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, support for a large choice of connections (JTAG, SWD, USB, UART, SPI, CAN, I2C), with manual operation or automation through scripting.

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

Contents

List of figures

1 Getting started

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

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

1.1 System requirements

Supported operating systems and architectures:

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

The Java™ SE Run Time Environment 1.8 must be installed (download available from *www.oracle.com*).

Note: Only Java8 is supported.

If OpenJDK is used, be sure to download and install the OpenJFx library.

The minimal supported screen resolution is 1024x768.

Note: STLINK-V3SET is not supported on Linux32.

1.2 Installing STM32CubeProgrammer

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

1.2.1 Linux install

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

sudo apt-get install libusb-1.0.0-dev

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

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

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

To get STM32CubeProgrammer working with Ubuntu®18.04

1. Install OpenJDK 8
   a) *sudo apt install openjdk-8-jre-headless*
2. Set OpenJDK 8 as your default Java Runtime Engine
   a) *sudo update-alternatives --config java*
3. Install OpenJFX
   a) *sudo apt purge openjfx*
   b) *sudo apt install openjfx=8u161-b12-1ubuntu2 libopenjfx-jni=8u161-b12-1ubuntu2 libopenjfx-java=8u161-b12-1ubuntu2*
   c) *sudo apt-mark hold openjfx libopenjfx-jni libopenjfx-java*

To install the STM32CubeProgrammer tool, you need to download and extract the zip package and execute *SetupSTM32CubeProgrammer-vx.y.z.linux*, which guides you through the installation process.

1.2.2 Windows install

To install the STM32CubeProgrammer tool, you need to download and extract the zip package and execute *SetupSTM32CubeProgrammer-vx.y.z.exe,* which guides you through the installation process.

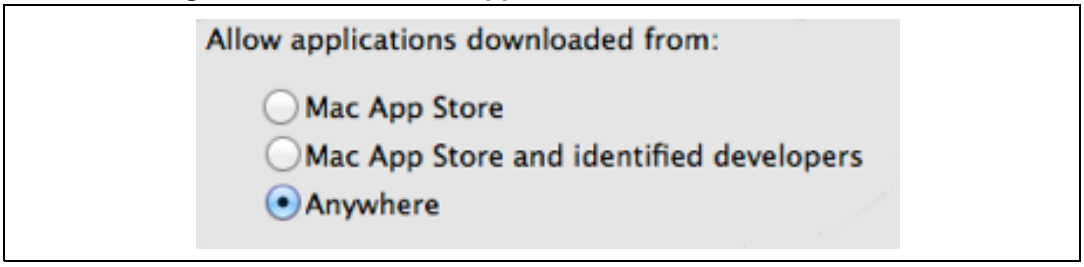
1.2.3 macOS install

To install the STM32CubeProgrammer tool, you need to download and extract the zip package and execute *SetupSTM32CubeProgrammer-vx.y.z.app,* which guides you through the installation process.

To install STM32CubeProgrammer on MacOs, execute the following steps:

1. Open a terminal and enter the following:
   sudo spctl --master-disable
2. Open the Apple menu > System Preferences > Security & Privacy > General tab. Under “Allow applications downloaded from:” select “Anywhere”.

Figure 1. macOS “Allow applications downloaded from:” tab



You now need to download and extract the zip package 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.

1.2.4 DFU driver

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

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

Figure 2. Deleting the old driver software

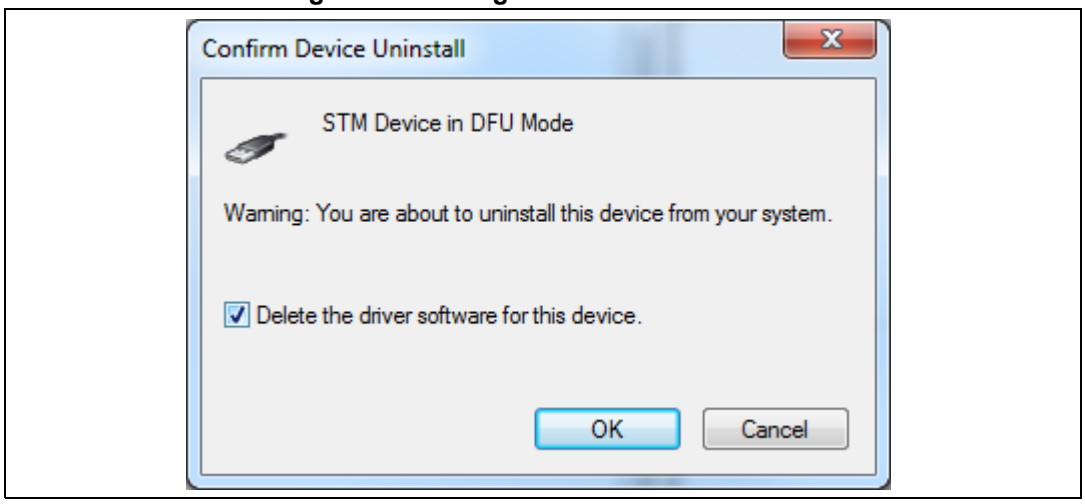


Figure 3. STM32 DFU device with DfuSe driver

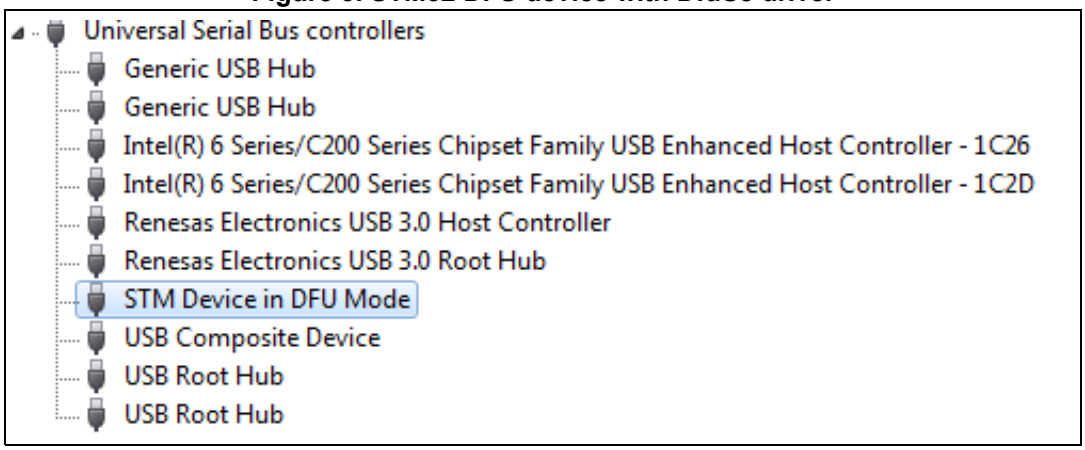
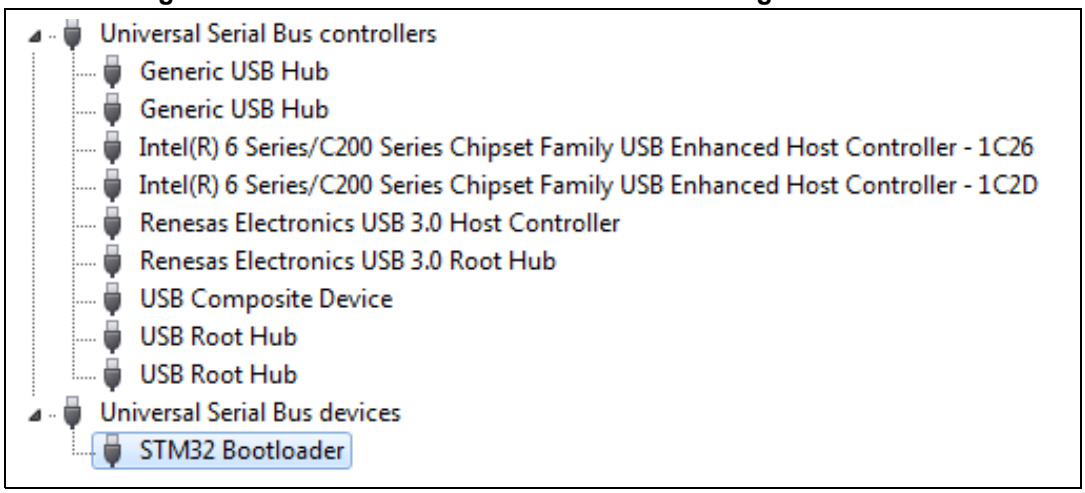


Figure 4. STM32 DFU device with STM32CubeProgrammer driver



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

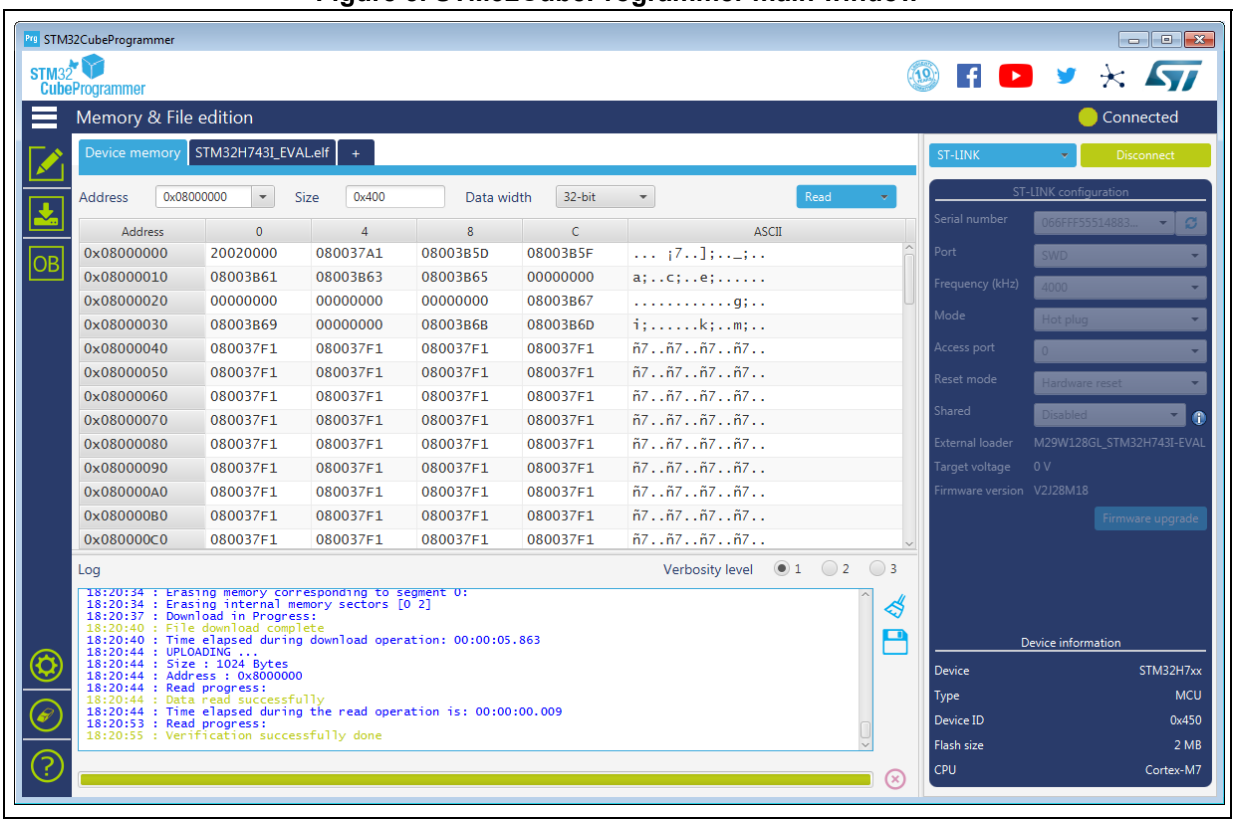
1.2.5 ST-LINK driver

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

2 STM32CubeProgrammer user interface for MCUs

2.1 Main window

Figure 5. STM32CubeProgrammer main window



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

2.1.1 Main menu

The Main menu allows the user to switch between the three main panels of the Memory and file edition, Memory programming and erasing, and Option byes tools.

By clicking on the Hamburger menu (the three-line button) on the top left corner, the Main menu expands and displays the textual description shown in *Figure 6*.

Figure 6. Expanded main menu

2.1.2 Log panel

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

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

2.1.3 Progress bar

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

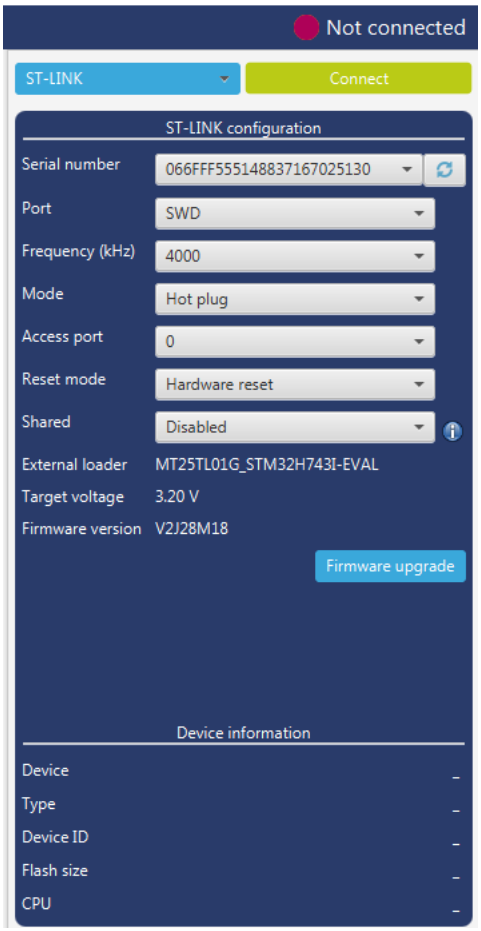
2.1.4 Target configuration panel

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

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

ST-LINK settings

Figure 7. ST-LINK configuration panel



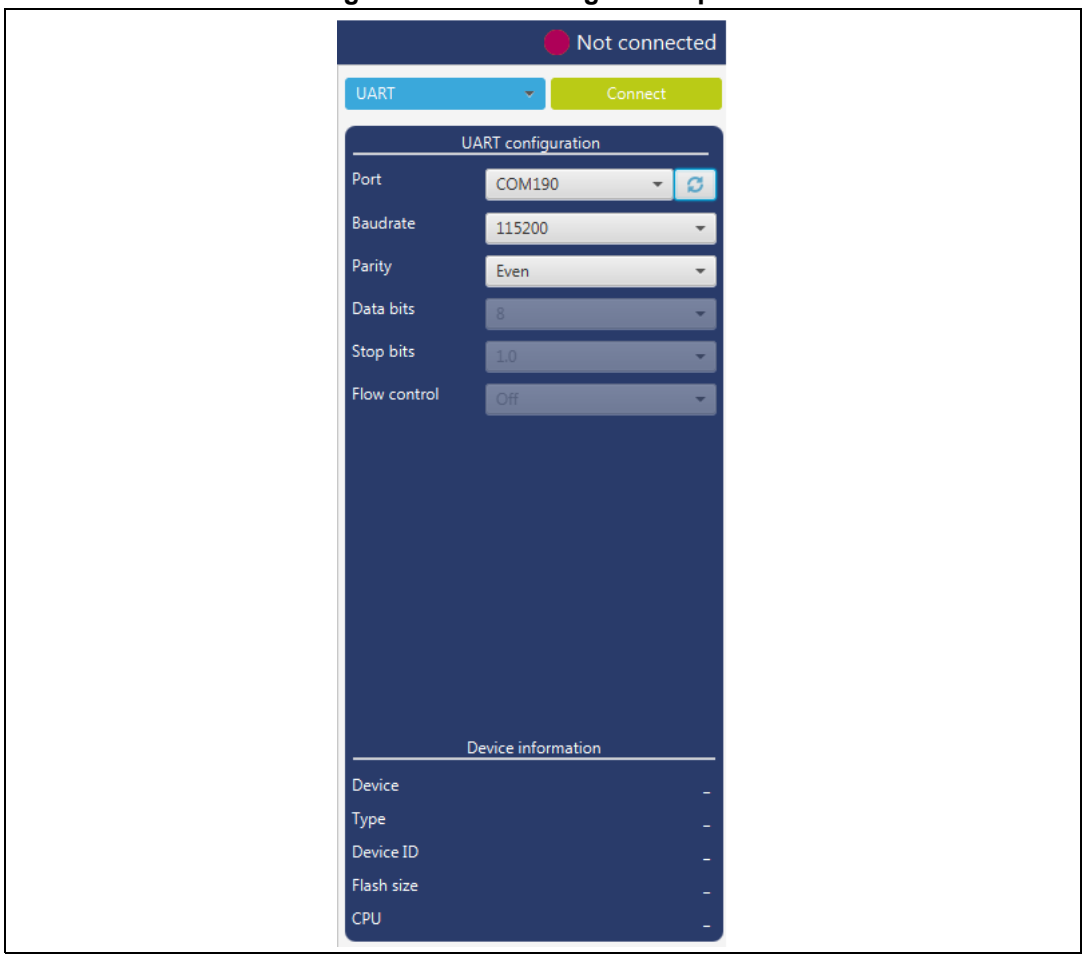
- **Serial number**: This field contains the serial numbers of all connected ST-LINK probes. The user can choose one of them, based on its serial number.
- **Port**: ST-LINK probe supports two debug protocols, JTAG and SWD.

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

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

UART settings

Figure 8. UART configuration panel



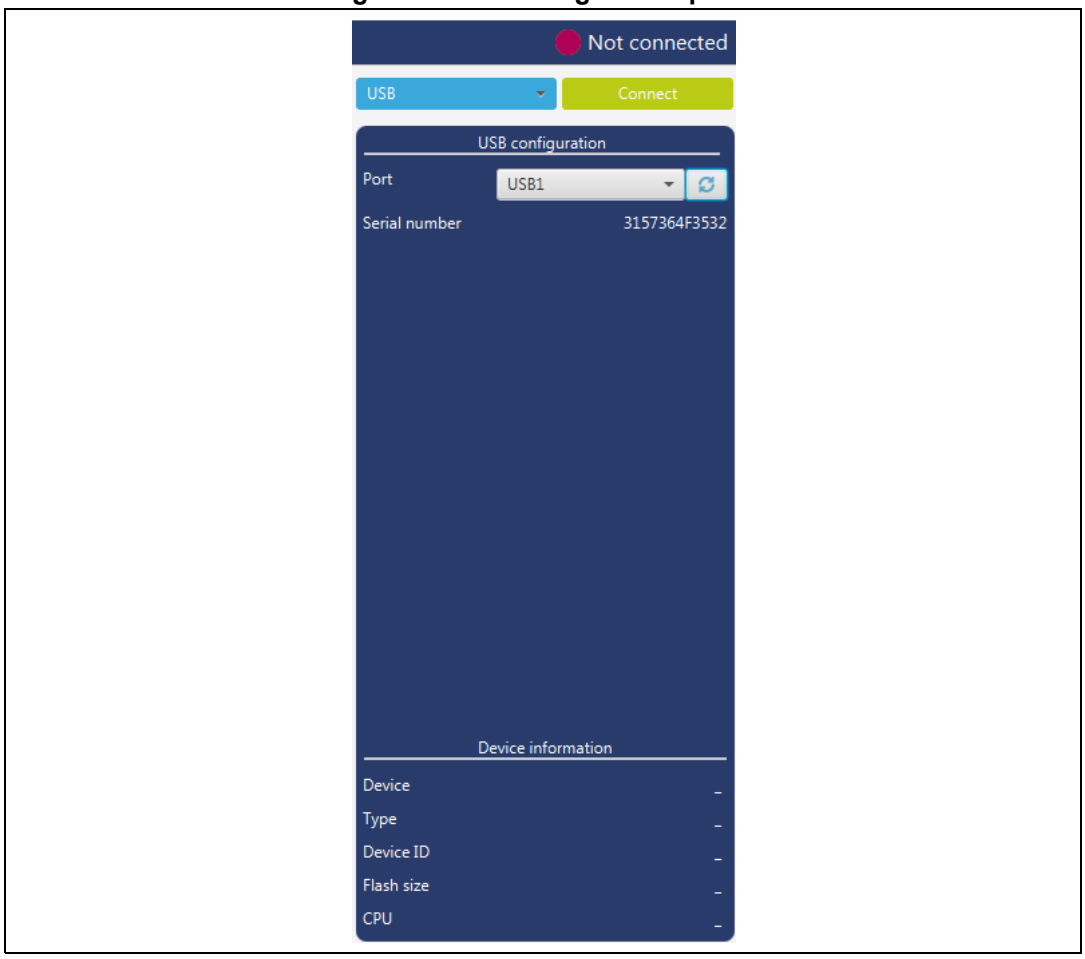
- **Port**: Selects the com port to which the target STM32 is connected. Use the refresh button to recheck the available com port on the PC.

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

- **Baudrate**: Selects the UART baud rate.
- **Parity**: Selects the parity (even, odd, none). Must be 'even' for all STM32 devices.
- **Data bits**: Must be always 8. Only 8-bit data is supported by the STM32.
- **Stop bits**: Must be always 1. Only 1-bit stop bit is supported by the STM32.
- **Flow control**: Must be always off.

USB settings

Figure 9. 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 the AN2606, available on www.st.com, for more information on the STM32 bootloader.

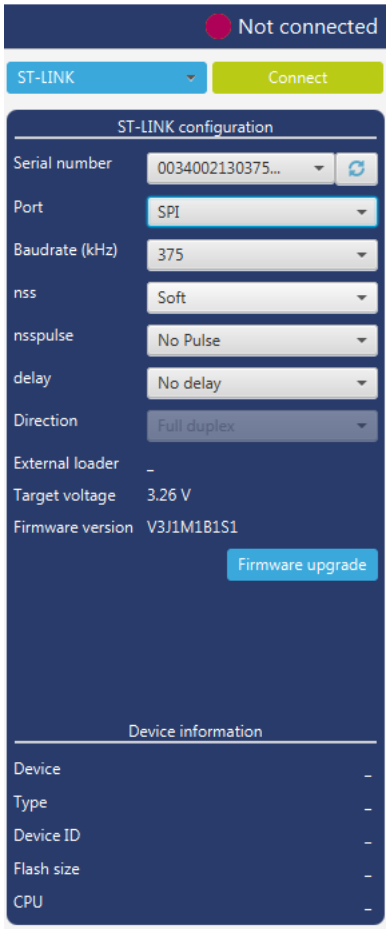
Once the correct interface settings are set, click on the 'Connect' button to connect to the target interface. If the connection succeeds, it is shown in the indicator above the button, which turns to green.

Once connected, the target information is displayed in the device information section below the settings section, which is then disabled as in *Figure 10*.

Figure 10. Target information panel

SPI settings

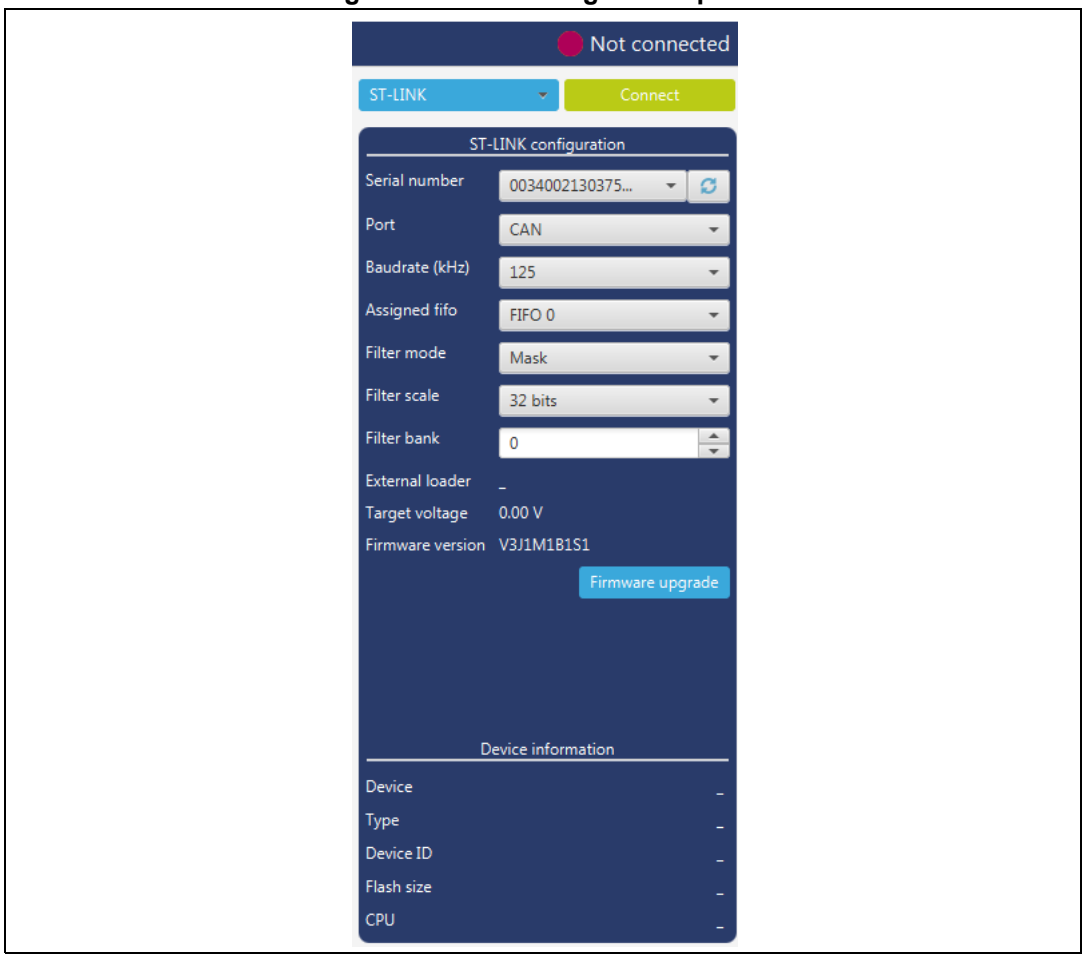
Figure 11. SPI configuration panel



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

CAN settings

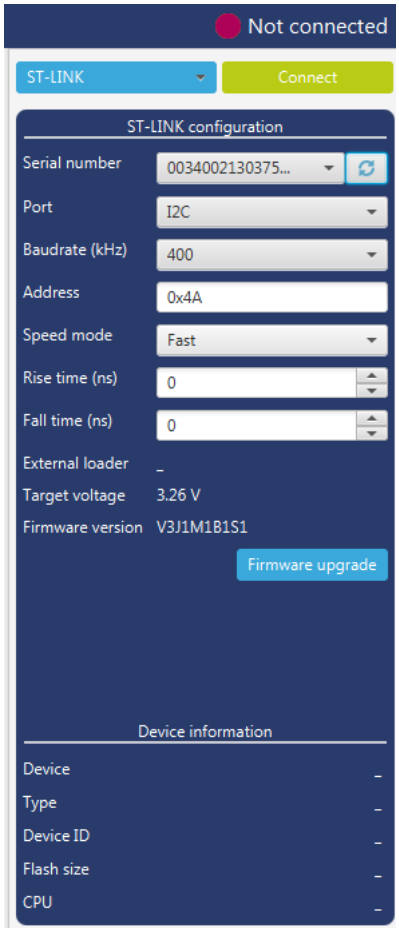
Figure 12. CAN configuration panel



- **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 13. I2C configuration panel



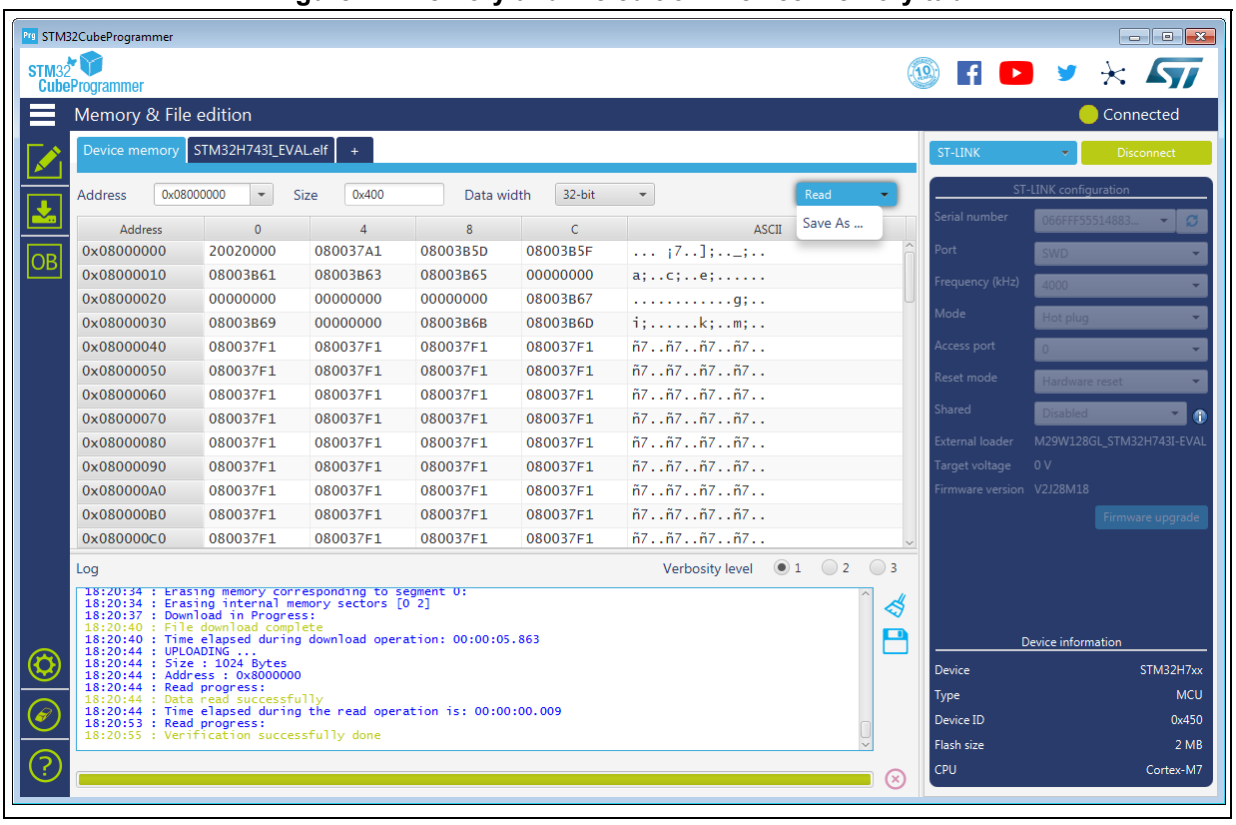
- **Serial number**: This field contains the serial numbers of all connected ST-LINK-V3 probes in case to use I2C bootloader.
- **Port**: Selects the I2C devices connected to the PC. You can use the refresh button to recheck the available devices.
- **Baudrate**: Selects the I2C baud rate.
- **Address**: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

2.2 Memory and file edition

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

2.2.1 Reading and displaying target memory

Figure 14. Memory and file edition: Device memory tab

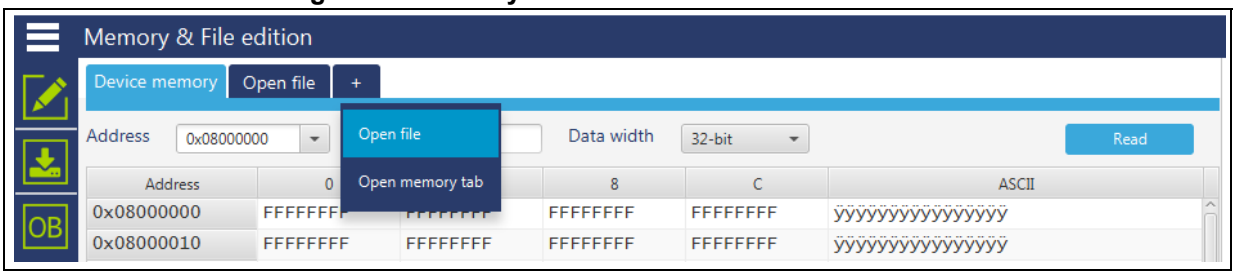


After target connection, you can read the STM32 target memory using this panel. To do this, specify the address and the size of the data to be read, then click on the Read button in the top-left corner. Data can be displayed in different formats (8-, 16- and 32-bit) using the ‘Data width’ combo box.

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

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

Figure 15. Memory and file edition: Contextual menu



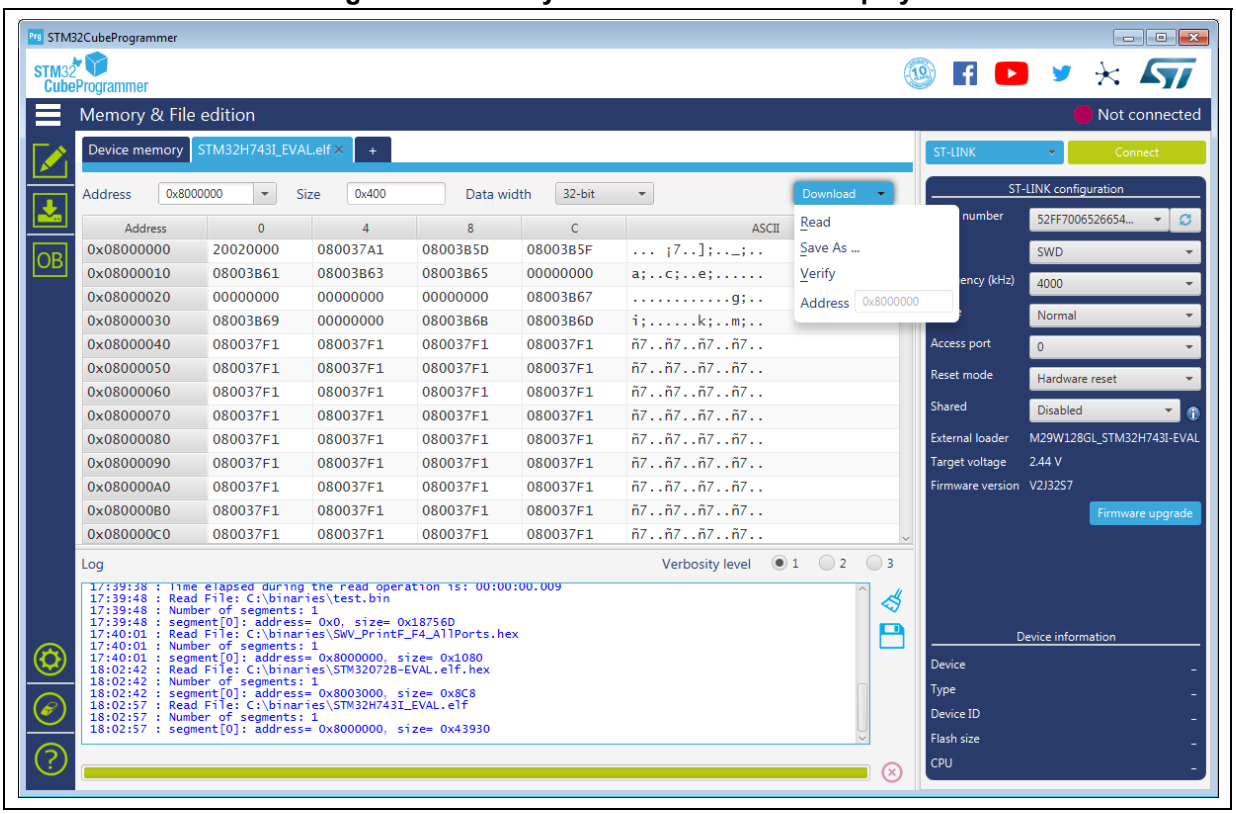
2.2.2 Reading and displaying a file

To open and display a file, just click on the “+” and select ‘Open File’ menu, as illustrated in *Figure 15*.

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

Once the file is opened and parsed, it is displayed in a dedicated tab with its name, as illustrated in *Figure 16*. 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 16. Memory and file edition: File display



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

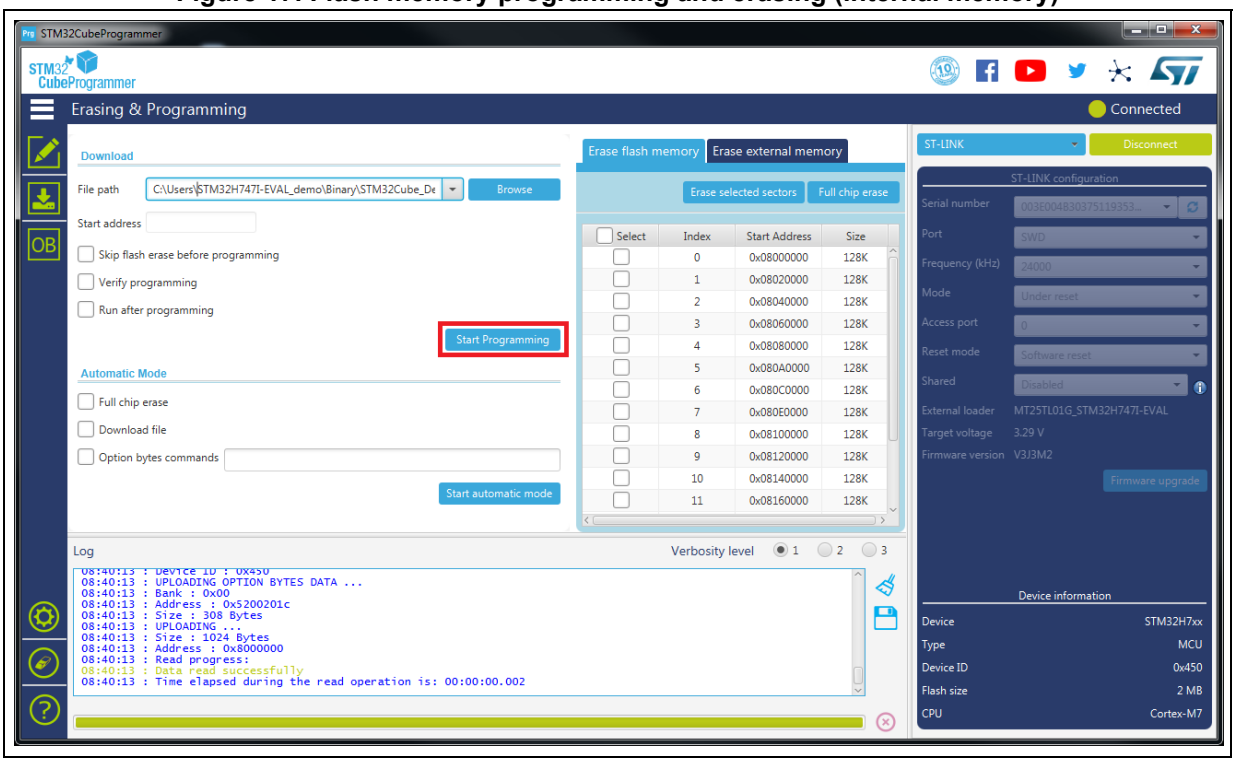
As for the ‘Device memory’ tab, you can display the file memory content in different formats (8-, 16- and 32-bit) using the ‘Data width’ combo box.

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 17. Flash memory programming and erasing (internal memory)



Memory erasing

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

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

Memory programming

To program a memory execute the following steps:

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

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

2.3.2 External Flash memory programming

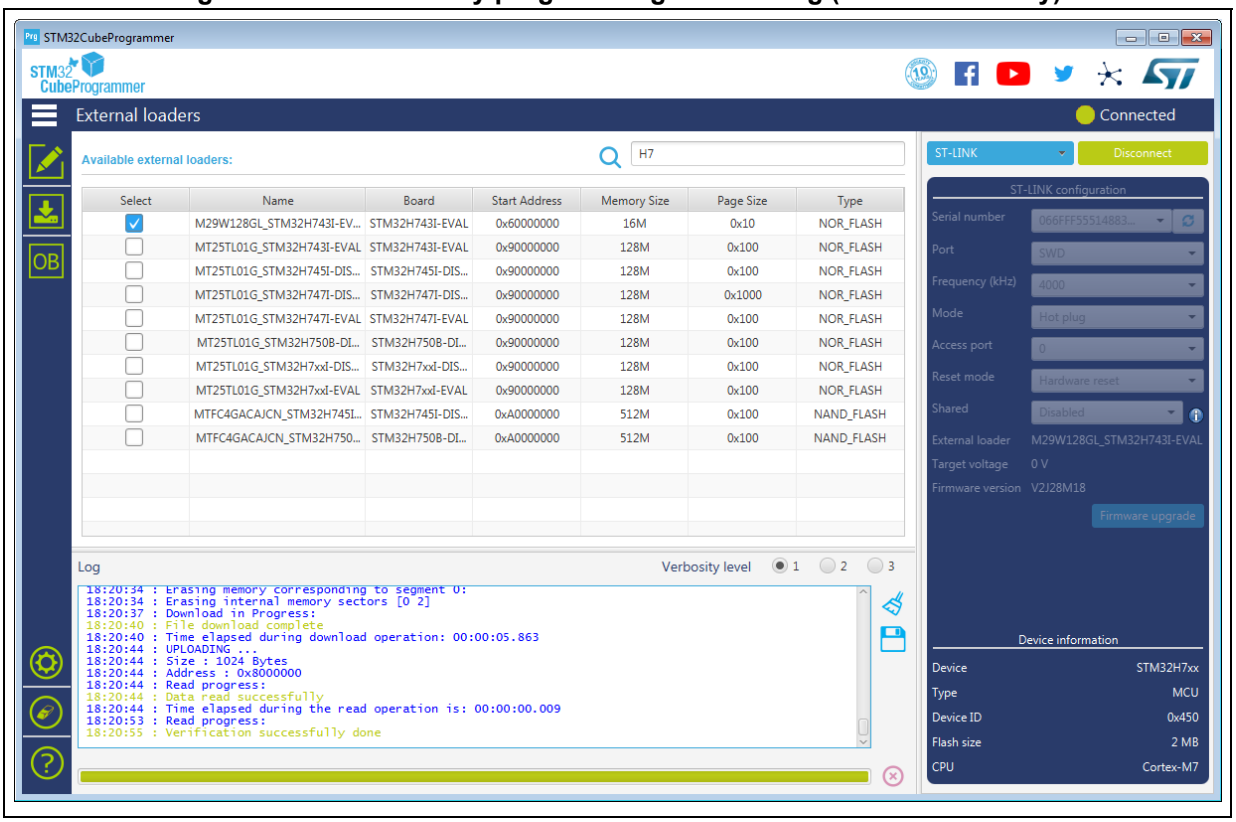
If you need to program an external memory connected to the STM32 via any of the available interfaces (e.g. SPI, FMC, FSMC, QSPI, OCTOSPI) you need an external loader.

STM32CubeProgrammer is delivered with external loaders for most available STM32 Evaluation and Discovery boards available under the “bin/ExternalLoader” directory. If you need to create a new external loader, see *Section 2.3.3* for more details.

To program an external memory, select the external loader from the “ExternalLoader” panel to be used by the tool to read, program, or erase external memories as shown in *Figure 18*. Once selected, this external loader is used for any memory operation in its memory range.

The “External flash erasing” tab on the right of the “Erasing and Programming” panel displays the memory sectors, and enables sector or full-chip erase.

Figure 18. Flash memory programming and 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: MDK-ARM™, EWARM and TrueSTUDIO®. The development of custom loaders can be performed using one of these toolchains, keeping the same compiler/linker configurations, as in the examples.

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

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

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

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

Linker or scatter files must not be modified.

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

The extension of the ELF file must be changed to '.stldr' and the file must be copied under the "*bin/ExternalLoader*" directory.

Loader_Src.c file

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

- `Init` function

  The `Init` function defines the used GPIO pins connecting the external memory to the device, and initializes the clock of the used IPs.

  Returns 1 if success, and 0 if failure.

  ```
  int Init (void)
  ```

- `Write` function

  The `Write` function programs a buffer defined by an address in the RAM range.

  Returns 1 if success, and 0 if failure.

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

- `SectorErase` function

The `SectorErase` function erases the memory specified sectors.

Returns 1 if success, and 0 if failure.

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

Where "`StartAddress`" equals the address of the first sector to be erased and "`EndAddress`" equals the address of the end sector to be erased.

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

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

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

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

- `Read` function

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

  Returns 1 if success, and 0 if failure.

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

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

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

- `Verify` function

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

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

where “AddressFirstError” is the address of the first mismatch, and “checksum” is the checksum value of the programmed buffer

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

- MassErase function

  The MassErase function erases the full memory.

  Returns 1 if success, and 0 if failure.

  ```
  int MassErase (void)
  ```

- A Checksum function

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

Dev_Inf.c file

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

```
#if defined (__ICCARM__)
     __root struct StorageInfo const StorageInfo = {
#else
     struct StorageInfo const StorageInfo = {
#endif
     "External_Loader_Name", // Device Name + version number
     MCU_FLASH, // Device Type
     0x08000000, // Device Start Address
     0x00100000, // Device Size in Bytes (1MBytes/8Mbits)
     0x00004000, // Programming Page Size 16KBytes
     0xFF, // Initial Content of Erased Memory
// Specify Size and Address of Sectors (view example below)
     0x00000004, 0x00004000, // Sector Num : 4 ,Sector Size: 16KBytes
     0x00000001, 0x00010000, // Sector Num : 1 ,Sector Size: 64KBytes
     0x00000007, 0x00020000, // Sector Num : 7 ,Sector Size: 128KBytes
     0x00000000, 0x00000000,
};
```

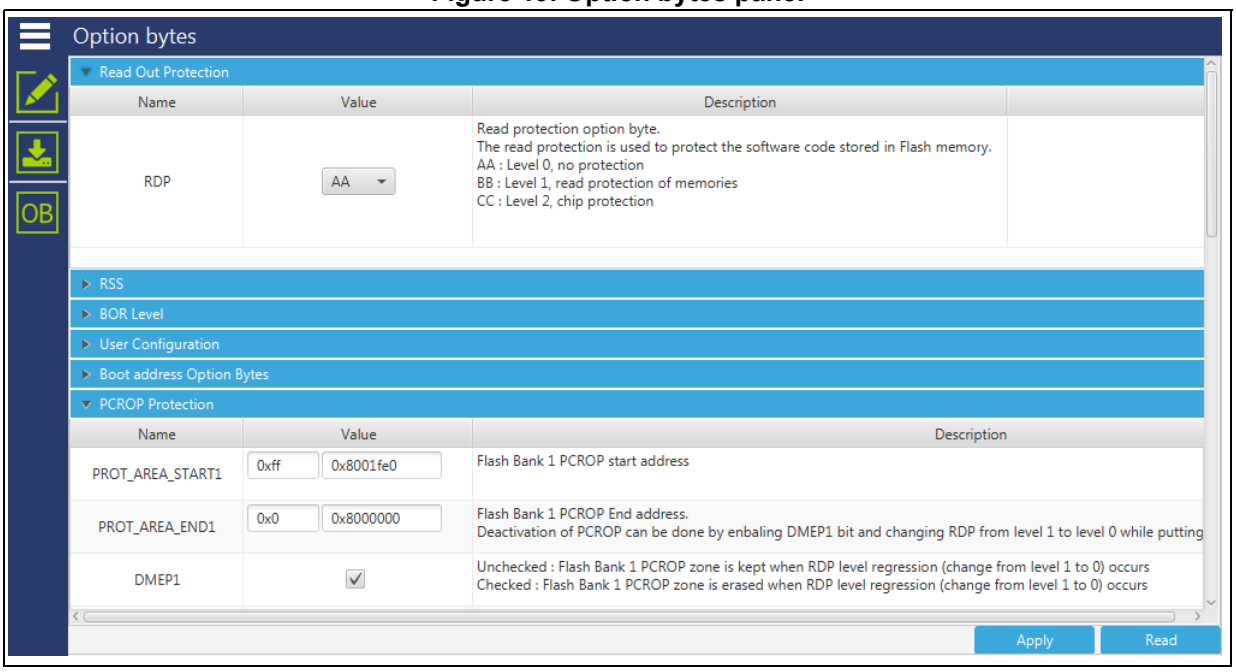
2.4 Option bytes

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

You 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.

You can click at any time on the read button, to read and refresh the displayed option bytes.

Figure 19. Option bytes panel



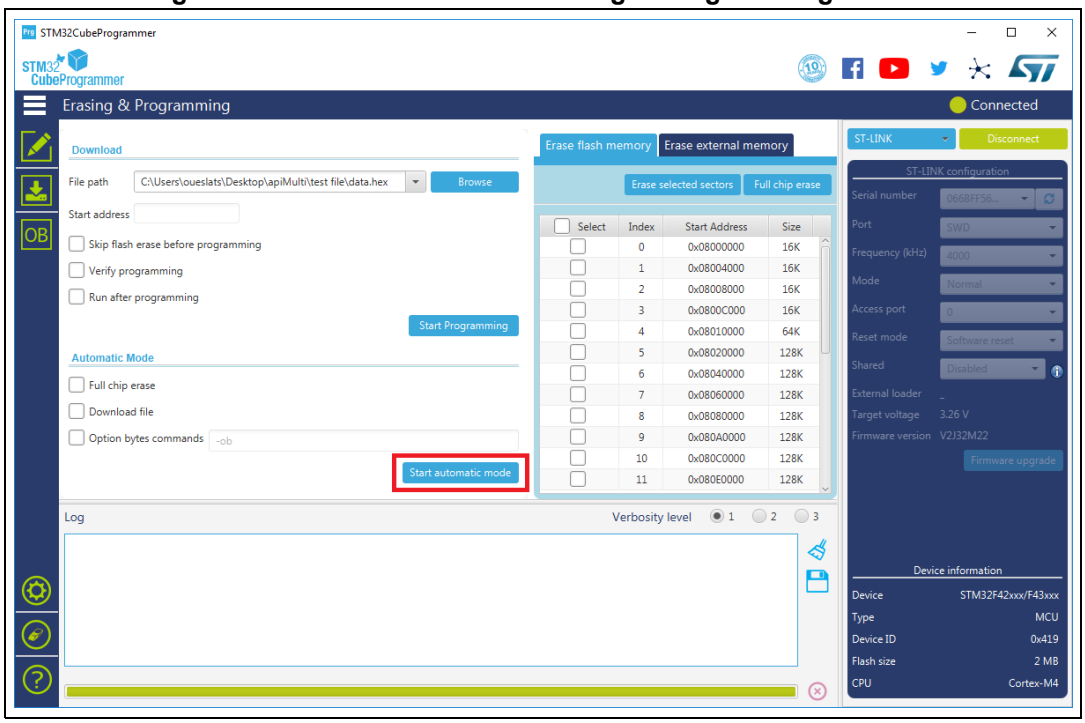
For more details, refer to the option bytes section in the Flash memory programming manual and reference manual available from *www.st.com*.

2.5 Automatic mode

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

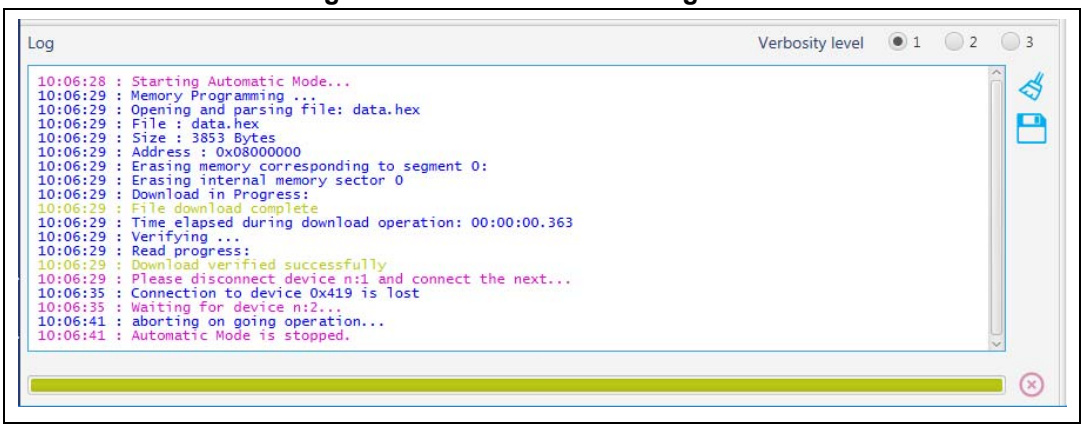
- Full chip erase: erase all the Flash memory
- Download file: activate and set programming options from Download section:
  - File path
  - Start address
  - Skip flash erase before programming
  - Verify programming
  - Run after programming
- Option bytes commands: configure the device by setting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

Figure 20. Automatic mode in Erasing & Programming window



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

Figure 21. Automatic mode Log traces



Graphical guide

- Connection to a first target must be established before performing automatic mode to collect connection parameters values associated to all next devices.
- If the Download file is checked, the system takes all Download file options in consideration, otherwise any Download option is performed.
- If the Option bytes commands is checked, the text field is activated, then the user can insert option bytes commands (like CLI commands), and make sure that there are no

white spaces at the beginning:
`-ob [OptionByte=value] [OptionByte=value] [OptionByte=value] …`

- Example of Option bytes command: "`-ob BOR_LEV=0 nBOOT0=1`"
- If the Start automatic mode button is pressed, the system enters in a loop, until a system stop is called.
- While the automatic mode is in execution state, all graphical objects are disabled.
- The user can stop the process at any preferred time by pressing cancel button or stop automatic mode button.

Log messages

- "Starting Automatic Mode..."

  Indicates that the system entered successfully in automatic process.
- "More than one ST-LINK probe detected! Keep only one ST-LINK probe! "

  The automatic mode cannot be used if more than one ST-LINK probe is connected to the computer when using JTAG/SWD interfaces. A message is displayed to prevent the user and ask him to keep only one ST-LINK probe connected to continue using this mode.
- "More than one ST-LINK Bridge detected! Keep only one ST-LINK Bridge!"

  The automatic mode cannot be used if more than one ST-LINK bridge is connected to the computer when using bootloader interface SPI/CAN/I^2C interfaces. A message is displayed to prevent the user and ask him to keep only one ST-LINK bridge connected to continue using this mode.
- "More than one ST-LINK USB DFU detected! Keep only one USB DFU!"

  The automatic mode cannot be used if more than one USB DFU is connected to the computer when using USB bootloader interface. A message is displayed to prevent the user and ask him to keep only one USB DFU connected to continue using this mode.
- "More UART ports detected than last connection!"

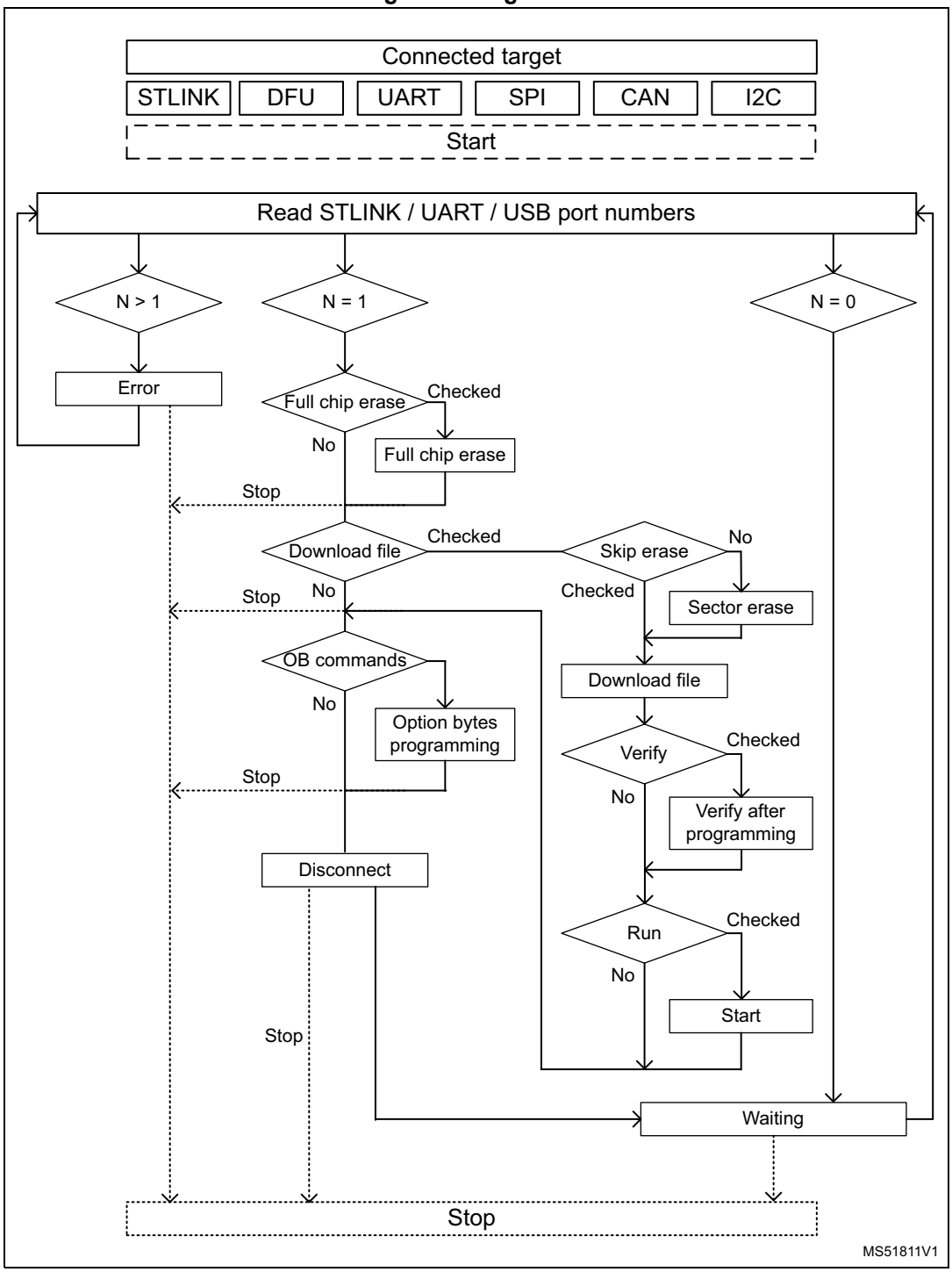
  In the first connection time the automatic mode calculates the number of the available Serial ports and put it as a reference to detect correctly that we use only one port UART for STM32 device.
- "Please disconnect device and connect the next..."

  If the system finishes the first process, and whatever the result, disconnect the current device to prepare the second device connection.
- "Waiting for device..."

  Once the connection to the previous device is correctly lost, the system keeps searching for a new device.
- "Automatic Mode is stopped."

  Indicates that there is a required cancel and the system stops the process.

Figure 22. Algorithm



2.6 STM32WB OTA programming

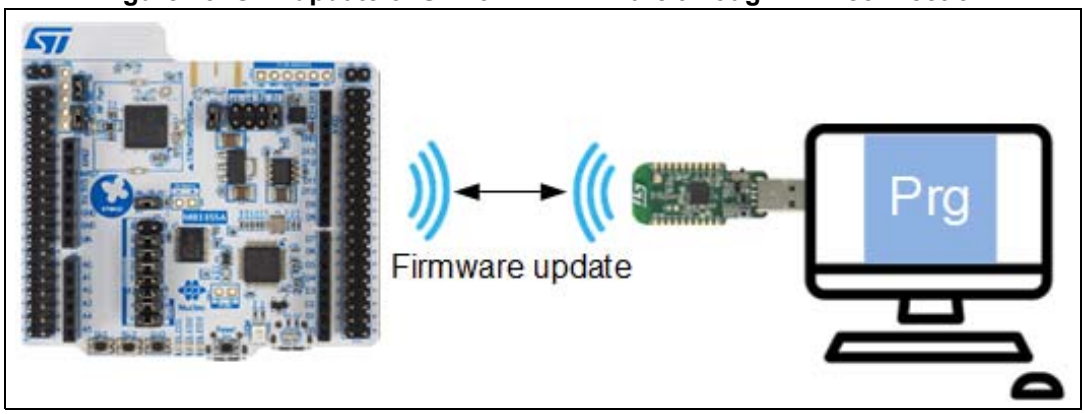
Over-the-air (OTA) programming mode in STM32CubeProgrammer tool allows the user to transfer data from a device to a remote device through a Bluetooth® Low Energy (BLE) connection.

The STM32CubeProgrammer tool is communicating with a BLE using an STM32WB dongle or a Nucleo board configured in HCI transparent mode as shown in *Figure 23*.

The implementation example does not include security in the transfer process. It is expected that users change their loader, or the application to perform the security verification based on the customer requirements.

It is not possible to load the BLE stack.

Figure 23. OTA update of STM32WB firmware through BLE connection



The target board to be programmed need to be running the OTA loader. When the user application is running in normal mode, the OTA loader is not active, and the OTA service is not available. To perform OTA transfer the application needs to reboot the device in OTA mode.

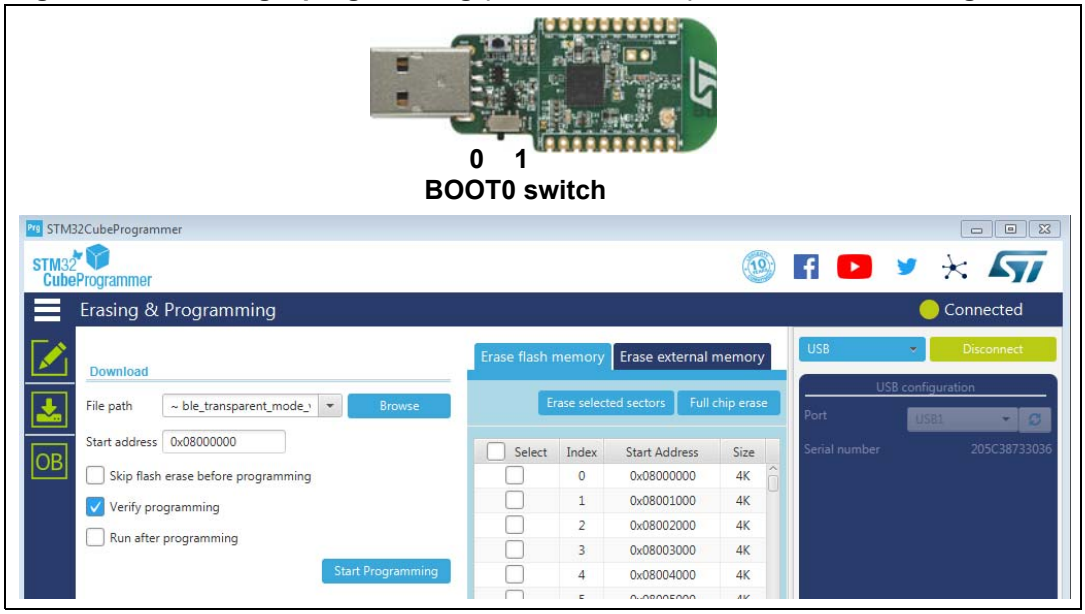
2.6.1 USB dongle configuration

The USB dongle provided with the Nucleo STM32WB package is configured in “transparent mode” for this purpose. Open and compile, within STM32Cube_FW_WB package available on *www.st.com*, the project ~\Projects\NUCLEO-WB55.USBDongle\Applications\BLE\ble_transparent_mode_vcp.

The USB dongle can be easily programmed in USB-DFU mode with STM32CubeProgrammer (version 2.0 and above) tool using USB DFU mode.

To access DFU mode, move the BOOT0 switch to 1 (to the right in *Figure 24*). Once programmed, move back the BOOT0 switch to 0 (to the left in *Figure 24*).

Figure 24. USB dongle programming (USB DFU mode) with STM32CubeProgrammer



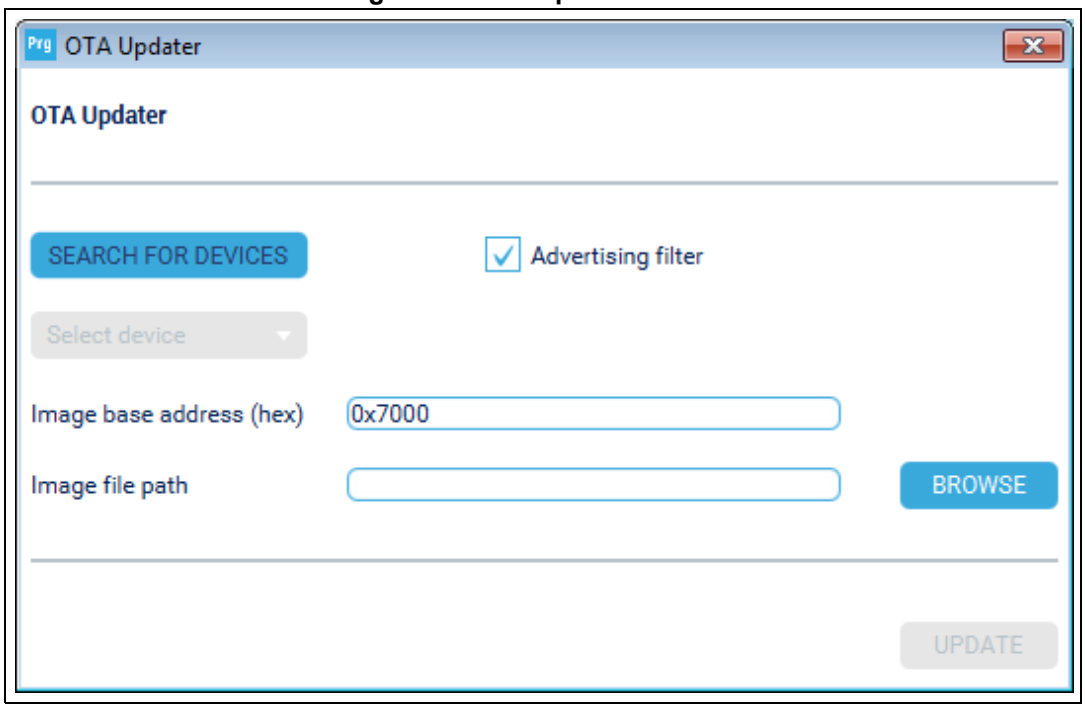
For more information on how to configure the source board in HCI transparent mode and the target board in OTA loader mode, check AN5247, available on *www.st.com*.

2.6.2 OTA update procedure

The OTA function is available in the target interface combo box in the configuration panel. Select the OTA interface and click the connect button.

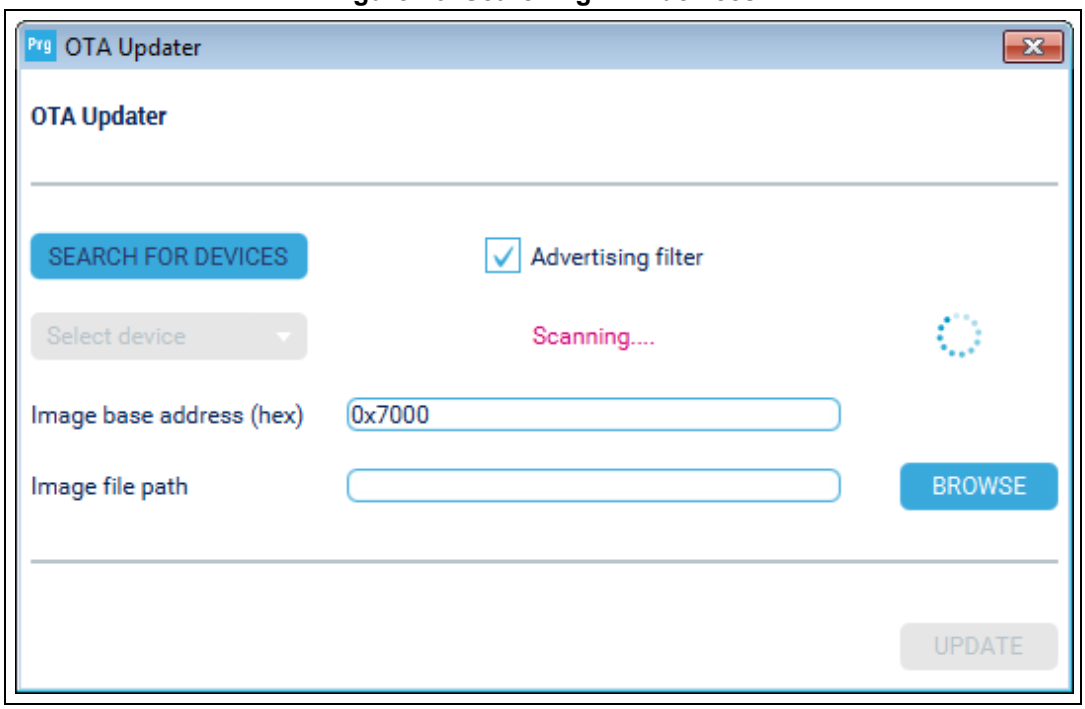
If the connected STM32WB board is configured in HCI transparent mode, the OTA updater window is displayed.

Figure 25. OTA updater window



The first operation is to find the target device. The tool needs to perform a scan of BLE devices and list all those with OTA capabilities.

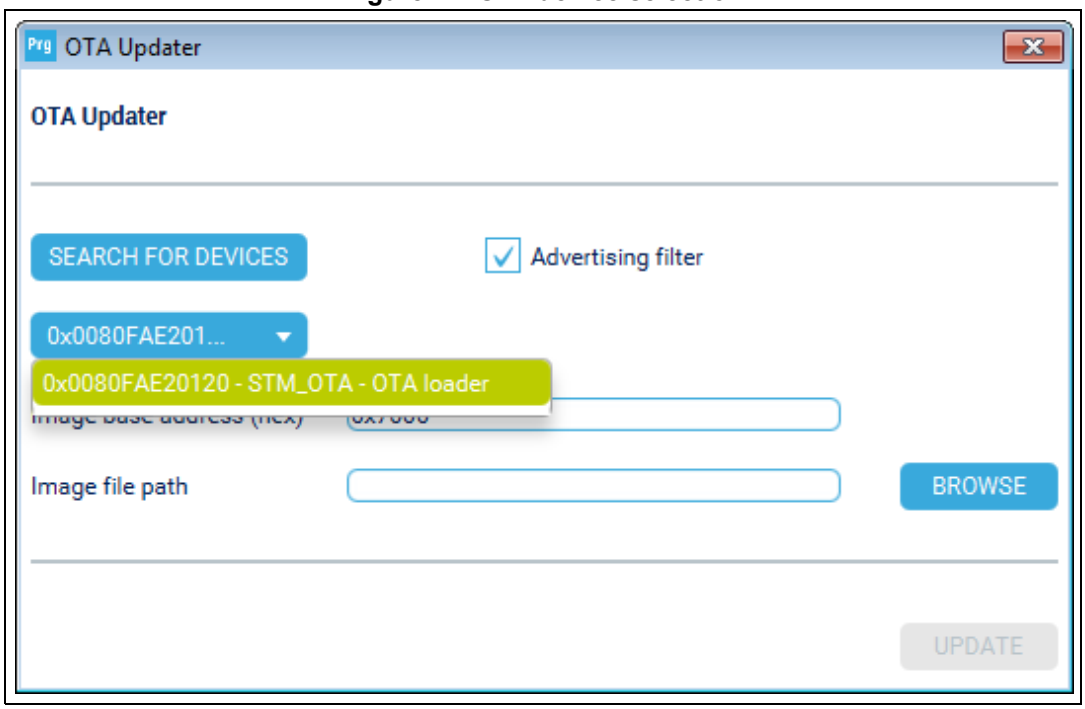
Figure 26. Searching BLE devices



The tool provides an advertising filter to refine the search procedure with advertising message.

To start search, click on the ‘SEARCH FOR DEVICES’ button. If Advertising filter is not checked, all BLE devices are listed, even if not compatible for OTA. It is recommended to check the Advertising filter option to list only devices with ST OTA information.

Figure 27. OTA device selection



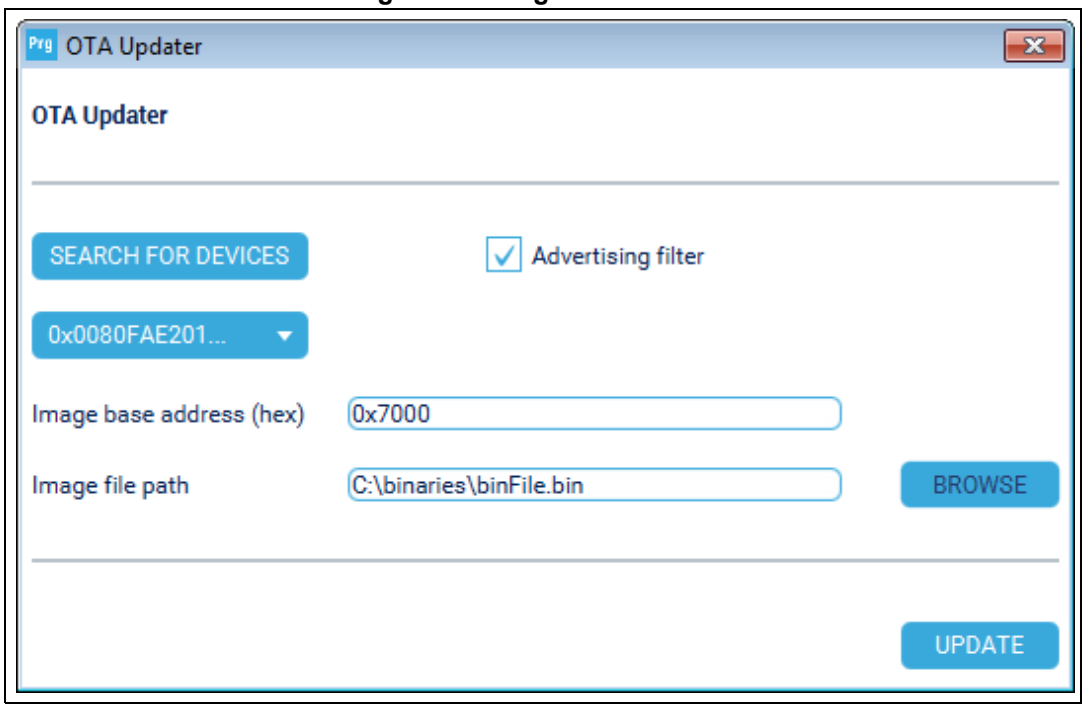
The picklist displays the list of detected boards:

- for devices with BLE characteristic: BLE address - Device name - OTA enabled
- for devices already in OTA mode: BLE address - Device name - OTA loader

The image base address is the place where the binary file must be stored on the target device. It is an hexadecimal value, and must be a multiple of 0x1000 to match with Flash memory sector.

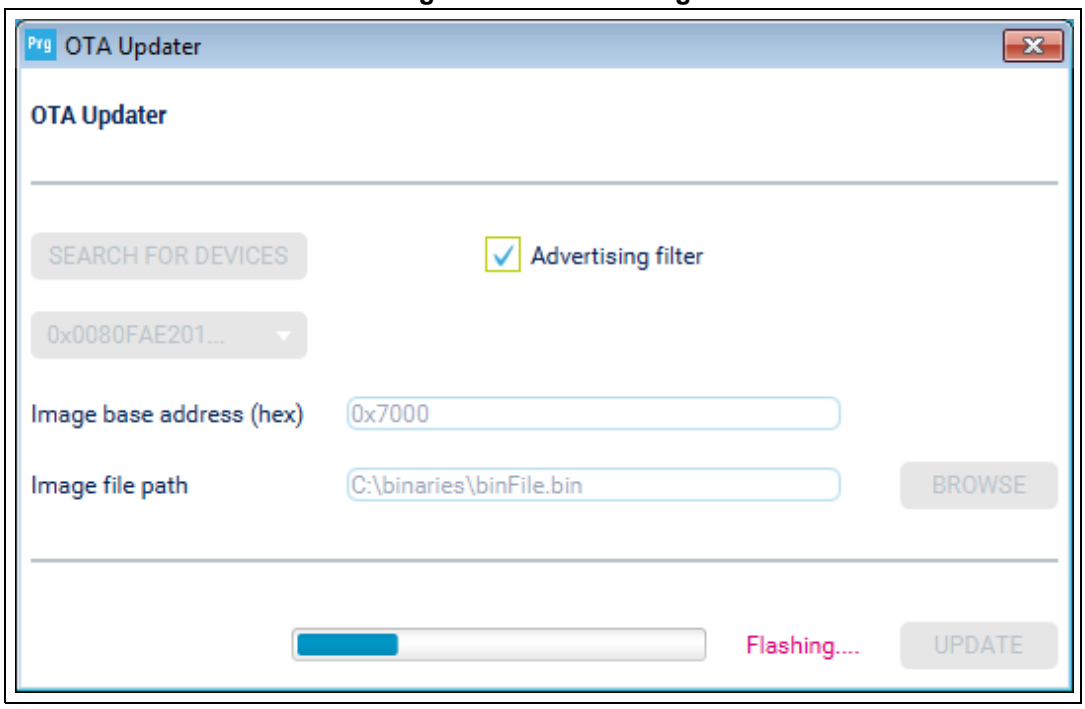
After selecting a device, click on “BROWSE” to select the binary file you want to program. Once selected the right path to the binary file click on “UPDATE”.

Figure 28. Image file selection



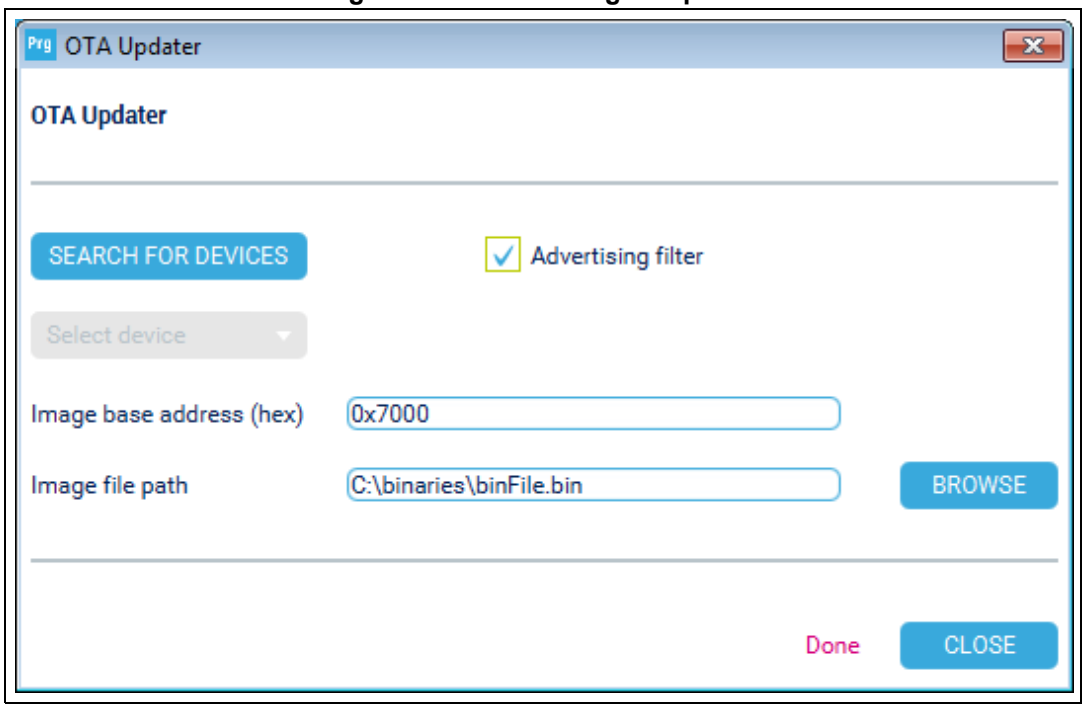
The progress of Flash memory programming (flashing) is indicated by a bar.

Figure 29. OTA flashing



Once the flashing is done, the target device reboots and starts the loaded application.

Figure 30. OTA flashing completed



For this feature STM32CubeMon-RF tool is called by STM32CubeProgrammer to perform OTA.

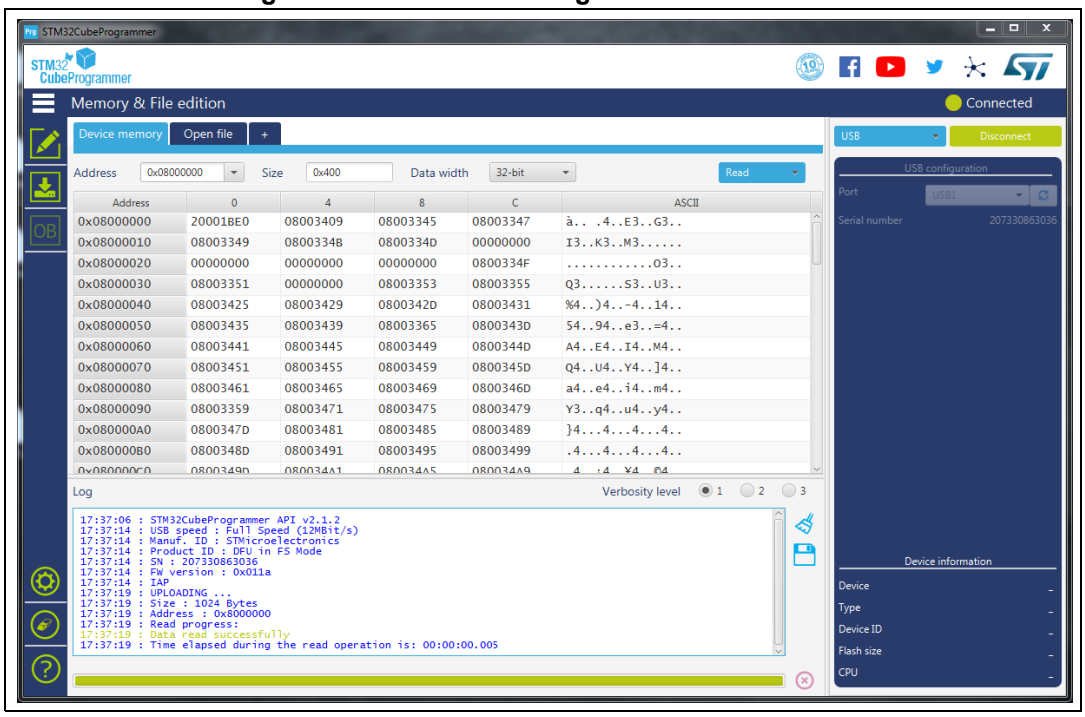
2.7 In application programming (IAP)

STM32CubeProgrammer supports IAP only with USB DFU connection mode. When USB connection is chosen and the boot is from Flash memory, STM32CubeProgrammer detects the IAP like DFU bootloader and after connection an IAP message appears in the log panel.

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

Sample IAPs are available in CubeFW on *www.st.com*.

Figure 31. STM32Cube Programmer in IAP mode



2.8 Flash the coprocessor binary using graphical interface

2.8.1 Using JTAG

1. Use STM32CubeProgrammer (version 2.4 or higher), see *Figure 32*
2. Access the SWD interface, see *Figure 33*
3. Delete the current wireless stack, see *Figure 34*
4. Upgrade the FUS version the same way you would download the stack when there is not an updated FUS version
5. Download the new FUS
6. Download the new wireless stack (pop-up must appear to ensure successful upgrade), see *Figure 35*

Figure 32. STM32CubeProgrammer API SWD connection

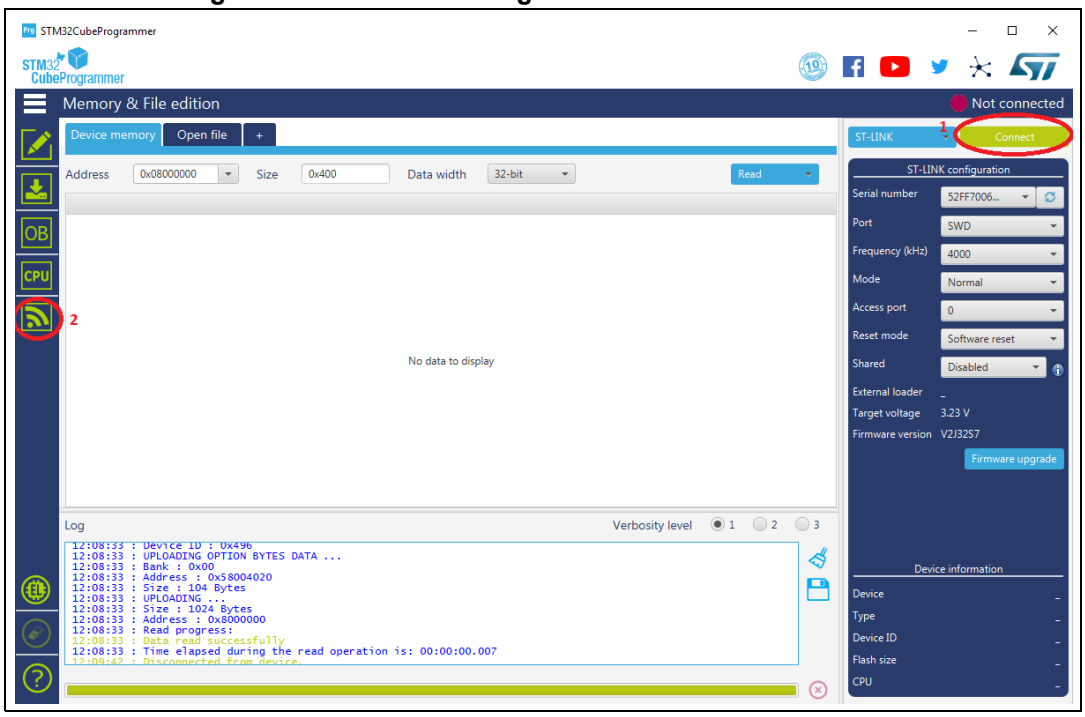


Figure 33. Steps for firmware upgrade

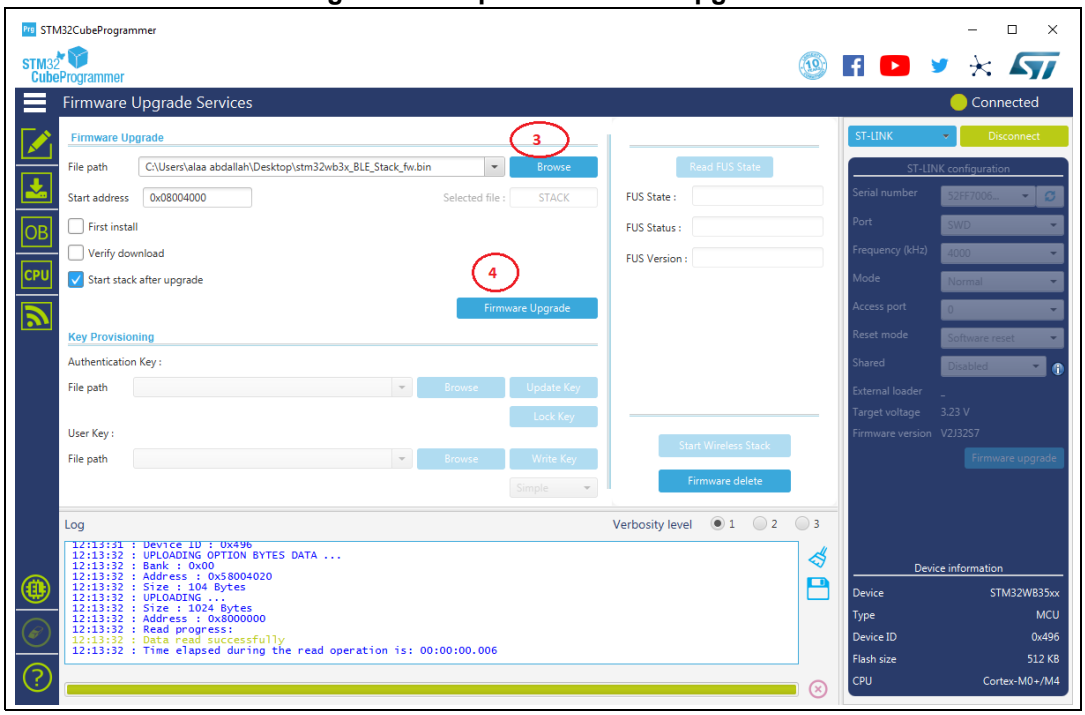


Figure 34. Pop-up confirming successful firmware delete

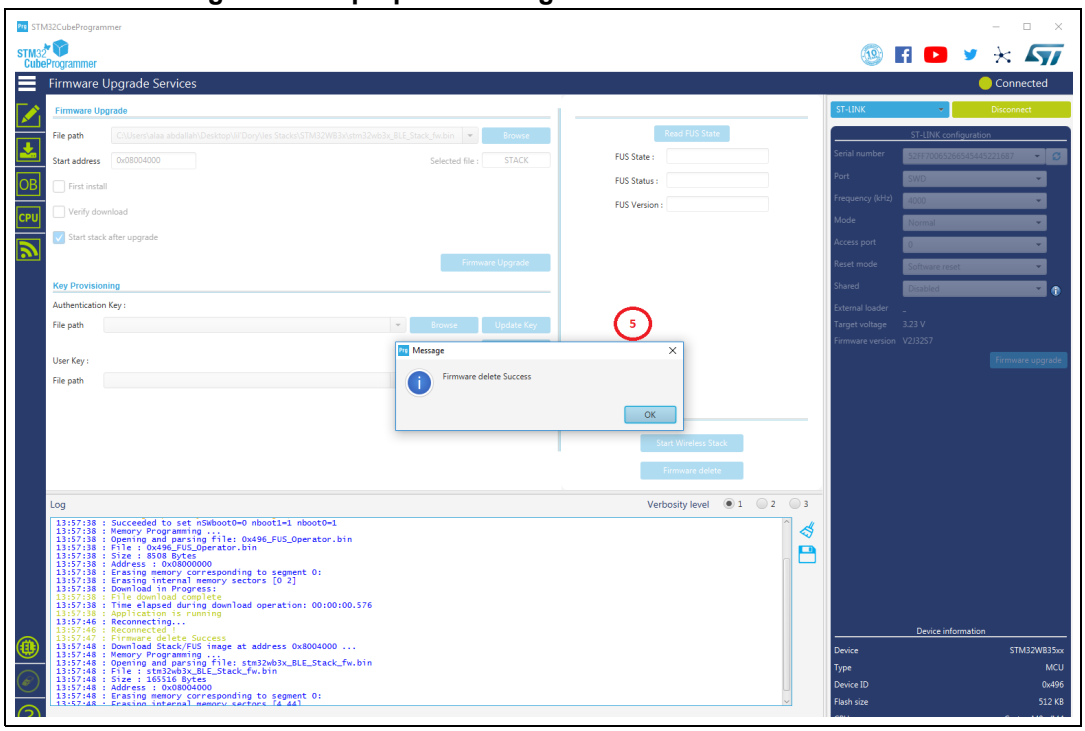
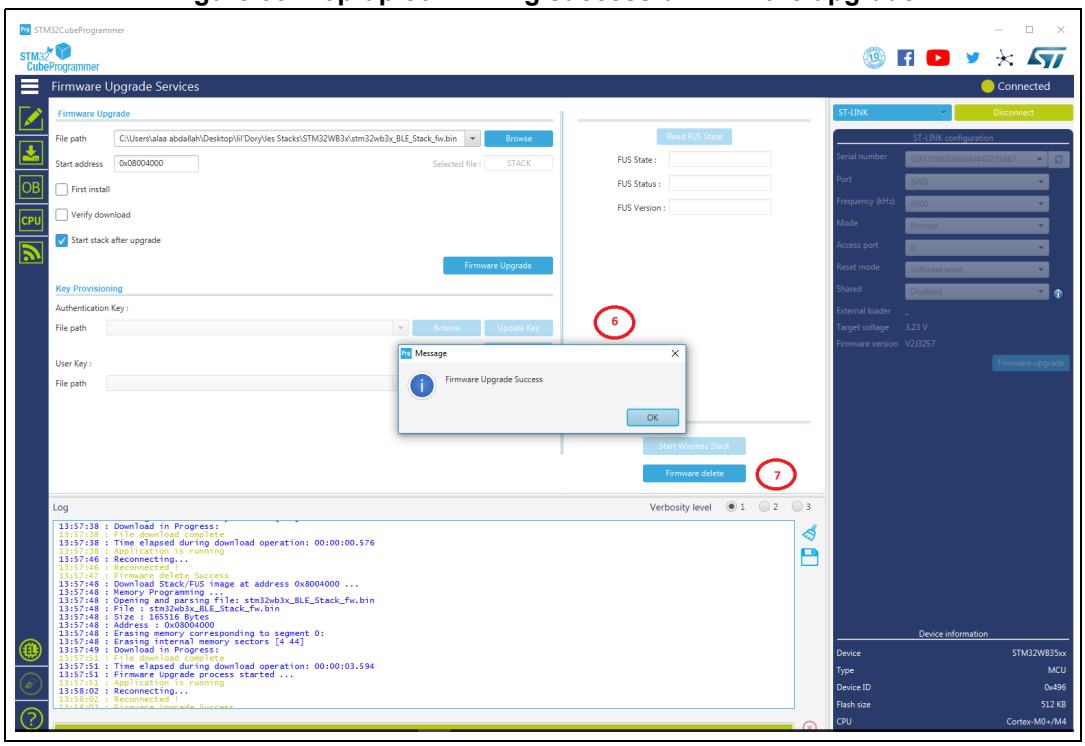


Figure 35. Pop-up confirming successful firmware upgrade

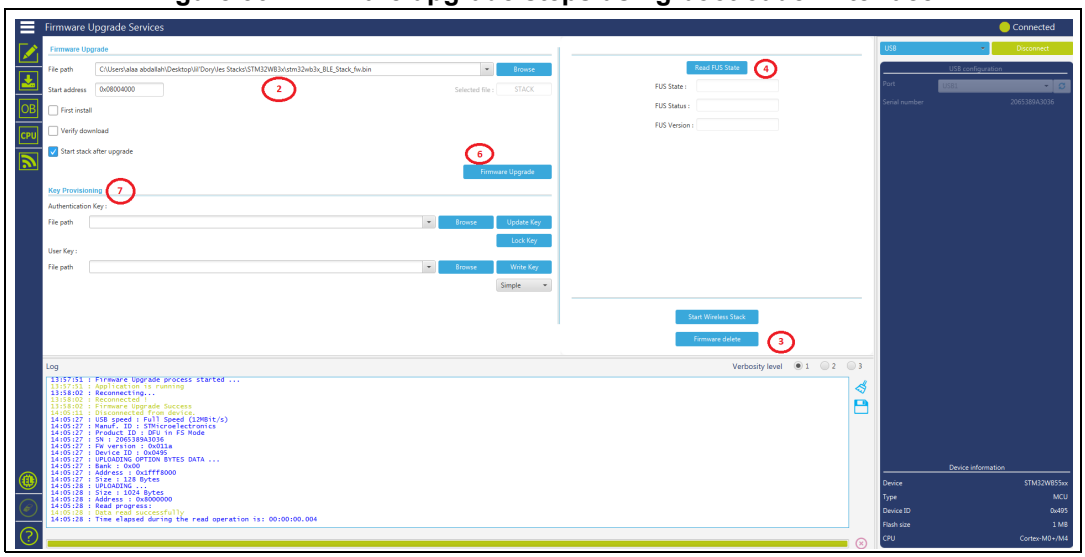


2.8.2 Using bootloader

1. Use STM32 CubeProgrammer (version 2.4.0 or higher)
2. Access the bootloader USB interface (system Flash memory)
3. Delete the current wireless stack
4. Read and upgrade the FUS version
5. Download the new FUS the same way you would download the stack when there is not an updated FUS version
6. Download the new wireless stack

Note: Provisioning section is only for users who already have a key to be implemented.

Figure 36. Firmware upgrade steps using bootloader interface



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

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 37*.

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

Figure 37. 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
    [rtr=<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
    [fb=<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 100
    [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 usually active.

- ST-LINK options

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

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

`[index=<index>]`	Index of the debug probe. Default index value is 0.
`[sn=<serialNumber>]`	Serial number of the debug probe. Use this option if you need to connect to a specific ST-LINK probe of which you know the serial number. Do not use this option with Index option in the same connect command.
`[mode=<mode>]`	Connection mode. Value in {NORMAL/UR/HOTPLUG}. Default value is NORMAL.
`Normal`	With ‘Normal’ connection mode, the target is reset then halted. The type of reset is selected using the ‘Reset Mode’ option.
`UR`	The ‘Connect Under Reset’ mode enables connection to the target using a reset vector catch before executing any instructions. This is useful in many cases, for example when the target contains a code that disables the JTAG/SWD pins.
`HOTPLUG`	The ‘Hot Plug’ mode enables connection to the target without a halt or reset. This is useful for updating the RAM addresses or the IP registers while the application is running.
`[ap=<accessPort>]`	Access port index. Default access port value is 0.

[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.

Note: Shared mode is supported only on Windows.

- USB options

The connection under the DFU interface does not support any option, knowing that defaults parameters are already included.

- SPI options

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

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

[cpha=<cpha_val>]	1Edge or 2Edge, default 1Edge
[cpol=<cpol_val>]	Low or high, default low
[crc=<crc_val>]	Enable or disable (0/1), default 0
[crcpol=<crc_pol>]	CRC polynomial value
[datasize=<size>]	8-bit 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.
[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.

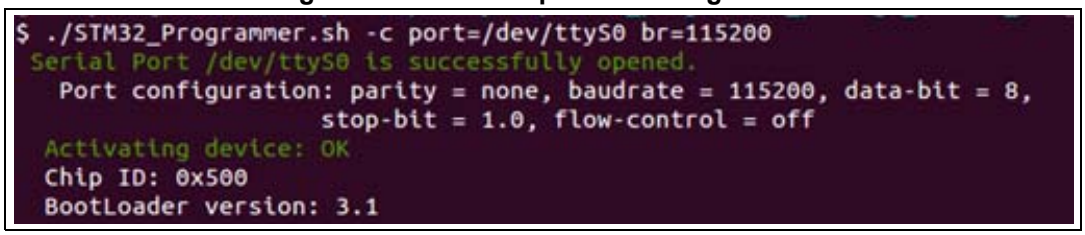
[ide=<type>]	Type: STANDARD or EXTENDED, default STANDARD
[rtr=<format>]	Frame format: DATA or REMOTE, default DATA
[fifo=<afifo>]	Assigned FIFO: FIFO0 or FIFO1, default FIFO0
[fm=<fmode]	Filter mode: MASK or LIST, default MASK
[fs=<fscale>]	Filter scale: 16 or 32, default 32
[fe=<fenable>]	Activation: 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 38*.

Figure 38. Connect operation using RS232



Example using USB

./STM32_Programmer.sh -c port=usb1

The result of this example is shown in *Figure 39*.

Figure 39. Connect operation using USB

```
establishing connection with the target device


USB speed             : FULL_SPEED(12MBit/s)
Manufacturer ID       : STMicroelectronics
Product ID            : STM32  BOOTLOADER
Serial number         : 326F37603234
Firmware version      : 1.1a
Device ID             : 0x0419

   AREA NAME          SECT.NBR        ADDRESS         SIZE          TYPE


   Internal Flash       0000          0x08000000      0016 KB       REW
                        0001          0x08004000      0016 KB       REW
                        0002          0x08008000      0016 KB       REW
                        0003          0x0800c000      0016 KB       REW
                        0004          0x08010000      0064 KB       REW
                        0005          0x08020000      0128 KB       REW
                        0006          0x08040000      0128 KB       REW
                        0007          0x08060000      0128 KB       REW
                        0008          0x08080000      0128 KB       REW
                        0009          0x080a0000      0128 KB       REW
                        0010          0x080c0000      0128 KB       REW
                        0011          0x080e0000      0128 KB       REW
                        0012          0x08100000      0016 KB       REW
                        0013          0x08104000      0016 KB       REW
                        0014          0x08108000      0016 KB       REW
                        0015          0x0810c000      0016 KB       REW
                        0016          0x08110000      0064 KB       REW
                        0017          0x08120000      0128 KB       REW
                        0018          0x08140000      0128 KB       REW
                        0019          0x08160000      0128 KB       REW
                        0020          0x08180000      0128 KB       REW
                        0021          0x081a0000      0128 KB       REW
                        0022          0x081c0000      0128 KB       REW
                        0023          0x081e0000      0128 KB       REW

   Option Bytes         0000          0x1fffc000      0016 B        RW
                        0001          0x1ffec000      0016 B        RW

   OTP Memory           0000          0x1fff7800      0512 B        RW
                        0001          0x1fff7a00      0016 B        RW

   Device Feature       0000          0xffff0000      0004 B        RW
```

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

Example using JTAG/SWD debug port

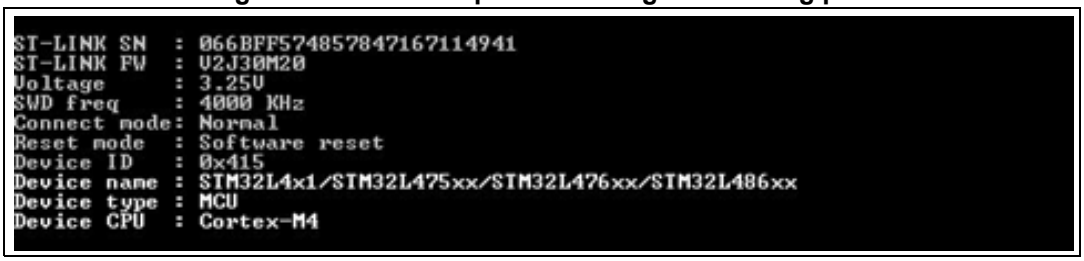
To connect using port connection mode with ST-LINK probe it is necessary to mention the port name with the connect command at least (for example: `-c port=JTAG`).

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 40. 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 41*.

Figure 41. 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 42*.

Figure 42. 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 on the speed mode parameter <sm>, for example, if sm=standard then the baudrate does not support the value 400.

The result of this example is shown in *Figure 43*.

Figure 43. 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).

3.2.2 Erase command

-e, --erase

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

Syntax: `-e [all] [sectorsCodes]`

`[all]`	: Erase the entire Flash memory.
`[sectorsCodes]`	: Erase only the specified sectors.
`[<[start end]>]`	: Erase all sectors starting from “start” to “end” code.

Example

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

This command erases only sectors 2 and 4.

3.2.3 Download command

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

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

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

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

`[start_address]` Start address of download

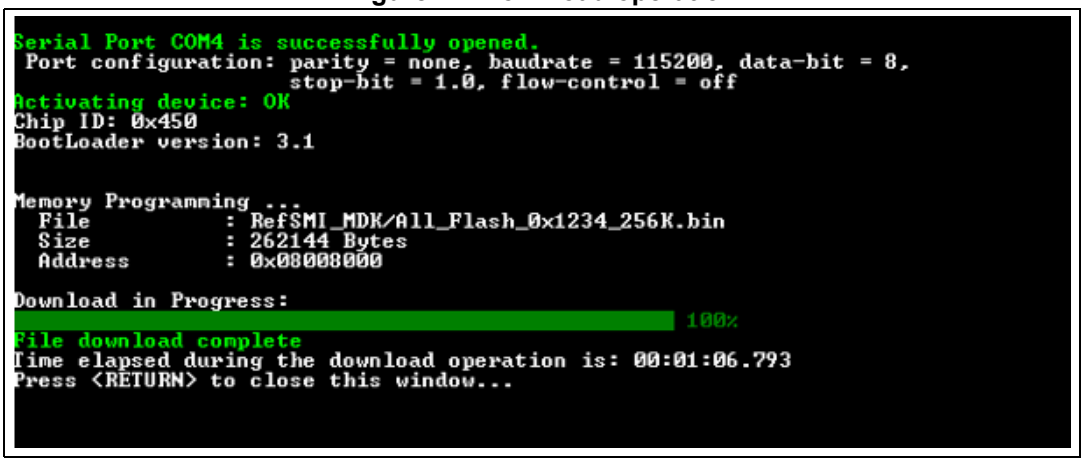
Example

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

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

The result of this example is shown in *Figure 44*.

Figure 44. Download operation



Note: To verify that the download was 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 the 32 data bits (0x12345678, 0xAABBCCFF, 0x12AB34CD) to be written into the Flash memory starting from address 0x08000000.

3.2.5 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.

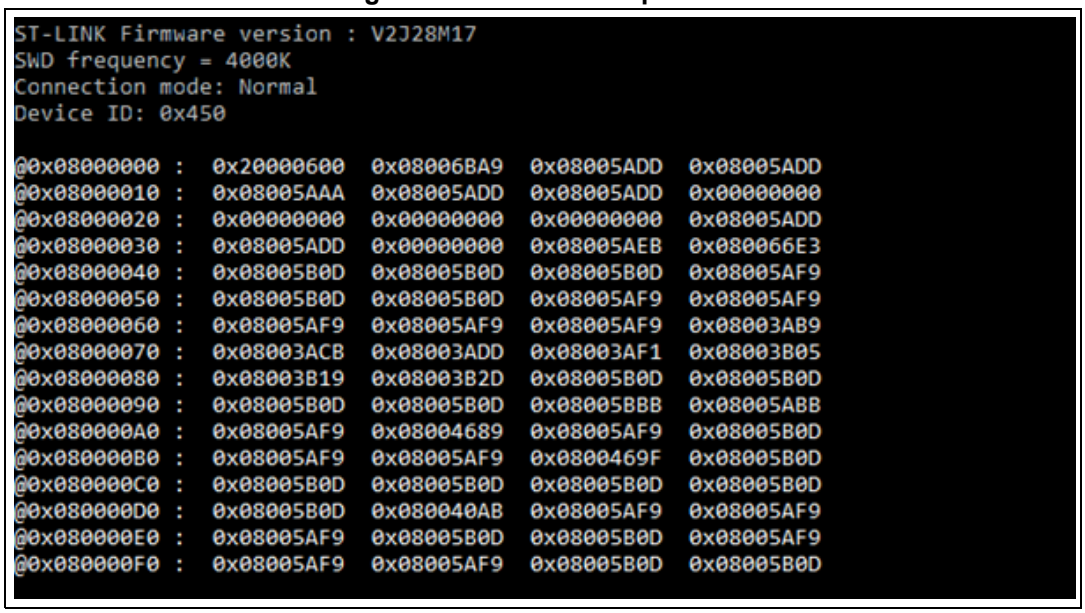
Syntax: -r32 <start_address> <size>

`<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 45. Read 32-bit operation



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

3.2.6 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.

3.2.7 Debug commands

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

-rst

Description: Executes a software system reset;

Syntax: `-rst`

-hardRst

Description: Generates a hardware reset through the RESET pin in the debug connector.

The RESET pin of the JTAG connector (pin 15) must be connected to the device reset pin.

Syntax: `-hardRst`

-halt

Description: Halts the core.

Syntax: `-halt`

-step

Description: Executes one instruction.

Syntax: `-step`

-score

Description: Displays the Cortex-M core status.

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

Syntax: `-score`

-coreReg

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

Syntax: `-coreReg [<core_register>]`
`R0/../R15/PC/LR/PSP/MSP/XPSR/APSR/IPSR/EPSR/PRIMASK/BASEPRI/FAULTMASK/CONTROL`

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

Example

```
-coreReg
```

This command displays the current values of the core registers.

```
-coreReg R0 R8
```

This command displays the current values of R0 and R8.

```
-coreReg R0=5 R8=10
```

This command modifies the values of R0 and R8.

3.2.8 List command

-l, -list

Description: This command lists all available RS232 serial ports.

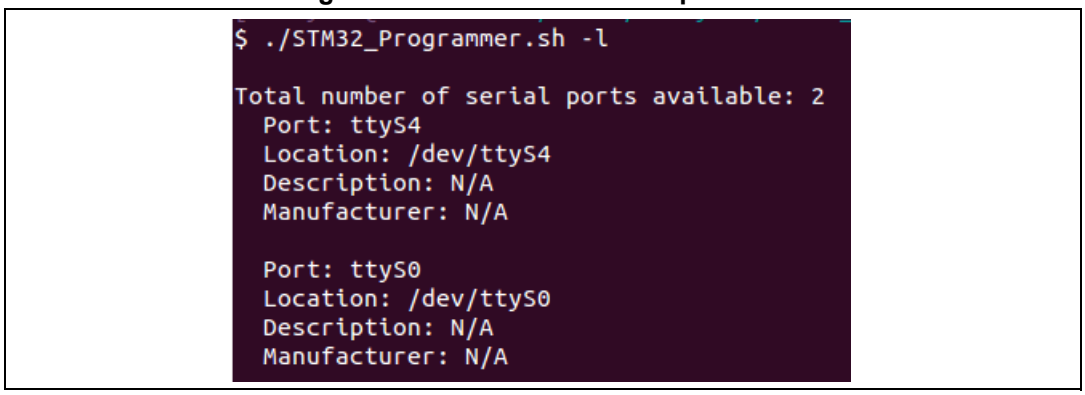
Syntax: `-l, --list`

Example

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

The result of this example is shown in *Figure 46*.

Figure 46. The available serial ports list



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

3.2.9 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
```

3.2.10 Verbosity command

-vb, --verbosity

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

Syntax: -vb <level>

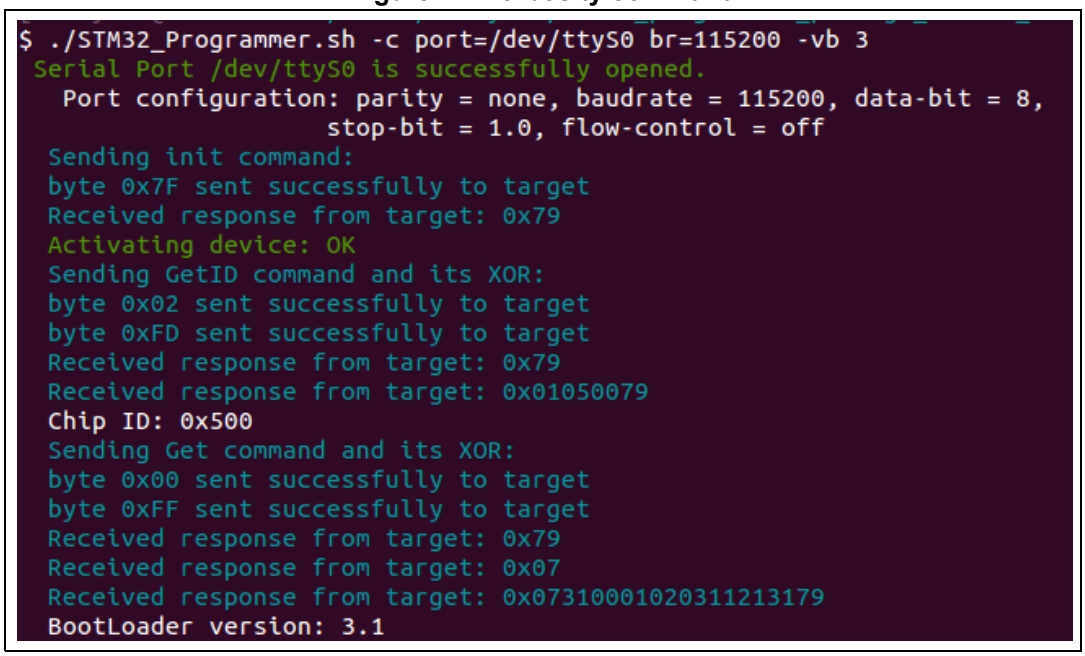
`<level>` : Verbosity level, value in {1, 2, 3} default value vb=1

Example

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

The result of this example is shown in *Figure 47*.

Figure 47. Verbosity command



3.2.11 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]`

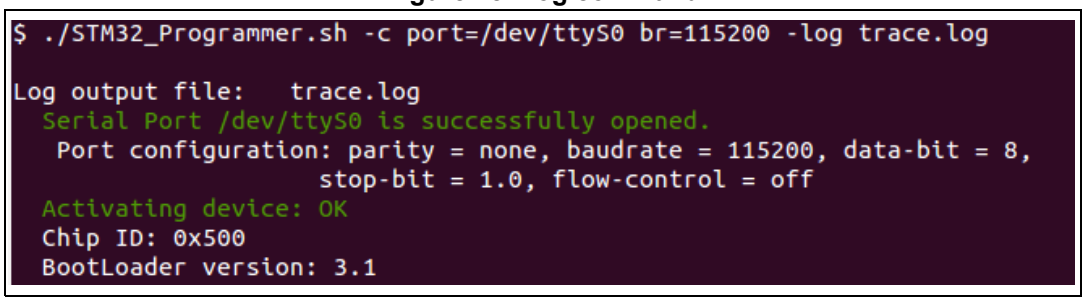
`[filePath.log]` Path of log file, default is $HOME/.STM32CubeProgrammer/trace.log.

Example

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

The result of this example is shown in *Figure 48*.

Figure 48. Log command



The log file trace.log contains verbose messages such as those shown in *Figure 49*.

Figure 49. 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
```

3.2.12 External loader command

-el

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

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

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"
```

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

3.2.13 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
```

3.2.14 TZ regression command

-tzenreg, --tzenregression

Description: This command removes TrustZone protection by disabling the 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.

3.2.15 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.

3.2.16 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 are: 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 50*.

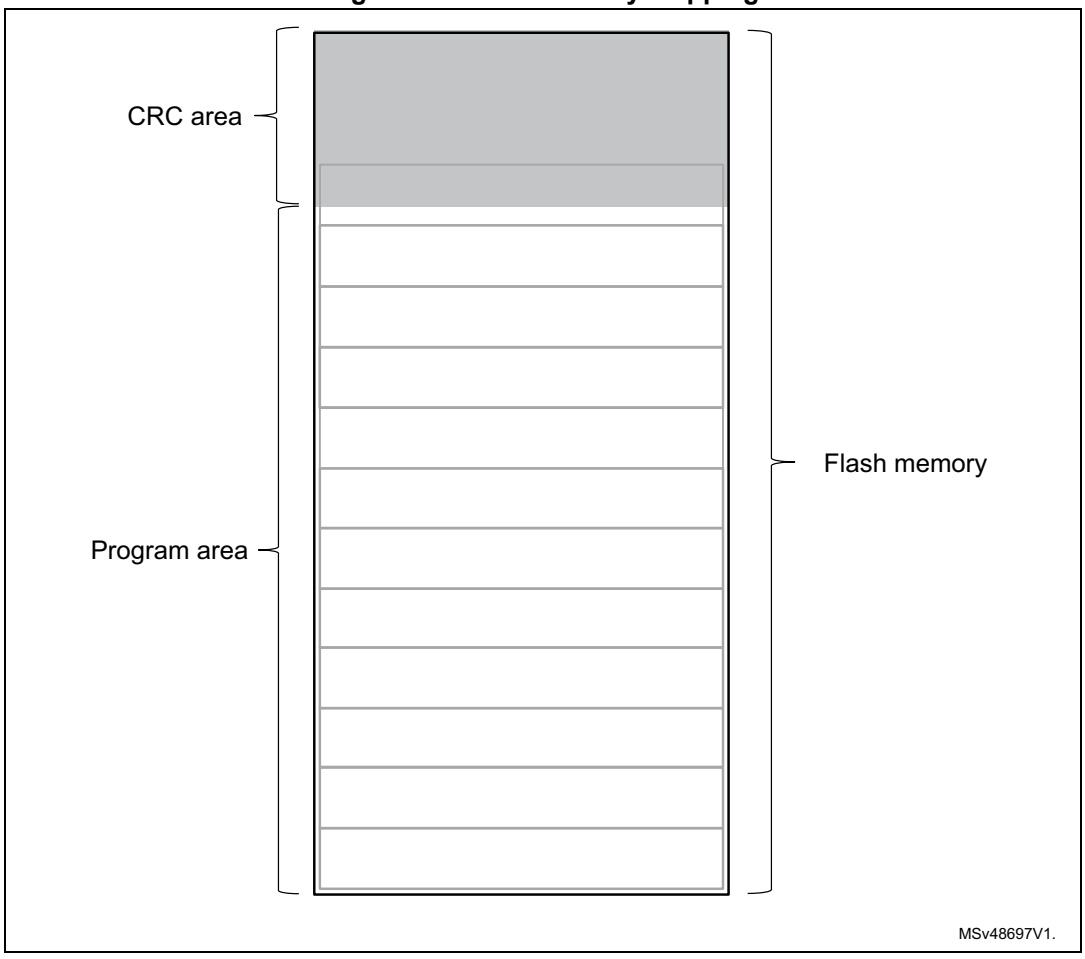
Figure 50. Safety lib command

```
C:\bin>STM32_Programmer_CLI.exe -sl TestCRC.axf 0x8000000 0x8010000 0x400
      -------------------------------------------------------------------
                        STM32CubeProgrammer v0.4.0-RC1
      -------------------------------------------------------------------

Warning: The ELF file will be overwritten
CRCs area injected succesfully
```

The Flash program memory is divided into slices, whose size is given as a parameter to the safety lib command as shown in the example above. For each slice a CRC value is computed and placed in the CRC area. The CRC area is placed at the end of the memory, as shown in *Figure 51*.

Figure 51. Flash memory mapping



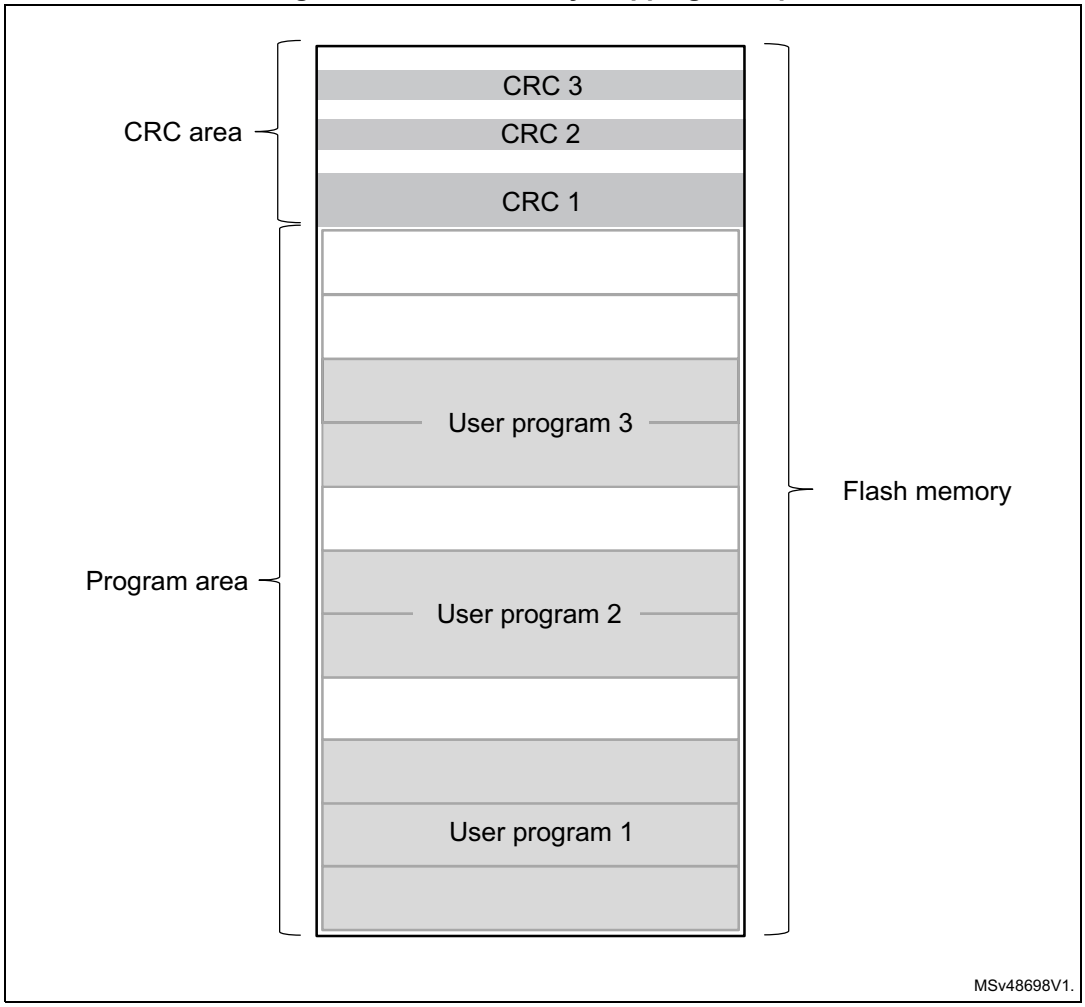
The address and size of the CRCs area are determined as follows:

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 52*.

Figure 52. Flash memory mapping example



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

$$@ = \text{CRCs\_Start\_Address} + \left(\frac{\text{UserProg\_Start\_Address} - \text{Flash\_Start\_Address}}{\text{Slice\_Size}} \cdot 4 \text{ bytes} \right)$$

3.2.17 Secure programming SFI specific commands

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

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

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

-sfi, --sfi

Description: Programs an sfi file

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

`[<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 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`

3.2.18 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 *Secure programming using STM32CubeProgrammer* (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.

-el --extload	Selects a custom external memory-loader
`<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 since some initializations are already done by RSS, and it is marked with –SFIX at the end of the External FlashLoader name.

3.2.19 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 *Secure programming using STM32CubeProgrammer* (AN5054).

-hsmgetinfo

Description: Reads the H,SM available information

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

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

-hsmgetcounter

Description: Reads the current value of the license counter

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

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

-hsmgetfwid

Description: Reads the Firmware/Module identifier

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

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

-hsmgetstatus

Description: Reads the current card life-cycle state

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

`[slot=<SlotID>]` :Slot ID of the Smart Card Reader
Default value: slot=1 (the PC integrated SC reader)

-hsmgetlicense

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

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

`<file_path>` : File path into which 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

3.2.20 STM32WB specific commands

-getuid64

Description: Read the device unique identificator (UID))

Syntax: `-getuid64`

-fusgetstate

Description: Read the FUS state

Syntax: `-fusgetstate`

-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 if it is a first install, otherwise 0
optional, Default value: `firstinstall=0`

`[-v]` : Verify if the download operation is achieved successfully before starting 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's no longer possible to change it using `-authkeyupdate` command

Syntax: `-authkeylock`

-wusrkey

Description: Customer key storage

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

<file.path>: customer key in binary format

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

Note: These commands are available only through USB DFU and UART bootloader interfaces, except for the "-fwdelete" and the "-fwupgrade" commands, available through USB DFU, UART and SWD interfaces.

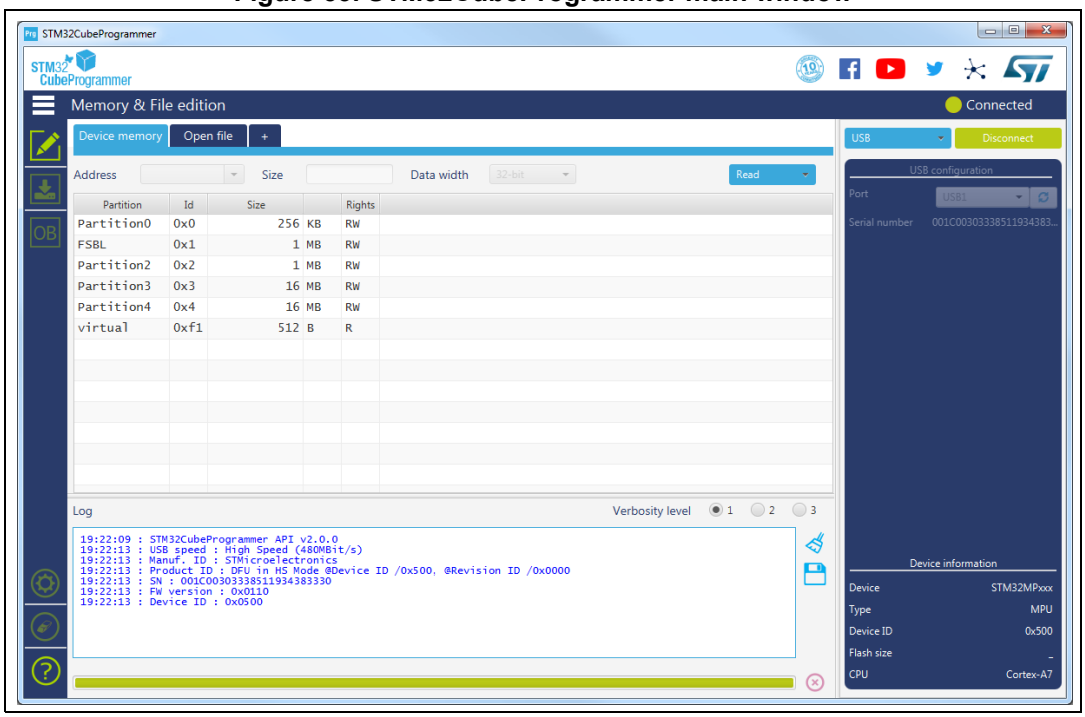
Usage example for SWD interface

- 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

4 STM32CubeProgrammer user interface for MPUs

4.1 Main window

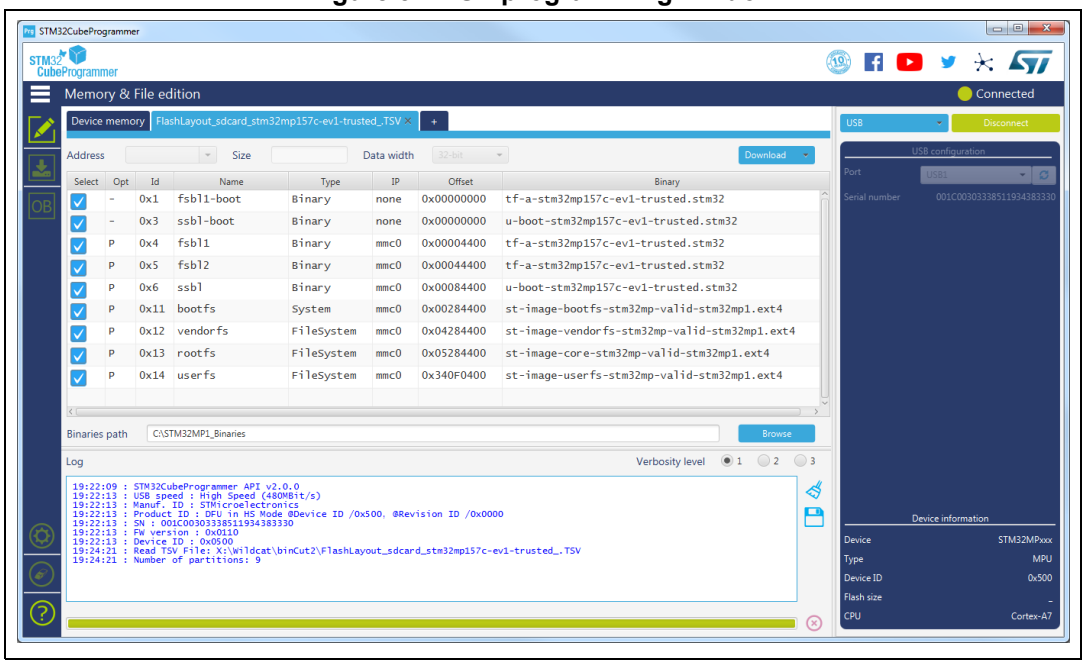
Figure 53. STM32CubeProgrammer main window



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

4.2 Programming windows

Figure 54. TSV programming window



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

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

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

5 STM32CubeProgrammer CLI for MPUs

5.1 Command line usage

The following sections describe how to use the STM32CubeProgrammer for MPU from the command line.

Figure 55. Available commands for MPUs

```
Available commands for STM32 MPU:

-c,      --connect       : Establish connection to the device
    <port=<PortName>    : Interface identifer. ex COM1, /dev/ttyS0, usb1>
   UART port optional parameters:
    [br=<baudrate>]     : Baudrate. ex: 115200, 9600, etc, default 115200
    [P=<parity>]        : Parity bit, value in <NONE,ODD,EVEN>, default NONE
    [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 <Not yet supported for STM32MP)
                          Value in <OFF,Hardware,Software> ..., default OFF
    [noinit=noinit_bit]: Set No Init bits, value in <0,1> ..., default 0
-s,      --start
-g,      --go            : Run the code at the specified partition ID.
    [<partitionID>]     : Partition ID
                          If not specified, last loaded partition
                          will be started

-otp program            : This command allows the user to program SAFMEM
                          memory by modifying the OTP words
     [wordID=<value>]   : This field contains the OTP word number
     [value=<value>]    : Loads value into the chosen OTP word
     [sha_rsl=<value>]  : Loads value into the corresponding shadow read
                          sticky lock bit
     [sha_wsl=<value>]  : Loads value into the corresponding shadow write
                          sticky lock bit
     [sl=<value>]       : Loads value into the corresponding programming sticky
                          lock bit
     [pl=<value>]       : Loads value into the corresponding programming perma-
                          nent lock bit

-otp displ              : This command allows the user to display all or parts
                          of the OTP structure
     [upper]            : Displays the loaded upper OTP shadow registers
                          values and status
     [lower]            : Displays the loaded lower OTP shadow registers
                          values and status
     [ctrl]             : Displays the loaded BSEC control registers

-detach                 : Send detach command to DFU

-wb                     : Write blob

    <blob_file_path>    : Blob file path
-pmic                   : Program PMIC NVM

    <PMIC file_path>    : PMIC file_path
-gc, --getcertificate        : Get the chip Certificate,
                               supported for devices with security features
    <file_path>              : Certificate file path into which the chip
                               certificate will be uploaded

-p,      --phaseID      : Display the current partition ID to be loaded

-w,      --write
-d,      --download     : Download the content of a file into device memory
    <file_path>         : File path name to be downloaded: <bin, stm32 file
    <partition_id>      : Partition ID to be downloaded
-rp,     --readPart     : Upload a memory partion ID content to a .bin file
    <partionID>         : Partion to be read
    [<offset address>] : Offset address of read and upload
    <size>              : Size of partion content to be read
    <file_path>         : Binary file path
```

5.2 Available commands for STM32MP1

This section details the commands supported on STM32MP1 devices.

5.2.1 Connect command

-c, --connect

Description: Establish 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>]` : Set No Init bits, value in {0,1}, default 0.

Noinit=1 can be used if a previous connection is usually active (no need to send 0X7F)

`[br=<baudrate>]` : Baudrate, (e.g. 9600, 115200), default 115200.

`[P=<Parity>]` : Parity bit, value in (EVEN, NONE, ODD), default EVEN.

`[db=<data_bit>]` : Data bit, value in (6, 7, 8), default 8.

`[sb=<stop_bit>]` : Stop bit, value in (1, 1.5, 2), default 1.

`[fc=<flowControl>]` : Flow control, value in (OFF, Software, Hardware), Software and Hardware flow control are not yet supported for STM32MP1, default OFF.

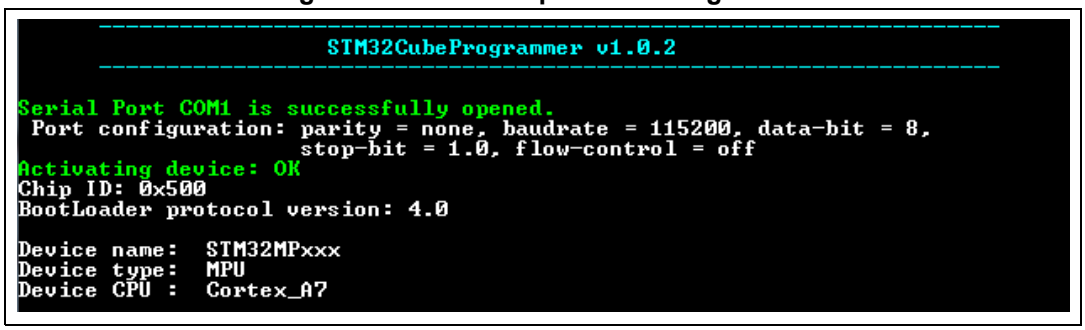
Example

Using UART:

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

The result of this example is shown *Figure 56*.

Figure 56. 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.

5.2.2 GetPhase command

-p, --phaseID

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

Syntax: `--phaseID`

Example

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

5.2.3 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 files extensions)

`[partition_ID]` Partition ID to be downloaded.

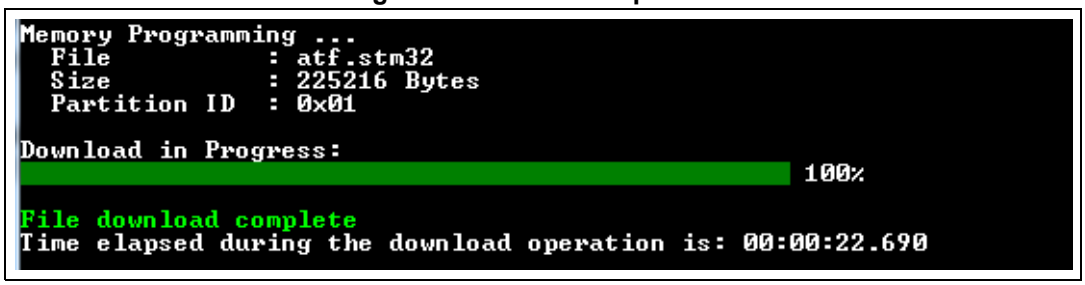
Example

```
./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 57*.

Figure 57. Download operation



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

Example: Download and manifestation on alternate 0x1

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

5.2.4 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 58. 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>).

5.2.5 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
```

5.2.6 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”

5.2.7 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
```

5.2.8 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
```

5.2.9 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
```

5.2.10 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 59*).

Figure 59. 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
```

5.2.11 OTP programming

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

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

- Loading shadow registers values to the tool:

  For load operation, the host requests the OTP partition data and the platform replies with the structure described on https://wiki.st.com/stm32mpu/index.php/STM32CubeProgrammer_OTP_management.
- Writing the modified shadow registers to the target:

  This operation is executed by performing the following sequence:

  a) The user types in the value and the status of each chosen OTP shadow register.

  b) The tool updates the OTP structure with the newly given OTP shadow registers values and status.

  c) The tool proceeds with sending the updated structure, with bit0 in the “Write/read conf” field set to 0 (“Write/read conf” is word number 7 in the OTP structure).

  d) Once the structure is sent, the shadow register values are reloaded to update the OTP structure in the tool.
- Programming the OTP with the modified shadow registers values:
  once the user updates the OTP values and the OTP structure is refreshed, the host sends the OTP structure with bit0 in the “Write/read conf” field set to 1 (“Write/read conf” is word number 7 in the OTP structure).
- Reloading the OTP value to the shadow registers:
  once the OTP words are successfully programmed, the host uploads the OTP structure in order to update the OTP shadow registers. This operation allows the host to verify the status of the last SAFMEM programming via bit4 in the “Status” field.
- BSEC control register programming:
  once the user updates the values of the given BSEC control register (Configuration, Debug configuration, Feature configuration and General lock configuration) the host updates the OTP structure and sends it to the device with bit0 in the “Write/read conf” field set to 0.
- OTP programming CLI:
  the user is given a set of commands to perform a chosen sequence of operations on the OTP partition. Each one of these commands is described below.

5.2.12 Programming SAFMEM command

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

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

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

`[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
```

5.2.13 Detach command

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

Syntax: -detach

5.2.14 GetCertif command

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

Syntax: `-gc certification.bin`

5.2.15 Write blob command

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

Syntax: `-wb blob.bin`

5.2.16 Display command

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

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

`[upper]` : This option allows the user to display the loaded upper OTP shadow registers' values and status.

`[lower]` : This option allows the user to display the loaded lower OTP shadow registers' values and status.

`[ctrl]` : This option allows the user to display the loaded BSEC control registers.

Example

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

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 1. 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 3.2.7: 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 7: ST-LINK configuration panel* – *Figure 37: STM32CubeProgrammer: available commands*. – *Figure 40: 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 5: STM32CubeProgrammer main window*, *Figure 6: Expanded main menu*, *Figure 7: ST-LINK configuration panel*, *Figure 8: UART configuration panel*, *Figure 9: USB configuration panel*, *Figure 10: Target information panel*, *Figure 11: SPI configuration panel*, *Figure 12: CAN configuration panel*, *Figure 13: I2C configuration panel*, *Figure 14: Memory and file edition: Device memory tab*, *Figure 16: Memory and file edition: File display*, *Figure 17: Flash memory programming and erasing (internal memory)* and *Figure 18: Flash memory programming and erasing (external memory)*. Minor text edits across the whole document.
03-Jan-2019	5	Updated *Section 1.2.4: DFU driver*. Added *Section 3.2.17: Secure programming SFI specific commands*, *Section 3.2.19: 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 1. 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 3.2.17: Secure programming SFI specific commands*, *Section 3.2.19: HSM related commands* and *Section 3.2.20: STM32WB specific commands*. Updated *Figure 17: Flash memory programming and erasing (internal memory)*.
11-Oct-2019	8	Updated *Graphical guide*, *Section 3.2.17: Secure programming SFI specific commands*, *Section 3.2.19: HSM related commands* and *Section 3.2.20: STM32WB specific commands*. Added *Section 2.7: In application programming (IAP)*. Minor text edits across the whole document.
08-Nov-2019	9	Updated *Section 1.2.1: Linux install*, *Section 3.2.20: STM32WB specific commands* and *Section 5.2.6: 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 3.2.17: Secure programming SFI specific commands*. Added *Section 3.2.14: TZ regression command* and *Section 3.2.18: 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 2.8: Flash the coprocessor binary using graphical interface* and its subsections.