UM2237
User manual

STM32CubeProgrammer software description

Introduction

STM32CubeProgrammer (STM32CubeProg) provides an all-in-one software tool for STM32 microcontroller programming 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 user manual details the hardware and software environment prerequisites, as well as the available STM32CubeProgrammer software features.

Contents

List of tables

List of figures

1 Getting started

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

STM32CubeProgrammer supports STM32 32-bit devices based on Arm®(a) Cortex®-M processors.

1.1 System requirements

Supported operating systems and architectures:

- Linux® 32-bit and 64-bit (tested on Ubuntu 14.04)
- Windows® 7/8/10 32-bit and 64-bit
- macOS® (minimum version OS X® Yosemite)

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

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 procedure for the use of the STM32CubeProgrammer software. The setup also offers optional installation of the 'STM32 trusted package creator' tool, which is used to create secure firmware files for secure firmware install and update. For more information, check user manual UM2238.

1.2.1 Linux install

If you are using a USB port to connect to the STM32 device, you need to install the libusb1.0 package by typing the following command in your machine's terminal:

sudo apt-get install libusb-1.0.0-dev

To use ST-LINK probe or USB DFU to connect to a target, you need to copy the rules files loacated 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.

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.

arm

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

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.

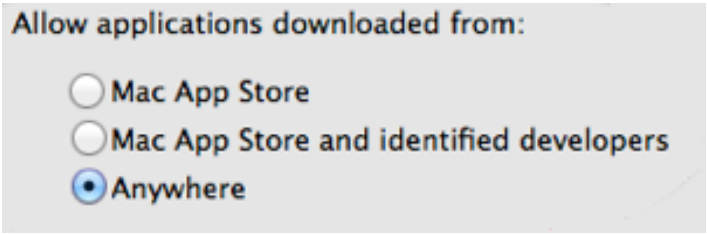
1.2.3 macOS install

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 be able to install STM32CubeProgrammer on MacOs, you need to 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.

1.2.4 DFU driver

If you are using the STM32 device in USB DFU mode, you need to 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.

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

Figure 2. Deleting the old driver software

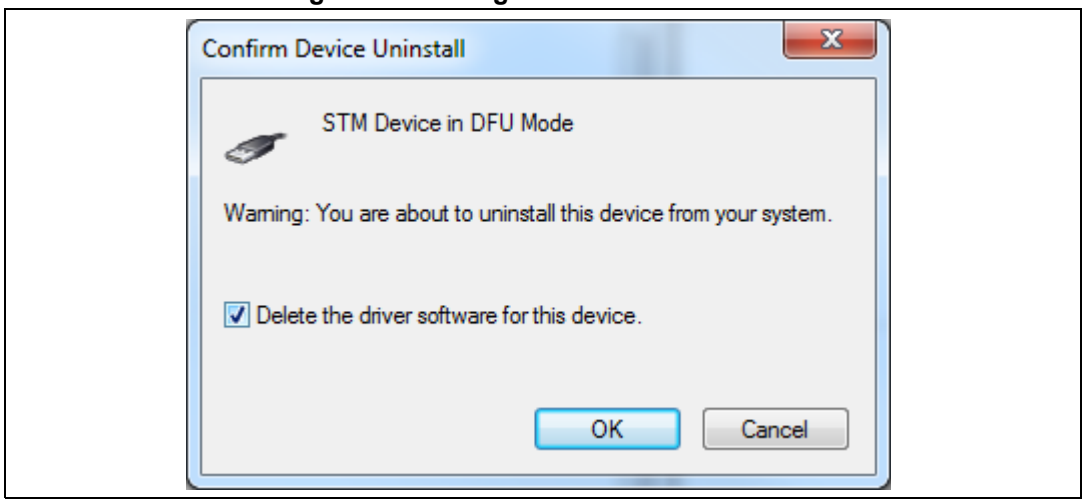


Figure 3. STM32 DFU device with DfuSe driver

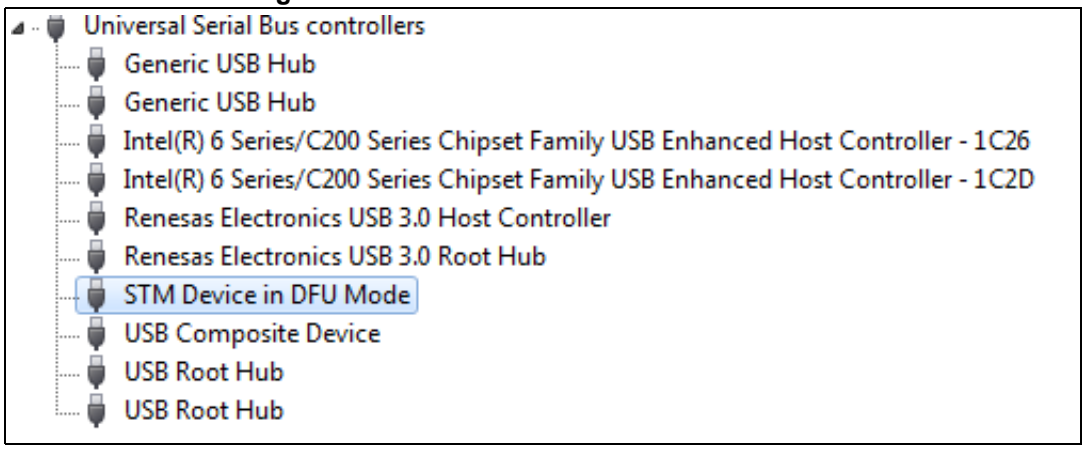
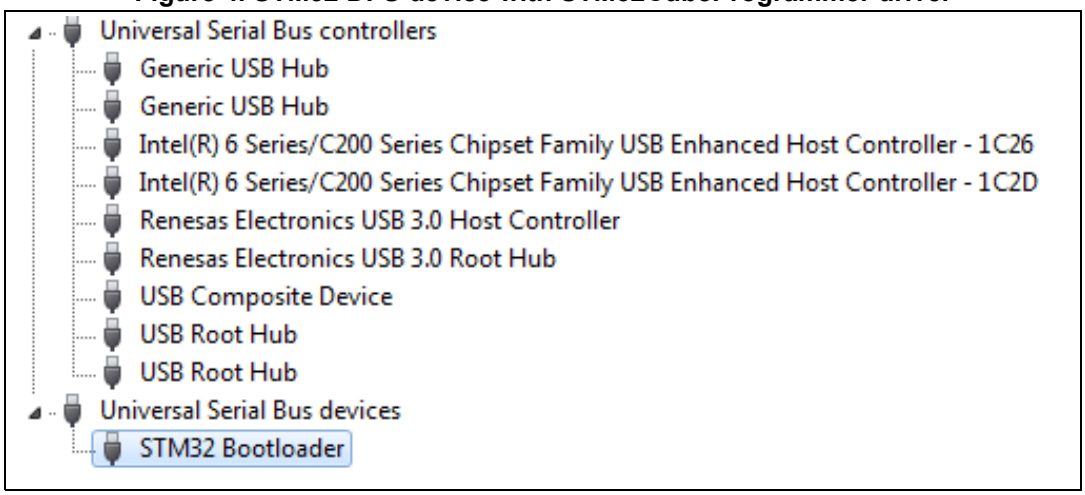


Figure 4. STM32 DFU device with STM32CubeProgrammer driver



Note: When using USB DFU interface or STLink interface on a Windows 7 PC, make sure that all of your USB 3.0 controller's drivers are up to date. Older versions of the drivers may have a bug that prevents access or causes connection problems with USB devices.

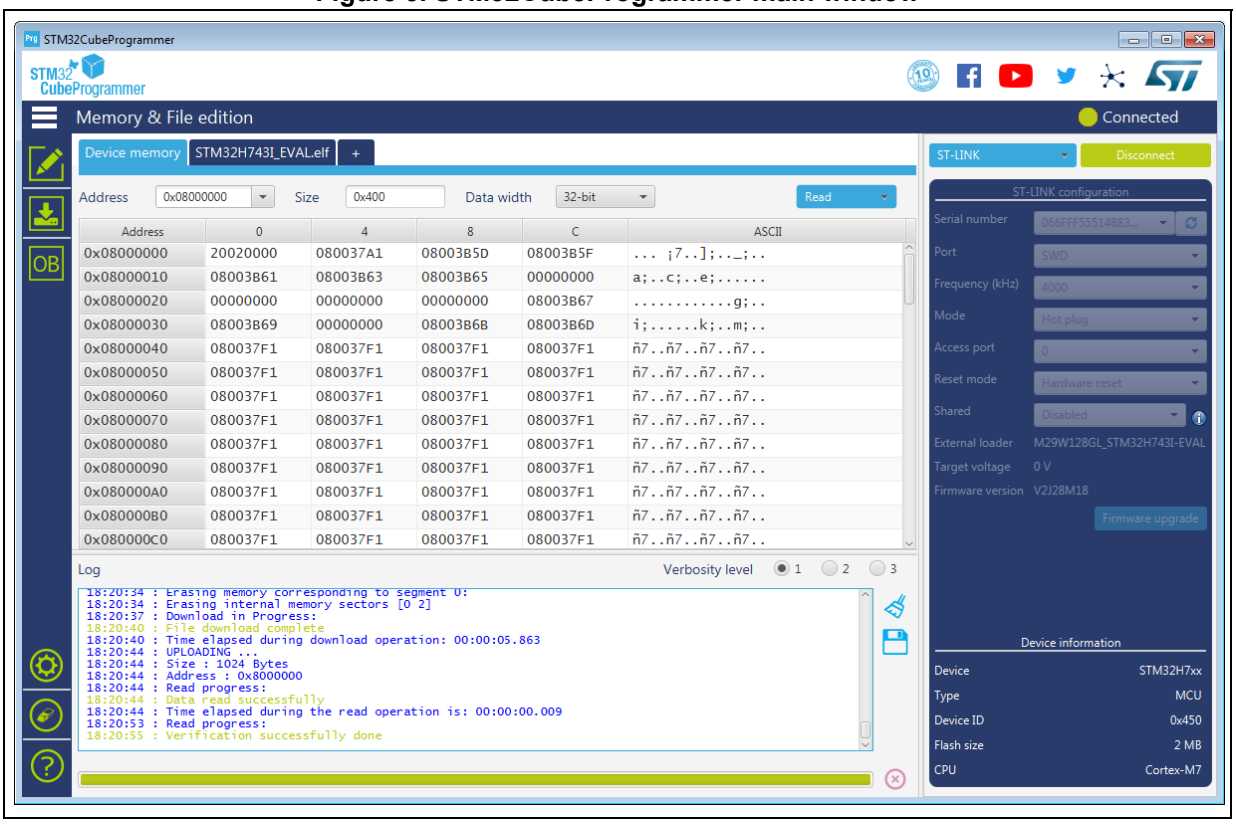
1.2.5 ST-LINK driver

To be able to connect to a STM32 device through a debug interface using ST-LINK/V2, ST-LINKV2-1 or ST-LINK-V3, you need to 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

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 switching 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-lined button) on the top left corner, the Main menu expands and displays a textual description:

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 following 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 progression 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.

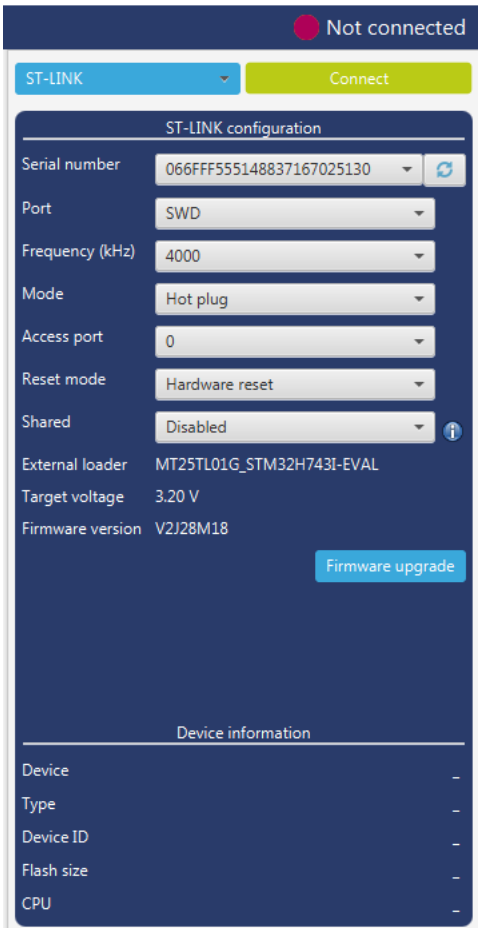
2.1.4 Target configuration panel

This is the first panel to look at before connecting to a target. It allows you 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 checking of 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 connecting.

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**: Select 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**: The 'Connect Under Reset' mode allows 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**: The 'Hot Plug' mode allows 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 the 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 Application Interrupt and Reset Control Register (AIRCR).
- **Shared**: Enable 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

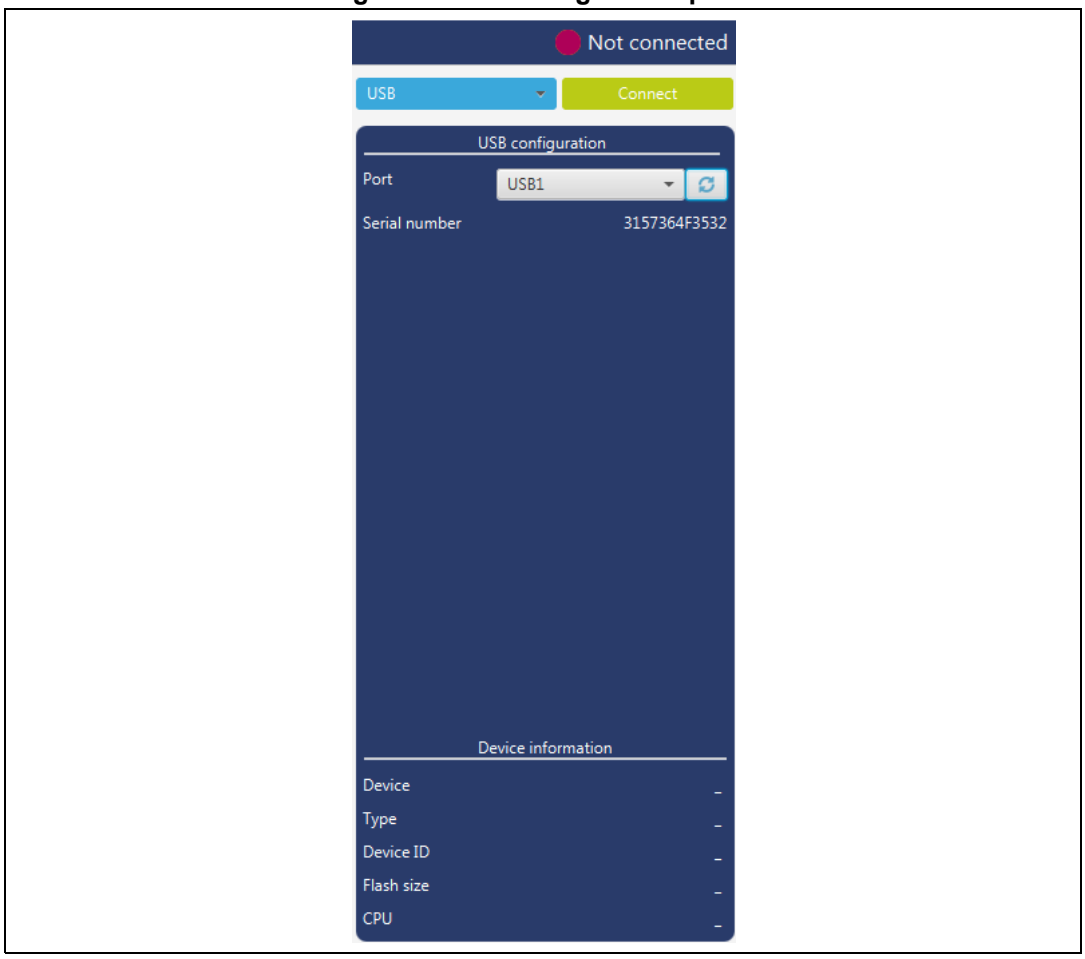
- **Port**: Selects the com port to which the target STM32 is connected. Use the refresh button to recheck the available com port on the PC.

Note: The STM32 mulst boot in bootloader mode using boot pins and/or the option bits. Check AN2606 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 for more information on the STM32 bootloader.

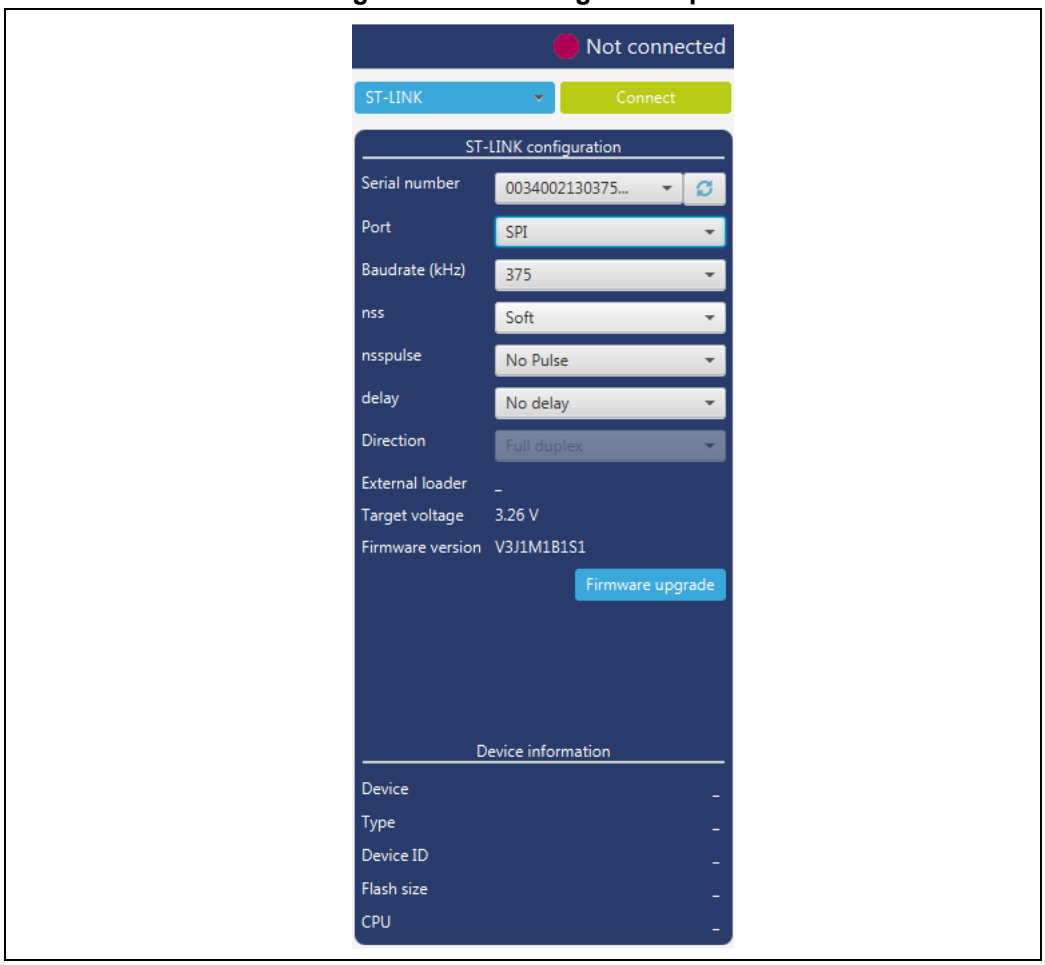
Once the correct interface settings are set, click on the ‘connect’ button to connect to the target interface. If the connection succeeds, it is shown in the indicator above the button that turns to green.

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

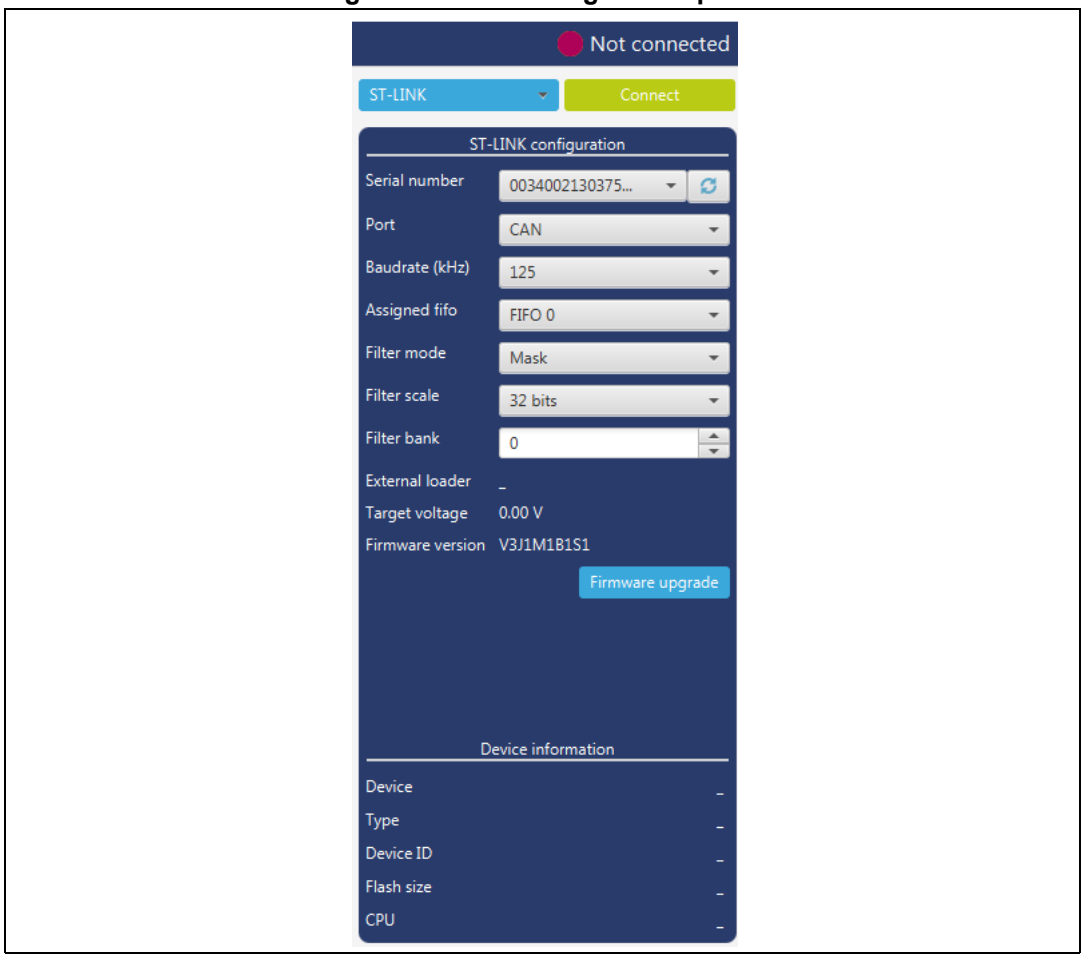
Figure 11. SPI configuration panel



- **Serial number**: This field contains the serial numbers of all connected ST-LINK-V3 probes in case to use 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 periods.
- **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

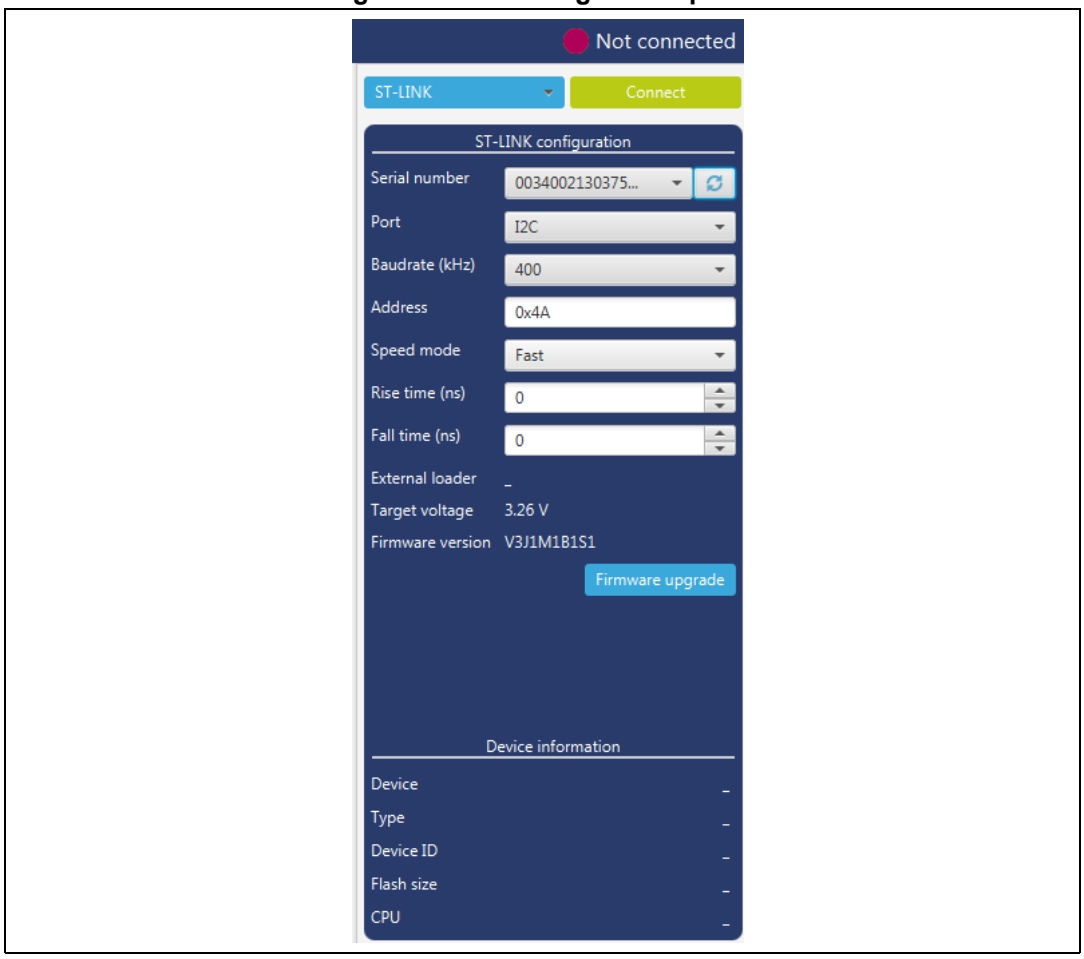
Figure 12. CAN configuration panel



- **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**: Value between 0 and 13 to choose the filter bank number.

I2C settings

Figure 13. I2C configuration panel



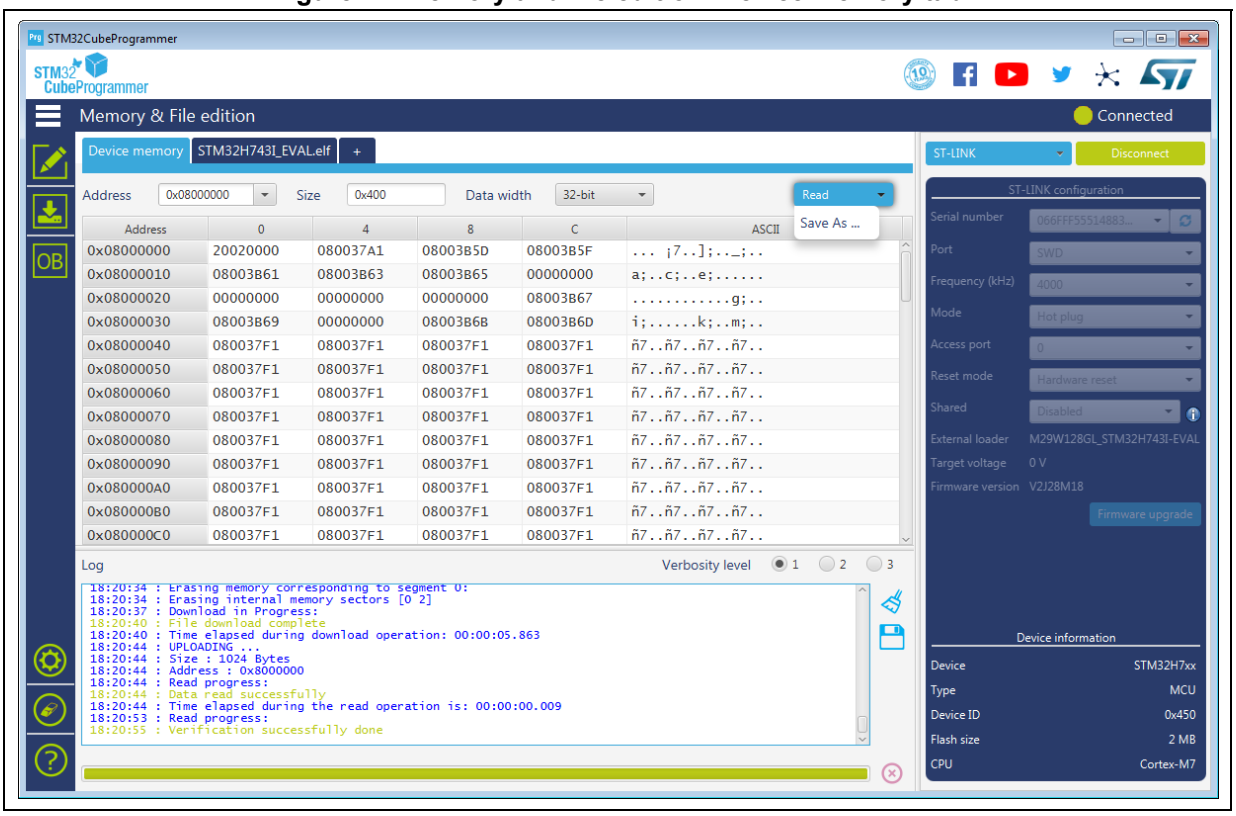
- **Serial number**: This field contains the serial numbers of all connected ST-LINK-V3 probes in case to use I2C Bootloader.
- **Port**: Selects the I2C devices connected to the PC. You can use the refresh button to recheck the available devices.
- **Baudrate**: Selects the I2C baud rate.
- **Address**: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

2.2 Memory and file edition

The Memory and file edition panel allows you to do two things: Reading and displaying target memory and file contents.

2.2.1 Reading and displaying target memory

Figure 14. Memory and file edition: Device memory tab

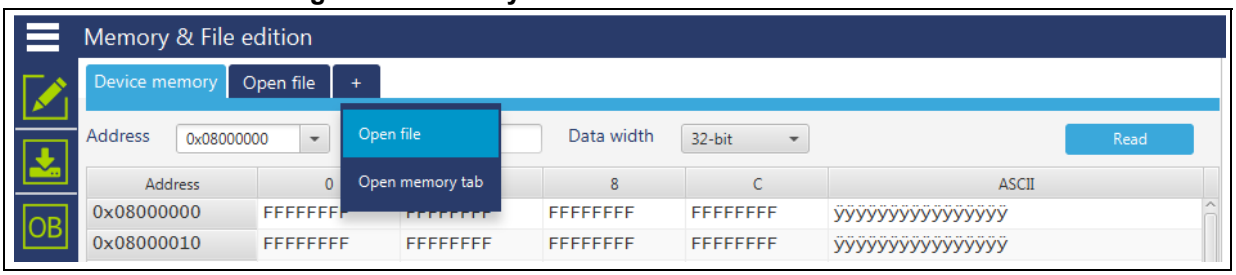


After target connection, you can read the STM32 target memory using this panel. To do this, specify the address and the size of the data to be read, then click on the Read button in the top-left corner. You can display the data in different formats (8, 16- and 32-bit) using the ‘Data width’ combo box.

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

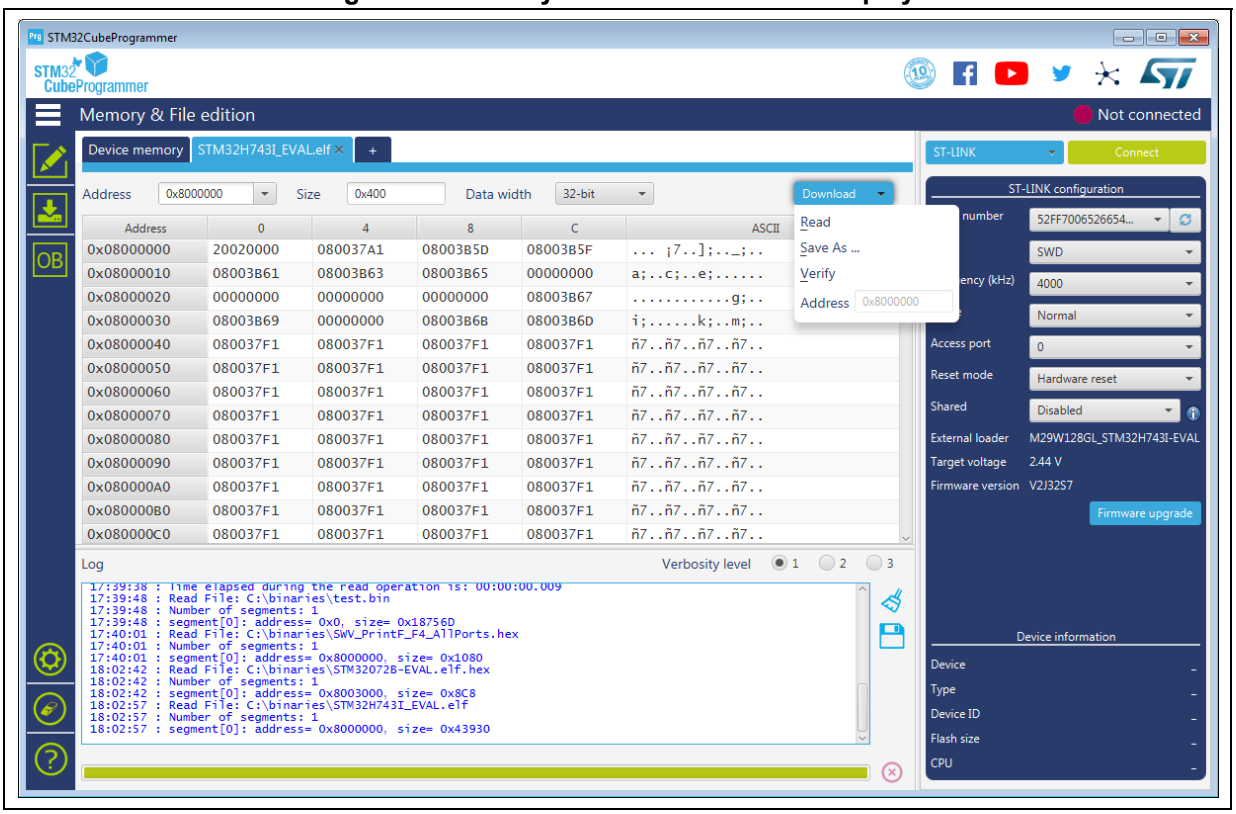


2.2.2 Reading and displaying a file

To open and display a file, just click on the + and select ‘Open File’ menu as illustrated in *Figure 16*.

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

Figure 16. Memory and file edition: File Display



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.

You can modify the address field 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. In case of binary file you need to specify the download address in the “Address” menu. You can also verify if the file is already downloaded using the “Verify” menu.

In addition, you can save the file in another format (.bin, .hex or .srec).

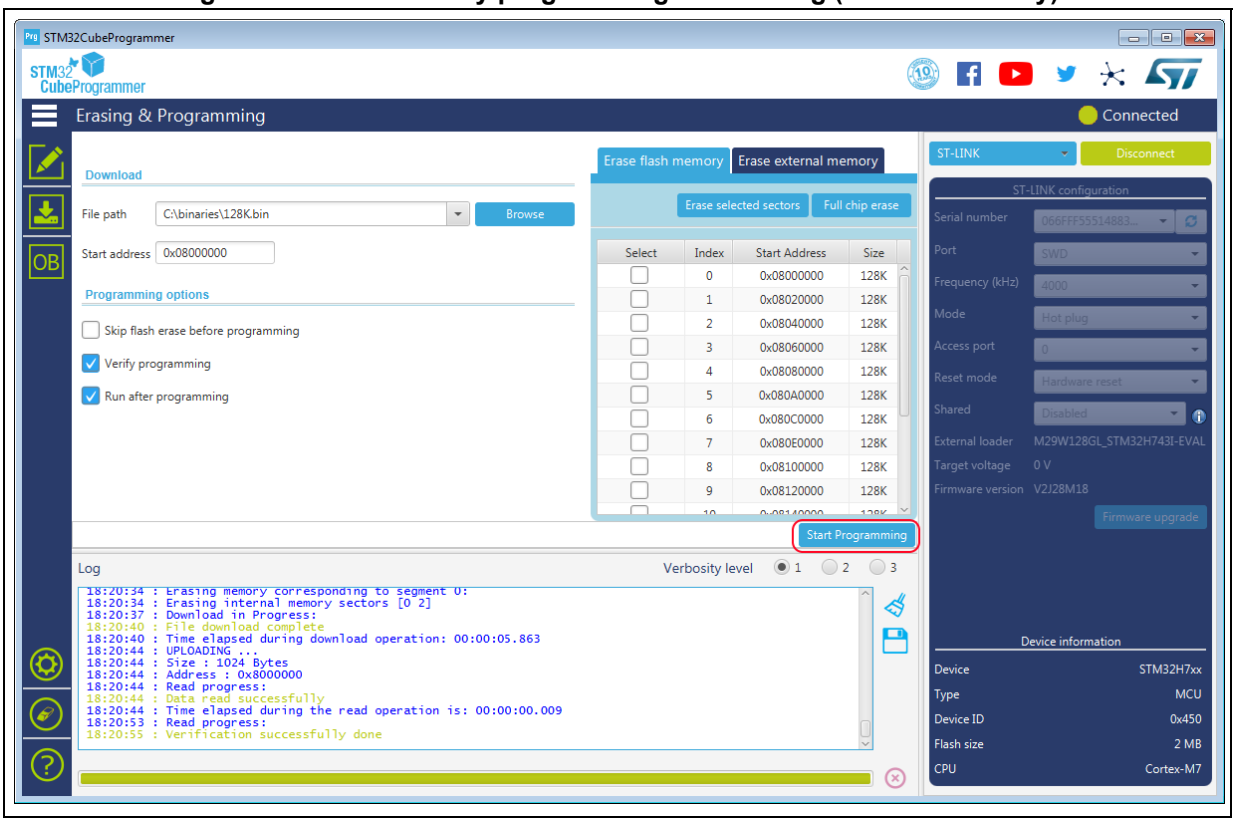
As for the ‘Device memory’ tab, you can display the file memory content in different formats (8-bit, 16-bit and 32-bit) using the ‘Data width’ combo box.

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 all the Flash memory.

Memory programming

To program a memory you need to 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 tools do not erase the memory 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.

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

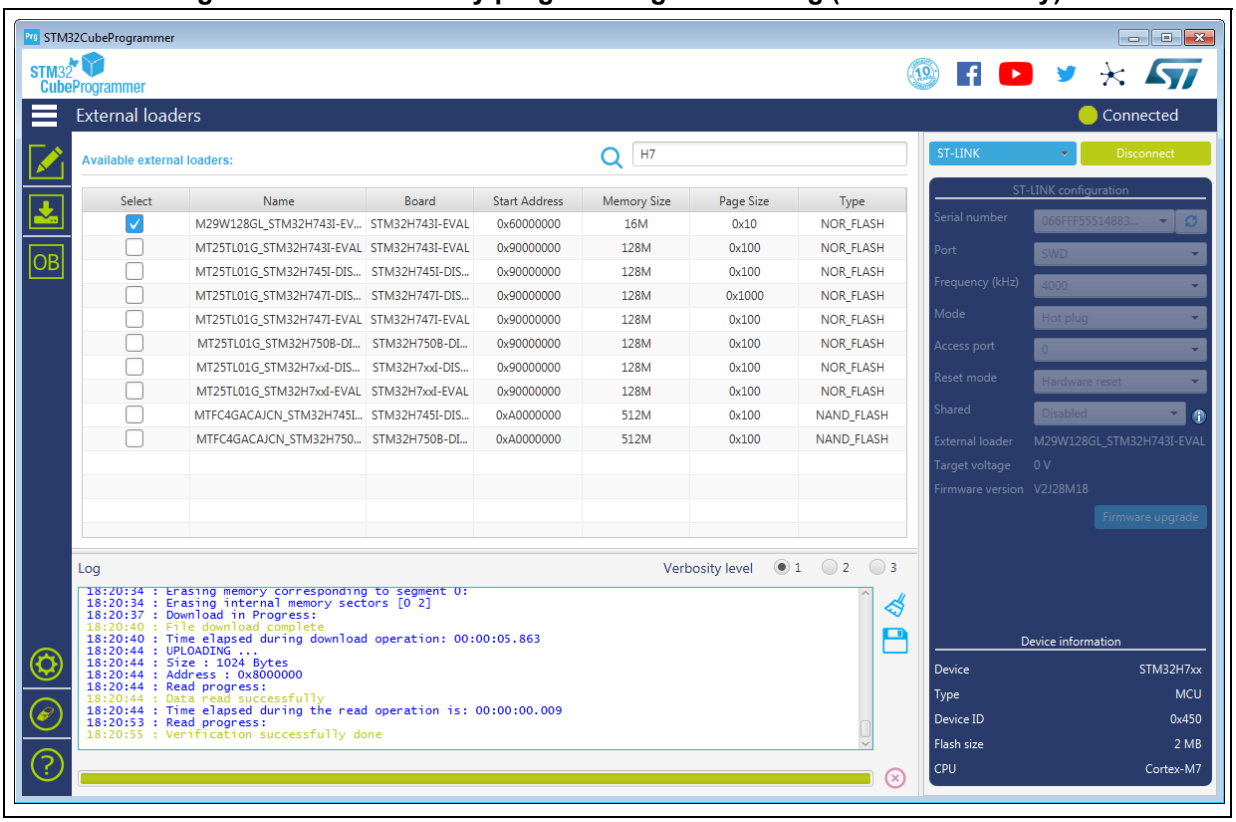
If you need to program an external memory connected to the STM32 via any of the available interfaces (SPI, FMC, FSMC, QSPI, OCTOSPI…) you need and external loader.

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 loaders, see *Section 2.3.3* for more details on how to create it.

To program 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 (read, erase and program) in its memory range.

The ‘External flash erasing’ tab on the right of the “Erasing and Programming” panel displays the memory sectors, and allows sector, or a 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 the three toolchains keeping the same compiler/linker configurations, as in the examples.

The external Flash programming mechanism is the same as that 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 on 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 which are 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`" = the address of the first sector to be erased and "`EndAddress`" = the address of the end sector to be erased.

Note: This function is not used in case of an external SRAM memory 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 sectors
- Calls the `Write()` function to program the memory.

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

- `Read` function

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

  Returns 1 if success, and 0 if failure.

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

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

Note: For QSPI/OSPI (Quad-SPI/ Octo-SPI) memories, the memory mapped mode can be defined in the Init function; in that case the Read function is useless since the data could 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 the case of a successful operation, and 0 in the 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 presented 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,
};
```

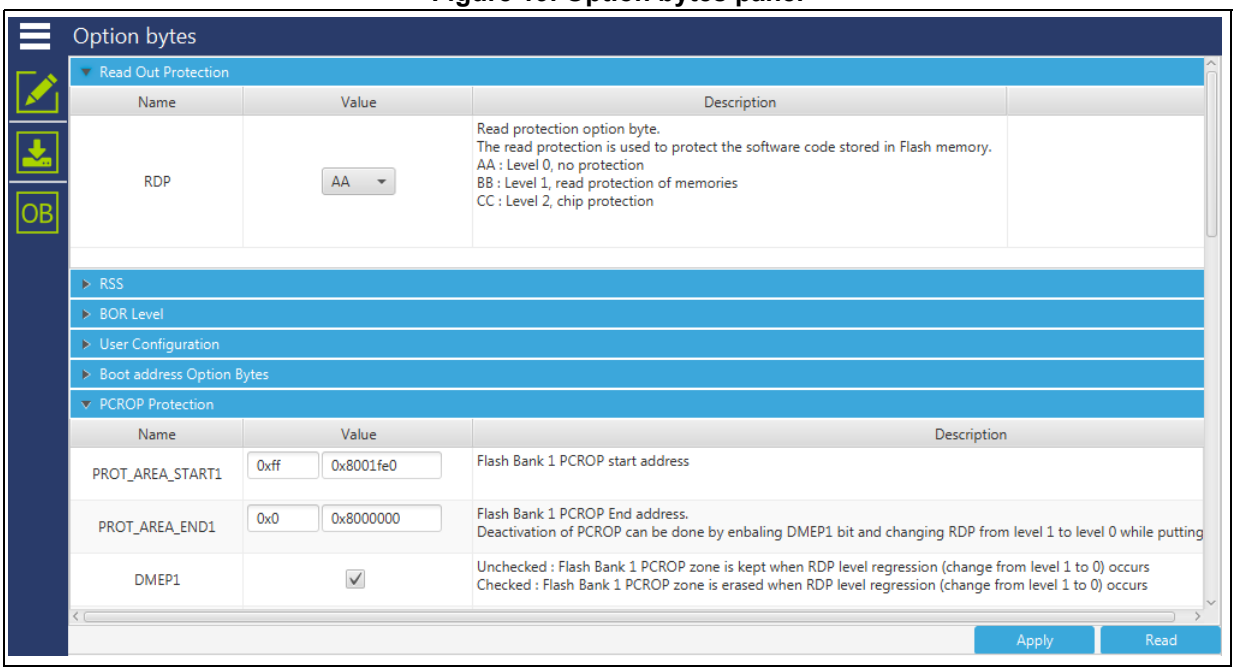
2.4 Option bytes

The option bytes panel allows 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, its value and a description of its 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 will program then verify that the modified option bytes are well 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 programming manual and reference manual available from *www.st.com*.

3 STM32CubeProgrammer command line interface (CLI)

3.1 Command line usage

The following sections describe how to use the STM32CubeProgrammer from the command line. Available commands are shown in *Figure 20*.

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

Figure 20. 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 few microseconds
     [noinit=noinit_bit]: Set No Init bits, value in <0,1) ..., default 0
    CAN port optional parameters:
     [br=<rbaudrate>]    : Baudrate : 125, 250, 500, 1000 Kbps, default 125
     [mode=<canmode>]    : CAN Mode : NORMAL, LOOPBACK..., default NORMAL
     [ide=<type>]        : CAN Type : STANDARD or EXTENDED, default STANDARD
     [rt=<format>]       : Frame Format: DATA or REMOTE, default DATA
     [fifo=<afifo>]      : Msg Receive : FIFO0 or FIFO1, default FIFO0
     [fm=<fmode]         : Filter Mode : MASK or LIST, default MASK
     [fs=<fscale>)]      : Filter Scale: 16 or 32, default 32
     [fe=<fenable>)]     : Filter Activation : ENABLE or DISABLE, default ENABLE
     [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 400
     [sm=<smode>)]       : Speed Mode : STANDARD or FAST, default FAST
     [am=<addmode>)]     : Address Mode : 7 or 10 bits, default 7
     [af=<afilter>)]     : Analog filter : ENABLE or DISABLE, default ENABLE
     [df=<dfilter>)]     : Digital filter : ENABLE or DISABLE, default DISABLE
     [dnf=<dnfilter>)]   : Digital noise filter : 0 to 15, default 0
     [rt=<rtime>)]       : Rise time : 0-1000(STANDARD), 0-300(FAST), default 0
     [ft=<ftime>)]       : Fall time : 0-300 (STANDARD), 0-300(FAST), default 0
     [noinit=noinit_bit]: Set No Init bits, value in <0,1) ..., default 0
 -e,     --erase         : Erase memory pages/sectors devices:
                           Not supported for STM32MP
     [all]               : Erase all sectors
     [<sectorsCodes)>]   : Erase the specified sectors identified by sectors
                           codes. ex: 0, 1, 2 to erase sectors 0, 1 and 2
     [<[start end])>]    : Erase the specified sectors starting from
                           start code to end code, ex: -e [5 10]
 -w,     --write
 -d,     --download      : Download the content of a file into device memory
     <file_path>         : File path name to be downloaded: <bin, hex, srec,
                           elf, stm32 or tsv file>
     [<address>)]        : Start address of download
 -w32                    : Write a 32-bits data into device memory
     <address>           : Start address of download
     <32-bit_data>       : 32-bit data to be downloaded
                           values should be separated by space
 -v,     --verify        : Verify if the programming operation is achieved
                           successfully
 -r32                    : Read a 32-bit data from device memory
     <address>           : Read start address
     <size>              : Size of data
 -rst                    : Reset system
 -hardRst                : Hardware reset
                           Available only with JTAG/SWD debug port
 -halt                   : Halt core
 -step                   : Step core
                           Available only with JTAG/SWD debug port
 -score                  : Get core status
                           Available only with JTAG/SWD debug port
 -coreReg                : Read/Write core registers
     [<core_register>)]    R0/../R15/PC/LR/PSP/MSP/XPSR/APSR/IPSR/EPSR/
                           PRIMASK/BASEPRI/FAULTMASK/CONTROL
     [core_reg=<value>)]   value in case of write opration
                           Note: multiple registers can be handled at once
                           Available only with JTAG/SWD debug port

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

 -el,    --extload       : Select a custom external memory-loader
     <file_path>         : External memory-loader file path
 -s,     --start
 -g,     --go            : Run the code at the specified address.
     [<address>)]        : Start address
 -rdu,   --readunprotect: Remove memory's Read Protection by shifting the RDP
                           level from level 1 to level 0.

 -ob,    --optionbytes   : This command allows the user to manipulate the device
                           's OptionBytes by displaying or modifying them.
     [displ]             : This option allows the user to display the whole set
                           of Option Bytes.
     [OptByte=<value>)] : This option allows the user to program the given
                           Option Byte.
```

3.2 Generic commands

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

3.2.1 Connect command

-c, --connect

Description: Establish the connection to the device. This command allows the host to open the chosen device's 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 could 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 frequency entered 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 which you know its 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 allows 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 allows 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]` : Enable shared mode allowing connection of two or more instances of STM32CubeProgrammer or other debugger to the same ST-LINK probe.

`[tcpport=<Port>]` : Select the TCP Port to connect to an ST-Link Server. Shared option must be selected. Default value 7184

Note: Shared mode is supported only on windows.

- USB Options

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

- SPI Options

`[br=<baudrate>]` : Baudrate ex 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 polynom value.

`[datasize=<size>]` : 8bit/16bit, default 8.

`[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 can never be 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 be able to synchronize on the CAN bus and start reception and transmission between the Host and the CAN device. The mode Normal 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.

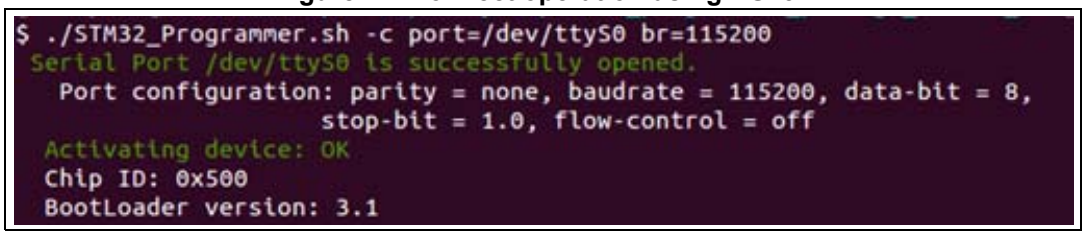
[fbn=<fbanknb>] : Filter Bank Number: 0 to 13, default 0.

- Using UART

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

The result of this example is shown in *Figure 21*.

Figure 21. Connect operation using RS232



Example using USB

./STM32_Programmer.sh -c port=usb1

The result of this example is shown in *Figure 22*:

Figure 22. 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 are ignored (baud rate, parity, data-bits, frequency, index, and so on) 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 device used.

Example using JTAG/SWD debug port

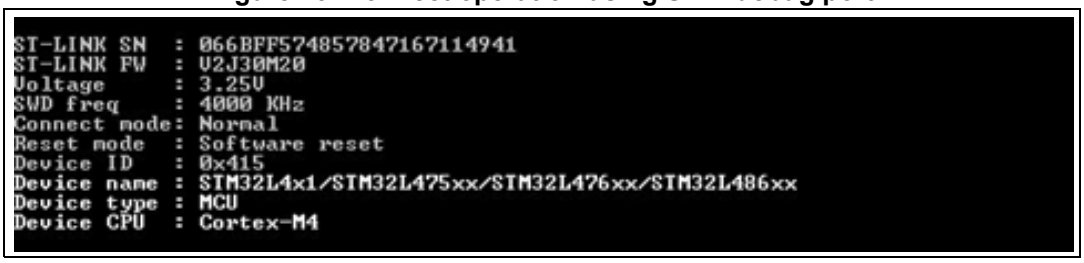
To connect using port connection mode with ST-LINK probe it is necessary to mention the port name with the connect command at least (for example: `-c port=JTAG`).

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 in *Figure 23* shows a connection example with an STM32 with device ID 0x415.

Figure 23. 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 a 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 value 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 in this example, the ap parameter is used, 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, which is available on ST web site (STSW-LINK007).

Example using SPI

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

The result of this example is shown in *Figure 24*.

Figure 24. 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 25*.

Figure 25. Connect operation using CAN port

Note: Not all devices implements this feature, make sure that the device 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> change from a device to another, refer to the document AN2606 to extract the correct one. In our case, the MCU 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 do not support the value 400.

The result of this example is shown in *Figure 26*.

Figure 26. Connect operation using I2C port

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

Note: Not all devices implements 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 of memory, or to erase 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 the 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

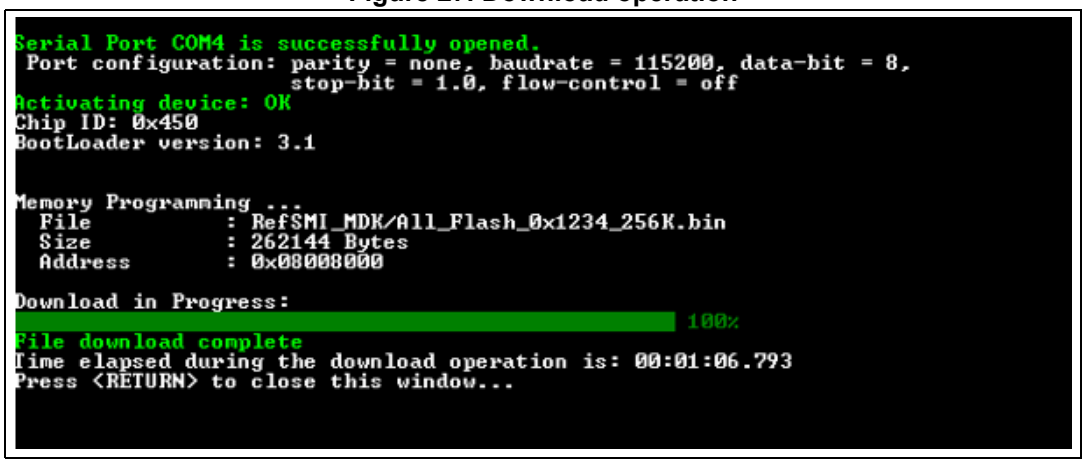
Example:

```
-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 27*.

Figure 27. Download operation



Note: To verify that the download was successful, you can 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 allows 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 allows 2000 bytes to be read, starting from address 0x20007000and upload its content to a binary file *"/local/benayedh/Binaries/read2000.bin"*

-r32

Description: Read 32bit data memory.

Syntax: -r32 <start_address> <size>

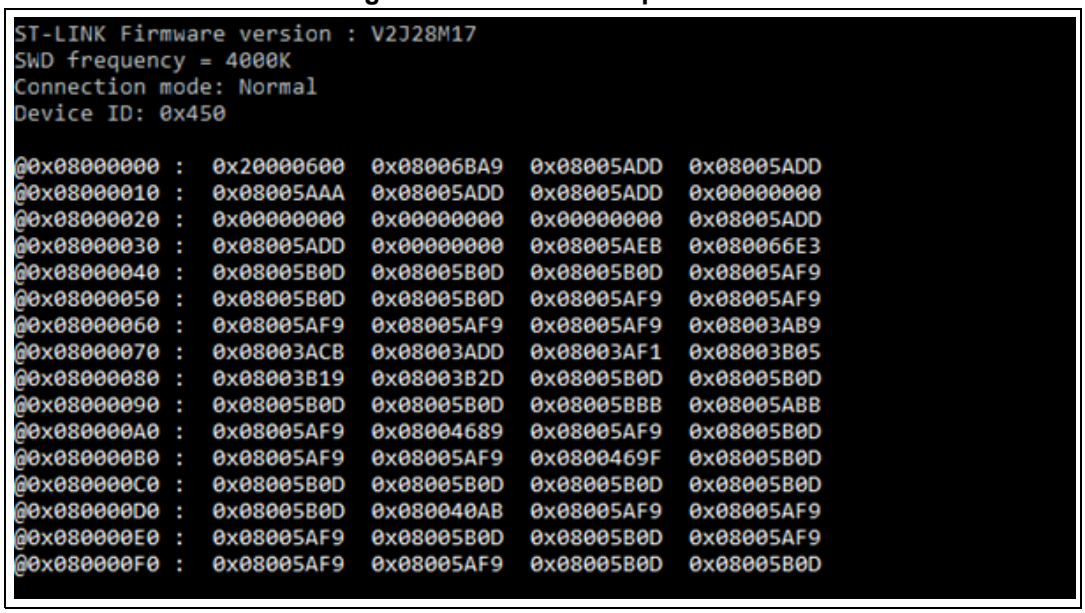
`<start_address>` : Start address of read.

`<size>` : Size of memory content to be read.

Example:

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

Figure 28. 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 allows 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: Execute a software system reset;

Syntax: `-rst`

-hardRst

Description: Generate 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: Halt the core.

Syntax: `-halt`

-step

Description: Execute one instruction.

Syntax: `-step`

-score

Description: Display the Cortex-M core status.

The core status could 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 in the case of a write opration. 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.

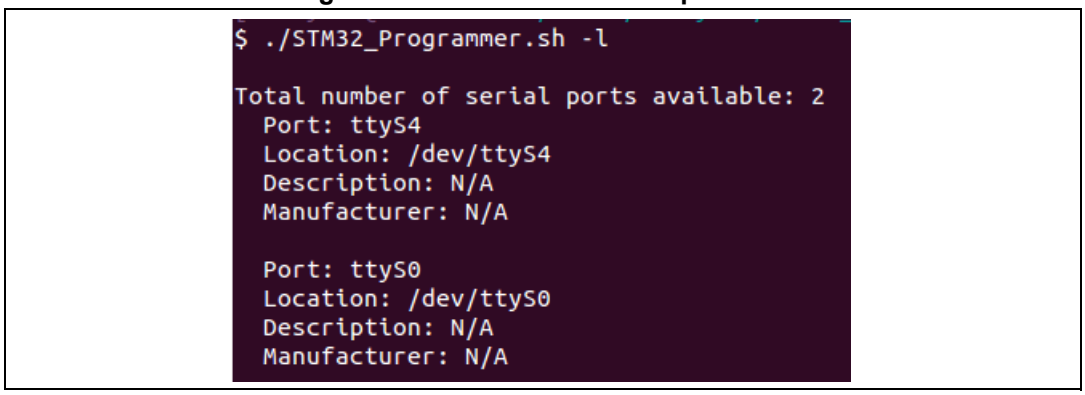
Syntax: `-l, --list`

Example:

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

The result of this example is shown in *Figure 29*:

Figure 29. 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 allows more messages to be displayed in order to be more verbose.

Syntax: -vb <level>

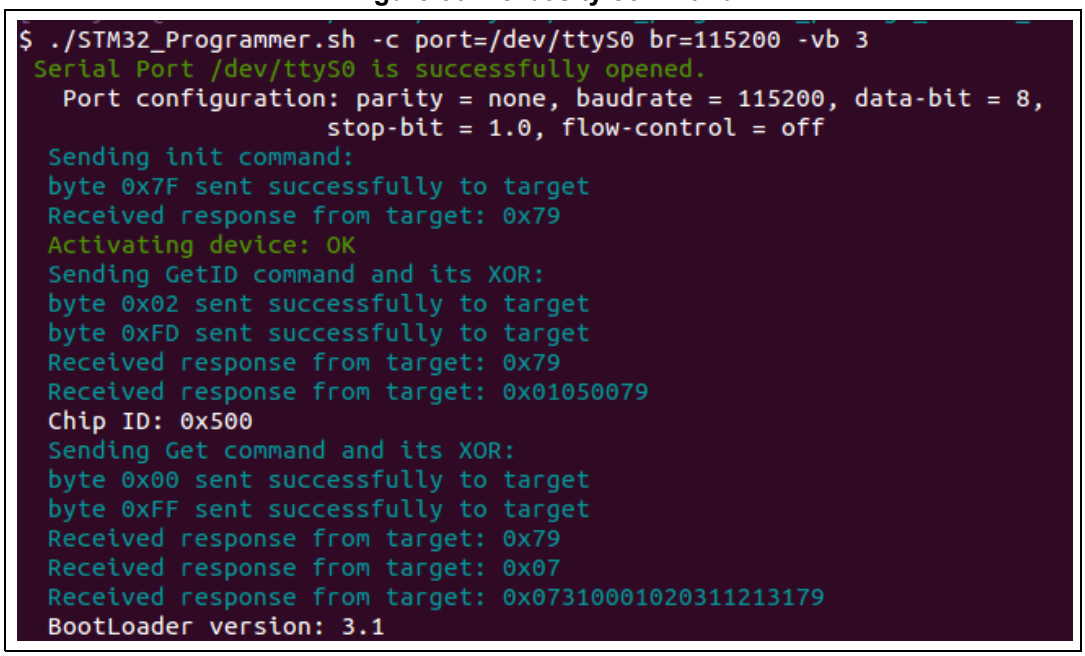
`<level>` : Verbosity level, value in {1, 2, 3} default value vb=1

Example:

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

The result of this example is shown in *Figure 30*:

Figure 30. Verbosity command



3.2.11 Log command

-log, --log

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

Syntax: `-log [filePath.log]`

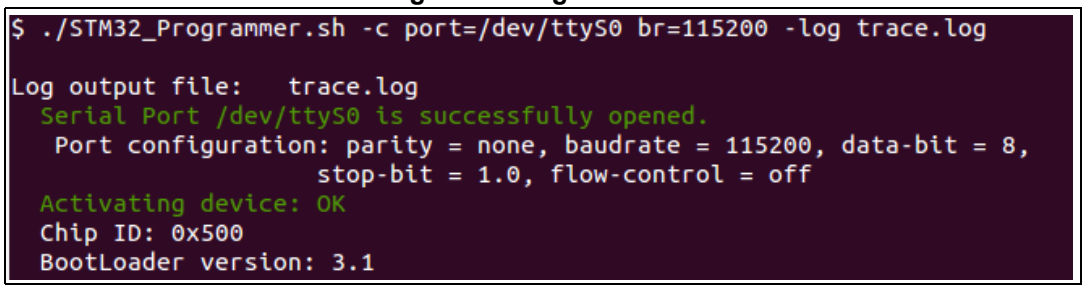
`[filePath.log]` :path of log file, default path is $HOME/.STM32CubeProgrammer/trace.log

Example:

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

The result of this example is shown in Log command and *Figure 31*.

Figure 31. Log command



The log file trace.log contains verbose messages such as those shown in *Figure 32*.

Figure 32. 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

-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 Option Bytes command

-ob, --optionbytes

Description: This command allows the user to manipulate the device's Option Bytes by displaying or modifying them.

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

`[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.

Example:

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

Note: For more information about device's option bytes, refer to the option bytes section in the device Flash memory programming manual and reference manual available from the www.st.com website.

3.2.15 Safety lib command

-sl, --safelib

Description: This command allows a firmware file t 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 the *Figure 33*:

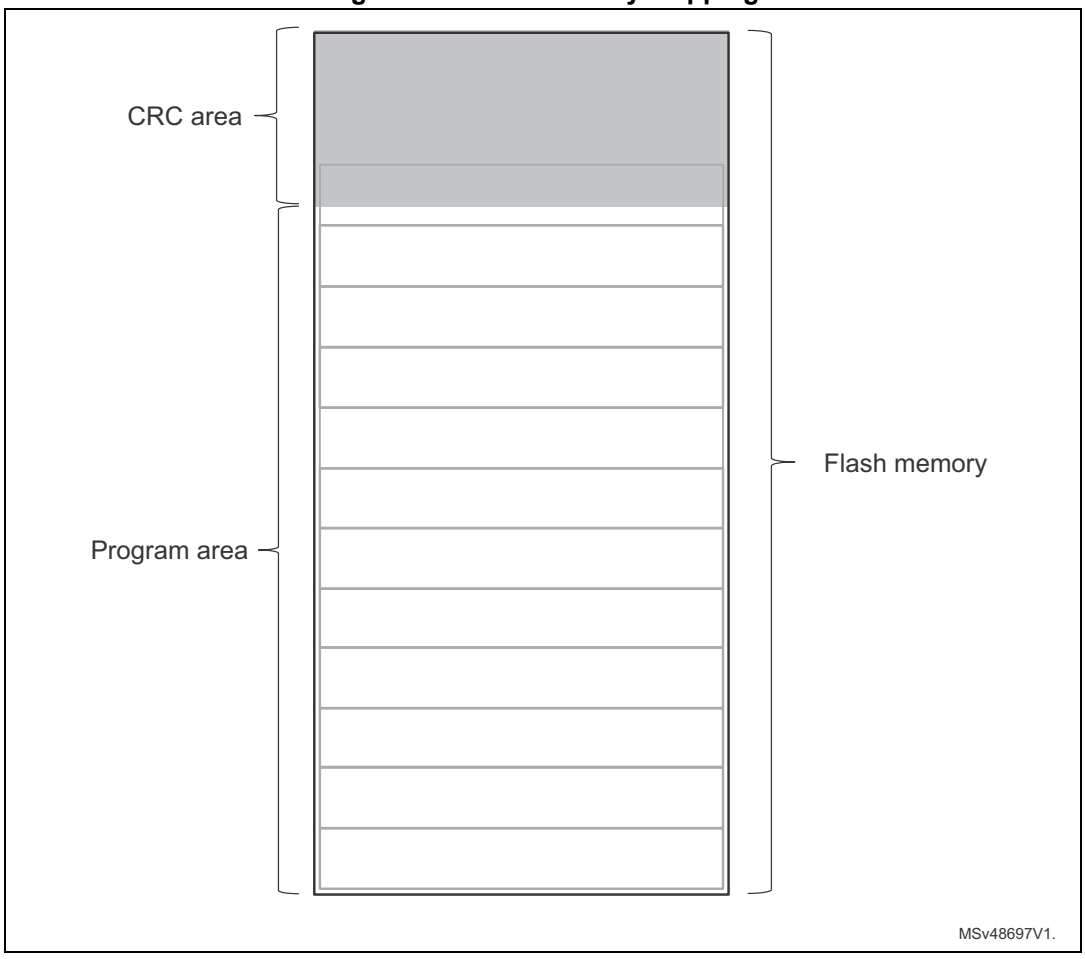
Figure 33. Safety lib command

```
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
```

Flash program memory is divided into slices (the slice size is given as a parameter to the safety lib command as shown in the example above). To 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 34*:

Figure 34. Flash memory mapping



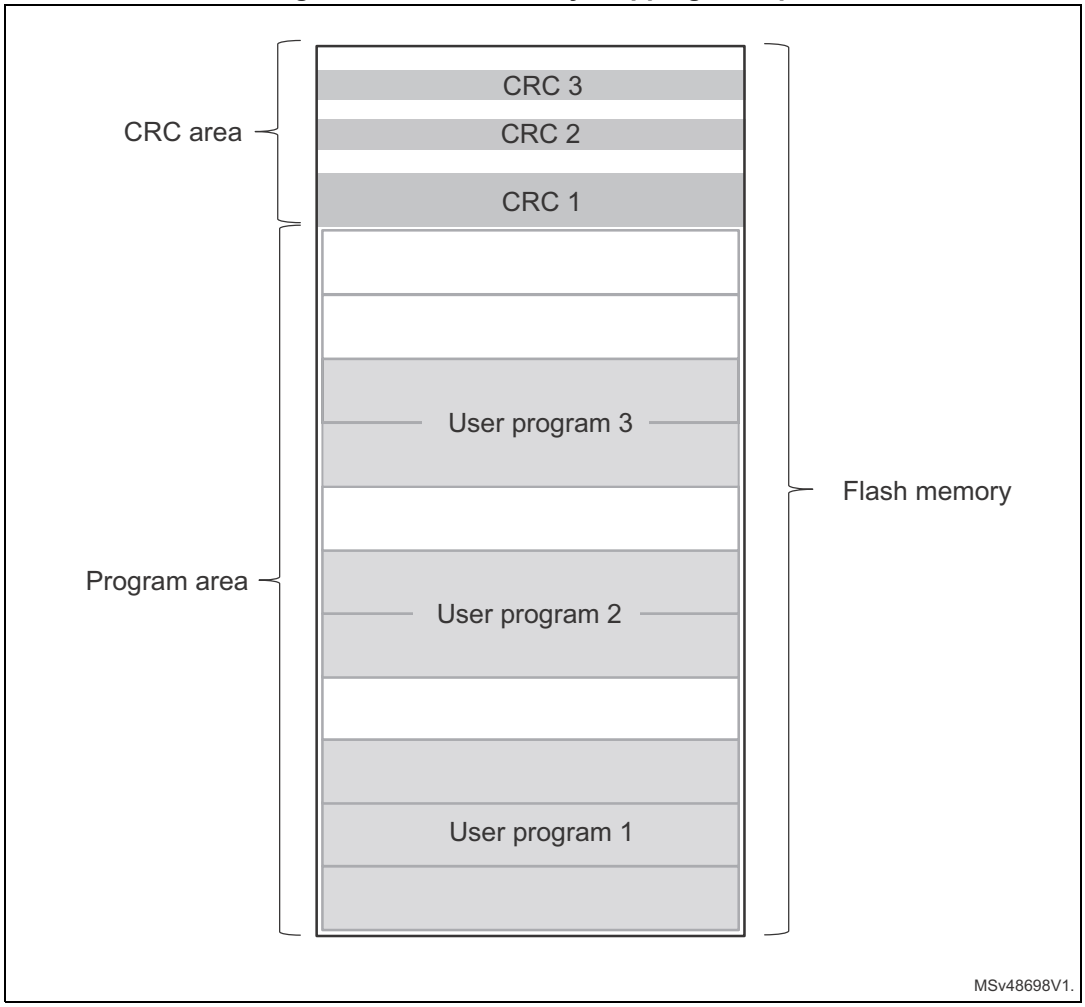
The address and size of the CRCs area are determined as follows:

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 35*.

Figure 35. Flash memory mapping example



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

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

4 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 20: STM32CubeProgrammer: available commands*. – *Figure 23: 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.