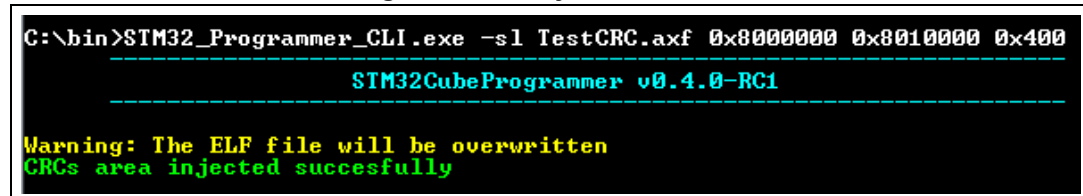
<file_path> The file path (bin, elf, hex or Srec)
<start_address> Flash memory start address
<end_address> Flash memory end address
<slice_size> Size of data per CRC value

Example

```
STM32_Programmer_CLI.exe -sl TestCRC.axf 0x8000000 0x8010000 0x400
```

The result is shown in [Figure 96](#).

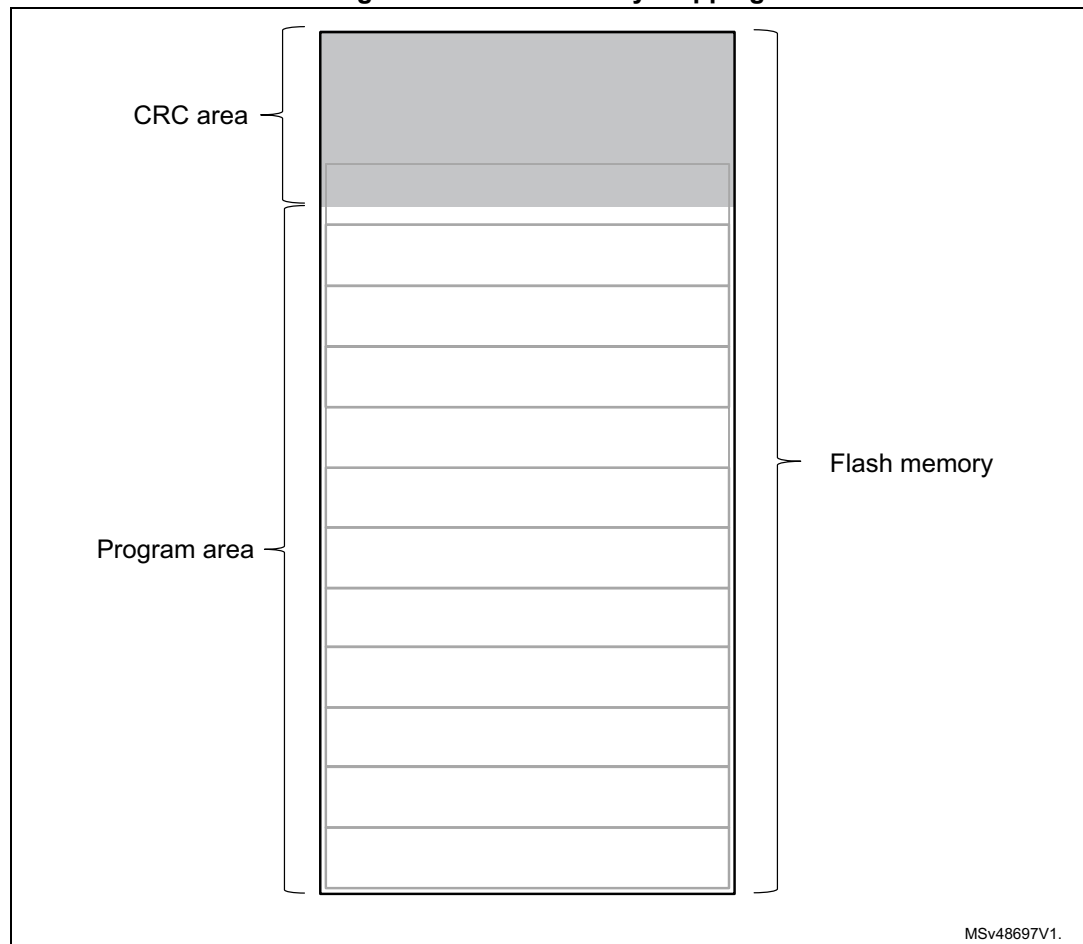
Figure 96. Safety lib command



```
C:\bin>STM32_Programmer_CLI.exe -sl TestCRC.axf 0x8000000 0x8010000 0x400
-----
STM32CubeProgrammer v0.4.0-RC1
-----
Warning: The ELF file will be overwritten
CRCs area injected succesfully
```

The Flash program memory is divided into slices, whose size is given as a parameter to the safety lib command as shown in the example above. For each slice a CRC value is computed and placed in the CRC area. The CRC area is placed at the end of the memory, as shown in [Figure 97](#).

Figure 97. Flash memory mapping



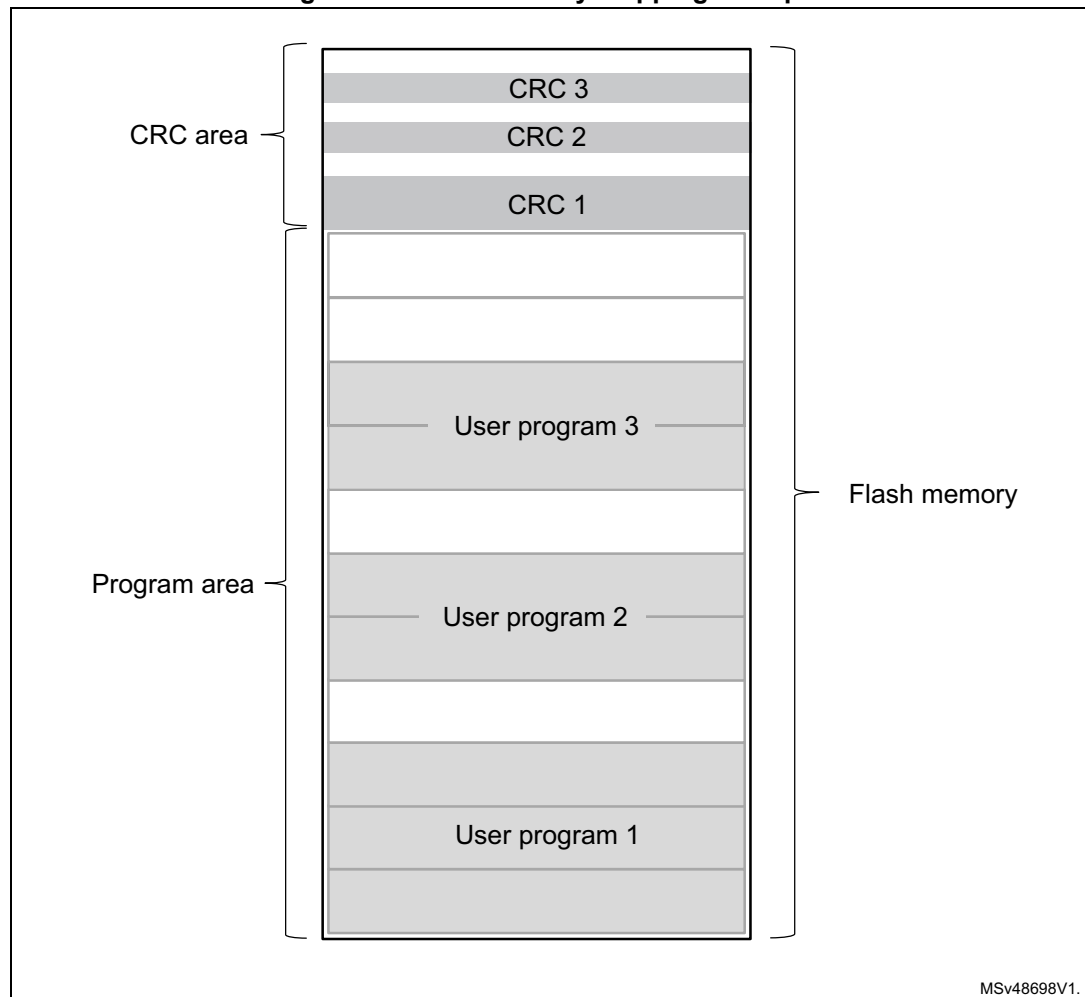
The address and size of the CRCs area are determined as follows:

$$\text{CRCs_Area_Size} = \text{Flash_Size} / \text{Slice_Size} * 4 \text{ bytes}$$

$$\text{CRCs_Start_Address} = \text{Flash_End_Address} - \text{CRCs_Area_Size}$$

The CRC values in the CRC area are placed according to the position(s) of the user program in the Flash memory, see [Figure 98](#).

Figure 98. Flash memory mapping example



The address of a CRCs region inside the CRCs area is calculated as:

$$@ = \text{CRCs_Start_Address} + \left(\frac{\text{UserProg_Start_Address} - \text{Flash_Start_Address}}{\text{Slice_Size}} \cdot 4 \text{ bytes} \right)$$

3.2.19 Secure programming SFI specific commands

Secure firmware install (SFI) is a feature supporting secure firmware flashing, available on some STM32 devices. The firmware provider has the possibility to protect its internal firmware against any illegal access, and to control the number of devices that can be programmed.

The protected firmware installation can be performed using different communication channels, such as JTAG/SWD or bootloader interfaces (UART, SPI and USB).

For more details refer to *Secure programming using STM32CubeProgrammer* (AN5054), available on www.st.com.

-sfi, --sfi

Description: Programs an sfi file

Syntax: `-sfi [<protocol=Ptype>] <.sfi file_path> [hsm=0|1]
<lic_path|slot=slotID> [<licMod_path>|slot=slotID]`

<code>[<protocol=Ptype>]</code>	Protocol type to be used: static/live (only static protocol is supported so far), default: static.
<code><file_path></code>	Path of sfi file to be programmed.
<code>[hsm=0 1]</code>	Sets user option for HSM use value {0 (do not use HSM), 1 (use HSM)}, default: hsm=0.
<code><lic_path slot=slotID></code>	Path to the SFI license file (if hsm=0) or reader slot ID if HSM is used (hsm=1).
<code>[<licMod_path> slot=slotID]</code>	List of the integrated SMI license files paths if HSM is not used (hsm=0) or readers slot IDs list if HSM is used (hsm=1). Used only in combined case, the list order must correspond to modules integration order within the SFI file.

-rsse, --rsse

Description: This command allows the user to select the root secure services extension library (RSSe). Mandatory for devices using RSSe to make secure firmware install (SFI). The RSSe binary file can be found in STM32CubeProgrammer bin/RSSe folder.

Syntax: `-rsse <file_path>`

`<file_path>` Path of RSSe file

-a, --abort

Description: This command allows the user to clean a not properly finished process. The currently ongoing operation stops and the system returns to idle state.

Syntax: `-a`

3.2.20 Secure programming SFlx specific commands

Secure firmware install (SFlx) is a feature supporting secure external firmware flashing, available on some STM32 devices with OTFDEC capability. The firmware provider has the

possibility to protect its external firmware/data against any illegal access, and to control the number of devices that can be programmed.

The SFlx secure programming can be carried out only with JTAG/SWD interface.

For more details refer to AN5054 *Secure programming using STM32CubeProgrammer*.

-sfi, --sfi

Description: Programs an sfix file

Syntax: `-sfi [<protocol=Ptype>] <.sfix file_path> [hsm=0|1]
<lic_path|slot=slotID> [<licMod_path>|slot=slotID]`

<code>[<protocol=Ptype>]</code>	Protocol type to be used: static/live (only static protocol is supported so far), default: static.
<code><file_path></code>	Path of sfi file to be programmed.
<code>[hsm=0 1]</code>	Sets user option for HSM use value {0 (do not use HSM), 1 (use HSM)}, default: hsm=0.
<code><lic_path slot=slotID></code>	Path to the SFI license file (if hsm=0) or reader slot ID if HSM is used (hsm=1).
<code>[<licMod_path> slot=slotID]</code>	List of the integrated SMI license file paths if HSM is not used (hsm=0) or readers slot IDs list if HSM is used (hsm=1). Used only in combined case, the list order must correspond to modules integration order within the SFI file.

-elbl --extload Selects a custom external memory-loader, only for the JTAG/SWD interfaces

`<file_path>` External memory-loader file path

-elbl --extloadbl Selects a custom external memory-loader for the bootloader interface

`<file_path>` External memory-loader file path

-rsse, --rsse

Description: This command allows the user to select the root secure services extension library (RSSe). Mandatory for devices using RSSe to make secure firmware install (SFI). The RSSe binary file can be found in STM32CubeProgrammer bin/RSSe folder.

Syntax: `-rsse <file_path>`

`<file_path>` Path of RSSe file

-a, --abort

Description: This command allows the user to clean a not properly finished process. The ongoing operation stops and the system returns to idle state.

Syntax: `-a`

Note: The ExternalLoader is different for SFlx use case since some initializations are already done by RSS, and it is marked with –SFlx at the end of the External FlashLoader name.

3.2.21 HSM related commands

To control the number of devices that can be programmed ST offers a secure firmware flashing service based on HSM (hardware secure module) as a license generation tool to be deployed in the programming house.

Two HSM versions are available:

- HSMv1: static HSM, it allows the user to generate firmware licenses for STM32 secure programming of devices selected in advance.
- HSMv2: dynamic HSM, it is an updated version of the previous one, allows the generation of firmware licenses targeting STM32 secure programming of devices chosen via personalization data at the OEM site.

Before using the HSM, it must be programmed using Trusted Package Creator, this tool can program both versions with some specific input configurations, as detailed in UM2238.

For more details refer to AN5054 *Secure programming using STM32CubeProgrammer*.

-hsmgetinfo

Description: Reads the HSM available information

Syntax: **-hsmgetinfo** [**slot=<SlotID>**]

[**slot=<SlotID>**] Slot ID of the smart card reader
Default value: slot=1 (the PC integrated SC reader)

-hsmgetcounter

Description: Reads the current value of the license counter

Syntax: **-hsmgetcounter** [**slot=<SlotID>**]

[**slot=<SlotID>**] Slot ID of the smart card reader
Default value: slot=1 (the PC integrated SC reader)

-hsmgetfwid

Description: Reads the Firmware/Module identifier

Syntax: **-hsmgetfwid** [**slot=<SlotID>**]

[**slot=<SlotID>**] Slot ID of the smart card reader
Default value: slot=1 (the PC integrated SC reader)

-hsmgetstatus

Description: Reads the current card life-cycle state

Syntax: **-hsmgetstatus** [**slot=<SlotID>**]

[slot=<SlotID>] Slot ID of the smart card reader
Default value: slot=1 (the PC integrated SC reader)

-hsmgetlicense

Description: Gets a license for the current chip if counter is not null

Syntax: **-hsmgetlicense** <file_path> [slot=<SlotID>] [protocol=<Ptype>]

<file_path> File path into where the received license is stored

[slot=<SlotID>] Slot ID of the smart card reader
Default value: slot=1 (the PC integrated SC reader)

[<protocol=Ptype>] Protocol type to be used: static/live
Only static protocol is supported so far
Default value: static

-hsmgetlicensefromcertifbin, -hsmglfcb

Description: Gets a license for the current certificate binary file if counter is not null.

Syntax: **-hsmglfcb** <certif_file_path.bin> <license_file_path.bin>
[slot=<SlotID>] [protocol=<Ptype>]

<certif_file_path.bin> File path from which the input certificate is read.

<license_file_path.bin> File path where the received license is stored

[slot=<SlotID>] Slot ID of the smart card reader.
Default value: slot=1 (the PC integrated SC reader)

3.2.22 STM32WB specific commands

-antirollback

Description: Perform the antirollback operation

Syntax: **-antirollback**

-startfus

Description: Start the FUS

Syntax: **-startfus**

-getuid64

Description: Read the device unique identifier (UID)

Syntax: **-getuid64**

-fusgetstate

Description: Read the FUS state

Syntax: `-fusgetstate`

-fusopgetversion

Description: Read the FUS Operator version

Syntax: `-fusgetversion`

Note: FUS Operator version is not available via bootloader interfaces.

-fwdelete

Description: Delete the BLE stack firmware

Syntax: `-fwdelete`

-fwupgrade

Description: Upgrade of BLE stack firmware or FUS firmware.

Syntax: `-fwupgrade <file_path> <address> [firstinstall=0|1] [startstack=0|1] [-v]`

<code><file_path></code>	New firmware image file path
<code><address></code>	Start address of download
<code>[firstinstall=0 1]</code>	1 if it is the first installation, otherwise 0 Optional, default value <code>firstinstall=0</code>
<code>[-v]</code>	Verify if the download operation is achieved successfully before starting the upgrade

-startwirelessstack

Description: Start the wireless stack

Syntax: `-startwirelessstack`

-authkeyupdate

Description: Authentication key update

Syntax: `-authkeyupdate <file_path>`

<code><file_path></code>	Authentication key file path. This is the public key generated by STM32TrustedPackageCreator when signing the firmware using <code>-sign</code> command.
--------------------------------	--

-authkeylock

Description: Authentication key lock

Once locked, it is no longer possible to change it using `-authkeyupdate` command

Syntax: `-authkeylock`

-wusrkey

Description: Customer key storage

Syntax: `-wusrkey <file_path> <keytype=1|2|3>`

`<file_path>`: customer key in binary format

`<keytype=1|2|3>`: User key type values: 1 (simple), 2 (master) or 3 (encrypted)

-startwirelessstack

Description: Starts the wireless stack

Syntax: `-startwirelessstack`

Note: These commands are available only through SWD, USB DFU and UART interfaces.

Note: Under Reset mode is mandatory.

Usage example for SWD interface

- FUS upgrade:
STM32_Programmer_CLI.exe -c port=swd mode=UR -ob nSWboot0=0 nboot1=1 nboot0=1 -fwupgrade stm32wb5x_FUS_fw.bin 0x080EC000 firstinstall=1
- Stack install:
STM32_Programmer_CLI.exe -c port=swd mode=UR -ob nSWboot0=0 nboot1=1 nboot0=1 -fwupgrade stm32wb5x_BLE_Stack_fw.bin 0x080EC000
- User application install:
STM32_Programmer_CLI.exe -c port=swd mode=UR -d UserApplication.bin 0x08000000 -v

Note: -antirollback command is available starting from FUS v1.2.0.

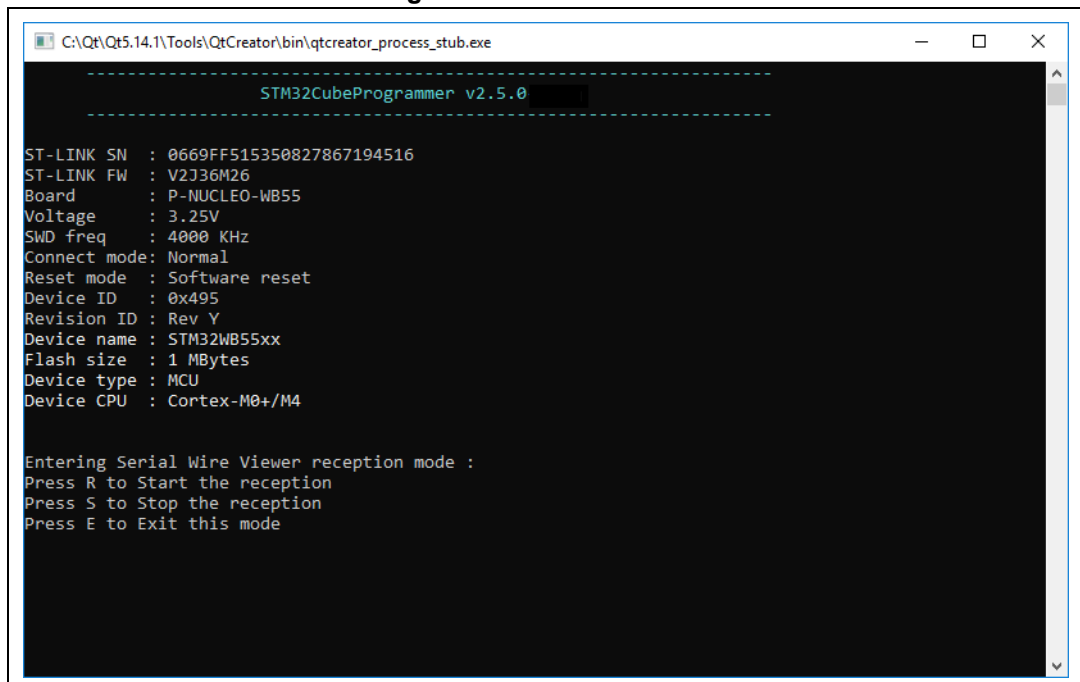
3.2.23 Serial wire viewer (SWV) command

-SWV

Description: This command allows the user to access the serial wire viewer console mode, which displays the printf data sent from the target through SWO.

In this mode (see [Figure 99](#)) the user can start and stop the reception of the SWO data by pressing, respectively, the “R” and “S” buttons on the keyboard. The received SWO data are displayed in the console. Pressing the “E” button allows the user to exit the serial wire viewer console mode and terminate the reception session.

Figure 99. SWV command



```

C:\Qt\Qt5.14.1\Tools\QtCreator\bin\qtcreator_process_stub.exe
-----
STM32CubeProgrammer v2.5.0
-----
ST-LINK SN : 0669FF515350827867194516
ST-LINK FW : V2J36M26
Board      : P-NUCLEO-WB55
Voltage    : 3.25V
SWD freq   : 4000 KHz
Connect mode: Normal
Reset mode : Software reset
Device ID  : 0x495
Revision ID : Rev Y
Device name : STM32WB55xx
Flash size : 1 MBytes
Device type : MCU
Device CPU  : Cortex-M0+/M4

Entering Serial Wire Viewer reception mode :
Press R to Start the reception
Press S to Stop the reception
Press E to Exit this mode

```

Syntax: `swv <freq=<frequency>> <portnumber=0-32> [<file_Path.log>]`

<freq=<frequency>> System clock frequency in MHz.

<portnumber=0-31 | all> ITM port number, values: 0-31, or "all" for all ports.

[<file_Path.log>] Path of the SWV log file (optional). If not specified default is: "\$USER_HOME/STMicroelectronics/STM32Programmer /SWV_Log/swv.log"

Example:

```
STM32_Programmer_CLI.exe -c port=swd -swv freq=32 portnumber=0
C:\Users\ST\swvLog\example.log
```

Note: The serial wire viewer is only available through SWD interface.

Note: Some SWV bytes can be lost during transfer due to ST-LINK hardware buffer size limitation.

3.2.24 Specific commands for STM32WL

Before performing the encrypted firmware installation, set the device in its default status, i.e. with security disabled (ESE = 0x0) and all the option bytes at their default values.

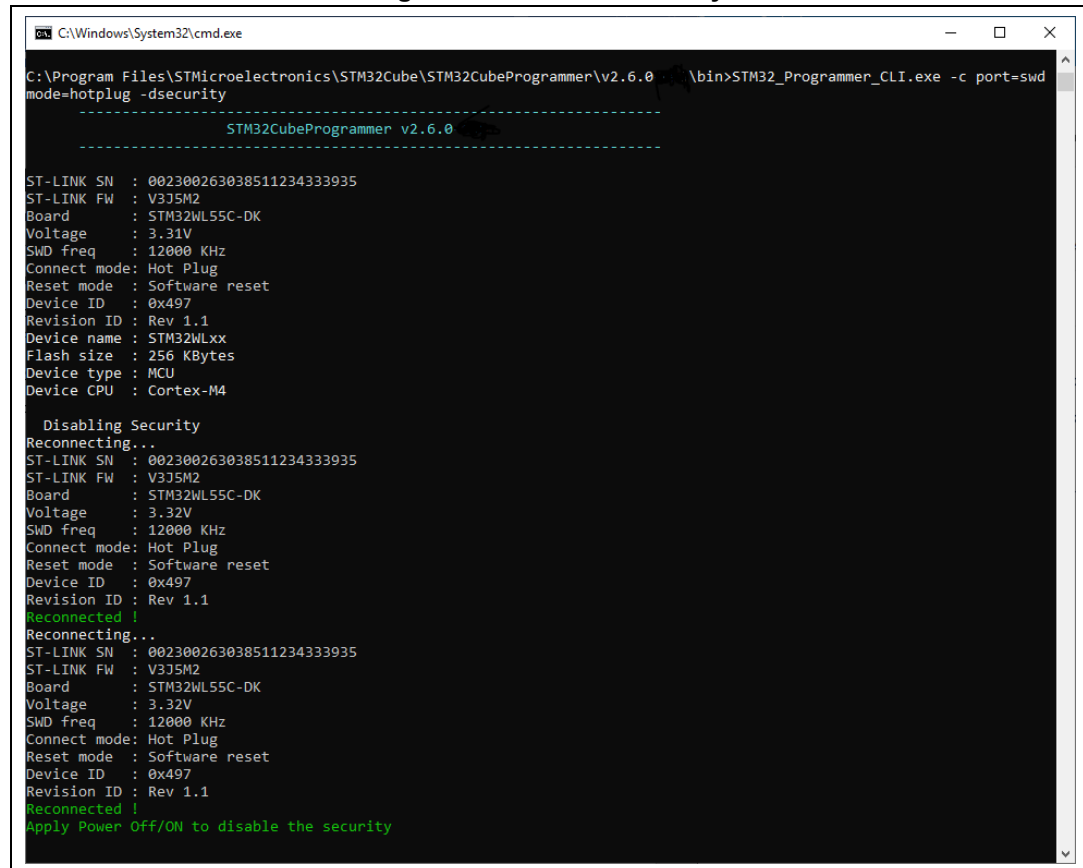
For this purpose STM32CubeProgrammer allows the user to perform these steps using two command lines:

1. **desurity**: allows the user to disable security.
Example: `STM32_Programmer_CLI.exe -c port=swd mode=hotplug -dsecurity`
2. **setdefaultob**: this command allows the user to configure option bytes to their default values.
Example: `STM32_Programmer_CLI.exe -c port=swd mode=hotplug -setdefaultob`

After the execution of these commands go through a power OFF / power ON sequence. These two commands allow the user to unlock the board in case of inability to change option bytes using the usual method.

Figure 100 and *Figure 101* show the results of these command lines.

Figure 100. Disable security



```
C:\Windows\System32\cmd.exe
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0\bin>STM32_Programmer_CLI.exe -c port=swd
mode=hotplug -dsecurity

-----
STM32CubeProgrammer v2.6.0
-----

ST-LINK SN : 002300263038511234333935
ST-LINK FW : V3J5M2
Board      : STM32WL55C-DK
Voltage    : 3.31V
SWD freq   : 12000 KHz
Connect mode: Hot Plug
Reset mode : Software reset
Device ID  : 0x497
Revision ID: Rev 1.1
Device name: STM32WLxx
Flash size : 256 KBytes
Device type: MCU
Device CPU : Cortex-M4

Disabling Security
Reconnecting...
ST-LINK SN : 002300263038511234333935
ST-LINK FW : V3J5M2
Board      : STM32WL55C-DK
Voltage    : 3.32V
SWD freq   : 12000 KHz
Connect mode: Hot Plug
Reset mode : Software reset
Device ID  : 0x497
Revision ID: Rev 1.1
Reconnected !
Reconnecting...
ST-LINK SN : 002300263038511234333935
ST-LINK FW : V3J5M2
Board      : STM32WL55C-DK
Voltage    : 3.32V
SWD freq   : 12000 KHz
Connect mode: Hot Plug
Reset mode : Software reset
Device ID  : 0x497
Revision ID: Rev 1.1
Reconnected !
Apply Power Off/ON to disable the security
```

Figure 101. Configure option bytes to their default values

```

C:\Windows\System32\cmd.exe
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0\bin>STM32_Programmer_CLI.exe -c port=swd
mode=hotplug -setdefaultob

-----
STM32CubeProgrammer v2.6.0
-----

ST-LINK SN : 002300263038511234333935
ST-LINK FW : V3J5M2
Board : STM32WL55C-DK
Voltage : 3.31V
SWD freq : 12000 KHz
Connect mode: Hot Plug
Reset mode : Software reset
Device ID : 0x497
Revision ID : Rev 1.1
Device name : STM32WLxx
Flash size : 256 KBytes
Device type : MCU
Device CPU : Cortex-M4

Set default OB for STM32WL
Reconnecting...
ST-LINK SN : 002300263038511234333935
ST-LINK FW : V3J5M2
Board : STM32WL55C-DK
Voltage : 3.31V
SWD freq : 12000 KHz
Connect mode: Hot Plug
Reset mode : Software reset
Device ID : 0x497
Revision ID : Rev 1.1
Reconnected !
Apply Power ON/Off to set default OB for STM32WL

C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0-A05\bin>

```

If the user locks the board and is unable to unlock it with these two commands, there are specific scripts to unlock it. These scripts are under “../bin/STM32WLScripts”, they contain a command line using `-wdbg` option to write directly scripts in the OPTR register.

The folder STM32Scripts contains two files and the Readme.txt:

1. “SetRDPLLevelCM0.bat” to unlock the board via Cortex M0+
2. “SetRDPLLevelCM4.bat” to unlock the board via Cortex M4

Note: *If SFI command finishes with a fail, the STM32WL chip must be set in its default status using the disable security command line (`-dsecurity`), then the set default option byte command line (`-setdefaultob`).*

3.2.25 SigFox credential commands

These commands are supported only for STM32WL devices.

-ssigfoxc

Description: This command allows to user to save the chip certificate to a binary file.

Syntax: `-ssigfoxc <binary_file_path>`

Example: `STM32_Programmer_CLI.exe -c port=swd -ssigfoxc “/local/user/chip_certif.bin”`

Figure 102. Example of -ssigfoxc command

```

ST-LINK SN   : 50FF6E067265575458302067
ST-LINK FW   : V2J37S7
Board       : --
Voltage     : 3.24V
SWD freq    : 4000 KHz
Connect mode: Normal
Reset mode  : Software reset
Device ID   : 0x497
Revision ID : Rev 1.1
Device name : STM32WLxx
Flash size  : 256 KBytes
Device type : MCU
Device CPU  : Cortex-M4

SigFox certificate File   : C:\test\sigfox.bin
Data read successfully
The Sigfox certificate file is saved successfully: C:\test\sigfox.bin

```

-wsigfoxc

Description: This command allows to user to write the chip certificate at address 0x0803E500

Syntax: **-wsigfoxc** <binary_file_path> <address> [The address is optional, by default is 0x0803E500]

Example 1: STM32_Programmer_CLI.exe -c port=swd -wsigfoxc
 "/local/user/sigfox_data.bin"0x0803E500

Figure 103. Example (1) of -wsigfoxc command

```

SigFox credential file : C:\SOFT_DOCS\KmsCredentials\sigfox_data.bin

Memory Programming ...
Opening and parsing file: sigfox_data.bin
File       : sigfox_data.bin
Size      : 48 Bytes
Address   : 0x0803E500

Erasing memory corresponding to segment 0:
Erasing internal memory sector 31
Download in Progress:
100%

File download complete
Time elapsed during download operation: 00:00:00.045

Verifying ...

Read progress:
100%

Download verified successfully

```

Example 2: STM32 Programmer CLI.exe -c port=swd -wsigfoxc "/local/user/sigfox data.h"

Figure 104. Example (2) of -wsigfoxc command

```
SigFox credential file : C:\SOFT_DOCS\KmsCredentials\sigfox_data.h

Memory Programming ...
Opening and parsing file: Sigfox_EmbKey.bin
  File       : Sigfox_EmbKey.bin
  Size      : 592 Bytes
  Address   : 0x0803E500

Erasing memory corresponding to segment 0:
Erasing internal memory sector 31
Download in Progress:
 100%

File download complete
Time elapsed during download operation: 00:00:00.052

Verifying ...

Read progress:
 100%

Download verified successfully
```

3.2.26 Register Viewer

-regdump

Description: Reads and dumps core and MCU registers

Syntax: `-regdump <file_path.log> [choice=<number>]`

<file path.log>	Log file path
-----------------	---------------

[choice=<number>] Device number from the list of compatible devices (optional). This list is displayed if the command is performed without this optional argument.

Example: STM32_Programmer_CLI.exe -regdump C:\test\STM32F072.log

Figure 105. Read core and MCU registers

```

C:\Program Files (x86)\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.7.0\bin>STM32_Programmer_CLI.exe -c port=
swd mode=hotplug -regdump C:\test\STM32F072.log

-----
STM32CubeProgrammer v2.7.0
-----

getDebugInterfaceInfo
this->index = 0
ST-LINK SN : 0675FF555354885087101431
ST-LINK FW : V2J32M22
Board : NUCLEO-F072RB
Voltage : 3.24V
SwD freq : 4000 KHz
Connect mode: Hot Plug
Reset mode : Software reset
Device ID : 0x448
Revision ID : Rev Z
Device name : STM32F07x
Flash size : 128 KBytes
Device type : MCU
Device CPU : Cortex-M0

You can automatically select a device from this list by adding the parameter "choice=<device_number>" to the command.
Please select your device number from the list:

1. STM32F0x0
2. STM32F0x1
3. STM32F0x2
4. STM32F0x8
1

Choice: STM32F0x0.svd

Getting the registers information...

Read progress:
[Progress bar] 100%

Registers information saved !

```

3.2.27 Hard fault analyzer

To start the analysis (see [Section 2.13](#)), use a specific command line.

Syntax: **-hf**

The output trace contains different kinds of essential information to better understand the reason(s) that caused a particular fault.

An informative message “STM32CubeProgrammer Fault Analyzer” is displayed to indicate that the detection flow has started.

Note: Connection to target must be established before performing Fault Analyzer command.

Example

Using the same example as GUI mode (division by 0).

Command: **-c port=swd mode=hotplug -hf**

From the command line output, a Green message indicates a “Hard Fault Detected” and “The processor has executed a SDIV or UDIV instruction with a divisor of 0”.

Useful informations can be extracted:

- Faulty instruction address: 0x80002E4
- Faulty instruction called by a function located at this address: 0x800022D
- NVIC position: 0, Window Watchdog interrupt
- Execution mode: Handler
- Core registers capture.

Figure 106. Fault Analyzer CLI view when Hard Fault detected

```

STM32CubeProgrammer Fault Analyzer

Core Registers :

r ap 0 reg 0 0x48000000
r ap 0 reg 1 0x00000020
r ap 0 reg 2 0x00000020
r ap 0 reg 3 0x00000020
r ap 0 reg 4 0x00000006
r ap 0 reg 5 0x00000000
r ap 0 reg 6 0x00000000
r ap 0 reg 7 0x00000000
r ap 0 reg 8 0x00000000
r ap 0 reg 9 0x00000000
r ap 0 reg 10 0x00000000
r ap 0 reg 11 0x00000000
r ap 0 reg 12 0x00000000
r ap 0 SP 13 0x20003E0
r ap 0 LR 14 0xFFFFF9
r ap 0 PC 15 0x000032E
r ap 0 XPSR 16 0x2100003
r ap 0 MSP - 0x20003E0
r ap 0 PSP - 0x0000000
r ap 0 CR 20 0x0000000

Execution Mode : Handler

Usage Fault detected in instruction located at 0x00002E4

FWIC position : 0

DIVBYZERO : The processor has executed a SDIV or UDIV instruction with a divisor of 0.

Hard Fault detected :

Faulty function called at this location 0x000022D

Hard Fault State Register information :

FORCED : forced Hard Fault.

Exception return information :

Return to Thread mode, exception return uses non-floating-point
state from MSP and execution uses MSP after return.

```

3.2.28 RDP regression with password

Some STM32 products (e.g. those of the STM32U5 Series) offer the possibility to use an optional password-based RDP level regression, including RDP Level 2.

Detailed information about this hardware mechanism is available in reference manuals.

Password lock and unlock CLI commands are:

- lockRDP1

Description: Allows the user to lock the RDP regression from level 1 with a password.

Syntax: - lockRDP1 <Password first 32 bits> <Password second 32 bits>

Example:

```
STM32_Programmer_CLI -c port=swd mode=hotplug -lockRDP1 0x12345678
0xDEADBEEF
```

- lockRDP2

Description: This command allows the user to lock the RDP regression from level 2 with a password.

Syntax: - lockRDP2 <Password first 32 bits> <Password second 32 bits>

Example:

```
STM32_Programmer_CLI -c port=swd mode=hotplug - lockRDP2 0x12345678 0xDEADBEEF
```

- unlockRDP1

Description: This command allows to unlock the RDP regression from level 1 with a password.

Syntax: - unlockRDP1 <Password first 32 bits> <Password second 32 bits>

Example:

```
STM32_Programmer_CLI -c port=swd mode=hotplug -unlockRDP1 0x12345678 0xDEADBEEF
```

- unlockRDP2

Description: This command allows the user to unlock the RDP regression from level 2 with a password.

Syntax: - unlockRDP2 <Password first 32 bits> <Password second 32 bits>

Example:

```
STM32_Programmer_CLI -c port=swd mode=hotplug - unlockRDP2 0x12345678 0xDEADBEEF
```

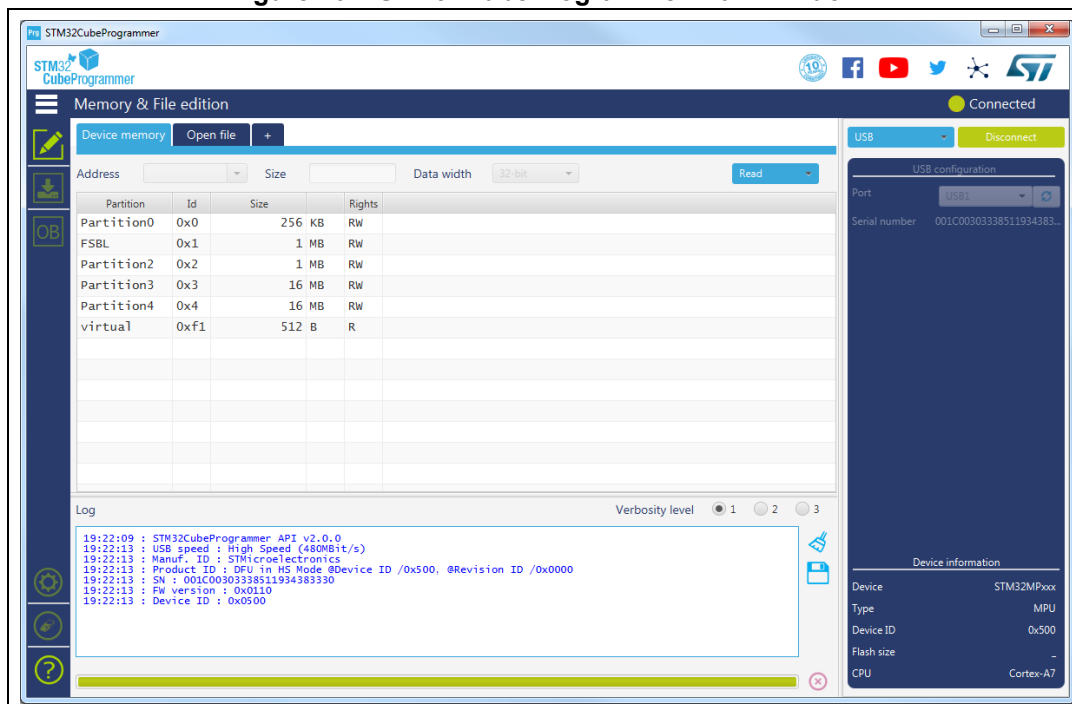
Note: After unlocking the RDP, the user must perform an RDP regression, as the listed commands do not include the RDP regression operation.

Note: To remove RDP regression with password, the user must use the Lock command and a password with value 0x00000000 0x00000000, such as **STM32_Programmer_CLI -c port=swd mode=hotplug -lockRDP1 0x00000000 0x00000000**

4 STM32CubeProgrammer user interface for MPUs

4.1 Main window

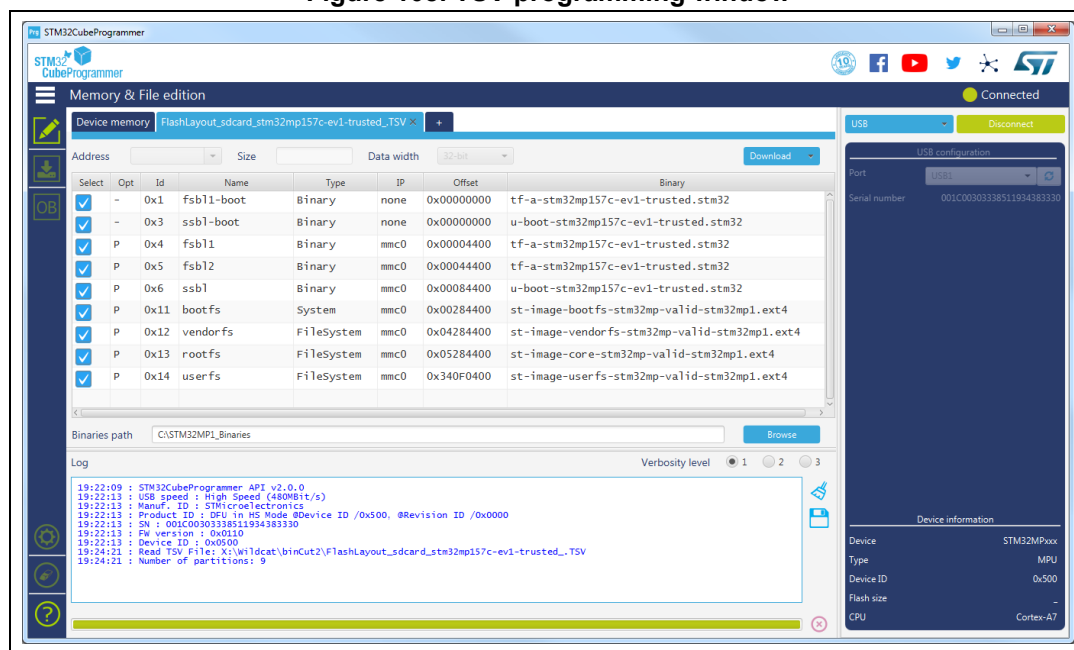
Figure 107. STM32CubeProgrammer main window



The main window allows the user to select the interface used to connect to STM32MP1 BootROM, possible interfaces are USB-DFU and UART (programming through stlink interface is not possible with STM32MP1 Series). Once connected (using connect button) available partitions are displayed, the user is able to open a TSV file for programming.

4.2 Programming windows

Figure 108. TSV programming window



To perform TSV files programming the user must perform the following operations:

- Open a TSV file by using “Open file” tab, if TSV file format is correct then TSV content is displayed in the main window. TSV Files are available in STM32MP1 Linux distributions, refer to STM32MP1 wiki for more details.
- Specify binaries path in “Binaries path” text box.
- Select the list of partitions to be programmed in “select” column, by default all partitions are selected.
- Launch download using “Download” button.

For more details concerning flashing operations refer to AN5275, available on www.st.com.

5 STM32CubeProgrammer CLI for MPUs

5.1 Available commands for STM32MP1

This section details the commands supported on STM32MP1 devices.

5.1.1 Connect command

-c, --connect

Description: Establishes the connection to the device. This command allows the host to open the chosen device port (UART/USB)

Syntax: `-c port=<Portname> [noinit=<noinit_bit>] [br=<baudrate>] [P=<Parity>] [db=<data_bits>] [sb=<stop_bits>] [fc=<flowControl>]`

<code>port=<Portname></code>	Interface identifier: – ex COMx (for Windows) – /dev/ttySx (for Linux) – usbx for USB interface
<code>[noinit=<noinit_bit>]</code>	Sets No Init bits, value in {0,1}, default 0. Noinit=1 can be used if a previous connection is active (no need to send 0X7F).
<code>[br=<baudrate>]</code>	Baudrate, (e.g. 9600, 115200), default 115200.
<code>[P=<Parity>]</code>	Parity bit, value in (EVEN, NONE, ODD), default EVEN.
<code>[db=<data_bits>]</code>	Data bit, value in (6, 7, 8), default 8.
<code>[sb=<stop_bits>]</code>	Stop bit, value in (1, 1.5, 2), default 1.
<code>[fc=<flowControl>]</code>	Flow control, value in (OFF, Software, Hardware). Software and Hardware flow controls are not yet supported for STM32MP1 Series, default OFF.

Example

Using UART:

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none
```

The result of this example is shown [Figure 109](#).

Figure 109. Connect operation using RS232

```

-----
STM32CubeProgrammer v1.0.2
-----
Serial Port COM1 is successfully opened.
Port configuration: parity = none, baudrate = 115200, data-bit = 8,
                  stop-bit = 1.0, flow-control = off
Activating device: OK
Chip ID: 0x500
BootLoader protocol version: 4.0

Device name: STM32MPxxx
Device type: MPU
Device CPU : Cortex_A7

```

Note: When using the USB interface, all the configuration parameters (such as baudrate, parity, data-bits, frequency, index) are ignored.

Note: To connect using UART interface, the port configuration (baudrate, parity, data-bits, stop-bits and flow-control) must have a valid combination.

5.1.2 GetPhase command

-p, --phaseID

Description: This command allows the user to know the next partition ID to be executed.

Syntax: --phaseID

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none br=115200 --phaseID
```

5.1.3 Download command

-w, --write, -d, --download

Description: Downloads the content of the specified binary file into a specific partition in the Flash or SYSRAM memories.

Syntax: -w <file_path> [partitionID]

[file_path] File path to be downloaded (bin, stm32, vfat, jffs2, ubi, ext2/3/4 and img file extensions).

[partition_ID] Partition ID to be downloaded.

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none -d atf.stm32 0x01
```

This command allows the user to download the atf binary file at Atf partition (partition ID: 0x01).

The result of this example is shown in [Figure 110](#).

Syntax: `--start [start_address/Partition_ID]`

[start_address] Start address of application to be executed. If not specified with STM32MP and UART interface, last loaded partition is started.

[Partition_ID] This parameter is needed only with STM32MP devices. It specifies the partition ID to be started.

Example

```
./STM32_Programmer.sh --connect port=/dev/ttyS0 p=none br=115200 --start 0x03
```

This command allows the user to run the code specified at partition 0x03.

Note: For U-boot with USB interface, to program the NVM with the loaded partition using download command, you need to execute a start command with the partition ID. To execute an application loaded in the NVM, you need to specify the start address.

Example 1: Download and manifestation on alternate 0x1

```
./STM32_Programmer.sh -c port=usb0 -w atf.stm32 0x01 -s 0x01
```

Example 2: Execute code at a specific address

```
./STM32_Programmer.sh -c port=usb0 -s 0xC0000000
```

5.1.6 Read partition command

-rp, --readPart

Description: Reads and uploads the specified partition content into a specified binary file starting from an offset address. This command is supported only by U-boot.

Syntax: `--readPart <partition_ID> [offset_address] <size> <file_path>`

<partition_ID> Partition ID

[offset_address] Offset address of read

<size> Size of memory content to be read

<file_path> Binary file path to upload the memory content

Example:

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none br=115200 -rp 0x01 0x200 0x1000 readPart1.bin
```

This command allows the user to read 0x1000 bytes from the sebl1 partition at offset address 0x200 and to upload its content to a binary file "readPart1.bin"

5.1.7 List command

-l, -list

Description: This command lists all available communication interfaces UART and USB.

Syntax: `-l, --list <interface_name>`

<uart/usb>: UART or USB interface

Example:

```
./STM32_Programmer.sh -list uart
```

5.1.8 QuietMode command

-q, --quietMode

Description: This command disables the progress bar display during Download and Read partition commands.

Syntax: **-q, --quietMode**

Example:

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none br=115200 --quietMode -w  
binaryPath.bin 0x01
```

5.1.9 Verbosity command

-vb, --verbosity

Description: This command allows the user to display more messages, to be more verbose.

Syntax: **-vb <level>**

<level> : Verbosity level, value in {1, 2, 3} default value vb=1

Example:

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none br=115200 -vb 3
```

5.1.10 Log command

-log, --log

Description: This traceability command allows the user to store the whole traffic (with maximum verbosity level) into log file.

Syntax: **-log [filePath.log]**

[filePath.log] : path of log file (default is \$HOME/.STM32CubeProgrammer/trace.log)

Example:

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none br=115200 -log trace.log
```

This command generates a log file “trace.log” containing verbose messages (see an example in [Figure 112](#)).

Figure 112. Log file content

```
16:41:19:345
Log output file:  trace.log
16:41:19:368 Serial Port /dev/ttyS0 is successfully opened.
16:41:19:368 Port configuration: parity = none, baudrate = 115200, data-bit = 8,
      stop-bit = 1.0, flow-control = off
16:41:19:368 Sending init command:
16:41:19:368 byte 0x7F sent successfully to target
16:41:19:369 Received response from target: 0x79
16:41:19:369 Activating device: OK
16:41:19:369 Sending GetID command and its XOR:
16:41:19:369 byte 0x02 sent successfully to target
16:41:19:369 byte 0xFD sent successfully to target
16:41:19:370 Received response from target: 0x79
16:41:19:370 Received response from target: 0x01050079
16:41:19:370 Chip ID: 0x500
16:41:19:370 Sending Get command and its XOR:
16:41:19:370 byte 0x00 sent successfully to target
16:41:19:370 byte 0xFF sent successfully to target
16:41:19:371 Received response from target: 0x79
16:41:19:371 Received response from target: 0x07
16:41:19:371 Received response from target: 0x07310001020311213179
16:41:19:371 BootLoader version: 3.1
```

5.1.11 OTP programming

Description: These commands allow the user to program the OTP from a host computer. The OTP programming commands functionalities, such as downloading or uploading a full OTP image and modifying an OTP value or proprieties, are explained below.

Note: The following commands are not supported in JTAG/SWD debug port connection mode.

- Loading shadow registers values to the tool:
For load operation, the host requests the OTP partition data and the platform replies with the structure described on https://wiki.st.com/stm32mpu/index.php/STM32CubeProgrammer_OTP_management.
- Writing the modified shadow registers to the target:
This operation is executed by performing the following sequence:
 - a) The user types in the value and the status of each chosen OTP shadow register.
 - b) The tool updates the OTP structure with the newly given OTP shadow registers values and status.
 - c) The tool proceeds with sending the updated structure, with bit0 in the "Write/read conf" field set to 0 ("Write/read conf" is word number 7 in the OTP structure).
 - d) Once the structure is sent, the shadow register values are reloaded to update the OTP structure in the tool.
- Programming the OTP with the modified shadow registers values:
once the user updates the OTP values and the OTP structure is refreshed, the host sends the OTP structure with bit0 in the "Write/read conf" field (word number 7 in the OTP structure) set to 1.
- Reloading the OTP value to the shadow registers:
once the OTP words are successfully programmed, the host uploads the OTP structure in order to update the OTP shadow registers. This operation allows the host to verify the status of the last SAFMEM programming via bit4 in the "Status" field.
- BSEC control register programming:
once the user updates the values of the given BSEC control register (Configuration, Debug configuration, Feature configuration and General lock configuration) the host updates the OTP structure and sends it to the device with bit0 in the "Write/read conf" field set to 0.
- OTP programming CLI:
the user is given a set of commands to perform a chosen sequence of operations on the OTP partition. Each one of these commands is described below.

5.1.12 Programming SAFMEM command

Description: This command allows the user to program SAFMEM memory by modifying the OTP words.

Syntax: `-otp program [wordID=(value)] [value=(value)] [sha_rsl=(value)] [sha_wsl=(value)] [sl=(value)] [pl=(value)]`

[wordID=(value)] This field contains the shadow register number (between 0 and 95). Value must be written in hexadecimal form.

[value=(value)] Loads value into the chosen OTP shadow register. Value must be written in hexadecimal form.

<code>[sha_rsl=(value)]</code>	Loads value into the corresponding shadow read sticky lock bit. Value can be either 0 or 1.
<code>[sha_wsl=(value)]</code>	Loads value into the corresponding shadow write sticky lock bit. Value can be either 0 or 1.
<code>[sl=(value)]</code>	Loads value into the corresponding programming sticky lock bit. Value can be either 0 or 1.
<code>[pl=(value)]</code>	Loads value into the corresponding programming permanent lock bit. Value can be either 0 or 1.

Example

```
./STM32_Programmer.sh --connect port=usb1 -otp program wordID=0x00
value=0x3f sl=1 wordID=0x08 value=0x18
```

5.1.13 Detach command

Description: This command allows the user to send detach command to USB DFU.

Syntax: `-detach`

5.1.14 GetCertif command

Description: This command allows the user to read the chip certificate.

Syntax: `-gc certification.bin`

5.1.15 Write blob command

Description: This command allows the user to send the blob (secrets and license).

Syntax: `-wb blob.bin`

5.1.16 Display command

Description: This command allows the user to display all or parts of the OTP structure.

Syntax: `-otp displ [upper] [lower] [ctrl]`

[upper] Option to display the loaded upper OTP shadow registers values and status.

[lower] Option to display the loaded lower OTP shadow registers values and status.

[ctrl] Option to display the loaded BSEC control registers.

Example

```
./STM32_Programmer.sh --connect port=usb1 -otp displ
```

5.2 Secure programming SSP specific commands

Secure secret provisioning (SSP) is a feature supporting secure secret flashing procedure, available on STM32 MPU devices. STM32MP1 Series supports protection mechanisms allowing the user to protect critical operations (such as cryptography algorithms) and critical data (such as secret keys) against unexpected accesses.

This section gives an overview of the STM32 SSP command with its associated tools ecosystem and explains how to use it to protect OEM secrets during the CM product manufacturing stage.

For more details refer to AN5054 *Secure programming using STM32CubeProgrammer*.

STM32CubeProgrammer exports a simple SSP command with some options to perform the SSP programming flow.

-ssp, --ssp

Description: Program an SSP file

Syntax: `-ssp <ssp_file_path> <ssp-fw-path> <hsm=0|1>
<license_path|slot=slotID>`

<code><ssp_file_path></code>	SSP file path to be programmed, bin or ssp extensions.
<code><ssp-fw-path></code>	SSP signed firmware path.
<code><hsm=0 1></code>	Set user option for HSM use (do not use / use HSM). Default value: hsm=0.
<code><license_path slot=slotID></code>	• Path to the license file (if hsm=0) • Reader slot ID if HSM is used (if hsm=1)

Example using USB DFU bootloader interface:

```
STM32_Programmer_CLI.exe -c port=usb1 -ssp "out.ssp" "tf-a-ssp-
stm32mp157f-dk2-trusted.stm32" hsm=1 slot=1
```

Note: All SSP traces are shown on the output console.

Figure 113. SSP successfully installed

```
Requesting Chip Certificate...
Get Certificate done successfully
requesting license for the current STM32 device
Init Communication ...
ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62000000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x62062FD8
P11 lib initialization Success!
Opening session with solt ID 1...
Succeed to Open session with reader solt ID 1
Succeed to generate license for the current STM32 device
Closing session with reader slot ID 1...
Session closed with reader slot ID 1
Closing communication with HSM...
Communication closed with HSM
Succeed to get License for Firmware from HSM slot ID 1
Starting Firmware Install operation...
Writing blob
Blob successfully written
Start operation achieved successfully
Send detach command
Detach command executed
SSP file out.ssp Install Operation Success
```

If there is any faulty input the SSP process is aborted, and an error message is displayed to indicate the root cause of the issue.

6 STM32CubeProgrammer C++ API

In addition to the graphical user interface and to the command line interface STM32CubeProgrammer offers a C++ API that can be used to develop your application and benefit of the wide range of features to program the memories embedded in STM32 microcontrollers, either over the debug interface or the bootloader interface (USB DFU, UART, I²C, SPI and CAN).

For more information about the C++ API, read the help file provided within the STM32CubeProgrammer package under API\doc folder.

7 Revision history

Table 2. Document revision history

Date	Revision	Changes
15-Dec-2017	1	Initial release.
02-Aug-2018	2	Updated: – Section 1.1: System requirements – Section 1.2.3: macOS install – Section 1.2.4: DFU driver Added: – Section 3.2.8: Debug commands – Figure 1: macOS “Allow applications downloaded from:” tab – Figure 2: Deleting the old driver software
12-Sep-2018	3	Added SPI, CAN and I2C settings on cover page and in Section 2.1.4: Target configuration panel . Updated: – Figure 6: ST-LINK configuration panel – Figure 81: STM32CubeProgrammer: available commands. – Figure 86: Connect operation using SWD debug port Replaced Section 3.2.1: Connect command .
16-Nov-2018	4	Updated Section 2.1.4: Target configuration panel , Section 2.2.1: Reading and displaying target memory , Section 2.2.2: Reading and displaying a file and Section 2.3.2: External Flash memory programming . Updated Figure 4: STM32CubeProgrammer main window , Figure 5: Expanded main menu , Figure 6: ST-LINK configuration panel , Figure 7: UART configuration panel , Figure 8: USB configuration panel , Figure 9: Target information panel , Figure 10: SPI configuration panel , Figure 11: CAN configuration panel , Figure 12: I2C configuration panel , Figure 13: Memory and file edition: Device memory tab , Figure 15: Memory and file edition: File display , Figure 16: Flash memory programming and erasing (internal memory) and Figure 17: Flash memory programming (external memory) . Minor text edits across the whole document.
03-Jan-2019	5	Updated Section 1.2.4: DFU driver . Added Section 3.2.19: Secure programming SFI specific commands , Section 3.2.21: HSM related commands and Section 6: STM32CubeProgrammer C++ API . Minor text edits across the whole document.
04-Mar-2019	6	Updated Introduction and Section 1: Getting started . Updated title of Section 2: STM32CubeProgrammer user interface for MCUs and of Section 3: STM32CubeProgrammer command line interface (CLI) for MCUs . Added Section 2.5: Automatic mode , Section 2.6: STM32WB OTA programming , Section 4: STM32CubeProgrammer user interface for MPUs , Section 5: STM32CubeProgrammer CLI for MPUs and their subsections.

Table 2. Document revision history (continued)

Date	Revision	Changes
19-Apr-2019	7	Updated Section 1.1: System requirements , Section 2.2.2: Reading and displaying a file , Section 2.6.2: OTA update procedure , Section 3.2.19: Secure programming SFI specific commands , Section 3.2.21: HSM related commands and Section 3.2.22: STM32WB specific commands . Updated Figure 16: Flash memory programming and erasing (internal memory) .
11-Oct-2019	8	Updated Graphical guide , Section 3.2.19: Secure programming SFI specific commands , Section 3.2.21: HSM related commands and Section 3.2.22: STM32WB specific commands . Added Section 2.6: In application programming (IAP/USBx) . Minor text edits across the whole document.
08-Nov-2019	9	Updated Section 1.2.1: Linux install , Section 3.2.22: STM32WB specific commands and Section 5.1.6: Read partition command . Minor text edits across the whole document.
07-Jan-2020	10	Updated Section 1.1: System requirements , Section 1.2.3: macOS install and Section 3.2.19: Secure programming SFI specific commands . Added Section 3.2.16: TZ regression command and Section 3.2.20: Secure programming SFlx specific commands . Removed former Section 5.2.12: Writing to BSEC command . Minor text edits across the whole document.
24-Feb-2020	11	Added Section 2.7: Flash the co-processor binary using graphical interface and its subsections.
23-Jul-2020	12	Added Section 2.8: Serial wire viewer (SWV) , Section 3.2.23: Serial wire viewer (SWV) command and Section 5.2: Secure programming SSP specific commands . Updated Section 3.2.1: Connect command and Section 3.2.2: Erase command . Minor text edits across the whole document.
17-Nov-2020	13	Updated Section 1.1: System requirements , Section 1.2.1: Linux install , Section 1.2.2: Windows install , Section 1.2.3: macOS install , Section 2.3.2: External Flash memory programming , Section 2.8: Serial wire viewer (SWV) , Section 3.2.1: Connect command , Section 3.2.2: Erase command , Section 3.2.13: External loader command , Section 3.2.21: HSM related commands , Section 3.2.20: Secure programming SFlx specific commands , Section 3.2.22: STM32WB specific commands and Section 5.1.1: Connect command . Added Section 2.10: DFU IAP/USBx with custom PID and VID , Section 2.11: SigFox™ credentials , Example using DFU IAP/USBx options , Section 3.2.5: Download 64-bit data command , Section 3.2.14: External loader command with bootloader interface , Section 3.2.24: Specific commands for STM32WL and Section 5.2.5: Flashing service via USB serial gadget . Updated Figure 17: Flash memory programming (external memory) , Figure 31: SWV window and Figure 66: Available commands for MPUs .
19-Nov-2020	14	Updated Section 5.1.1: Connect command . Removed former Section 5.1: Command line usage and Section 5.2.5: Flashing service via USB serial gadget .

Table 2. Document revision history (continued)

Date	Revision	Changes
11-Mar-2021	15	Updated Section 1.1: System requirements, Section 1.2.1: Linux install, Section 1.2.3: macOS install, Section 2.11: SigFox™ credentials and Section 3.2.22: STM32WB specific commands. Added Section 2.12: Register Viewer, Section 2.13: Hard Fault analyzer with its subsections, Section 3.2.26: Register Viewer and Section 3.2.27: Hard fault analyzer. Minor text edits across the whole document.
22-Jul-2021	16	Updated Section 2.1.4: Target configuration panel, Section 3.2.1: Connect command, Section 3.2.2: Erase command and Section 3.2.22: STM32WB specific commands. Added Section 2.14: Fill memory command, Section 2.15: Fill memory operation, Section 2.16: Blank check command, Section 2.17: Blank check operation, Section 2.18: Compare Flash memory with file, Section 2.19: Comparison between two files, Section 2.20: LiveUpdate feature and Section 3.2.28: RDP regression with password. Updated Figure 7: UART configuration panel and Figure 83: Enabling COM DTR pin. Added Figure 84: Connect operation using USB. Minor text edits across the whole document.
23-Nov-2021	17	Added Section 2.9: STM32CubeProgrammer Script Manager platform for MCUs and its subsections. Updated Section 2.1.1: Main menu, Section 2.1.4: Target configuration panel, Section 2.6: In application programming (IAP/USBx), Section 2.7: Flash the co-processor binary using graphical interface and its subsections, Section 2.9: STM32CubeProgrammer Script Manager platform for MCUs, Section 3.2.1: Connect command and Section 3.2.22: STM32WB specific commands. Removed former Section 2.6: STM32WB OTA programming. Updated Figure 4: STM32CubeProgrammer main window, Figure 5: Expanded main menu, Figure 6: ST-LINK configuration panel and Figure 9: Target information panel. Minor text edits across the whole document.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2021 STMicroelectronics – All rights reserved