UM2237
User manual

STM32CubeProgrammer software description

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, and supports a wide 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

List of tables

List of figures

1 Getting started

This section describes the requirements and procedures to install the STM32CubeProgrammer software tool, which 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® 7/8/10/11 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 the 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 software usage. The setup offers also the optional installation of the “STM32 Trusted Package Creator” tool, used to create secure firmware files for secure firmware install and update. For more information, refer to UM2238 “*STM32 Trusted Package Creator tool software description”*, 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.

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 this fix:

1. Right-Click on SetupSTM32CubeProgrammer-2.12.0
2. Select "Show Package Contents"
3. Navigate to Contents/MacOs
4. Launch SetupSTM32CubeProgrammer-2_12_0_macos

1.2.4 DFU driver

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

If the DFUSE driver is installed on your machine, first uninstall it, then reboot the machine and run the previously mentioned ".bat" file. 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

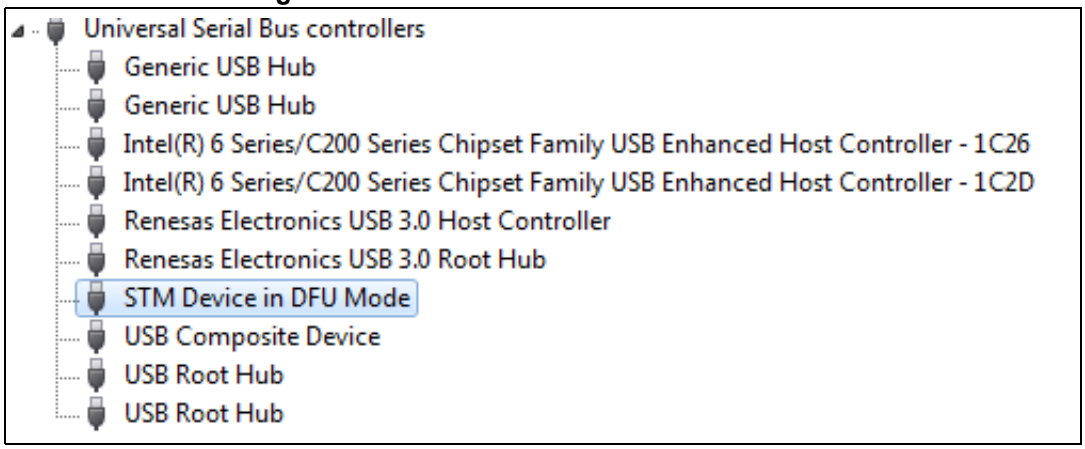
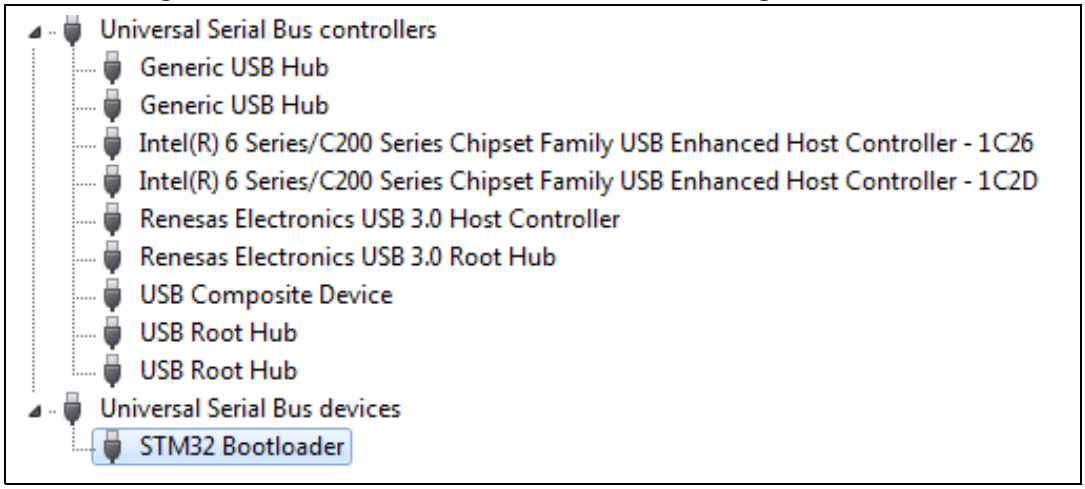


Figure 3. STM32 DFU device with STM32CubeProgrammer driver



Note: When using USB DFU interface or ST-LINK 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, ST-LINK-V3, or ST-LINKV3Power, 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.

1.3 Updater

STM32CubeProgrammer updater allows users to make automatic updates of the software and its associated packages. The updater is available in all supported operating systems, namely Windows 10/11, Linux, and macOS.

1.3.1 Update steps

1. Check the connection and update its settings if needed.
2. Check for updates.
3. Download the new version.
4. Install the downloaded version (the tool restarts once updated).

1.3.2 Proxy settings

The user can manually check the connection by using the "Proxy Settings" window opened with the submenu available in the help button (see *Figure 4*). Three settings are available (see *Figure 5*):

- No proxy
- Use the system parameters
- Use manual configuration of server: add the HTTP proxy name, port, and credentials

Figure 4. Proxy settings submenu

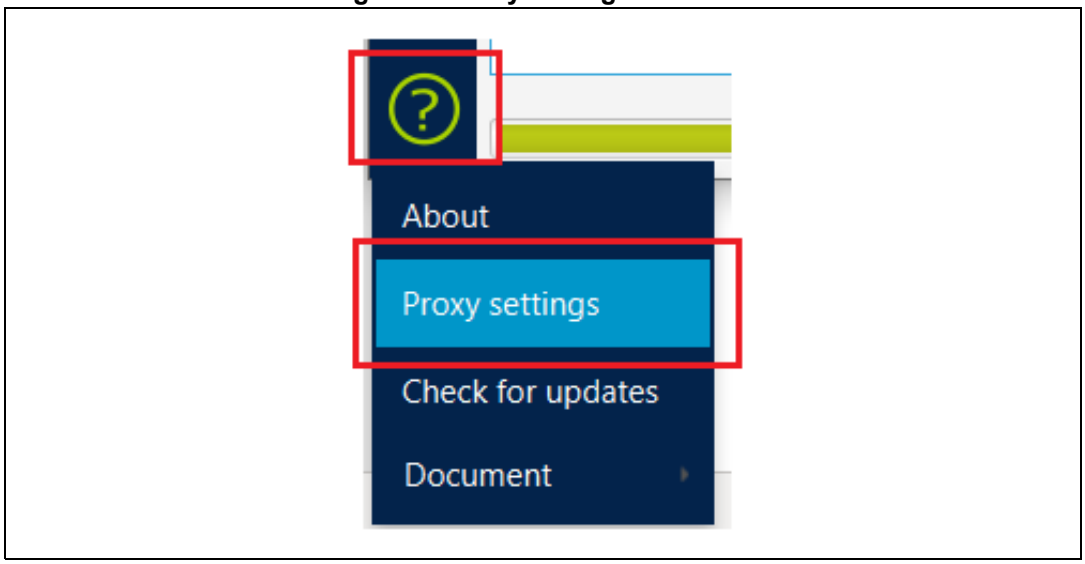
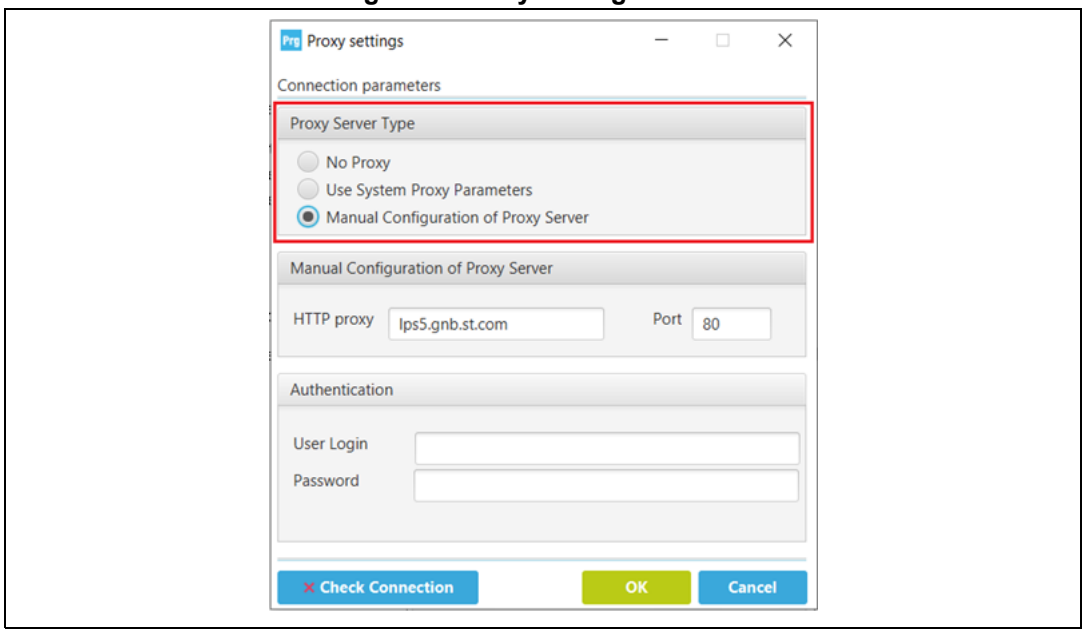


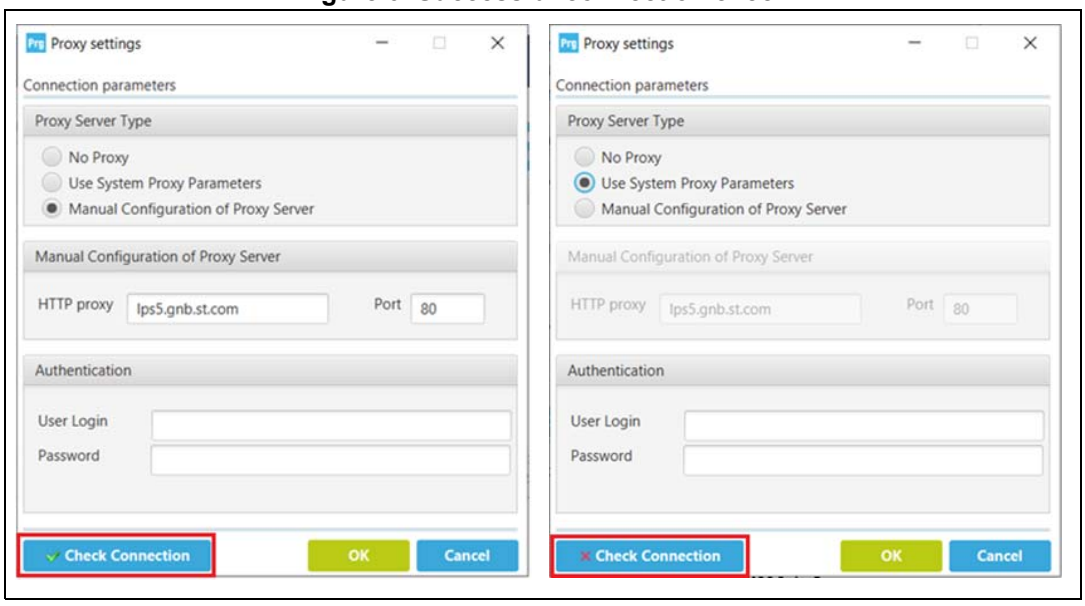
Figure 5. Proxy settings window



The status of connection check is displayed in the “check Connection” button:

- A green icon indicates success (left side of *Figure 6*).
- A red icon indicates that the connection is down (right side of *Figure 6*).

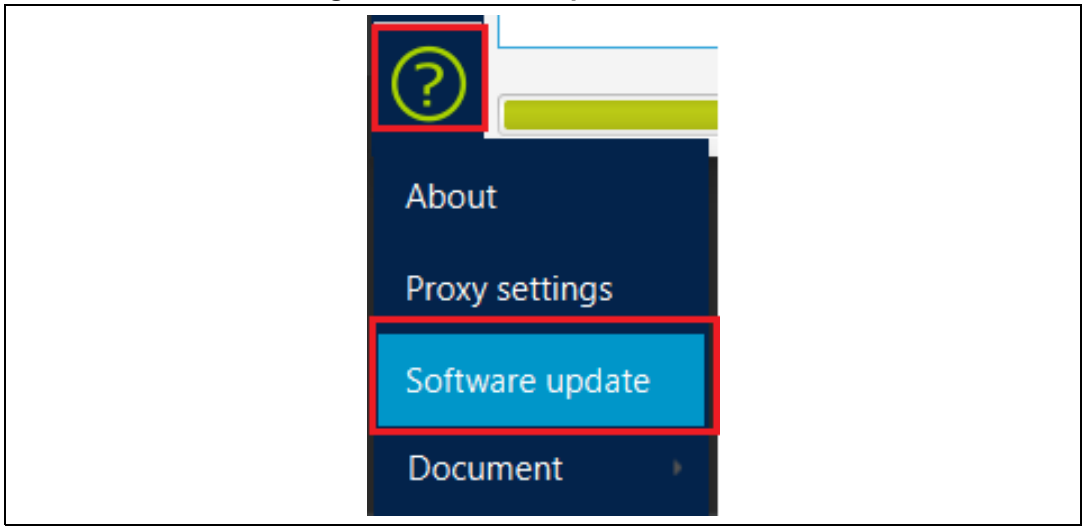
Figure 6. Successful connection check



1.3.3 Check for updates

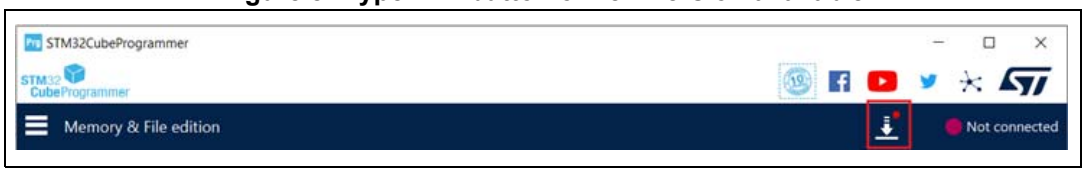
User can launch the process of update using the Updater window opened with the submenu “Software update” added in the help button.

Figure 7. Check for updates submenu



If there is a new version available, an update button appears in the main menu (*Figure 8*).

Figure 8. Hyperlink button of new version available



Note: If the user has already updated the STM32CubeProgrammer, the hyperlink button is no longer displayed at startup.

If a new version is available, the user can make updates through the updater window.

This window displays:

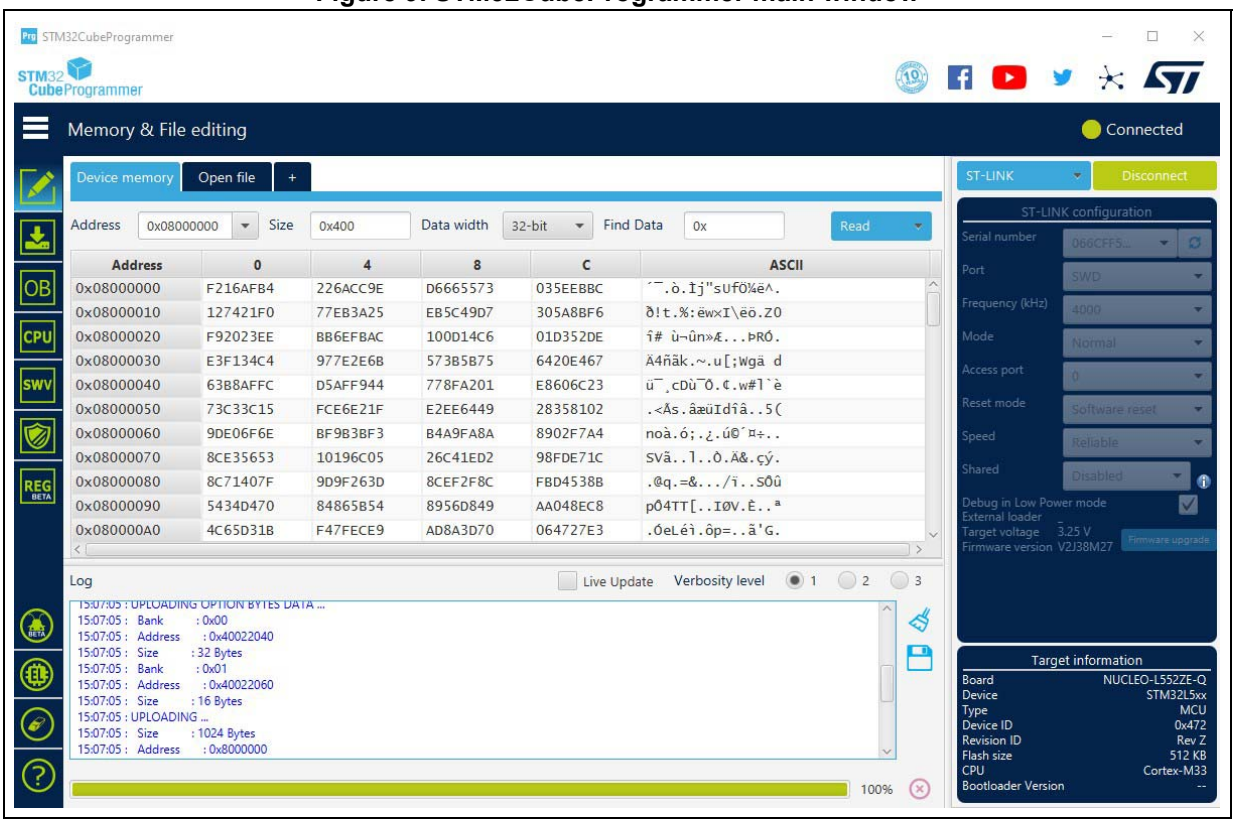
- The current version of the STM32CubeProgrammer
- The available version in server of STM32CubeProgrammer
- Change log (contains the main changes delivered in the new package)
- License
- Last update (contains the date of the last update, or the message “No previous updates are done”)
- The current version of the updater tool
- Refresh button (used to check if there is a new version)
- Close button (used to stop the installation of the new version)

Note: Administrator rights are required to download the new package. Once the update is done, the updater window displays only the new version.

2 STM32CubeProgrammer user interface for MCUs

2.1 Main window

Figure 9. STM32CubeProgrammer main window



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

2.1.1 Main menu

This menu allows the user to switch between the three main panels of the Memory and file editing, Erasing & programming, and Option bytes tools. The other panels are displayed according to 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 10*.

Figure 10. 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 ratio buttons above the log text zone. The minimum verbosity level is 1, and the maximum is 3 (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, 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 (for example, Read, Write, Erase). You can abort ongoing operations by pressing the ”Stop” button in front of the progress bar.

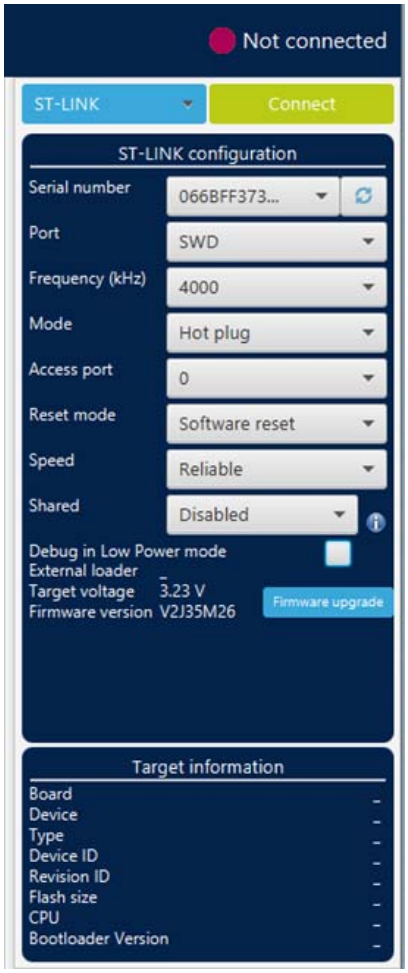
2.1.4 Target configuration panel

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. If 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 communication 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 in the Port combo box. Each interface has its own settings, to be set before connection.

ST-LINK settings

Figure 11. 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.

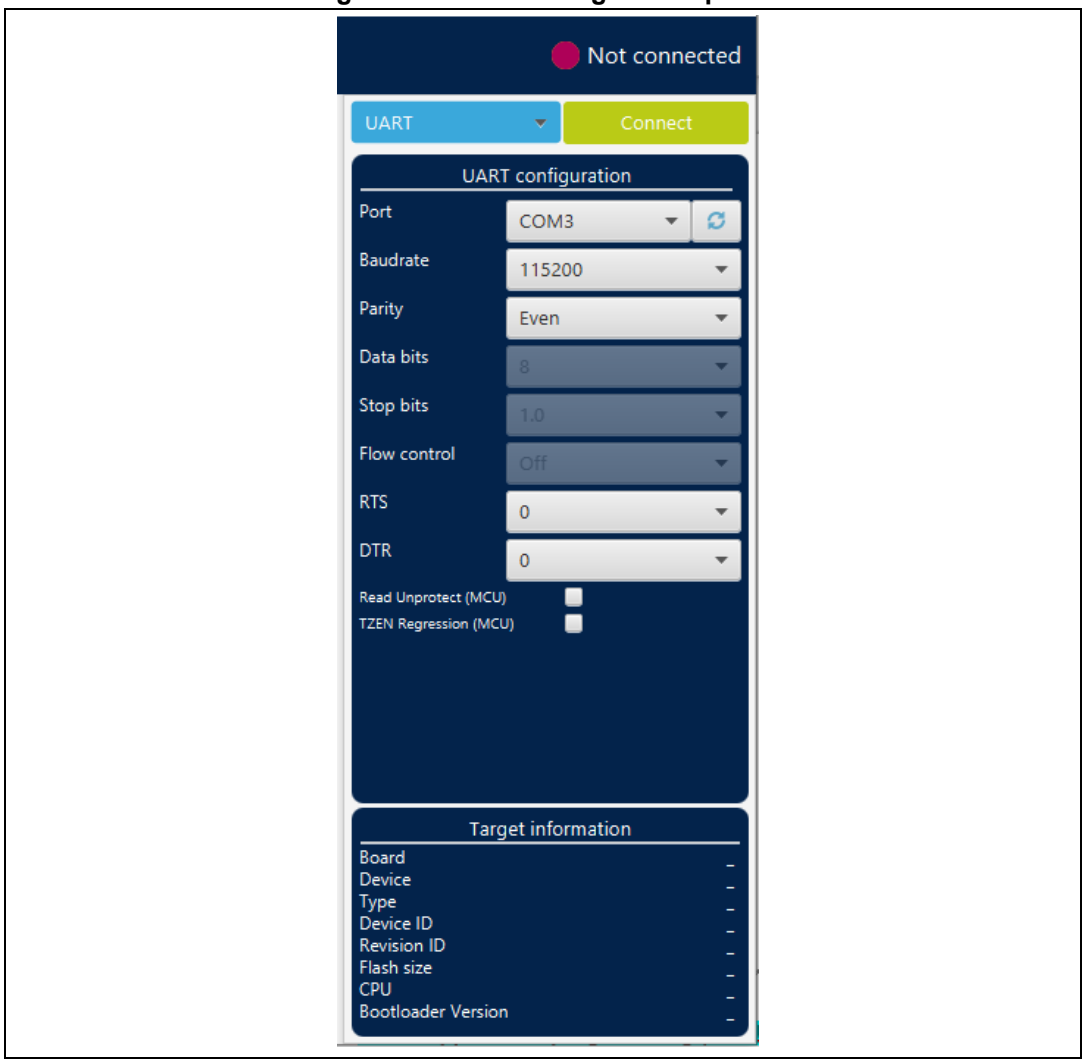
- **Frequency**: the JTAG or SWD clock frequency
- **Access port**: selects the access port to connect to. Most of the STM32 devices have only one access port, which is Access port 0.
- **Mode**:
  - **Normal**: with “Normal” connection mode, the target is reset then halted. The type of reset is selected 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.
  - **Power down**: used to put the target in debug mode, even if the application has not

started since the target power-up. The hardware reset signal must be connected between ST-LINK and the target. This feature can be not fully effective on some boards (MB1360, MB1319, MB1361, MB1355) with STMPS2141 power switch.

- **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 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): sets 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**: target voltage is measured and displayed.
- **Firmware version**: displays the ST-LINK firmware version. The firmware upgrade button allows you to upgrade the ST-LINK firmware.

UART settings

Figure 12. UART configuration panel



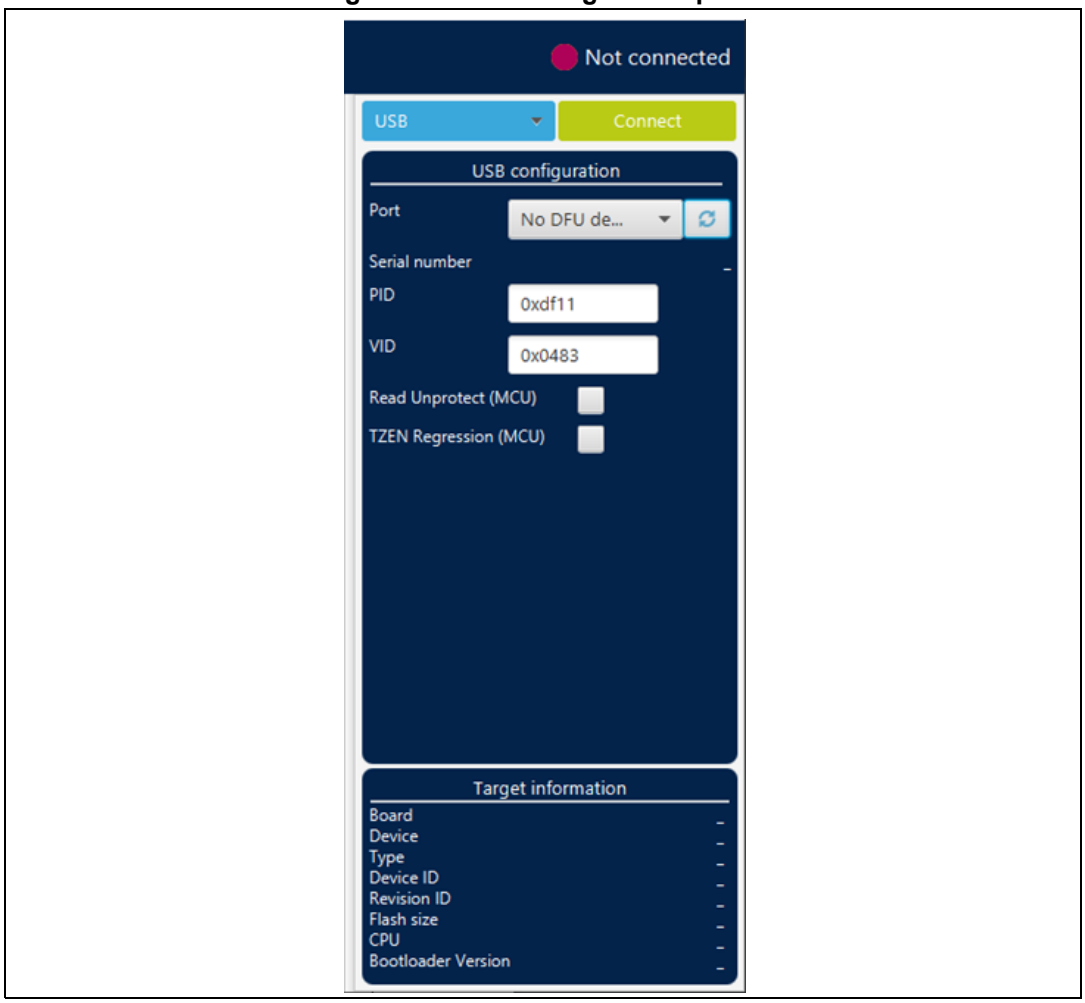
- **Port**: selects the com port to which the target STM32 is connected. Use the refresh button to recheck the available com port on the PC.

Note: The STM32 must boot in bootloader mode using boot pins and/or the option bits. Check AN2606 "STM32 microcontroller system memory boot mode", 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 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 13. 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.

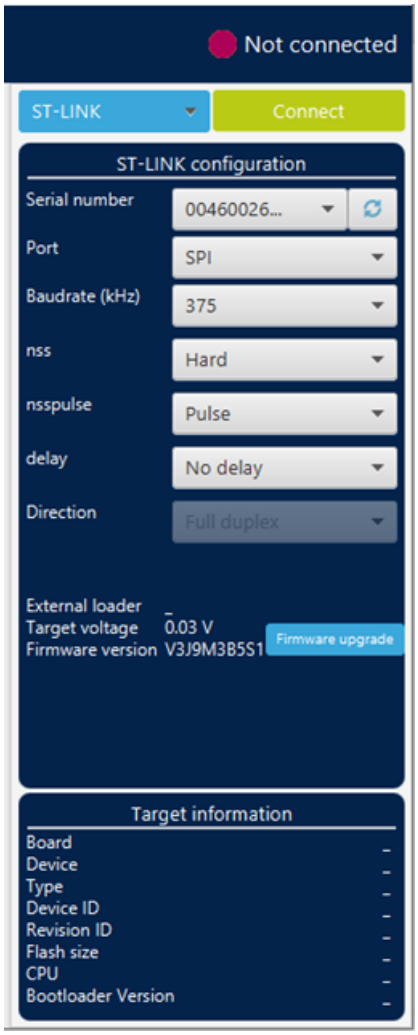
Once the correct interface settings are set, click on the “Connect” button to connect to the target interface. If the connection succeeds, it is shown in the indicator above the button, which turns to green.

Once connected, the target information is displayed in the device information section below the settings section, which is then disabled as in *Figure 14*.

Figure 14. Target information panel

SPI settings

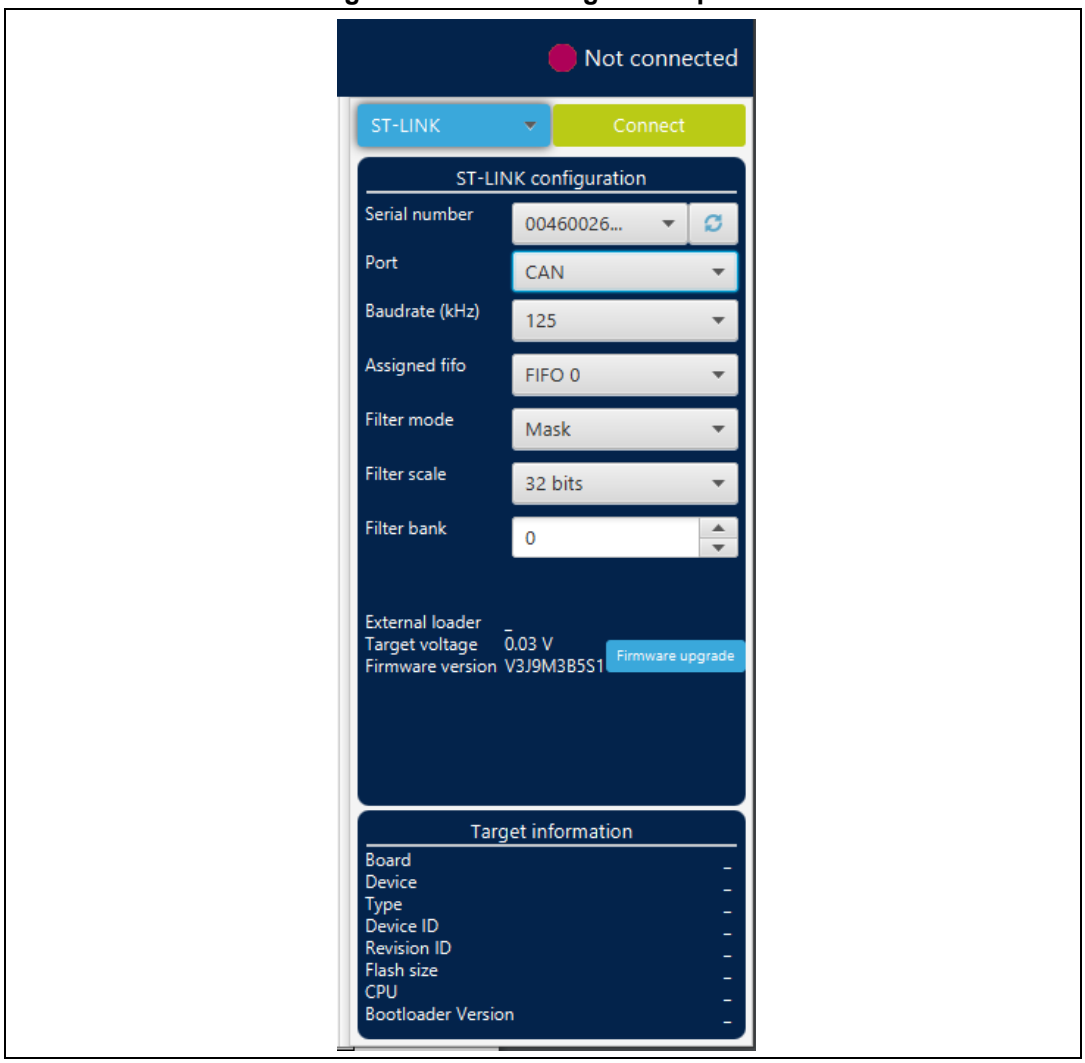
Figure 15. SPI configuration panel



- **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. 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

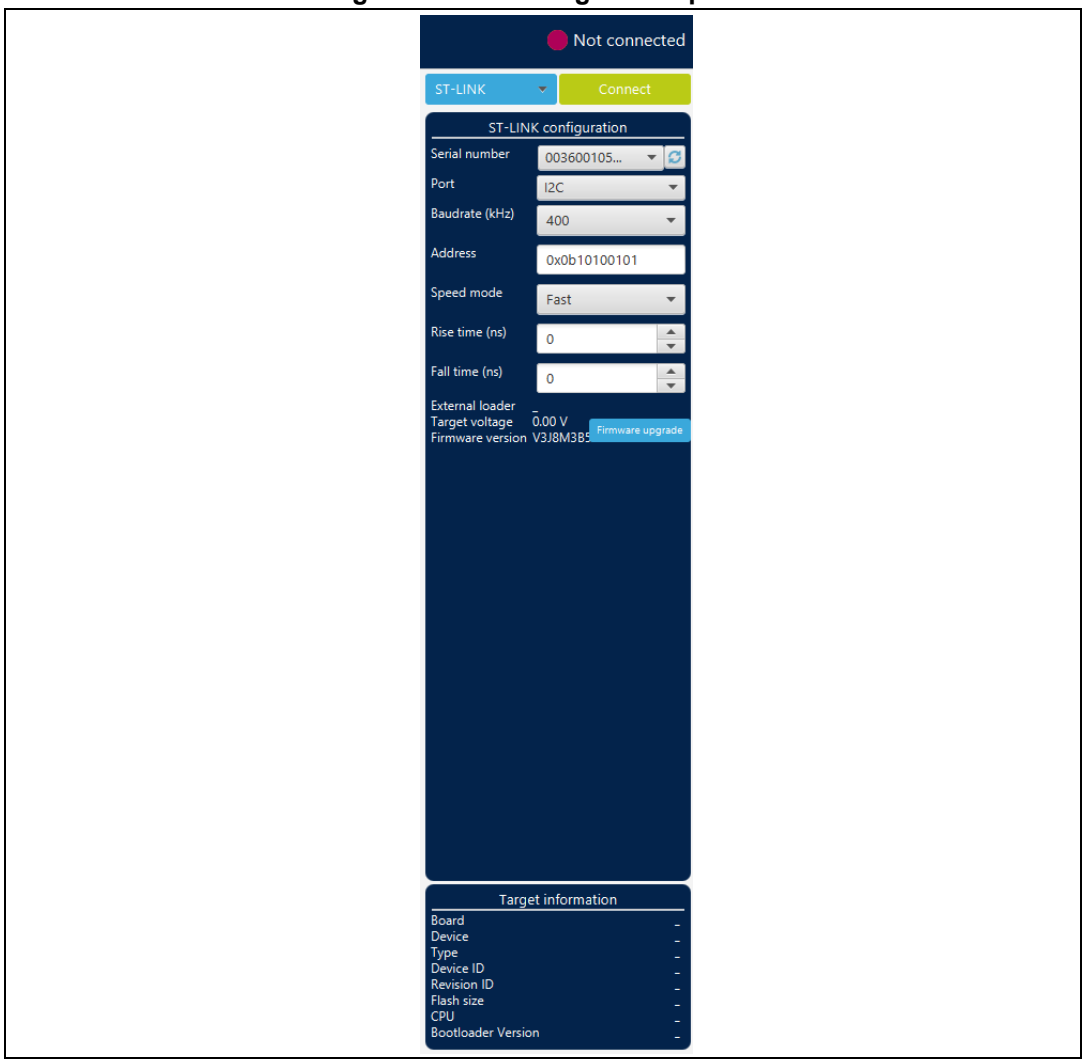
Figure 16. CAN configuration panel



- **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 17. I2C configuration panel



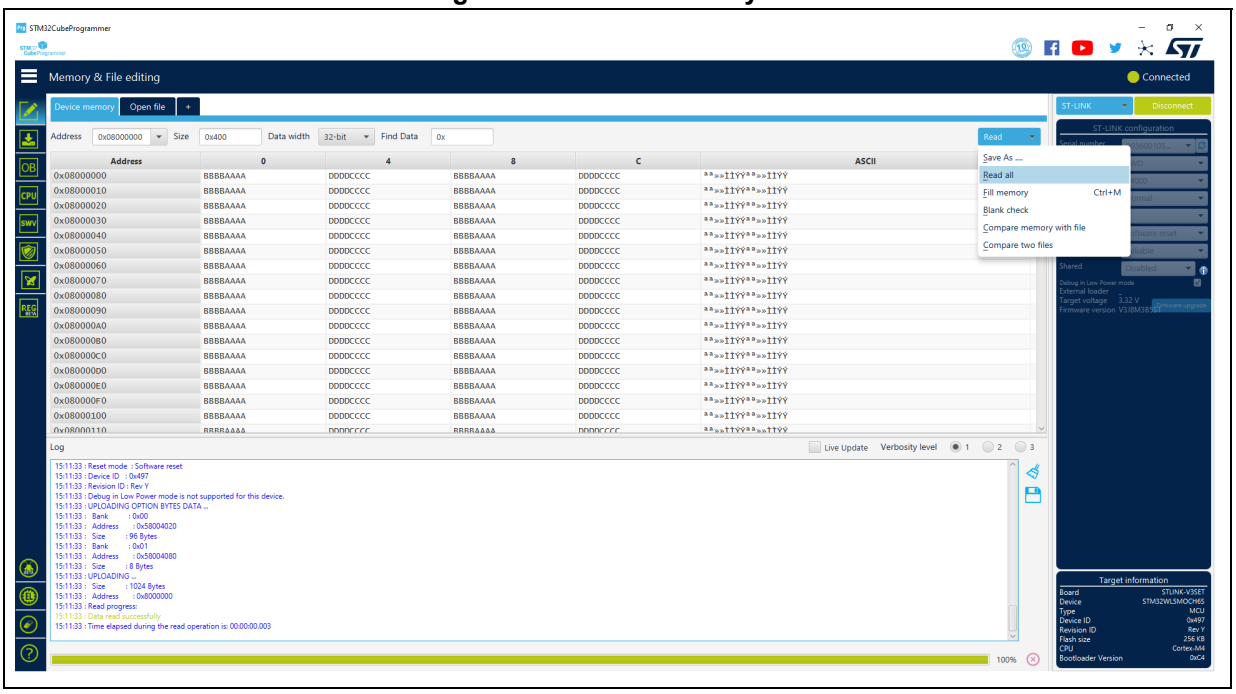
- **Serial number**: this field contains the serial numbers of all connected ST-LINK-V3 probes when using the I2C bootloader.
- **Port**: selects the I2C devices connected to the PC. You can use the refresh button to recheck the available devices.
- **Baudrate**: selects the I2C baud rate.
- **Address**: 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), 0-400 (FAST).

2.2 Memory & file edition

This panel allows the user to read and display target memory and file contents.

2.2.1 Reading and displaying target memory

Figure 18. Device memory tab

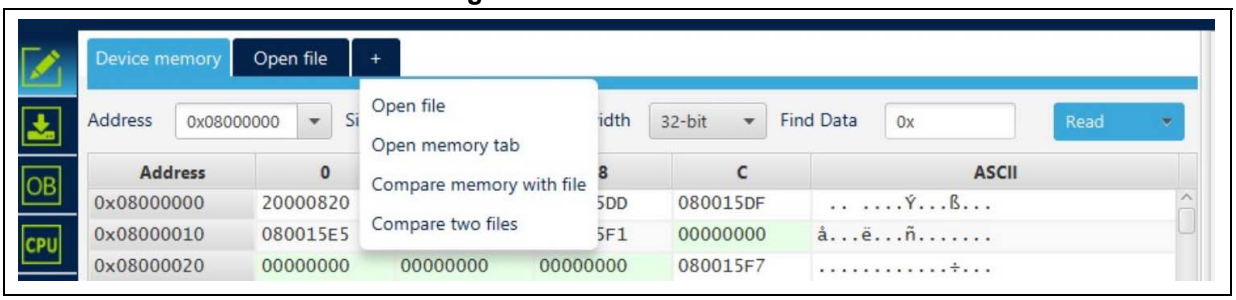


After target connection, the STM32 target memory can be read using this panel. To do this, specify the address and the size of the data to be read, then press the Read button in the top-left corner. Data can be displayed in different formats (8-, 16-, and 32-bit) using the “Data width” combo box.

The user can read all the flash memory using the “Read All” button, save the device memory content in a .bin, .hex, or .srec file using the “Save As...” menu from the tab contextual menu or the action button.

Multiple device memory tabs can be opened to display different locations of the target memory. To do this, 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 19. Contextual menu



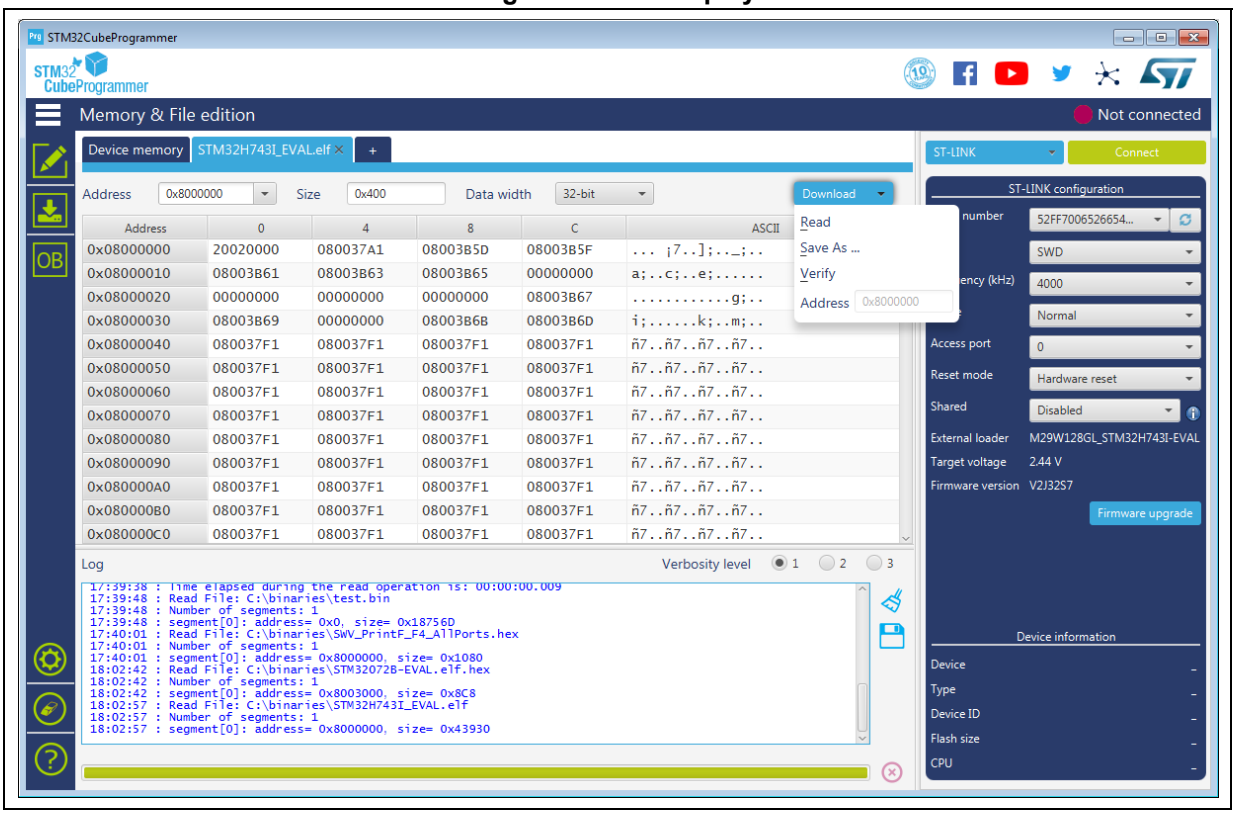
2.2.2 Reading and displaying a file

To open and display a file, just click on the “+” and select “Open File” menu, as illustrated in *Figure 19*.

The supported formats 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 20*. 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 20. 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, the file can be downloaded using the “Download” button/menu. For a binary file, specify the download address in the “Address” menu. The user can verify if the file is downloaded using the “Verify” menu, and save it in another format (.bin, .hex or .srec).

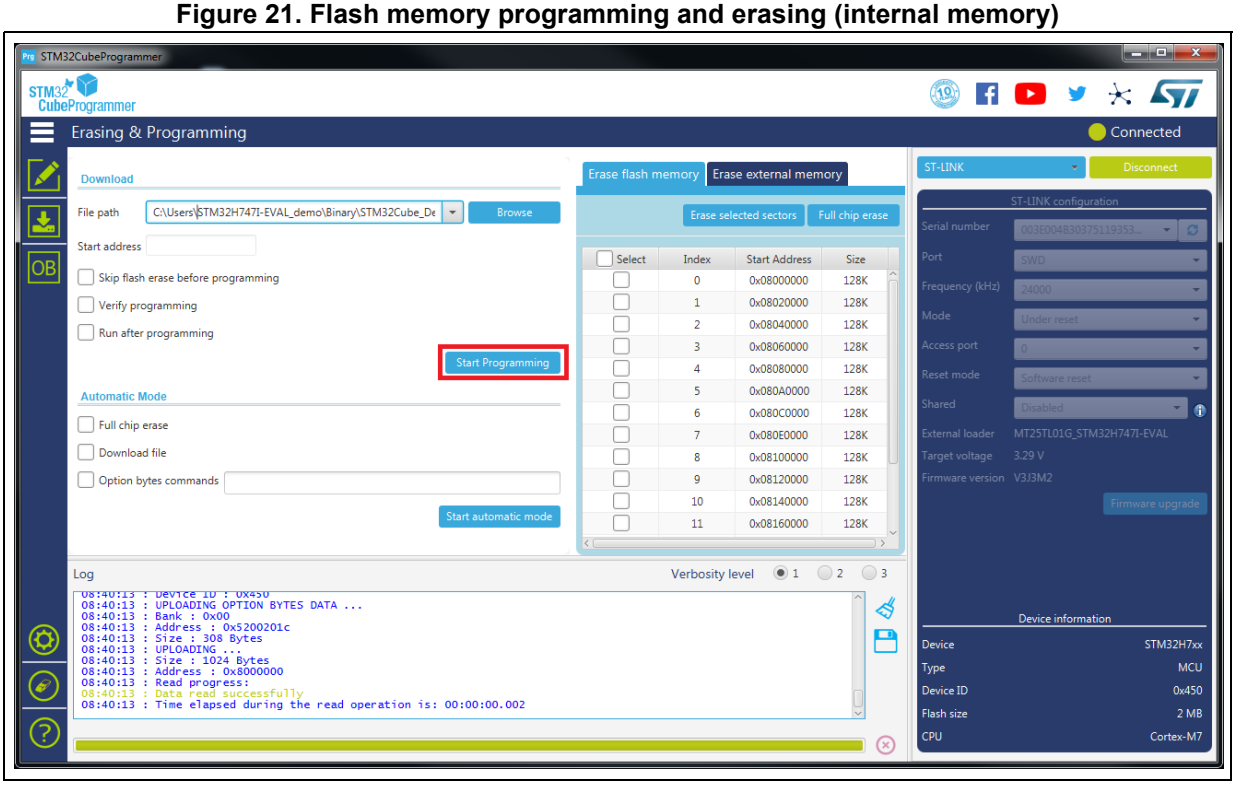
As for the “Device memory” tab, the user can display the file memory content in different formats (8-, 16-, and 32-bit), using the “Data width” combo box.

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 21. 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 memory.

Memory programming

To program a memory, go through the following steps:

1. Click on the browse button and select the file to be programmed. The supported formats 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 memory 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 (for example SPI, FMC, FSMC, QSPI, OCTOSPI) you need an external loader.

STM32CubeProgrammer is delivered with external loaders for several STM32 evaluation and discovery boards (refer to 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 loader(s) from the “ExternalLoader” panel, which is (are) used by the tool to read, program, or erase external memories, as shown in *Figure 22*. 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 23*.

Figure 22. Flash memory programming (external memory)

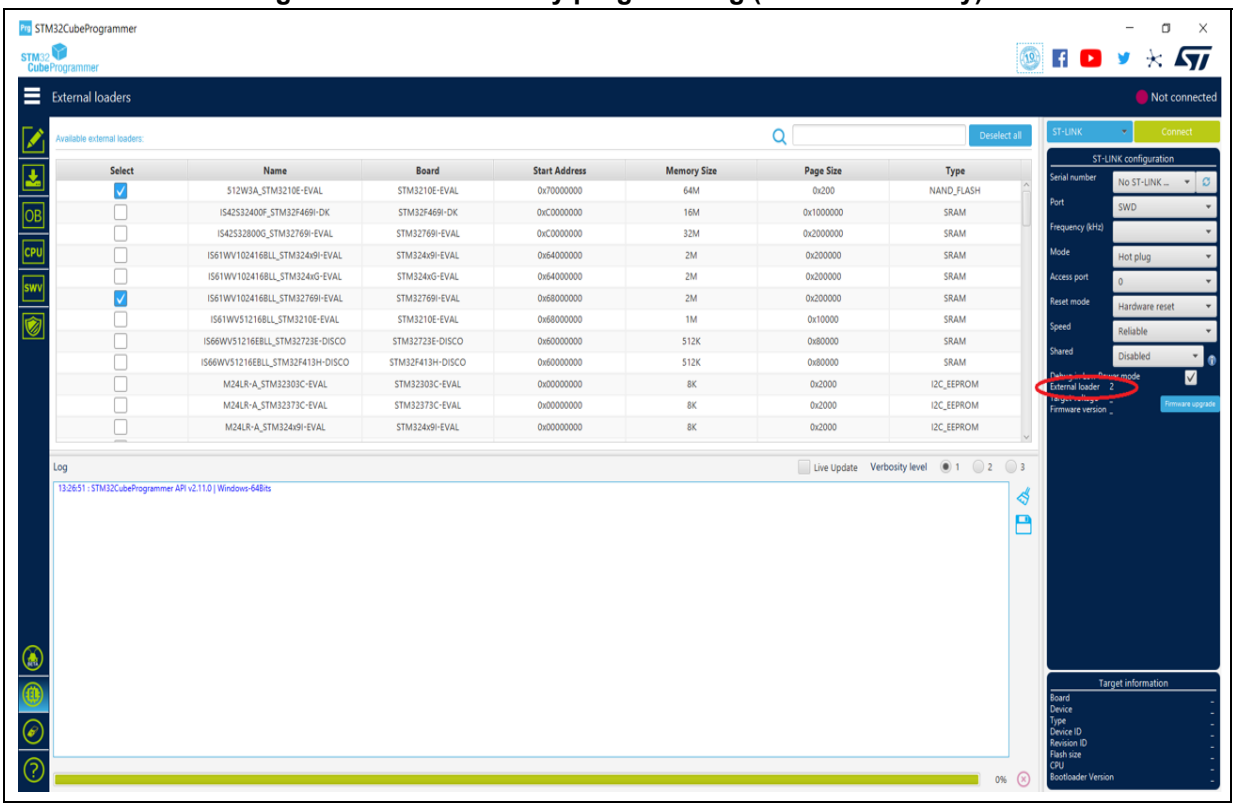
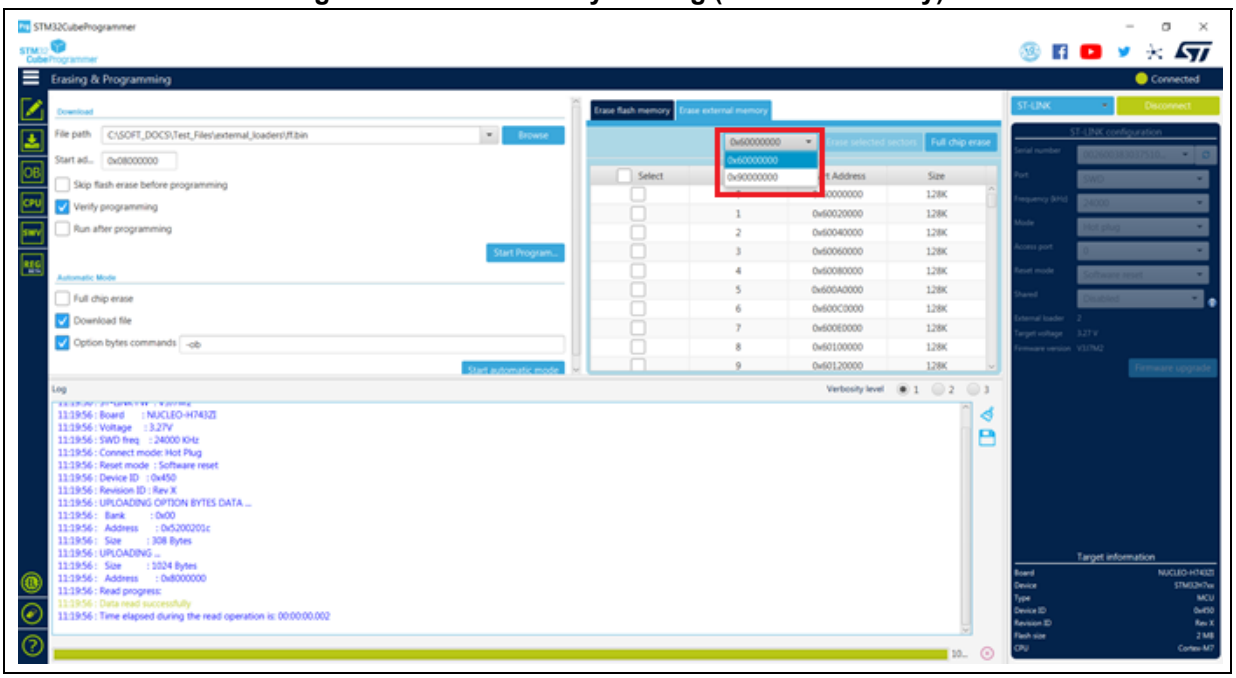


Figure 23. Flash memory erasing (external memory)



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 programming mechanism is the same used by the STM32 ST-LINK utility tool. Any flash loader developed for use 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 this 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 ".stldr" and the file must be copied under the "*bin/ExternalLoader*" directory.

Loader_Src.c file

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

- **`Init`**

  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`**

  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`**

  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 (such as 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 Quad-/Octo-SPI memories, the memory mapped mode can be defined in the Init function; in that case, the Read function is useless, as 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, 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 defined by this structure 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
```

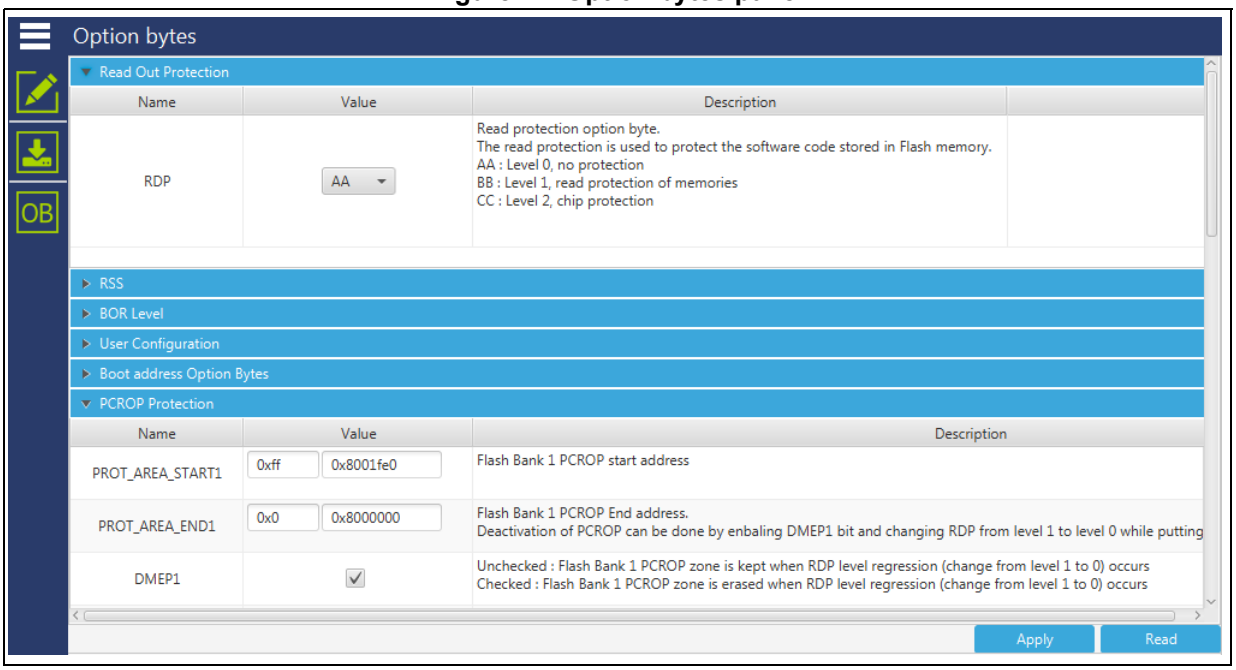
```
      0x00100000, // Device Size in Bytes (1MBytes/8Mbits)
      0x00004000, // Programming Page Size 16KBytes
      0xFF, // Initial Content of Erased Memory
// Specify Size and Address of Sectors (view example below)
      0x00000004, 0x00004000, // Sector Num : 4, Sector Size: 16KBytes
      0x00000001, 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 name, the bit 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 24. Option bytes panel

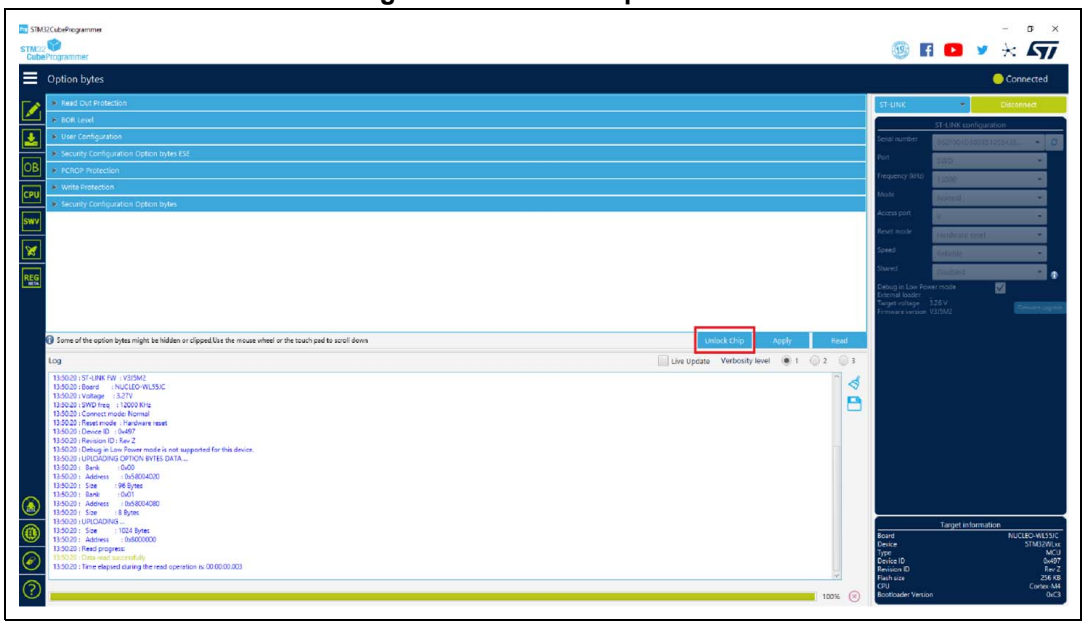


For more details refer to the option bytes section in the programming and reference manuals, available from *www.st.com*.

2.4.1 MCU unlock (specific for the STM32WL series)

The user can unlock the device if bad option bytes are already programmed by clicking on the “Unlock chip” button (available only for STLink connection). After the unlock execution a power cycle is needed.

Figure 25. Unlock chip button



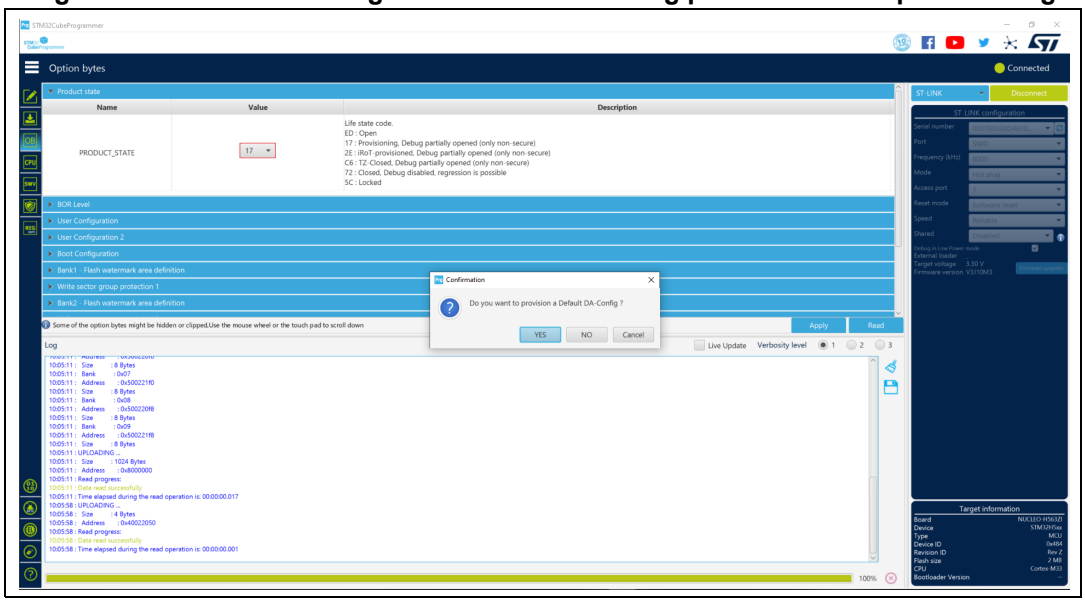
2.4.2 Debug authentication default configuration (STM32H573/STM32H563/STM32H562 only)

The default configuration is used when programming the product state. It allows the user to provision the configuration after programming the product state to any value. Afterwards, the user can provision its own OBK file.

If the user does not configure the debug authentication (DA) and switches PRODUCT_STATE to provisioned/TZ-closed or closed, it is no longer possible to perform regressions, nor to go back to product state open. All debug features are disabled.

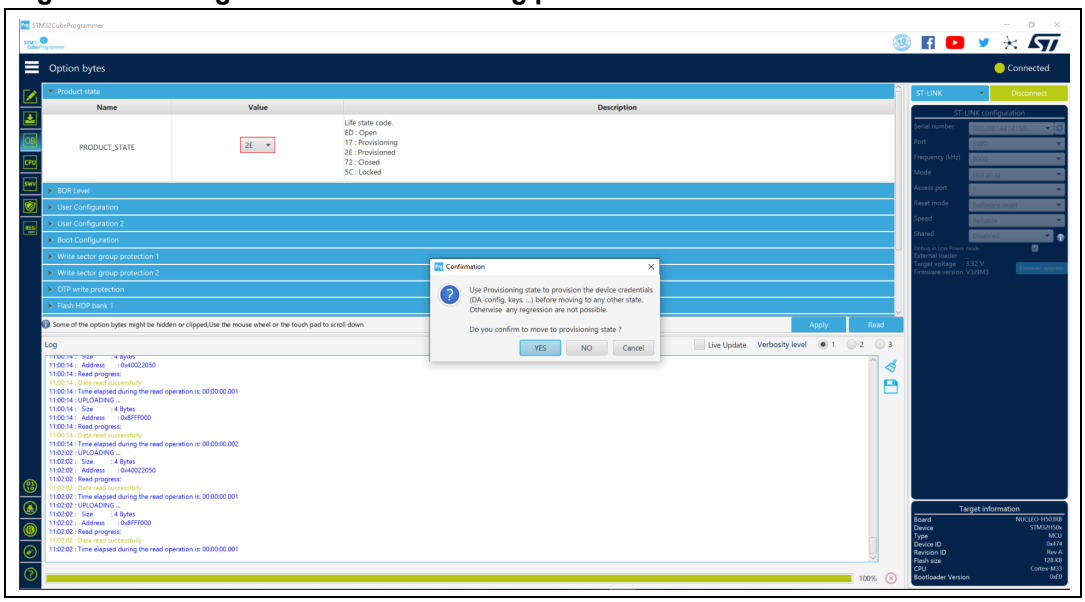
When setting PRODUCT_STATE to 0x17 (provisioning), the user is asked to provision the DA default configuration, or to use its own (see *Figure 26*).

Figure 26. DA default configuration when switching product state to provisioning



When switching PRODUCT_STATE from 0xED (open) to values different from 0x17 (provisioning), the user is asked to pass by the provisioning state first (*Figure 27*).

Figure 27. Configuration when switching product state to values different from 0x17



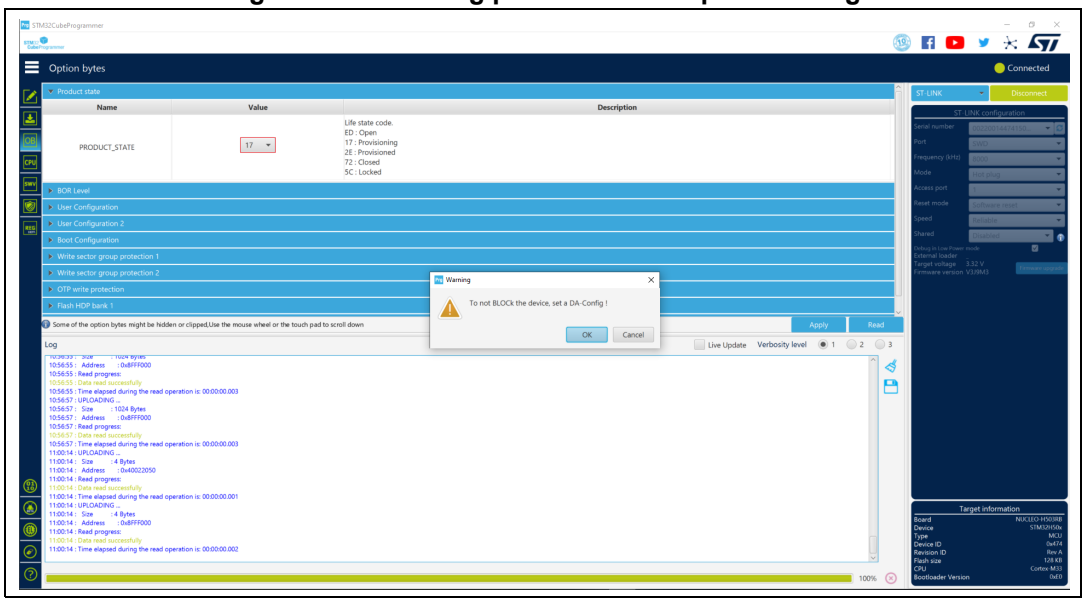
If the user chooses to provision a default DA configuration, the tool provisions the OBK file under the “bin/ DA_Default_Config” directory. To perform debug authentication, the files under “bin/DA_Default_Config” directory are required.

2.4.3 Debug authentication configuration (STM32H503 only)

If the user does not configure the DA and switches to PRODUCT_STATE provisioned or closed, it is no longer possible to perform regressions, nor to go back to product state open. All debug features are disabled.

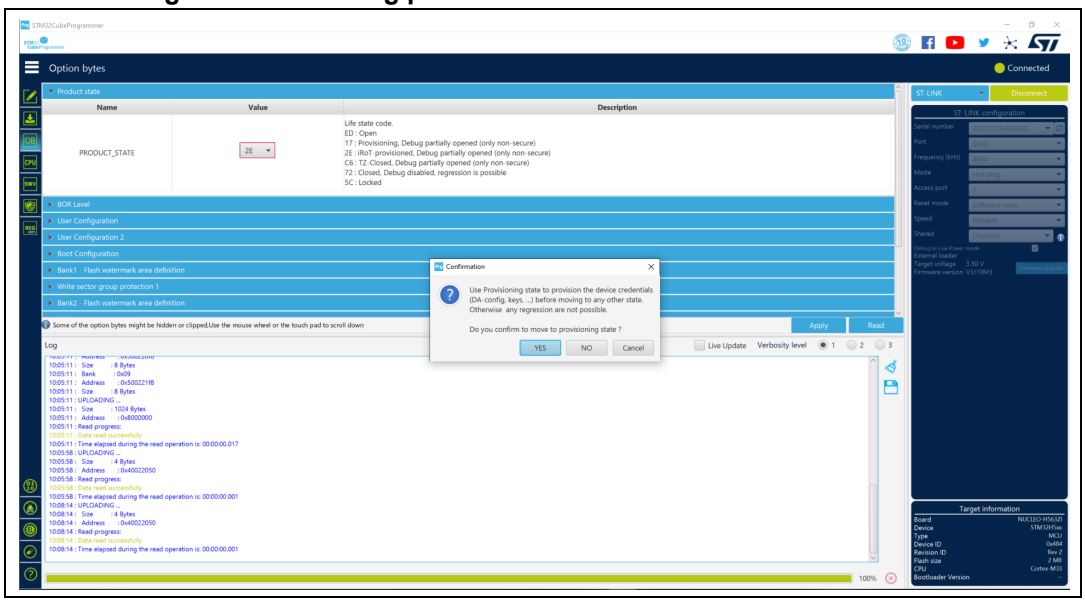
When setting PRODUCT_STATE to 0x17 (provisioning), the tool checks if there is a password provisioned in OTP. If not, a popup asks the user to set a configuration, to be able to perform a regression later.

Figure 28. Switching product state to provisioning



When switching PRODUCT_STATE from 0xED (open) to values different from 0x17 (provisioning), the tool checks if there is a password provisioned in OTP. If not, the user is asked to pass by the provisioning state first.

Figure 29. Switching product state to values different from 0x17

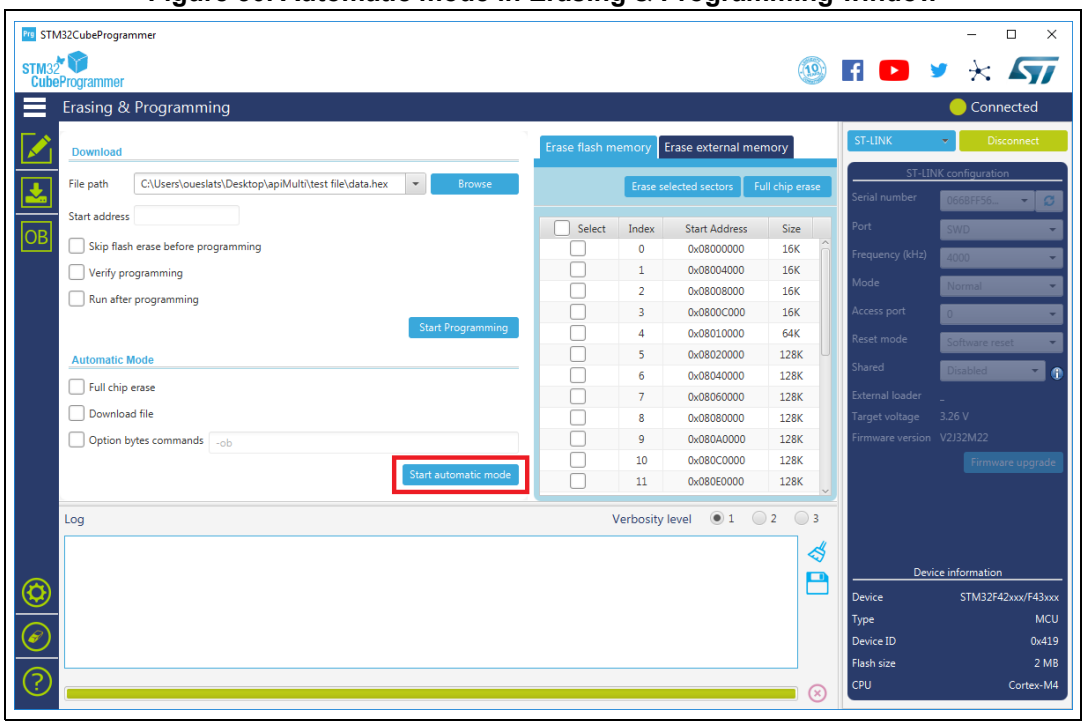


2.5 Automatic mode

This feature, shown in Erasing & Programming window (see *Figure 30*), allows the user to program and configure STM32 devices in loop. Allowed actions:

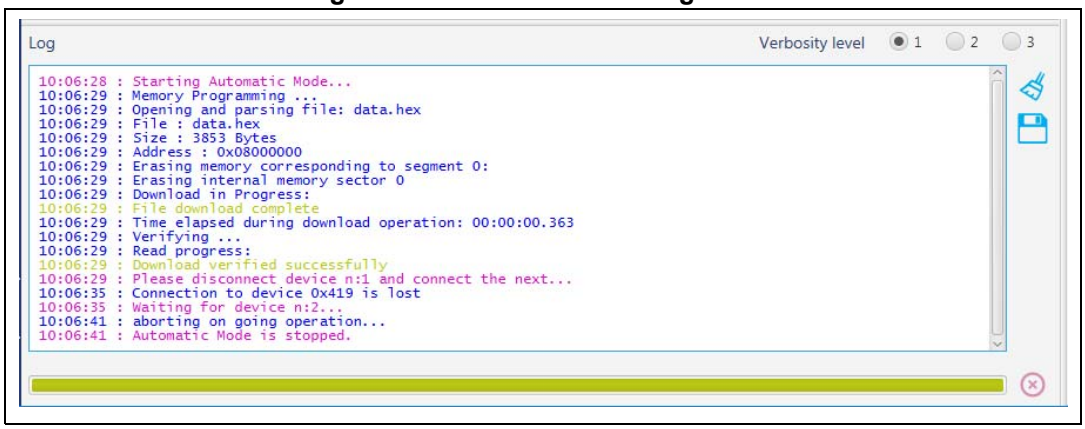
- Full chip erase: erases all the flash memory
- Download file: activates and sets programming options from Download section:
  - File path
  - Start address
  - Skip erase before programming
  - Verify programming
  - Run after programming
- Option bytes commands: configures the device by setting option bytes command line

Figure 30. Automatic mode in Erasing & Programming window



All automatic mode traces are indicated in the Log panel (see *Figure 31*), to show the process evolution and user intervention messages.

Figure 31. 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 time by pressing Cancel or Stop automatic mode buttons.

Log messages

- “Starting Automatic Mode...”

  Indicates that the system successfully entered the 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, asking the user 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/I^2C interfaces. A message is displayed, asking the user 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, asking the user to keep only one USB DFU connected to continue using this mode.
- “More UART ports detected than last connection!”

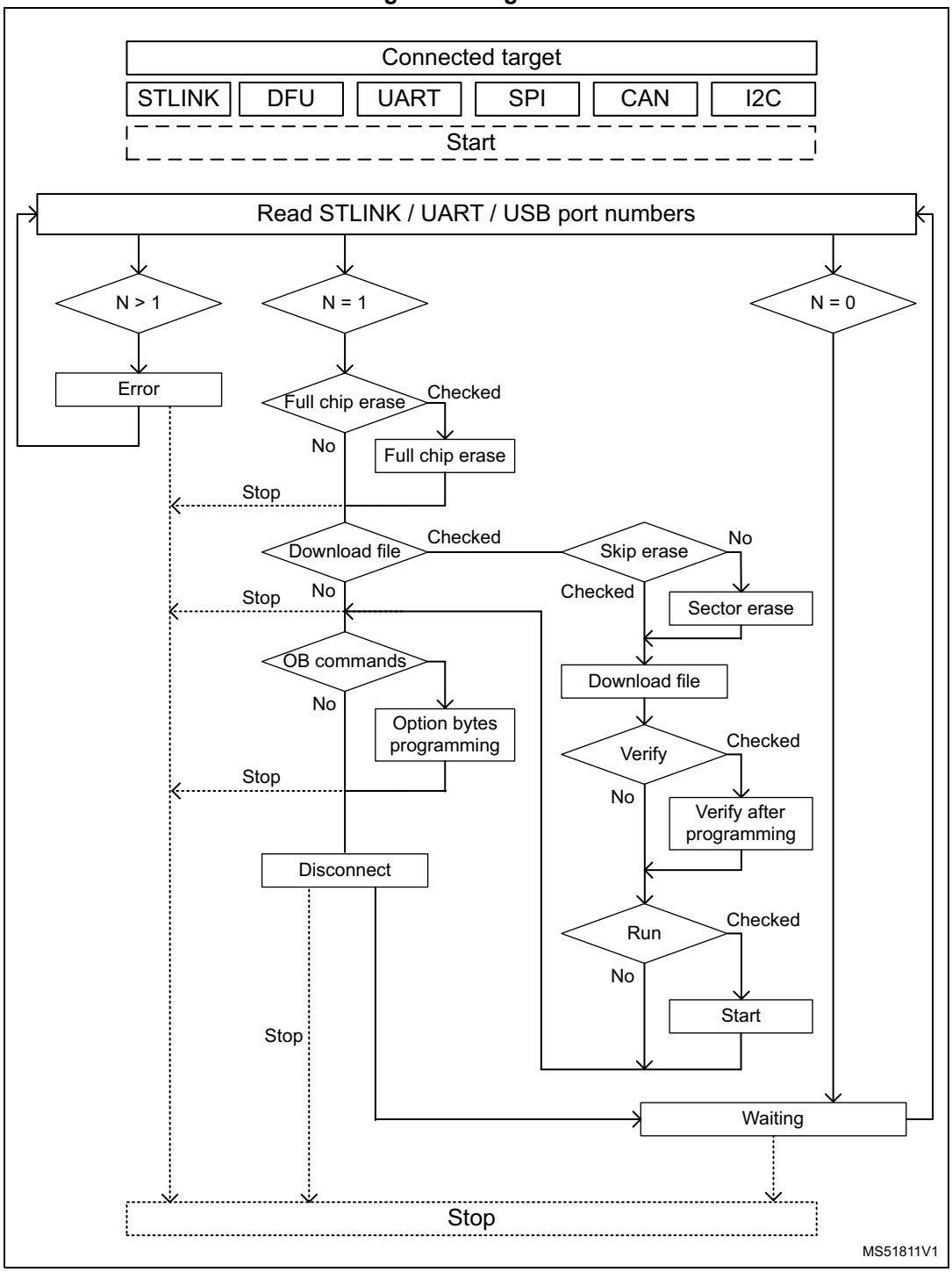
  During the first connection the automatic mode calculates the number of the available serial ports, and puts it as a reference, to detect correctly that only one port UART is used for each STM32 device.
- “Please disconnect device and connect the next...”

  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 cancel request, and the system stops the process.

Figure 32. Algorithm



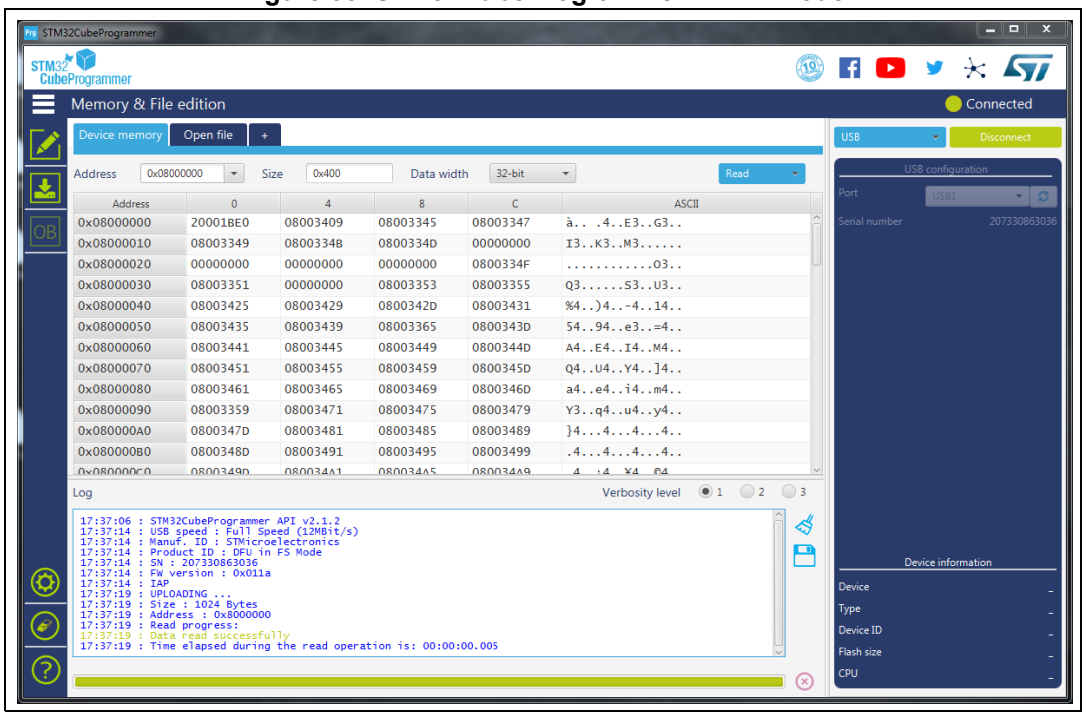
2.6 In application programming (IAP/USBx)

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 33. STM32CubeProgrammer 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 34*
2. Access the SWD/bootloader USB interface, see *Figure 35*
3. Delete the current wireless stack, see *Figure 36*
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 (a pop-up must appear to ensure successful upgrade), see *Figure 37*

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.

To download WB stacks and FUS from *www.st.com*, press on the logo, as shown in *Figure 38*.

Figure 34. STM32CubeProgrammer API SWD connection

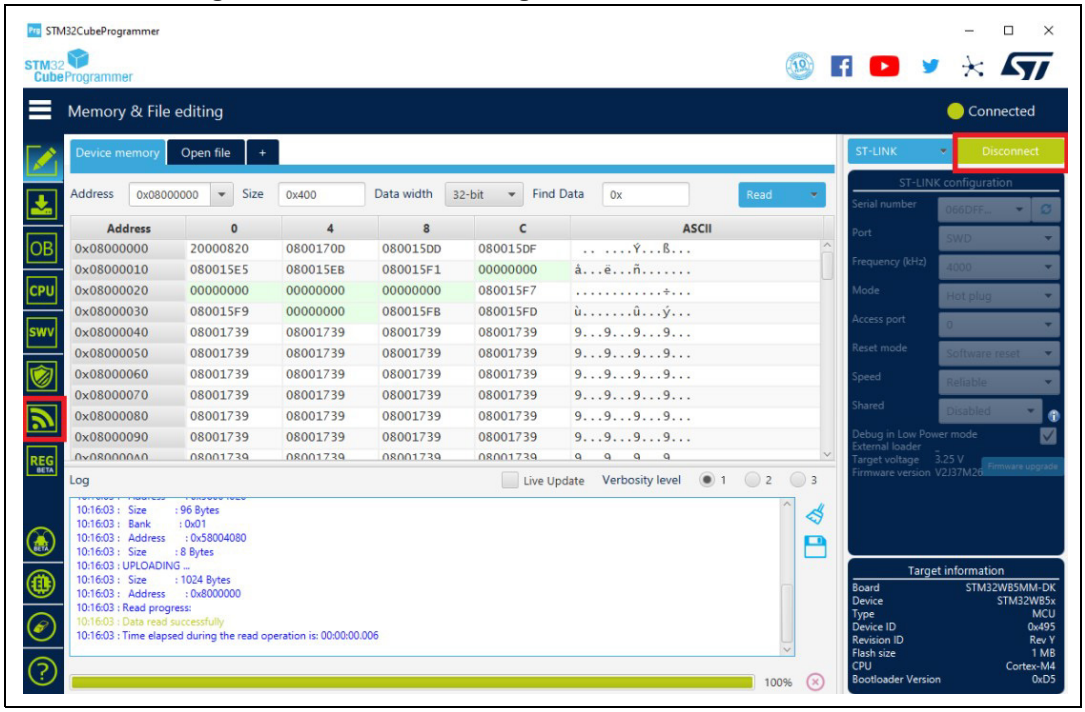


Figure 35. Steps for firmware upgrade

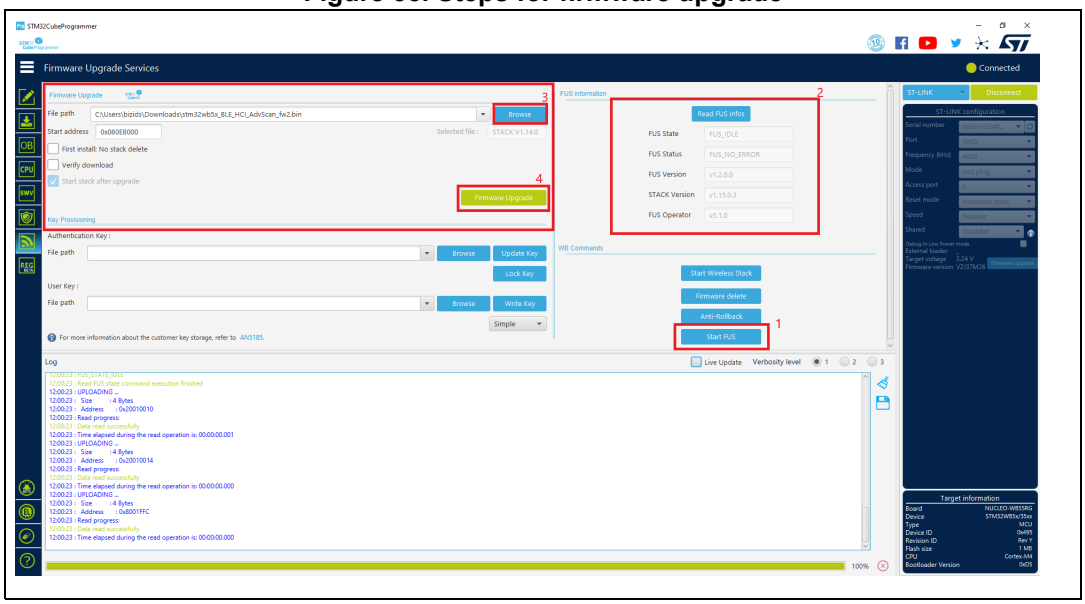


Figure 36. Pop-up confirming successful firmware delete

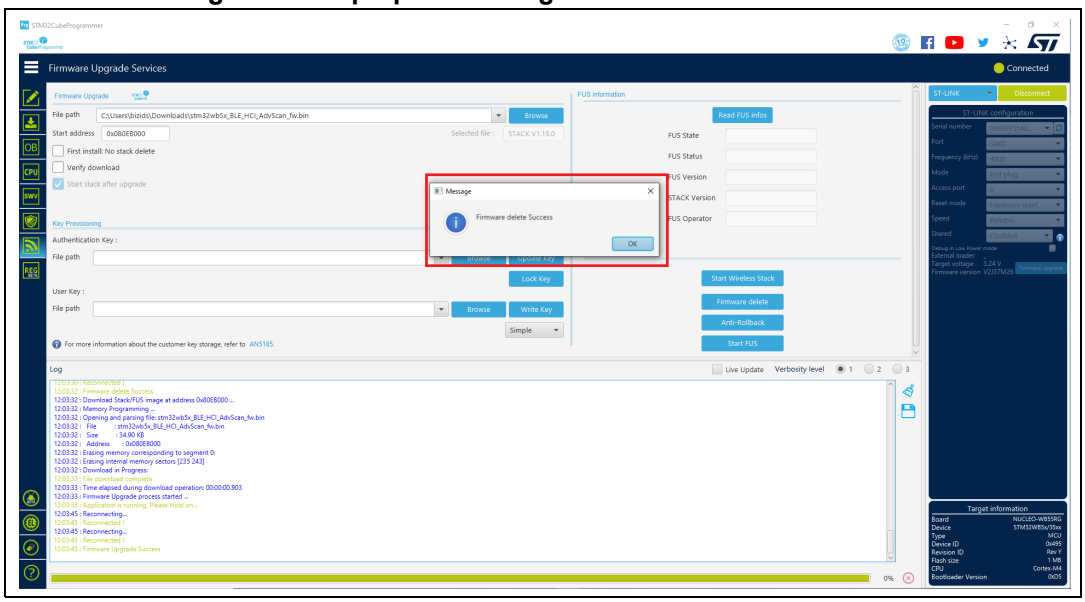


Figure 37. Pop-up confirming successful firmware upgrade

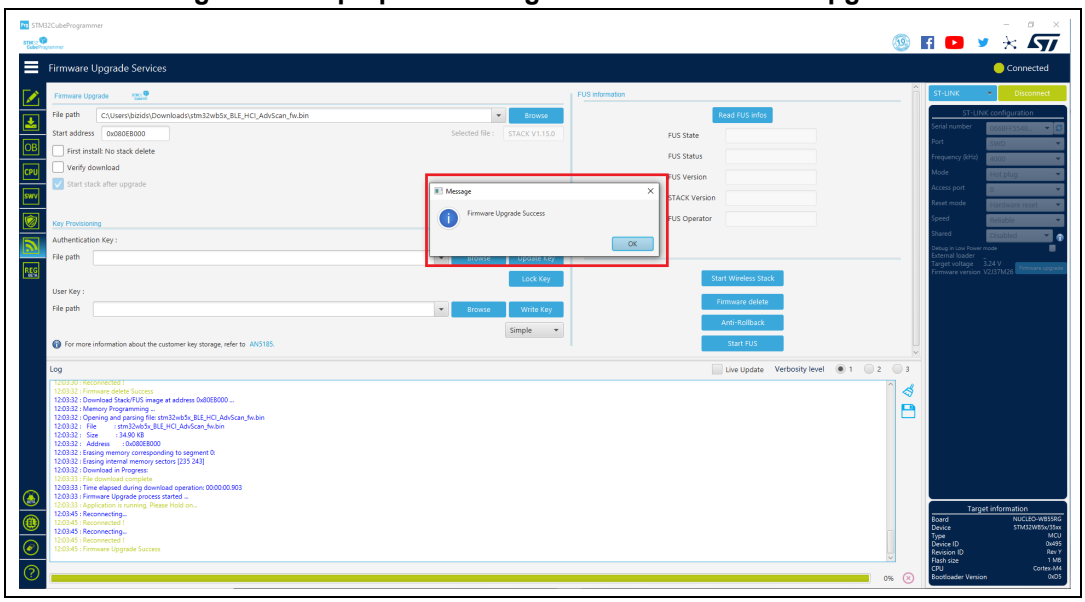
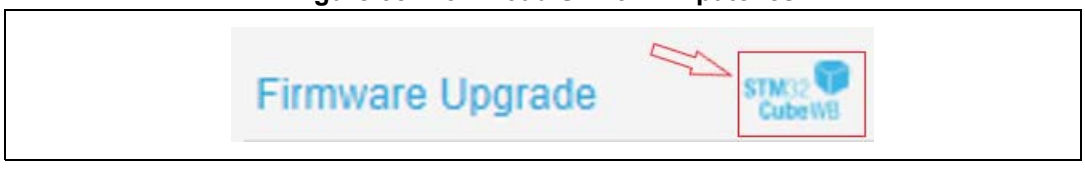


Figure 38. Download STM32WB patches



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 39*).

Once the user authentication key is installed, it can be changed, unless the lock user authentication key button is selected (see *Figure 40*). Once the authentication key is installed, the install or upgrade services must be done with the double signed FUS/Stack, or it is rejected.

Figure 39. Update authentication key

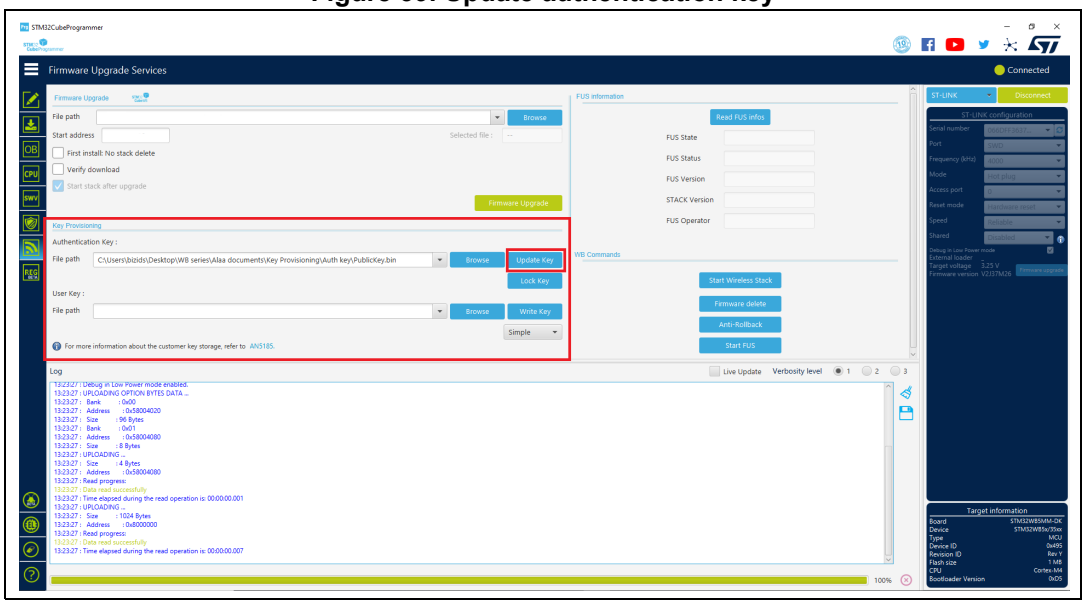
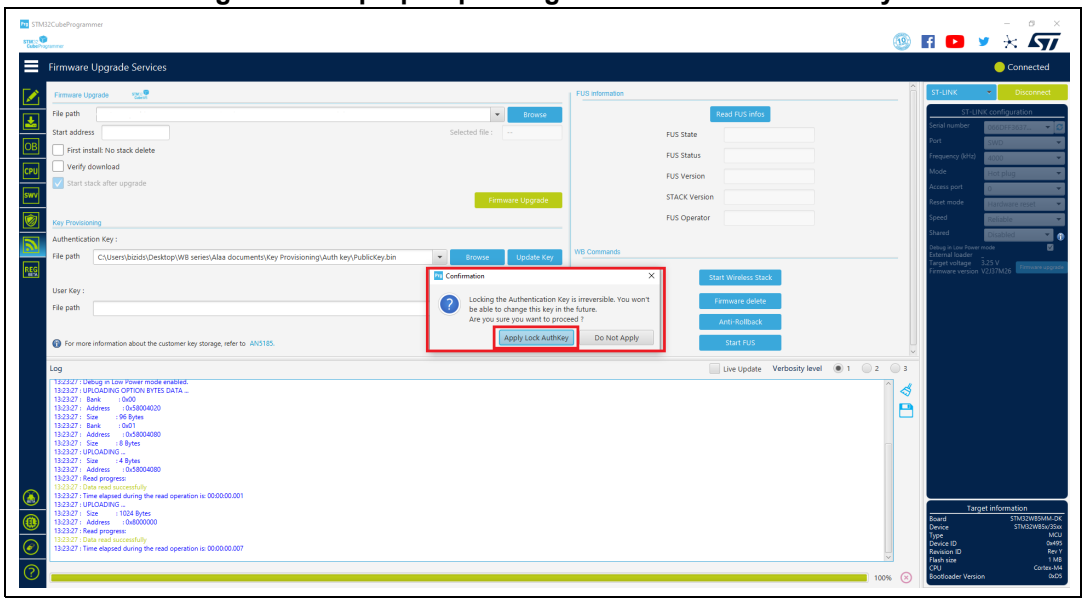


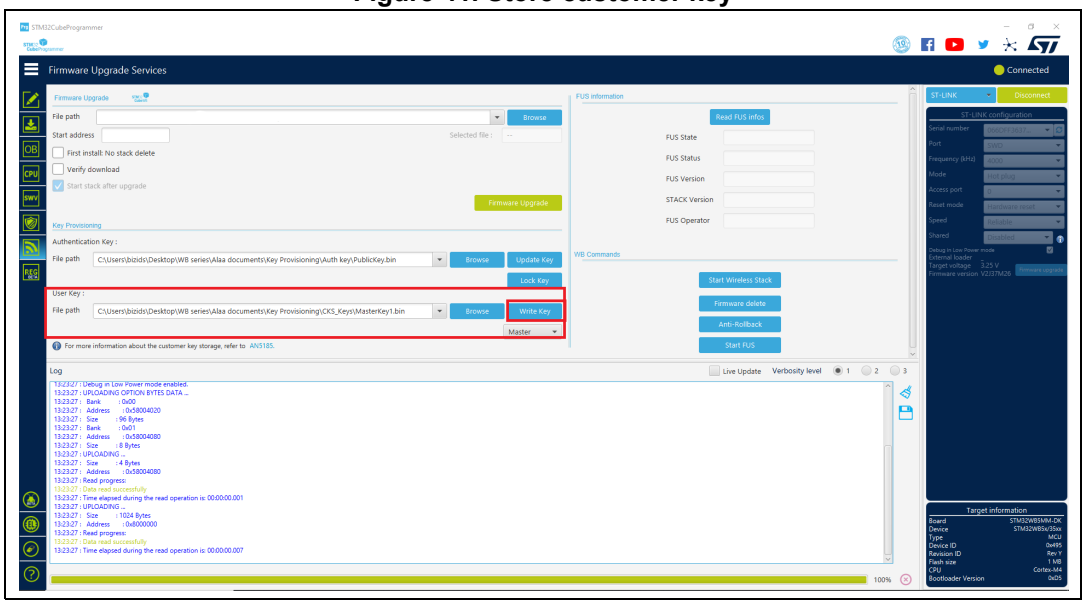
Figure 40. Pop-up requesting to lock authentication key



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 41*.

Figure 41. Store customer key



For more information about the customer key storage, refer to AN5185 “*ST firmware upgrade services for STM32WB Series*”. 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 42*) displays the printf data sent from the target through SWO, and useful information on the running firmware.

Note: The serial wire viewer is available only 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 of them.

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:

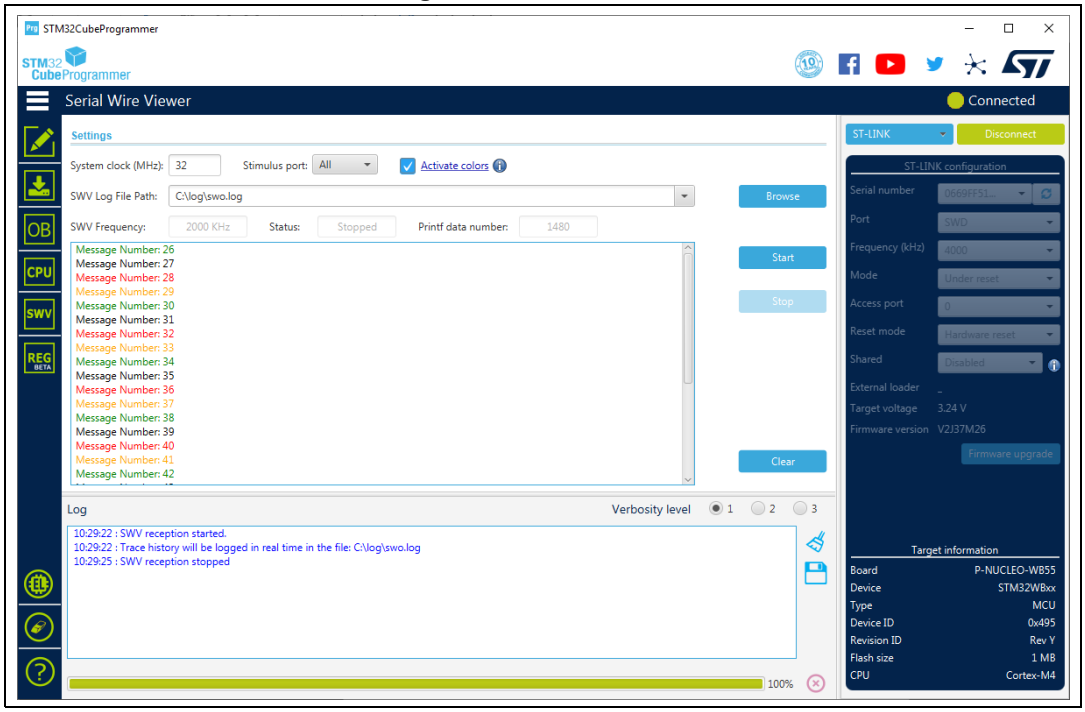
- #GRN# for green color
- #RED# for red color
- #ORG# for orange color

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 42. 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 Secure programming interface

2.9.1 Introduction

This window facilitates STM32CubeProgrammer CLI commands for secure programming:

- RDP regression with password feature: available for STM32U5 series
- SFI/SFIx feature: available for STM32H7, STM32U5 and STM32L5 series

2.9.2 RDP regression with password (STM32U5 series only)

Some STM32 products offer the possibility to use an optional password-based RDP level regression, including RDP level 2 (*Figure 43*).

- RDP level 1: the OEM1 RDP lock mechanism is active when the OEM1 key is set. It blocks the regression from the RDP level1 (*Figure 45*)
  - To unlock the RDP from level 1 regression, the user must write the OEM1 password, press on “RDP regression” button and then perform the RDP regression from “Option Bytes” interface (*Figure 46*).
  - To remove RDP regression with password from level 1, the user must press on “Disable password” button, as shown in *Figure 47*.
- RDP level 2: provision OEM2KEY to authorize RDP level 2 to level 1 regression: “Set password” button.
  - To unlock the RDP from level 2 regression, the user must write the OEM2 password, press on “RDP regression” button, and then try to connect with STM32CubeProgrammer. If this key matches the OEM2KEY value, the RDP regression to level 1 is launched by hardware.
  - To remove RDP regression with password from level 2, the user must press on “Disable password” button.
- Device authentication ID: Get device identification. Unless the JTAG port is deactivated (OEM2LOCK = 0 and RDP level = 2), a 32-bit device specific quantity can be always read through the JTAG port (see *Figure 48*). The OEM can use this 32-bit information to derive the expected OEM password keys to unlock the device.

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

Figure 43. RDP regression with password tab

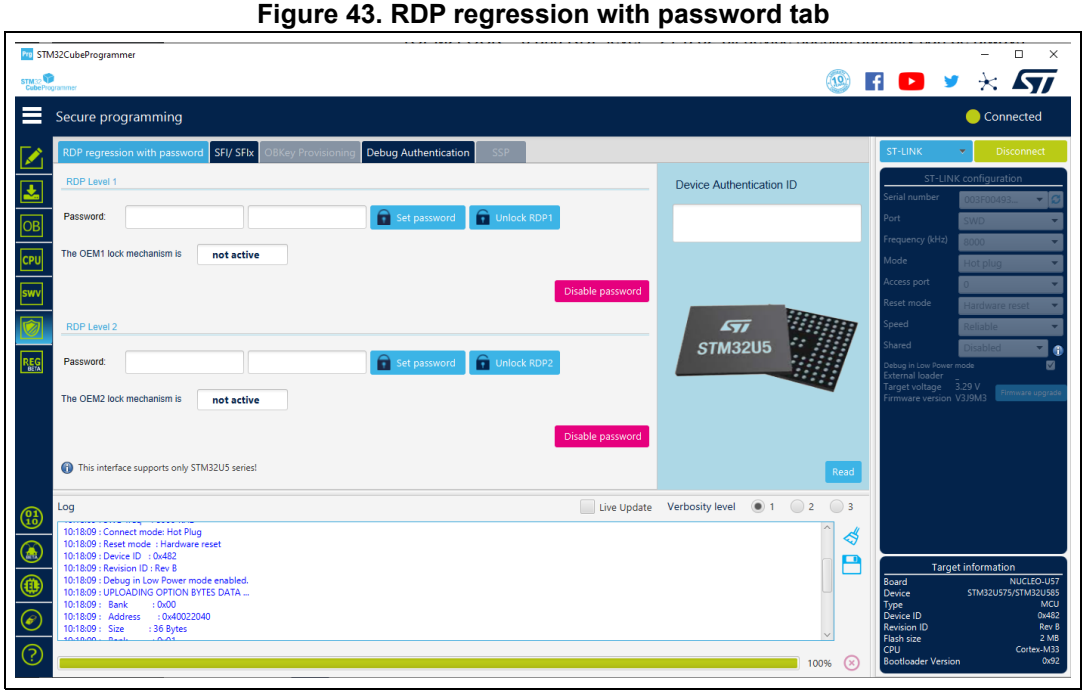


Figure 44. RDP regression with password from level 1

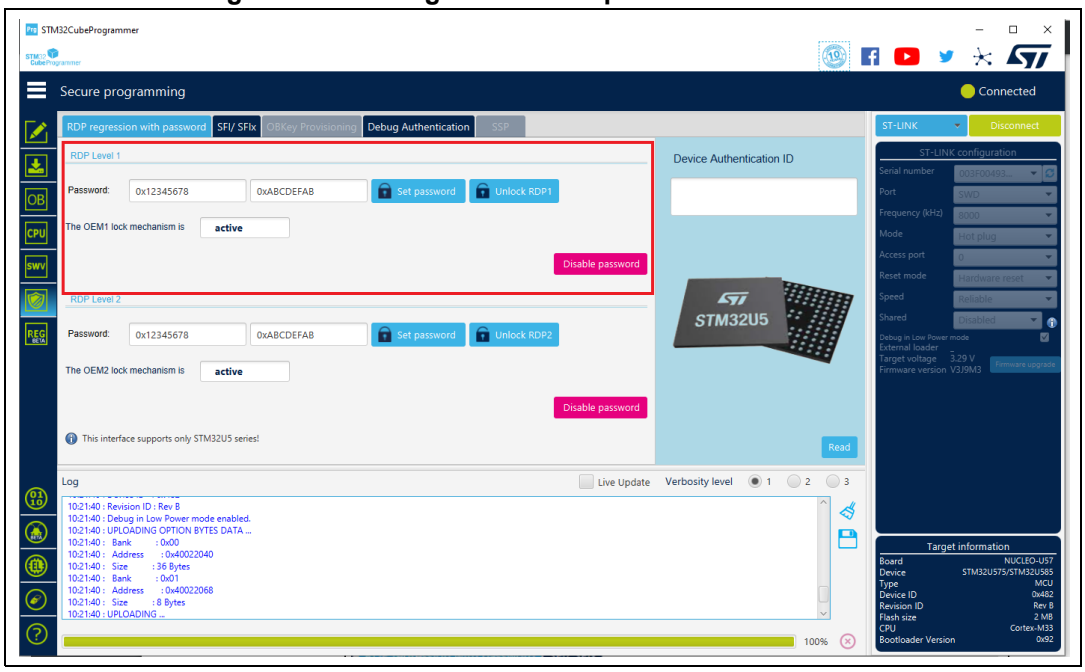


Figure 45. Set OEM1 key (LOCK RDP1)

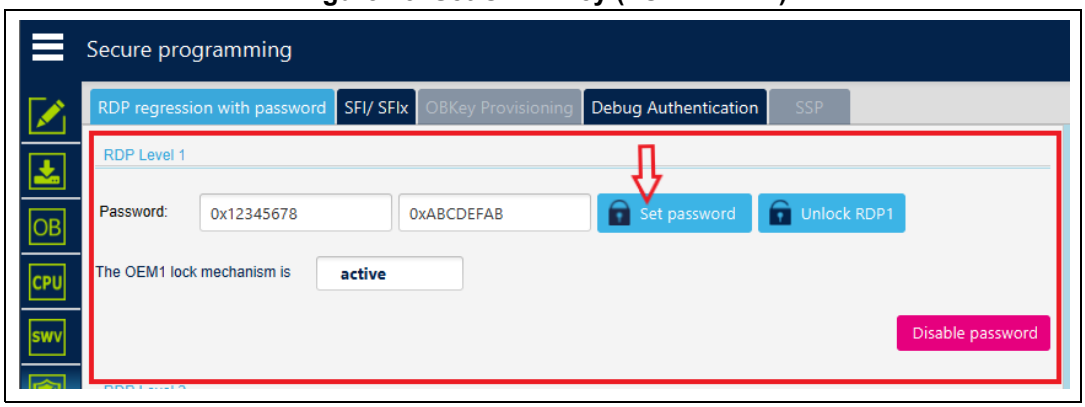


Figure 46. Unlock RDP1

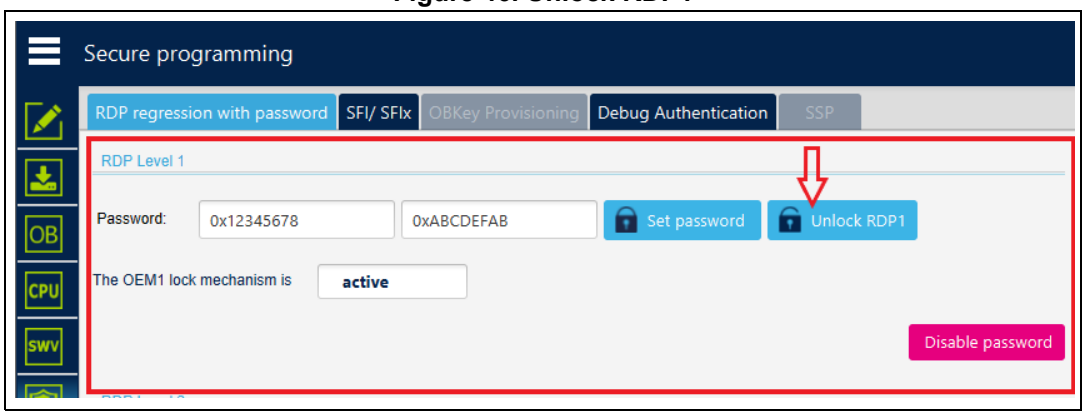


Figure 47. Disable password

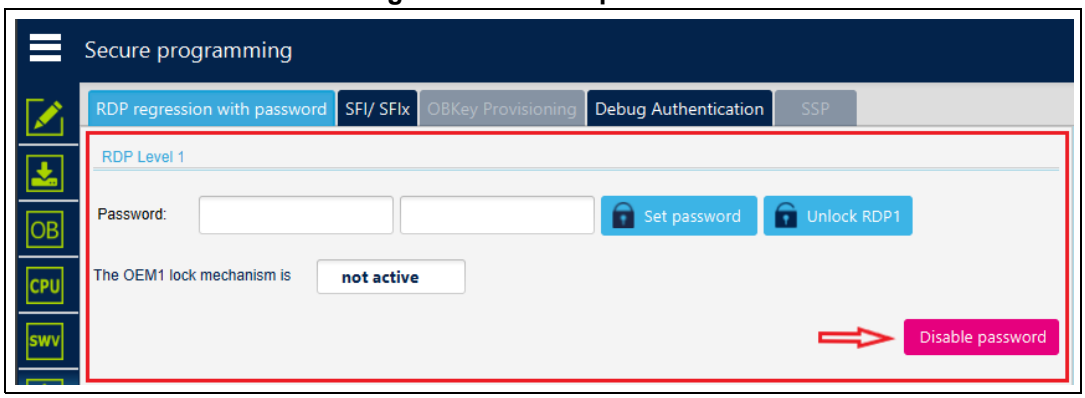


Figure 48. Get authentication ID

2.9.3 SFI/SFIx

SFI GUI

1. Use STM32CubeProgrammer (version 2.11 or higher)
2. Access the SWD/bootloader interface
3. Open Secure Programming interface, then SFI tab (*Figure 49*)
4. Select the license source (from a license file or directly from your connected HSM)
5. Select the sfi file, once selected, the sfi parsed info is displayed as in *Figure 51*
6. Select the RSSe file (if needed)
7. Start sfi sequence (see *Figure 50*)

Note: To open TPC to generate the sfi file, press on the TPC logo, as shown in Figure 52.

Figure 49. SFI/SFIx tab

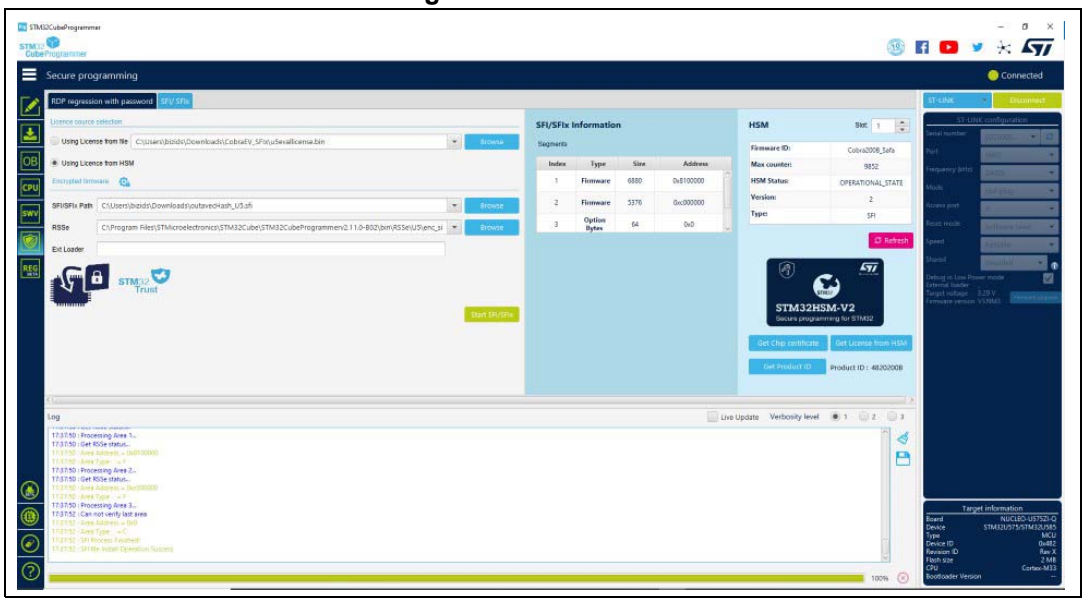


Figure 50. Steps for SFI programming

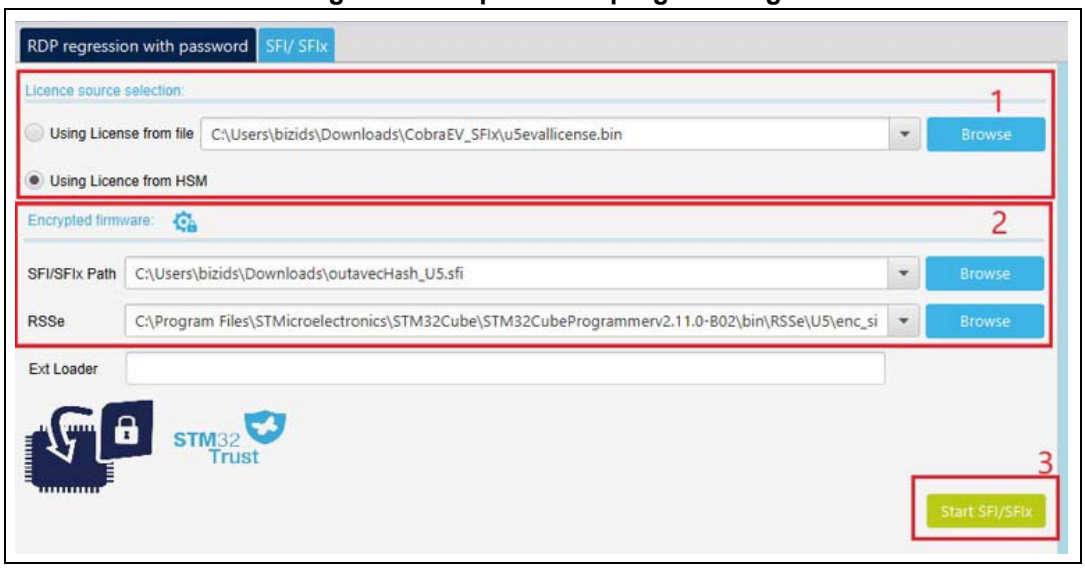
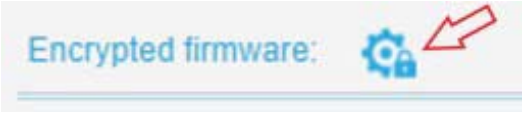


Figure 51. SFI parsed info

Figure 52. Open TPC from STM32CubeProgrammer



SFIx GUI

To perform a successful sfix operation using this graphical interface, perform the same steps described in *SFI GUI*, with two minor modifications:

1. Select an sfix (not an sfi).
2. Select the external loader via External loaders window. Once done, the name is displayed automatically in the text field below (*Figure 53*).

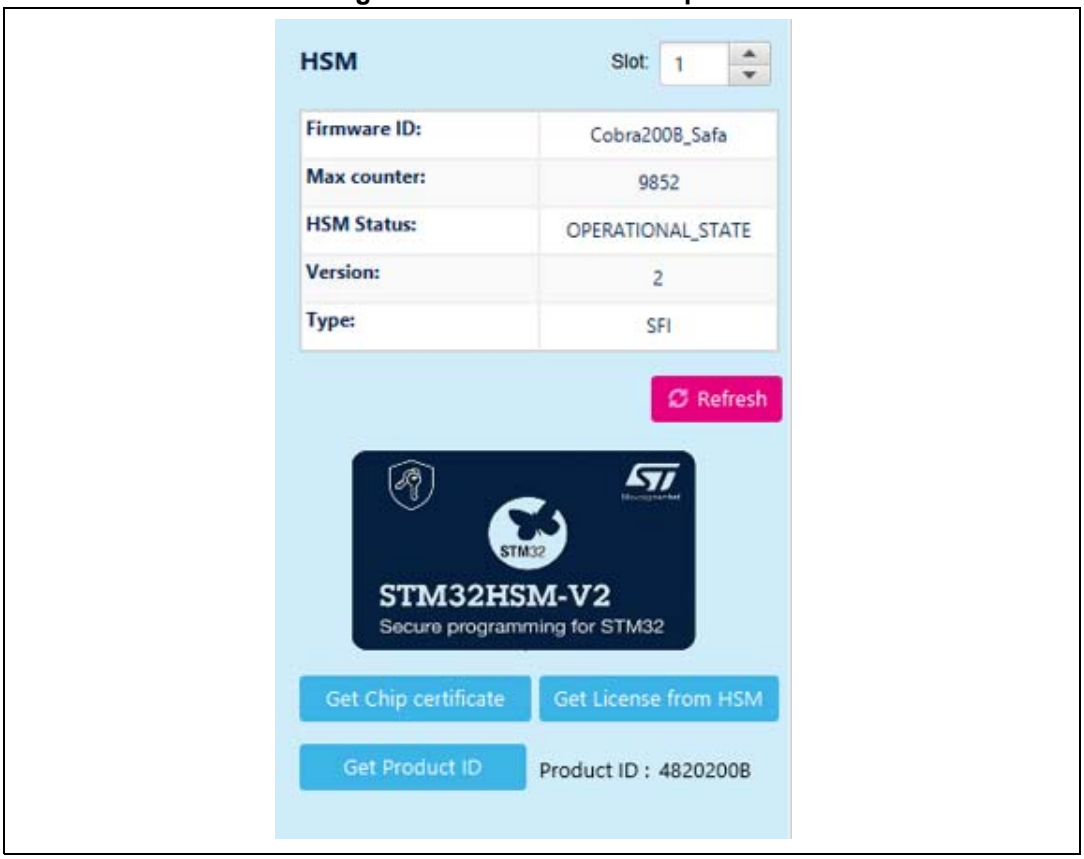
Then you can start the sfix sequence.

Figure 53. Display external loader name

HSM related info

This panel contains all the needed information in the sfi process. It allows the user to read the available HSM information when a card is detected, get the license from HSM, get the chip certificate, and read the product ID.

Figure 54. HSM-related info panel



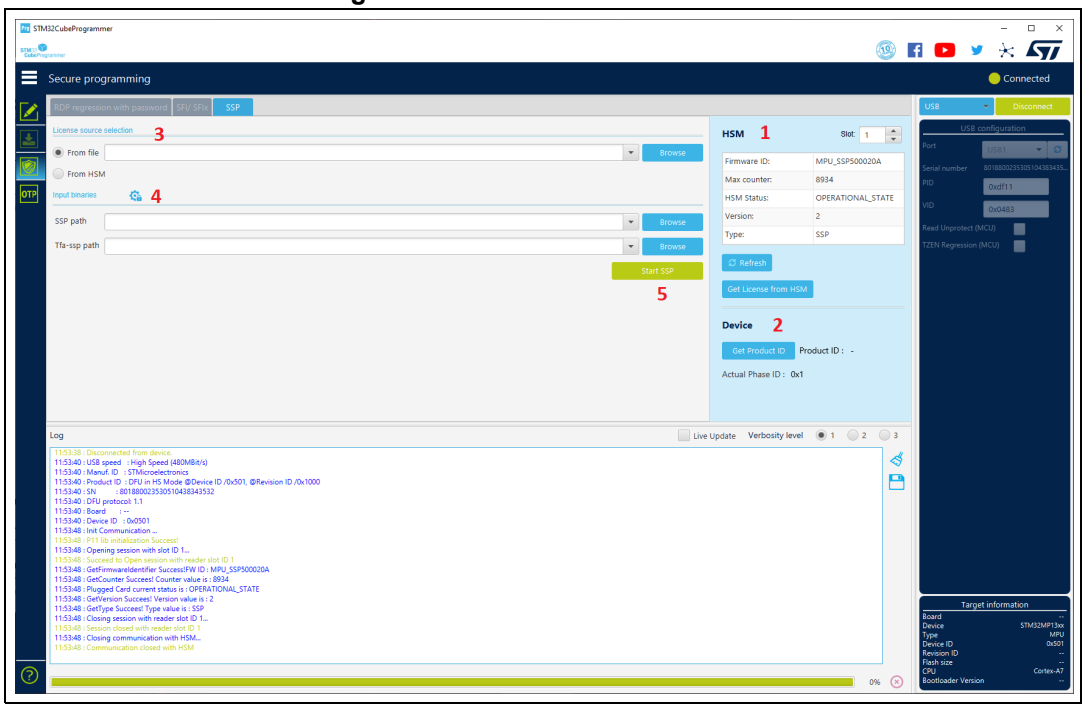
For more details refer to AN5054 “*Secure programming using STM32CubeProgrammer*”, available on *www.st.com*.

2.9.4 SSP

STM32CubeProgrammer user interface (UI) exports several capabilities that can be used to execute the SSP flow.

To open the SSP window, connect an MPU device via DFU interface, click on security panel, and then choose the SSP tab. The window contains the graphical components needed to perform SSP operations.

Figure 55. SSP PRG user interface



The UI is composed of five elements:

1. *HSM section*
2. *Device section*
3. *Input license*
4. *SSP input binaries*
5. *Verify and Start SSP install*

HSM section

This section allows the user to read HSM information when the smart card is detected in the slot selected by the user. With this tab it is also possible to get a license from HSM.

Press “Refresh” button to read and display the related information for the plugged HSM.

Device section

This part allows to get the product ID of the connected device, needed to choose the correct personalization package to be provisioned in the HSM card by the STM32Trusted Package Creator.

If the device is not in a configuration adequate to get the chip certificate, it is mandatory to provide a tfa-ssp file in the “SSP inputs” section. STM32CubeProgrammer will then set the device in the correct state.

The current phase ID is displayed to highlight the device configuration.

Input license

The user must select the source of the license to be used in the SSP flow. Possible sources:

- From file: to select a binary file already generated by the HSM.
- From HSM: the SSP procedure extracts the license directly from the HSM.

SSP input binaries

This part is needed to select the secure input files:

- SSP file: an encrypted SSP image generated by STM32Trusted Package Creator
- tfa-ssp: binary file with .bin or .stm32 extension

To generate a SSP image, launch directly the SSP generation window of STM32Trusted Package Creator tool by clicking on the TPC icon.

Verify and Start SSP install

When the user clicks on “Start SSP” button, STM32CubeProgrammer verifies all mandatory inputs and starts the SSP procedure.

If the operation is successful, an informative popup is displayed, and the device is disconnected.

If an error occurs at any step, the operation stops the flow and displays an error.

Notes

At the end of the flow STM32CubeProgrammer does not make any verification step for the OTP fused words.

If the mentioned tfa-ssp is not the adequate one, the SSP flow fails.

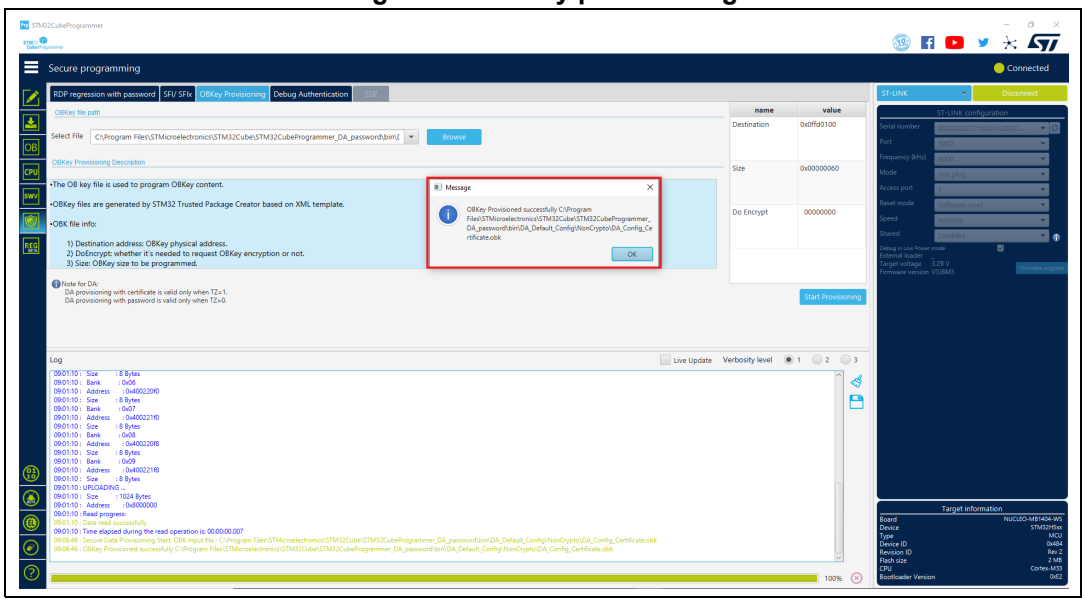
2.9.5 OBKey provisioning (STM32H573/STM32H563/STM32H562 only)

This is a security feature that allows to program OBKey content.

There are two possible cases of OBK provisioning for debug authentication:

- Provisioning with password: before launching it, set TZEN at “0xC3” (disabled), and product state at “0x17” (provisioning).
- Provisioning with certificate: before launching it, set TZEN at “0xB4” (enabled) and product state at “0x17” (provisioning).

Figure 56. OBKey provisioning



OBKey file generation is managed by STM32 Trusted Package Creator. For more information, refer to UM2238.

2.9.6 Debug authentication (STM32H5 series only)

This feature allows regression or open debug when the target is provisioned.

To perform debug authentication the user must provide the needed credentials by following these steps:

1. Discovery: this operation displays the info about the target.
   To make sure that provisioning is correctly performed, “Integrity status” field must be checked: it must indicate “0xEAEAEAEA” as value (displayed in the log). This operation cannot be launched while the target is connected.
2. Credentials input: beside discovery related data, debug authentication panel displays at this step a form including the necessary inputs:
   a) TZEN = 0xC3 (disabled): the form includes only password file path. In this case, only full regression with password is available.
   b) TZEN = 0xB4 (enabled): the form includes key and certificate path inputs. In this case, the user can select the possible permission.

Figure 57. Debug authentication with password

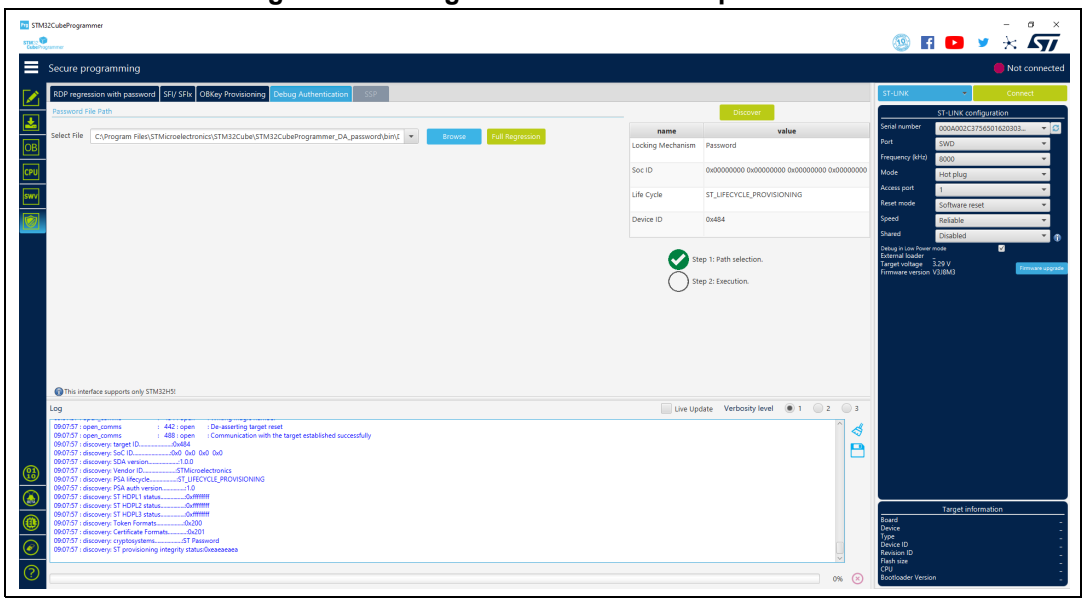
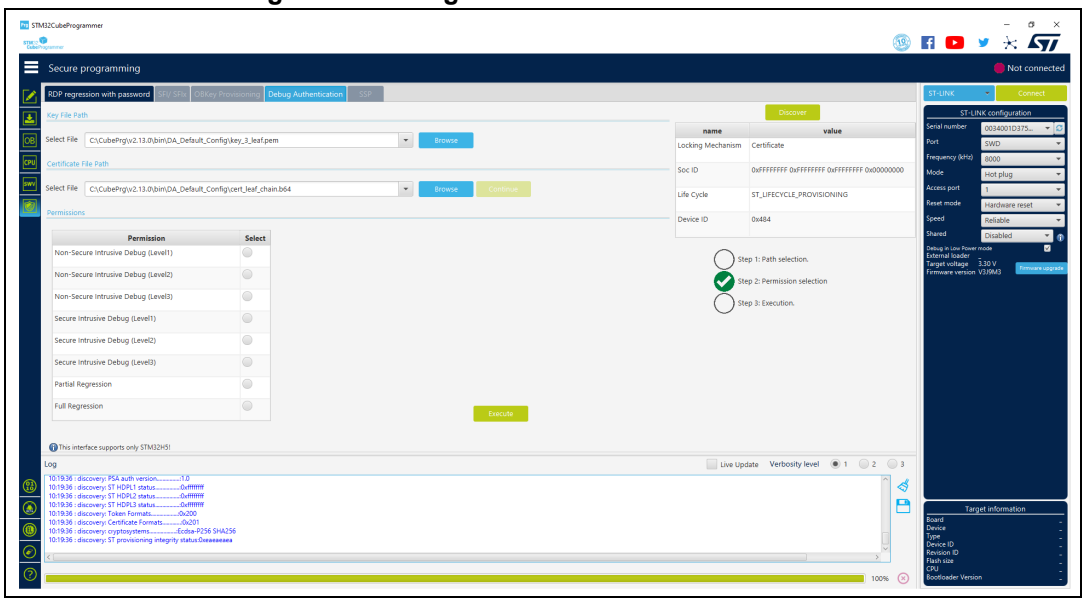


Figure 58. Debug authentication with certificate



Note: For STM32H503, only a full regression with password is available.

2.10 STM32CubeProgrammer Script Manager platform for MCUs

2.10.1 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.

2.10.2 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 reads 32-bit data memory from a specified address, and puts it in the used variable.

#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: Comments in the Script Manager can be added by using "//", as shown in the examples.

Script Manager example 1 (CLI and Script macro), see *Figure 59*

```
-c port=swd
-e 0 1
#Write32(0x08000000,0xAAAABBBB)
#var0=#Read(0x08000000)
#Display(#var0)
```

Script Manager example 2, see *Figure 60*

```
-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 59. Output of Script Manager example 1

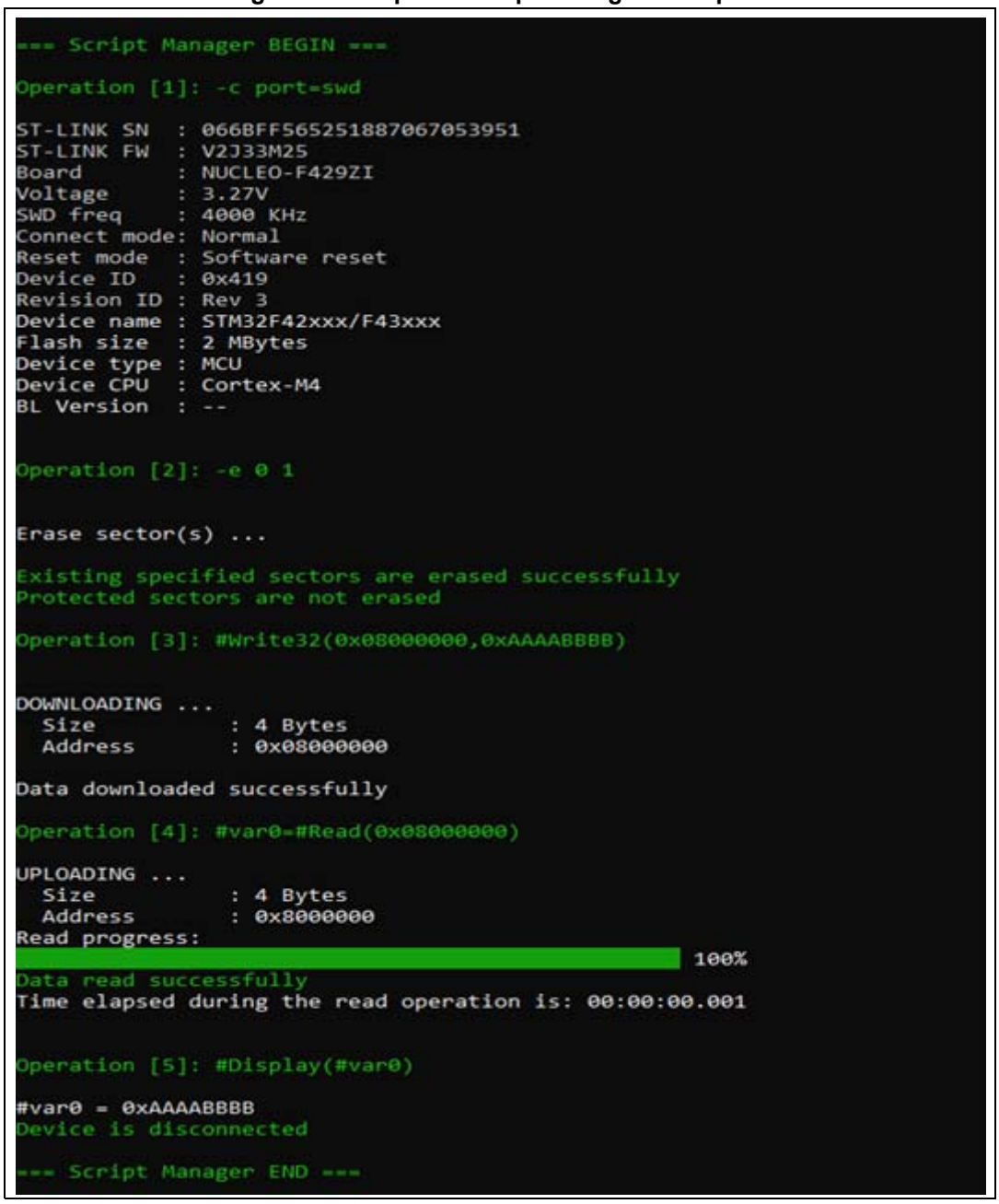
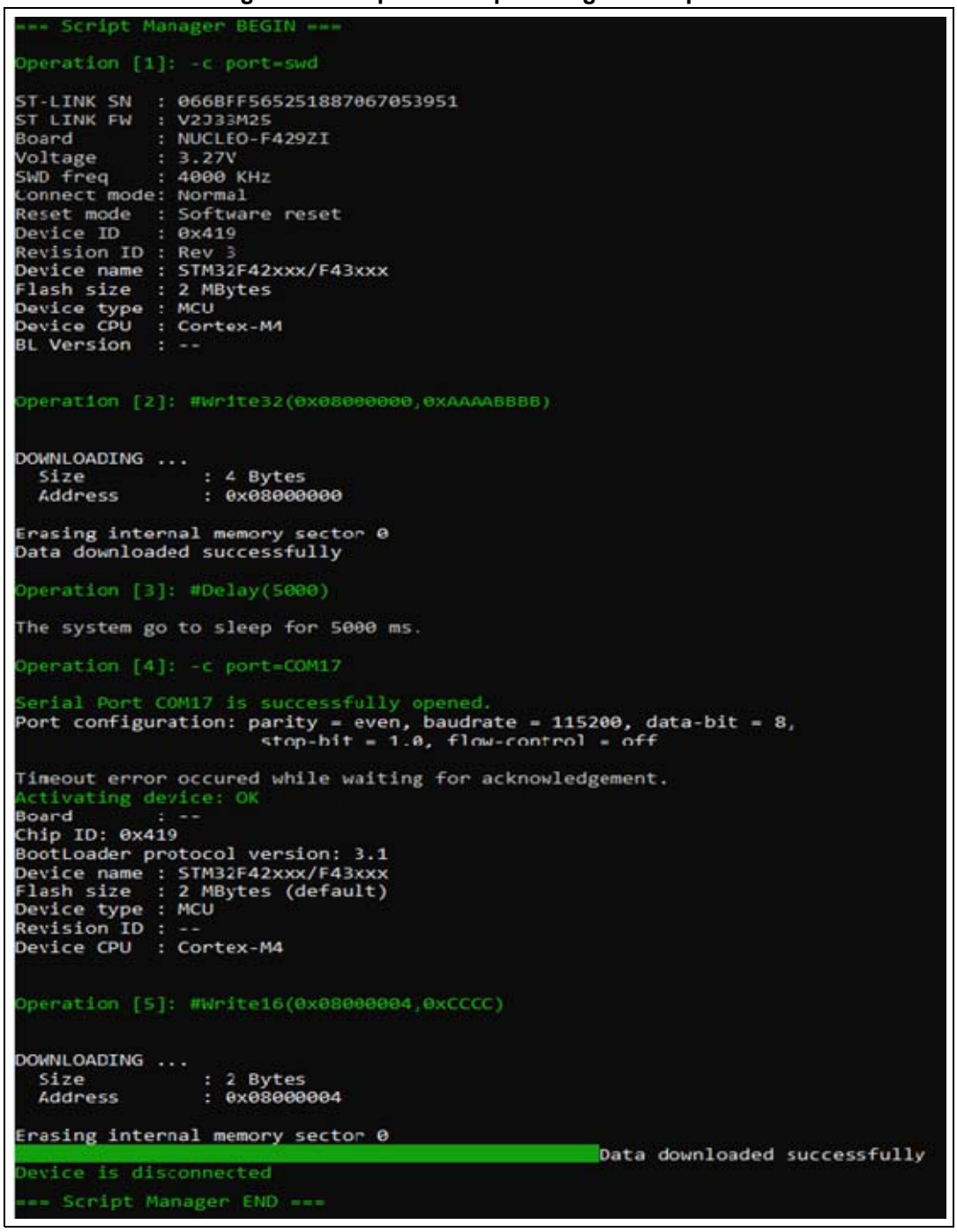


Figure 60. Output of Script Manager example 2



2.10.3 Loops and conditional statements

The Script Manager supports three macros for looping and conditional statements:

```
#Start
While (***)
{
 Command Line or macros
}
#End
```

```
#Start
for (***)
{
 Command Line or macros
}
#End
```

```
#Start
if (***)
{
 Command Line or macros
}
else
{
 Command Line or macros
}
#End
```

To use the conditional statements (if, else) and the loopings (While, for), begin with **#Start**, and finish with **#End**.

If-else condition example

```
-c port=swd
-e 0
#x=#Read(0x08000000)

#Start
if (#x > 0x1000)
{
  #Display("Condition 1")
  #Display(#x)
  #Write32(0x08000000,0x1123)
}
else
{
  #Display("Condition 2")
  #Display(#x)
  #Write32(0x08000008,0x1124)
}

#End
```

For loop

```
-c port=swd
#ADD=0x08000004
#x=#Read(0x08000004)
#Start
#Display(#x)
```

```
for(#ADD=0x08000000;#ADD<0x0800000C;#ADD=#ADD+4)
{
   #x=#Read(#ADD)
   #Display(#x)
}
#End
```

While loop (example 1)

```
-c port=swd
-e 0
#Write32(0x08000008,0xCCCCCCCC)
#ADD=0x08000000
#x=#Read(#ADD)

#Start
while(#x!=0xCCCCCCCC)
{
    #Display(#x)
    #ADD=([#ADD]+(4))
    #x=#Read(#ADD)
}
#End
```

While loop (example 2)

```
-c port=swd

-e 0

#Write32(0x08000000,0xAAAAAAAA,0xBBBBBBBB,0xCCCCCCCC,0xDDDDDDDD)
#ADD=0x08000000
#x=#Read(#ADD)
#Display(#x)

#Start

while(#x!=0xDDDDDDDD)
{
    #Display(#x)
    #ADD=([#ADD]+(4))
    #x=#Read(#ADD)
}

#End
```

2.11 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 61 shows the steps to connect via the new USB DFU panel, and *Figure 62* the main window of STM32CubeProgrammer after connection.

Figure 61. Connect via USB DFU panel

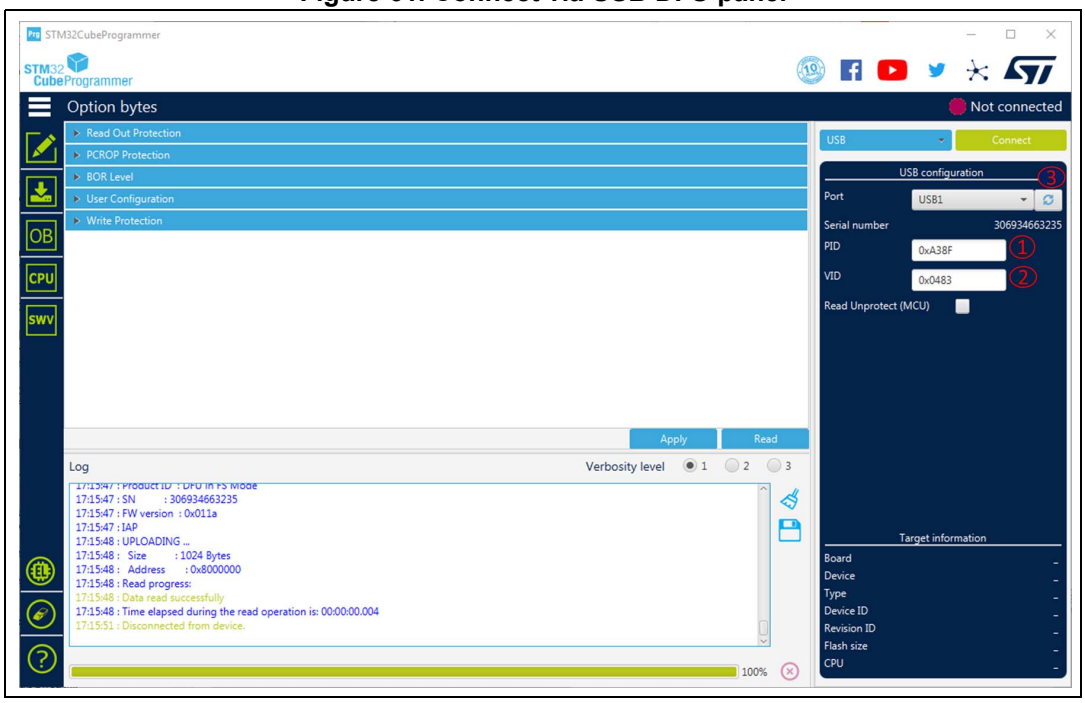
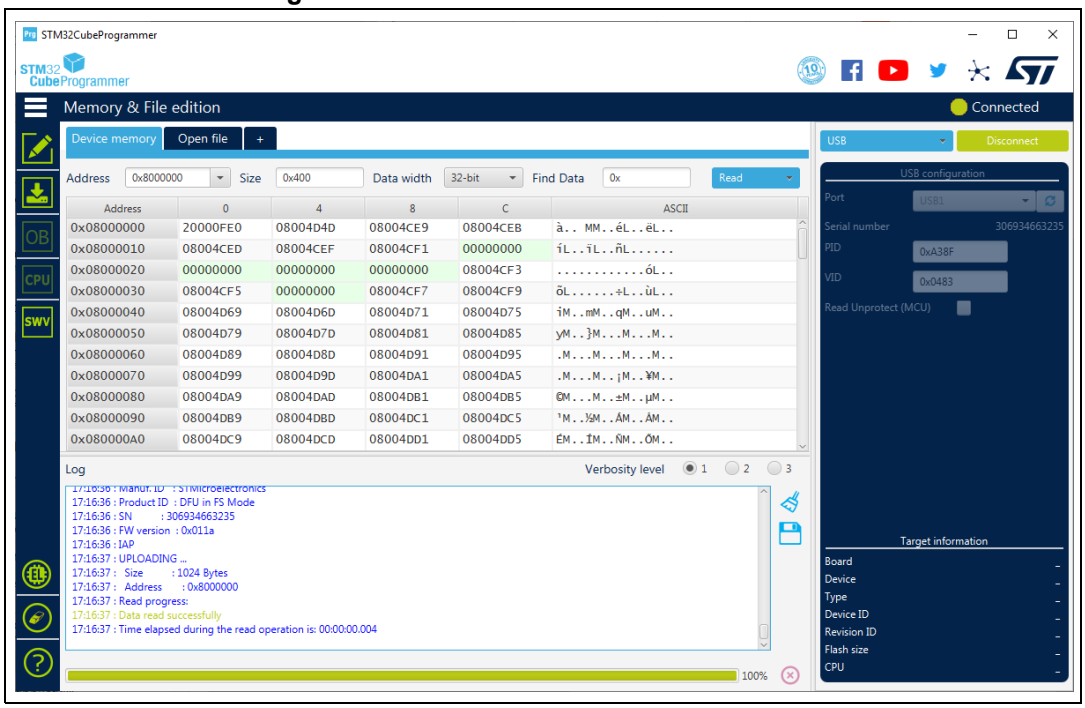


Figure 62. Main window after the connection



Note: For CLI mode check the Section 3.2.1: Connect command.

2.12 SigFox™ credentials

As soon as an STM32WL device is connected, the window shown in *Figure 63* 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

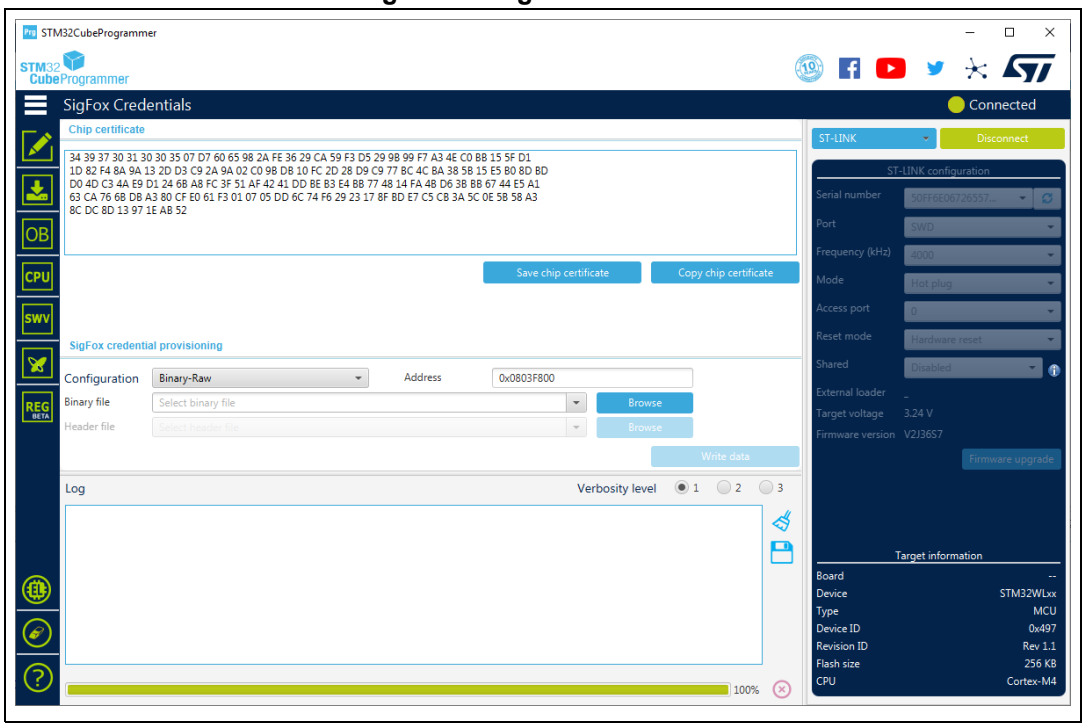
Use the binary file returned by the back-end web-service. The size of this file must be equal to 48 bytes, it is written at the default address 0x0803E500.

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).

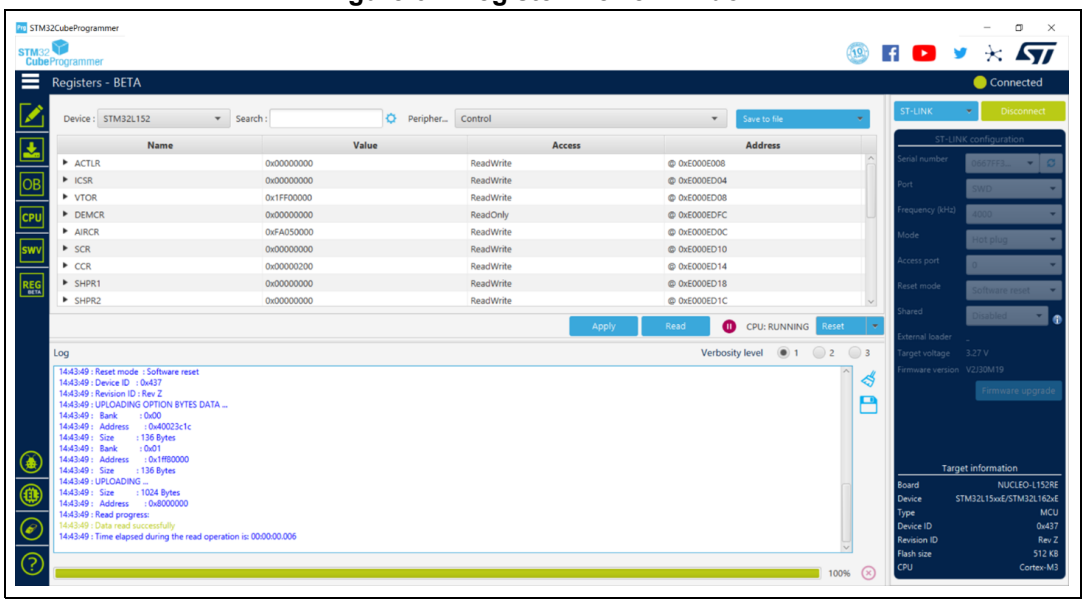
Figure 63. SigFox credentials



2.13 Register Viewer

STM32CubeProgrammer supports the Register Viewer feature (see *Figure 64*), 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 64. Register Viewer window



Note: The register viewer is available only 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).

2.14 Hard Fault analyzer

2.14.1 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.

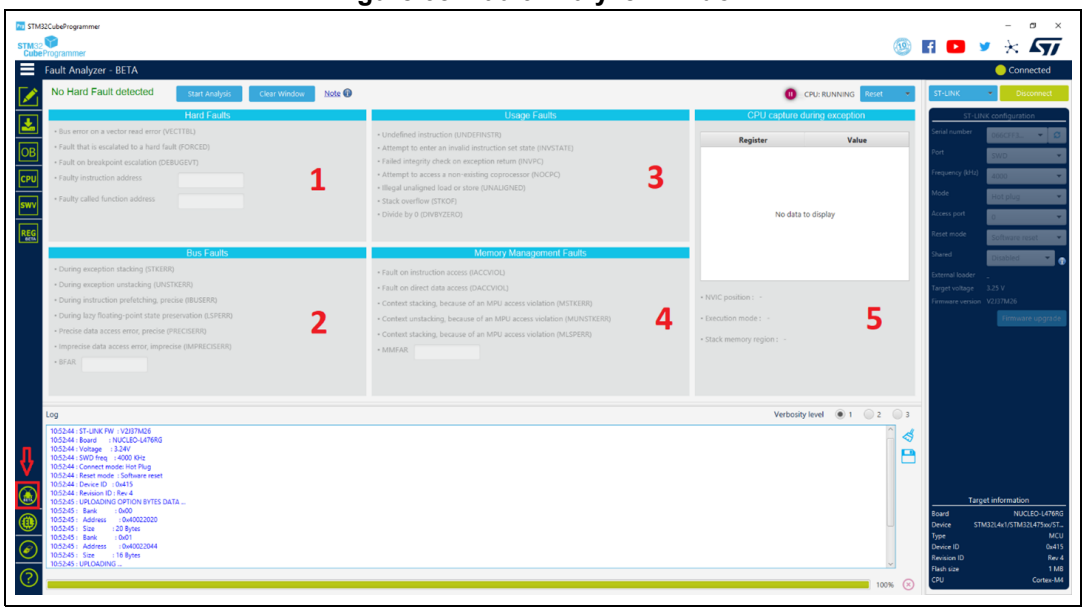
Possible detected fault exceptions:

- 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 65*, the Fault Analyzer window has five main sections.

Figure 65. 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.
   c) NVIC position: indicates the number of the interrupt imposing the error, if it is “-“ the interrupt/exception vector has no specific position.
   d) Execution mode: indicates the operation mode Handler/Thread.
   e) Stack memory region: indicates the used stack memory during the fault, Main or Process stack.

2.14.2 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;

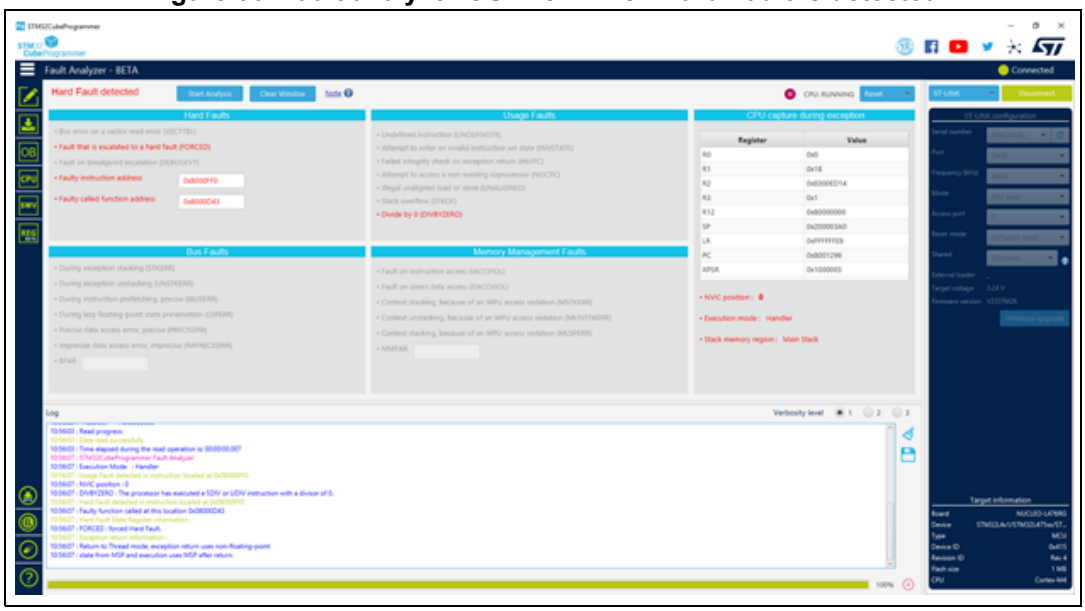
c = a / b;

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 66. Fault analyzer GUI view when Hard Fault is detected

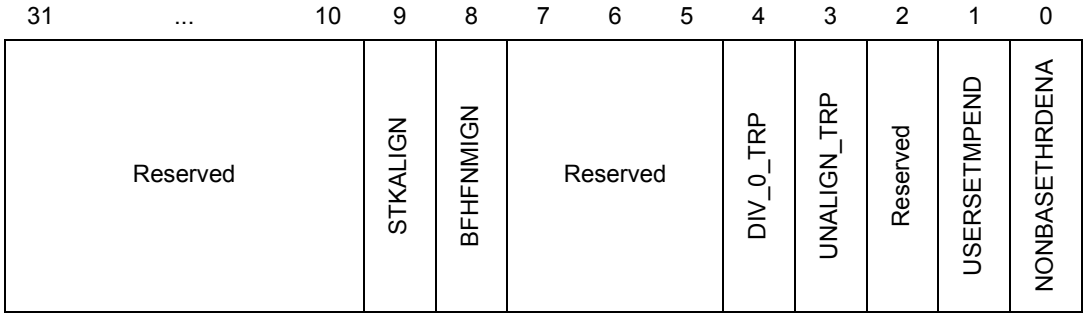


2.14.3 Fault analyzer note

Fault analyzer can be unable to detect untracked faults 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.

Figure 67. CCR bits



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

- 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 is detected an informative popup is displayed to indicate that you must reproduce the scenario and restart the analysis.

2.14.4 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 connecting a Cortex-M33-based device (such as an 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.

2.15 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 68*)
- Example 2:
  `STM32_Programmer_CLI.exe -c port=swd -fillmemory 0x08000000 size= 0x10 pattern=0XCC datawidth=32` (*Figure 69*)

Figure 68. Example 1

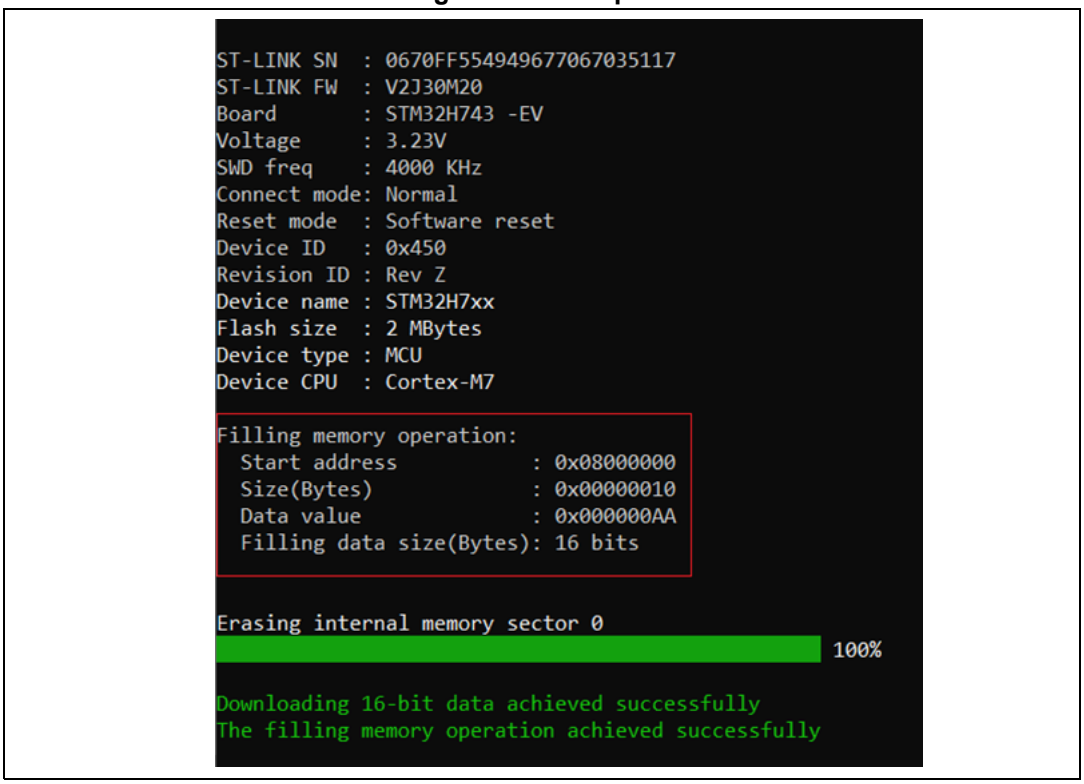
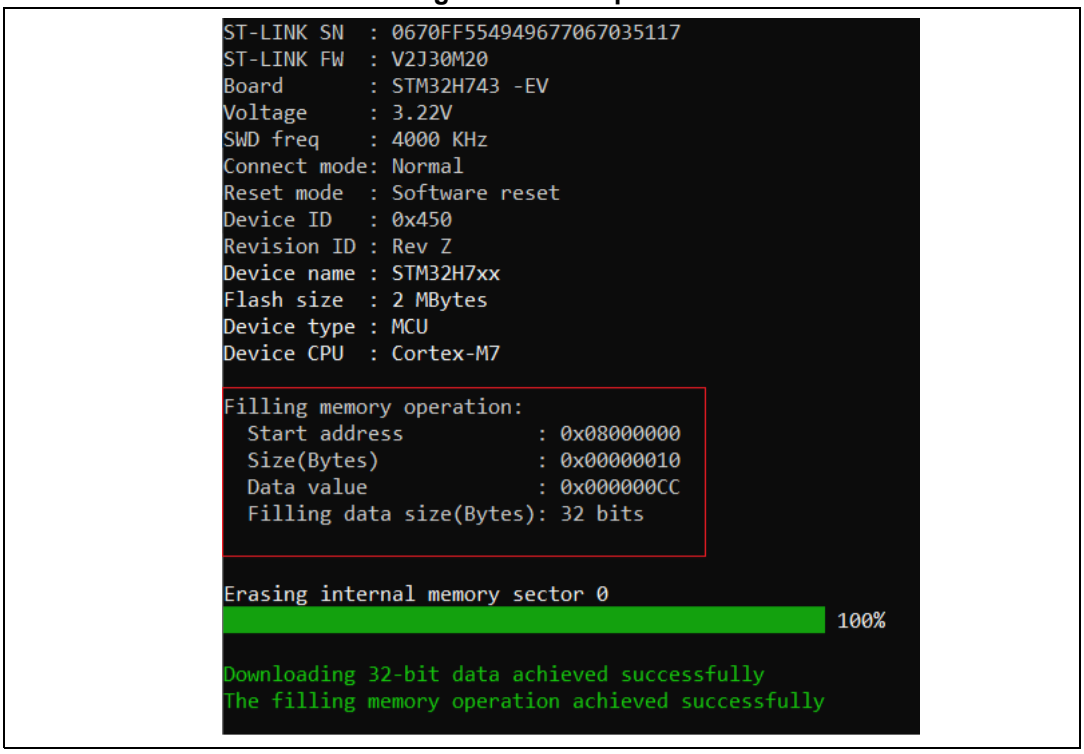


Figure 69. Example 2



2.16 Fill memory operation

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

Figure 70. Sub-menu displayed from “Read” combo-box

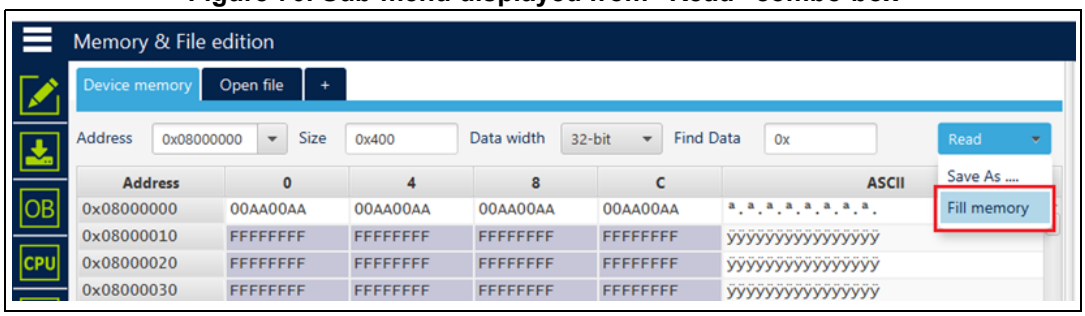


Figure 71. Sub-menu displayed with right click on “Device memory” tab

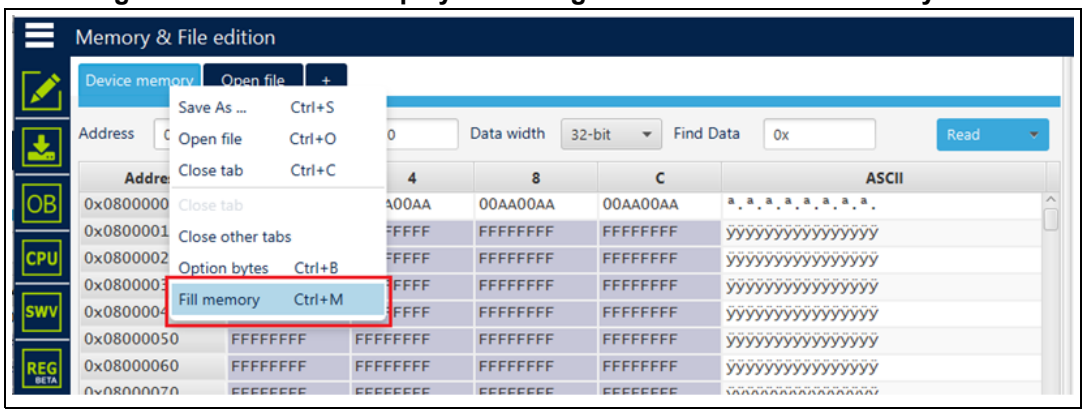
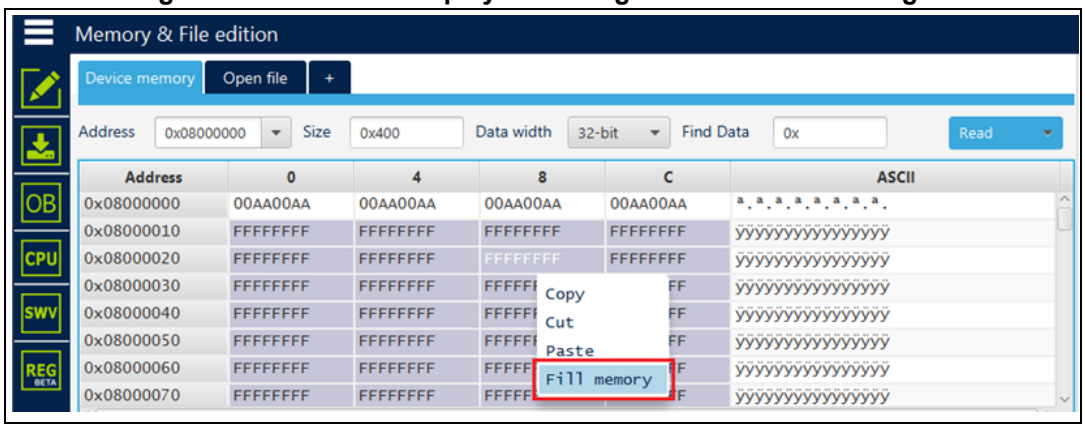


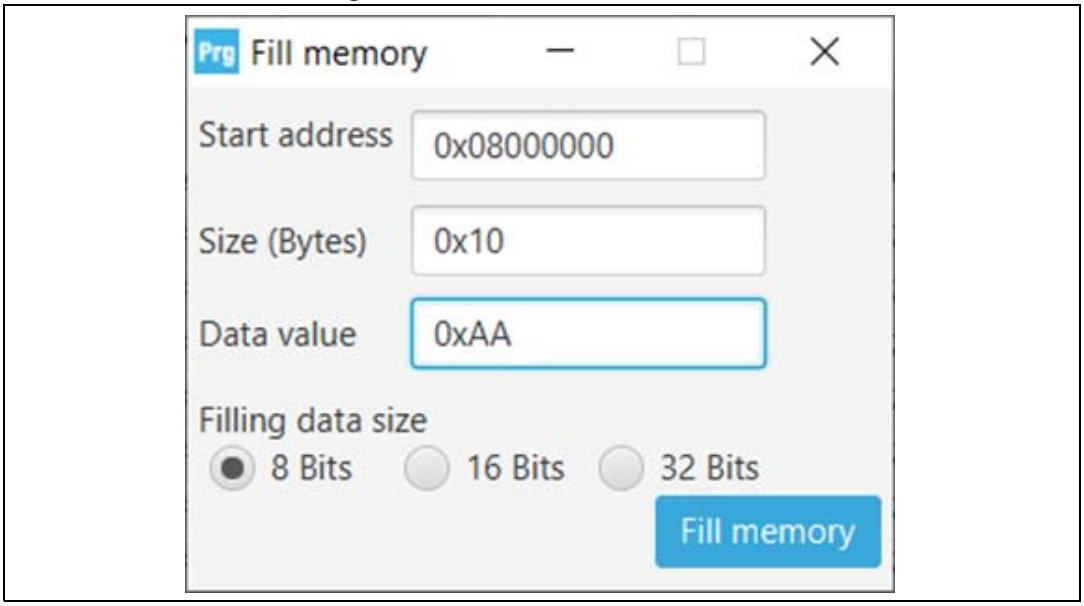
Figure 72. Sub-menu displayed with right click on the cell of grid



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 73*).

Figure 73. Parameters initialization



2.17 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 74. Example 1: memory is not blank at address 0x08000014

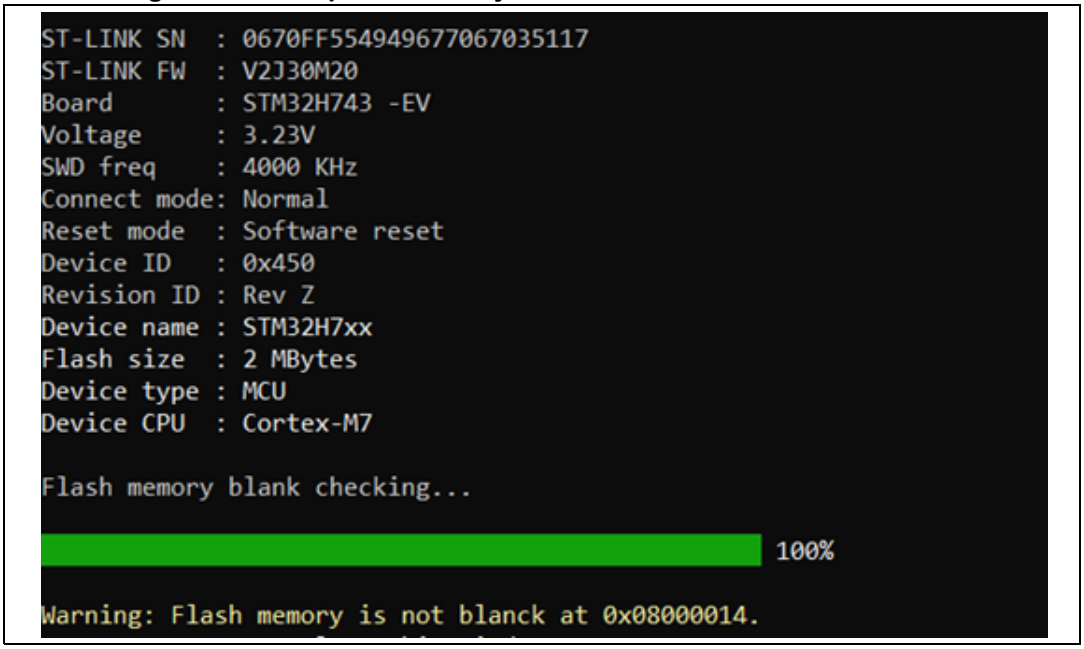
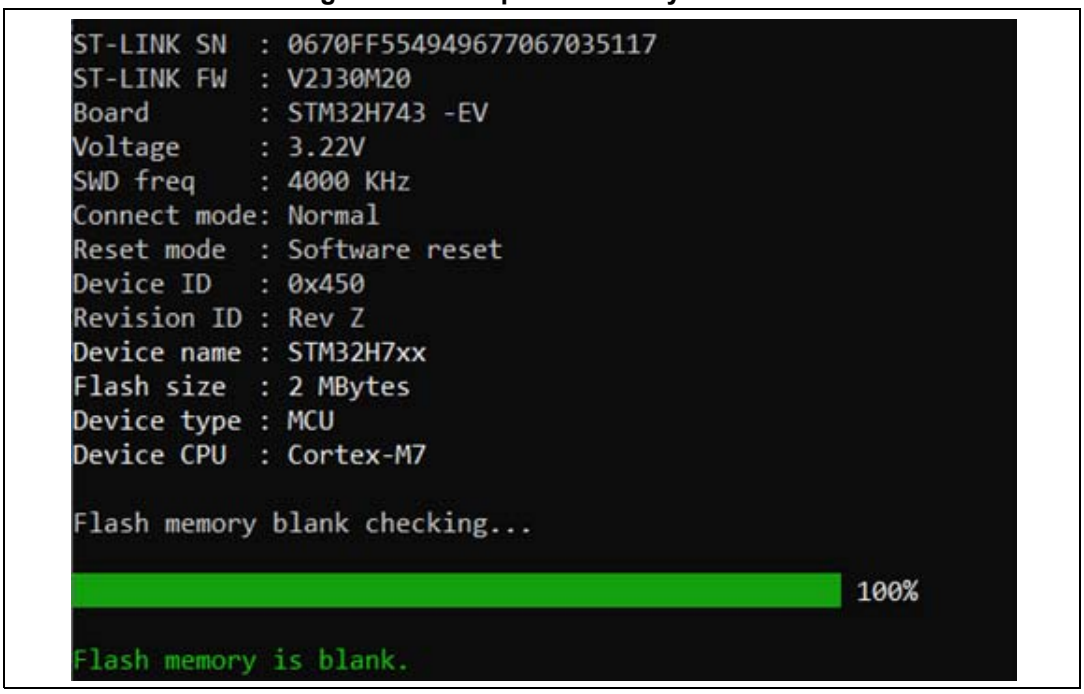


Figure 75. Example 1: memory is blank



2.18 Blank check operation

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

Figure 76. Sub-menu displayed from “Read” combo-box

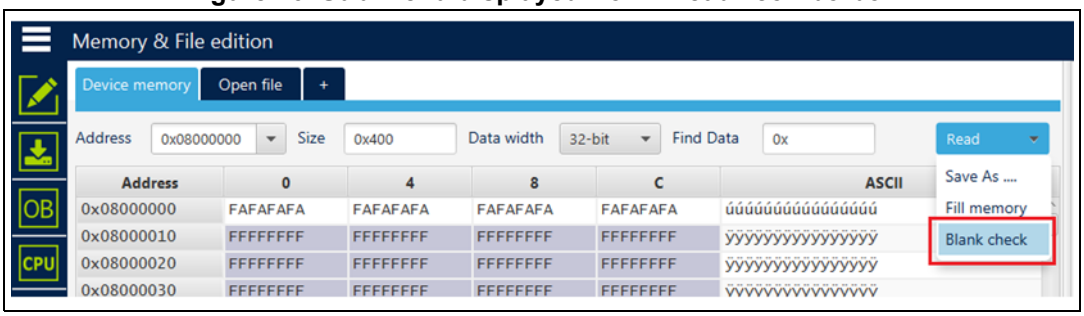


Figure 77. Sub-menu displayed with right click on “Device memory” tab

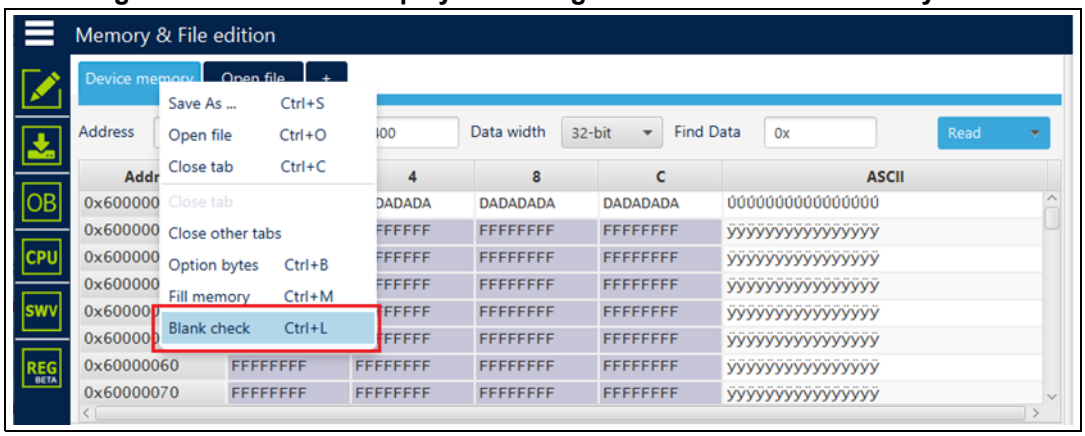
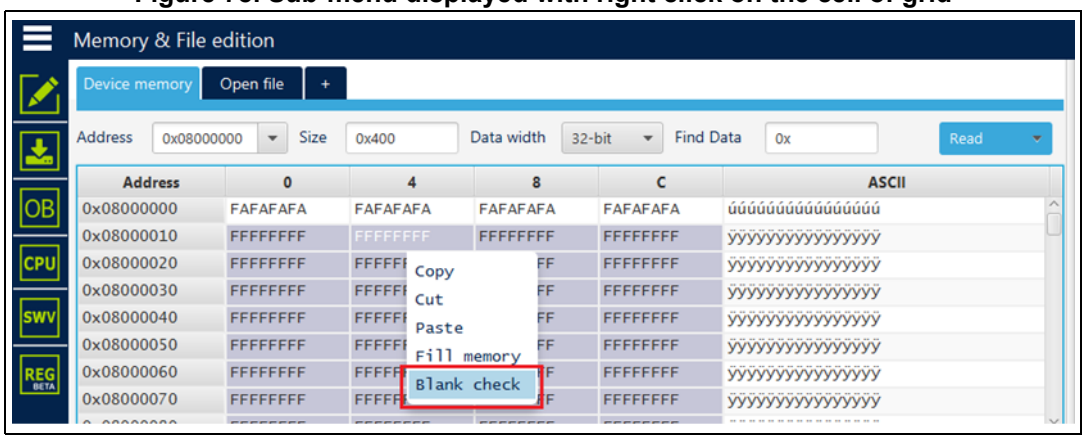


Figure 78. Sub-menu displayed with right click on the cell of grid



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 79*.

The expected results are shown in figures *80* and *81*.

Figure 79. First address with data

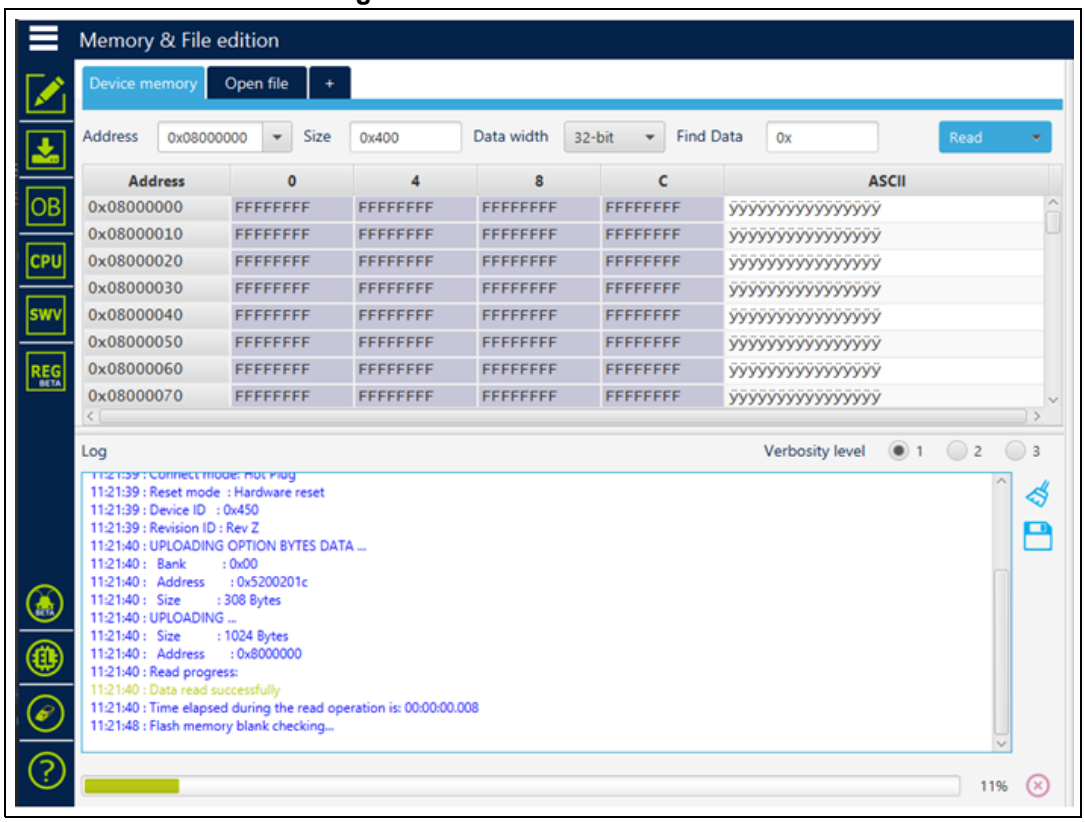


Figure 80. Example 1: memory is blank

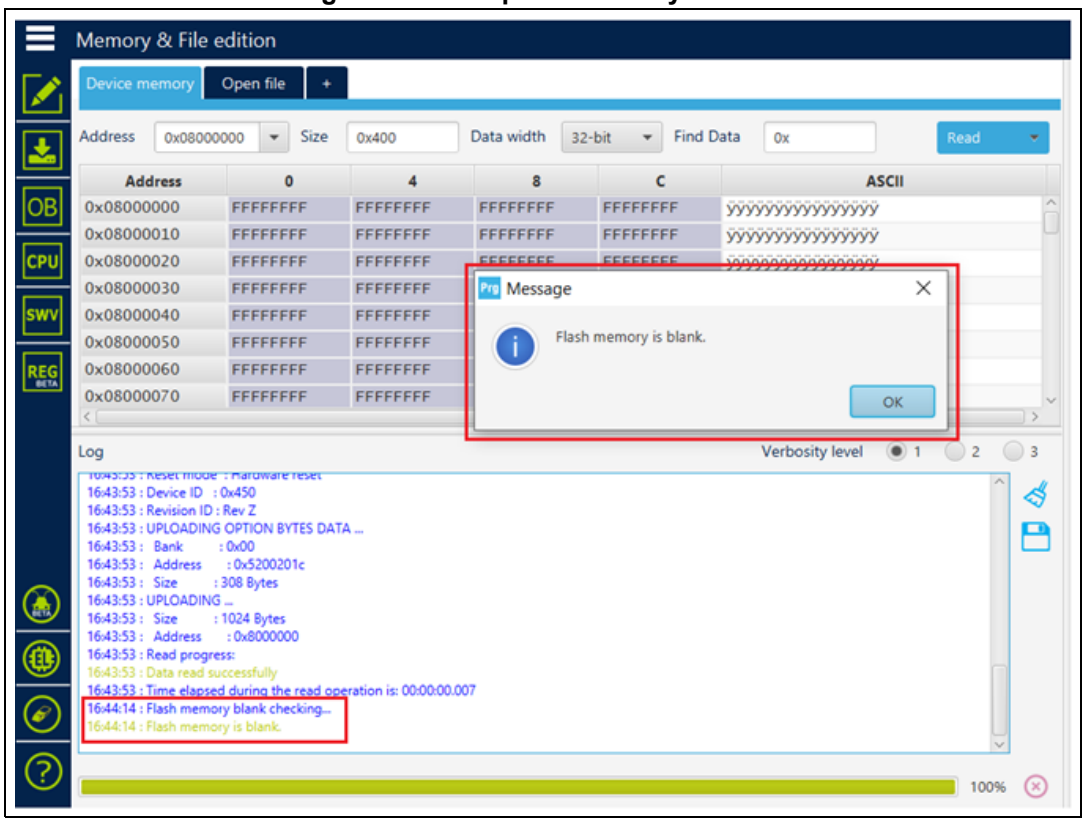
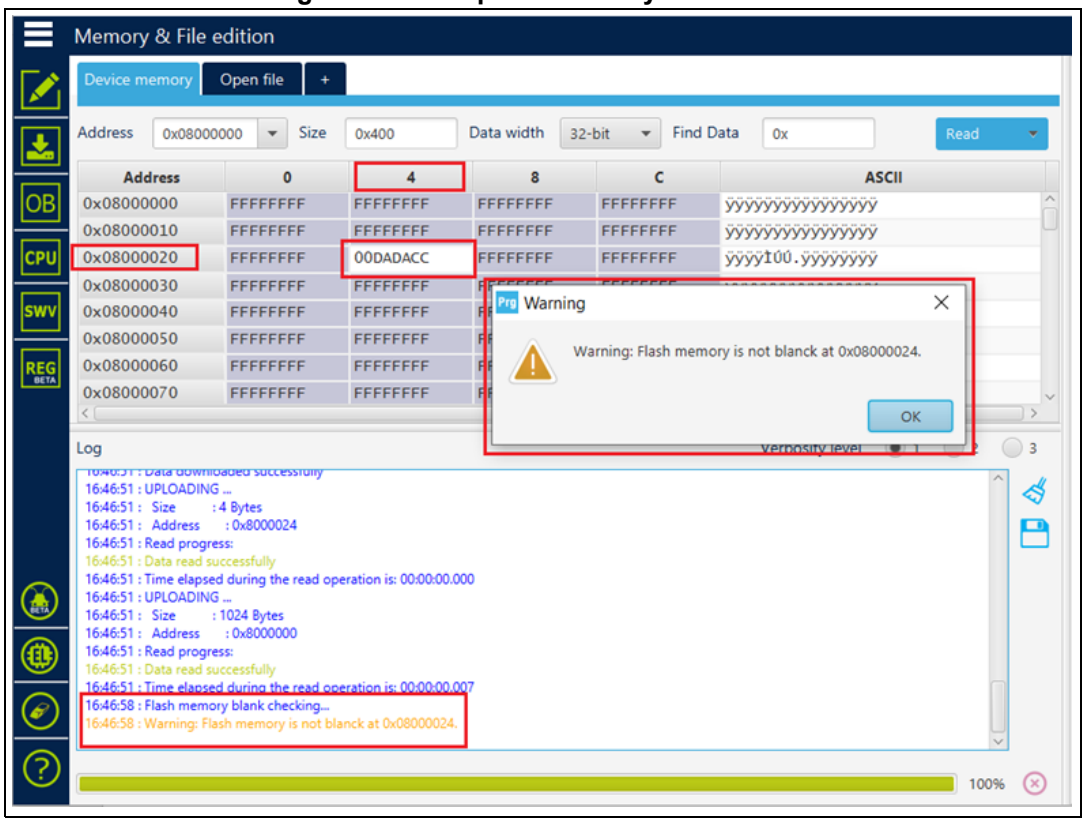


Figure 81. Example 2: memory is not blank



2.19 Compare flash memory with file

Description: Compares the MCU device memory content with a binary, hex, srec, 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 82. Sub-menu displayed from “Read” combo-box

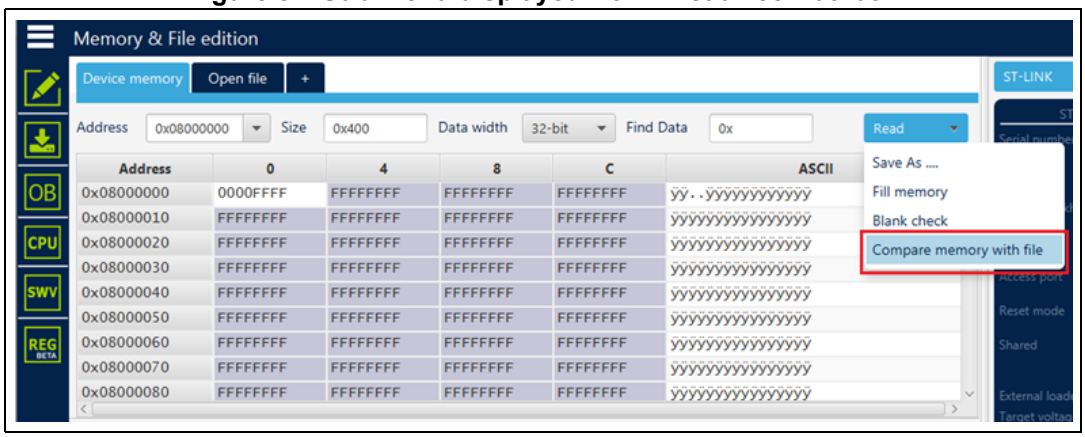


Figure 83. Sub-menu displayed with right click on “Device memory” tab

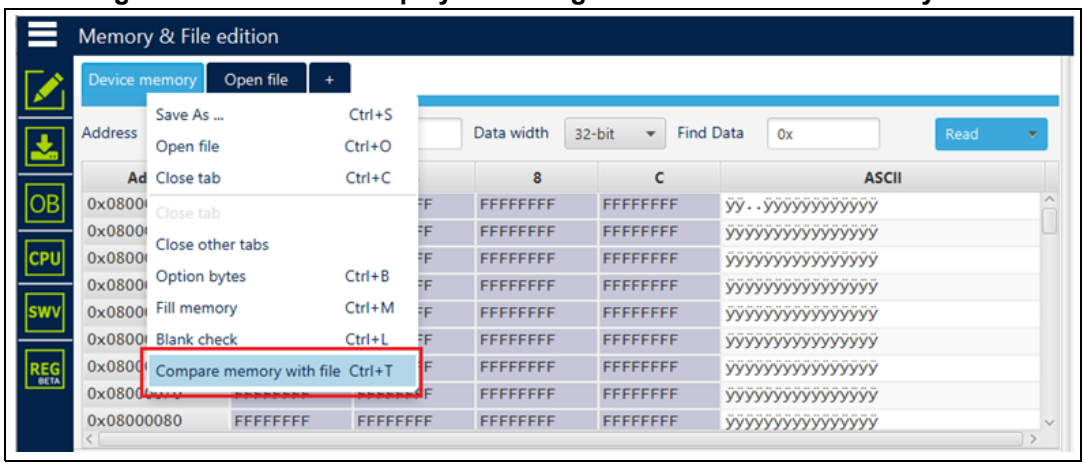


Figure 84. Sub-menu displayed with right click on the cell of grid

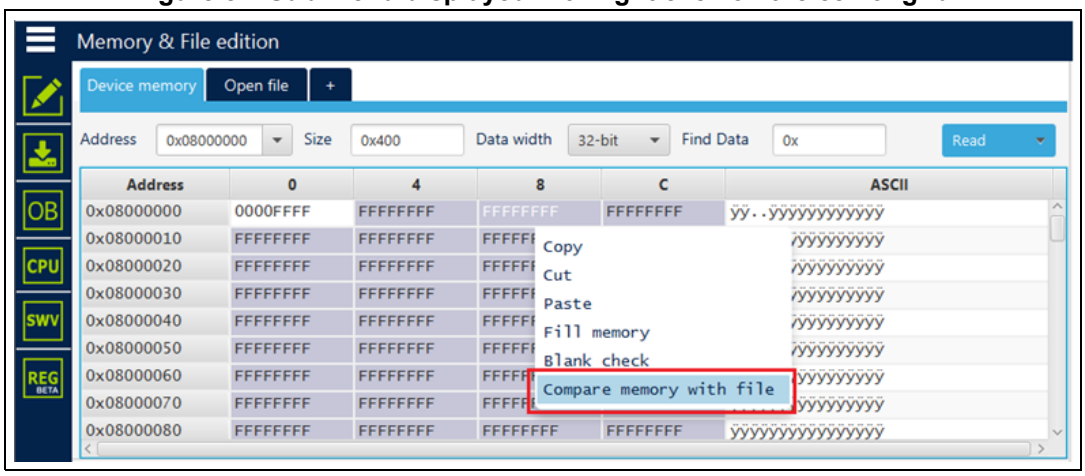


Figure 85. Sub-menu displayed with add tab button

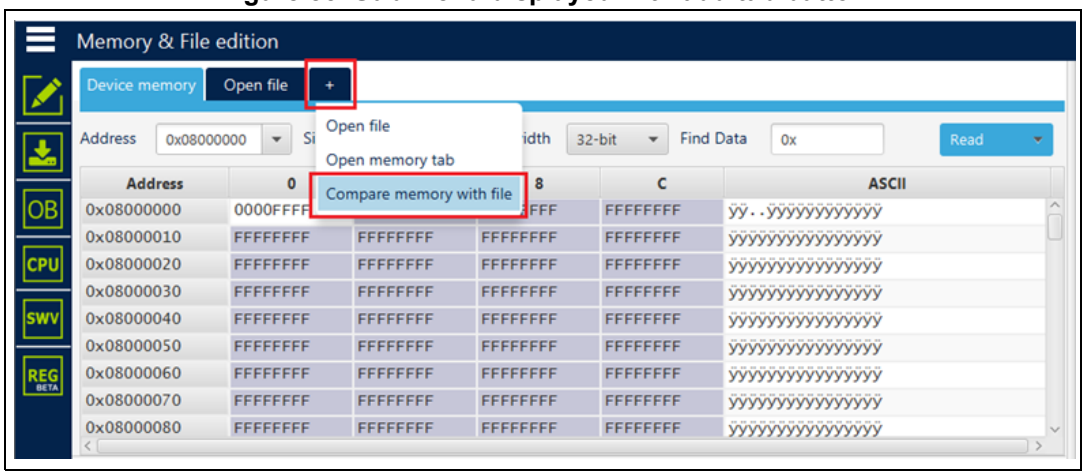


Figure 86. Sub-menu displayed with right click on the opened file tab

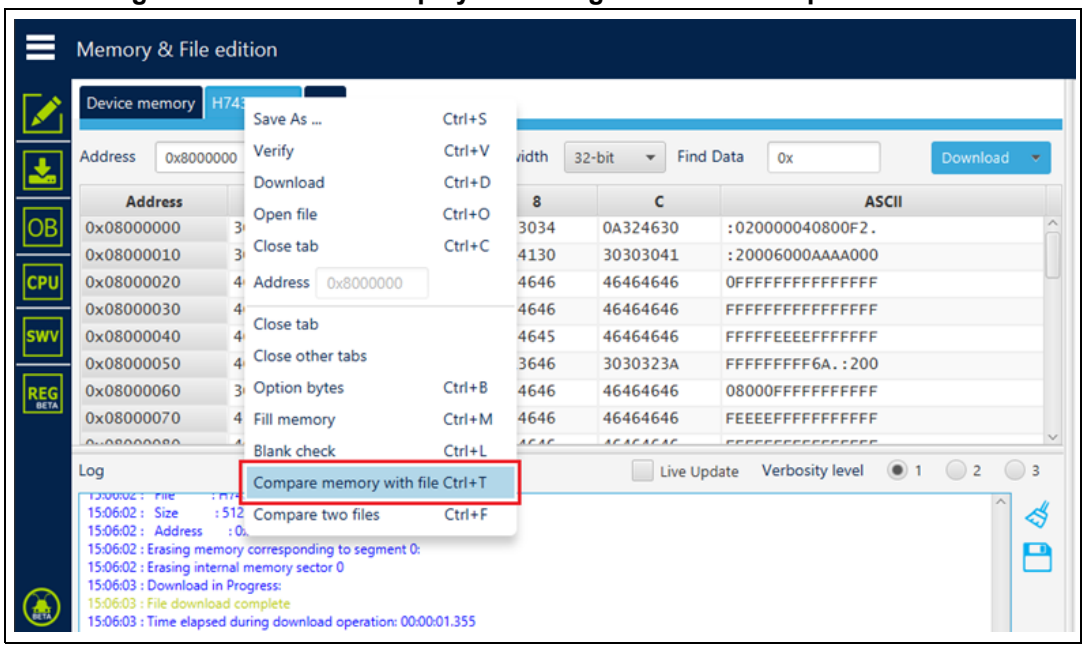
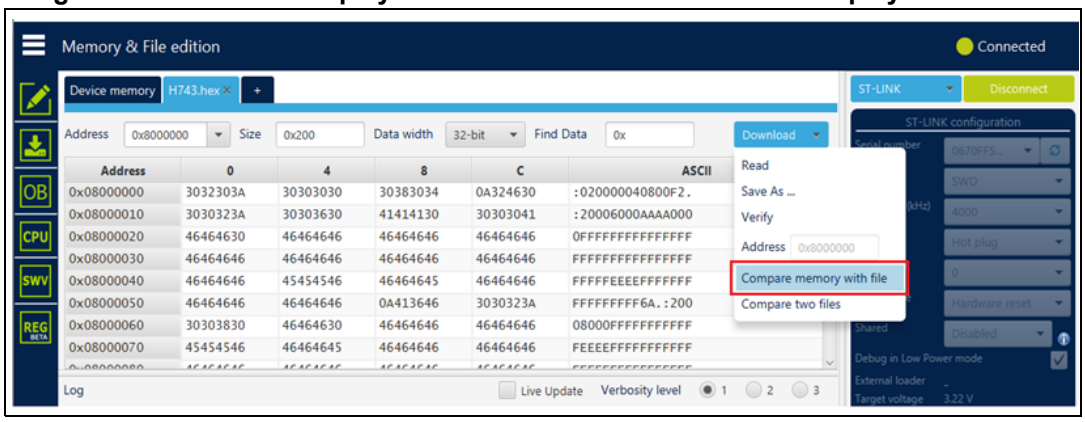


Figure 87. Sub-menu displayed from “Download” combo-box displayed in file tab



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 88. Data width: 32 bits

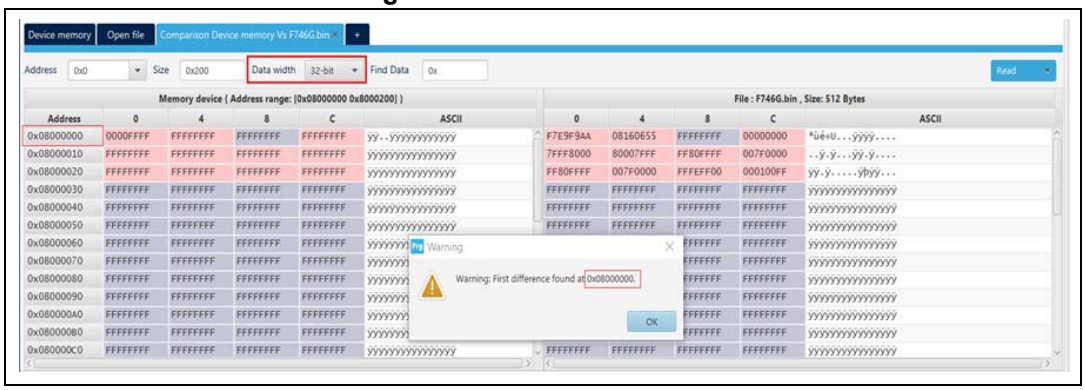


Figure 89. Data width: 16 bits

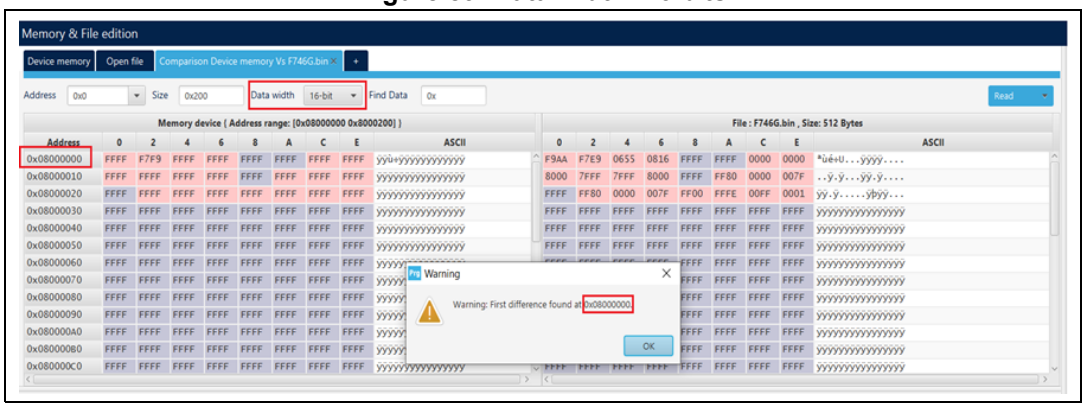
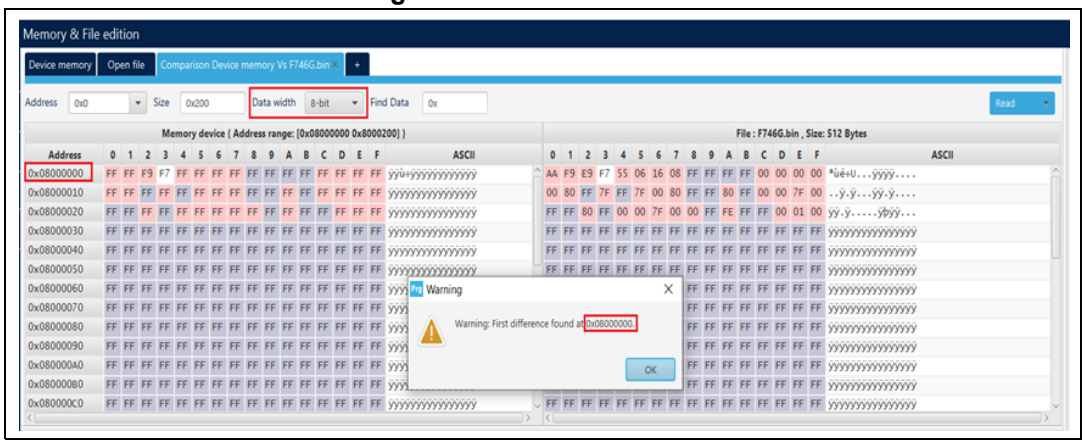


Figure 90. Data width: 8 bits



Example 2: Difference between external flash memory and hex file

Figure 91. Data width: 32 bits

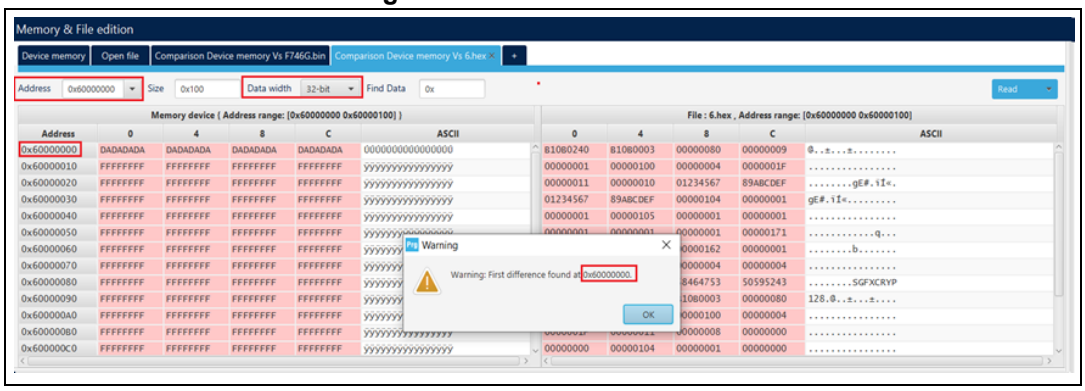


Figure 92. Data width: 16 bits

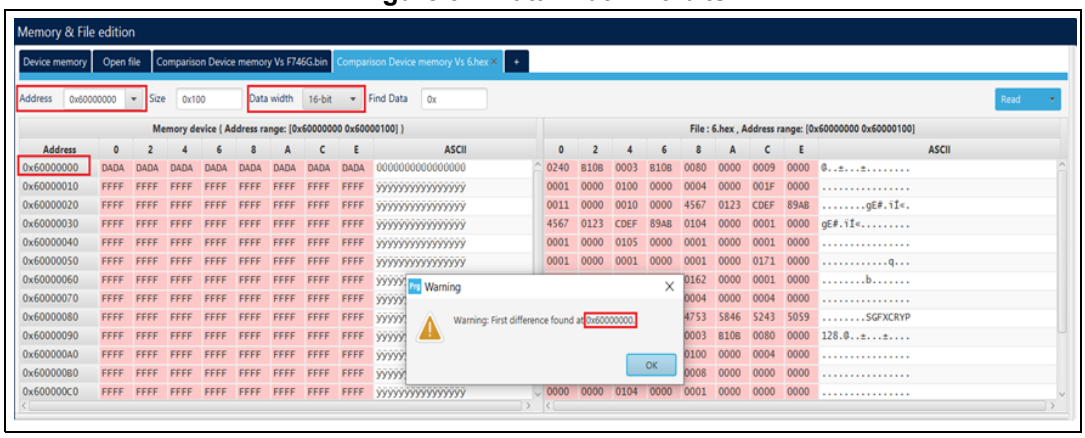
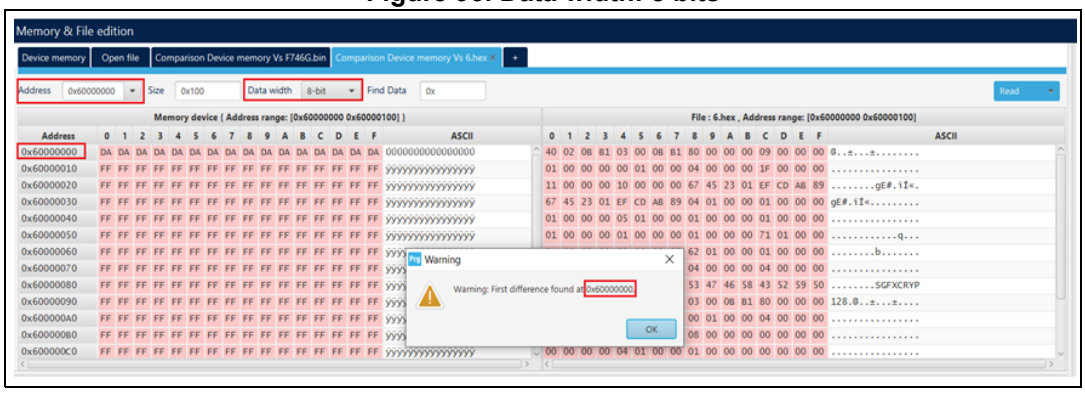


Figure 93. Data width: 8 bits



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 94. Before editing the flash memory

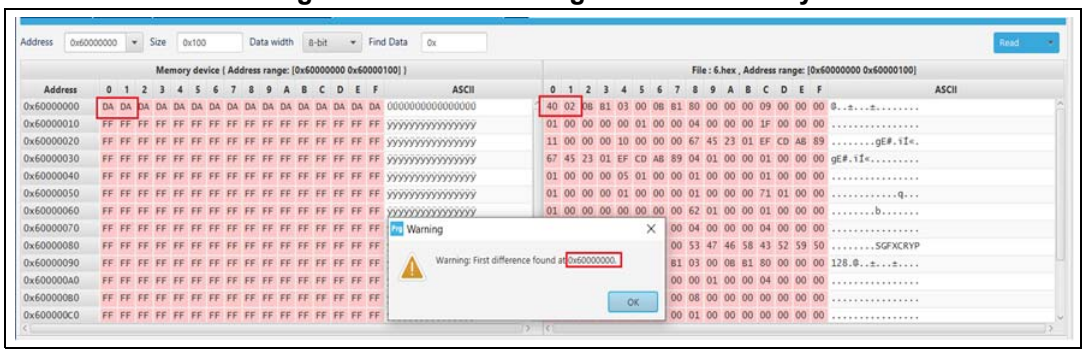
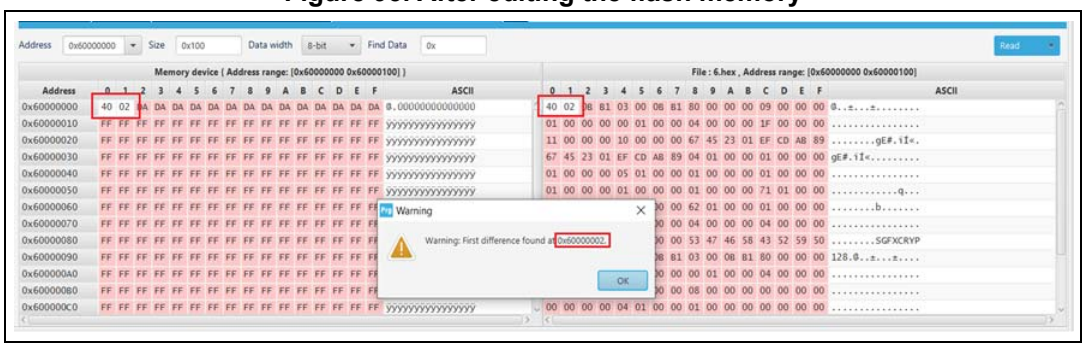
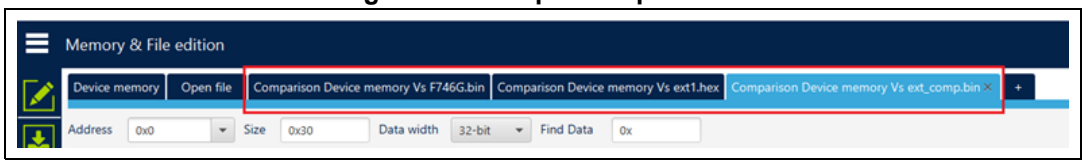


Figure 95. After editing the flash memory



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

Figure 96. Multiple comparisons



2.20 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 97. Sub-menu displayed from “Read” combo-box in device memory tab

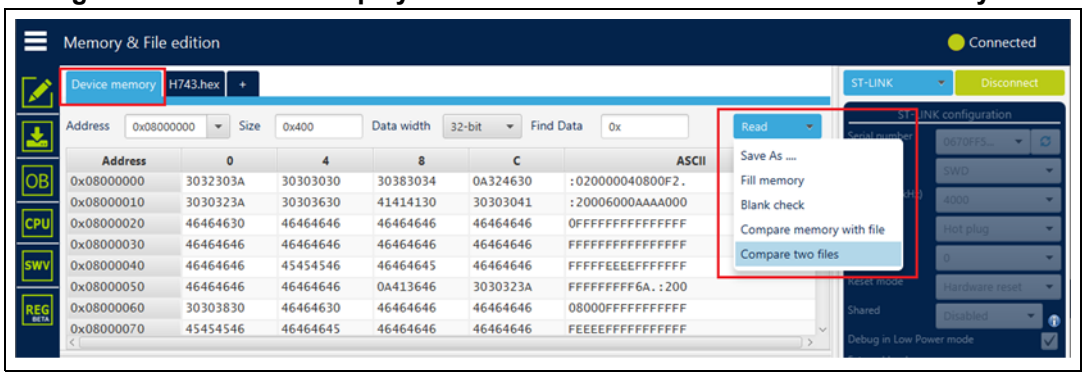


Figure 98. Sub-menu displayed with right click on “Device memory” tab

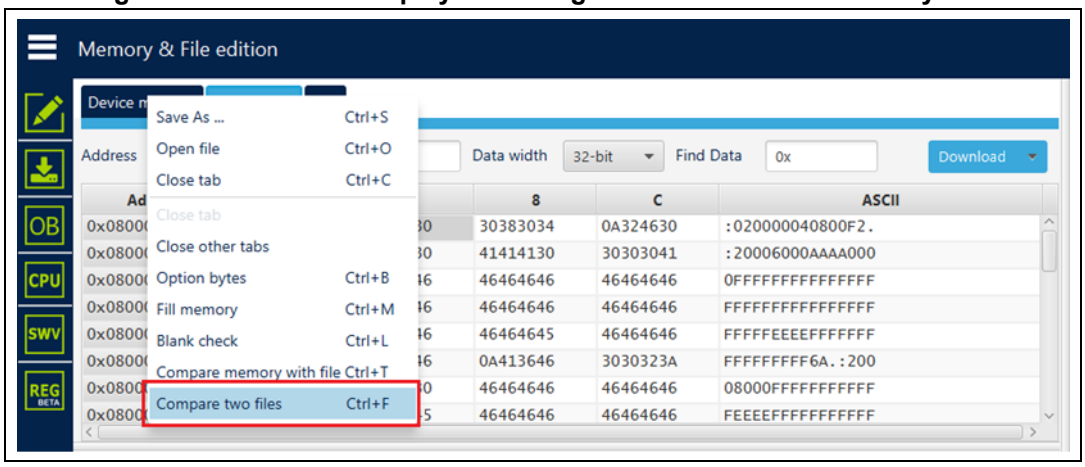


Figure 99. Sub-menu displayed with right click on the cell of grid

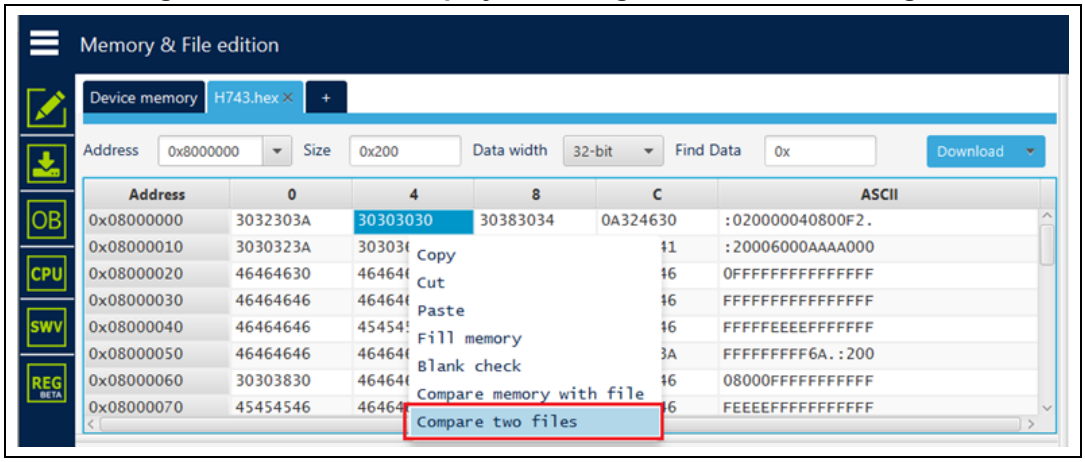


Figure 100. Sub-menu displayed with add tab button

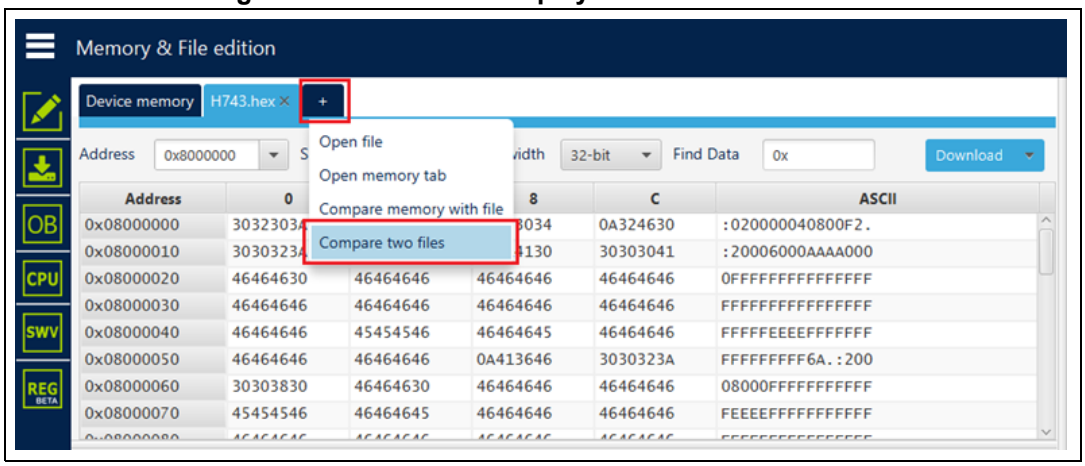


Figure 101. Sub-menu displayed with right click on the opened file tab

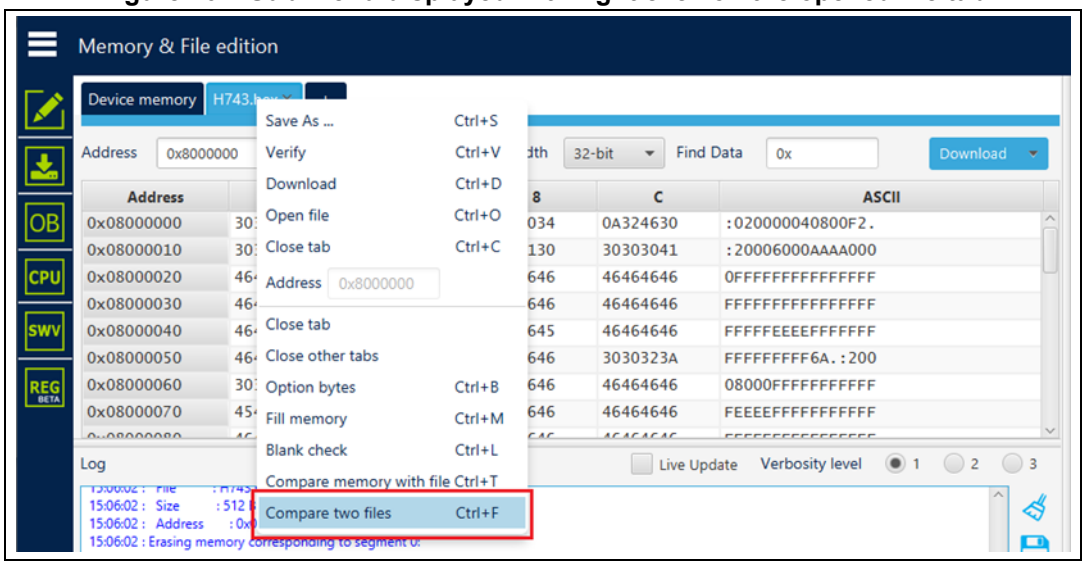
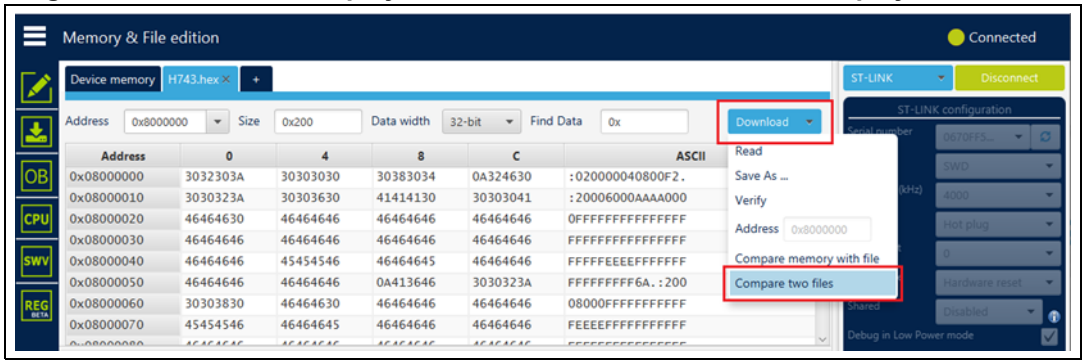


Figure 102. Sub-menu displayed from “Download” combo-box displayed in file tab



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 of the same type and different sizes

Figure 103. Data width: 32 bits

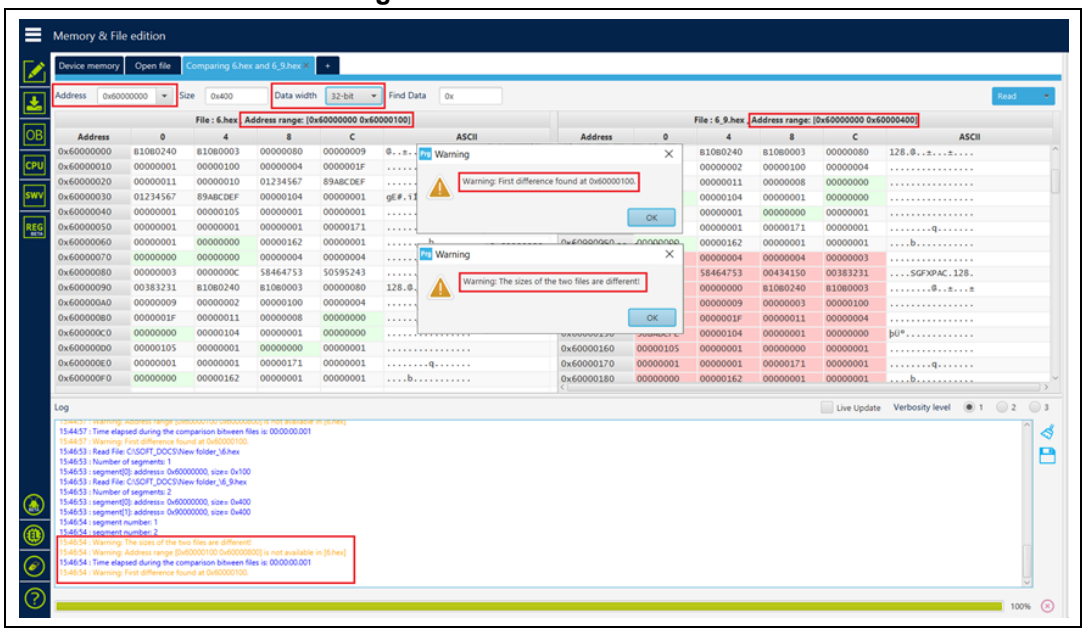


Figure 104. Data width: 16 bits

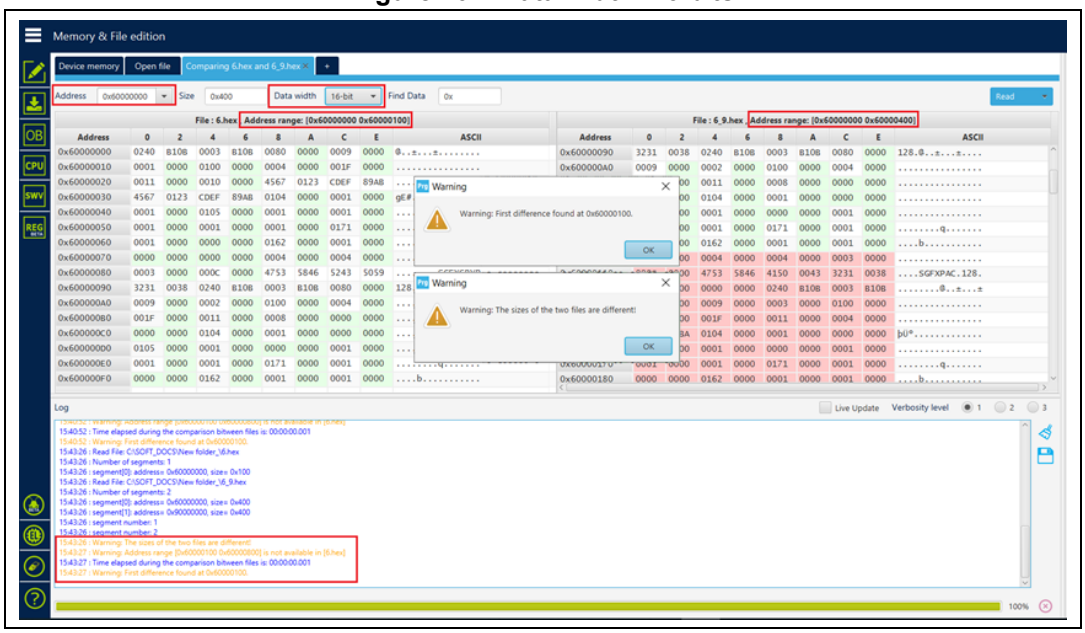
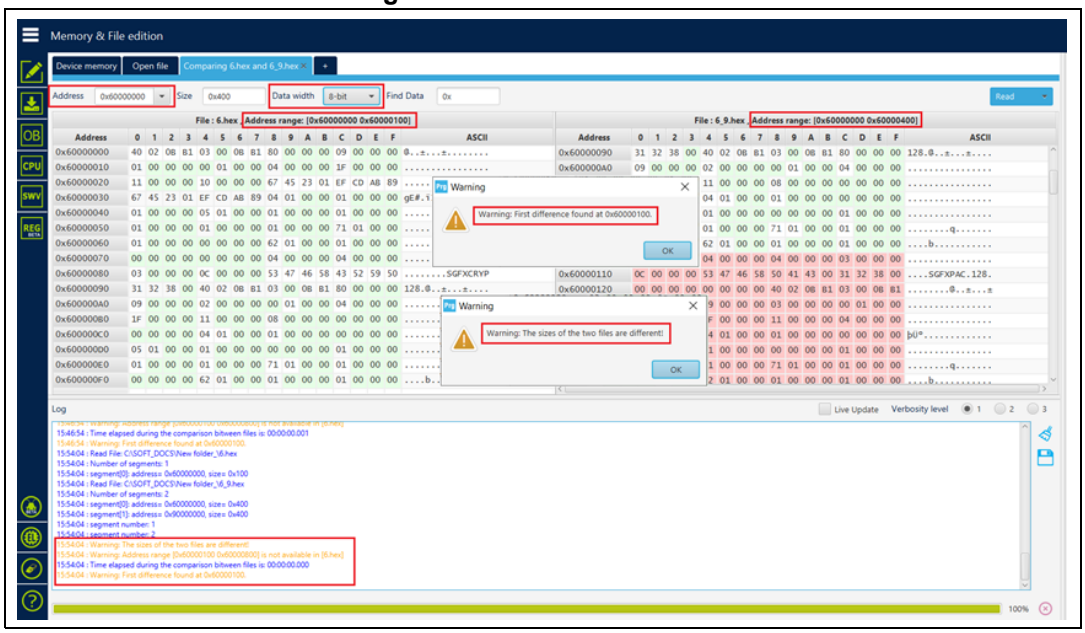
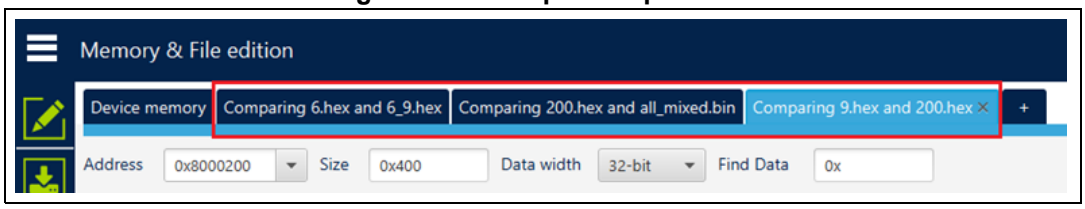


Figure 105. Data width: 8 bits



Note: The user can make multiple comparisons between files.

Figure 106. Multiple comparisons



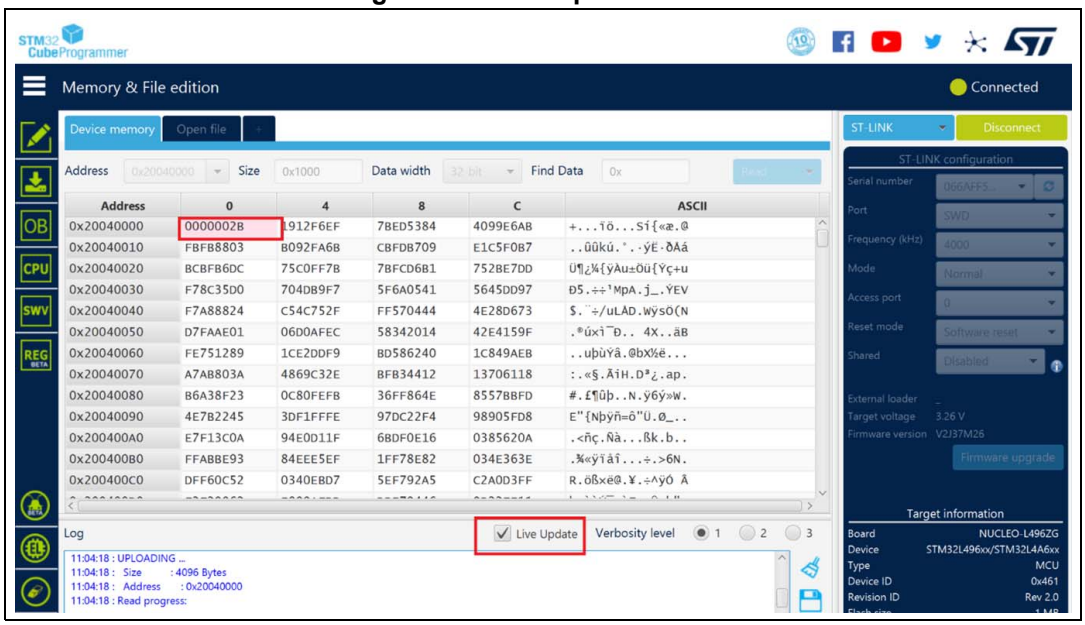
2.21 LiveUpdate feature

-liveUpdate checkbox

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

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

Figure 107. Live update of data

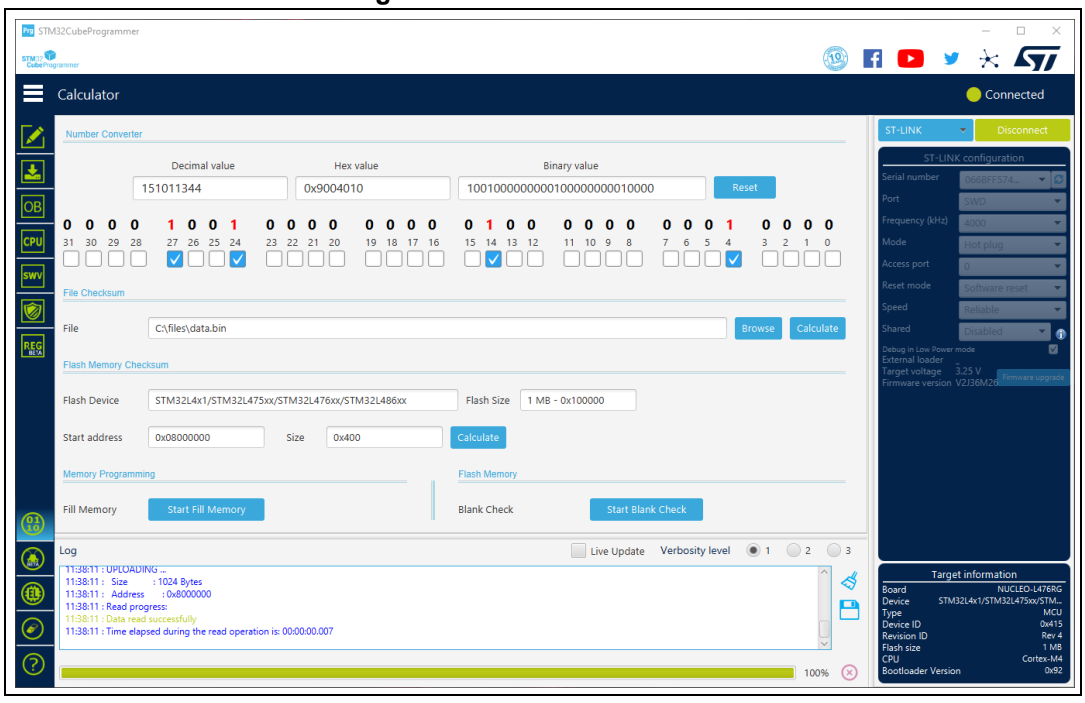


2.22 Calculator

Description: The Calculator window, created for general purposes, is always available, even if the device is not connected. The user interface has three main goals:

1. Number converter
   - Composed by several graphical components, to facilitate the number conversion between decimal, hexadecimal, and binary formats.
   - Use the 32 check boxes, representing a word of 32 bits, to activate or deactivate the relevant bit(s).
   - Use the “Reset” button to reinitialize the number to 0.
   - Any time a bit is changed, the number value is updated.
2. Checksum calculation
   - To calculate the checksum value, based on addition algorithm applicable on the file content or on the flash memory.
   - File Checksum: choose your binary file and click on “Calculate” button to display the corresponding result on the log panel.
   - Flash Memory Checksum: calculates the checksum value of a region (based on start address and size of the desired region) once the device is connected.
   - To calculate the full flash memory checksum retrieve the memory size, displayed in the “Flash size” field.
3. Memory programming
   - To expose the generic memory edition options
   - Fill memory: see *Section 2.16*
   - Blank check: see *Section 2.18*

Figure 108. Calculator window



Number converter and *File checksum* can be used even if there is no device connected. *Flash Memory Checksum*, *Fill Memory*, and *Blank Check* are applicable only if a device is already connected.

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 109*.

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

Figure 109. STM32CubeProgrammer: available commands

```
Usage :
STM32_Programmer_CLI.exe [command_1] [Arguments_1][[command_2] [Arguments_2]...

Generic commands:

 -?, -h, --help          : Show this help
 -version, --version     : Displays the tool's version
 -l,     --list          : List all available communication interfaces
     <uart>              : UART interface
     <usb>               : USB interface
 -q,     --quietMode     : Enable quiet mode. No progress bar displayed
 -log,   --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,    --mergesbsfu    : Add a binary header and a sbsfu segment to an elf fil

     <elf_file_path>     : File path to be modified
     <header_file_path>  : Header file path
     <sbsfu_file_path>   : SBSFU file path
 -c,     --connect       : Establish connection to the device
     <port=<PortName>    : Interface identifier. ex COM1, /dev/ttyS0, usb1,
                           JTAG, SWD...>
    UART port optional parameters:
     [br=<baudrate>]     : Baudrate. ex: 115200, 9600, etc, default 115200
     [P=<parity>]        : Parity bit, value in <NONE,ODD,EVEN>, default EVEN
     [db=<data_bits>]    : Data bit, value in <6, 7, 8> ..., default 8
     [sb=<stop_bits>]    : Stop bit, value in <1, 1.5, 2> ..., default 1
     [fc=<flowControl>]  : Flow control
                           Value in <OFF,Hardware,Software) ..., default OFF
                           Not supported for STM32MP
     [noinit=noinit_bit]: Set No Init bits, value in <0,1) ..., default 0
     [console]           : Enter UART console mode
    JTAG/SWD debug port optional parameters:
     [freq=<frequency>] : Frequency in KHz. Default frequencies:
                           4000 SWD 9000 JTAG with STLINKv2
                           24000 SWD 21333 with STLINKv3
     [index=<index>]     : Index of the debug probe. default index 0
     [sn=<serialNumber>]: Serial Number of the debug probe
     [ap=<accessPort>]   : Access Port index to connect to. default ap 0
     [mode=<mode>]       : Connection mode. Value in <UR/HOTPLUG/NORMAL>
                           default mode: NORMAL
     [reset=<mode>]      : Reset modes: SWrst/HWrst/Crst. Default mode: SWreset
    SPI port optional parameters:
     [br=<baudrate>]     : Baudrate.
     [cpha=<cpha_val>]   : 1Edge or 2Edge. default 1Edge
     [cpol=<cpol_val>]   : low or high
     [crc=<crc_val>]     : enable or disable <0/1>.
     [crcpol=<crc_pol>]  : crc polynom value.
     [datasize=<size>]   : 8bit/16bit
     [direction=<val>]   : Direction: 2LFullDuplex/2LRxOnly/1LRx/1LTx
     [firstbit=<val>]    : First Bit: MSB/LSB
     [frameformat=<val>]: Frame Format: Motorola/TI
     [mode=<val>]        : Mode: master/slave
     [nss=<val>]         : NSS: soft/hard
     [nsspulse=<val>]    : NSS pulse: Pulse/NoPulse
     [delay=<val>]       : Delay: Delay/NoDelay, delay of fiew microseconds
     [noinit=noinit_bit]: Set No Init bits, value in <0,1) ..., default 0
    CAN port optional parameters:
     [br=<rbaudrate>]    : Baudrate : 125, 250, 500, 1000 Kbps, default 125
     [mode=<canmode>]    : CAN Mode : NORMAL, LOOPBACK..., default NORMAL
     [ide=<type>]        : CAN Type : STANDARD or EXTENDED, default STANDARD
     [rt=<format>]       : Frame Format: DATA or REMOTE, default DATA
     [fifo=<afifo>]      : Msg Receive : 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>)]     : Filter Activation : ENABLE or DISABLE, default ENABLE
     [fbn=<fbanknb>)]    : Filter Bank Number : 0 to 13, default 0
     [noinit=noinit_bit]: Set No Init bits, value in <0,1) ..., default 0
    I2C port optional parameters:
     [add=<ownadd>)]     : Slave address : address in hex format
     [br=<sbaudrate>)]   : Baudrate : 100 or 400 Kbps, default 400
     [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
     [noinit=noinit_bit]: Set No Init bits, value in <0,1) ..., default 0
 -e,     --erase         : Erase memory pages/sectors devices:
                           Not supported for STM32MP
     [all]               : Erase all sectors
     [<sectorsCodes)>]   : Erase the specified sectors identified by sectors
                           codes. ex: 0, 1, 2 to erase sectors 0, 1 and 2
     [<[start end])>]    : Erase the specified sectors starting from
                           start code to end code, ex: -e [5 10]
 -w,     --write
 -d,     --download      : Download the content of a file into device memory
     <file_path>         : File path name to be downloaded: <bin, hex, srec,
                           elf, stm32 or tsv file>
     [<address>)]        : Start address of download
 -w32                    : Write a 32-bits data into device memory
     <address>           : Start address of download
     <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                    : Read a 32-bit data from device memory
     <address>           : Read start address
     <size>              : Size of data
 -rst                    : Reset system
 -hardRst                : Hardware reset
                           Available only with JTAG/SWD debug port
 -halt                   : Halt core
 -step                   : Step core
                           Available only with JTAG/SWD debug port
 -score                  : Get core status
                           Available only with JTAG/SWD debug port
 -coreReg                : Read/Write core registers
     [<core_register>)]    R0/../R15/PC/LR/PSP/MSP/XPSR/APSR/IPSR/EPSR/
                           PRIMASK/BASEPRI/FAULTMASK/CONTROL
     [core_reg=<value>)]   value in case of write opration
                           Note: multiple registers can be handled at once
                           Available only with JTAG/SWD debug port

 -r,     --read
 -u,     --upload        : Upload the device memory content to a .bin file
     <address>           : Start address of read and upload
     <size>              : Size of memory content to be read
     <file_path>         : Binary file path

 -el,    --extload       : Select a custom external memory-loader
     <file_path>         : External memory-loader file path
 -s,     --start
 -g,     --go            : Run the code at the specified address.
     [<address>)]        : 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
                           's OptionBytes by displaying or modifying them.
     [displ]             : This option allows the user to display the whole set
                           of Option Bytes.
     [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 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.
`POWERDOWN`	Allows to put the target in debug mode, even if the application has not started since the target power up. The hardware reset signal must be connected between ST-Link and the target. This feature might be not fully effective on some boards (MB1360, MB1319, MB1361, MB1355) with STMPS2141 power switch.

[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).
[getAuthID]	Get device identification (only for STM32U5 series): is a 32-bit device specific quantity that can be read though the JTAG port. This 32-bit information is used to derive the expected OEM password keys to unlock this specific device. This command is not applicable when RDP level = 0 (MCU constraint).

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 (for example 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

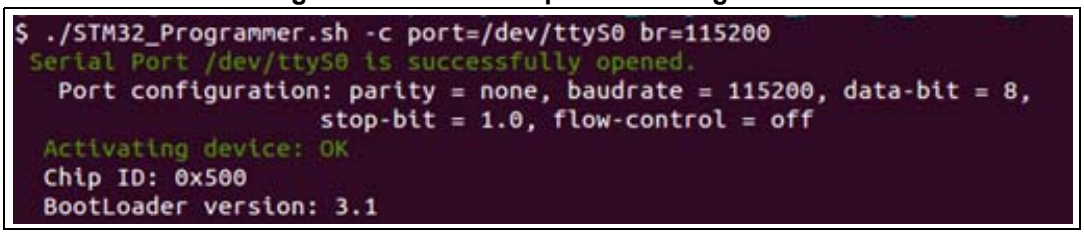
[fbn=<fbanknb>] Filter bank number: 0 to 13, default 0

- Using UART

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

The result of this example is shown in *Figure 110*.

Figure 110. Connect operation using RS232

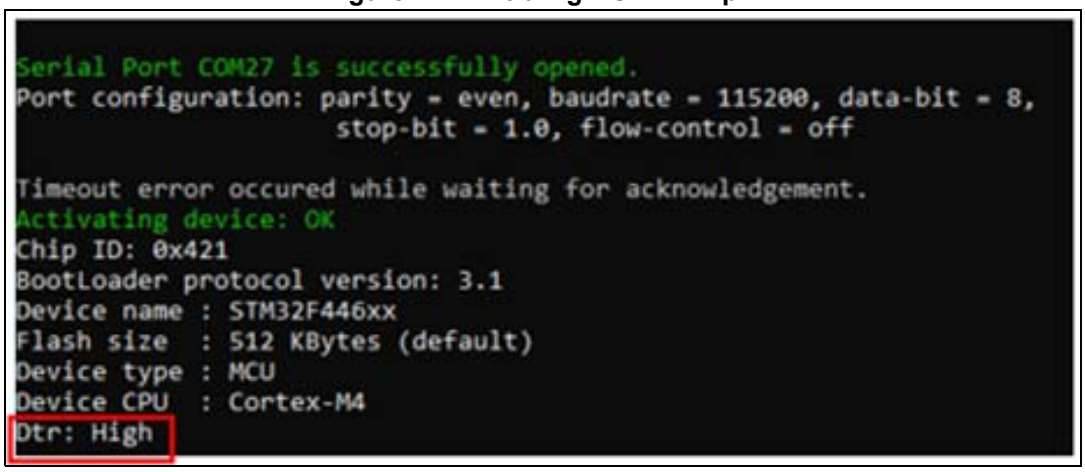


STM32CubeProgrammer provides the possibility to configure RTS and DTR pins:

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

Example: STM32_Programmer_CLI.exe -c port=COM27 dtr=high (see *Figure 111*).

Figure 111. Enabling COM DTR pin

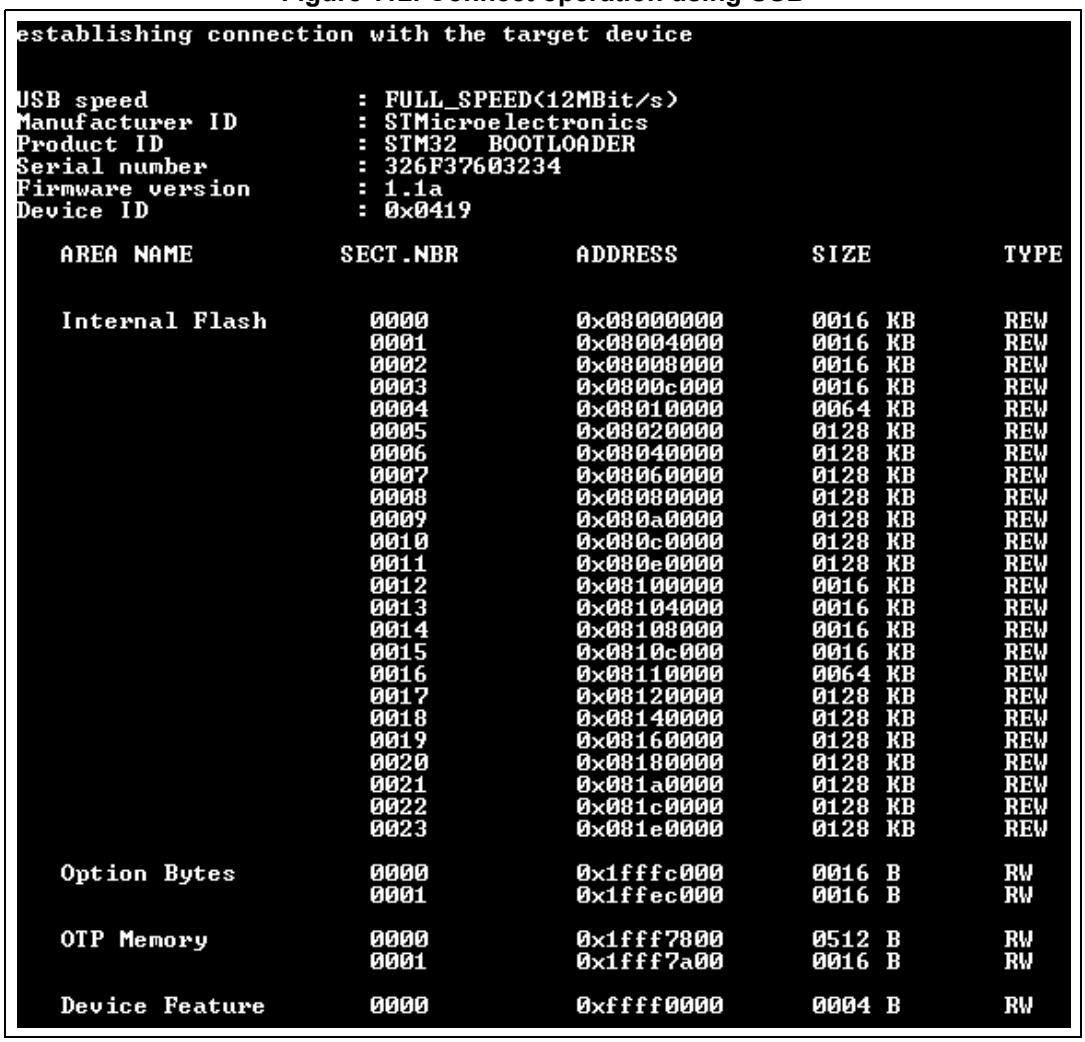


Example using USB

./STM32_Programmer.sh -c port=usb1

The result of this example is shown in *Figure 112*.

Figure 112. Connect operation using USB



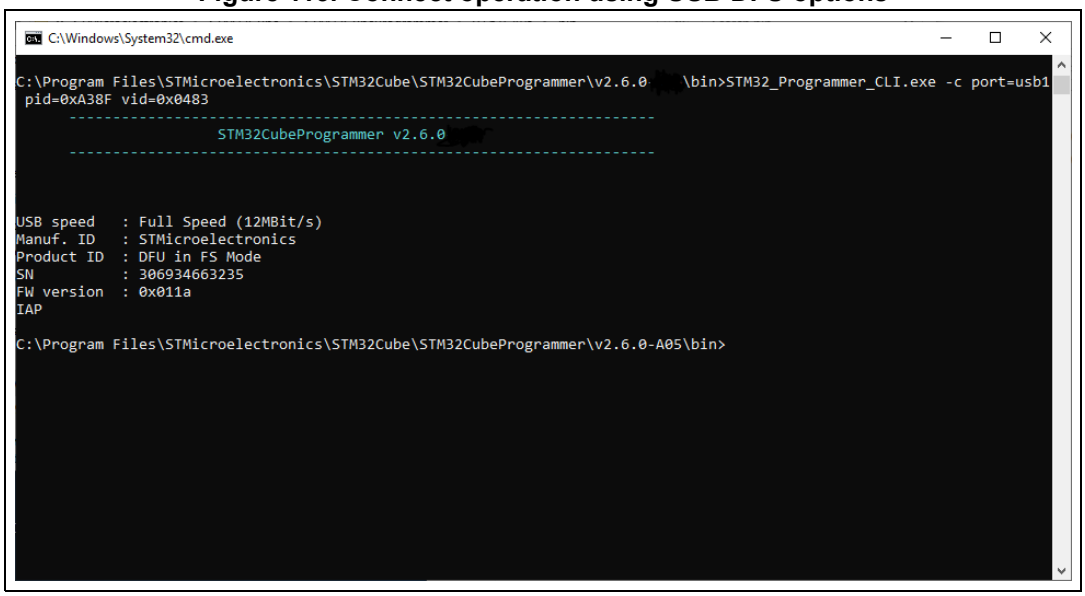
Note: When using a USB interface, all the configuration parameters (for example baud rate, parity, data-bits, frequency, index) are ignored. To connect using a UART interface the port configuration (baudrate, parity, data-bits, stopbits and flow-control) must have a valid combination, depending on the 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 113*.

Figure 113. Connect operation using USB DFU options



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

Example using JTAG/SWD debug port

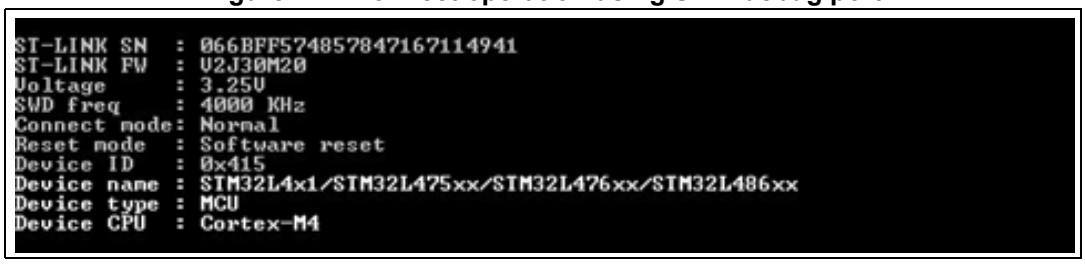
To connect using port connection mode with ST-LINK probe it is necessary to mention the port name with at least the connect command (for example: `-c port=JTAG`).

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 114. Connect operation using SWD debug port



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.

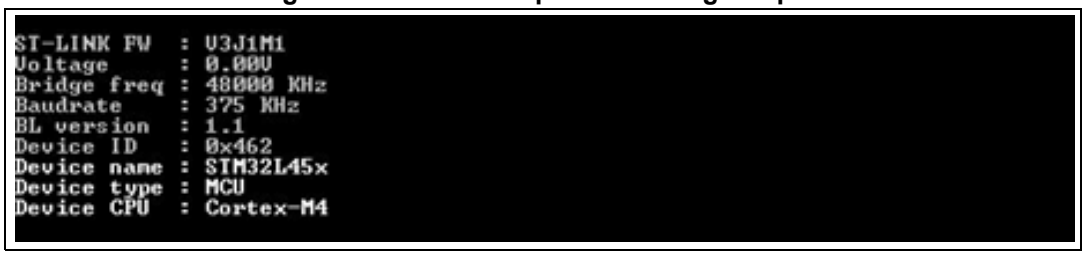
Note: The ST-LINK probe firmware version is shown when connecting to the device. Make sure that you have the latest version of ST-LINK firmware V2J28M17 (STSW-LINK007), available on www.st.com.

Example using SPI

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

The result of this example is shown in *Figure 115*.

Figure 115. Connect operation using SPI port



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 116*.

Figure 116. Connect operation using CAN port

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 upon 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 117*.

Figure 117. Connect operation using I2C port

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 (for example `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, for example `-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 the memory of the device. The download operation is preceded by the erase operation. A write address is 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

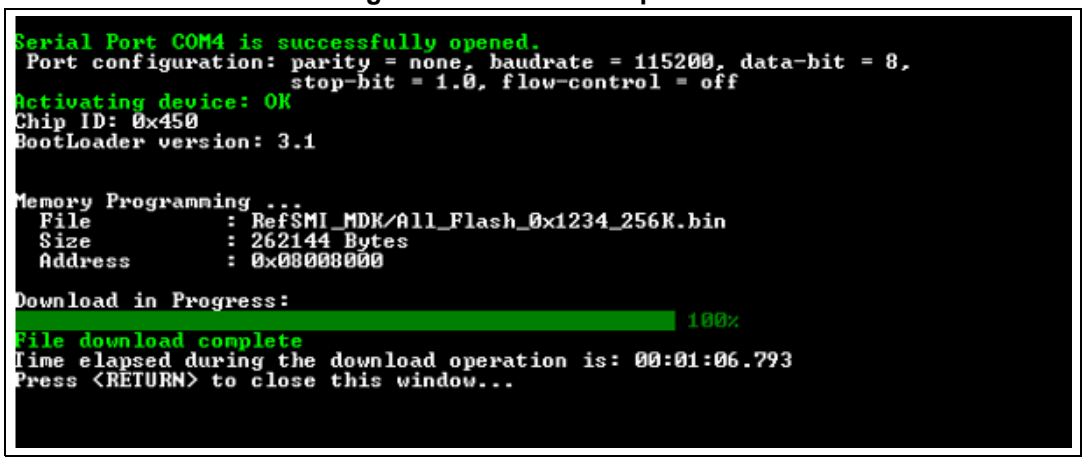
Example

```
-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 118*.

Figure 118. Download operation



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.

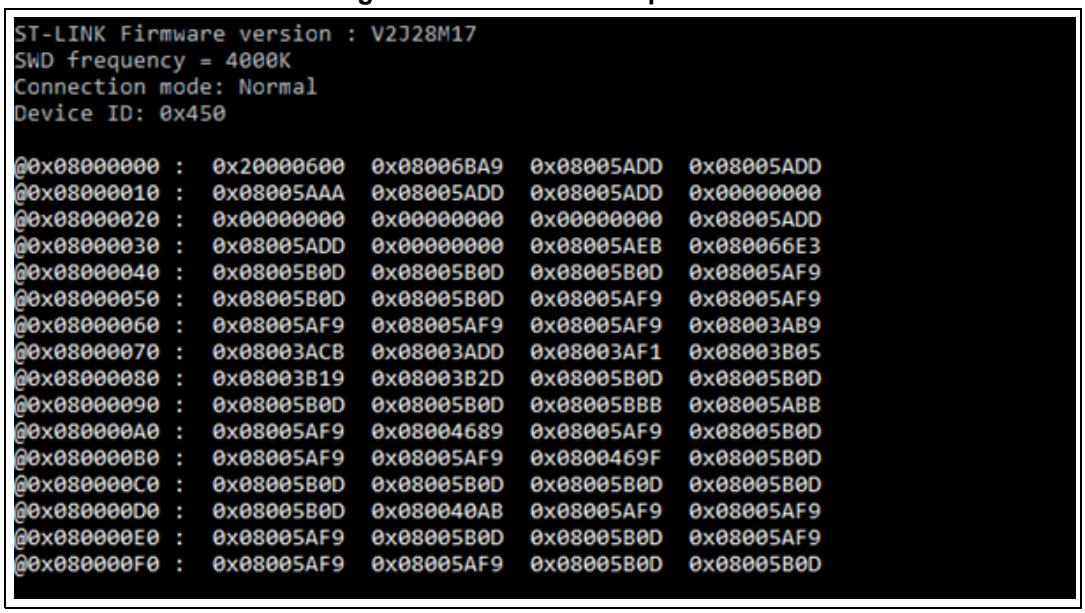
Syntax: **`-r32 <start_address> <size>`**

`<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 119. Read 32-bit operation



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: **`-hardRs`**t

-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 UART, DFU and STLink interfaces.

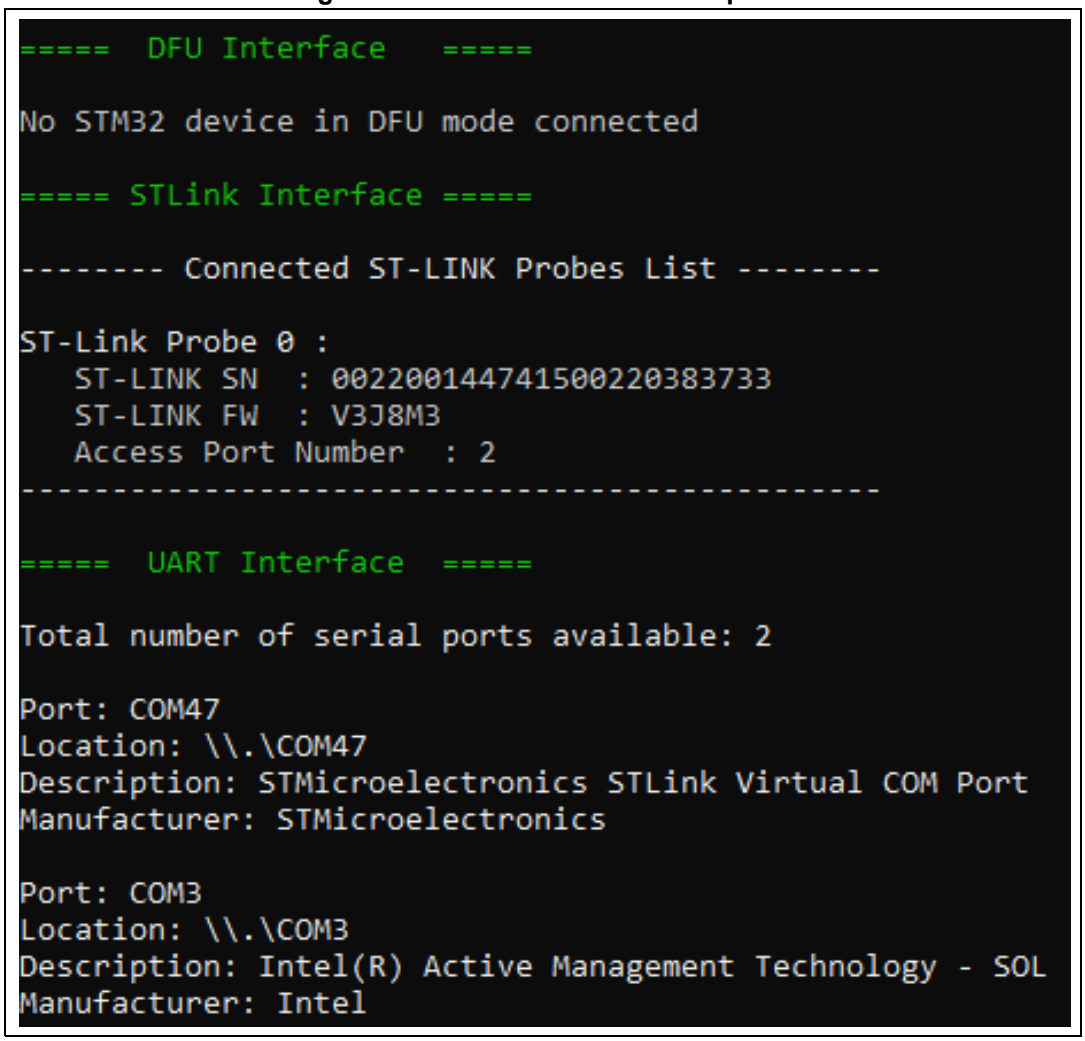
Syntax: **`-l, --list`**

Example

`./STM32_Programmer.sh --list`

The result of this example is shown in *Figure 120*.

Figure 120. List of available serial ports



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 121*.

Figure 121. 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 122*.

Figure 122. 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 123*.

Figure 123. 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 SFIx 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 SFIx

The external loader for SFIx 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>**

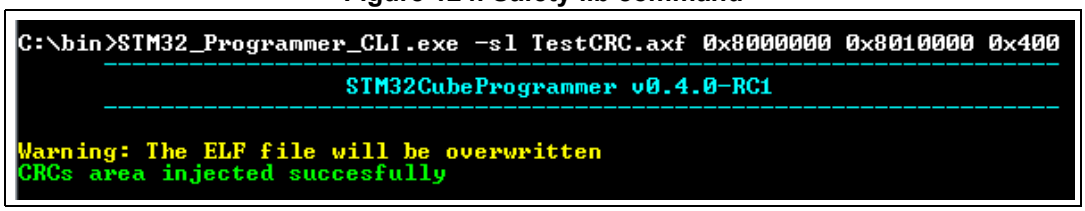
<file_path>	The file path (bin, elf, hex or Srec)
<start_address>	Flash memory start address
<end_address>	Flash memory end address
<slice_size>	Size of data per CRC value

Example

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

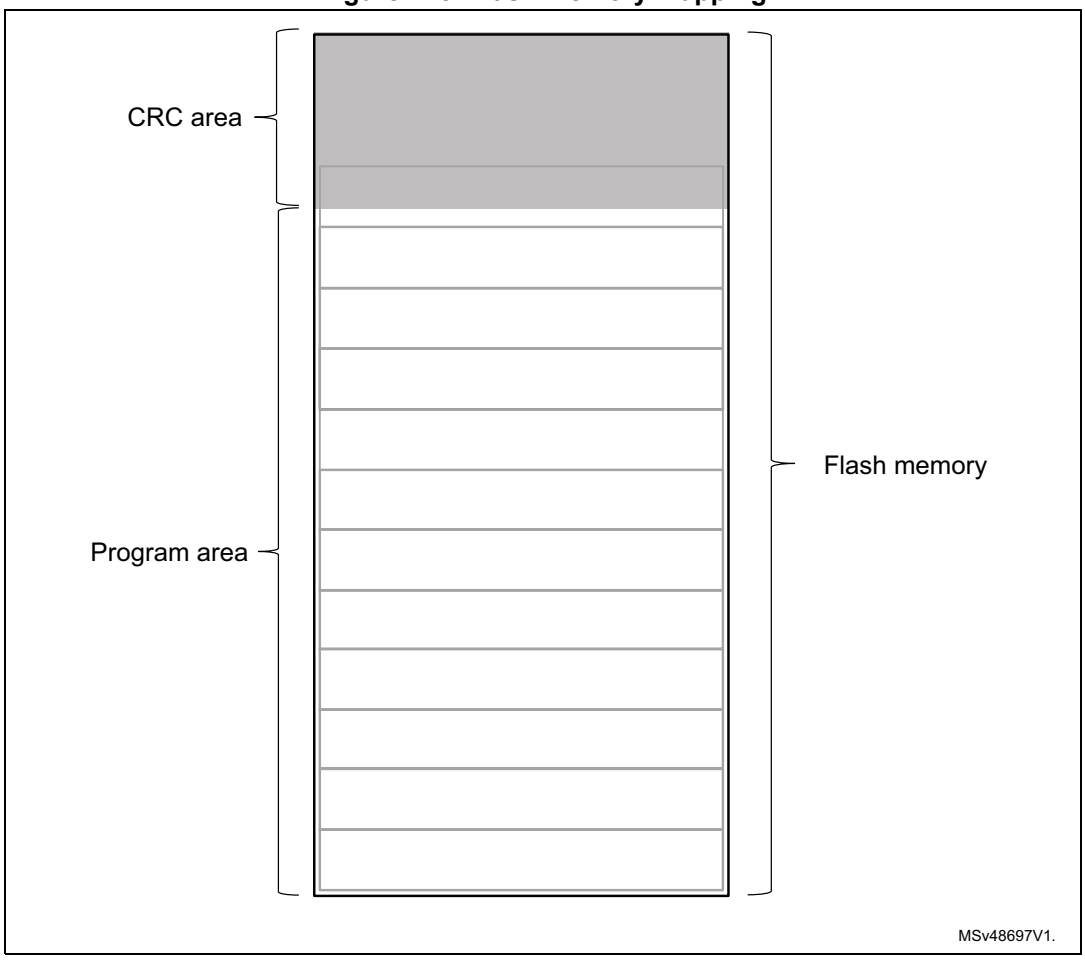
The result is shown in *Figure 124*.

Figure 124. Safety lib command



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 125*.

Figure 125. Flash memory mapping



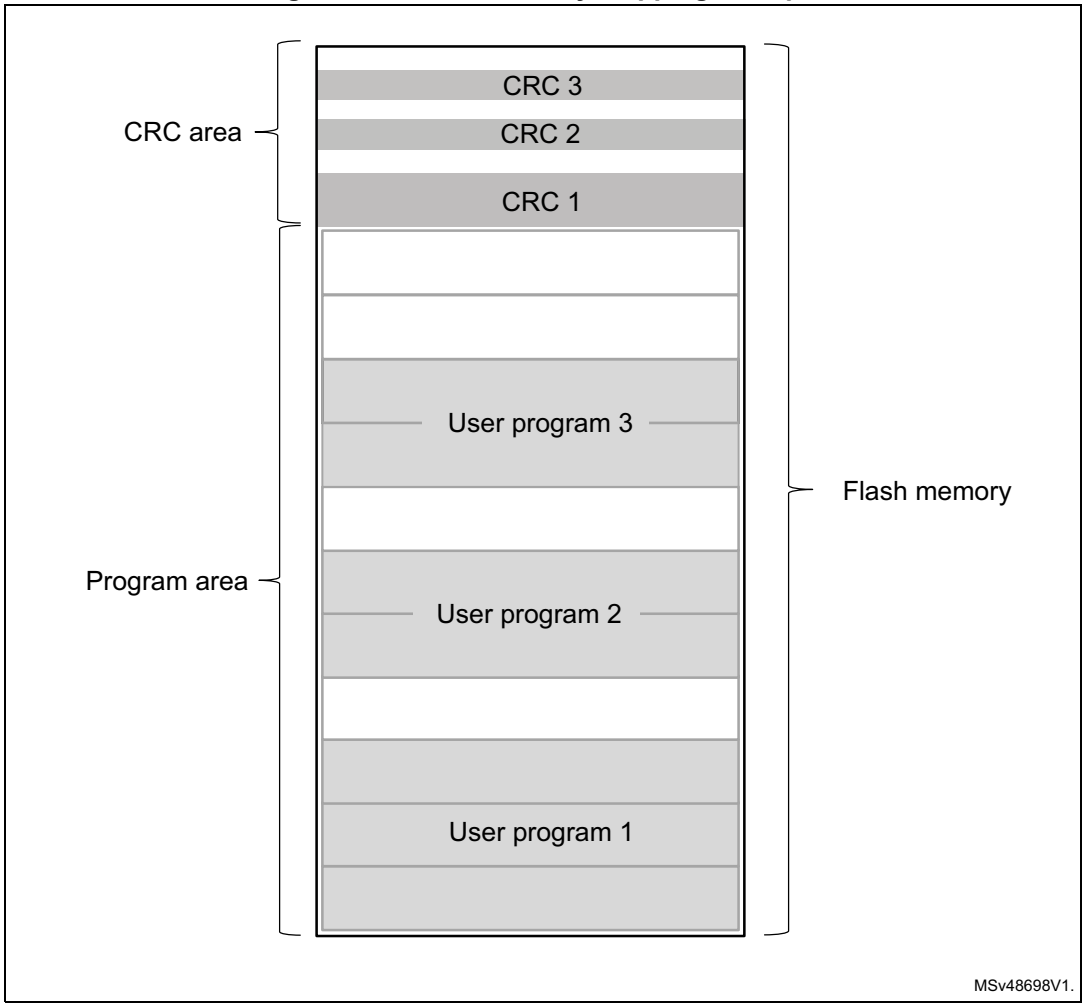
The address and size of the CRCs area are determined as follows:

CRCs_Area_Size = Flash_Size / Slice_Size * 4 bytes

CRCs_Start_Address = Flash_End_Address - 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 126*.

Figure 126. 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 AN5054.

-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]`

`[<protocol=Ptype>]`	Protocol type to be used: static/live (only static protocol is supported so far), default: static.
`<file_path>`	Path of sfi file to be programmed.
`[hsm=0|1]`	Sets user option for HSM use value {0 (do not use HSM), 1 (use HSM)}, default: hsm = 0.
`<lic_path|slot=slotID>`	Path to the SFI license file (if hsm = 0) or reader slot ID if HSM is used (hsm = 1).
`[<licMod_path>|slot=slotID]`	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 the 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 SFIx specific commands

Secure firmware install (SFIx) 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 SFIx secure programming can be carried out only with JTAG/SWD interface. For more details refer to AN5054.

-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]`**

`[<protocol=Ptype>]`	Protocol type to be used: static/live (only static protocol is supported so far), default: static.
`<file_path>`	Path of sfi file to be programmed.
`[hsm=0|1]`	Sets user option for HSM use value {0 (do not use HSM), 1 (use HSM)}, default: hsm = 0.
`<lic_path|slot=slotID>`	Path to the SFI license file (if hsm = 0) or reader slot ID if HSM is used (hsm = 1).
`[<licMod_path>|slot=slotID]`	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 SFIx use case, as some initializations are already done by RSS, and it is marked with –SFIX 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.

-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: slott = 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 o//perator 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]`

`<file_path>`	New firmware image file path
`<address>`	Start address of download
`[firstinstall=0|1]`	1 for the first installation, otherwise 0 Optional, default value **`firstinstall=0`**
`[startstack=0|1]`	1 to start the stack after the upgrade, otherwise 0 Optional, default value **`startstack=1`**
`[-v]`	Verify if the download operation is completed successfully before starting the upgrade

-startwirelessstack

Description: Start the wireless stack

Syntax: `-startwirelessstack`

-authkeyupdate

Description: Authentication key update

Syntax: `-authkeyupdate <file_path>`

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

-authkeylock

Description: Authentication key lock

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

Syntax: `-authkeylock`

-wusrkey

For more information about the customer key storage, refer to already cited AN5185.

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.

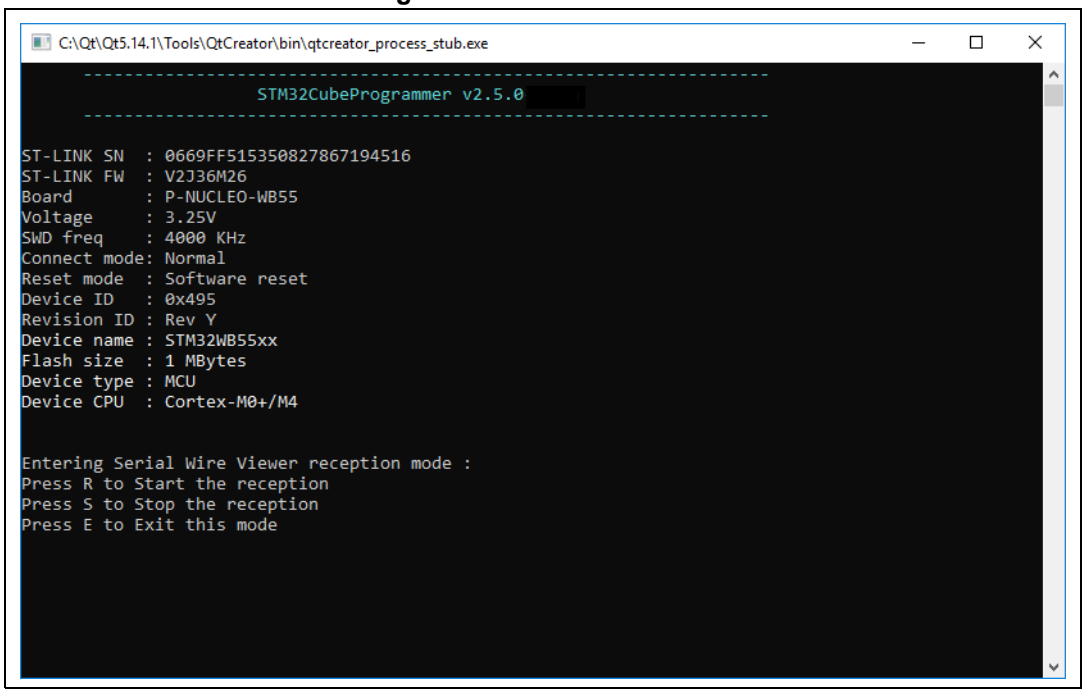
3.2.23 Serial wire viewer (SWV) command

-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 127*) 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 to terminate the reception session.

Figure 127. SWV command



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”.
`[-RA]`	Option that automatically starts SWV data reception.

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 available only through SWD interface.

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

-startswv

Description: This command allows the user to access the serial wire viewer console mode.

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

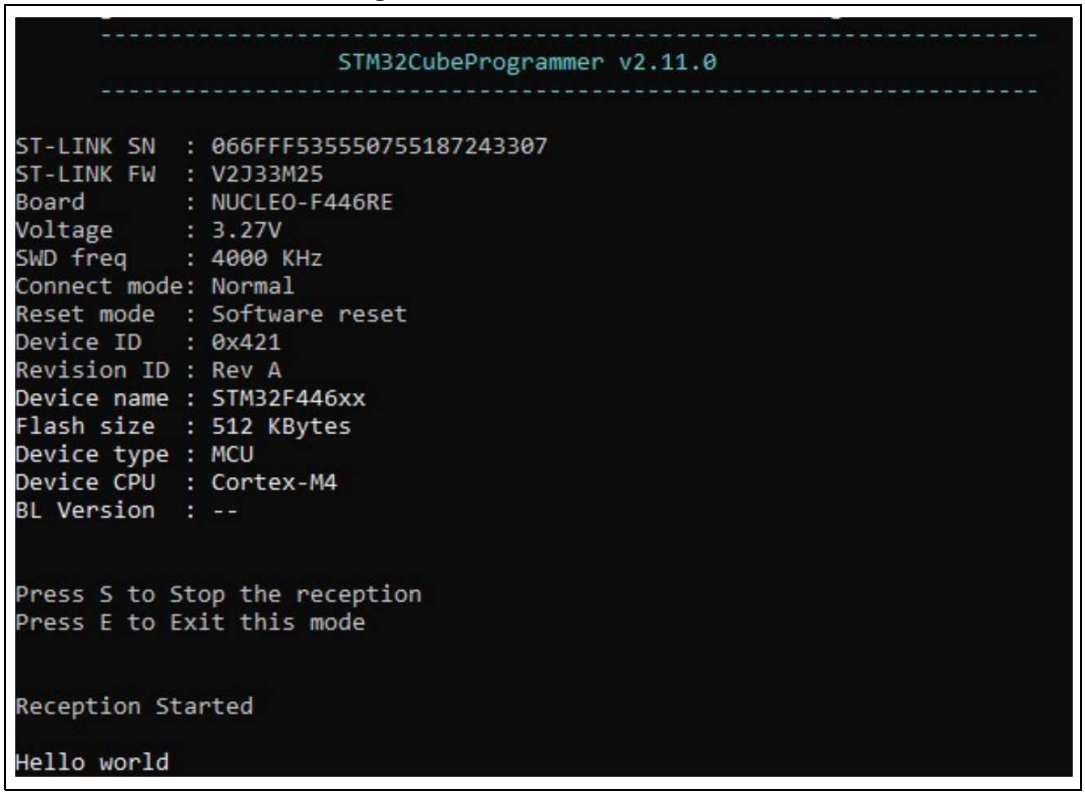
`<freq=<frequency>>`	System clock frequency in MHz.

<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 -startswv freq=32 portnumber=0
C:\example.log
```

Figure 128. startswv command



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.

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

1. **dsecurity**: 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:

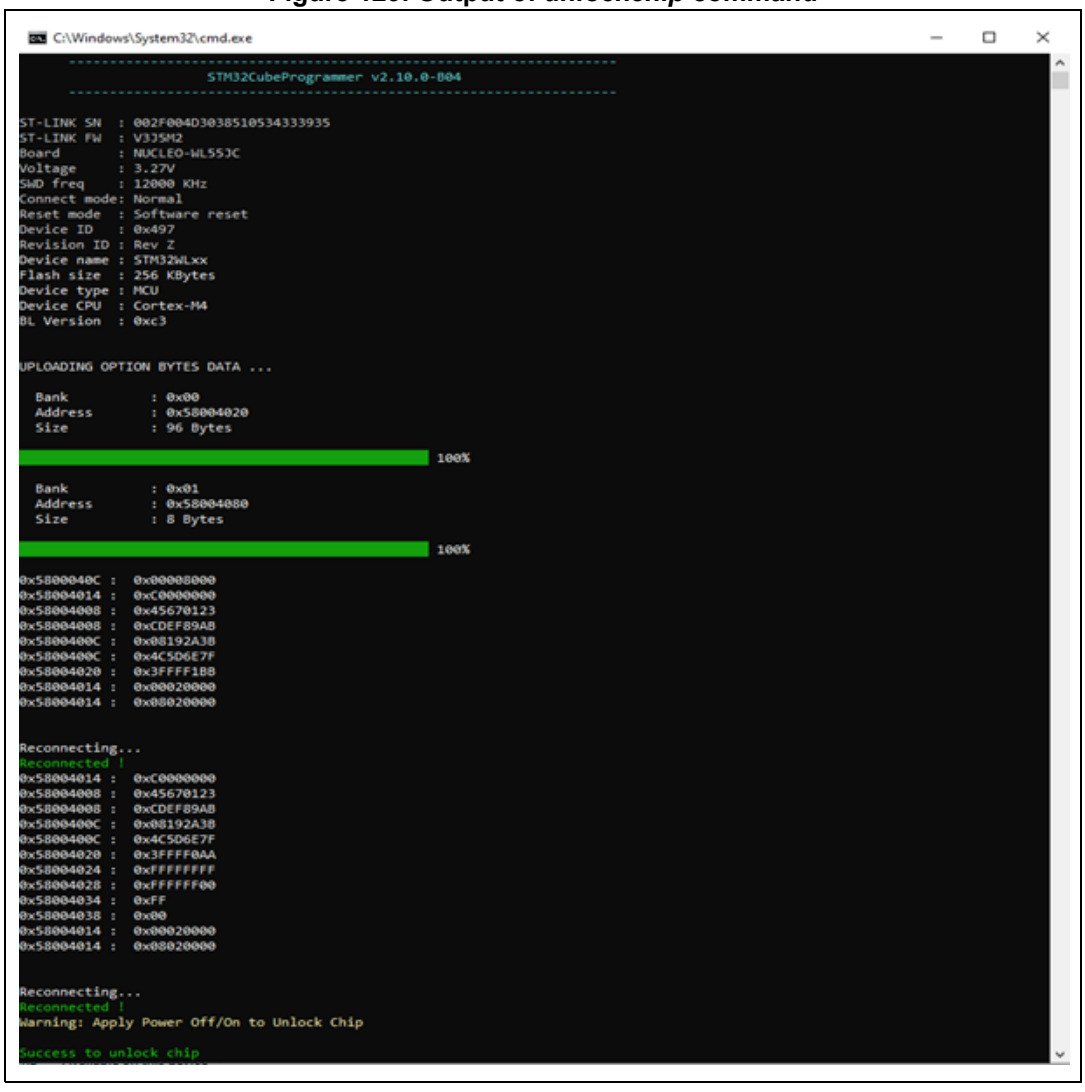
   STM32_Programmer_CLI.exe -c port=swd mode=hotplug -setdefaultob

3. **-ob unlockchip**: this command allows the user to unlock the device if bad option bytes are programmed.

   Example:

   STM32_Programmer_CLI.exe -c port=swd -ob unlockchip

Figure 129. Output of *unlockchip* command



Note: Unlockchip command is available only for STLink connection.

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 130 and *Figure 131* show the results of these command lines.

Figure 130. Disable security

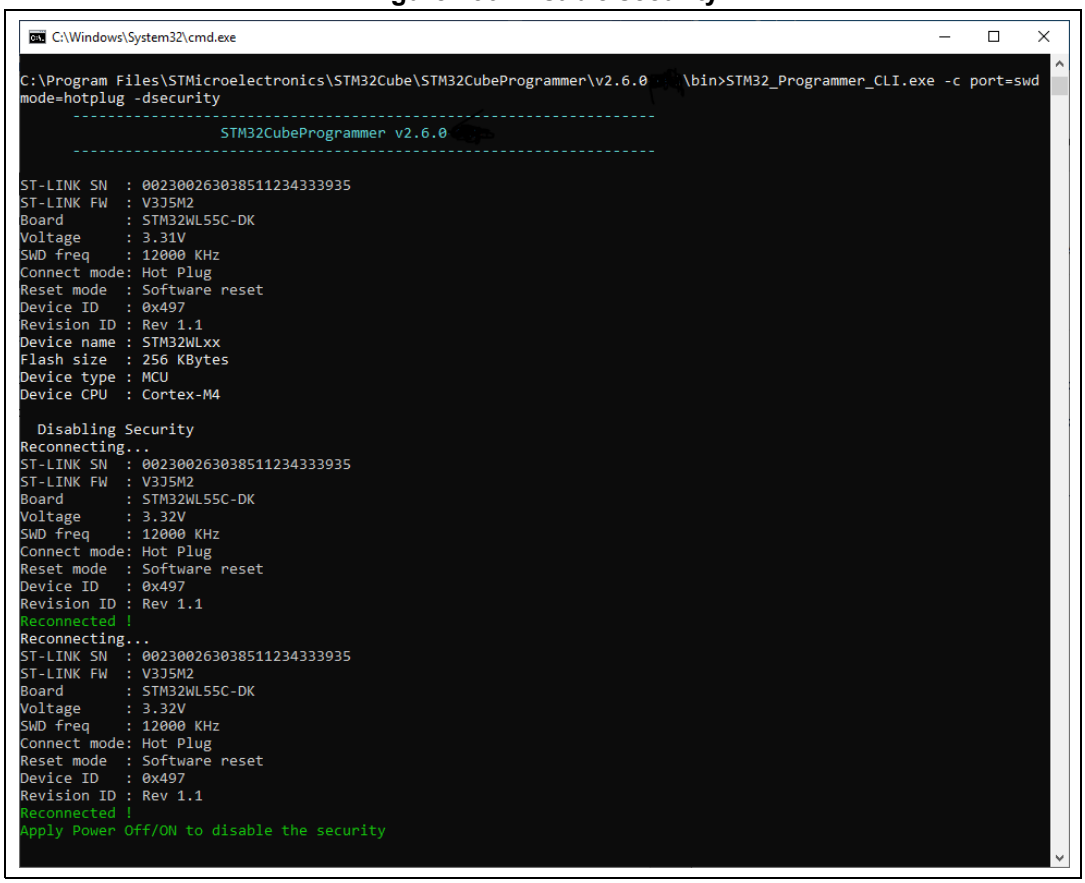
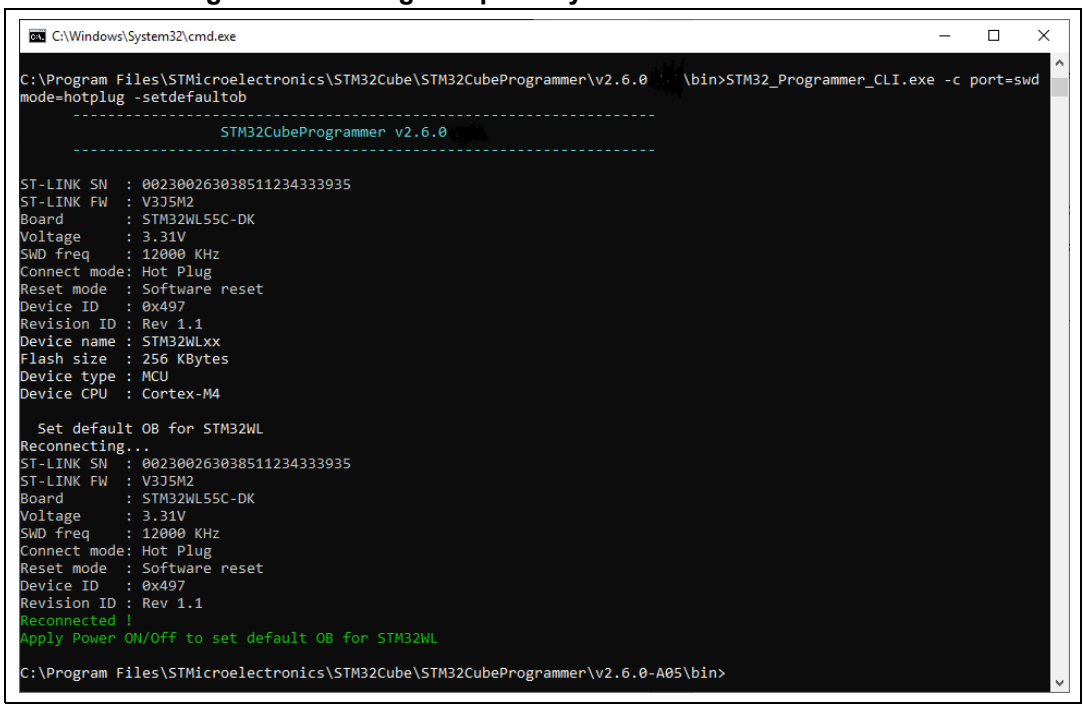


Figure 131. Configure option bytes to their default values



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. "SetRDPLevelCM0.bat" to unlock the board via Cortex M0+
2. "SetRDPLevelCM4.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.

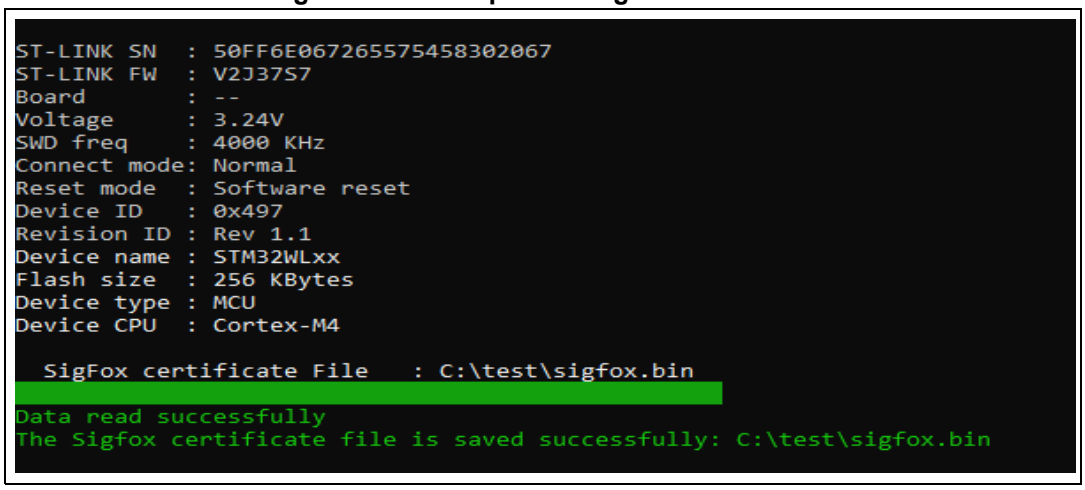
`-ssigfoxc`

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 132. Example of -ssigfoxc command



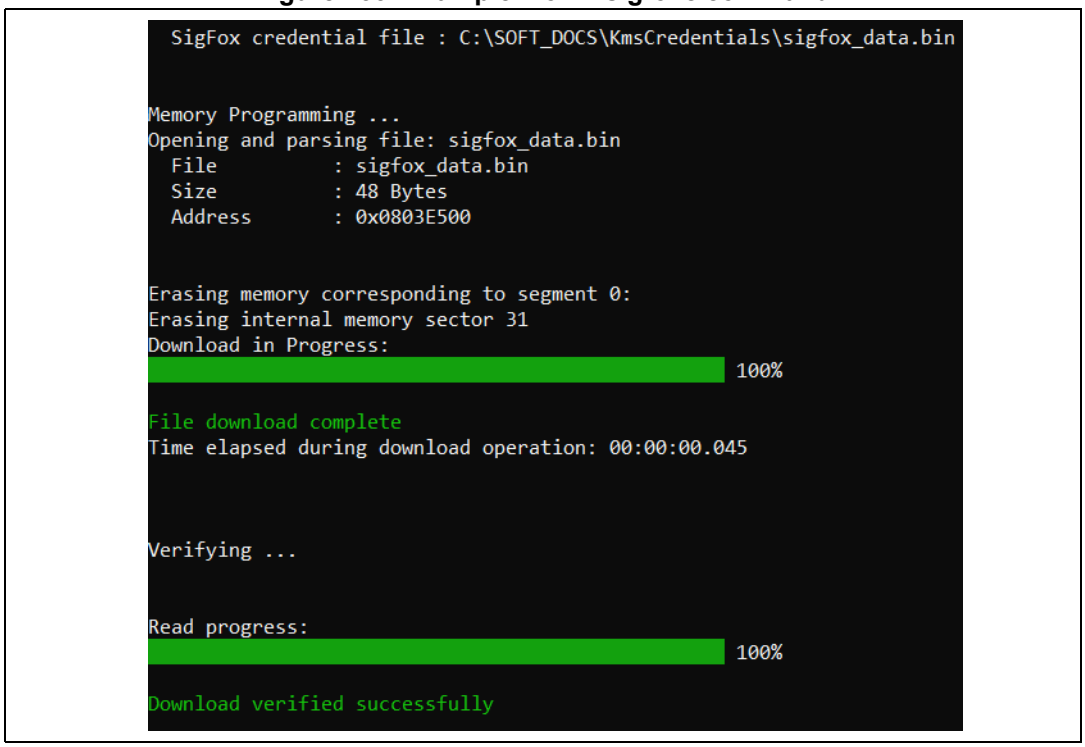
-wsigfoxc

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]

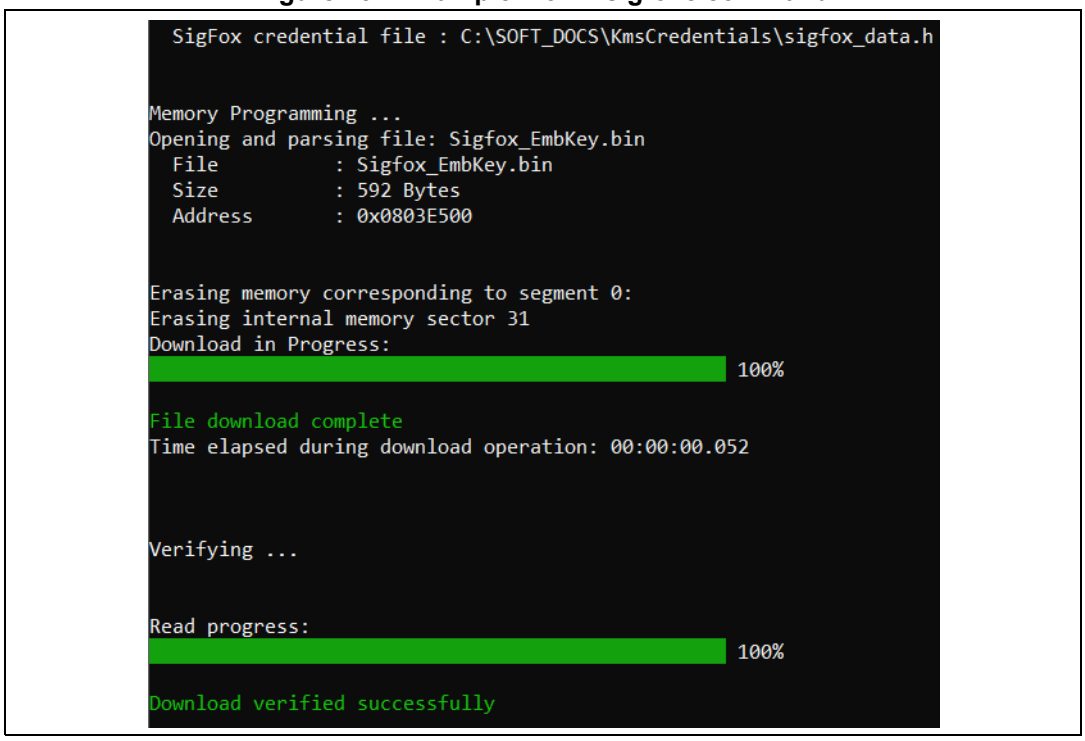
Example 1: STM32_Programmer_CLI.exe -c port=swd -wsigfoxc “/local/user/sigfox_data.bin”0x0803E500

Figure 133. Example 1 of -wsigfoxc command



Example 2: STM32_Programmer_CLI.exe -c port=swd -wsigfoxc “/local/user/sigfox_data.h”

Figure 134. Example 2 of -wsigfoxc command



3.2.26 Register viewer

`-regdump`

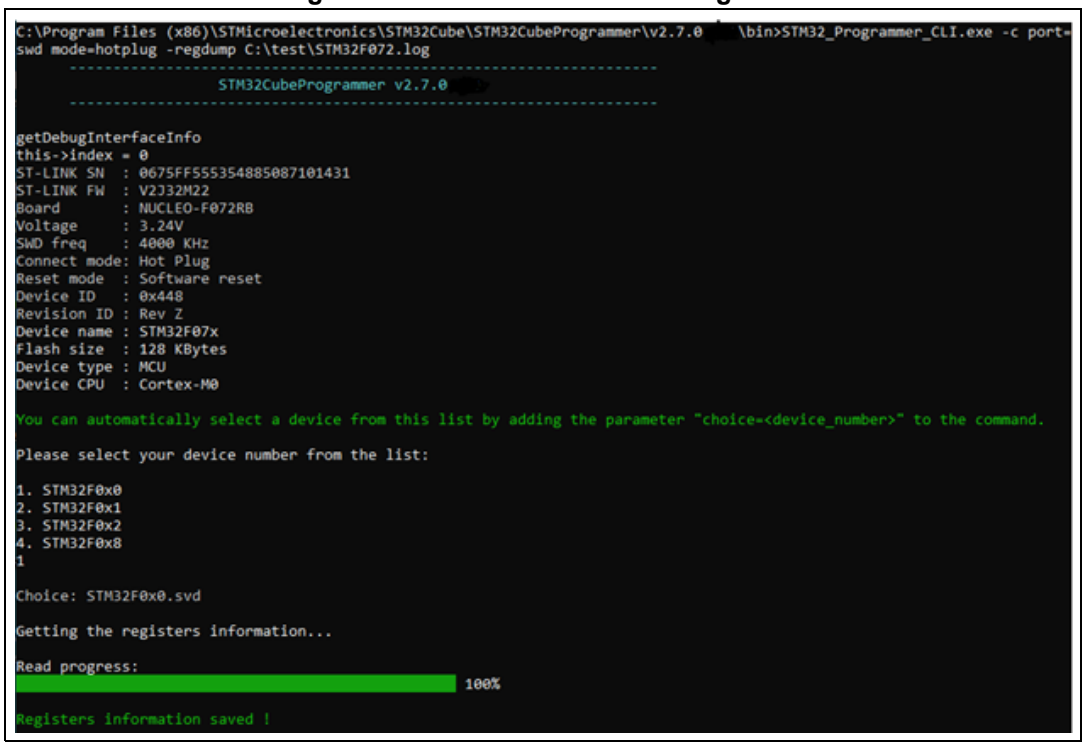
Description: Reads and dumps core and MCU registers

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 -c port=swd -regdump C:\test\STM32F072.log

Figure 135. Read core and MCU registers



3.2.27 Hard fault analyzer

To start the analysis (see *Section 2.14*), 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).

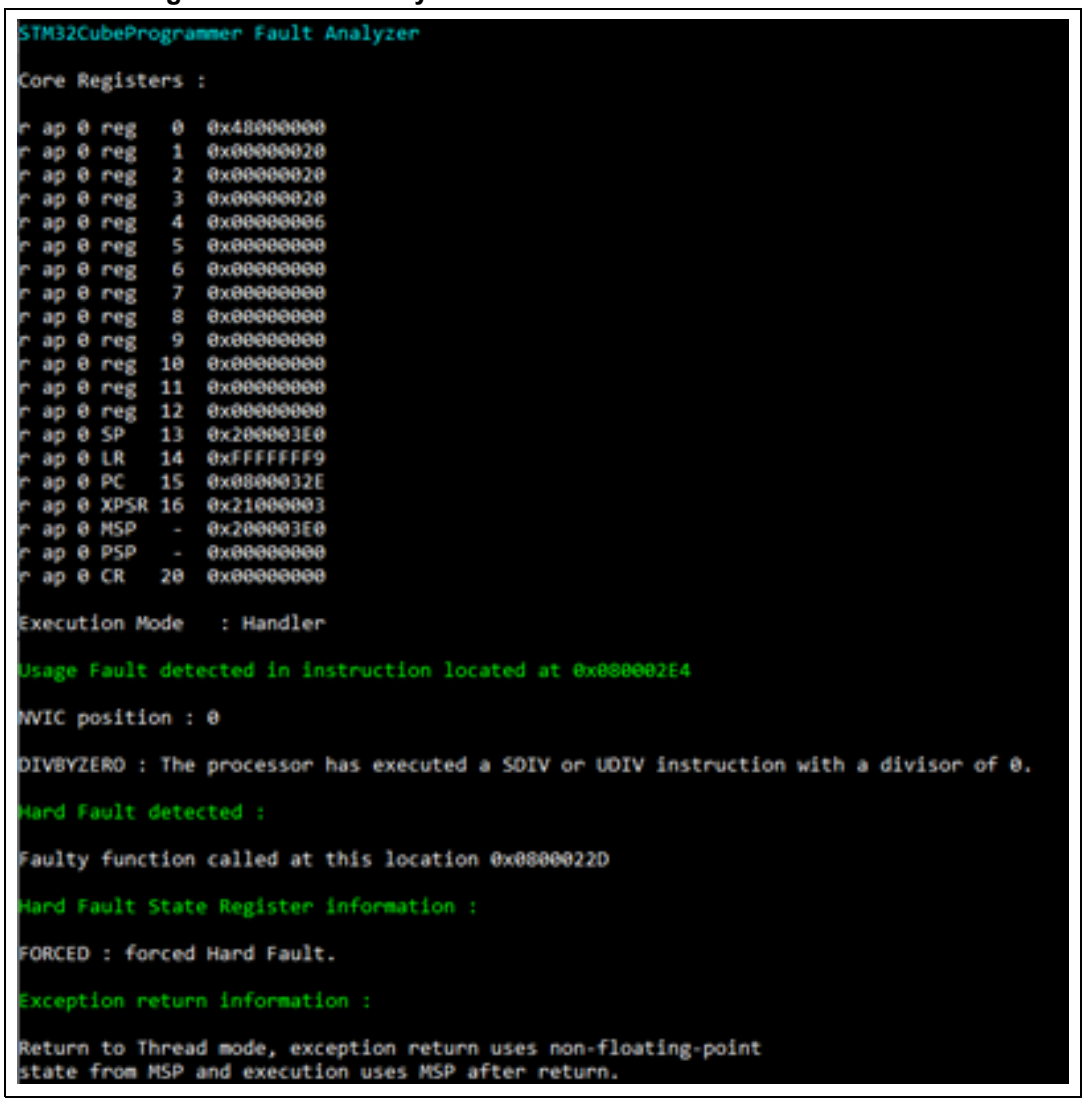
Command: **-c port=swd mode=hotplug -hf**

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 136. Fault analyzer CLI view when Hard Fault is detected



3.2.28 RDP regression with password

Some STM32 products (for example 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 0xFFFFFFFF 0xFFFFFFFF, such as ***`STM32_Programmer_CLI -c port=swd mode=hotplug -lockRDP1 0xFFFFFFFF 0xFFFFFFFF`****.*

3.2.29 GetCertif command

-gc

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

Syntax: **-gc certification.bin**

3.2.30 Write DBG MCU authentication command

-w32dbgmcu

Description: Downloads the specified 32-bit data into the DBGMCU AUTH HOST register to be able to place a message in the mailbox shared between the device and the host.

Syntax: **-w32dbgmcu <32_data_bits>**

Example:

```
-w32dbgmcu 0x12345678
```

Only STM32H5 devices support this command, use the verbosity to check the message, DBG MCU address, and the verification process for write trace.

Note: After the upcoming reset, the device is able to interpret the message.

3.2.31 OBKey provisioning (STM32H573/STM32H563/STM32H562 only)

-sdp

Description: This is a security feature to program OBKey content.

Syntax: `-sdp [OBKey_File_Path.obk]`

[OBKey_File_Path.obk]	Path of OBK file

Example:

```
/STM32_Programmer_CLI.exe -c port=swd mode=hotplug -sdp "C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer_DA_password\bin\DA_
Default_Config\NonCrypto\DA_Config_Certificate.obk"
```

Figure 137. OBKey provisioning example

OBKey file generation is managed by STM32 Trusted Package Creator.

3.2.32 Password provisioning (STM32H503 only)

-pwd

Description: This command provisions the password in OTP, and generates a password.bin file, to be used later for regression.

Syntax: `-pwd value=[Password_Value] path=[Password_Path]`

Password value	Value that will be programed in OTP
Password path	Location where to save “password.bin” file

Example:

```
STM32_Programmer_CLI.exe -c port=swd -pwd value=1mc4l path=C:\my_folder
```

The password size must be between 4 and 16 bytes.

Once the target is successfully provisioned, the “password.bin” file is generated, to be used while performing debug authentication.

Password programming can be executed only once for each target.

3.2.33 Debug authentication commands (STM32H5 series only)

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

[Debugauth=<value>]

Discovery: ***debugauth=2***

Launches discovery, to display information about the target.

Example:

```
/STM32_Programmer_CLI.exe -c port=swd debugauth=2
```

Figure 138. Discovery log

```
Start Debug Authentication Sequence

SDM 0.6.0 Init Sequence
Open SDM Lib
open_comms                    :   434 : open       : Asserting target reset

open_comms                    :   438 : open       : Writing magic number

open_comms                    :   446 : open       : De-asserting target reset

open_comms                    :   492 : open       : Communication with the target established successfully

response_packet_lock
discovery: target ID.....................:0x484
discovery: SoC ID........................:0x0  0x0  0x0  0x0
discovery: SDA version...................:1.0.0
discovery: Vendor ID.....................:STMicroelectronics
discovery: PSA lifecycle.................:ST_LIFECYCLE_PROVISIONING
discovery: PSA auth version..............:1.0
discovery: ST HDPL1 status...............:0xffffffff
discovery: ST HDPL2 status...............:0xffffffff
discovery: ST HDPL3 status...............:0xffffffff
discovery: Token Formats.................:0x200
discovery: Certificate Formats...........:0x201
discovery: cryptosystems.................:Ecdsa-P256 SHA256
discovery: ST provisioning integrity status:0xeaeaeaea

Debug Authentication: Discovery Success
```

Authentication: ***debugauth=1***

Performs debug authentication using credential files.

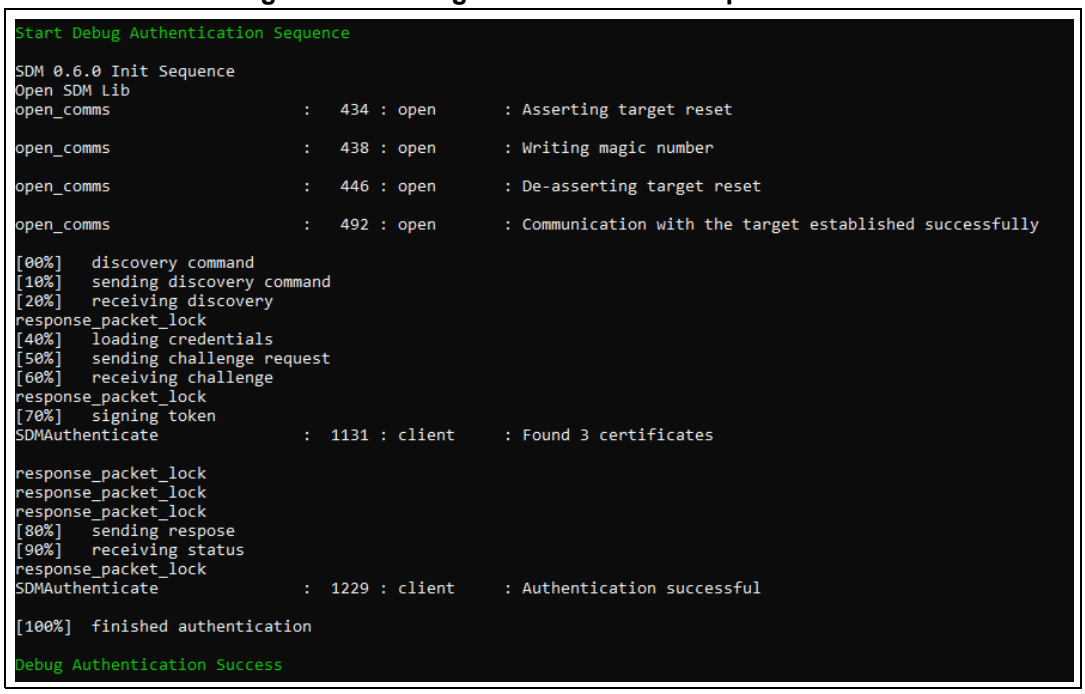
Credential files (passwords, keys, and certificates) are generated by STM32 Trusted Package Creator.

Syntax: **`pwd=[password_path.bin] debugauth=1`** (authentication with password)

Example:

```
STM32_Programmer_CLI.exe -c port=swd pwd=C:\password.bin debugauth=1
```

Figure 139. Debug authentication with password



Syntax: **`per=[Permission] key=[Key_Path.pem] cert=[Certificate_Path.b64] debugauth=1`** (authentication with certificate)

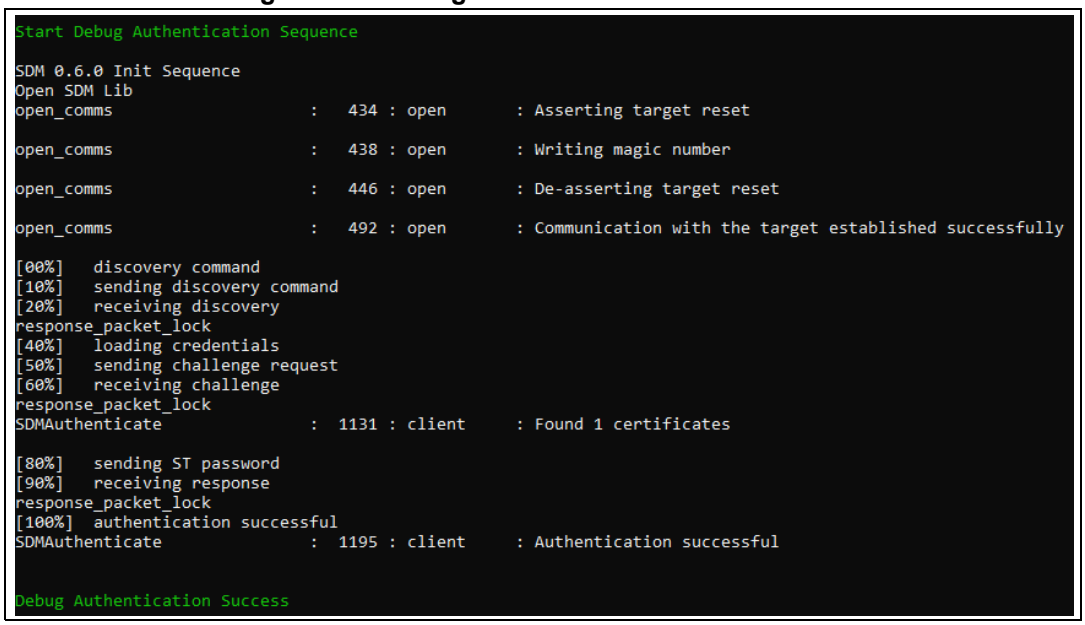
Permission: the possible values to be chosen by the user are:

- “a”: Full regression
- “b”: Partial regression.
- “c”: Debug secure level 3
- “d”: Debug secure level 2
- “e”: Debug secure level 1
- “f”: Debug Non-secure level 3
- “g”: Debug Non-secure level 2
- “h”: Debug Non-secure level 1

Example:

```
STM32_Programmer_CLI.exe -c port=swd per=a key=C:\key_3_leaf.pem
cert=C\cert_leaf_chain.b64 debugauth=1
```

Figure 140. Debug authentication with certificate



3.2.34 Force no debug authentication command

--force_no_da

Description: This option allows to pass an information to the tool, to force the product state to OB programming, even if the debug authentication is not configured (password not programmed in OTP). In this case, it is no longer possible to perform regression, all debug features are disabled.

This option is available only for STM32H50x devices, it is handled only if there is a request for OB programming.

Syntax: `--force_no_da`

Prompt a warning message to highlight the case:

```
> You are trying to modify the PRODUCT_STATE while OTP are not set,
> Force No DA option is active!
```

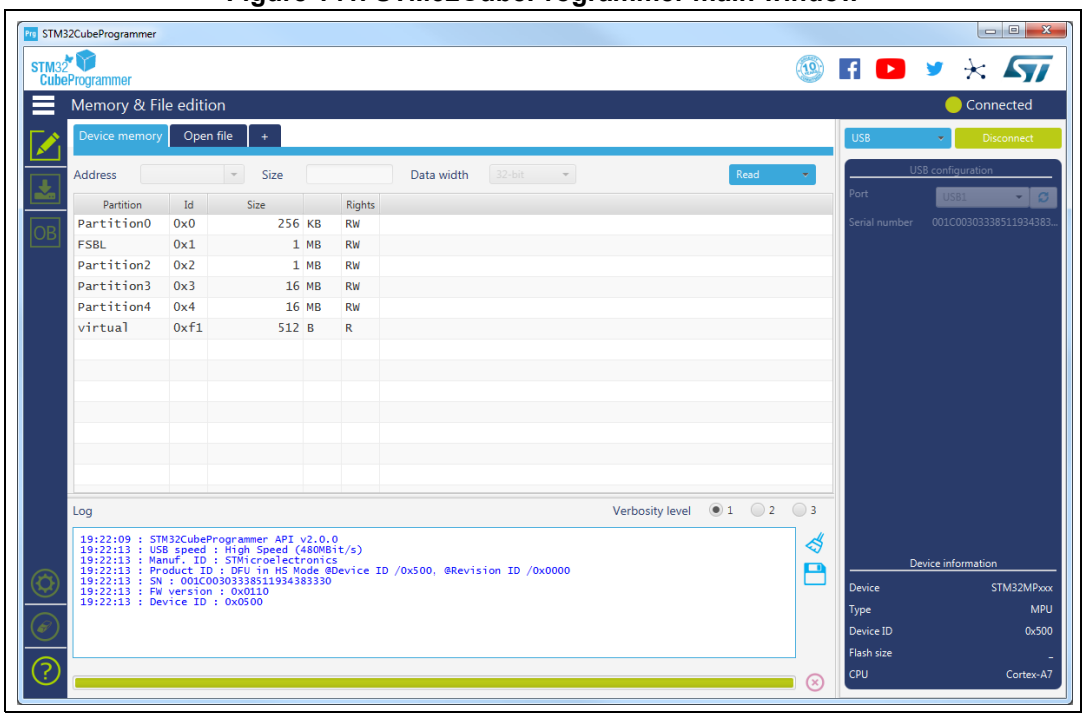
If this option is not used and you are trying to modify the Product State with OTP not configured, to avoid damages the tool stops the execution and prompts an error message:

```
> You are trying to modify the PRODUCT_STATE while OTP are not set, data
won't be downloaded.
> Please configure your device and try again.
```

4 STM32CubeProgrammer user interface for MPUs

4.1 Main window

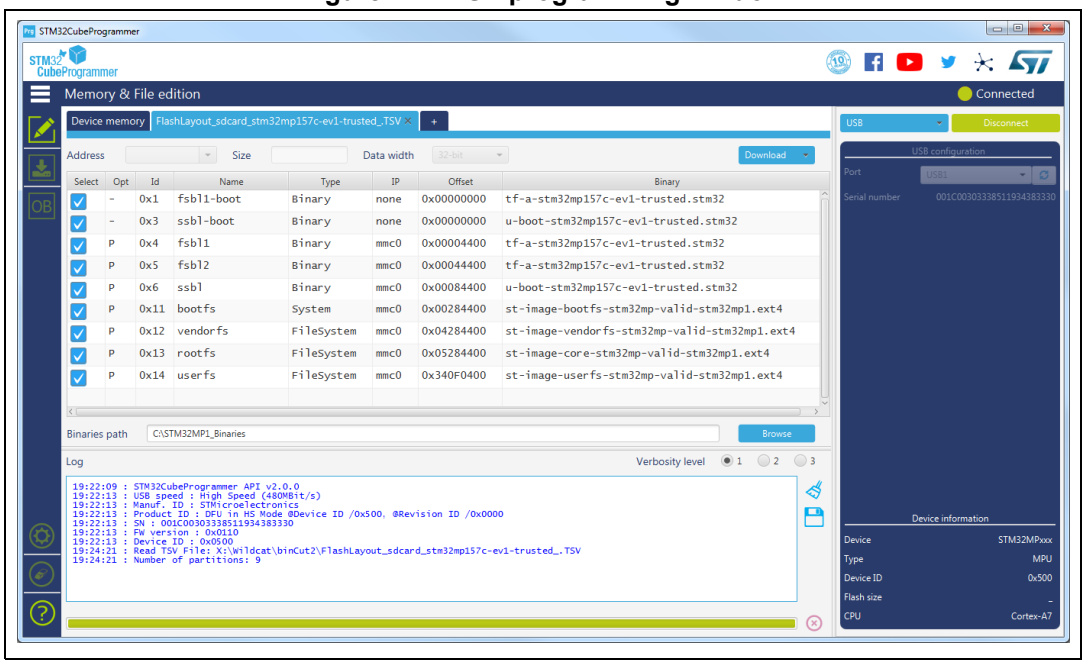
Figure 141. STM32CubeProgrammer main window



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 142. 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*.

4.3 OTP programming window

The OTP window is available exclusively for MPUs. It extracts the OTP partition [ID 0xF2] using u-boot interface to read, display and fuse the OTP registers.

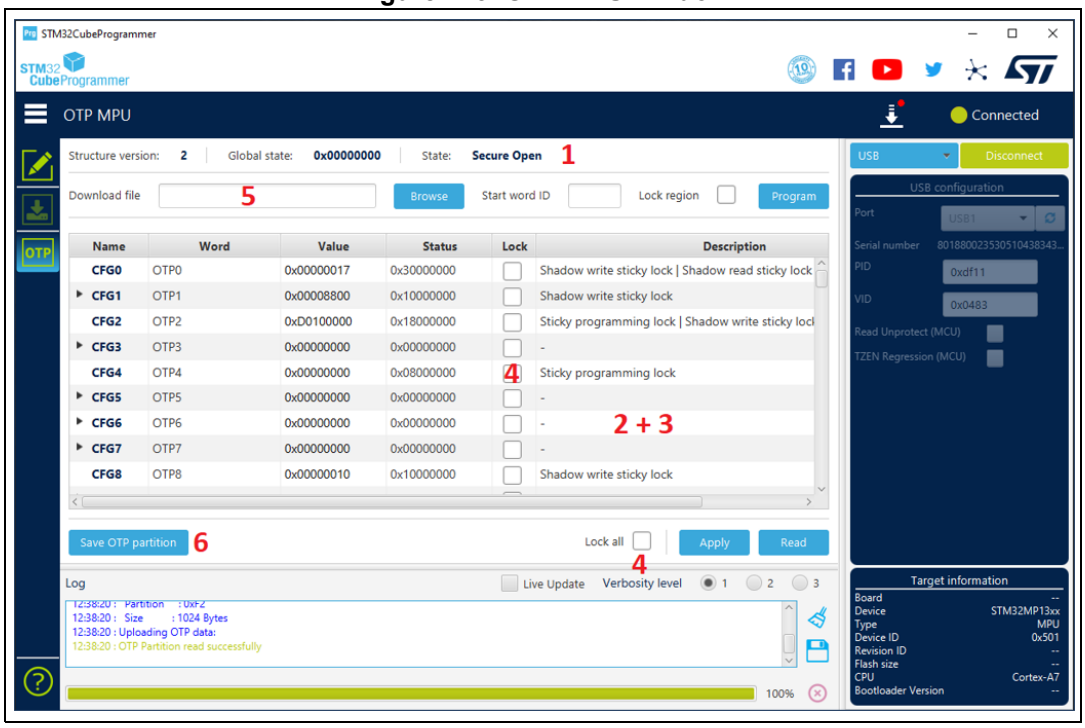
The STM32MP13xx devices have 3072 OTP (one time programmable) bits, which can be read-accessed in 96 words: OTPx (x = 0 to 95). Some OTP words are programmed by STMicroelectronics at factory (product differentiations or keys).

The UI has six main goals:

1. Get OTP structure information
2. Read and display words in table view format
3. Edit and fuse OTP registers
4. Lock specific/all words
5. Program binary files (frequently for keys fuse)
6. Save the OTP partition in output binary file (for debug)

Open the OTP window by pressing the “OTP” button from the main window to start reading the OTP partition, when this is correctly executed, it displays the words in table view.

Figure 143. OTP MPU window



4.3.1 Get OTP structure information

This section displays an overview of the connected device.

- Structure version
- Global state
- State: Secure open / Secure closed / Invalid

Note: The OTP UI is available only for structure version 2.

4.3.2 Read and display words

After checking of the OTP partition, the tool decodes and shows the words in table view format including five columns:

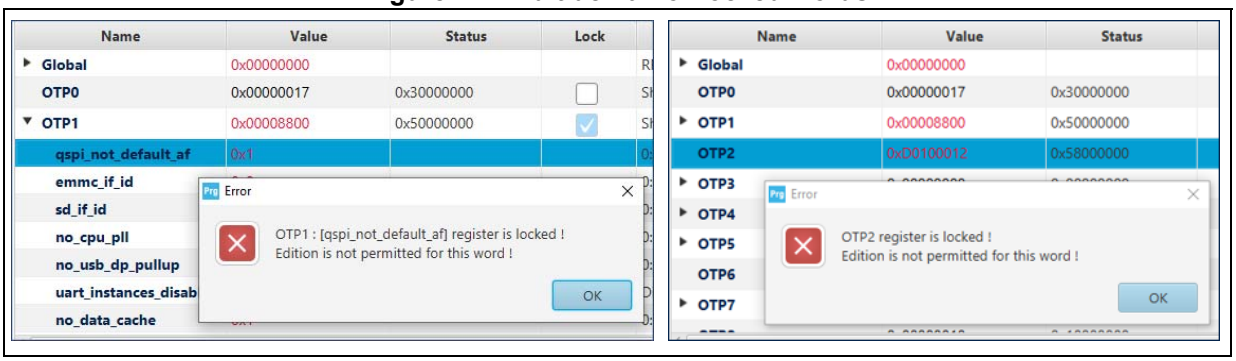
- Name: OTP word name.
- Word: OTP word ID as a tree component.
- Value: hexadecimal value (red color for locked words) as editable field.
- Status: hexadecimal value.
- Lock: indicate the lock state of the considered word as checkbox component. Checked if the item is locked, otherwise it will be unchecked. The column is disabled if the item is in permanent write lock state.
- Description: decodes the status of the OTP word and displays a brief description for children's items.

Note: Press "Read" button to refresh all table fields or to discard changes.

4.3.3 Edit and fuse words

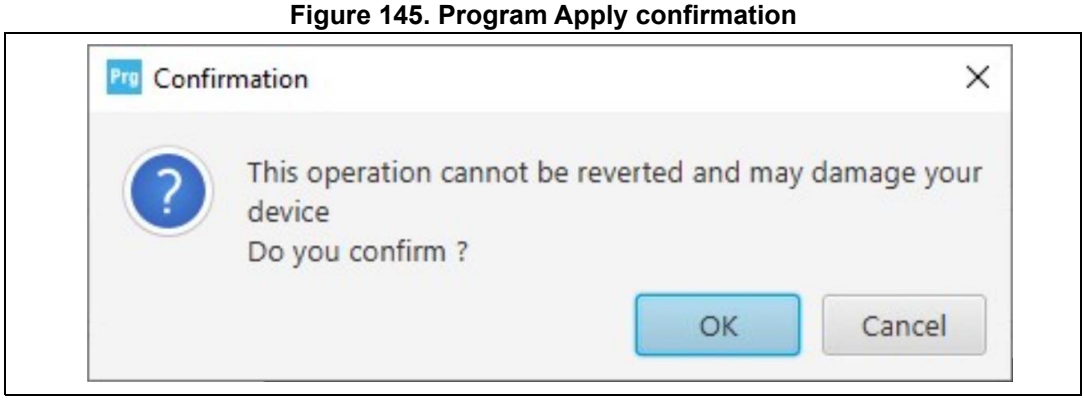
It is possible to directly edit the "Value" table cell to write a new value (press Enter after each change). The tool verifies the syntax of the input item to respect hexadecimal format and item width, then it checks the locking state of the current item before start fusing.

Figure 144. Edit denial for locked words



After modification, press "Apply" button and confirm the operation to start the update and refresh the table view.

Figure 145. Program Apply confirmation



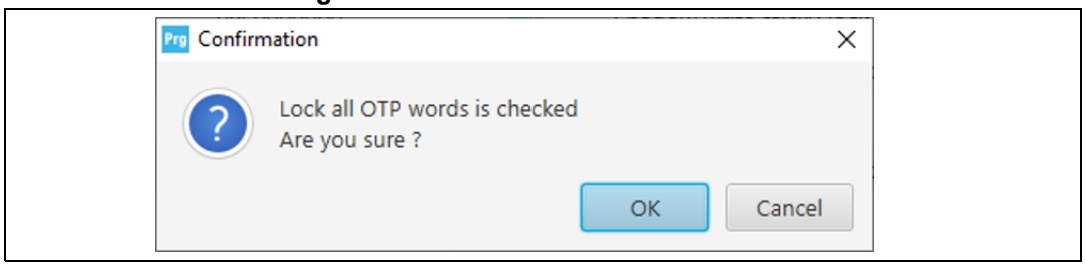
4.3.4 Lock specific/all words

The lock operation allows the user to close the write programming against any modification of the considered OTP word. It is possible to lock several words to lock on one-shot by setting the assigned checkboxes, then clicking on “Apply” to start the operation.

Note: To go faster to the initial OTP lock state, it is recommended to press the “Read” button.

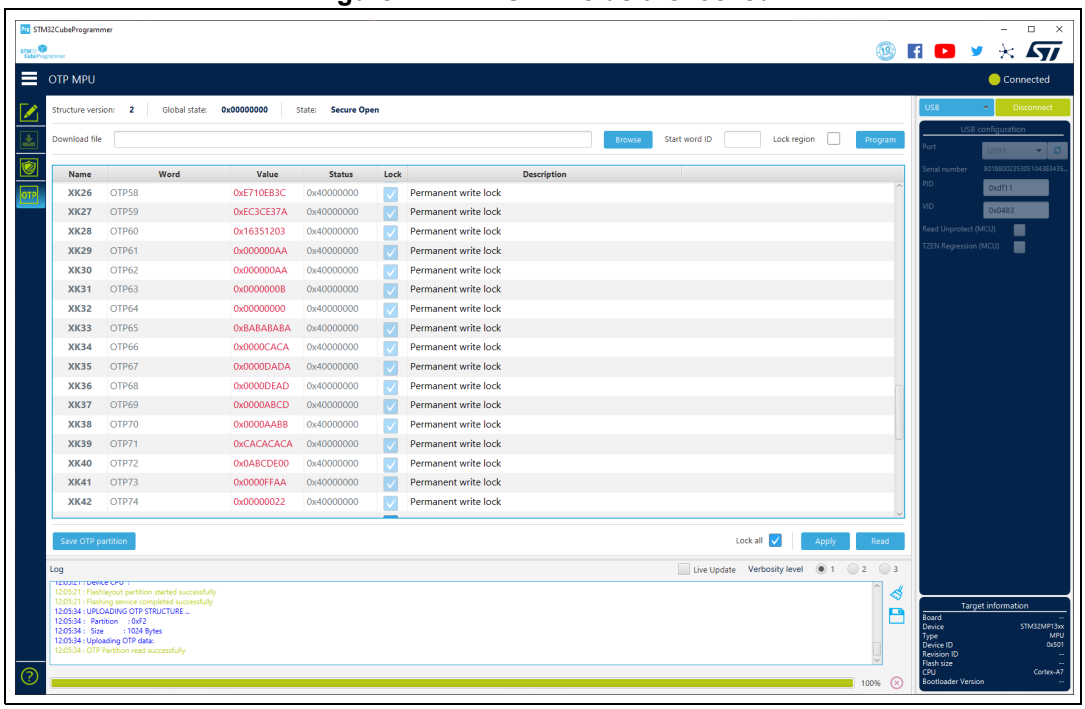
It is possible to lock all words at once by setting the “Lock all” checkbox. A message pops up to emphasize the procedure, which may result in all words being closed and no further changes allowed.

Figure 146. Lock all words confirmation



If the operation is successful, the table view is disabled for all “Lock” columns.

Figure 147. All OTP fields are locked



4.3.5 Program binary file

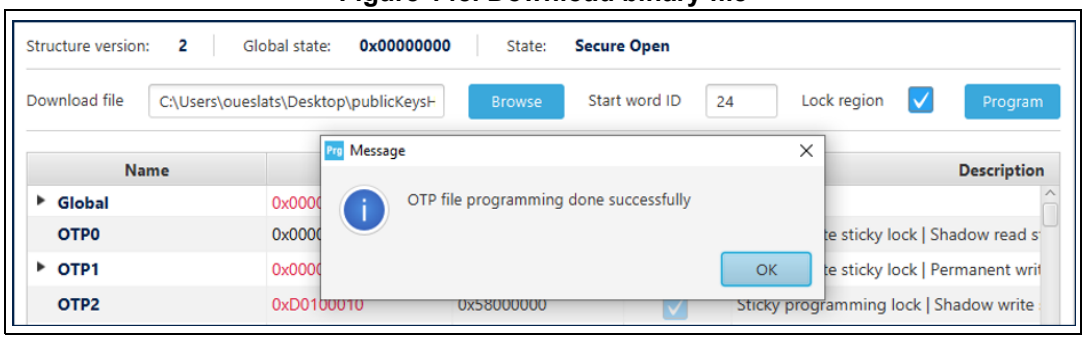
This section contains different graphical components, needed to program a binary file on the OTP registers starting from a word ID, and based on the following steps:

1. Choose the adequate binary file (with .bin extension) by clicking on “Browse”
2. Mention the start word ID in decimal format (0 to 95)
3. Check/Uncheck the “Lock region” checkbox to indicate the operation type (update or write permanent lock)
4. Press “Program” button to start the download flow

If the procedure is completed correctly, an informational dialog appears to confirm that the operation is completed.

Note: The input binary is a 32-bit aligned file, the tool adds padding values if the file is not aligned (a warning message is displayed in the log panel).

Figure 148. Download binary file



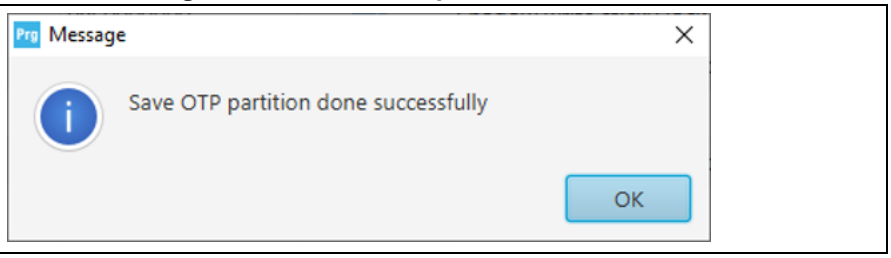
4.3.6 Save OTP partition

This capability allows the user to save the whole current OTP partition in a binary file (.bin extension), which can be used for future analysis or to archive the current device configuration.

Press on “Save OTP partition” button and choose the desired output name and directory (check permissions).

If the save is completed correctly, an informational dialog appears to confirm that the operation is completed.

Figure 149. Save OTP partition



Note: The size of the output file must be 1024 bytes.
U-boot program must be installed before launching OTP window, which is necessary to expose the OTP partition.
Words 32 to 95 do not have child fields, can be edited only once, and must be permanently

locked after programming.
Word editing and Lock operation can be performed at the same time, after clicking “Apply”.

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>]`**

`port=<Portname>`	Interface identifier: – ex COMx (for Windows) – /dev/ttySx (for Linux) – usbx for USB interface
`[noinit=<noinit_bit>]`	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).
`[br=<baudrate>]`	Baudrate, (for example 9600, 115200), default 115200.
`[P=<Parity>]`	Parity bit, value in (EVEN, NONE, ODD), default EVEN.
`[db=<data_bits>]`	Data bit, value in (6, 7, 8), default 8.
`[sb=<stop_bits>]`	Stop bit, value in (1, 1.5, 2), default 1.
`[fc=<flowControl>]`	Flow control, value in (OFF, Software, Hardware). Software and Hardware flow controls are not yet supported for STM32MP1 series, default OFF.

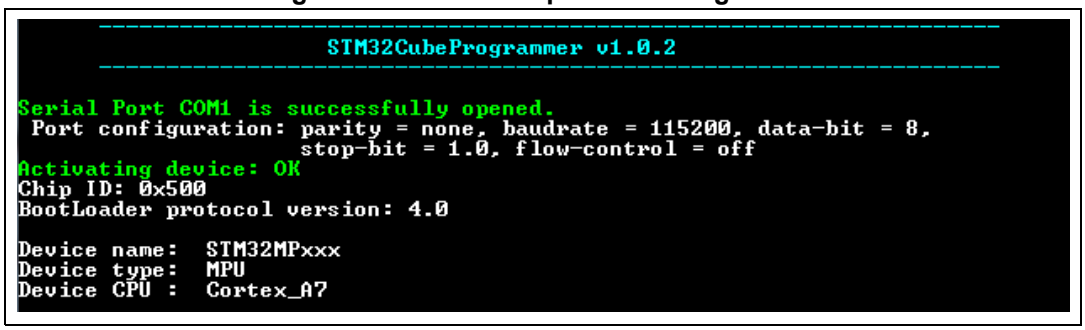
Example

Using UART:

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

The result of this example is shown in *Figure 150*.

Figure 150. Connect operation using RS232



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.

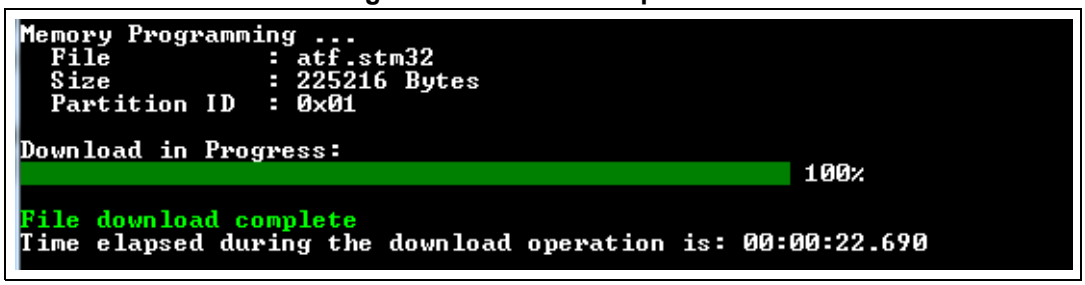
Example

```
./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 151*.

Figure 151. Download operation



Note: For U-boot with USB interface, to program the nonvolatile memory (NVM) with the loaded partition using download command, the user must execute a start command with the partition ID. Besides, to execute an application loaded in the NVM, the start address. must be specified

Example: Download and manifestation on alternate 0x1

```
./STM32_Programmer.sh -c port=usb0 -w atf.stm32 0x1 -s 0x01
```

5.1.4 Flashing service

Description: The embedded flashing service aims to load sequentially the partitions requested by the bootloader. To do this STM32CubeProgrammer needs the TSV file, which contains information about the requested partitions to be loaded.

STM32CubeProgrammer downloads and starts the requested partition ID until the end of operation (phaseID = 0xFE).

Syntax: **`-w < tsv file_path >`**

`<tsv file_path>` Path of the tsv file to be downloaded.

Figure 152. TSV file format

#Opt	Id	Name	Type	IP	Offset	Binary
-	0x01	fsbl1-boot	Binary	none	0x0	tf-a-stm32mp157c-dk2-trusted.stm32
-	0x03	ssbl-boot	Binary	none	0x0	u-boot-stm32mp157c-dk2-trusted.stm32
P	0x04	fsbl1	Binary	mmc0	0x00004400	tf-a-stm32mp157c-dk2-trusted.stm32
P	0x05	fsbl2	Binary	mmc0	0x00044400	tf-a-stm32mp157c-dk2-trusted.stm32
P	0x06	ssbl	Binary	mmc0	0x00084400	u-boot-stm32mp157c-dk2-trusted.stm32
P	0x21	bootfs	System	mmc0	0x00284400	st-image-bootfs-openstlinux-weston-extra-stm32mp1.ext4
P	0x22	vendorfs	FileSystem	mmc0	0x04284400	st-image-vendorfs-openstlinux-weston-extra-stm32mp1.ext4
P	0x23	rootfs	FileSystem	mmc0	0x05284400	st-image-weston-openstlinux-weston-extra-stm32mp1.ext4
P	0x24	userfs	FileSystem	mmc0	0x340F0400	st-image-userfs-openstlinux-weston-extra-stm32mp1.ext4

Example

```
./STM32_Programmer.sh -c port=/dev/ttyS0 p=none br=115200 -d
Flashlayout.tsv
```

Note: While programming the Flashlayout.tsv file, U-boot can spend a long time to start correctly, for this reason configure the timeout value by using the timeout command (-tm <timeout>).

5.1.5 Start command

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

Description: This command allows executing the device memory starting from the specified address.

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 153*).

Figure 153. 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. Their functionality (such as downloading or uploading a full OTP image, modifying an OTP value or proprieties) is 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 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 OTP commands

STM32CubeProgrammer exports several capabilities that can be used to manage the OTP region via various commands based on the OTP structure version, as detailed below

OTP structure 1

Programming SAFMEM

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

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]` Loads value into the chosen OTP shadow register. Value must be written in hexadecimal form.

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

Example

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

Download file command

Description: To fuse a binary file from a start word ID

Syntax: -otp fwrite {lock} [path.bin] wordid=[index]

`{lock}` Optional, to indicate the operation type, update, or permanent lock.

`[path,bin]` 32-bit aligned file, the tool makes padding values if the file is not aligned (a warning message is displayed).

`[index]` Value in hex/dec format (from 0 to 95 in decimal).

Note: At the execution time the tool displays the operation to be performed, and prompts a message for confirmation. To skip the prompt confirmation, use the (-y or --yes) command.

OTP structure 2

Programming SAFMEM

Description: This command allows the user to program SAFMEM memory by modifying the OTP words. Able to write up to 96 words in the same command line.

Syntax: **`-otp write {lock} {word=[index] value=[val]…}`**

`{lock}` Optional, to indicate that a lock has been requested. If lock option is mentioned, all words passed in line are locked.
– With lock: writes the word, then performs permanent lock.
– Without lock: updates the word.

`[index]` The word index can be written in decimal or hex format.

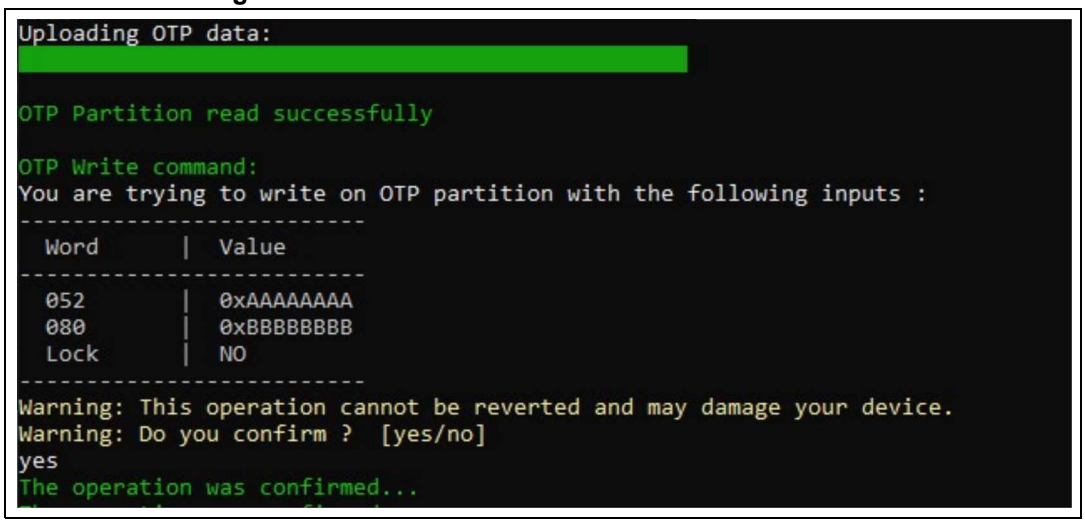
`[val]` The value option accepts hex values.

The tool prints the requests, the user can verify the operation before going on. It will then prompt a confirmation message, the user can press yes/y or no/n to, respectively, continue or stop the write operation.

Example

```
STM32_Programmer_CLI.exe --connect port=usb1 -otp write word=52
value=0xAAAAAAA word=0x50 value=0xBBBBBBBB
```

Figure 154. OTP write command for OTP structure v2



Lock OTP command

Description: This command allows to permanent lock the mentioned words, already written. Up to 96 words can be written in the same command line.

Syntax: **`-otp lock {word=[index]…]`**

`[index]` The word index can be written in decimal or hex format.

The tool prints the requested modifications, and the user can verify the operation before going ahead (use yes/y or no/n to continue or to stop)

Example

```
STM32_Programmer_CLI.exe --connect port=usb1 -otp lock word=20 word=0x30
```

Display command

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

Syntax: **`-otp displ {word=[index]…}`**

`{word=[index]...}` Optional, able to display up to 96 specific words in the same command,
The index value used to indicate the OTP word ID is in decimal or hex format.

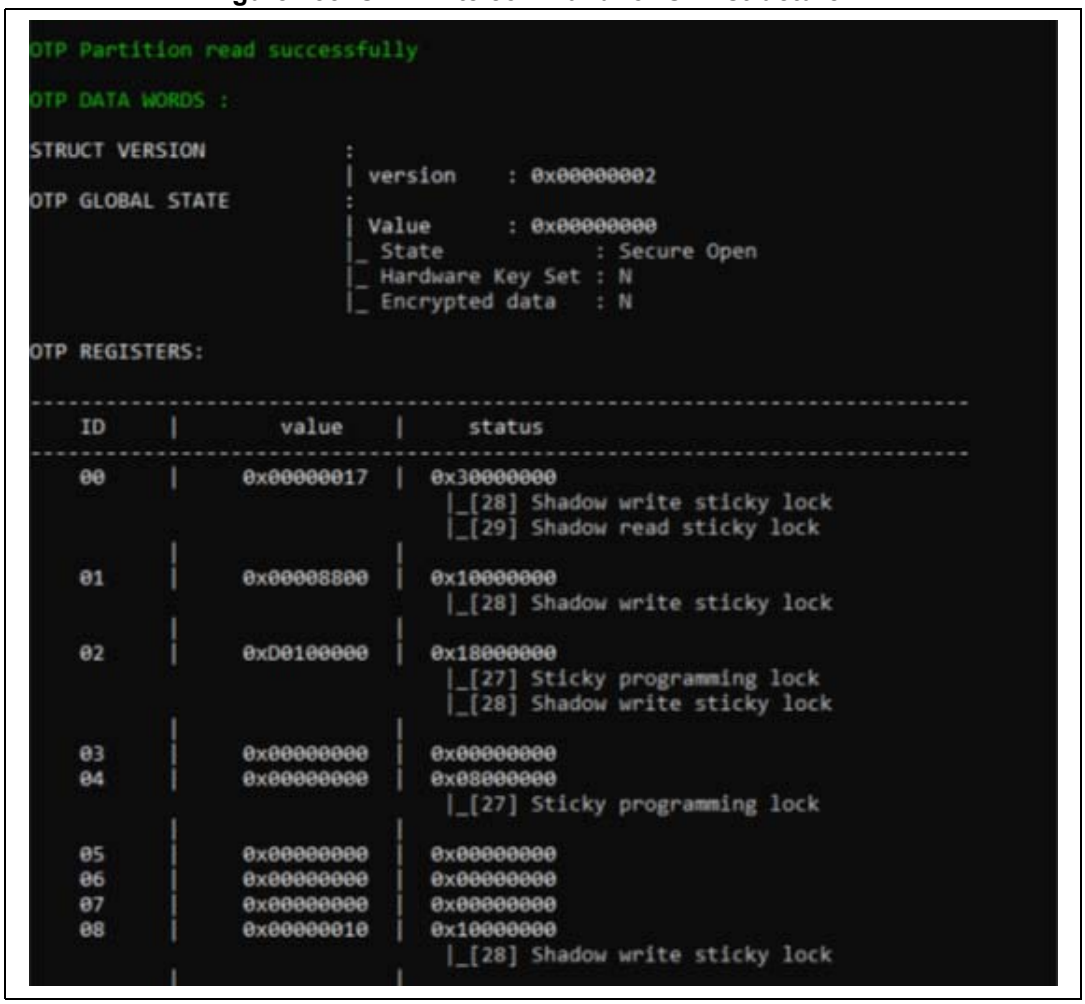
`-otp displ` Displays all OTP words (version + Global State + OTP words). Highlights the status word containing a state information (prog lock, read lock, read error, invalid).

Example

```
STM32_Programmer_CLI.exe --connect port=usb1 -otp displ
                word=8 word=0x10
```

```
STM32_Programmer_CLI.exe --connect port=usb1 -otp displ
```

Figure 155. OTP write command for OTP structure v2



Download file command

Description: to fuse a binary file from a start word ID

Syntax: **`-otp fwrite {lock} [path.bin] word=[index]`**

`{lock}`	Optional, to indicate the operation type, update, or permanent lock.
`[path,bin]`	32-bit aligned file, the tool makes padding values if the file is not aligned (a warning message is displayed).
`[index]`	Value in hex/dec format (from 0 to 95 in decimal).

Example

Program a PKH binary file starting from word number 24

```
STM32_Programmer_CLI.exe --connect port=usb1 -otp fwrite lock
/user/home/pkh.bin word=24
```

```
OTP File write command:
You are trying to program a binary file on OTP partition with the following
inputs:
---------------------------------
File name      | pkh.bin
File size      | 32 Bytes
Start word ID | 24
Lock           | YES
---------------------------------
```

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 can be used to read the chip certificate and save the data to a binary file. The resulting file is required to obtain the associated device product ID, which can then be used to select the appropriate personalization data for the HSM card before using the SSP procedure.

Syntax: **`-gc <Output_Path>`**

This command can be used only if a specific firmware (tfa-ssp) is installed, as it is the basis to retreive the stored certificate. Go through the following steps:

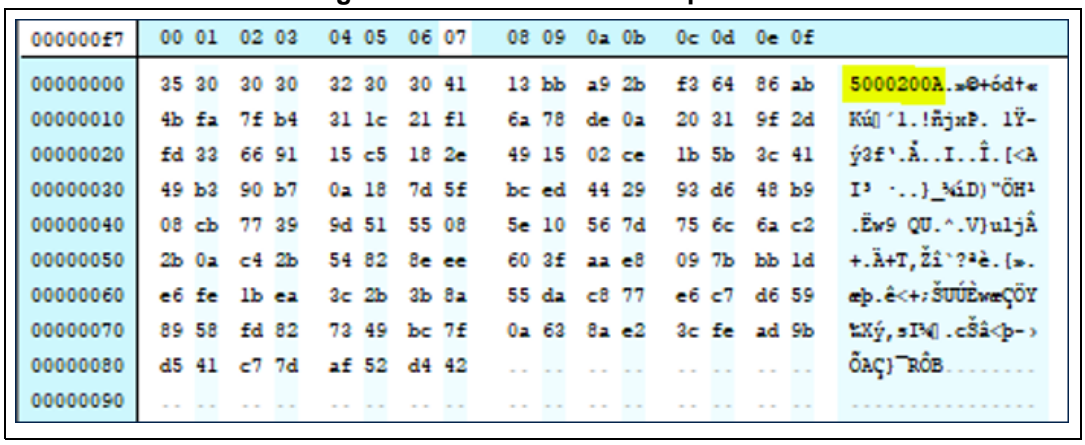
For STM32MP15xx

- *STM32_Programmer_CLI -c port=usb1 -d tf-a-ssp-trusted.stm32 0x01 -s*
- *STM32_Programmer_CLI -c port=usb1 -gc "Certificate.bin"*

For STM32MP13xx

- *STM32_Programmer_CLI -c port=usb1 -d tf-a-ssp-trusted.stm32 0x01 -s*
- *STM32_Programmer_CLI -c port=usb1 -detach*
- *STM32_Programmer_CLI -c port=usb1 -d tf-a-ssp-trusted.stm32 0x01 -s*
- *STM32_Programmer_CLI -c port=usb1 -gc "Certificate.bin"*

Figure 156. Get certificate output file



5.1.15 Write blob command

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

Syntax: **`-wb blob.bin`**

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.

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>`**

`<ssp_file_path>`	SSP file path to be programmed, bin or ssp extensions.
`<ssp-fw-path>`	SSP signed firmware path.
`<hsm=0|1>`	Set user option for HSM use (do not use / use HSM). Default value: hsm = 0.
`<license_path|slot=slotID>`	• 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 157. 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, I^2C, 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 11: ST-LINK configuration panel* – *Figure 109: STM32CubeProgrammer: available commands*. – *Figure 114: 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 9: STM32CubeProgrammer main window*, *Figure 10: Expanded main menu*, *Figure 11: ST-LINK configuration panel*, *Figure 12: UART configuration panel*, *Figure 13: USB configuration panel*, *Figure 14: Target information panel*, *Figure 15: SPI configuration panel*, *Figure 16: CAN configuration panel*, *Figure 17: I2C configuration panel*, *Figure 18: Device memory tab*, *Figure 20: File display*, *Figure 21: Flash memory programming and erasing (internal memory)* and *Figure 22: 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 21: 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 SFIx 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 SFIx specific commands*, *Section 3.2.22: STM32WB specific commands* and *Section 5.1.1: Connect command*. Added *Section 2.11: DFU IAP/USBx with custom PID and VID*, *Section 2.12: 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 22: Flash memory programming (external memory)*, *Figure 42: 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 2.12: SigFox™ credentials* and *Section 3.2.22: STM32WB specific commands*. Added *Section 2.13: Register Viewer*, *Section 2.14: 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 2.15: Fill memory command*, *Section 2.16: Fill memory operation*, *Section 2.17: Blank check command*, *Section 2.18: Blank check operation*, *Section 2.19: Compare flash memory with file*, *Section 2.20: Comparison between two files*, *Section 2.21: LiveUpdate feature* and *Section 3.2.28: RDP regression with password*. Updated *Figure 12: UART configuration panel* and *Figure 111: Enabling COM DTR pin*. Added *Figure 112: Connect operation using USB*. Minor text edits across the whole document.
17-Nov-2021	17	Added *Section 2.10: 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 2.10: 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 9: STM32CubeProgrammer main window*, *Figure 10: Expanded main menu*, *Figure 11: ST-LINK configuration panel* and *Figure 14: Target information panel*. Minor text edits across the whole document.
28-Feb-2022	18	Added *Section 1.3: Updater* with its subsections, *Section 2.4.1: MCU unlock (specific for the STM32WL series)*, and *Section 3.2.29: GetCertif command*. Updated *Section 1.1: System requirements*, *Section 2.1.4: Target configuration panel*, *Section 2.7.2: Key provisioning*, *Section 3.2.1: Connect command*, *Section 3.2.9: List command*, *Section 3.2.22: STM32WB specific commands*, and *Section 3.2.24: Specific commands for STM32WL*. Updated *Figure 120: List of available serial ports*.

Table 2. Document revision history (continued)

Date	Revision	Changes
29-Jun-2022	19	Added *Section 2.9: Secure programming interface*, *Section 4.3: OTP programming window*, and their subsections. Updated *Section 2.1.1: Main menu*, *I2C settings*, *Section 3.2.1: Connect command*, *Section 3.2.24: Specific commands for STM32WL*, *Section 3.2.26: Register viewer*, *Section 5.1.12: Programming OTP commands*, and *Section 5.1.14: GetCertif command*. Removed former *Section 5.1.16: Display command*. Updated figures *9* to *17* and *34* to *41*. Minor text edits across the whole document.
28-Nov-2022	20	Updated *Section 1.2.3: macOS install*, *Section 2.2.1: Reading and displaying target memory*, *Section 2.7.1: FUS/Stack upgrade*, *Section 3.2.4: Download 32-bit data command*, *Section 3.2.10: QuietMode command*, *Section 3.2.22: STM32WB specific commands*, *Section 3.2.23: Serial wire viewer (SWV) command*, and *Section 3.2.28: RDP regression with password*. Updated figures *17* to *19*, *35* to *37*, *39* to *41*, *Figure 145: Program Apply confirmation*, and *Figure 147: All OTP fields are locked*. Added *Section 2.9.4: SSP*, *Section 2.10.3: Loops and conditional statements*, and their subsections. Minor text edits across the whole document.
24-Feb-2023	21	Updated *Section 1.2.5: ST-LINK driver* and *Section 3.2.23: Serial wire viewer (SWV) command*. Added *Section 2.4.2: Debug authentication default configuration (STM32H573/STM32H563/STM32H562 only)*, *Section 2.4.3: Debug authentication configuration (STM32H503 only)*, *Section 2.9.5: OBKey provisioning (STM32H573/STM32H563/STM32H562 only)*, *Section 2.9.6: Debug authentication (STM32H5 series only)*, *Section 2.22: Calculator*, and sections *3.2.30* to *3.2.34*. Updated figures *43* to *47* in *Section 2.9.2: RDP regression with password (STM32U5 series only)*. Minor text edits across the whole document.