STM32CubeProgrammer documentation

Release 2.24.0

Nov 13, 2026

Table of contents

1 STM32CubeProgrammer documentation 3
1.1 Introduction . . . 3
1.2 STM32CubeProgrammer agentic AI connector . . . 5

2 Getting Started with STM32CubeProgrammer 6
2.1 Introduction . . . 6
2.2 Additional links . . . 17

3 User manual 18
3.1 Graphical User interface . . . 18
3.2 Command lines . . . 108

4 Supported Interfaces 172
4.1 Introduction . . . 172
4.2 Debug Interfaces . . . 172
4.3 Bootloader Interfaces . . . 172
4.4 Additional article . . . 172

5 Release notes 203
5.1 Release details . . . 203
5.2 Release limitations . . . 205

6 Community 210
6.1 Support link . . . 210
6.2 Community link . . . 210
6.3 Knowledge articles link . . . 210
6.4 Additional links . . . 210

7 FAQ 212
7.1 Basic questions . . . 212
7.2 Getting started questions . . . 213
7.3 Security questions . . . 214
7.4 Production question . . . 214
7.5 History . . . 214

1 STM32CubeProgrammer documentation

Welcome to the documentation portal for **STM32CubeProgrammer**.

This document covers the **STM32CubeProgrammer** tool only.

To determine which STM32 series and interfaces (JTAG/SWD, UART, USB, SPI, and others) are supported by STM32CubeProgrammer, refer to the AN2606 STM32 microcontroller system memory boot mode.

1.1 Introduction

STM32CubeProgrammer is a software tool designed for STM32 products. It is currently available as a standalone application and offers several important features, including:

- **Device programming:** Program STM32 microcontrollers and microprocessors via multiple interfaces (JTAG/SWD, UART, USB, SPI, I2C, etc.).
- **Memory management:** Read, write, erase internal and external memories (Flash, RAM, OTP, option bytes, external Flash when supported).
- **Option bytes configuration:** Configure protection and boot-related settings (read-out protection, write protection, boot address, etc.).
- **Secure programming support:** Handle secure services when supported by the target (TrustZone-enabled devices, secure firmware install, etc.).
- **Multi-interface connection:** Connect through ST-LINK probes, integrated bootloaders (UART/USB), or other supported interfaces.
- **Scripting & automation (CLI):** Use command-line interface to automate programming, testing, and production flows.
- **GUI monitoring & logs:** Visualize operations, statuses, error messages, and logs through an intuitive graphical interface.
- **Device information & diagnostics:** Read device ID, revision, memory map, and basic diagnostics information.

Note

STM32CubeProgrammer supports a wide range of STM32 devices and interfaces. Supported devices, bootloader interfaces, and features may vary depending on the STM32 series and the tool version. Please refer to the STM32CubeProgrammer page and release notes on st.com for the detailed list of supported products and functionalities.



Fig. 1.1: Overview of STM32CubeProgrammer

1.2 STM32CubeProgrammer agentic AI connector

STM32CubeProgrammer is available to AI coding agents through the STM32CubeProgrammer Context, part of the STM32Cube agentic AI toolkit. The context is delivered in the `cube-agentic-ai-programmer` bundle.

Through a local MCP service, the STM32CubeProgrammer Context allows an AI coding agent to:

- Program internal and external nonvolatile memory (NVM) of connected STM32 devices.
- Erase memory.
- Read memory.
- Program option bytes.

Connector content

- The connector includes its own MCP server.
- It depends on the `programmer` Cube tool bundle, which provides the **STM32CubeProgrammer CLI**.
- Cube installs this dependency automatically.

Device scope

- All STM32 MCUs are supported.
- STM32 MPUs are not covered.

2 Getting Started with STM32CubeProgrammer

2.1 Introduction

In the STM32Cube ecosystem, STM32CubeProgrammer is used to program STM32 boards during development.

With STM32CubeProgrammer, the following features are available:

- Program all memory areas of the STM32 device, including internal and external Flash memory, option bytes, and one-time programmable (OTP) memory.
- Connect using one amongst multiple interfaces, such as SWD, JTAG, USB, UART, SPI, I2C, or CAN, depending on the device.
- Perform mass erase, memory verification, readout protection control, and other operations.
- Use a graphical user interface for manual operations, or a command-line interface for automation.

2.1.1 System requirements

Supported operating systems and architectures

- **Windows®**:
  - Windows® 10 (32-bit/x86 or 64-bit/x64)
  - Windows® 11 (64-bit/x64)
- **Linux®**:
  - Tested on Ubuntu® 22.04, Ubuntu® 24.04, Fedora® 43
- **MacOS®**:
  - macOS® 15 (Sequoia), macOS® 26 (Tahoe)
  - Supported architectures: x86_64 and ARM-aarch64
- **Raspberry Pi®**:
  - *Raspberry Pi® OS* (Debian® 12 “Bookworm” or later)

Trademark information:

- Windows® is a trademark of Microsoft group of companies.
- Linux® is a registered trademark of Linus Torvalds.
- Ubuntu® is a registered trademark of Canonical Ltd.
- Fedora® is a trademark of Red Hat, Inc.
- macOS® is a trademark of Apple Inc., registered in the U.S. and other countries and regions.

Note

- – **Windows®** 64-bit: Intel® Core is used by default.
- – **macOS®**: ARM-aarch64 core is supported natively. Intel® x86_64 core is supported through Rosetta emulation.
- – There is no need to install any Java™ SE Run Time Environment since version 2.6.0. STM32CubeProgrammer runs with a bundled JRE available within the downloaded package, and no longer with the one installed on your machine.

Processor (CPU) requirements

Minimum configuration

- **Windows® / Linux®**:
  - Multi-core x86_64 processor (for example, Intel® Core i3)
  - Minimum frequency: 1.6 GHz
- **macOS®**:
  - Apple® Silicon M1 or newer
  - Intel® Core i5

Recommended configuration

- **Windows® / Linux®**:
  - Intel® Core i5 or equivalent
- **macOS®**:
  - Apple® Silicon M1 or newer
  - Intel® Core i7

> **Note**
>
> The bundled JRE is BellSoft Liberica.

Additional macOS® requirements

- Rosetta® must be installed on MacOS computers with Apple® M1 processor to run macOS arch x86_64 package. Rosetta® is installed automatically when installing the package.

Display requirements

- The minimum supported screen resolution is 1024 x 768.

Raspberry Pi support

STM32CubeProgrammer supports Linux ARM 64-bit for Raspberry Pi platforms. It enables command-line STM32 programming directly from Raspberry Pi and compatible ARM Linux hosts.

> **Note**
>
> On Raspberry Pi and ARM aarch64 Linux, STM32CubeProgrammer support is limited to the command-line interface. GUI, debug features, debug authentication, and SFI are not included.

Supported features

- All STM32 devices supported by STM32CubeProgrammer.
- Internal flash programming.
- External flash programming.
- OTP programming.
- Option bytes configuration.
- RDP regression with password for supported products.

Interfaces and connectivity

Existing interfaces remain supported:

- ST-LINK using SWD or JTAG.
- J-Link.
- UART.
- USB.
- Bootloader via ST-LINK V3.
- GPIO-based SPI communication (`/dev/spidevx.x`).
- GPIO-based I2C communication (`/dev/i2c-x`).

2.1.2 Installing STM32CubeProgrammer

This section describes the requirements and procedure for using the software. The setup also offers the optional installation of the *STM32 Trusted Package Creator* tool, used to create secure firmware files for secure firmware install and update. For more information, refer to UM2238 *STM32 Trusted Package Creator tool software description*.

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
```

When using ST-LINK/J-Link probes or USB DFU to connect to a target, copy the rules files located under the `Driver/rules` folder to `/etc/udev/rules.d/` on Ubuntu:

```
sudo cp *.* /etc/udev/rules.d
```

Note

`libusb1.0.12` or higher is required to run STM32CubeProgrammer. **Docker and Ubuntu**

When using USB DFU in a Docker context, it is often necessary to have the udev rules installed on the host machine and to create the container (`docker run`) using the following flags:

- `-v /dev:/dev --device /dev:/dev` (to give the container access to the devices on the host machine)
- `--net=host` (to propagate udev host events to the container)

VirtualBox

For optimal performance when using STM32CubeProgrammer in a VirtualBox VM, it is recommended to switch the USB Controller to **USB 3.0 (xHCI) controller** (found in *Settings* → *USB*).

Fedora

To install the libusb package on Fedora, execute the following command:

```
sudo dnf install libusb1
```

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

On Ubuntu 20, the STM32CubeProgrammer icon is not enabled by default. To enable it, right-click on the icon and choose **Allow launching**.

Windows install

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

macOS install

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

Note

If the installation fails, launch it in CLI mode using the command:

```
./SetupSTM32CubeProgrammer-x.y.z.app/Contents/MacOs/SetupSTM32CubeProgrammer-x_y_z_macos
```

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

In case of error, try this workaround:

- Right-click on `SetupSTM32CubeProgrammer-2.12.0`
- Select **Show Package Contents**
- Navigate to `Contents/MacOs`
- Launch `SetupSTM32CubeProgrammer-2_12_0_macos`

Note

The tool launches automatically after installation if the **Launch the tool after install** option in the finish install panel is selected.

Debian Package install

To install the STM32CubeProgrammer CLI tool, download the Debian package and execute the following command:

```
sudo apt install ./stm32cubeprogrammer_vx.y.z._arm64.deb
```

DFU driver

If you are using the STM32 device in USB DFU mode, install the STM32CubeProgrammer DFU driver by running the “STM32 Bootloader.bat” file. This driver is provided with the release package, and can be found in the DFU driver folder. If the DFUSE driver is installed on your machine, first uninstall it, then reboot the machine and run the previously mentioned “.bat” file. Check the ”Delete the driver software for this device option” to avoid reinstalling the old driver when, later, a board is plugged in.



Fig. 2.1: Deleting the old driver software

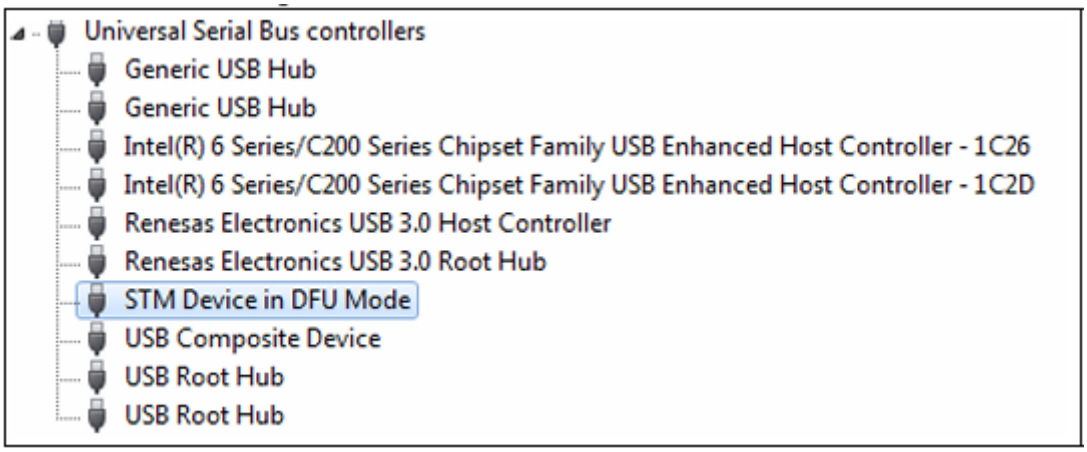


Fig. 2.2: STM32 DFU device with DfuSe driver

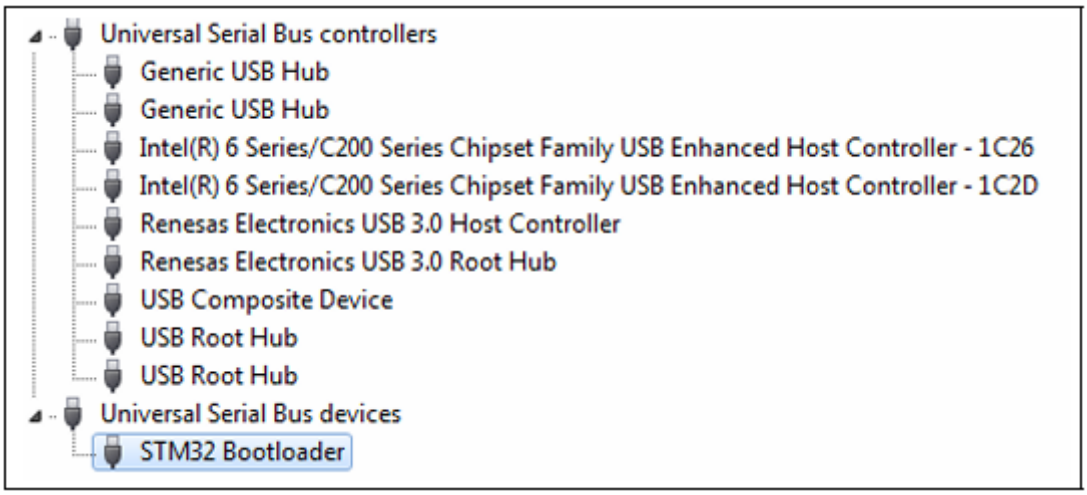


Fig. 2.3: STM32 DFU device with STM32CubeProgrammer driver

Note

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

ST-LINK driver

To connect to an STM32 device through a debug interface using ST-LINK/V2, ST-LINKV2-1, ST-LINK-V3, or ST-LINKV3Power, install the ST-LINK driver by running the "stlink_winusb_install.bat" file. This driver is provided with the release package, it can be found under the "Driver/stsw-link009_v3" folder.

Segger J-link/Flasher library

The Segger J-link/Flasher library is not mandatory for using STM32CubeProgrammer with other interfaces. If you want to build your own package without using it, the library can be deleted without impact on STM32CubeProgrammer.

Installing STM32CubeProgrammer from command line

An installation from console window can be launched in interactive mode, or with a script generated via the installer.

To perform interactive installation, proceed as follows:

- Extract (unzip) into a folder the installer file (`SetupSTM32CubeProgrammer_win64.exe`).
- Open a Standard console window with administrator rights.
- Navigate to the extracted folder using the command: `cd <folder path>`.
- Run the command `jre\bin\java -jar SetupSTM32CubeProgrammer-X.Y.Z.exe -console`.

An initial panel is displayed to start the installation process. At each installation step, an answer is requested.

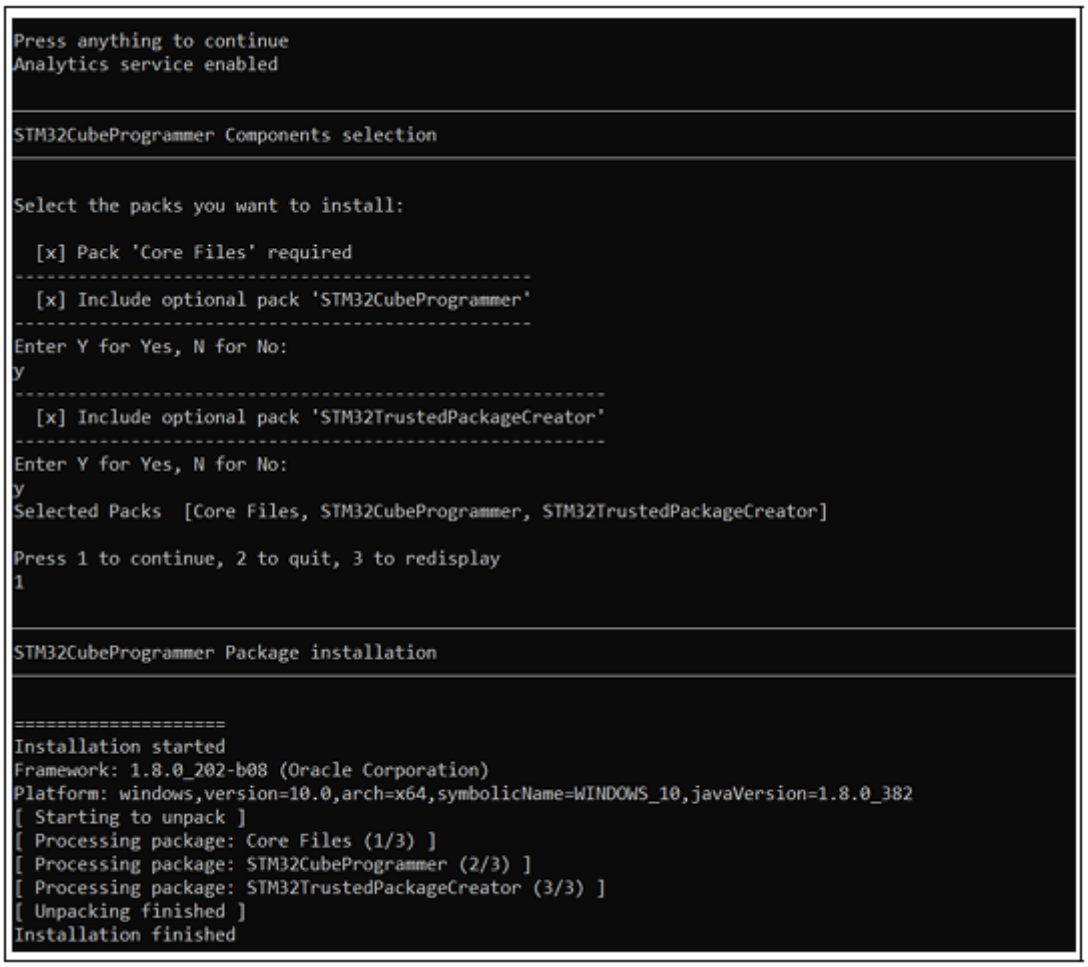


Fig. 2.4: Installation in interactive mode

Automatic/Silent installation mode

At end of an installation, performed either using STM32CubeProgrammer installer graphical interface or console mode, it is possible to generate an auto-installation script containing user configuration and preferences selected during the installation process.

To launch the installation in automatic/Silent mode, proceed as follows:

- Extract (unzip) into a folder the installer file (`SetupSTM32CubeProgrammer_win64.exe`).
- Open a Standard console window with administrator rights.
- Navigate to the extracted folder using the command: `cd <folder path>`.
- Run the Command `jre\bin\java -jar SetupSTM32CubeProgrammer-X.Y.Z.exe ABSOLUTE_PATH_TO_AUTO_INSTALL.xml`.

The installation starts without user interaction.



Fig. 2.5: Auto-install using Console mode

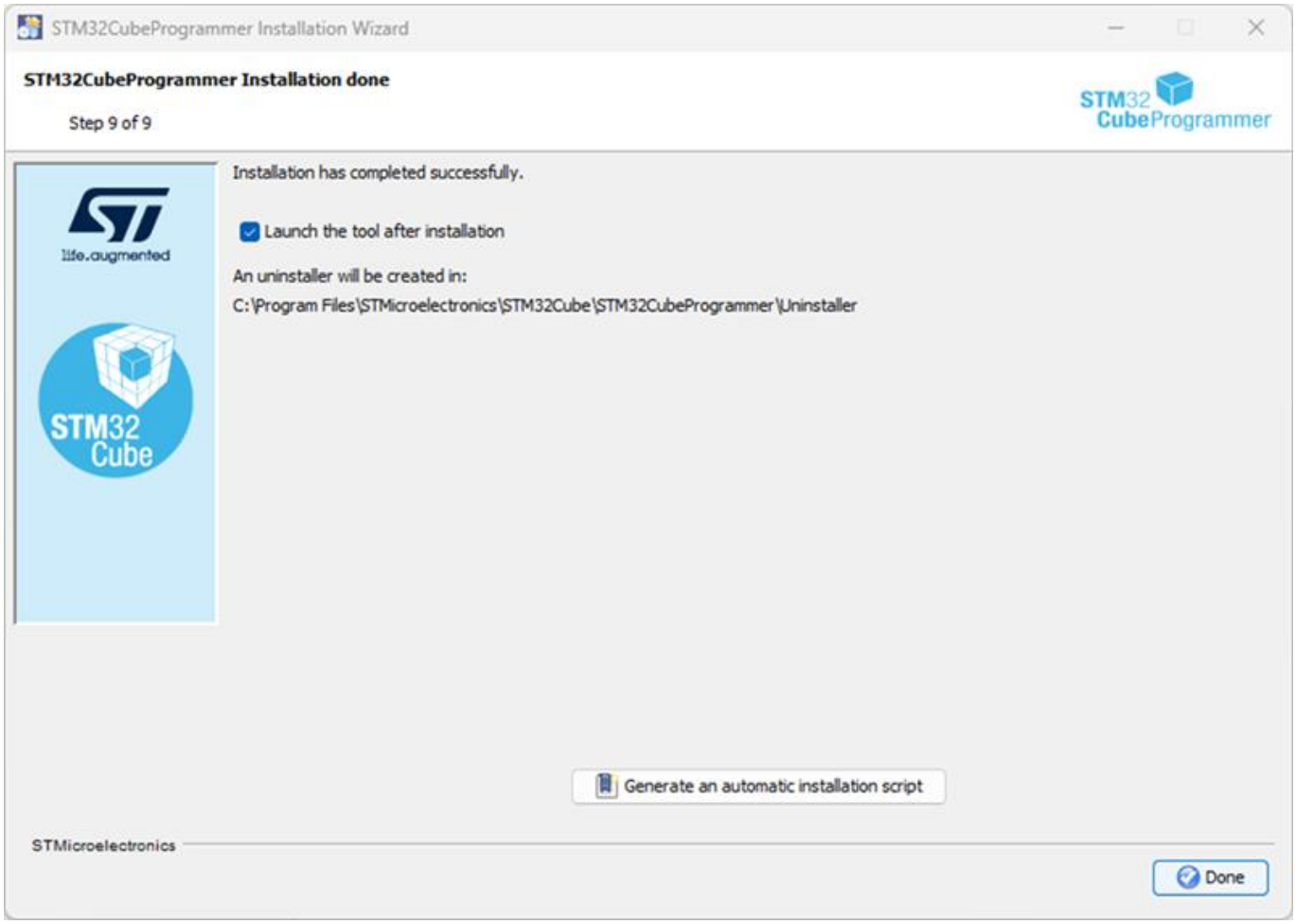


Fig. 2.6: STM32CubeProgrammer installer graphical interface

2.1.3 Uninstallation

Uninstalling STM32CubeProgrammer standalone version

Uninstalling STM32CubeProgrammer on Windows

- Through the Windows control panel:
  a. Select “Programs and Features” to display the list of installed programs.
  b. Right-click “STM32CubeProgrammer” and select “Uninstall”.
- To perform the uninstallation from a shell prompt:
  a. Open a standard console window with administrator rights.
  b. Navigate to the uninstaller folder:

```
cd <STM32CubeProgrammer installation path>\Uninstaller
```

  c. Execute:

```
unscript.bat
```

Uninstalling STM32CubeProgrammer on Linux

- From a terminal prompt:
  a. Navigate to the uninstaller folder:

```
cd <STM32CubeProgrammer installation path>/uninstaller
```

  b. Execute:

```
./uninstall.sh
```

Uninstalling STM32CubeProgrammer on macOS

- Delete the STM32CubeProgrammer installation folder.

Uninstalling STM32CubeProgrammer Debian Package

Uninstalling STM32CubeProgrammer CLI tool:

```
sudo apt remove stm32cubeprogrammer
```

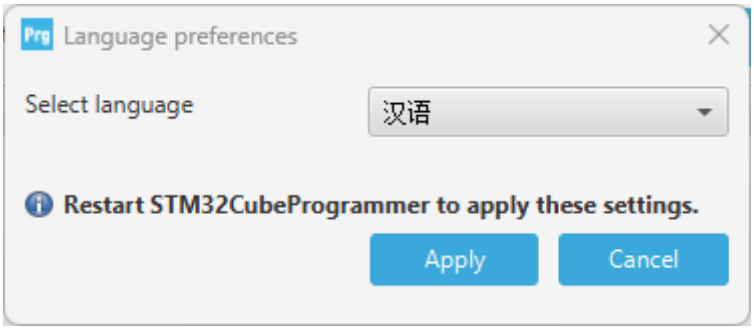
2.1.4 Language Preference

The STM32CubeProgrammer supports multiple languages in addition to default English. Users can change the language by navigating to Help → Language.

The new language settings are applied after restarting the tool. In addition to manual language selection, STM32CubeProgrammer can automatically detect system language.

Language selection overview

The following illustration shows the language selection options available in STM32CubeProgrammer.



2.1.5 Updater

STM32CubeProgrammer updater allows users to make automatic updates of the software and its associated packages. The updater is available in all supported operating systems.

Update steps

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

Proxy settings

The user can manually check the connection by using the “Proxy Settings” window opened with the submenu available in the help button (see the figure below). Three settings are available (see the following figure): - No proxy - Use the system parameters - Use manual configuration of server: add the HTTP proxy name, port, and credentials

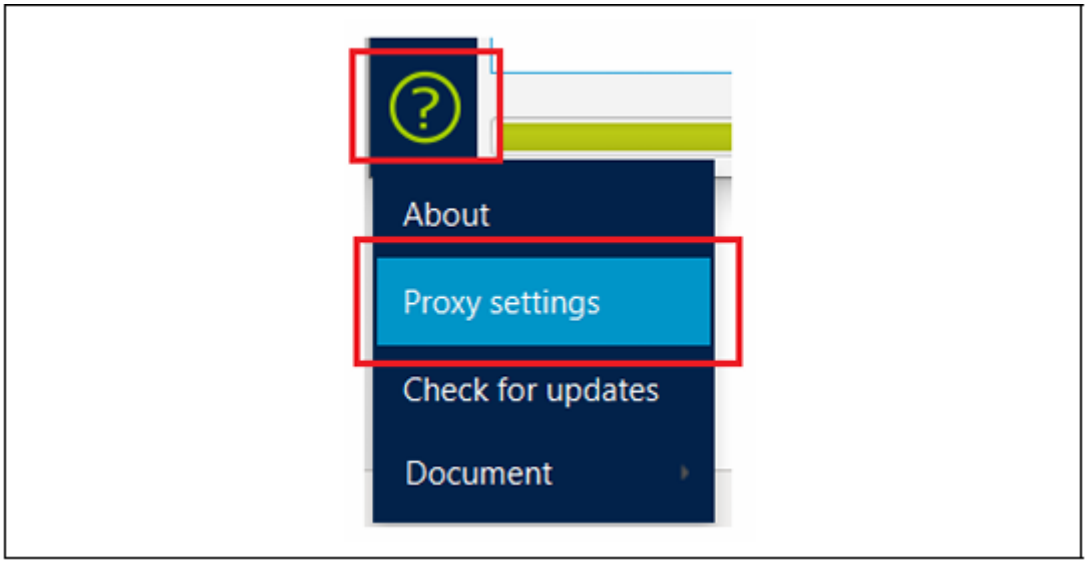


Fig. 2.7: Proxy settings submenu

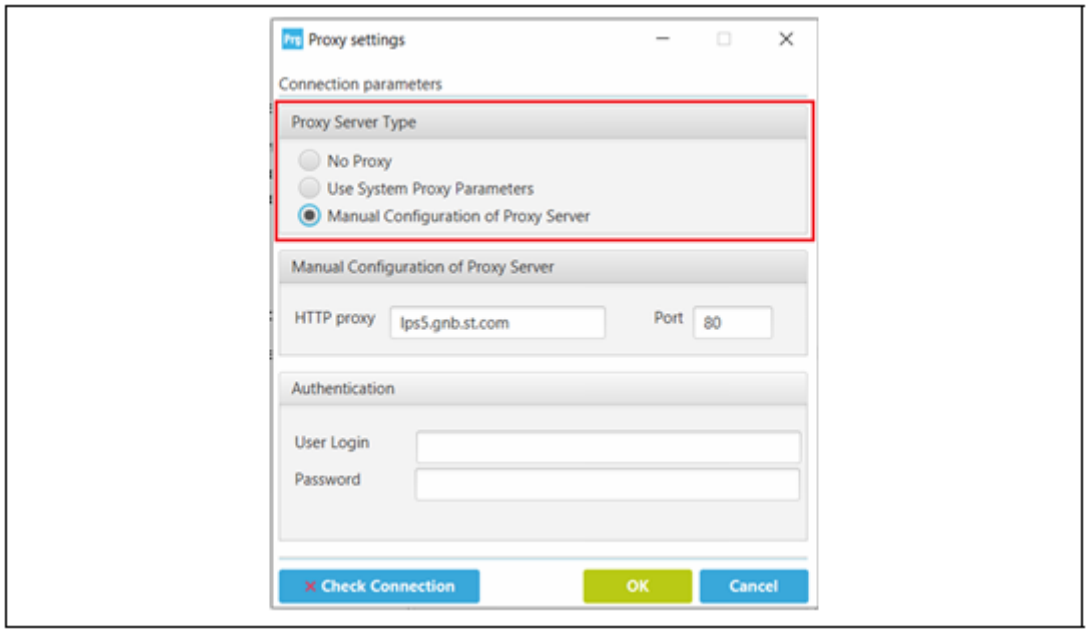


Fig. 2.8: Proxy settings window

The status of connection check is displayed in the “check Connection” button: - A green icon indicates success (left side of the figure below). - A red icon indicates that the connection is down (right side of the figure below).

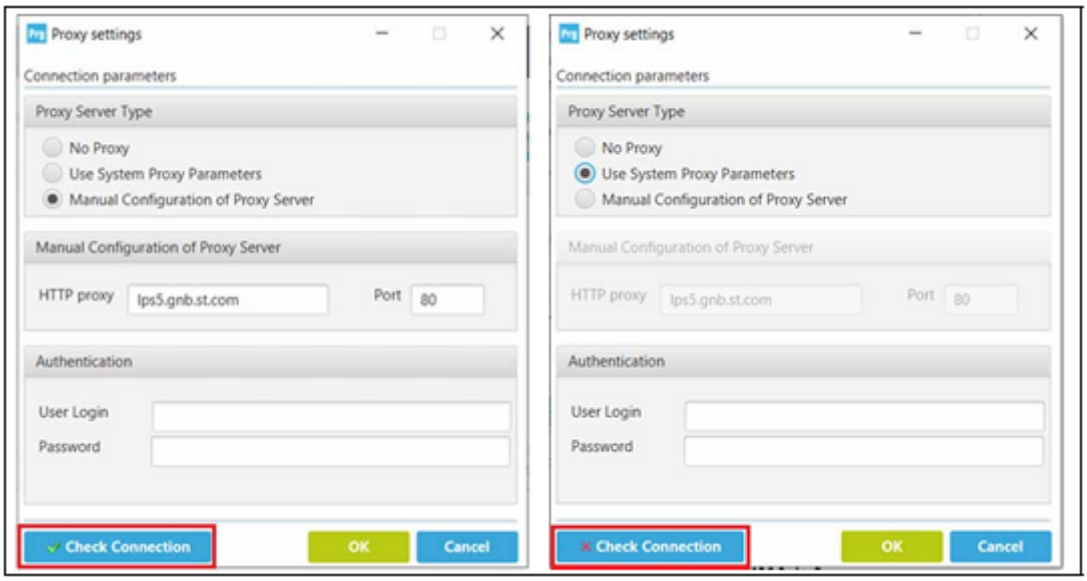


Fig. 2.9: Successful connection check

Check for updates

The user can launch the update using the Updater window opened with the submenu “Software update” added in the Help button. Both the STM32CubeProgrammer tool and the components associated with the version currently in use can be updated.

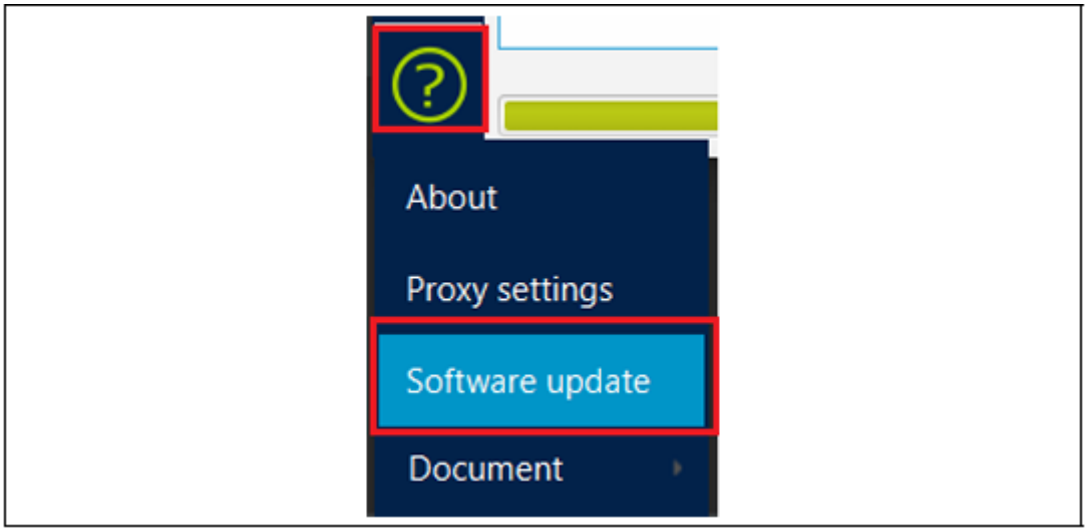


Fig. 2.10: Check for updates submenu

If there is a new version of the tool and/or of the components is available, an update button appears in the main menu (see the figure shown below).



Fig. 2.11: Hyperlink button of new version available

Note

If the user has already updated the STM32CubeProgrammer and the components patch, the hyperlink button is no longer displayed at startup. If a new version of the tool and/or of the components patch is available, the user can make updates through the updater window.

This window displays:

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

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

2.2 Additional links

ST X profile
STM32 Sidekick
YouTube
Facebook
Product longevity

3 User manual

3.1 Graphical User interface

3.1.1 MCU graphical user interface

Main window

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

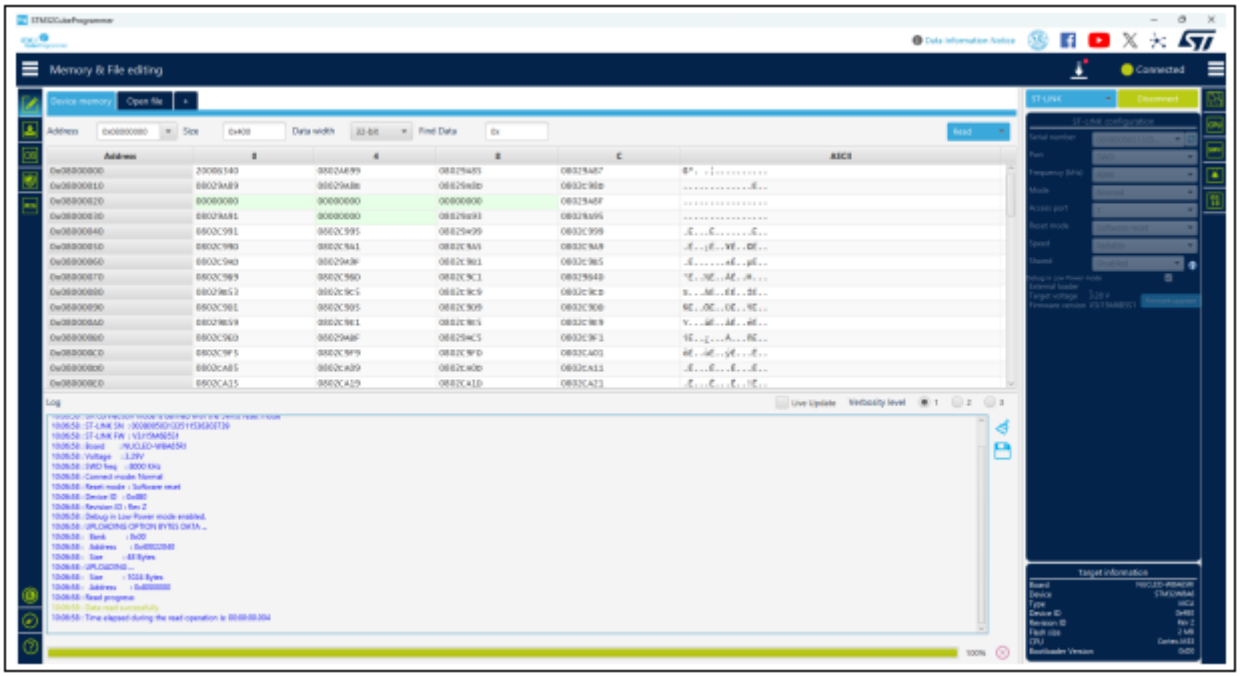


Fig. 3.1: STM32CubeProgrammer main window

Main menu

This menu allows the user to switch between the three main panels of the Memory and file editing, Erasing & programming, and Option bytes tools. The other panels are displayed according to the used device. By clicking on the Hamburger menu (the three-line button) on the top left corner, the menu expands and displays the textual description shown.



Fig. 3.2: Expanded main menu

Log panel

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

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

If the selected language of the tool is not English, two log files are saved in the same location, one in English, the other in the selected language.

Progress bar

The progress bar visualizes the progress of any operation or transaction done by the tool (for example, Read, Write, Erase). Ongoing operations can be aborted by pressing the “Stop” button in front of the progress bar.

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/J-Link debug probe, or the bootloader interface over UART, USB, SPI, CAN, or I2C).

With the refresh button the user can check the available interfaces connected to the PC. If this button is pressed while the ST-LINK/J-Link interface is selected, the tool checks the connected ST-LINK/J-Link probes, and lists them in the Serial numbers combo box. If the UART interface is selected, it checks the available communication ports of the PC, and lists them in the Port combo box. If the USB interface is selected, it checks the USB devices in DFU mode connected to the PC, and lists them in the Port combo box. Each interface has its own settings, to set before connection.

ST-LINK settings



Fig. 3.3: ST-LINK configuration panel

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

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

- Frequency: the JTAG or SWD clock frequency
- Access port: selects the access port to connect to. Most of the STM32 devices have only one access port, which is Access port 0.
- **Mode:**

  - Normal: with “Normal” connection mode, the target is reset then halted. The type of reset is selected using the “Reset Mode” option.
  - Connect under reset: this mode enables connection to the target using a reset vector catch before executing any instructions. This is useful in many cases, for example when the target contains a code that disables the JTAG/SWD pins.
  - Hot plug: enables connection to the target without a halt or reset. This is useful for updating the RAM addresses or the IP registers while the application is running.
  - Power down: used to put the target in debug mode, even if the application has not started since the target power-up. The hardware reset signal must be connected between ST-LINK and the target. This feature can be not fully effective on some boards (MB1360, MB1319, MB1361, MB1355) with STMPS2141 power switch.
  - hwRstPulse: the tool generates a reset pulse, and then connects to the target. This connection mode does not prevent application launch before connection. It is used in some devices where under mode is not available (such as STM32WB0x and STM32WL33)
- **Reset mode:**
  - Software system reset: resets all STM32 components except the Debug via the Cortex-M application interrupt and reset control register (AIRCR).
  - Hardware reset: resets the STM32 device via the nRST pin. The RESET pin of the JTAG connector (pin 15) must be connected to the device reset pin.
  - Core reset: resets only the Cortex-M via the AIRCR(a).
- **Speed (Cortex-M33 only):**
  - Reliable: allows the user to connect with a slow mode.
  - Fast: allows the user to connect with a fast mode.
- Shared: enables shared mode allowing connection of two or more instances of STM32CubeProgrammer or other debugger to the same ST-LINK probe.
- Debug in low-power mode (STM32U5/WB/L4 series only): sets the bits in DBGMCU_CR to 1.
- External loader: displays the name of the external memory loader selected in the “External loaders” panel accessible from the main menu (Hamburger menu).
- Target voltage: target voltage is measured and displayed.
- Firmware version: displays the ST-LINK firmware version. The firmware upgrade button allows you to upgrade the ST-LINK firmware.

J-Link settings

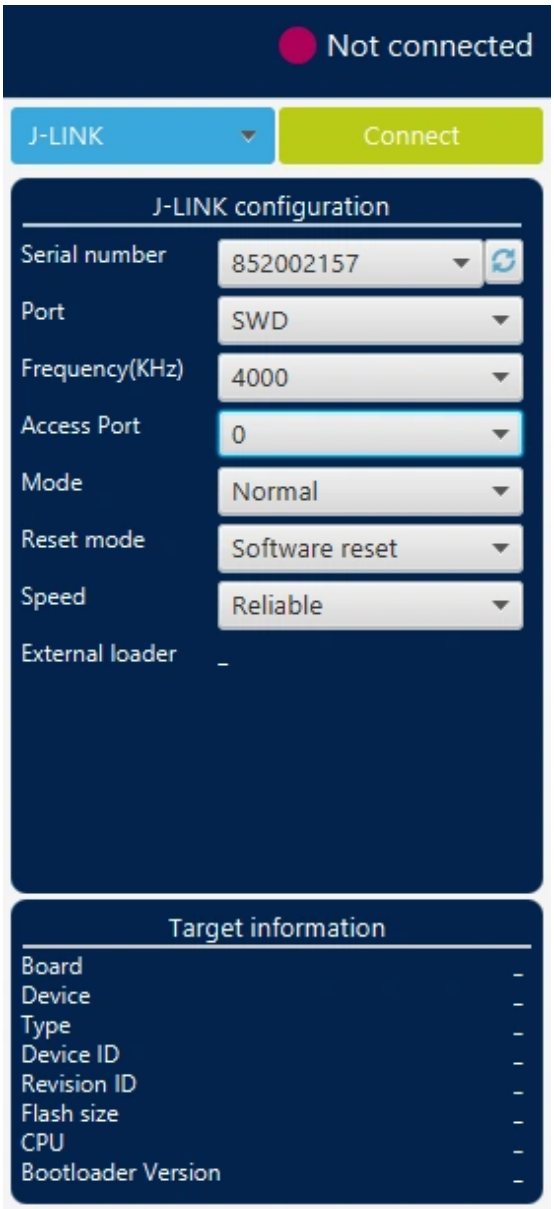


Fig. 3.4: J-Link configuration panel

- **Serial number: this field contains the serial numbers of all connected J-Link probes.**
  The user can choose one of them, based on its serial number.
- Port: J-Link probe supports only SWD debug protocol.
- Frequency: SWD clock frequency (only 4000 kHz is available).
- Access port: selects the access port to connect to. Most of the STM32 devices have only one access port (Access port 0).
- **Mode:**
  - Normal: the target is reset, then halted. Selected using the “Reset Mode” option.
  - Connect under reset: enables connection to the target using a reset vector catch before executing any instructions. Useful in many cases, for example when the target contains a code that disables the SWD pins.

  - Hot plug: enables connection to the target without a halt or reset. Useful to update 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 Cortex-M application interrupt and reset control register (AIRCR).
  - Hardware reset: resets the STM32 device via the nRST pin. The RESET pin of the SWD connector (pin 15) must be connected to the device reset pin.
  - Core reset: resets only the Cortex-M via the reset control register(a).
- **Speed (Cortex-M33 only):**
  - Reliable: allows the user to connect with a slow mode.
  - Fast: allows the user to connect with a fast mode.
- External loader: displays the name of the external memory loader selected in the External loaders panel accessible from the main menu (Hamburger menu).

UART settings

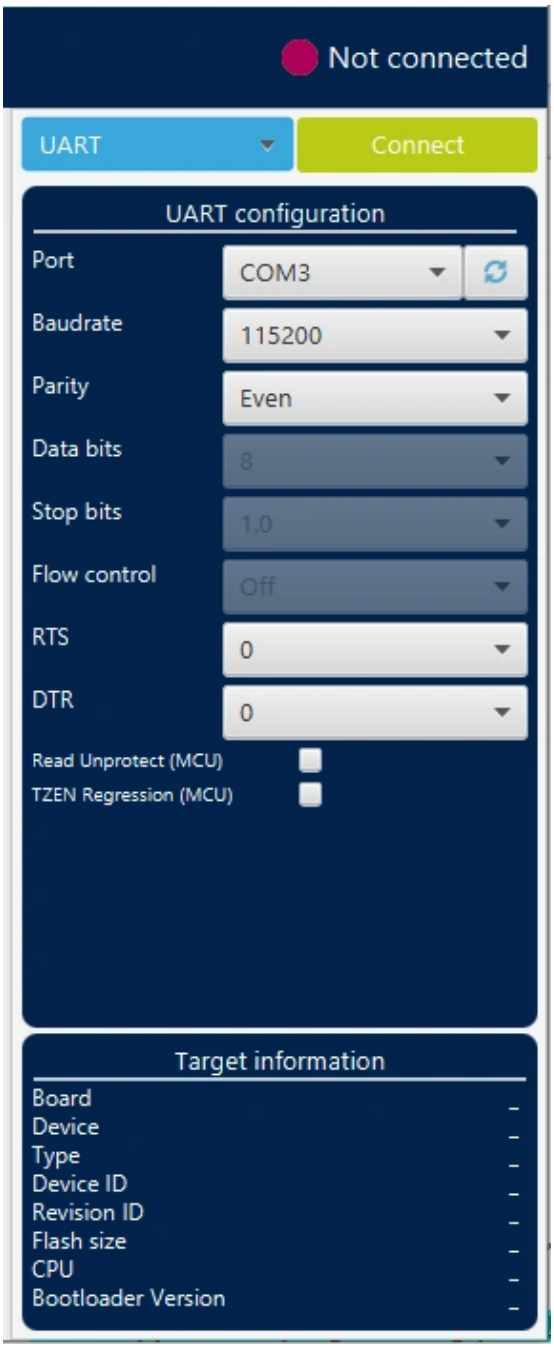


Fig. 3.5: UART configuration panel

Note

Core reset does not exist for Cortex-M33, Cortex-M55 and Cortex-M85. The STM32 must boot in bootloader mode using boot pins and/or the option bits. Check AN2606 “STM32 microcontroller system memory boot mode” , available on www.st.com, for more information on the STM32 bootloader.

- 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.
- Baudrate: selects the UART baud rate.
- Parity: selects the parity (even, odd, none), must be “even” for all STM32 devices.
- Data bits: must be always 8, only 8-bit data is supported by the STM32.
- Stop bits: must be always 1, only 1-bit stop is supported by the STM32.
- Flow control: must be always off.
- RTS (request to send): sets the COM RTS pin to either high or low level.
- DTR (data terminal ready): sets the COM DTR pin to either high or low level.

USB settings

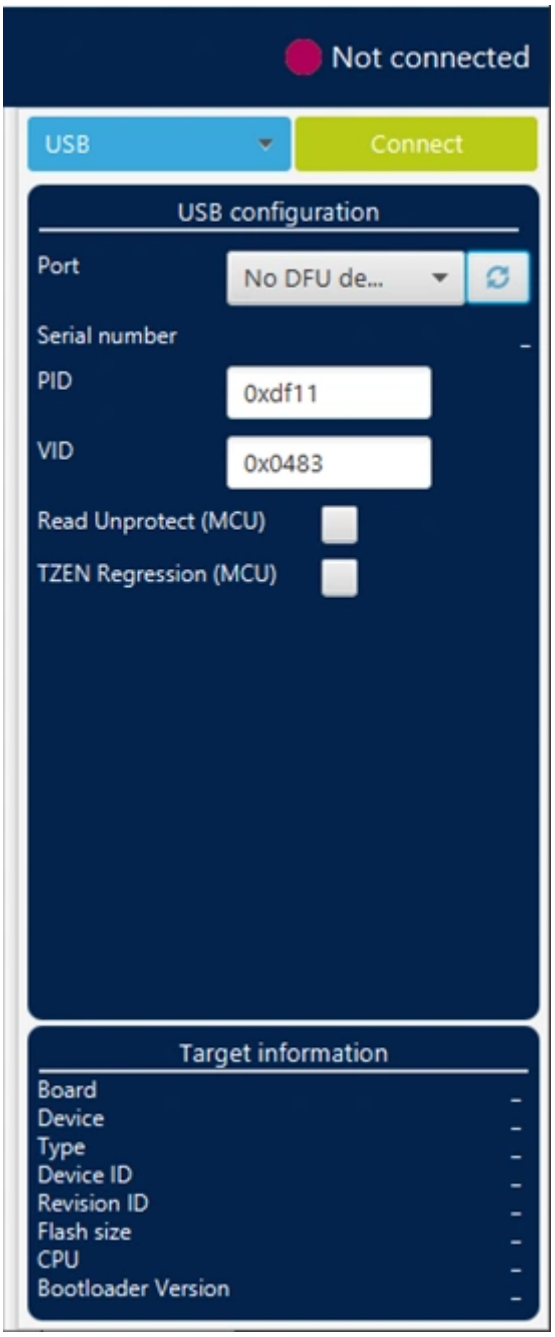


Fig. 3.6: USB configuration panel

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

Note:

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

Note:

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 shown in the figure below.



Fig. 3.7: Target information panel

SPI settings

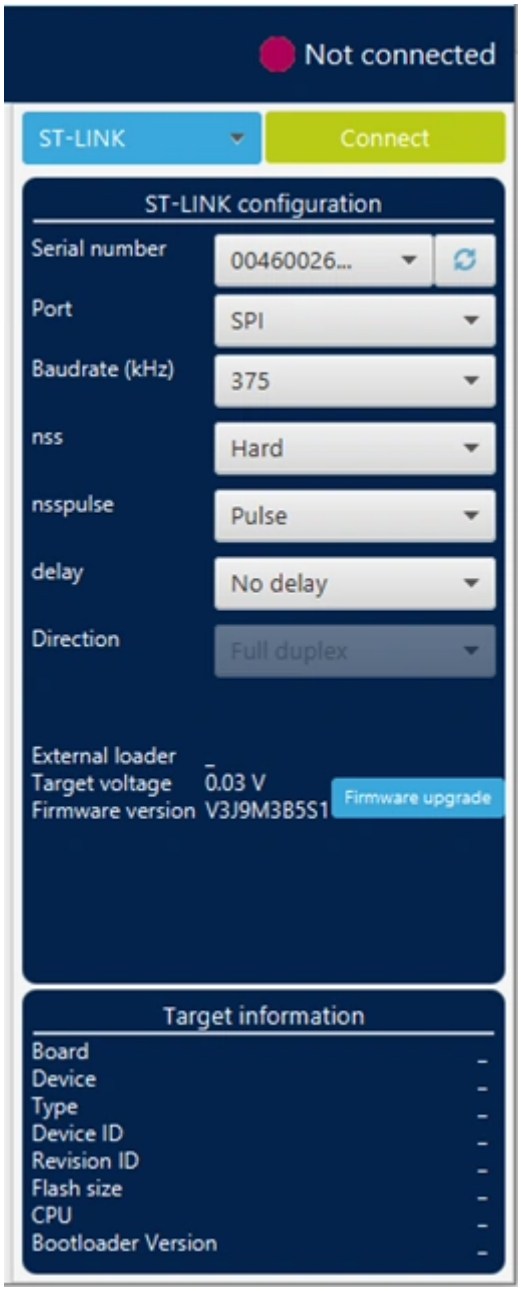


Fig. 3.8: SPI configuration panel

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

I2C settings

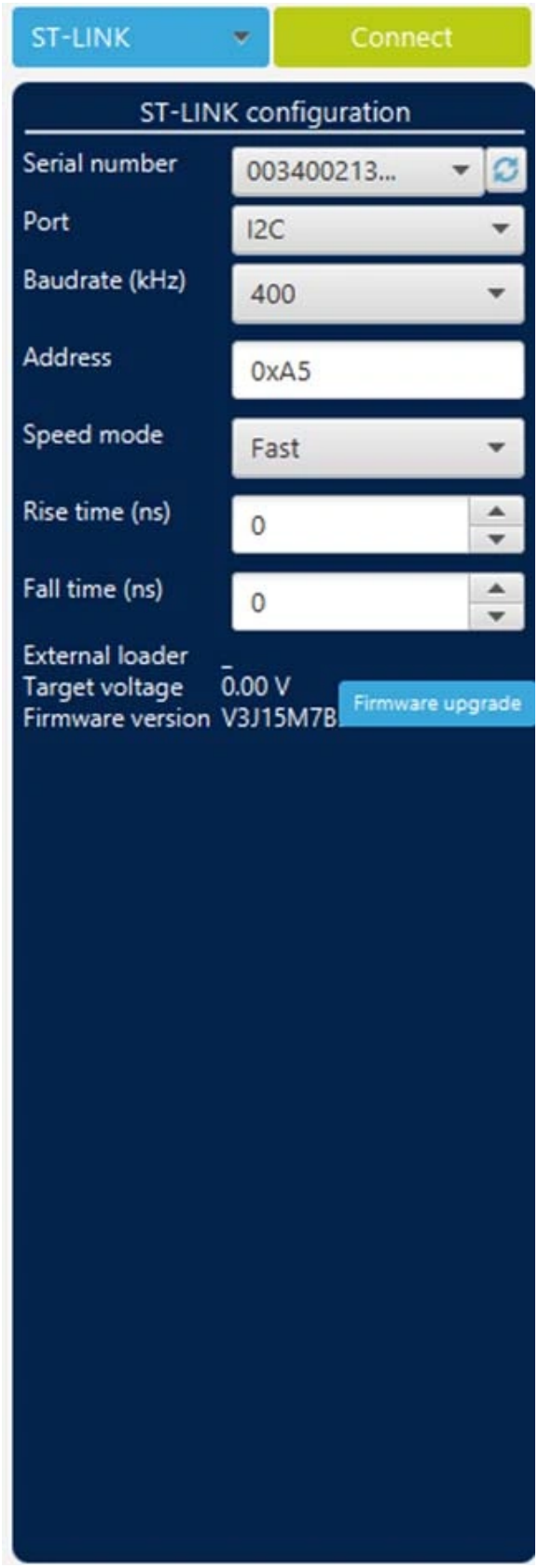


Fig. 3.9: I2C configuration panel

- Serial number: this field contains the serial numbers of all connected ST-LINK-V3 probes when using the I2C bootloader.
- Port: selects the I2C devices connected to the PC. You can use the refresh button to recheck the available devices.
- Baudrate: selects the I2C baud rate.
- Address: adds the address of the slave bootloader in hex format.
- Speed mode: selects the speed mode of the transmission Standard or Fast.
- Rise time: chooses values according to Speed mode, 0-1000 (STANDARD), 0-300 (FAST).
- Fall time: chooses values according to Speed mode, 0-300 (STANDARD), 0-400 (FAST).

Memory and file edition

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

Reading and displaying target memory

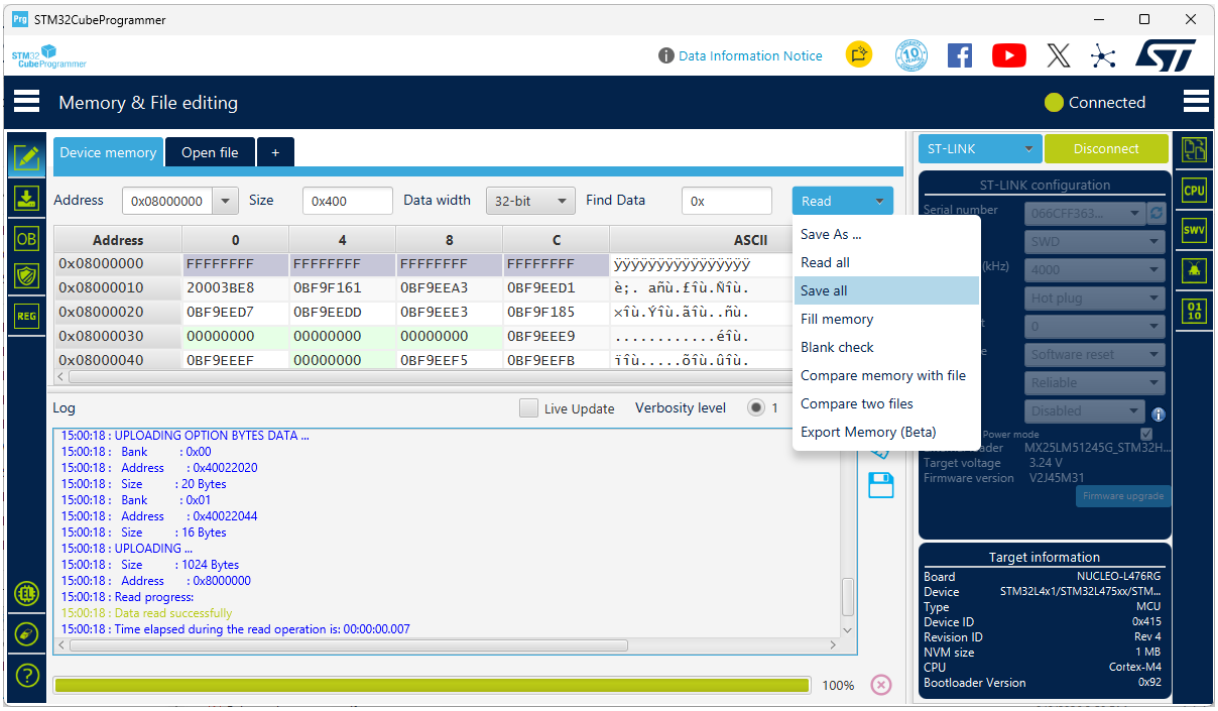


Fig. 3.10: Device memory tab

After connecting to a target, use this panel to read its memory. Specify the address and size, then click **Read** in the top-left corner. The data can be displayed in 8-bit, 16-bit, or 32-bit format using the **Data width** combo box. The device memory tab provides these actions:

- **Read all** reads all flash memory.
- **Save As...** saves the device memory content in a `.bin`, `.hex`, or `.srec` file.
- *Save all*: saves Internal and/or External NVM content to a single HEX file.
- *Fill memory*: fills memory with a specified value or pattern.
- *Blank check*: checks whether a memory region is blank.
- *Compare memory with file*: compares target memory with a file.
- *Compare two files*: compares the content of two files.

- *Export Memory (Beta)*: exports selected memory content to a HEX file.

Multiple device memory tabs can be opened to display different target memory locations. Click the + tab to open a contextual menu and add a **Device memory** tab or open a file in a **File** tab.

For information about saving internal and external NVM contents to a single `.hex` file, see *Save all memory*.

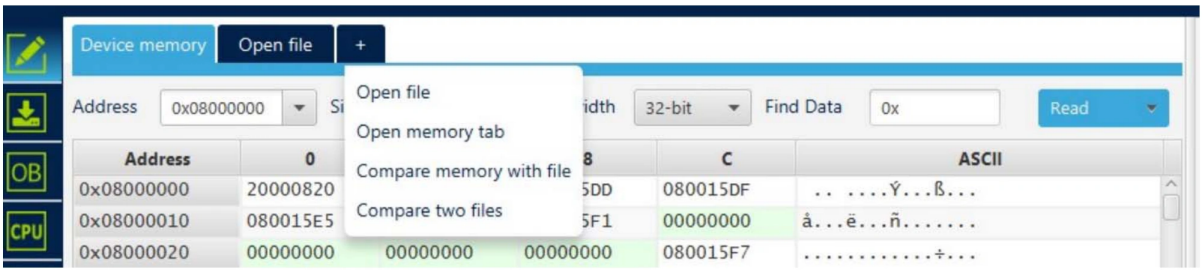


Fig. 3.11: Contextual menu

Reading and displaying a file

To open a file, click the + tab and select **Open File**, as shown in the figure above. The following formats are supported:

- Binary (`.bin`)
- ELF (`.elf`, `.axf`, `.out`)
- Intel HEX (`.hex`)
- Motorola S-record (`.Srec`)

After the file is parsed, it opens in a dedicated tab named after the file. The **Size** field shows the file size. The **Address** field shows the start address for HEX, S-record, and ELF files; for binary files, it defaults to `0`. You can change the address to view the file content from a different offset.

Use the tab's contextual menu or action button to perform file operations:

- **Download** programs the file to the target. For a binary file, specify the target address in the **Address** menu.
- **Verify** checks whether the file was downloaded correctly.
- Save the file in another supported output format (`.bin`, `.hex`, or `.srec`).

As in the **Device memory** tab, use the **Data width** combo box to display the file contents in 8-bit, 16-bit, or 32-bit format.

Memory programming and erasing

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

Note

STM32CubeProgrammer is able to write on aligned memory regions. Flash memory imposes a data alignment described in the STM32 reference manual. As an example, for STM32U5 devices, the reference manual indicates that this MCU supports "137 bits wide data read and write (128 effective bits plus 9 ECC bits)", which means that data must be aligned on 16 bytes.

Internal flash memory programming

All modifications involving the PC register need a board reset.

Memory programming

To program a memory, go through the following steps:

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

The progress bar at the bottom of the window shows the progress of the operation.

The user can also edit the memory through the displayed memory grid in the Memory & File Editing tab by double-clicking on the ASCII field or on one of the memory grid cells. The value is padded if necessary.

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 (flash or EEPROM), select them in the first column, and then click on the "Erase selected sectors" button.

Select "Full Flash memory checksum" to enable checksum calculation at the end of the download file.

The "Full chip erase" button erases the whole memory.

External flash memory programming

To program an external memory connected to the microcontroller via any of the available interfaces, such as SPI, FMC, FSMC, QSPI, or OCTOSPI, you need an external loader.

STM32CubeProgrammer is delivered with external loaders for several STM32 evaluation and discovery boards. Refer to the `bin/ExternalLoader` directory. If you need to create a new external loader, see the next section for more details.

To program an external memory, select one or more external loaders from the "ExternalLoader" panel. They are then used by the tool to read, program, or erase external memories. Once selected, the external loader or loaders are used for any memory operation in their memory range.

The "External flash erasing" tab on the right of the "Erasing and Programming" panel displays the memory sectors for each selected loader, and enables sector or full-chip erase.

Developing customized loaders for external memory

Based on the examples available on ST GitHub at https://github.com/STMicroelectronics/stm32-external-loader, users can develop their custom loaders for a given external memory.

The programming mechanism is the same one used by the STM32 ST-LINK Utility tool. Any flash loader developed for use with the ST-LINK Utility is compatible with STM32CubeProgrammer and can be used without any modification.

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

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

Note

Some functions are mandatory and cannot be omitted. Refer to the function descriptions in `Loader_Src.c`. Linker or scatter files must not be modified.

After building the external loader project, an ELF file is generated. The extension of this file depends on the toolchain used: `.axf` for Keil, `.out` for EWARM, and `.elf` for TrueSTUDIO or any GCC-based toolchain.

The extension of the ELF file must be changed to `.stldr`, and the file must be copied under the `bin/ExternalLoader` directory.

Loader_Src.c file

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

- `Init`

  Defines the GPIO pins used to connect the external memory to the device, and initializes the clocks of the used IPs. Returns 1 if successful, and 0 if it fails.

- `Write`

  Programs a buffer defined by an address in the RAM range. Returns 1 if successful, and 0 if it fails.

- `SectorErase`

  Erases the specified memory sectors. Returns 1 if successful, and 0 if it fails. `StartAddress` equals the address of the first sector to be erased, and `EndAddress` equals the address of the last sector to be erased.

Note

`SectorErase` is not used in the case of an external SRAM loader.

The functions mentioned above must be defined 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:

- Calls `Init` to initialize the interface, such as QSPI, and the flash memory.
- Calls `SectorErase()` to erase the needed flash memory sectors.
- Calls `Write()` to program the memory.

It is possible to define additional functions:

- `Read`

  The `Read` function is used to read a specific range of memory, and returns the data in a RAM buffer. It returns 1 if successful, and 0 if it fails.

Note

For Quad-SPI and Octo-SPI memories, the memory-mapped mode can be defined in the `Init` function. In that case, the `Read` function is useless, because data can be read directly from the JTAG or SWD interface.

- `Verify`

  The `Verify` function is called when selecting the “verify while programming” mode. This function checks whether the programmed memory matches the buffer defined in RAM. It returns a `uint64` value 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.

- `MassErase`

  The `MassErase` function erases the full memory. It returns 1 if successful, and 0 if it fails.

- A checksum function.

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

```
int Init (void)
int Write (uint32_t Address, uint32_t Size, uint8_t* buffer)
int SectorErase (uint32_t StartAddress, uint32_t EndAddress)
int Read (uint32_t Address, uint32_t Size, uint16_t* buffer)
uint64_t Verify (uint32_t FlashAddr, uint32_t RAMBufferAddr, uint32_t Size)
int MassErase (void)
```

Dev_Inf.c file

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

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

External memory programming with bootloader interfaces on GUI

This feature is supported by STM32H7Rx/7Sx products. Go through the sequence outlined below to successfully program the external memory using bootloader interfaces:

1. **Ensure that there is no external loader already selected before connecting** the board through STM32CubeProgrammer.
2. **Choose the bootloader interface. USB and USART are currently supported, then** connect.
3. **Select the external loader operating on your board. Only one can be** selected.
4. The open bootloader is loaded.
5. Write, read, and erase operations can be performed.

> **Note**
>
> If the board is disconnected, a hardware reset is needed to connect again to the interface. Go through step 1 again to perform another external memory programming via bootloader.

Option bytes

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

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

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

Synthetic option bytes view

The user has two ways to display and edit the option bytes:

- Detailed (default) view: contains the name, the value, and the description of the option bytes.

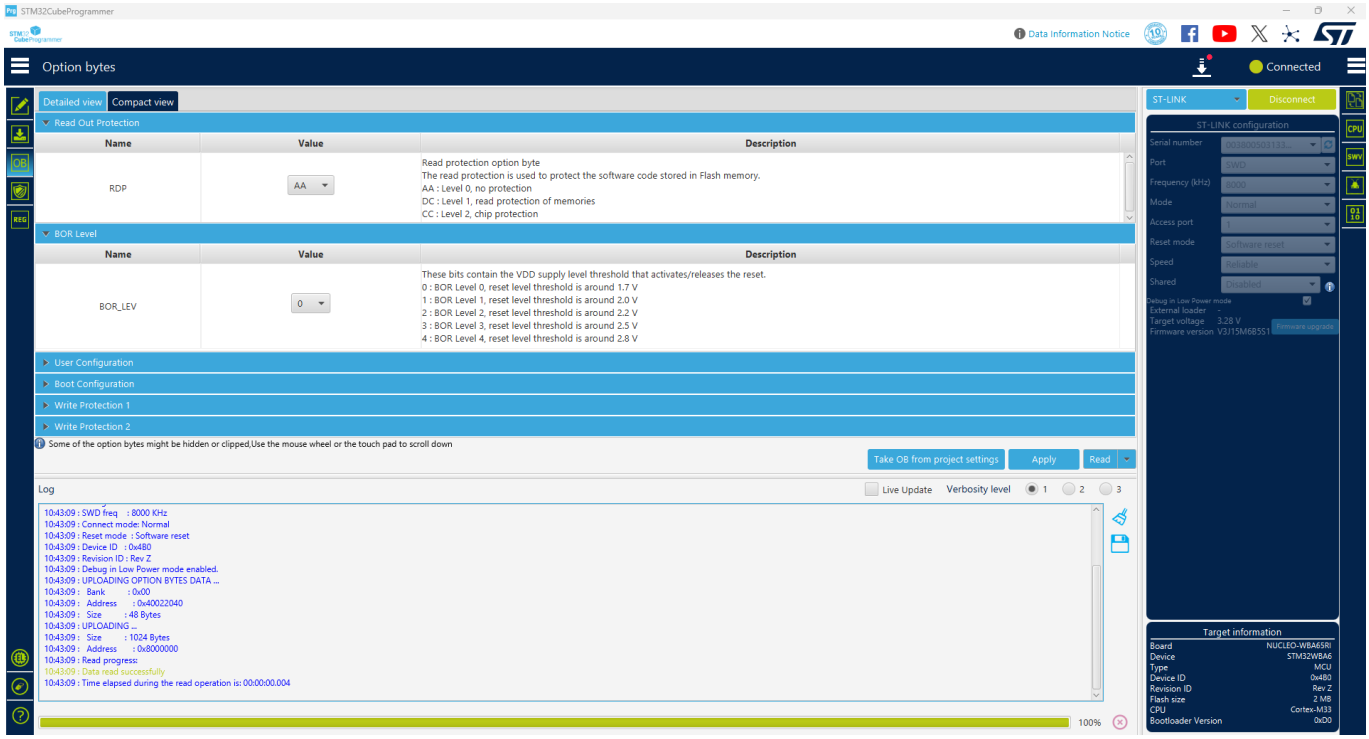


Fig. 3.12: Option bytes - Detailed view

- Compact view: presents a lower detail level, tailored for expert users who require a compact overview. It excludes the description field of each option byte, and focuses on presenting a list with their values. The description is displayed as a tooltip.

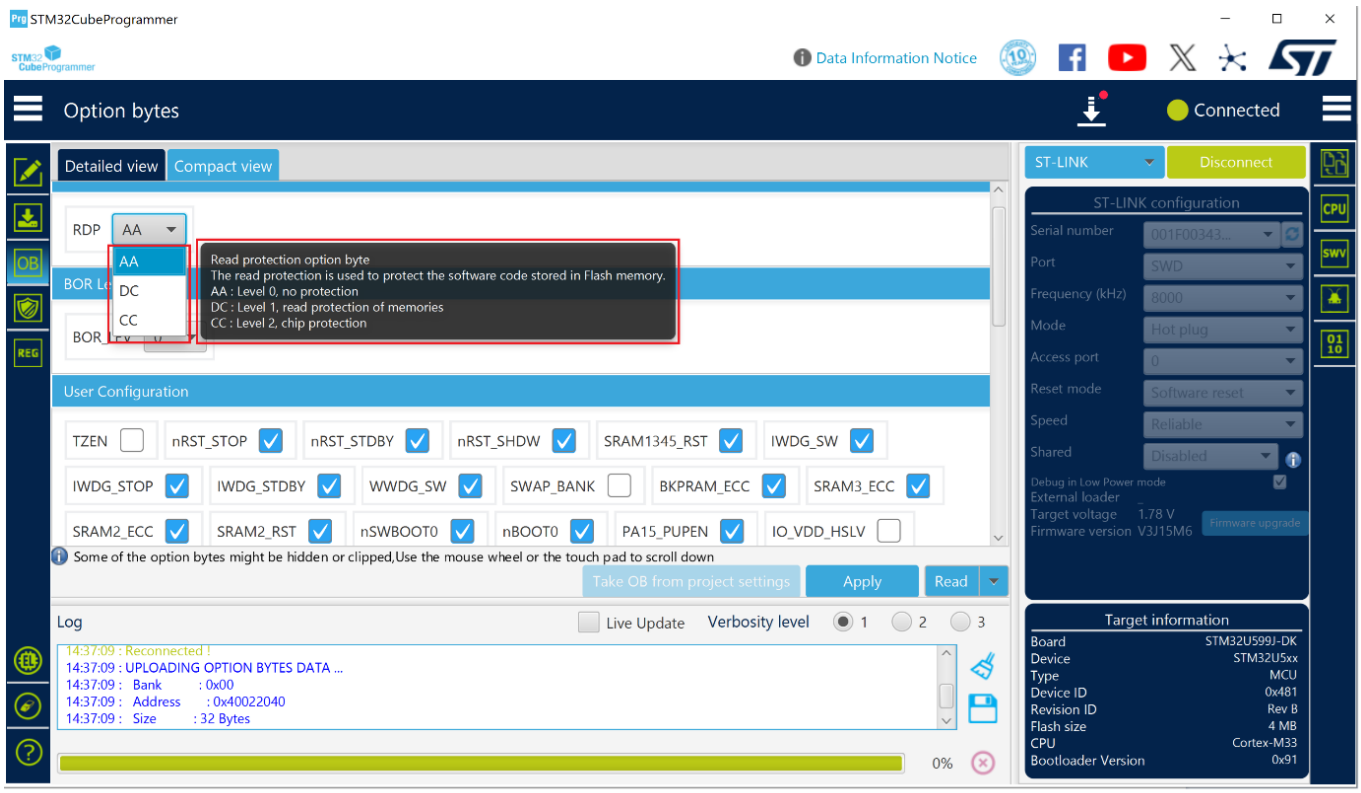


Fig. 3.13: Option bytes - Compact view

Recovery button

Option bytes can be reset to their default values as specified in the reference manual. To use this feature (available for the STM32WBA5/6, STM32U3, STM32U5, and STM32H5 series), select the Option Bytes tab, then choose the “Factory reset” button.

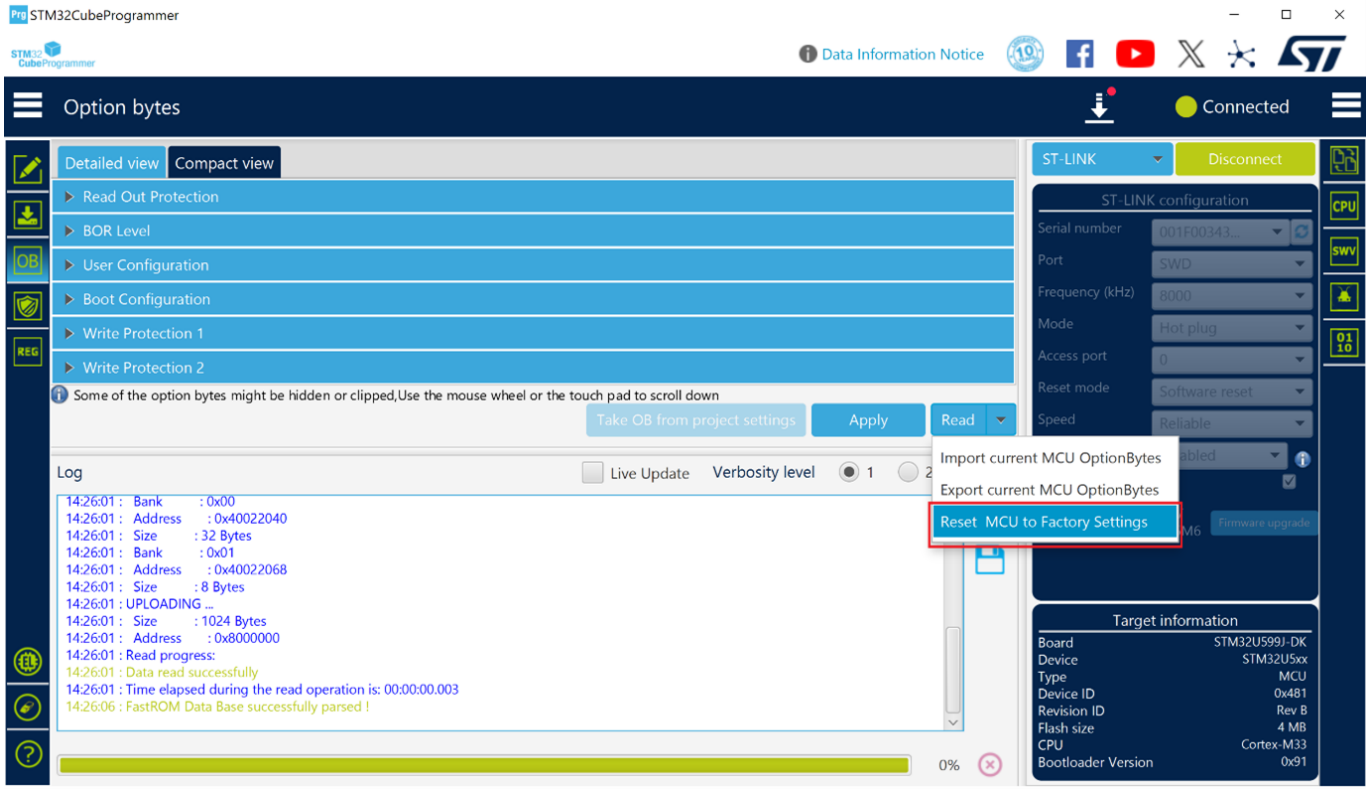


Fig. 3.14: Reset to factory settings submenu

After selecting the Reset MCU to Factory Settings sub-menu, a new window appears, displaying a list of operations to perform along with their status.

Clicking the “Start Factory Reset” button initiates the reset operation, and displays all performed operation and their status:

- PENDING: the operation is currently pending and awaiting further user actions.
- PASSED: the operation was successfully passed.
- FAILED: the attempt failed due to detected issue.
- ERROR: an error occurred during the process, causing it to stop unexpectedly.
- ABORTED: the operation was aborted before completion, halting all progress.

Note

The recovery button is supported only via STLINK.

Export/import option bytes

Users can export option bytes using a dedicated split button that can be used also to read them. The export format is JSON, ensuring compatibility and ease of use in data manipulation.

Upon exporting, users are prompted to select a desired name and location for the saved file, offering flexibility in file management.

The JSON configuration files can be seamlessly imported to other targets, provided they share the same device ID, facilitating device configuration consistency. Imported settings are displayed for user review before application. This ensures that users have full control over the changes. The settings are applied only if the user is satisfied with the proposed configuration.

If there are discrepancies, such as missing or surplus values, the tool displays a warning, providing details so that the user can take informed decisions about how to proceed.

MCU unlock (specific for the STM32WL series)

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

Debug authentication default configuration

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

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

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

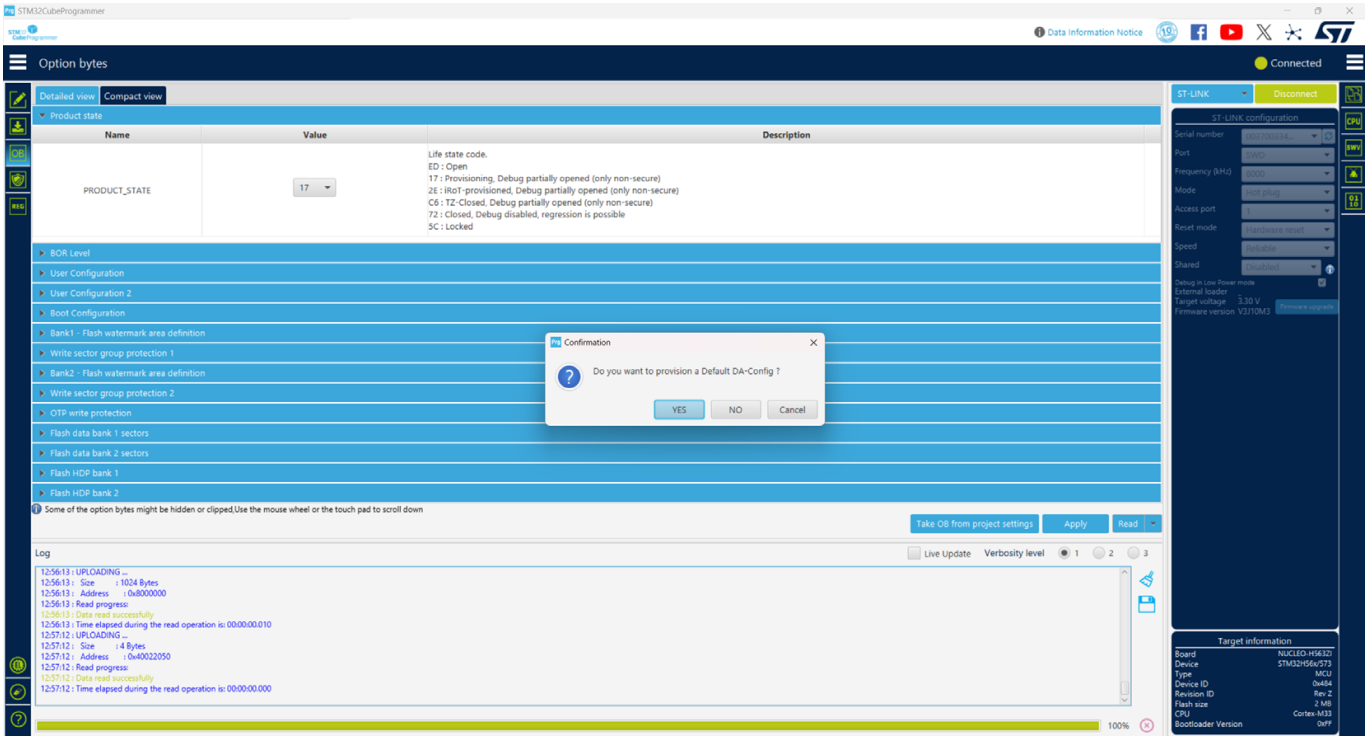


Fig. 3.15: DA default configuration when switching product state to provisioning

When switching PRODUCT_STATE from 0xED (open) to values different from 0x17 (provisioning), the user is asked to pass by the provisioning state first (shown in the following figure).

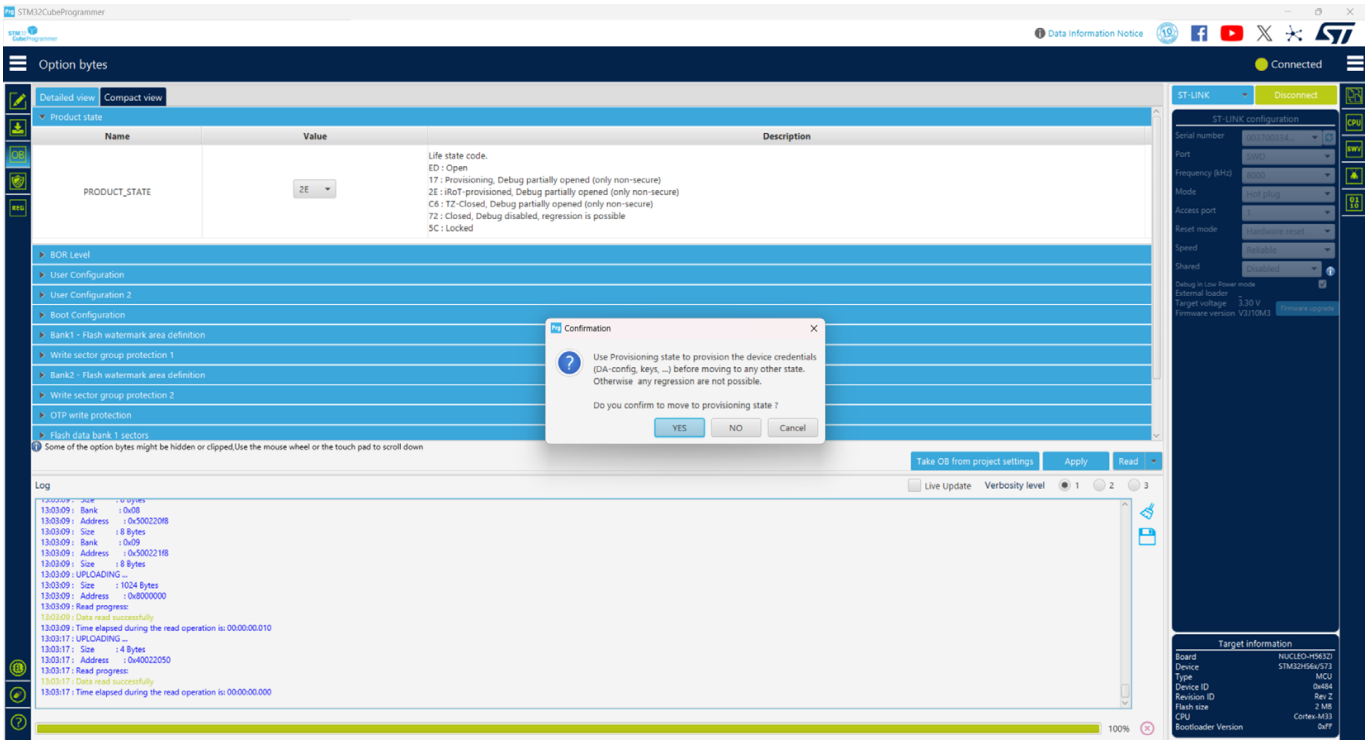


Fig. 3.16: Configuration when switching product state to values different from 0x17

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

Debug authentication configuration (STM32H503 only)

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

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

When switching PRODUCT_STATE from 0xED (open) to values different from 0x17 (provisioning), the tool checks if there is a password provisioned in OTP. If not, the user is asked to pass by the provisioning state first. Refer to the DA default configuration figure shown earlier for GUI details.

Automatic mode

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

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

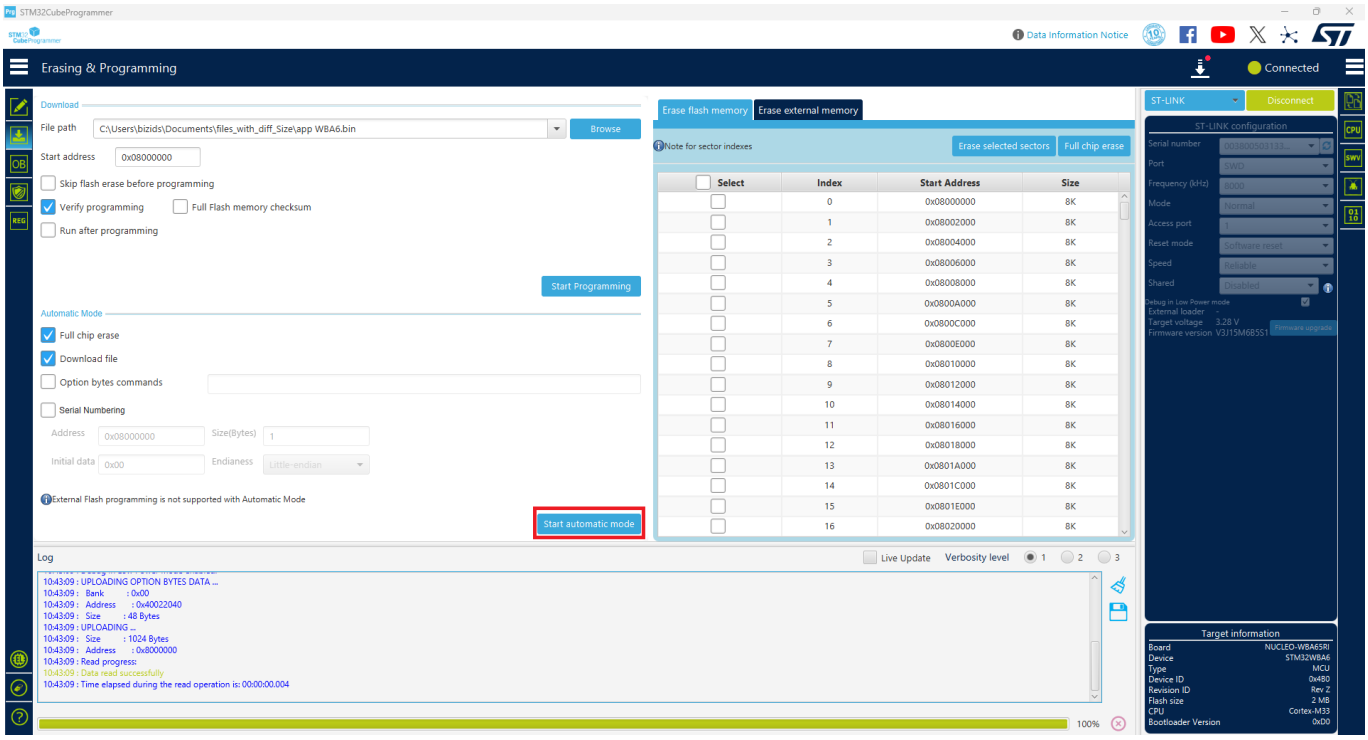


Fig. 3.17: Automatic mode in Erasing & Programming window

All automatic mode traces are indicated in the Log panel, to show the process evolution and user intervention messages.

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 no Download option is performed.
- **If the Option bytes commands is checked, the text field is activated, then** the user can insert option bytes CLI commands, and make sure that there are no white spaces at the beginning:

  `-ob [OptionByte=value] [OptionByte=value] [OptionByte=value] ...`

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

Log messages

- `Starting Automatic Mode...`

  Indicates that the system successfully entered the automatic process.

- `More than one ST-LINK probe detected! Keep only one ST-LINK probe!`

The automatic mode cannot be used if more than one ST-LINK probe is connected to the computer when using JTAG/SWD interfaces. A message is displayed, asking the user to keep only one ST-LINK probe connected to continue using this mode.

- `More than one ST-LINK Bridge detected! Keep only one ST-LINK Bridge!`

  The automatic mode cannot be used if more than one ST-LINK bridge is connected to the computer when using bootloader interface SPI/CAN/I2C interfaces. A message is displayed, asking the user to keep only one ST-LINK bridge connected to continue using this mode.

- `More than one ST-LINK USB DFU detected! Keep only one USB DFU!`

  The automatic mode cannot be used if more than one USB DFU is connected to the computer when using USB bootloader interface. A message is displayed, asking the user to keep only one USB DFU connected to continue using this mode.

- `More UART ports detected than last connection!`

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

- `Please disconnect device and connect the next...`

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

- `Waiting for device...`

  Once the connection to the previous device is correctly lost, the system keeps searching for a new device.

- `Automatic Mode is stopped.`

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

Figure 32. Algorithm

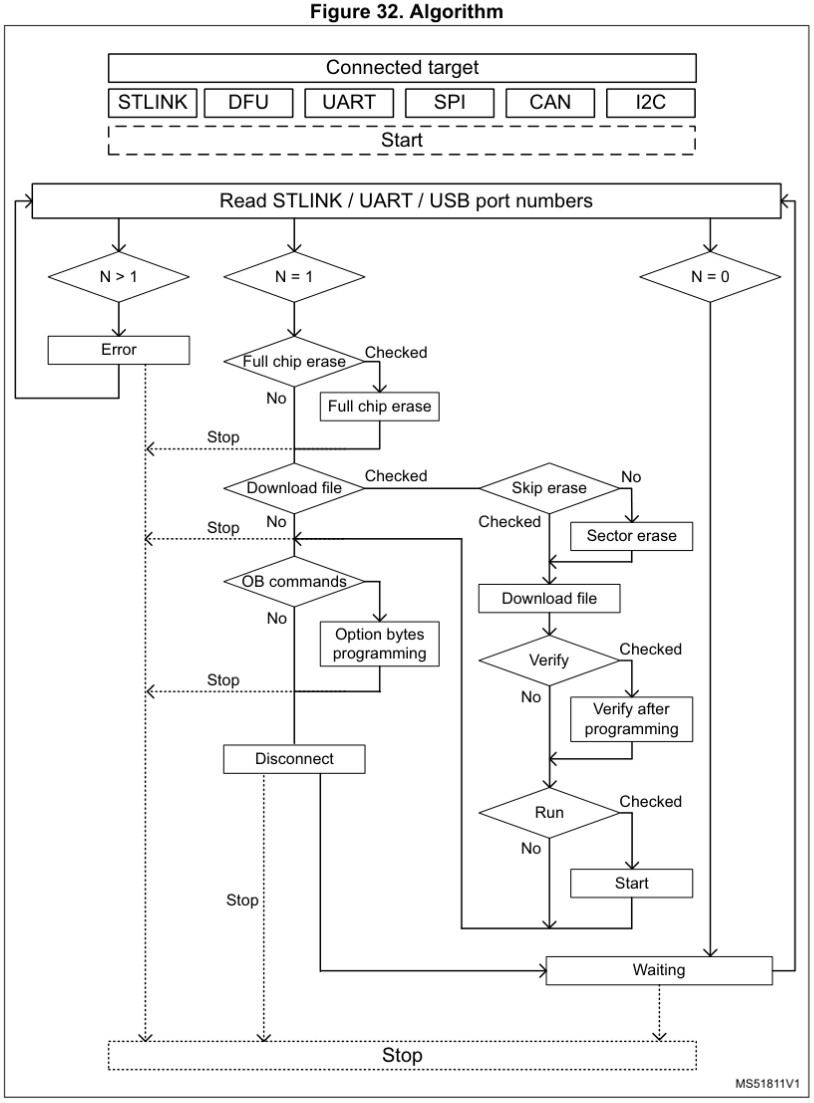


Fig. 3.18: Algorithm

Serial numbering

The automatic mode can be performed with this feature enabled, granting the user the capability to assign unique identifiers to targets in a sequential manner, for tracking and reference purposes. Four fields are needed:

1. Address: contains the serial numbers.
2. **Initial data: hexadecimal number, incremented sequentially by 1 for each** target in the automatic mode.
3. Size: the number of bytes used for the serial numbering.
4. **Endianness: the user can choose little- or big-endian for the increment** (default is little-endian).

When the maximum serial number for the input size is reached, the automatic mode stops, and the user is asked to click the Stop button.

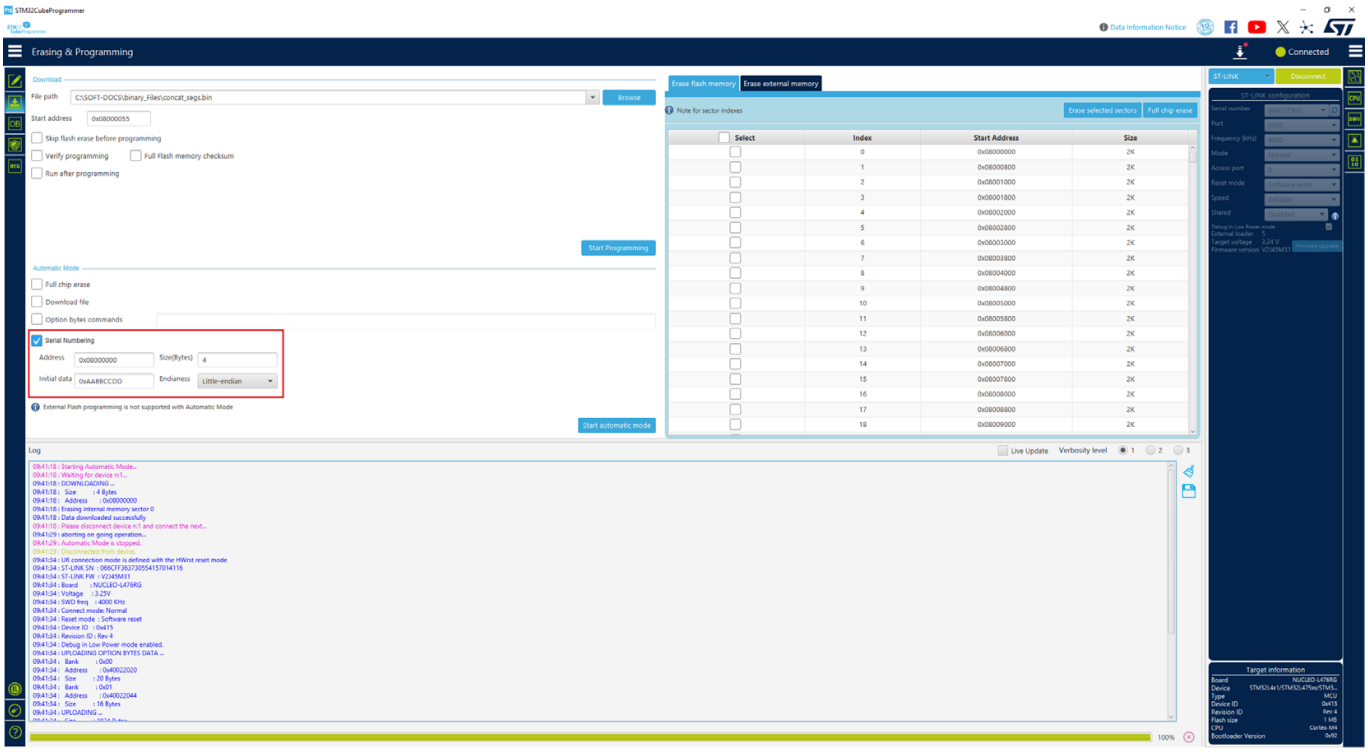


Fig. 3.19: Automatic mode with serial numbering

In application programming (IAP/USBx)

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

Note

Option byte, memory edition, and sector erase are not available with IAP/USBx.

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

Flash the wireless stack using the graphical interface

FUS/stack upgrade

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

Note

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

To download WB stacks and FUS from www.st.com, press on the logo.

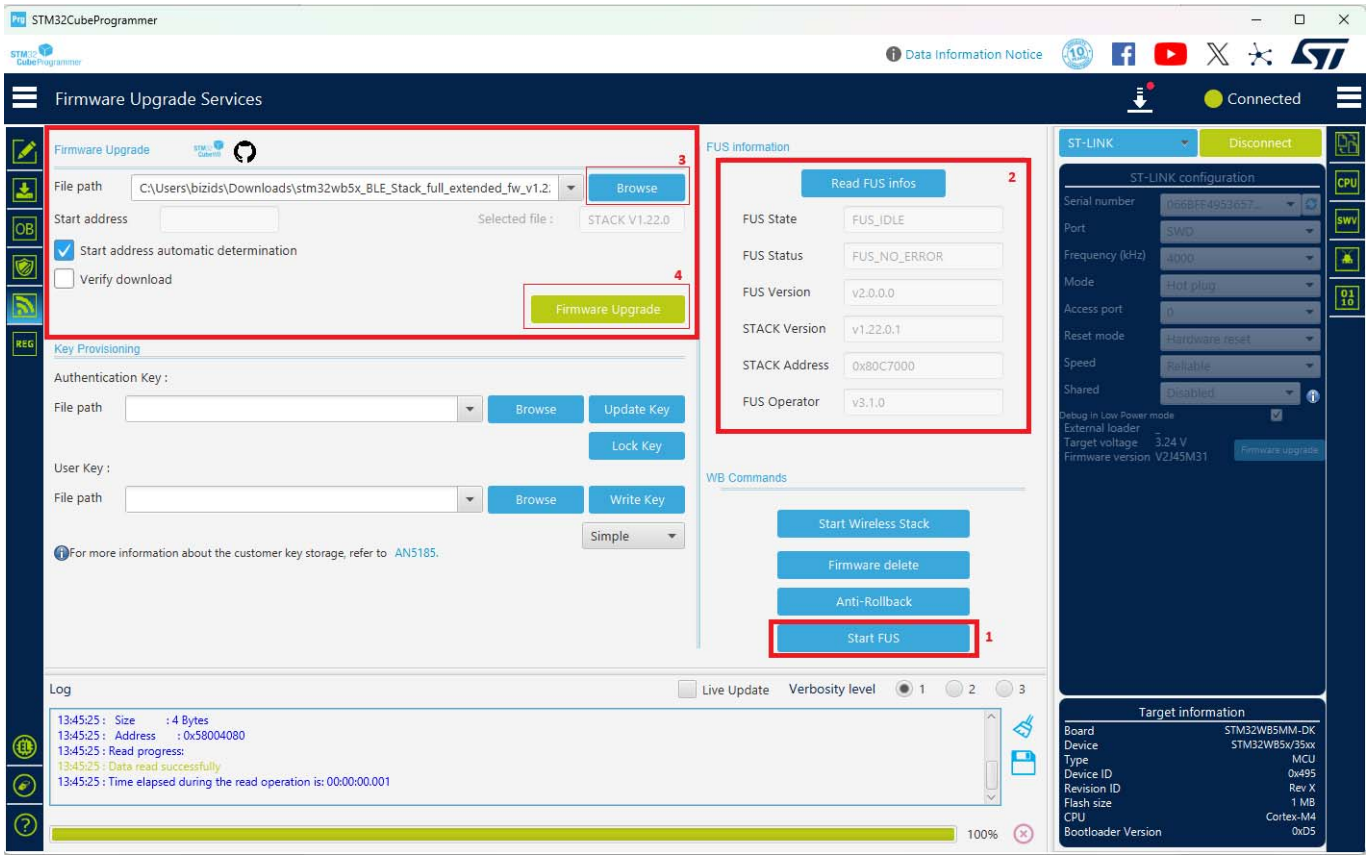


Fig. 3.20: Steps for firmware upgrade

STM32CubeProgrammer can install FUS or wireless stack in two modes:

1. **Manual: specify the desired installation address by entering it in the** Start Address field. Refer to the release notes for STM32WB Copro Wireless Binaries.
2. **Automatic: STM32CubeProgrammer automatically calculates the optimal** installation address based on the provided Stack or FUS file. To activate this option, check Start address automatic determination, then select the binary file using the Browse button.

After selecting the desired mode, click the Firmware Upgrade button to begin the upgrade process, as shown in the following figure.

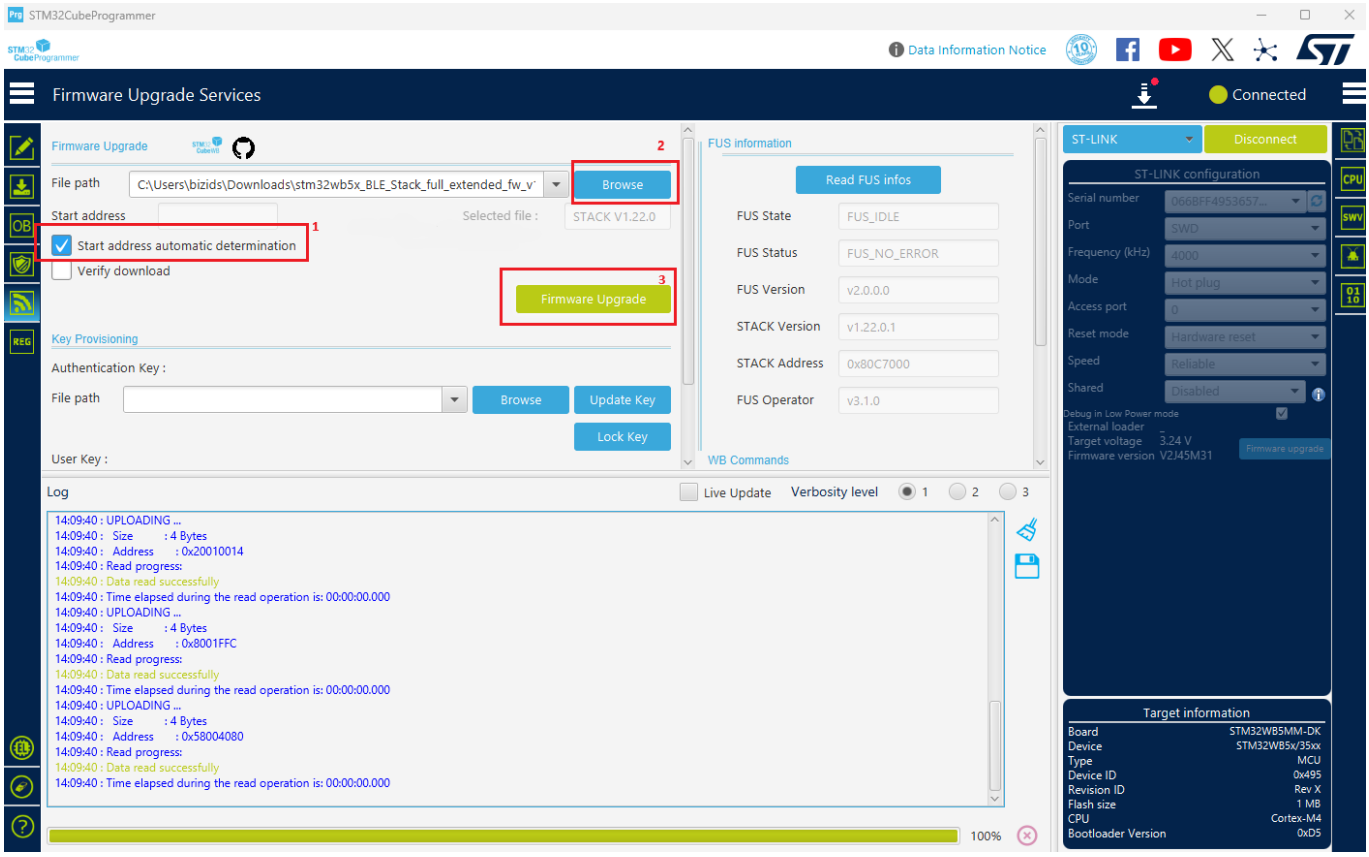


Fig. 3.21: Automatic load address determination functionality

Key provisioning

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

User authentication

FUS window allows a user authentication key to be stored through the Update Key button.

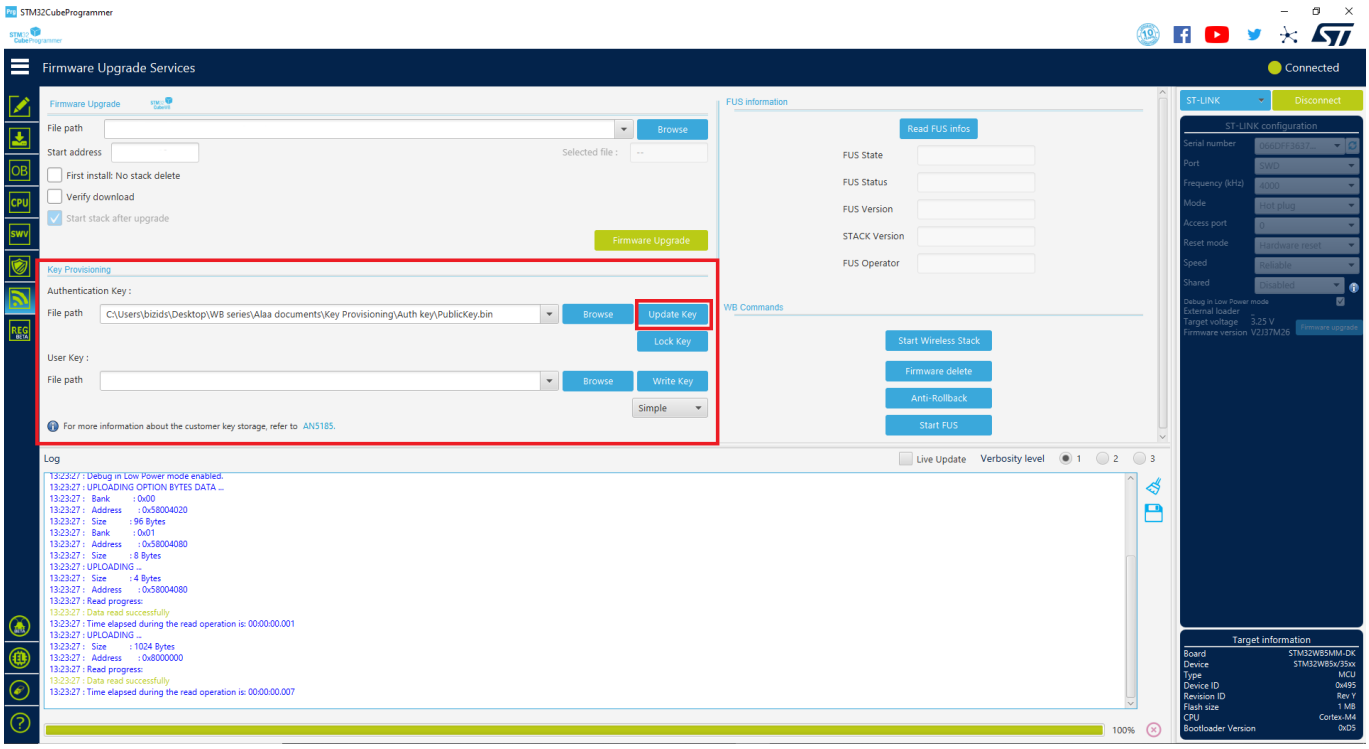


Fig. 3.22: Update authentication key

Once the user authentication key is installed, it can be changed, unless the lock user authentication key button is selected. The install or upgrade services must be done with the double signed FUS or Stack, or it is rejected.

Customer key storage

STM32CubeProgrammer allows customer keys to be stored in the dedicated FUS flash memory area in binary format, with user key types simple, master, or encrypted.

For more information about the customer key storage, refer to AN5185, ST firmware upgrade services for STM32WB Series. For complete documentation on these products, visit the dedicated pages on www.st.com.

Serial wire viewer (SWV)

The serial wire viewer window, shown in the following figure, displays the printf data sent from the target through SWO, and useful information on the running firmware.

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

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

The user can optionally specify a `.log` file to save the SWV trace log by using the Browse button. The default path is `$USER_HOME/STMicroelectronics/STM32CubeProgrammer/SWV_Log/swv.log`.

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

- `#GRN#` for green color

- `#RED#` for red color
- `#ORG#` for orange color

Example:

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

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

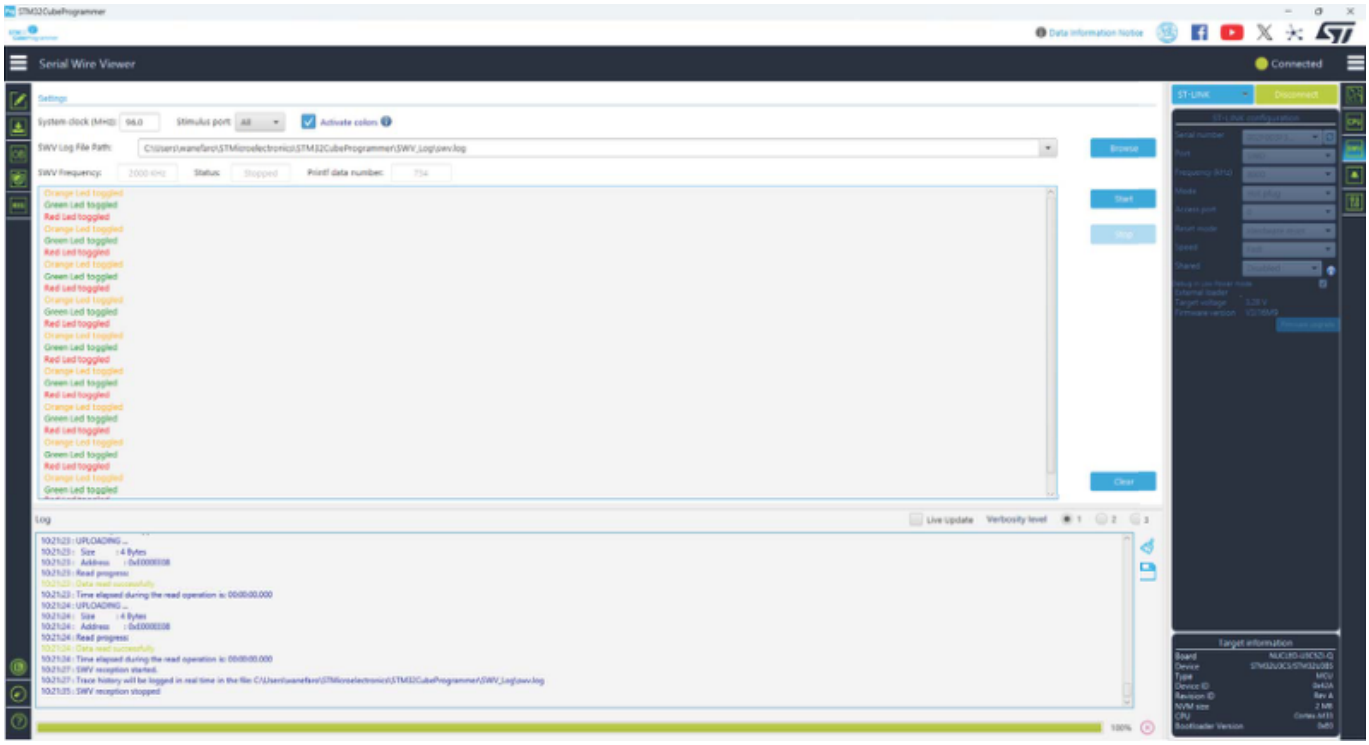


Fig. 3.23: SWV window

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

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

Note

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

Secure programming interface

Introduction

This window facilitates STM32CubeProgrammer CLI commands for secure programming:

- RDP regression with password feature: available for STM32U5, STM32U3, STM32U0, STM32WBA5/6/2, and STM32C5 series.
- SFI/SFIx feature: available for STM32H7, STM32U5, and STM32L5 series.

RDP regression with password

Some STM32 products, such as those of the STM32U0, STM32U5, and STM32C5 series, offer the possibility to use an optional password-based RDP level regression, including RDP level 2. Detailed information about this hardware mechanism is available in reference manuals.

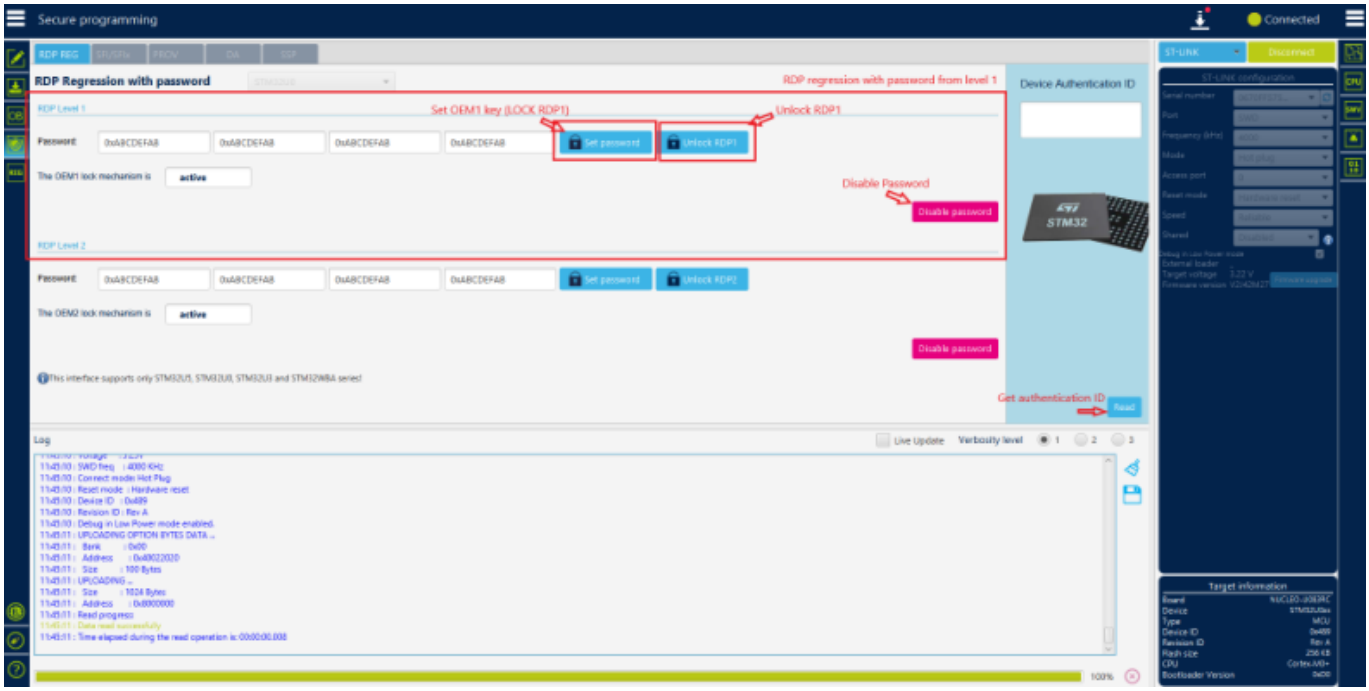


Fig. 3.24: RDP regression with password tab

STM32U5

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

STM32U0

- RDP level 1: the OEM1 RDP lock mechanism is active when the OEM1 key is set. It blocks the regression from RDP level 1.

  - **To unlock the RDP from level 1 regression, the user must connect in** hotplug mode on access port 1, write the OEM1 password, press on `RDP regression` button, and then perform the RDP regression from `Option Bytes` interface.
  - **To remove RDP regression with password from level 1, the user must press** on `Disable password` button.

- RDP level 2: provision OEM2KEY to authorize RDP level 2 to level 1 regression with the `Set password` button.
  - **To unlock the RDP from level 2 regression, write the OEM2 password, press** on `RDP regression` button, and then try to connect with STM32CubeProgrammer. If the key matches the OEM2KEY value, the RDP regression to level 1 is launched by hardware. In this use case, the connection is done with `ap = 1`.
  - **To remove RDP regression with password from level 2, the user must press** on the `Disable password` button.

Note

For more details, please refer to Annex 7, Usage of the `unlockrdp2` command.

STM32C5

- RDP level 2: provision OEM2KEY to authorize regression from RDP level 2 or level 2 with Boundary Scan to level 0 using `Set password` button.
  - **To perform the regression from RDP level 2, write the OEM2 password, press `RDP regression` button**, and then connect with STM32CubeProgrammer. If the key matches the OEM2KEY value stored in the device, the RDP regression to level 0 is launched by hardware.
- `RDP Lock Verification` button: verifies that the user-provided password matches the value provisioned during the RDP Lock step. This ensures the provisioned password is validated before setting RDP level 2.

  RDP lock verification sequence:
  - Set the OEM key values (`Set password`).
  - Verify whether the OEM keys are correct (`Verify`).

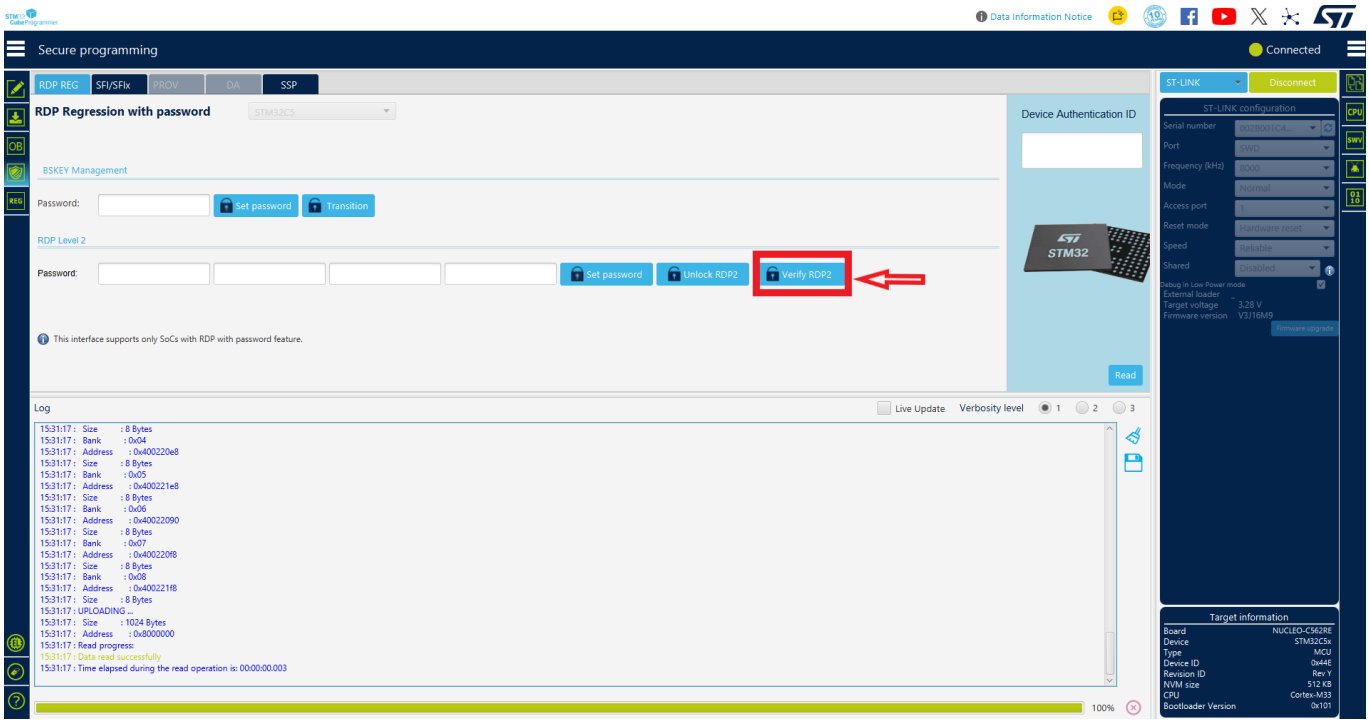


Fig. 3.25: RDP lock verification

- BSKEY management: RDP level 2 with Boundary Scan can transition to RDP level 2 using the BSKEY, a 32-bit password. The BSKEY must be set when in RDP level 0.

Note

A default value, `0xAAAAAAAA`, is preloaded during manufacturing.

- To transition RDP level 2 with Boundary Scan to RDP level 2, set the password then select the `transition` button, without connecting to the target.

SFI/SFIx

SFI GUI

1. Use STM32CubeProgrammer, version 2.11 or higher.
2. Access the SWD or bootloader interface.
3. Open Secure Programming interface, then SFI tab, as shown in the SFI/SFIx tab figure below.
4. **Select the license source, from a license file or directly from your** connected HSM.
5. **Select the SFI file. Once selected, the SFI parsed info is displayed, as** shown in the SFI parsed info figure below
6. Select the RSSe file, if needed.
7. Start SFI sequence, as shown in the Steps for SFI programming figure below.

Note

To open TPC to generate the SFI file, press on the TPC logo.

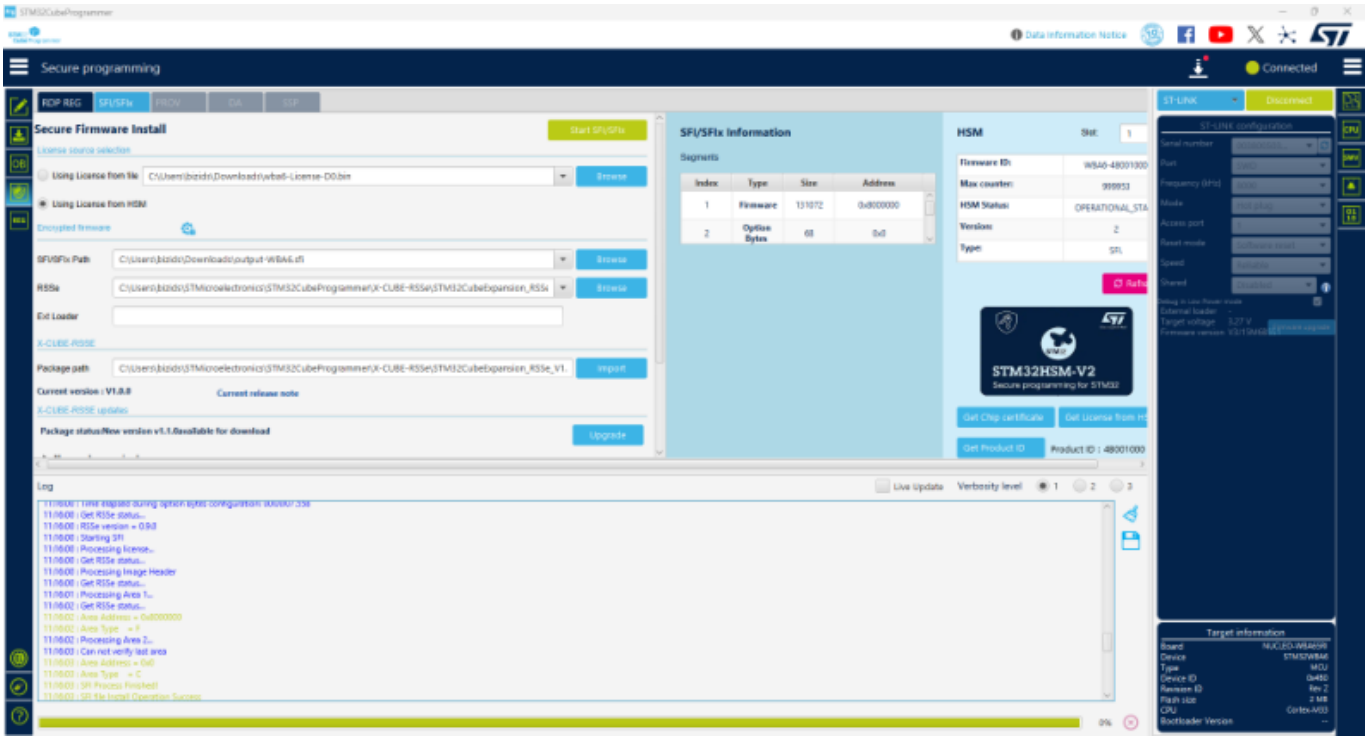


Fig. 3.26: SFI/SFIx tab

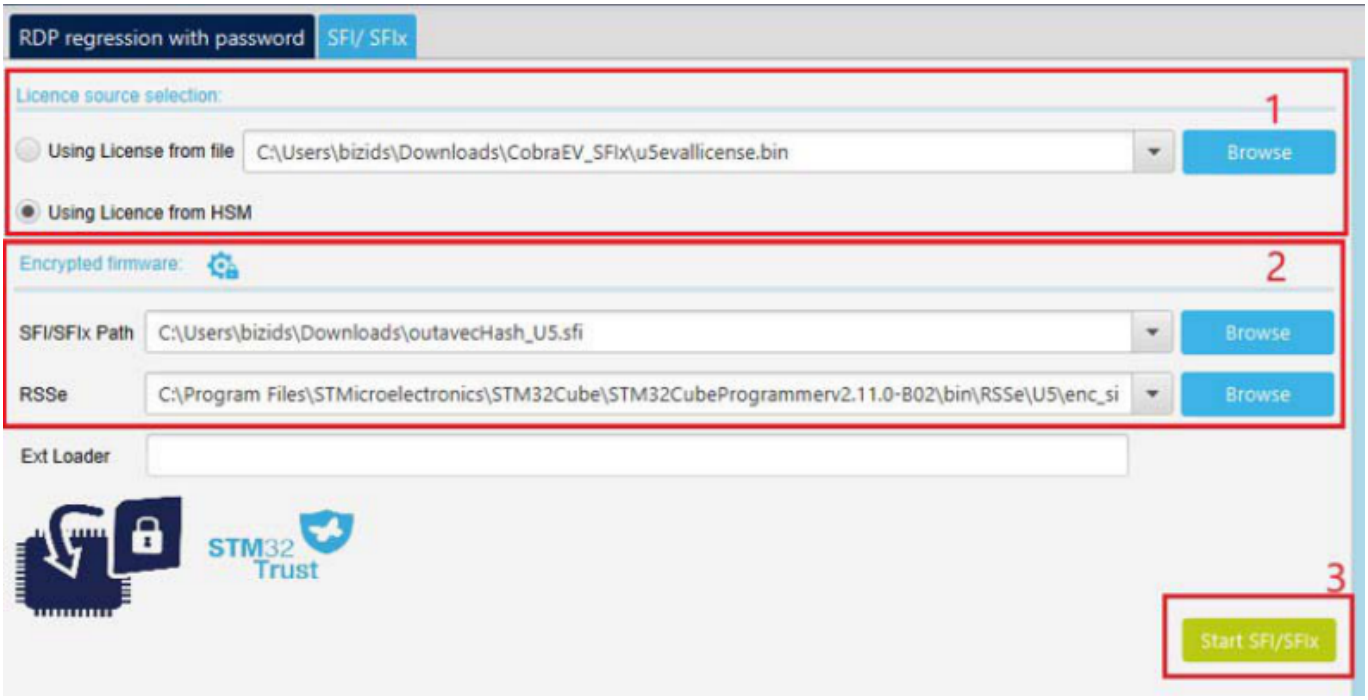


Fig. 3.27: Steps for SFI programming



Fig. 3.28: SFI parsed info

SFIx GUI

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

1. Select an SFIx, not an SFI.
2. **Select the external loader via External loaders window. Once done, the name** is displayed automatically in the text field below, as shown in the following figure.

Then you can start the SFIx sequence.



Fig. 3.29: Display external loader name

HSM related info

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

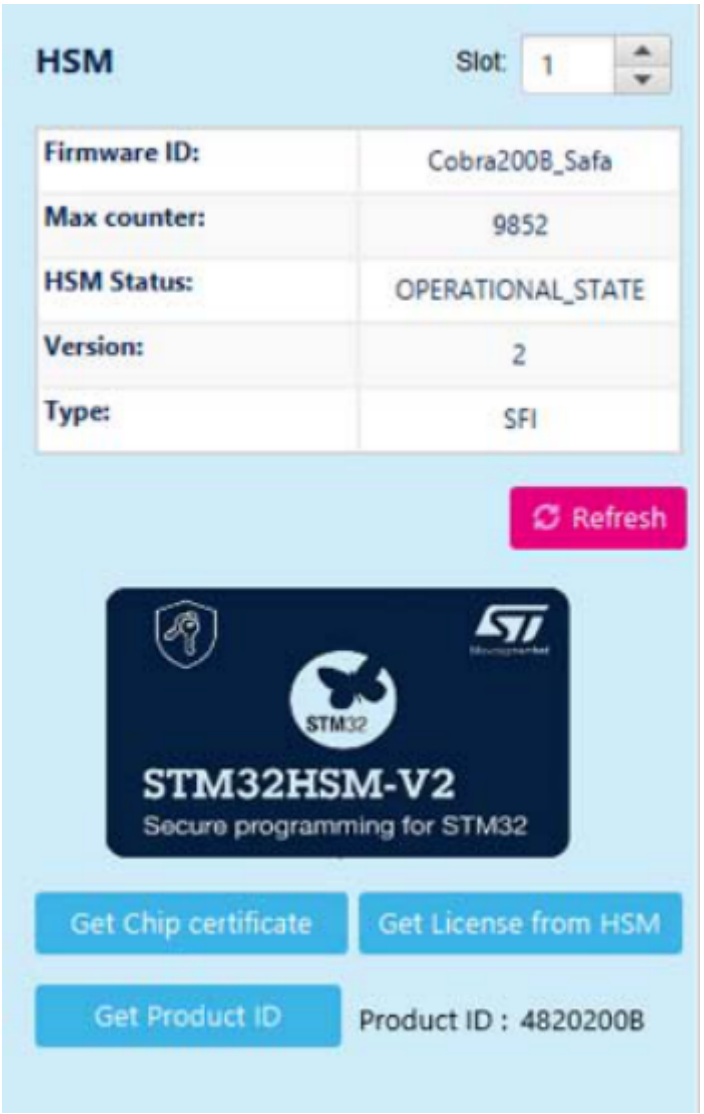


Fig. 3.30: HSM-related info panel

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

SFI/SFIx GUI for devices supporting Secure Manager

The tool displays additional fields, giving the option of passing a module to the SFI or SFIx and an external module to the SFIx:

- `Modules`: `.mcsv` file containing the list of internal modules.
- `exModules`: `.ecsv` file containing the list of external modules.

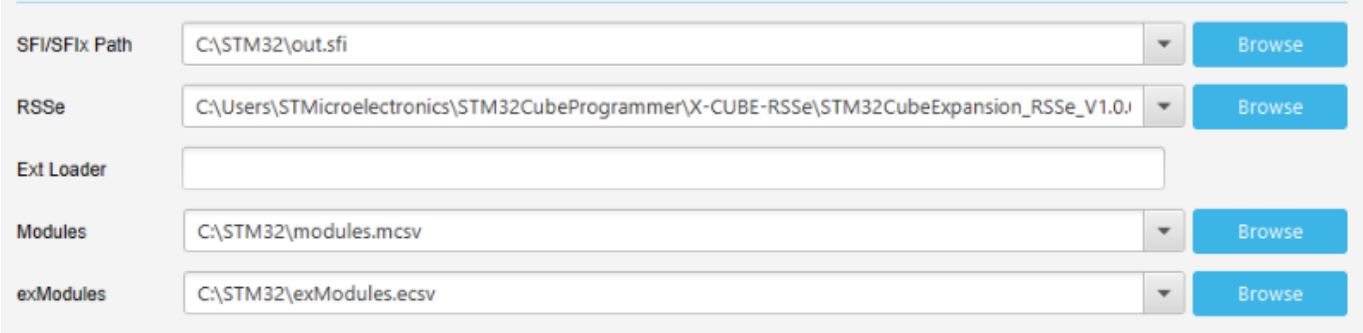


Fig. 3.31: SFI/SFIx modules for STM32H5

X-CUBE-RSSe

The X-CUBE-RSSe comes in zip format and can be directly imported. This package contains:

- RSSe binaries: Root Security Services Extension needed to perform the SFI/Key Wrapping.
- Personalization data: encrypted binaries to be programmed into HSM depending upon the used MCU.
- Option bytes CSV: template for OB CSV files to be used for SFI file creation.

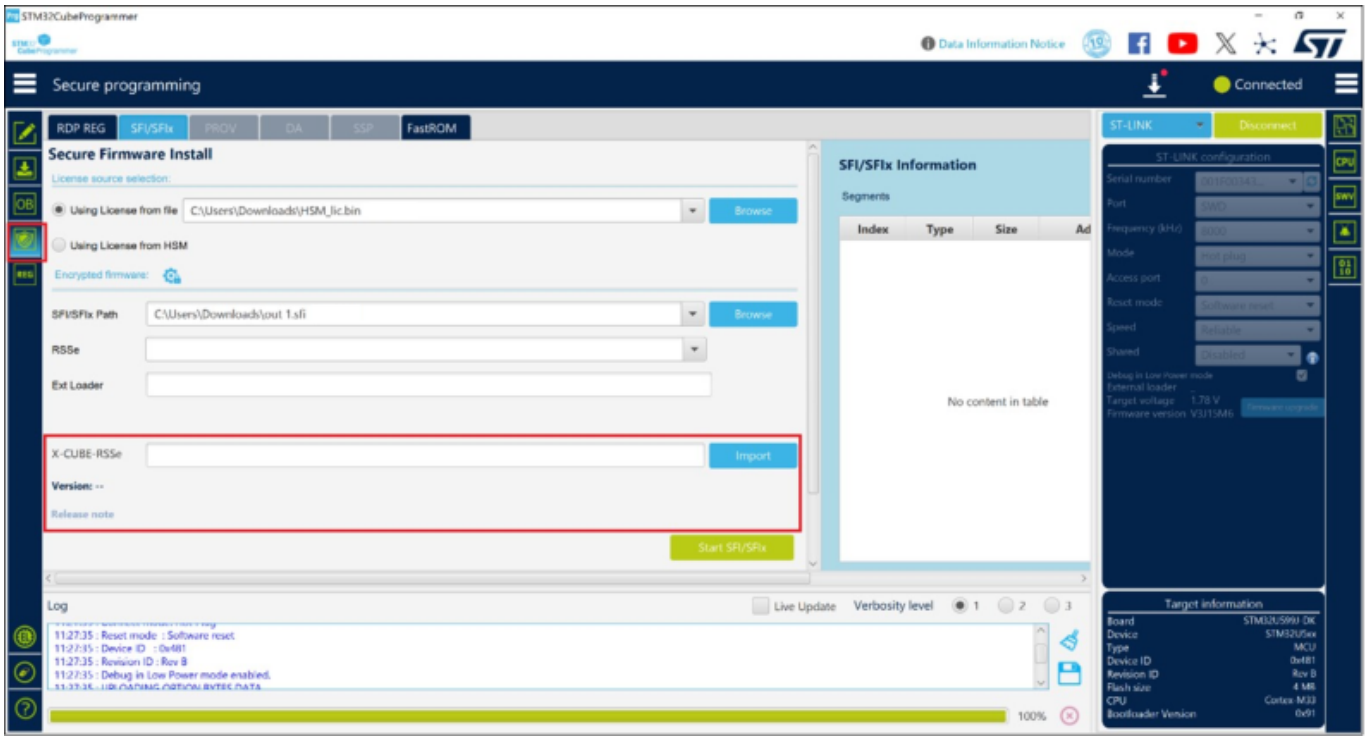


Fig. 3.32: X-CUBE-RSSe interface in the SFI/SFIx window

The user can import this package from the SFI/SFIx window within the Secure Programming window. This window contains:

- Imported package path: automatically populated after importing the package.
- Package version: automatically filled after importing the package.
- Release note link: the hyperlink is enabled after importing the package.

After successfully importing the package, a populated list of available RSSe binaries is automatically filled in its respective fields.

X-CUBE-RSSe and STM32MPUSSP-UTIL upgrade

In the SFI/SFIx and SSP tab, the user can check for the latest version of the package available for download.

After accepting the software license, if the download and upgrade of the new version are successful, the package status is marked as up-to-date and the new RSSe files are automatically filled in their respective fields.

For users using proxy to connect to the web, proxy settings must be configured.

SSP

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

To open the SSP window, connect an MPU or an STM32N6 device via DFU interface, click on security panel, and then choose the SSP tab. The window contains the graphical components needed to perform SSP operations.

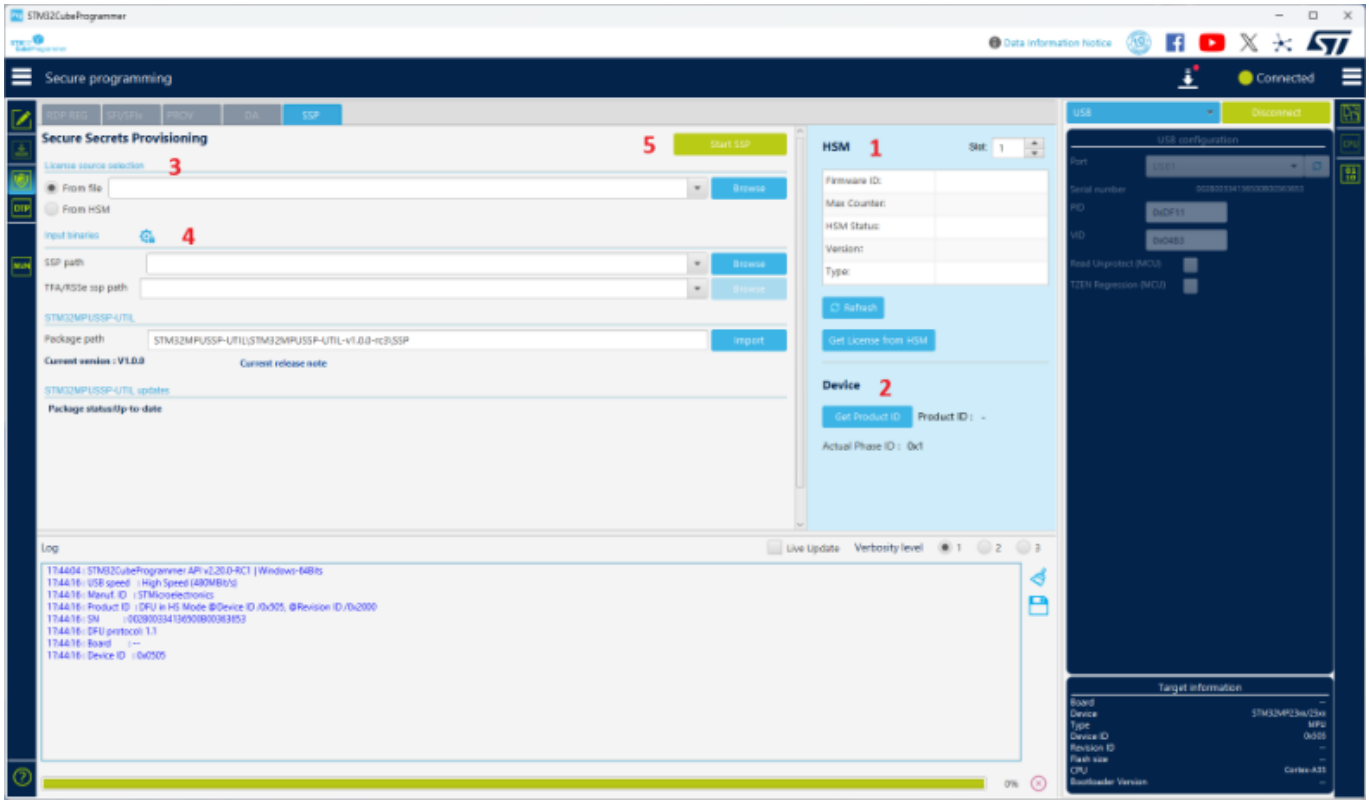


Fig. 3.33: SSP PRG user interface

The UI is composed of five elements:

1. HSM section.
2. Device section.
3. Input license.
4. SSP input binaries.

5. Verify and Start SSP install.

HSM section

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

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

Device section

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

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

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

Input license

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

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

SSP input binaries

This part is needed to select the secure input files:

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

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

Verify and Start SSP install

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

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

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

> **Note**
>
> At the end of the flow STM32CubeProgrammer does not make any verification step for the OTP fused words. If the mentioned `tfa-ssp` is not the adequate one, the SSP flow fails.

The STM32MPUSSP-UTIL comes in zip format and can be directly imported in the STM32TrustedPackageCreator. This package contains:

- RSSe SSP binaries: Root Security Services Extension needed to perform Secure Secret Provisioning.
- Personalization data: encrypted binaries to be programmed into HSM depending on used MPU.

The user can import the STM32MPUSSP-UTIL package from the SSP window within the Secure Programming window. This window contains:

- Package version: automatically filled after importing the package.
- Imported package path: automatically populated after importing the package.

- Release note link: the hyperlink is enabled after importing the package.

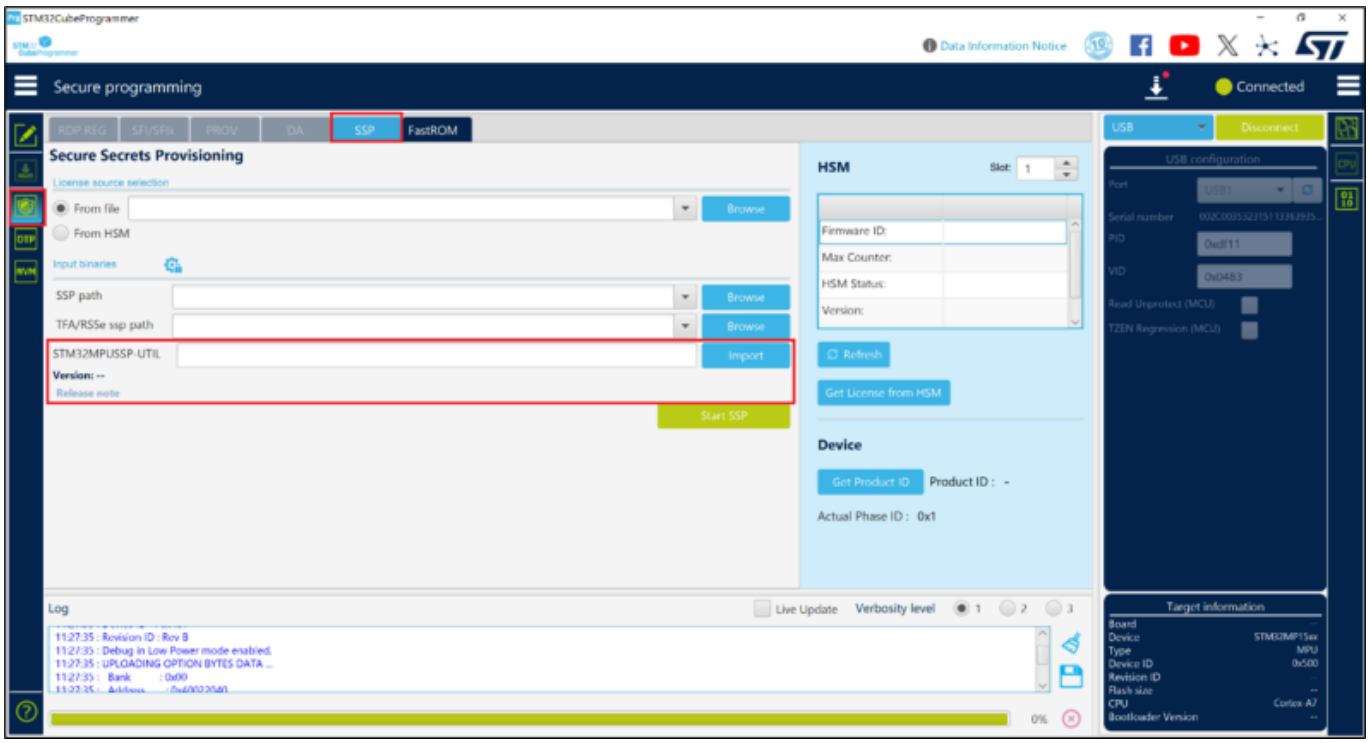


Fig. 3.34: STM32MPUSSP-UTIL interface in the SSP window

OBKey provisioning

This is a security feature that allows to program OBKey content. The file generation is managed by STM32 Trusted Package Creator. For more information, refer to UM2238.

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

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

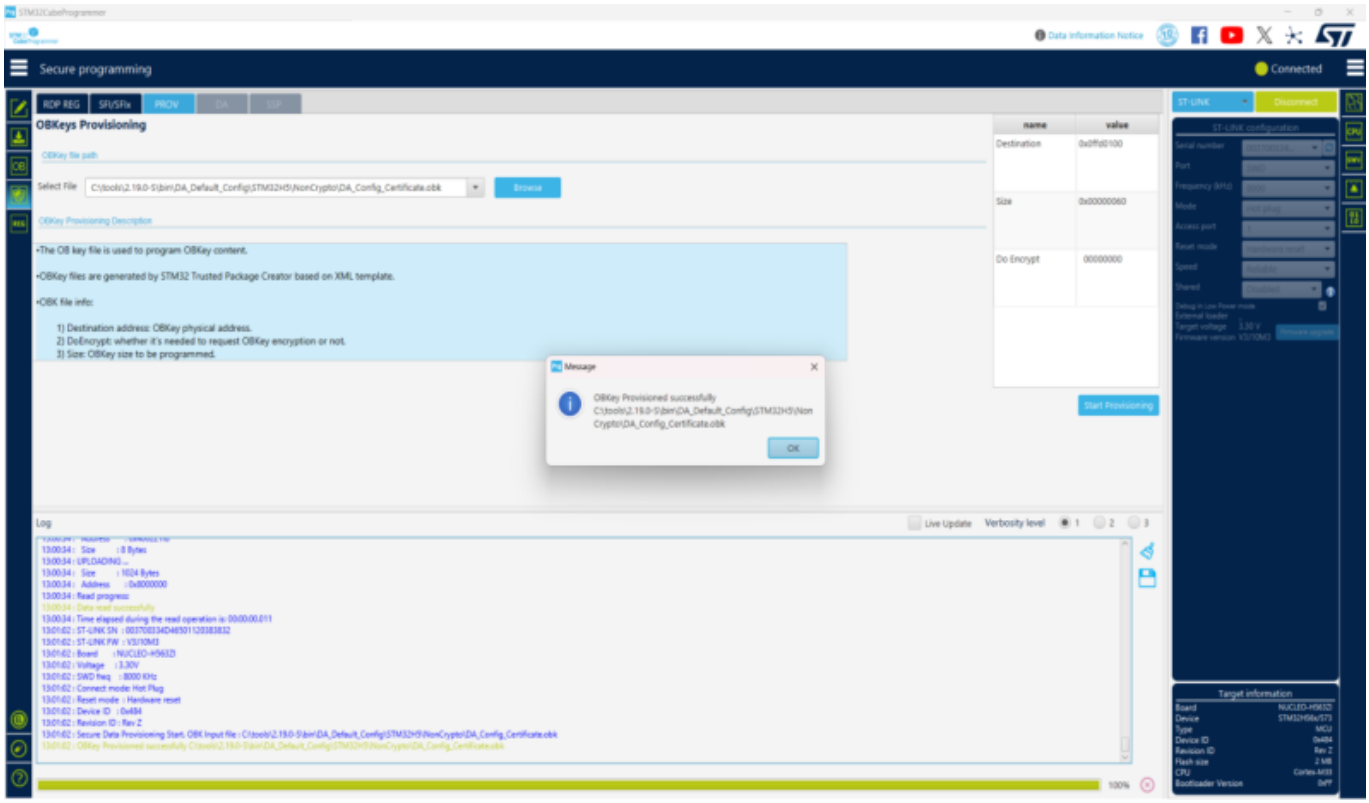


Fig. 3.35: OBKey provisioning

OTP provisioning panel

Debug authentication - Password provisioning

For devices supporting debug authentication without Arm TrustZone, the password hash, `hash256`, is stored in OTP. This panel allows password provisioning. To do it, enter a password value, size must be between 4 and 16 bytes, and a password path.

Once the OTP is written, the corresponding block is locked. The password value is used to calculate the hash to store in OTP. The password path is the location where to save the `password.bin` file, needed to open the device in a Debug Authentication sequence.

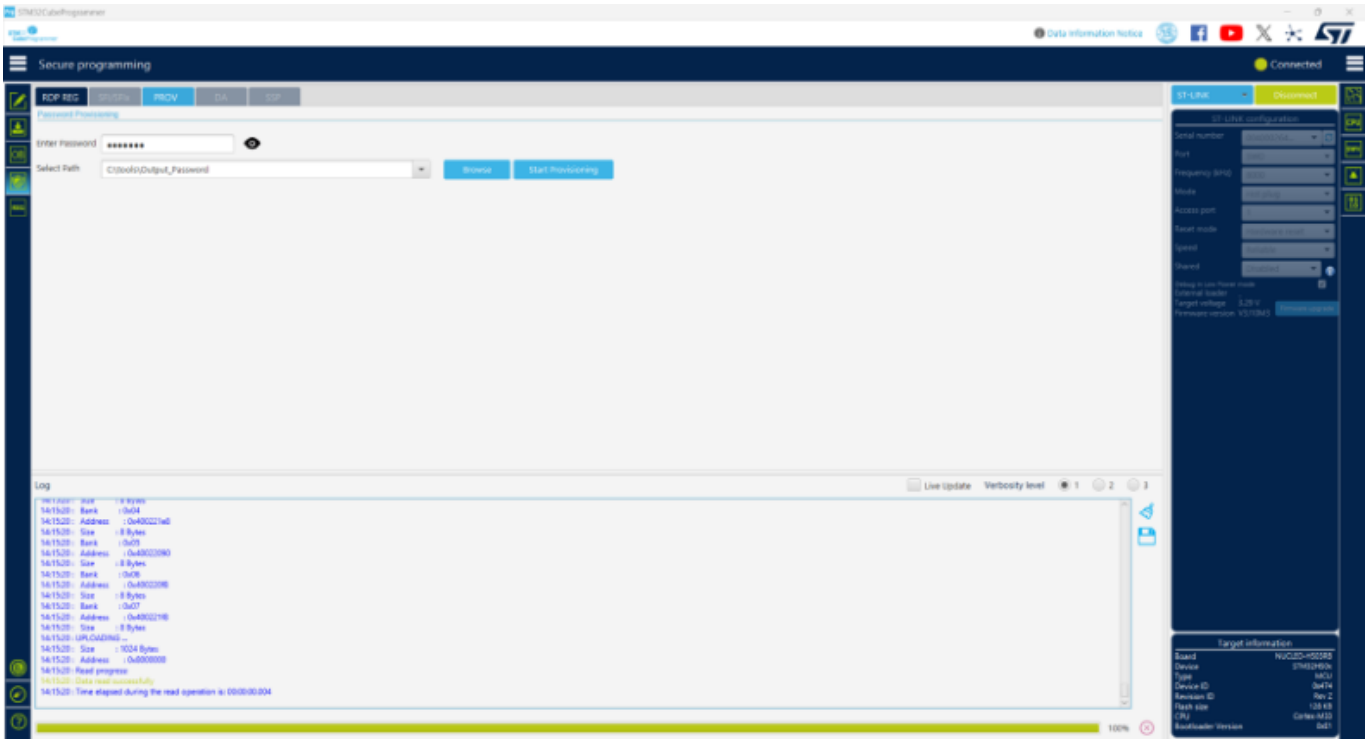


Fig. 3.36: Password provisioning

> **Note**
>
> For more information see Getting started with debug authentication (DA) for STM32 MCUs.

Password provisioning can be performed only once for each device, as the password is stored in OTP.

Public key provisioning for STM32WB0x/STM32WL3x devices

This panel is used to provision authentication public key used in secure boot scenario. The input key is generated by STM32TrustedPackage Creator. The associated private key is used to sign the firmware stored in the flash memory. For more information refer to AN5471, The BlueNRG-LP, BlueNRG-LPS UART bootloader protocol, available on www.st.com.

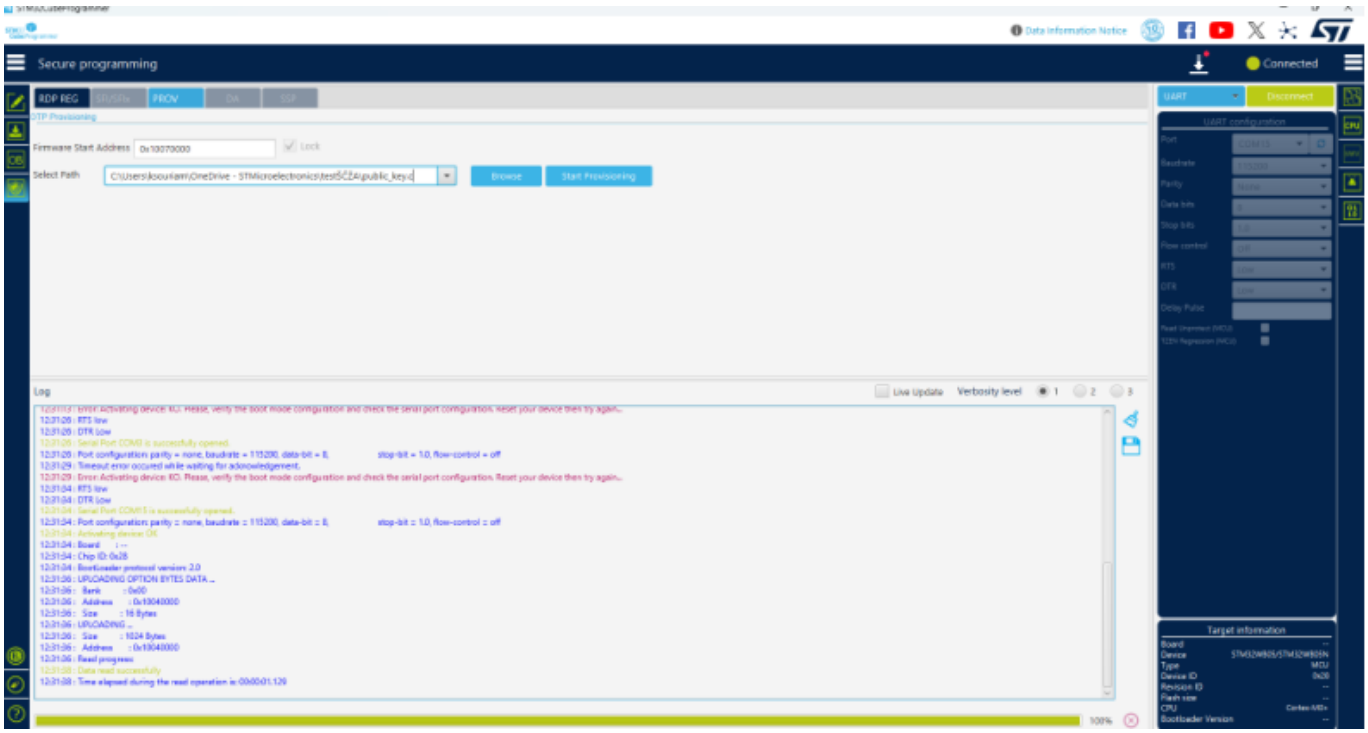


Fig. 3.37: Public key provisioning for STM32WB0x/STM32WL3x devices

Debug authentication

This feature allows regression or open debug when the target is provisioned. The user must provide the needed credentials by following these steps:

1. Discovery: this operation displays the info about the target.

   To make sure that provisioning is correctly performed, `Integrity status` field must be checked: it must indicate `0xEAEAEAEA` as value, displayed in the log. This operation cannot be launched while the target is connected.

   Before launching Discovery, select the appropriate Magic value (`STDA` or `OEDA`). The default value for STM32 devices is `STDA`. For STM32N6, you must select `OEDA` to establish communication with the target.

2. **Credentials input: beside discovery related data, Debug authentication** panel displays at this step a form including the necessary inputs:

   a. **Target provisioned with password: the panel includes only the password** file path.

b. **Target provisioned with certificate: the panel includes key and** certificate path inputs. In this case, the user can select the possible permission or actions, and multiple choices can be checked for some cases.

3. **For devices supporting debug authentication with TrustZone, the user can** close the debug after opening it, instead of powering off or on the target. To close debug, use the `Close Debug` button included in Debug Authentication panel.

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

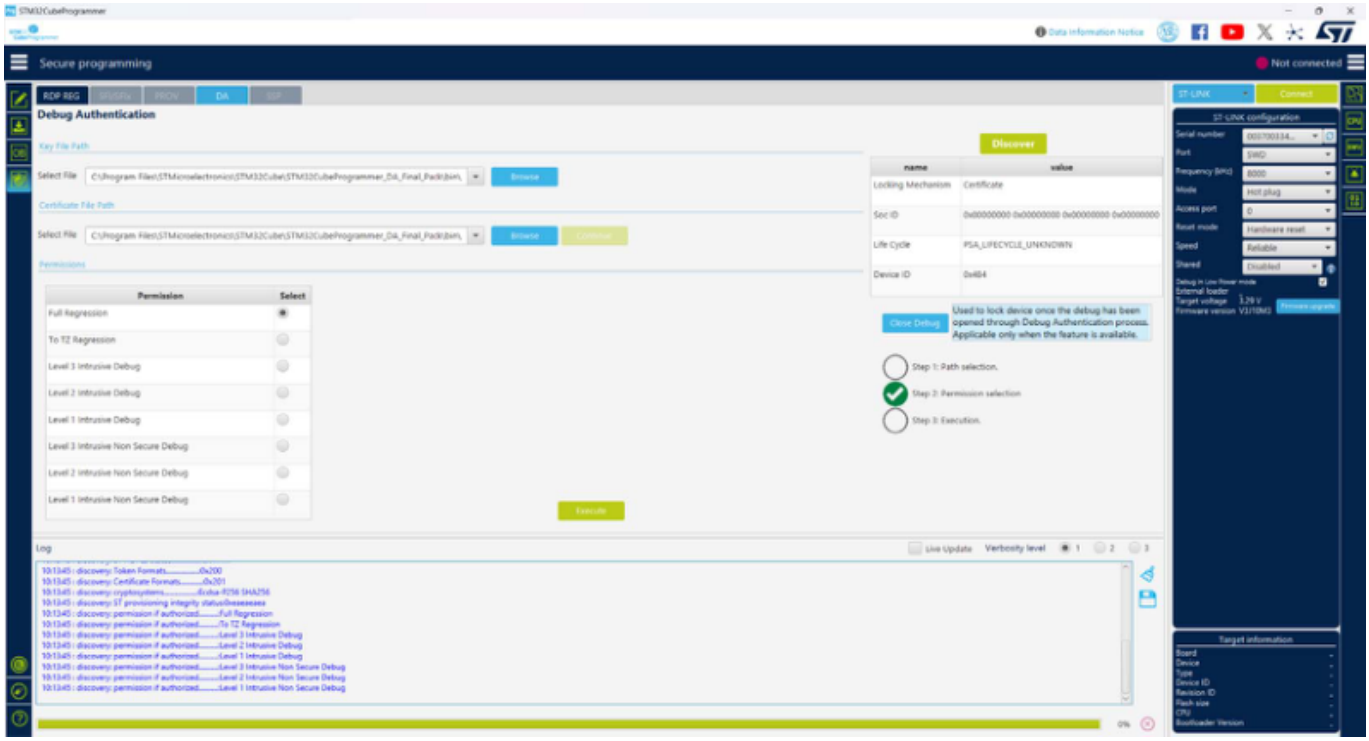


Fig. 3.38: Debug authentication with certificate

STM32CubeProgrammer Script Manager platform for MCUs

Introduction for the Usage Scenarios of Script Manager

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

Script Manager usage

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

Usage example:

```
STM32_Programmer_CLI -script myScript.prg
```

The Script Manager can apply mathematical and logical operations, as shown in Table 1.

Using command line interface, CLI: in this script, all CLI commands supported by STM32 MCUs can be used.

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

Write macros

```
#Write32(Address,data)
#Write16(Address,data)
#Write8(Address,data)
#WriteX(Address,#var)
```

X can be 8, 16, or 32.

Description: downloads the specified 32-, 16-, or 8-bit data into flash memory starting from a specified address.

Read macros

```
#Read(Address)
#variable=#Read(Address)
```

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

Display macros

```
#Display("message")
#Display(#errorLevel)
#Display(#variable)
```

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

Table 3.1: Table 1. Operations supported by Script Manager

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

Delay macro

```
#Delay(Time)
```

Description: allows user to put the system in standby for a period in ms.

Calculate macro

```
#variable=[var1] op [var2]
#variable=var1 shift (number of bits to shifted)
```

Description: calculates with mathematical and logical operations in Script Manager.

Disconnection command

```
--scriptdisconnect
```

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

> **Note**
>
> Comments in the Script Manager can be added by using `//`, as shown in the examples.

Script Manager example 1, CLI and Script macro

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

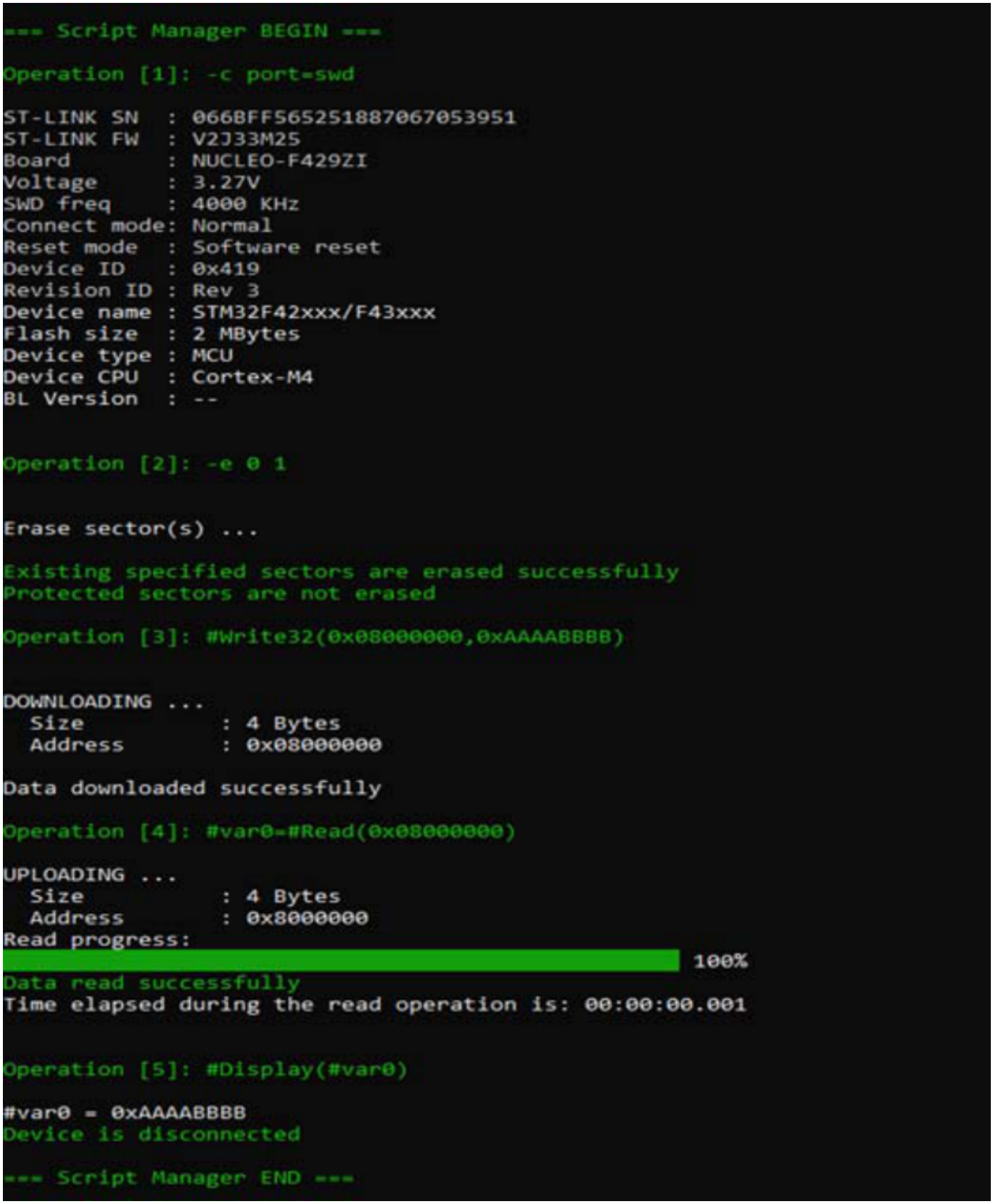


Fig. 3.39: Output of Script Manager - Example 1

Script Manager example 2

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

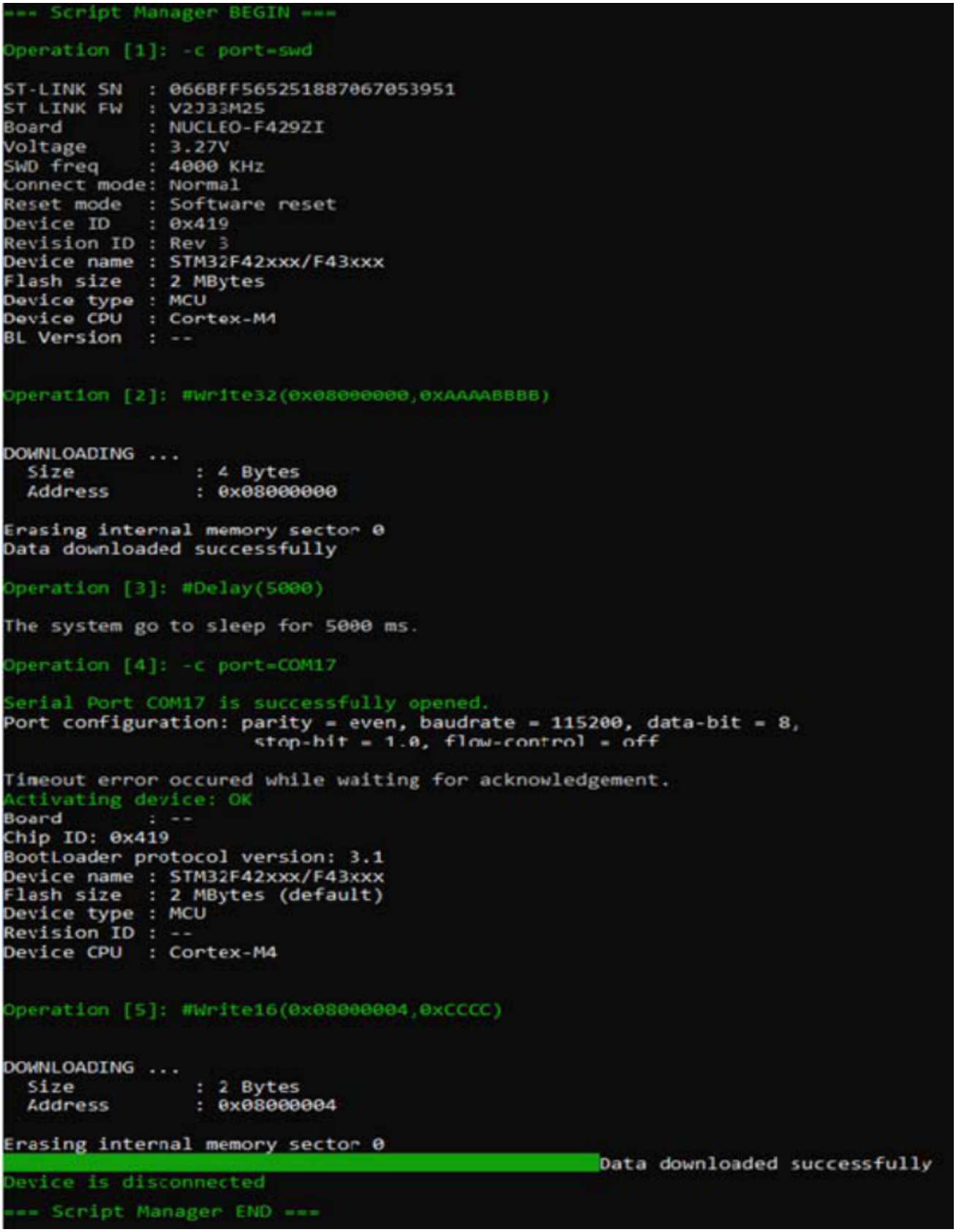


Fig. 3.40: Output of Script Manager - Example 2

Script Manager example 3

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

Loops and conditional statements

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

To use the conditional statements, `if` and `else`, and the loopings, `while` and `for`, begin with `#Start`, and finish with `#End`.

If-else condition example

```
-c port=swd
-e 0
#x=#Read(0x08000000)
#Start
if (#x > 0x1000)
{
  #Display("Condition 1")
  #Display(#x)
  #Write32(0x08000000,0x1123)
}
else
{
  #Display("Condition 2")
```

(continues on next page)

(continued from previous page)

```
  #Display(#x)
  #Write32(0x08000008,0x1124)
}
#End
```

If-else if condition example

```
-c port=swd
-e 0
#VAR=0x11111111
#Write32(0x08000000,#VAR)
#x=#Read(0x08000000)
#Start
if(#x==0x22222222)
{
  #Display("Condition 1")
}
else if(#x==0x11111111)
{
  #Display("Condition 2")
}
else
{
  #Display("Condition 3")
}
#End
```

Generic syntax

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

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

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

For loop

```
-c port=swd
#ADD=0x08000004
#x=#Read(0x08000004)
#Start
for(#ADD=0x08000000;#ADD<0x0800000C;#ADD=#ADD+4)
{
  #x=#Read(#ADD)
  #Display(#x)
}
#End
```

While loop, example 1

```
-c port=swd
-e 0
#Write32(0x08000008,0xCCCCCCCC)
#ADD=0x08000000
#x=#Read(#ADD)
#Start
while(#x!=0xCCCCCCCC)
{
         #Display(#x)
         #ADD=([#ADD]+(4))
         #x=#Read(#ADD)
}
#End
```

While loop, example 2

```
-c port=swd
-e 0
#Write32(0x08000000,0xAAAAAAAA,0xBBBBBBBB,0xCCCCCCCC,0xDDDDDDDD)
#ADD=0x08000000
#x=#Read(#ADD)
#Display(#x)
#Start
while(#x!=0xDDDDDDDD)
{
         #Display(#x)
         #ADD=([#ADD]+(4))
         #x=#Read(#ADD)
}
#End
```

DFU IAP/USBx with custom PID and VID

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

Before starting the DFU connection using a new product ID, sign your USB driver. For more information visit woshub.com.

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

To connect via the new USB DFU follow this sequence:

1. Modify the default product ID.
2. Modify the default vendor ID.
3. Click on refresh button then on the connect button.

> **Note**
>
> If the user does not enter a PID or VID value, STM32CubeProgrammer takes the default values of ST products, PID = `0xDF11` and VID = `0x0483`.

The following figure shows the steps to connect via the new USB DFU panel.

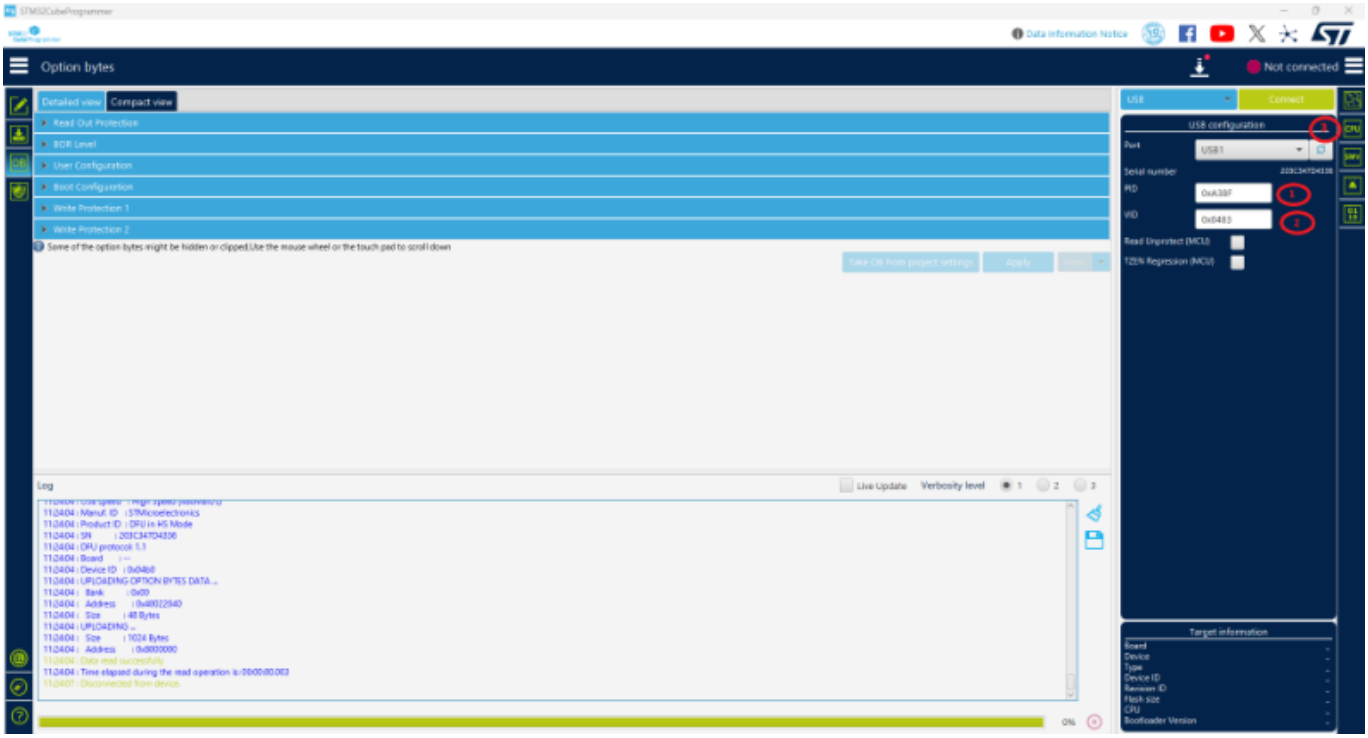


Fig. 3.41: Connect via USB DFU panel

> **Note**
>
> For CLI mode, check Section 3.2.1: Connect command.

SigFox credentials

As soon as an STM32WL device is connected, the window shown in the following figure is displayed. This window displays the chip certificate, having the size of 136 bytes. The user can save it in binary file and copy the data to the clipboard.

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

Case 1: Binary-Raw

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

Case 2: Binary KMS

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

Note

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

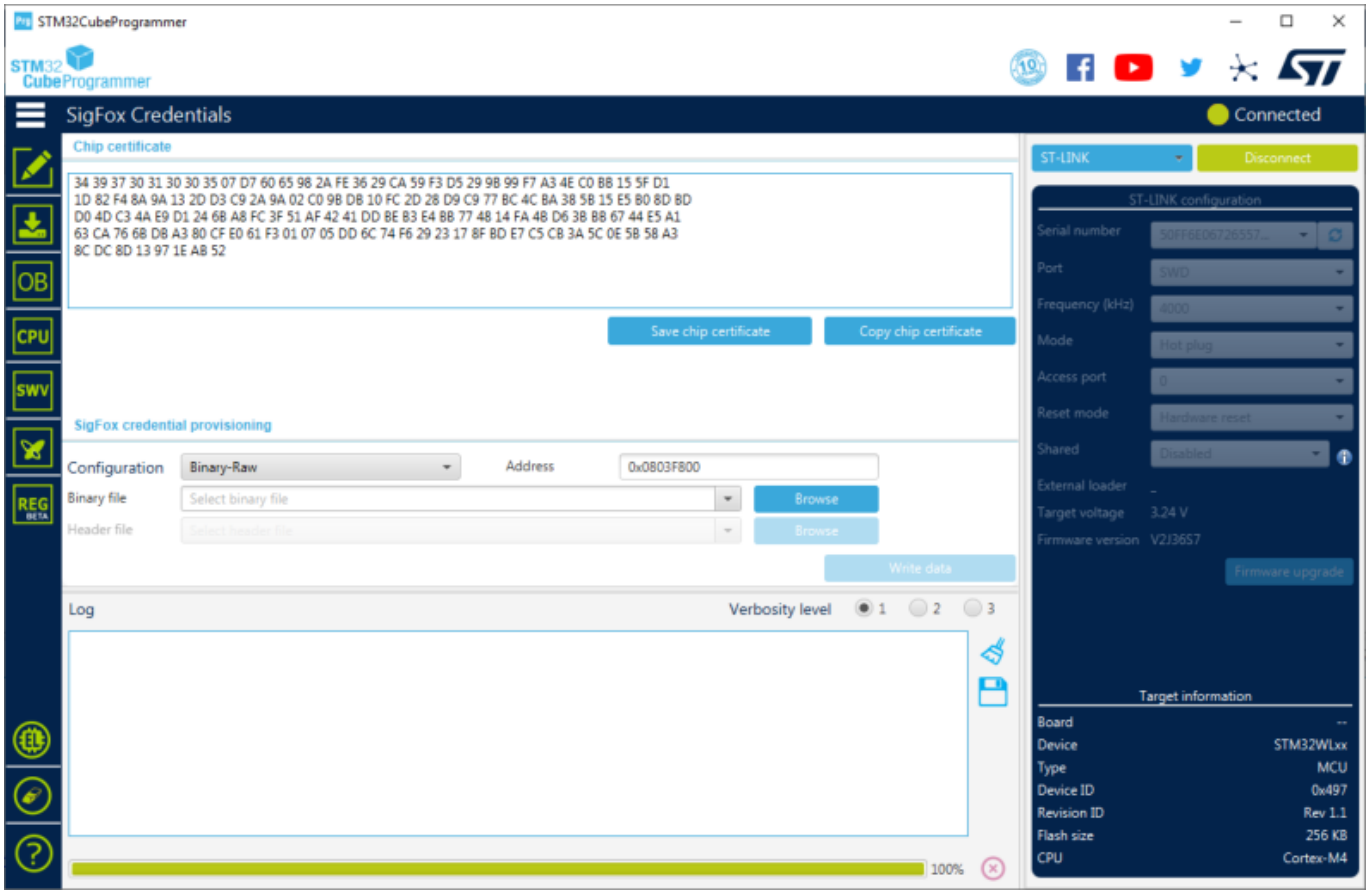


Fig. 3.42: SigFox credentials

Register Viewer

STM32CubeProgrammer supports the Register Viewer feature, shown in the following figure, allowing the user to visualize all the MCU and core registers in real time while running the application. It also allows the modification of MCU registers values or saving them into a log file.

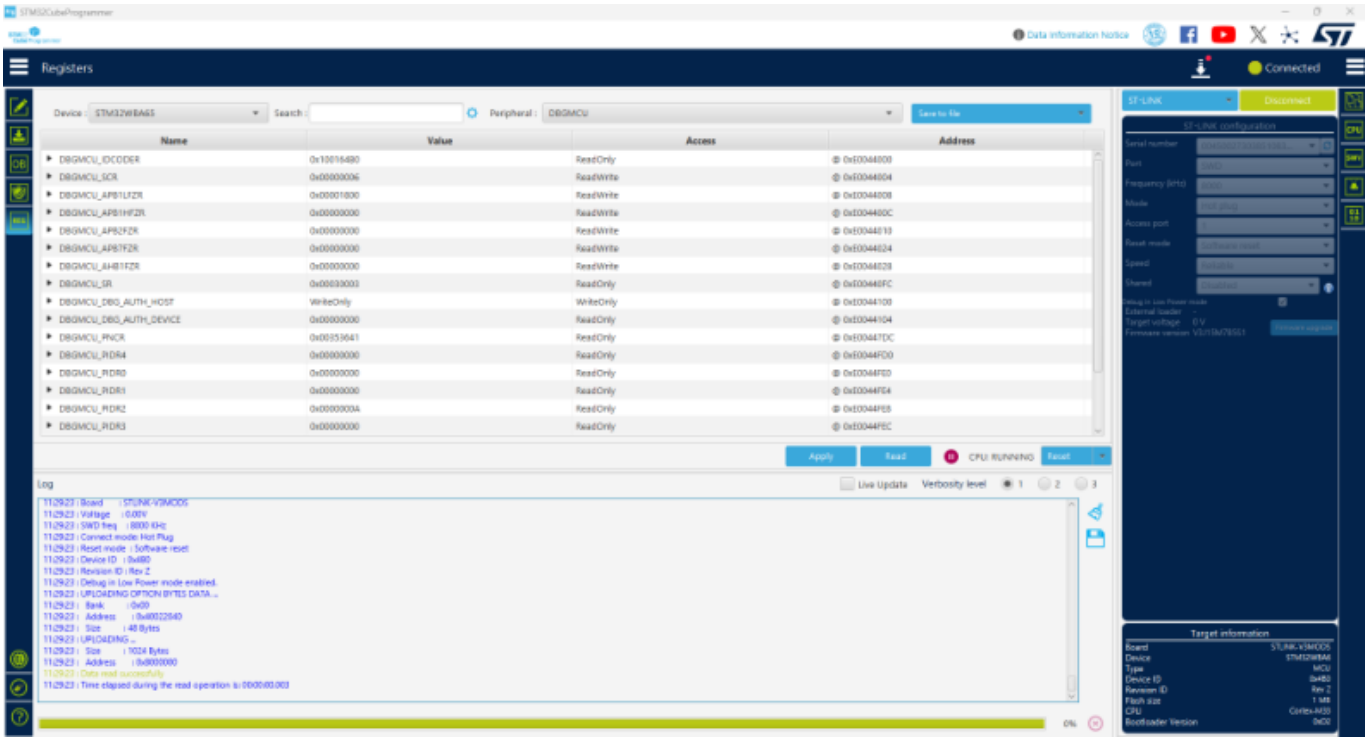


Fig. 3.43: Register Viewer window

Note

Register Viewer is available only through SWD/JTAG interfaces.

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

Hard Fault analyzer

Description

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

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

Possible detected fault exceptions:

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

- **Secure Fault: provides information about security related faults for** Cortex-M33 based devices.

Note

Fault analyzer is available only for ST-LINK interfaces.

As shown in the following figure, the Fault Analyzer window has five main sections.

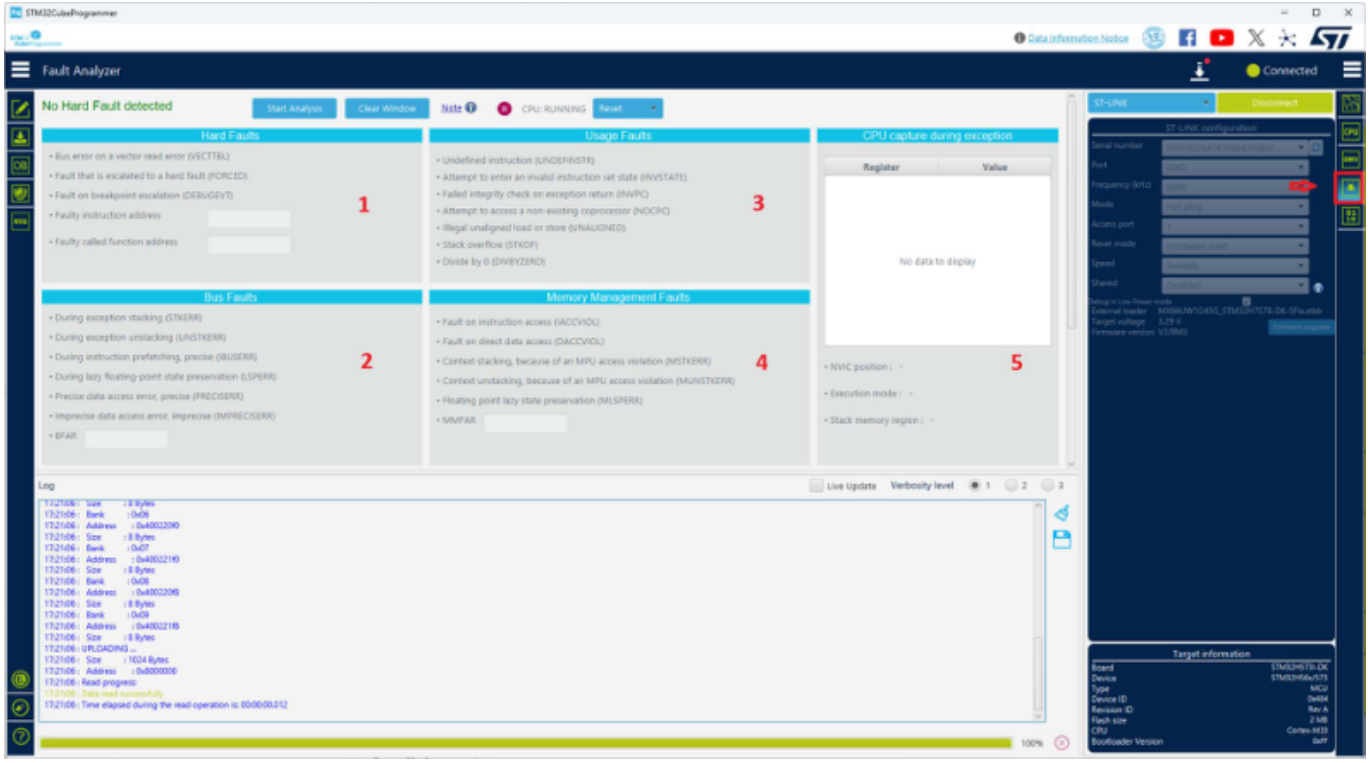


Fig. 3.44: Fault Analyzer window

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

Example

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

- `int a = 4, b = 0, c = 0`
- `c = a / b`

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

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

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

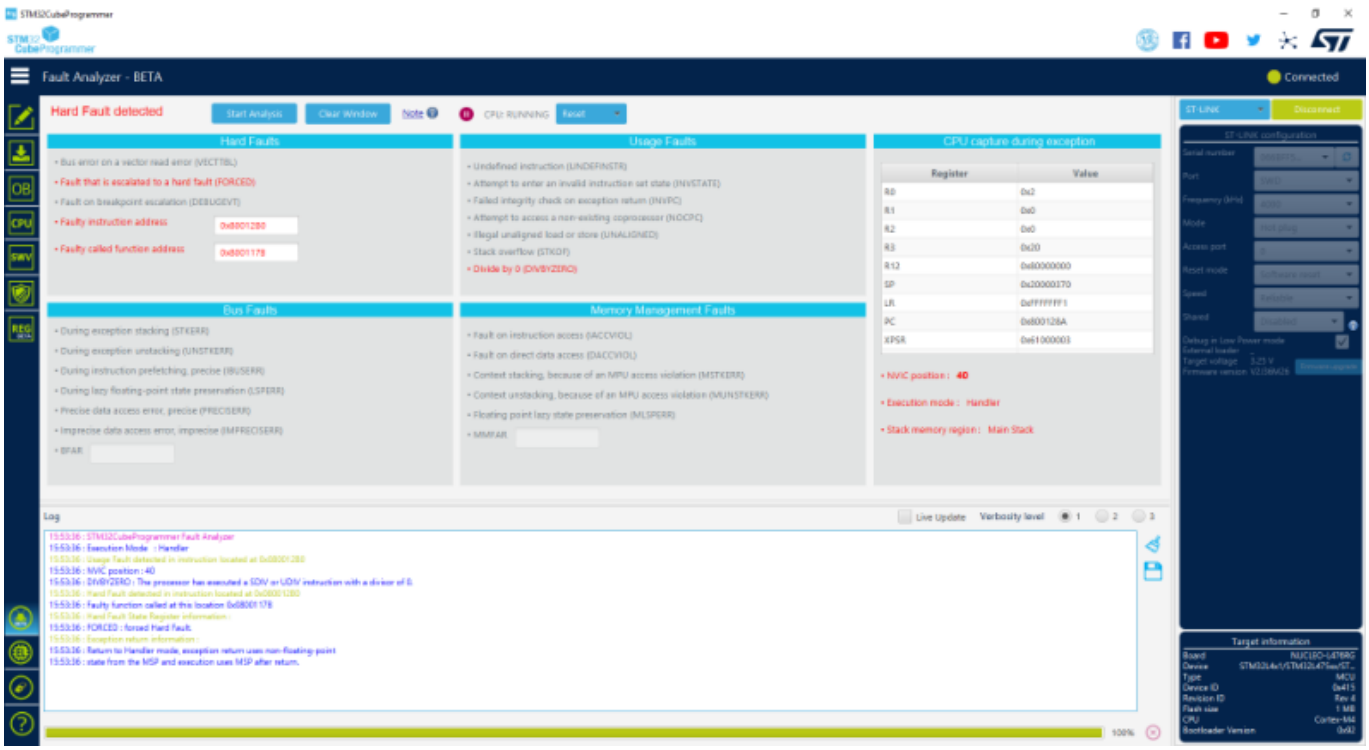


Fig. 3.45: Fault analyzer GUI view when Hard Fault is detected

Fault analyzer note

Fault analyzer can be unable to detect untracked faults not enabled by software.

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

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

- **`DIV_0_TRP`: enable Usage Fault when the processor executes an SDIV or** UDIV instruction with a 0 divider.

Figure 59. CCR bits

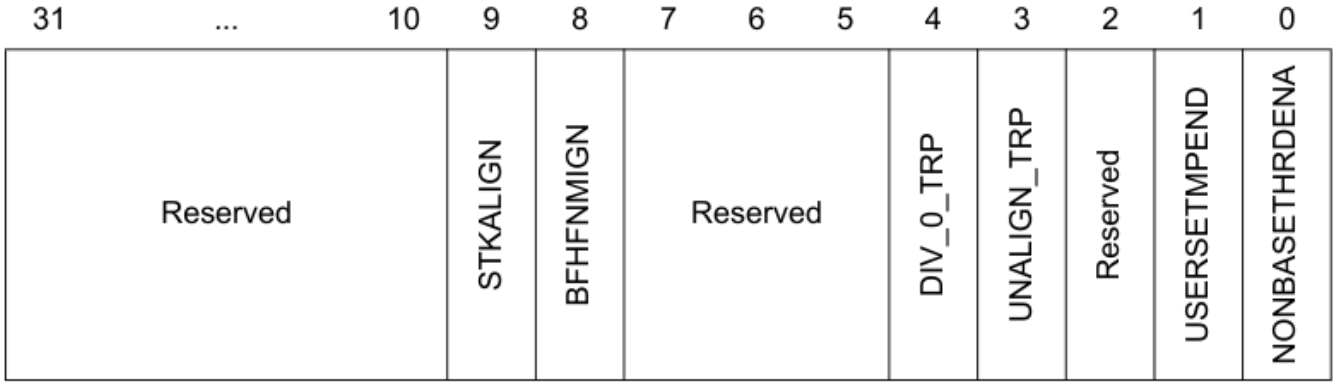


Fig. 3.46: CCR bits

  - `0` = do not trap divide by 0; a divide by 0 returns a quotient of 0.
  - `1` = trap divide by 0.

- **`UNALIGN_TRP`: enable usage fault when a memory access to unaligned** addresses is performed.
  - `0` = do not trap unaligned half-word and word accesses.
  - `1` = **trap unaligned half-word and word accesses; an unaligned access** generates a usage fault.

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

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

Secure Fault analyzer for Cortex-M33

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

A new field named `Secure Faults` is added to Fault Analyzer window when connecting a Cortex-M33-based device, such as an MCU of the STM32L5 series. The result analysis is based on Secure Fault Status Register, SFSR, settings and a fault is triggered if an error occurs:

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

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

Fill memory operation

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

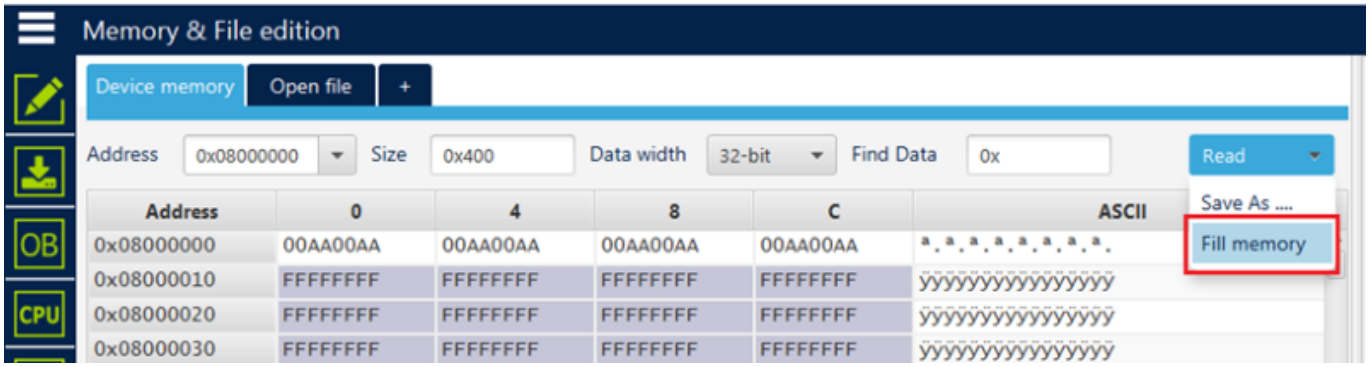


Fig. 3.47: Sub-menu displayed from Read combo-box

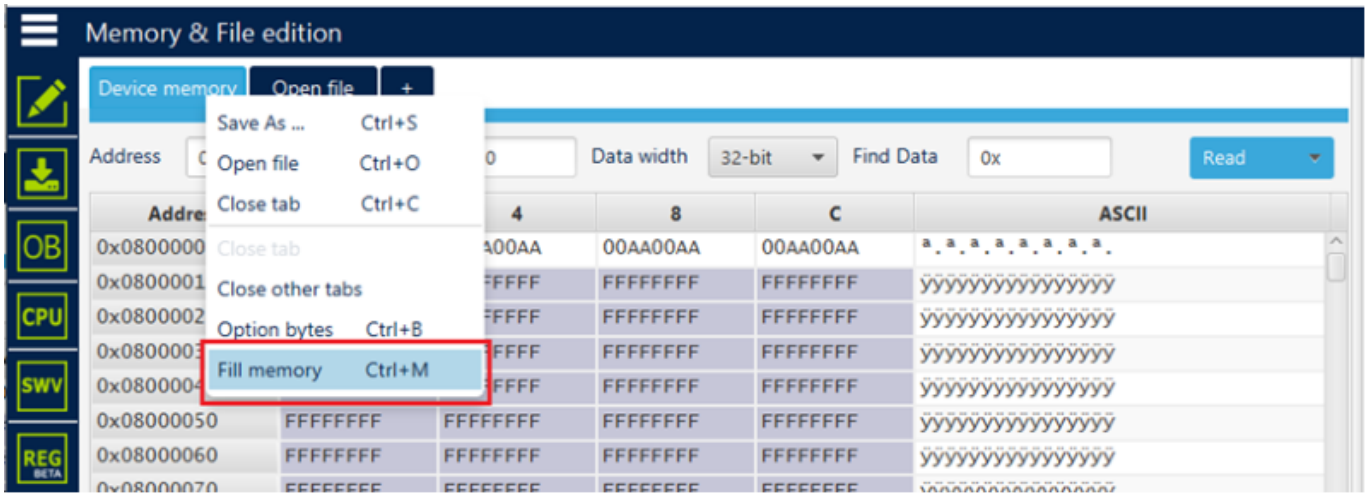


Fig. 3.48: Sub-menu displayed with right click on Device memory tab

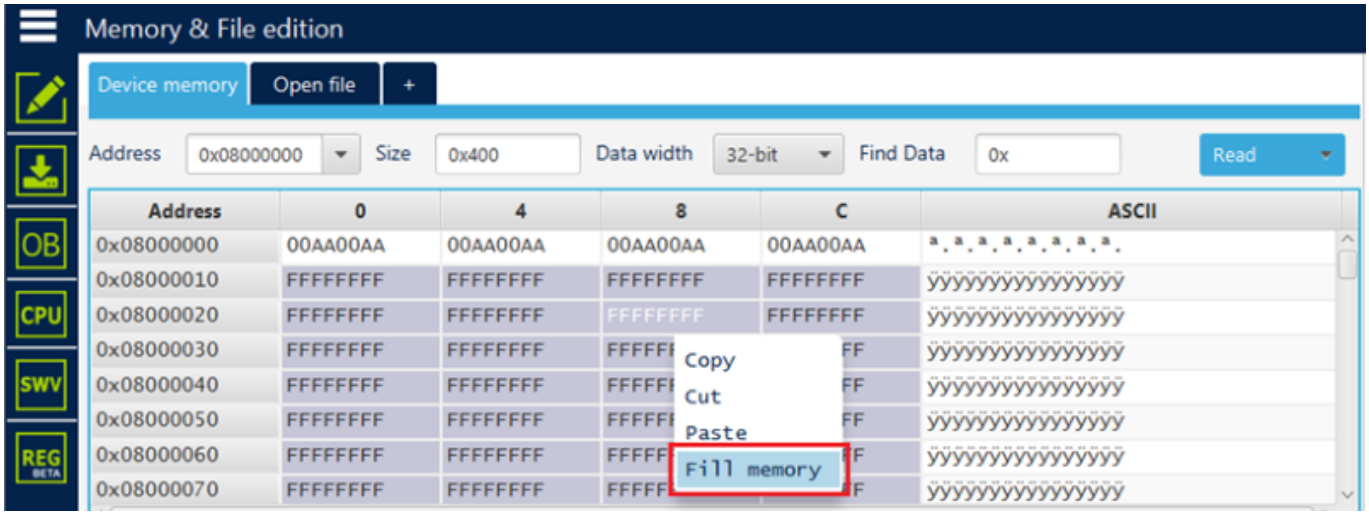


Fig. 3.49: Sub-menu displayed with right click on the cell of grid

Note

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

After clicking on `Fill memory` option, a window is displayed so that the user can initialize the parameters of the operation, as shown in the following figure.

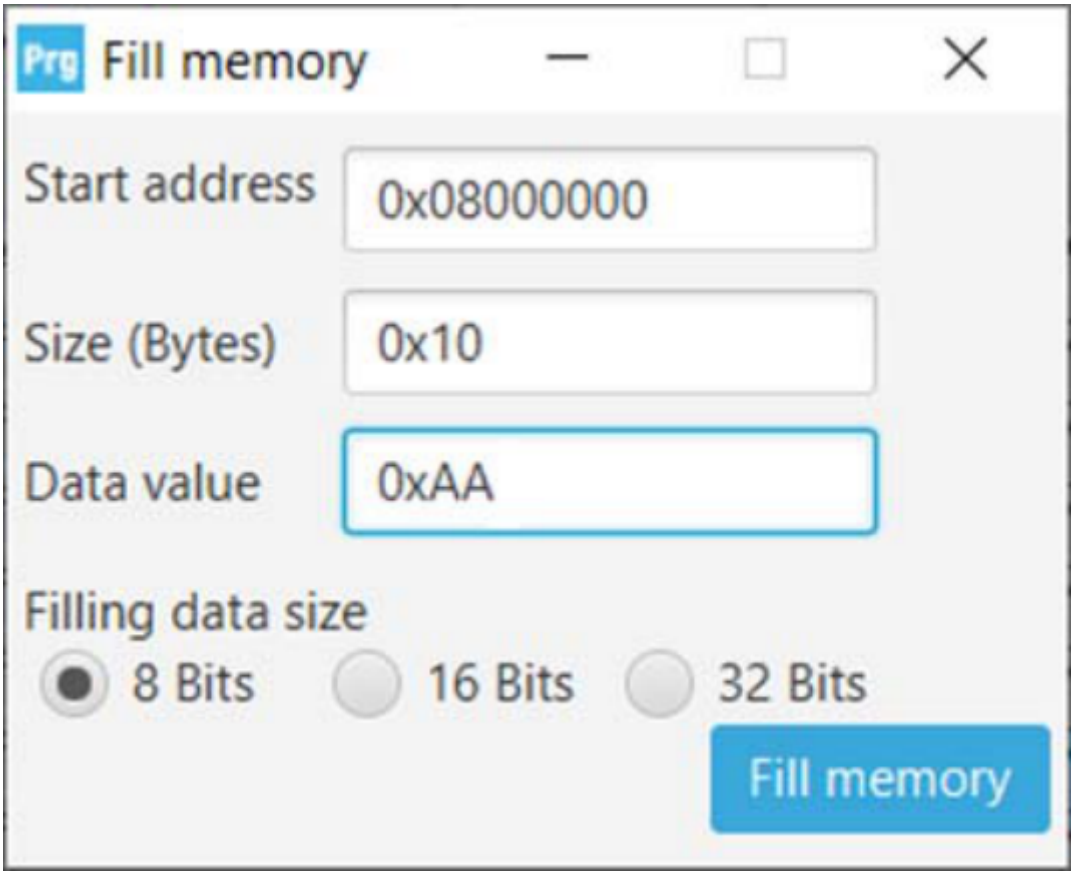


Fig. 3.50: Parameters initialization

Blank check operation

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

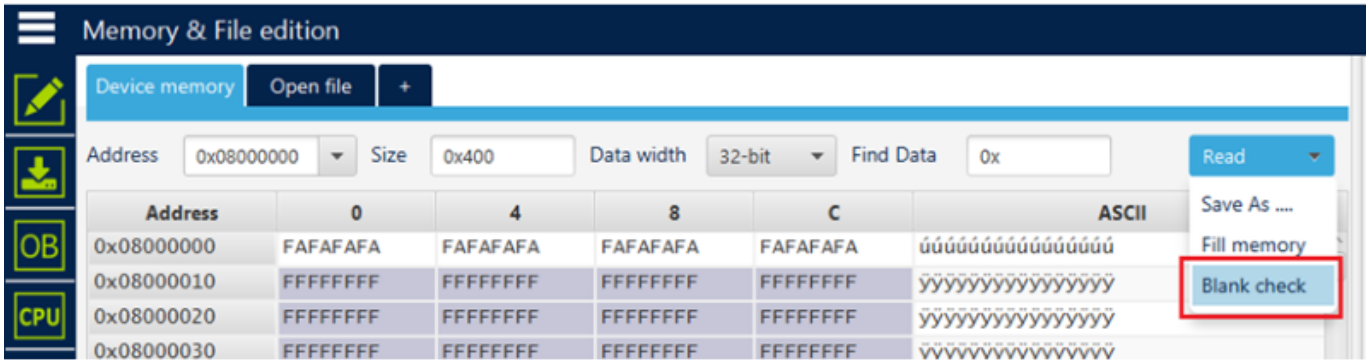


Fig. 3.51: Sub-menu displayed from Read combo-box

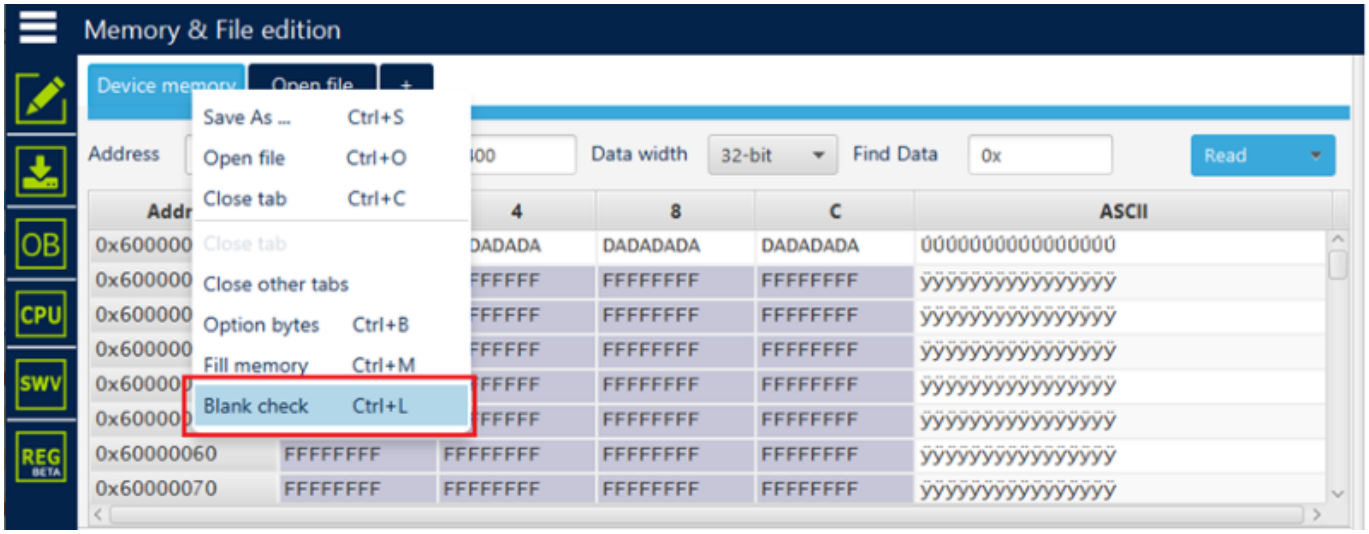


Fig. 3.52: Sub-menu displayed with right click on Device memory tab

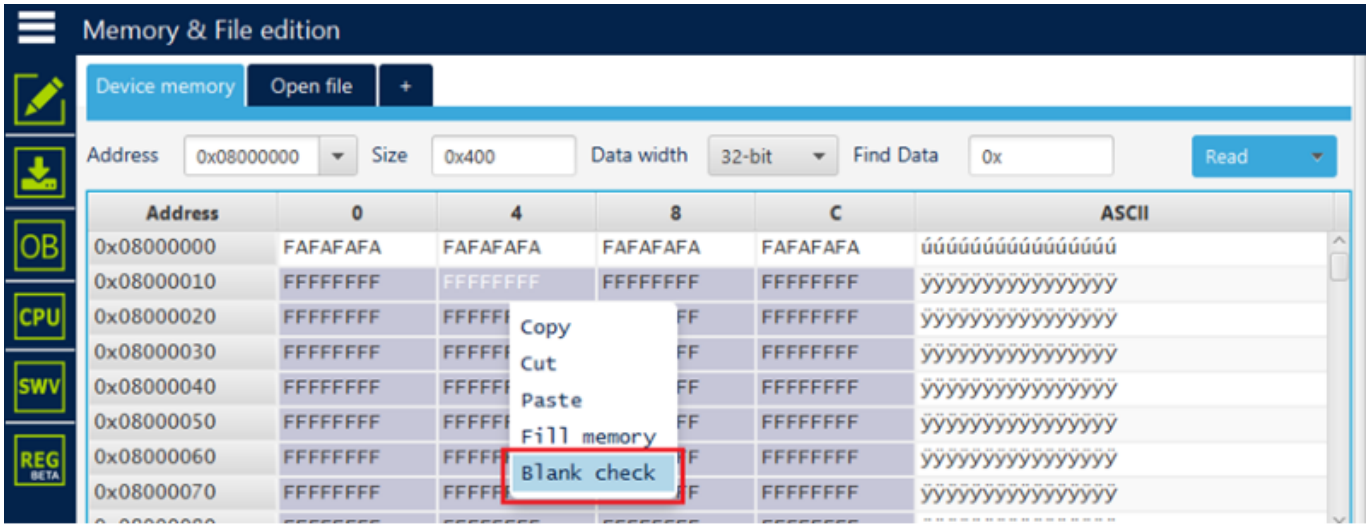


Fig. 3.53: Sub-menu displayed with right click on the cell of grid

Note

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

After clicking on Blank check sub-menu, the process starts to verify that the STM32 flash memory is blank. If the flash memory is not blank, the first address with data is highlighted in a message, as shown in the following figure.

The expected results are shown in the following figures.

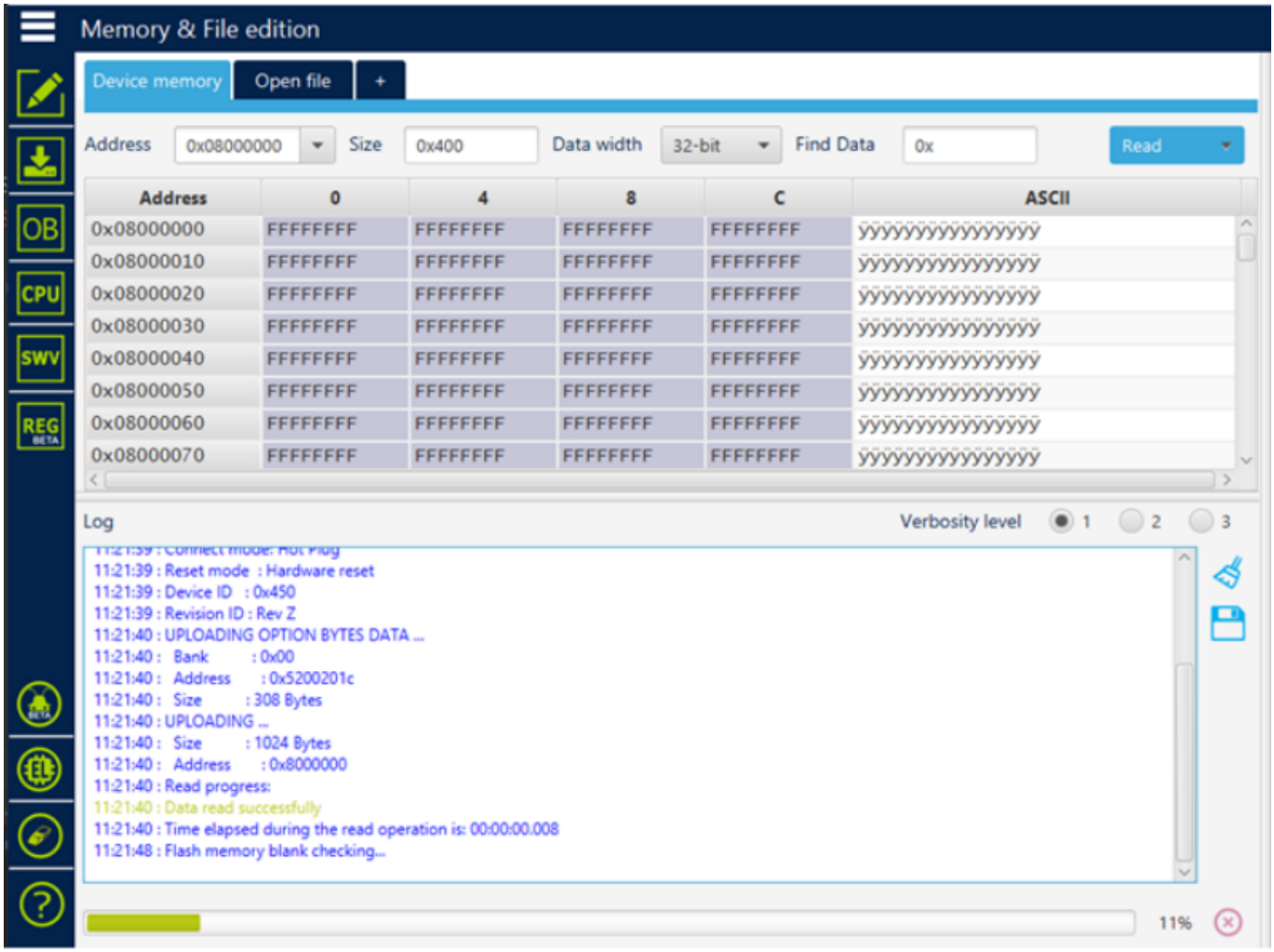


Fig. 3.54: First address with data

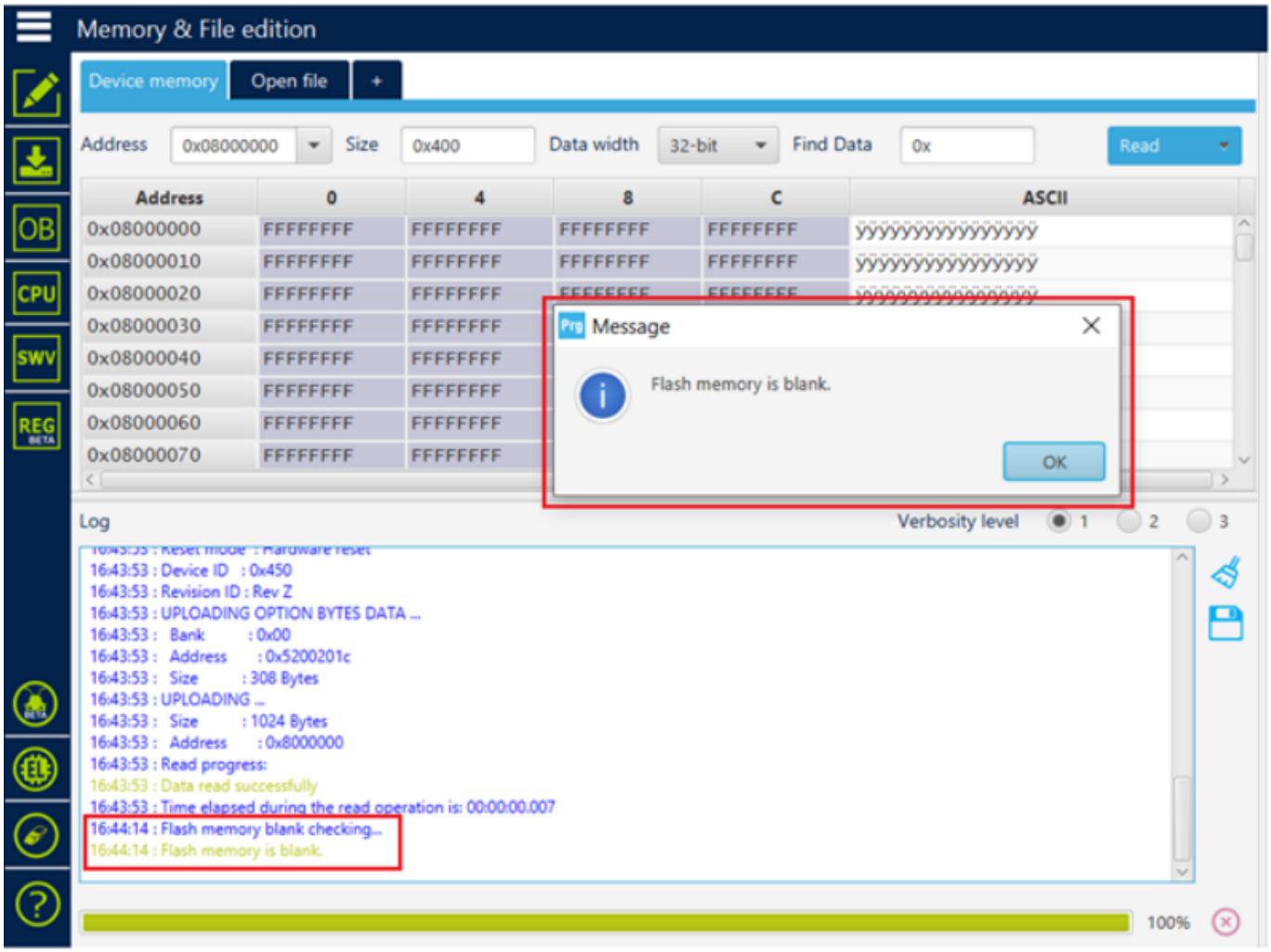


Fig. 3.55: Example 1: memory is blank

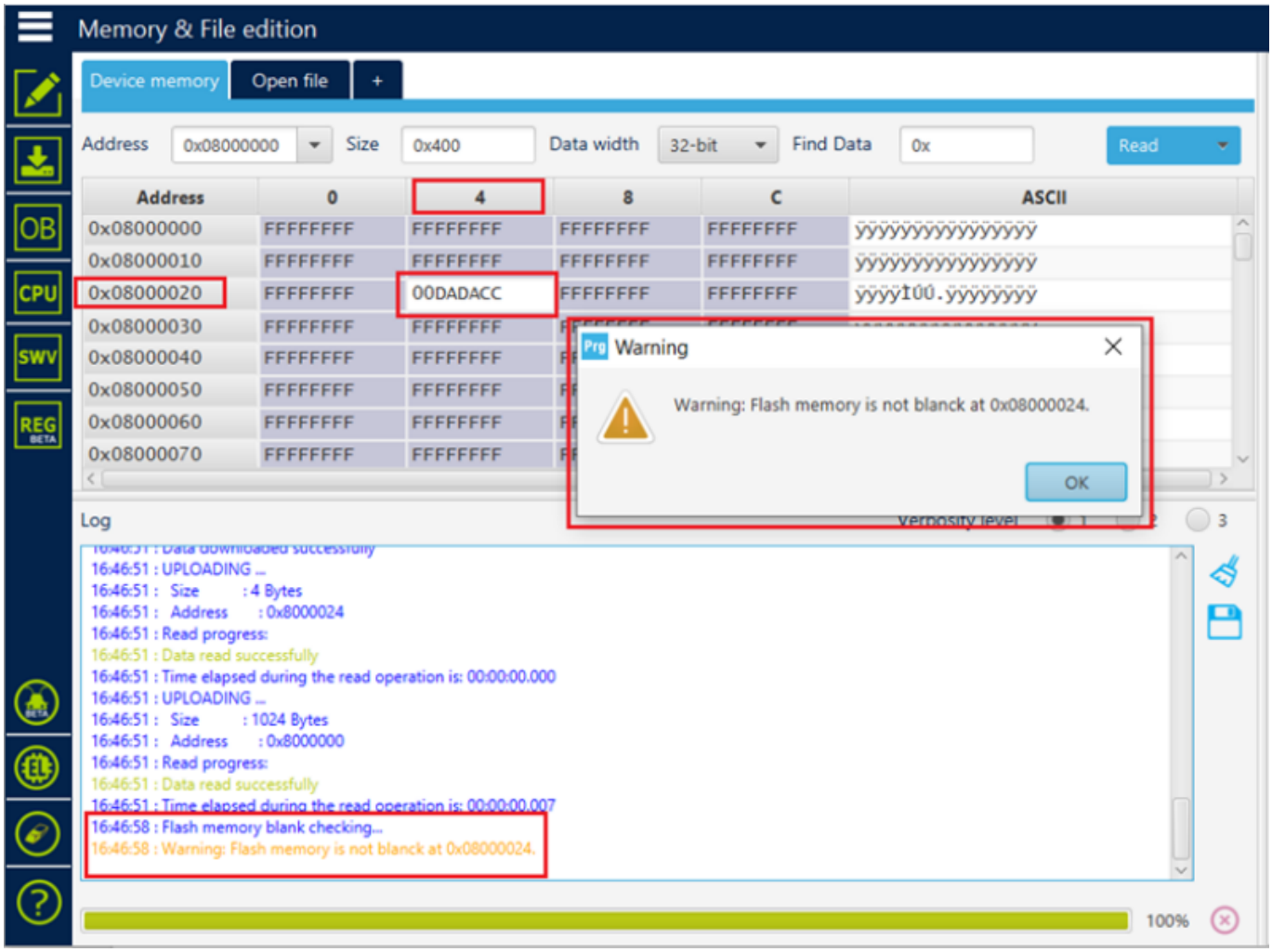


Fig. 3.56: Example 2: memory is not blank

Save all memory

This feature allows the user to save the contents of the Internal NVM and/or the configured External NVM loaders into a single `.hex` file.

To open this window, connect a device, then select **Save all** from the **Read** menu in the Device Edition window.

- **Save Internal NVM data**: By default, the **Internal NVM** checkbox is selected. Clear it if you do not want to save internal NVM data.
- **Save data from one or more External NVM loaders**: By default, the **External NVM** checkbox is cleared. When selected, it enables the loader list, which displays all loaders configured in the External Loaders window. By default, all loaders are selected.
- **Automatically generate a .hex output file**: The output file always uses the `.hex` extension. If another extension is provided, it is automatically replaced.

Notes:

- Internal NVM saving is not yet supported in USB-DFU-IAP mode.
- The **Save** button is enabled only when an output path is set and at least one NVM source is selected.

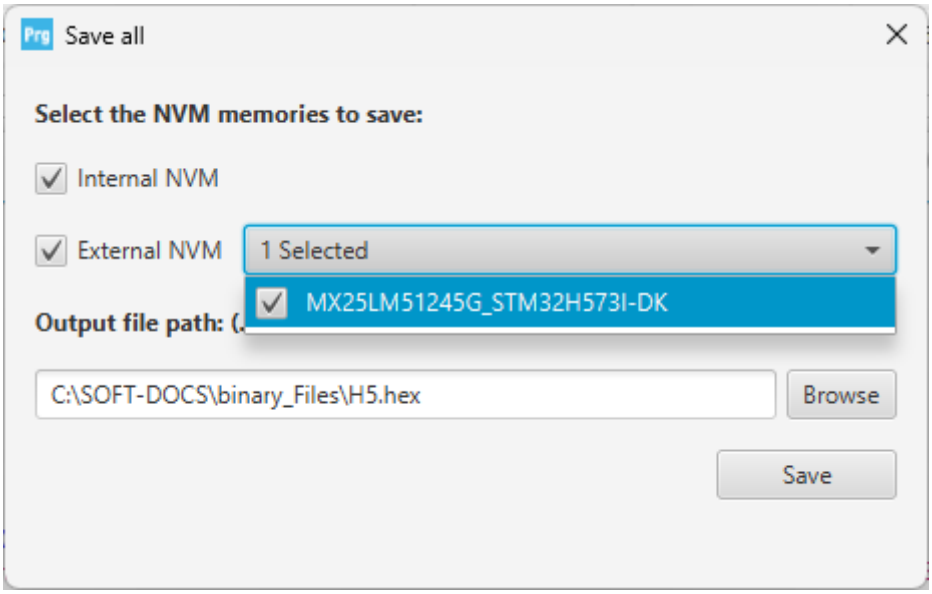


Fig. 3.57: Save all memory window

Compare flash memory with file

Description: compares the MCU device memory content with a binary, hex, srec, elf, out, or axf file. The difference is shown in red in the file and in the flash memory panel.

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

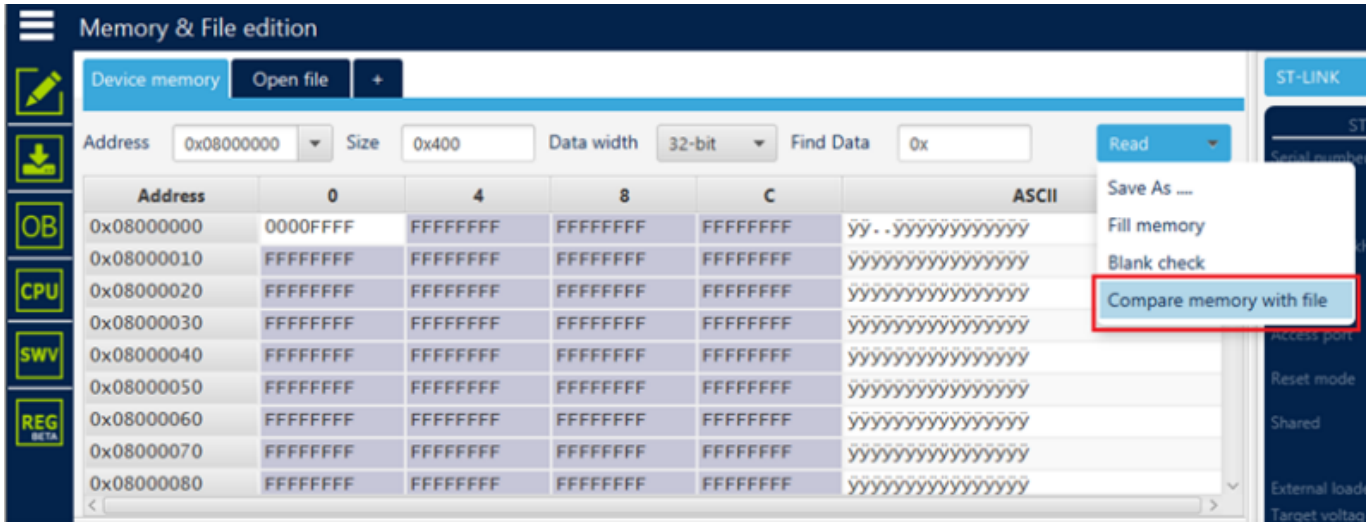


Fig. 3.58: Sub-menu displayed from Read combo-box

Note

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

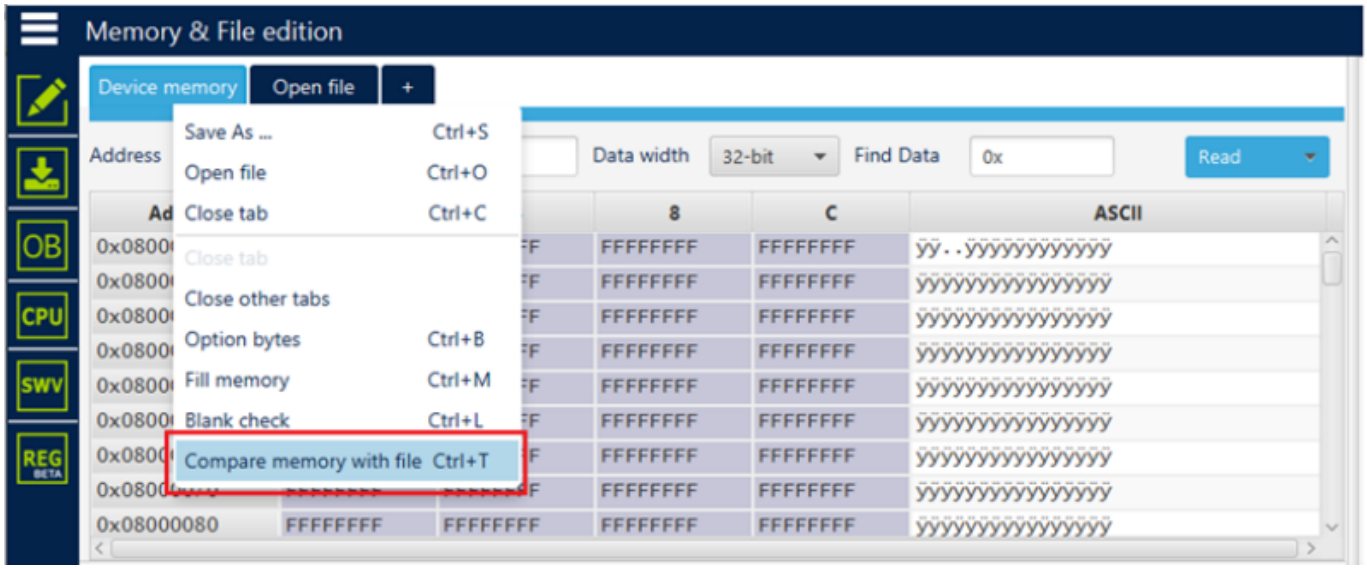


Fig. 3.59: Sub-menu displayed with right click on Device memory tab

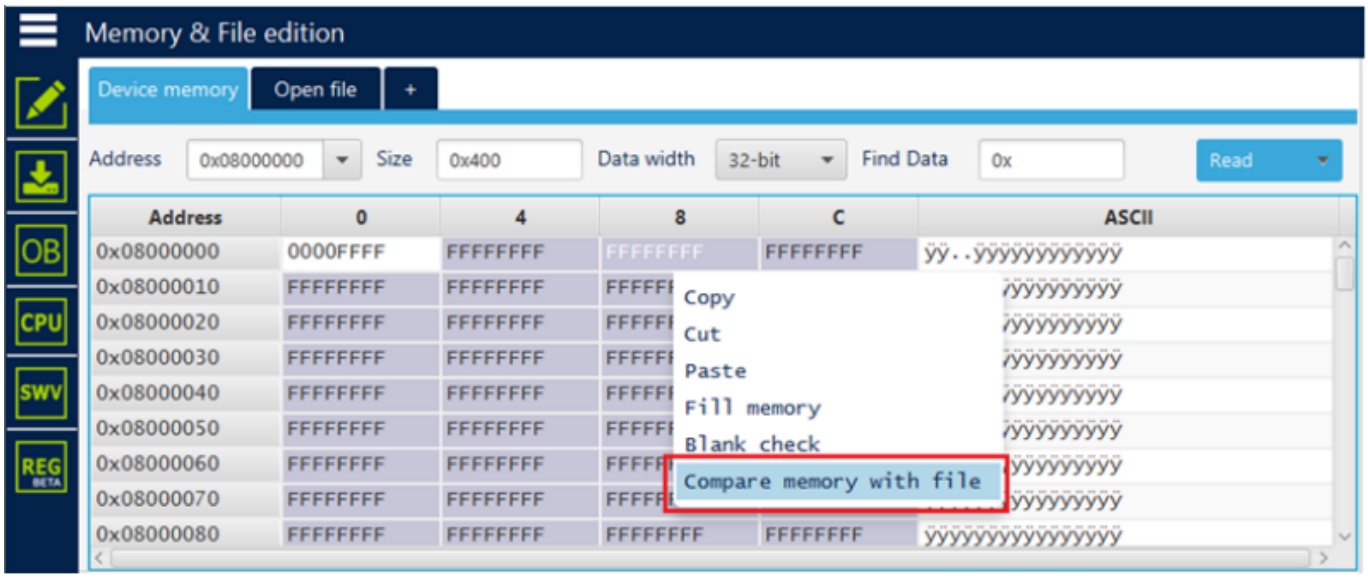


Fig. 3.60: Sub-menu displayed with right click on the cell of grid

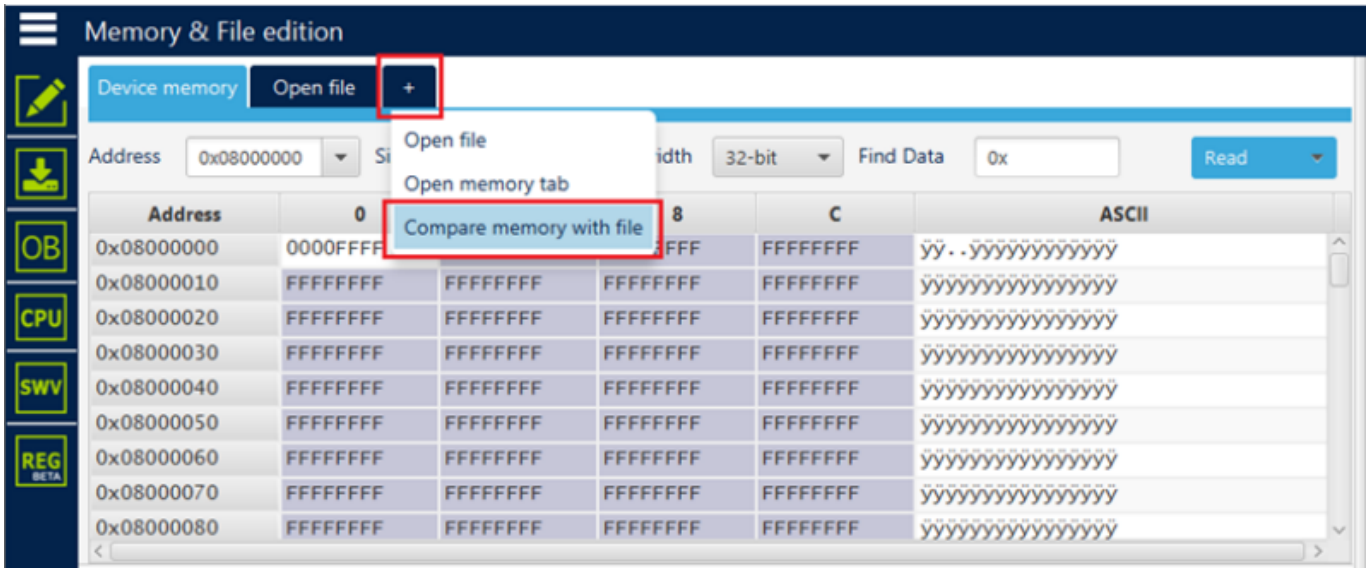


Fig. 3.61: Sub-menu displayed with add tab button

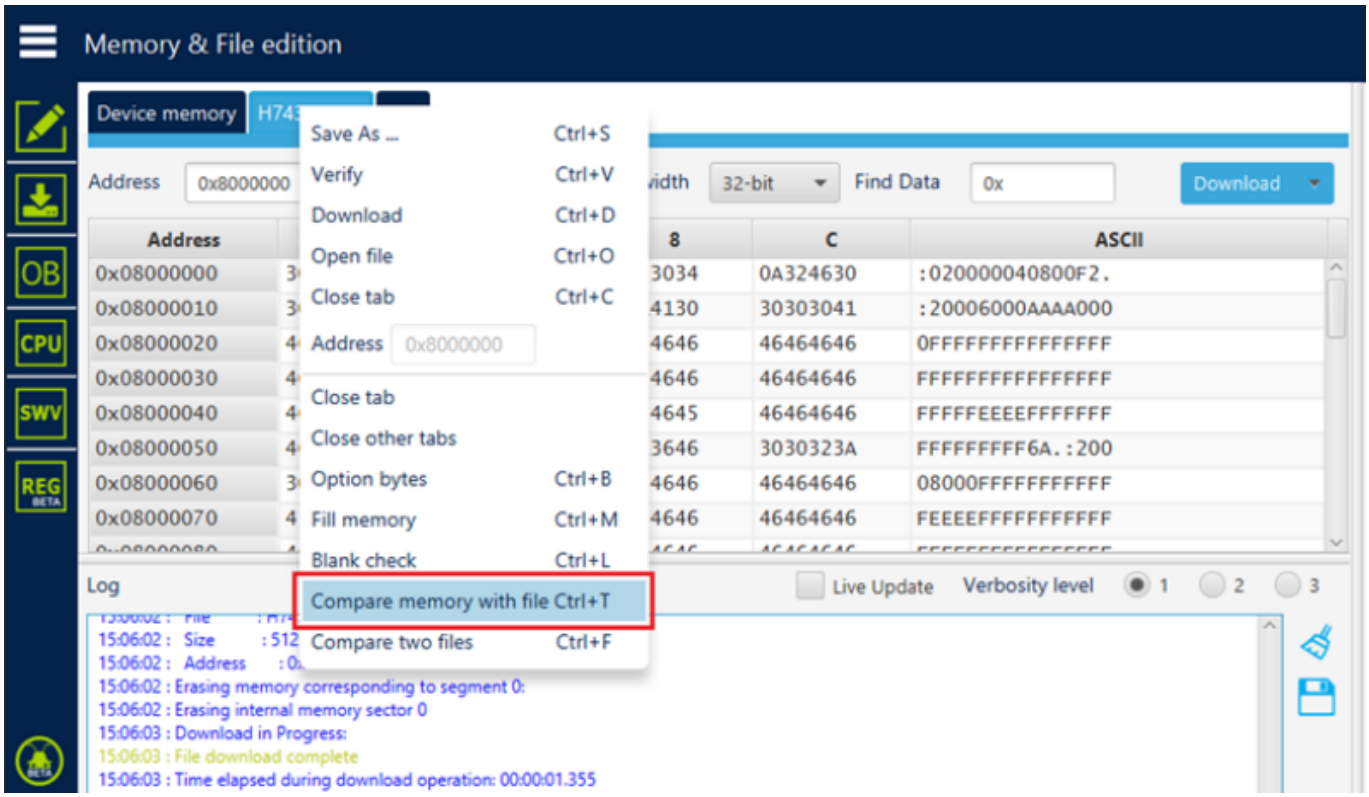


Fig. 3.62: Sub-menu displayed with right click on the opened file tab

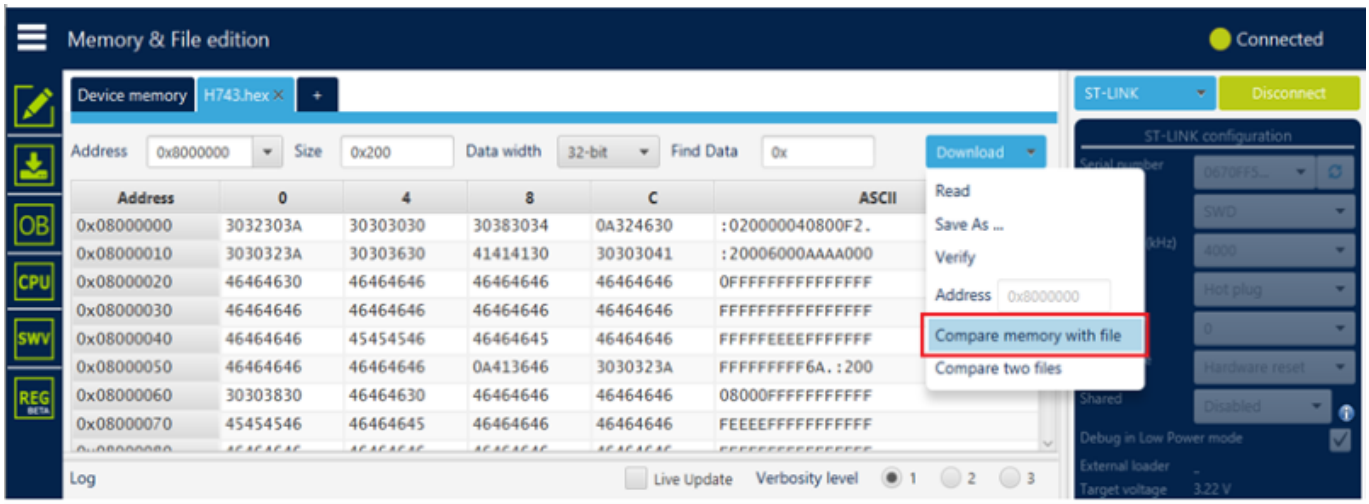


Fig. 3.63: Sub-menu displayed from Download combo-box displayed in file tab

Example 1: Difference between internal flash memory and binary file

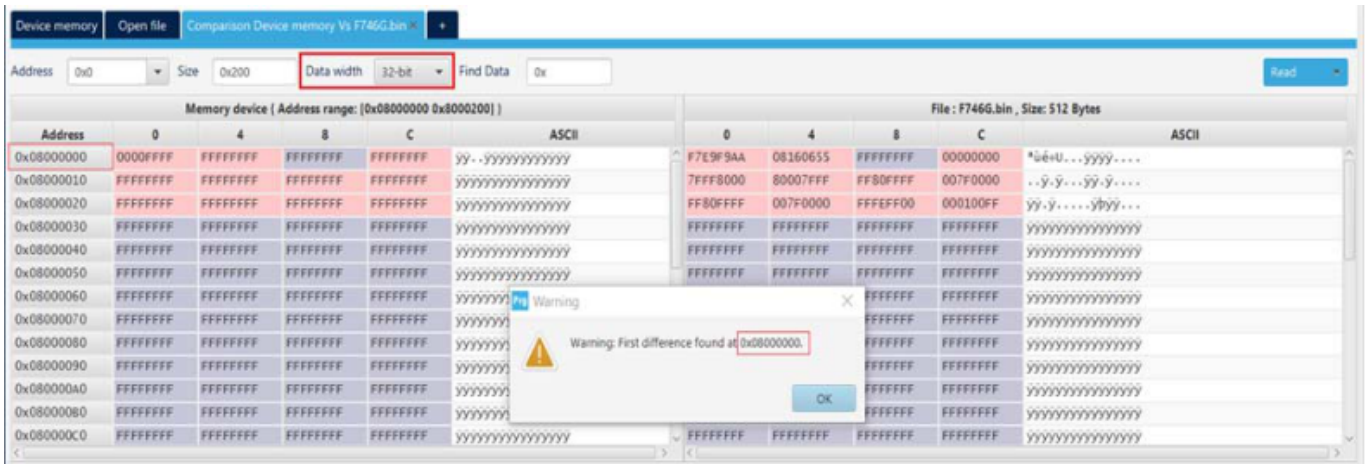


Fig. 3.64: Data width: 32 bits

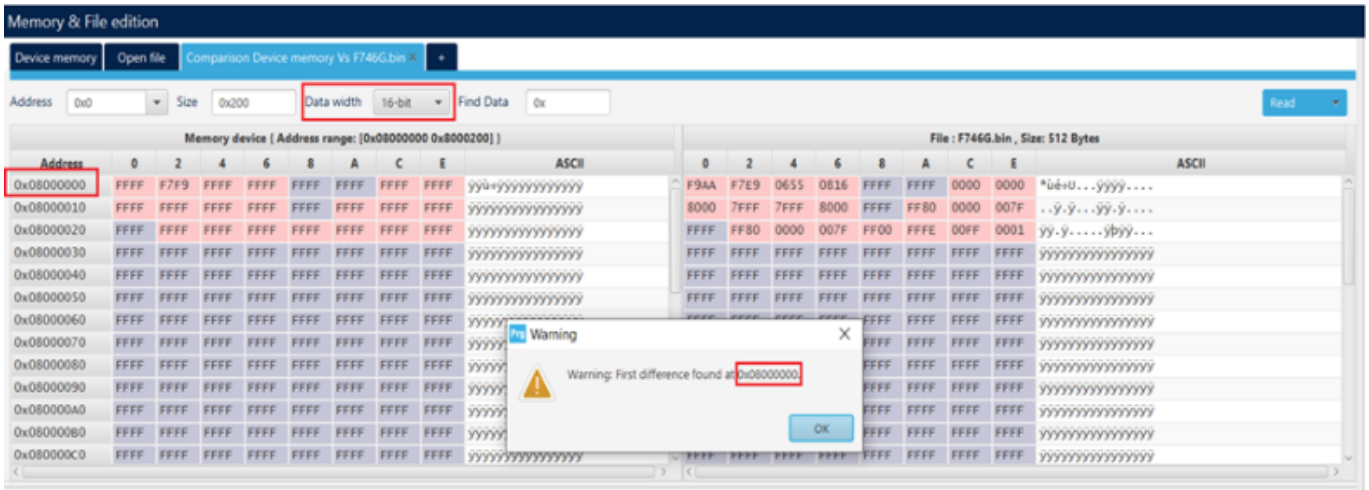


Fig. 3.65: Data width: 16 bits

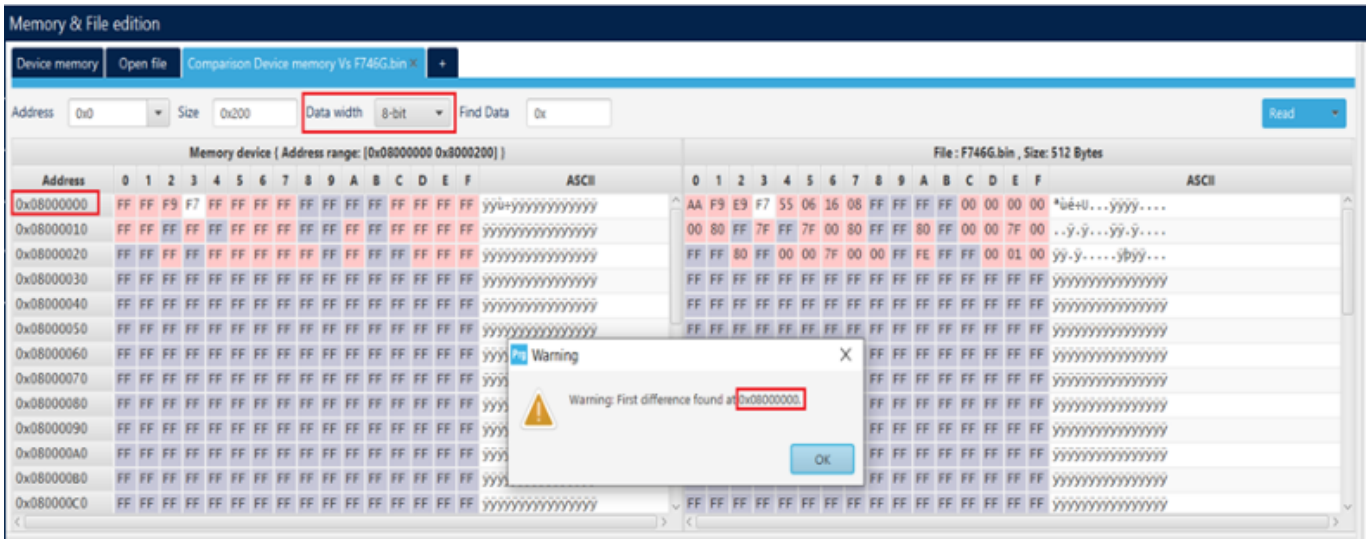


Fig. 3.66: Data width: 8 bits

Example 2: Difference between external flash memory and hex file

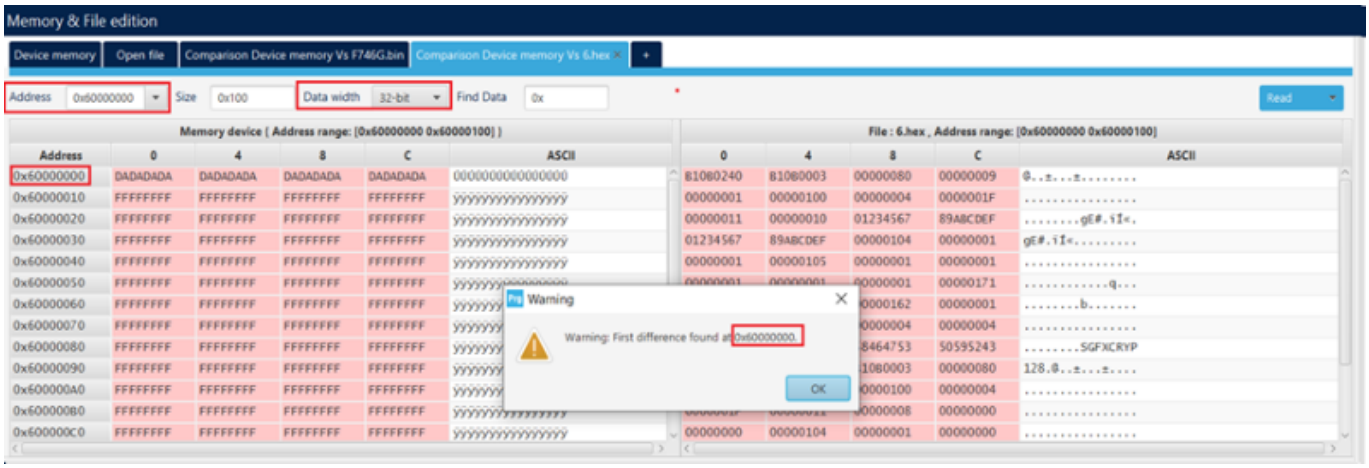


Fig. 3.67: Data width: 32 bits

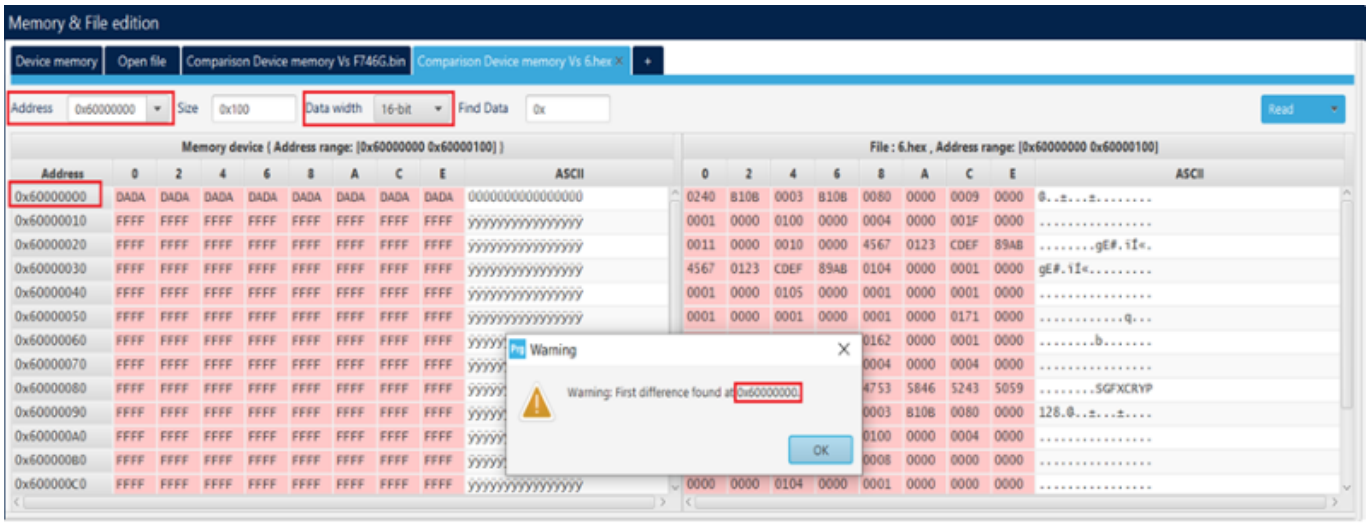


Fig. 3.68: Data width: 16 bits

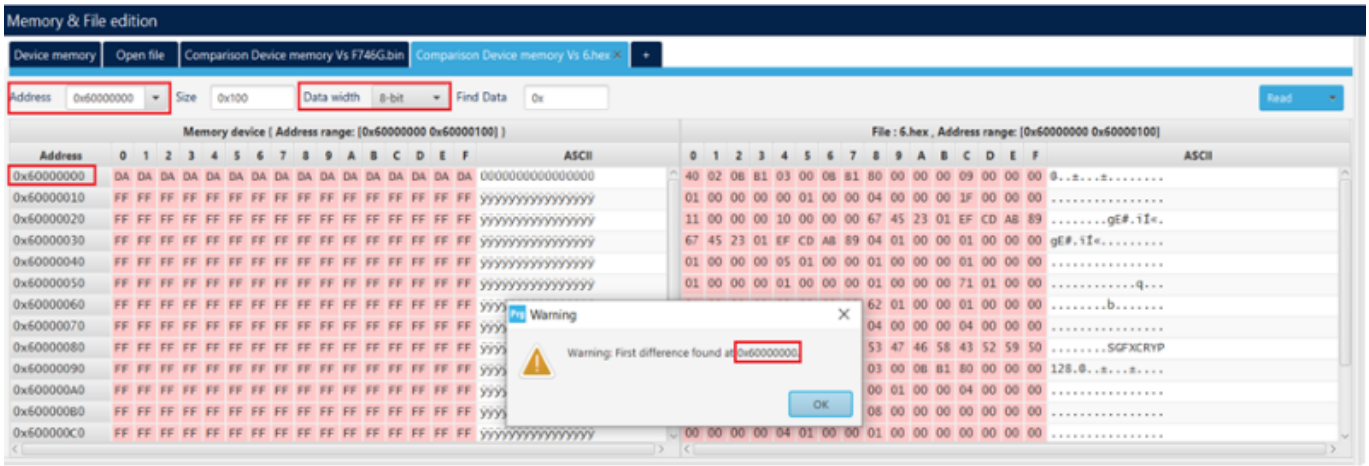


Fig. 3.69: Data width: 8 bits

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

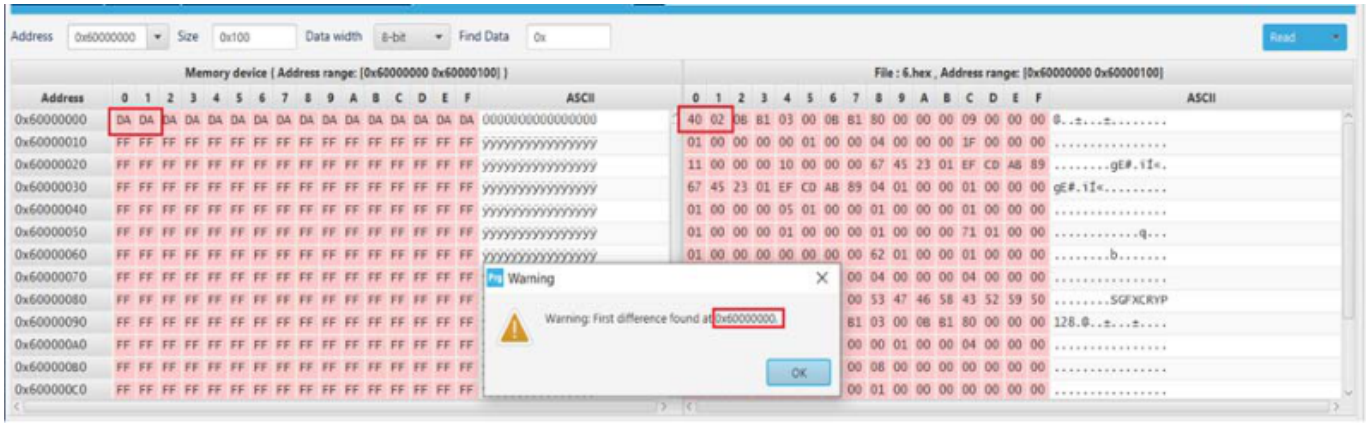


Fig. 3.70: Before editing the flash memory

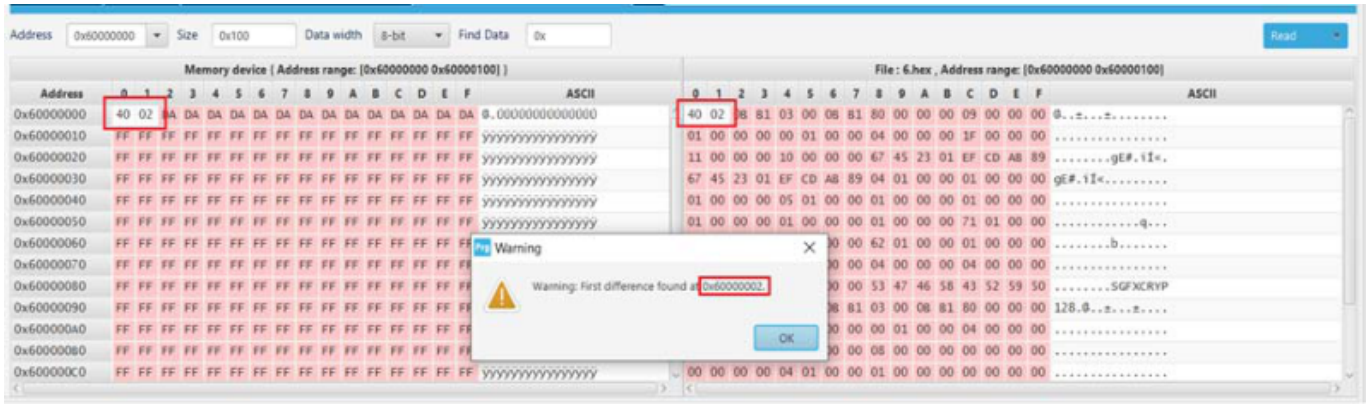


Fig. 3.71: After editing the flash memory

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

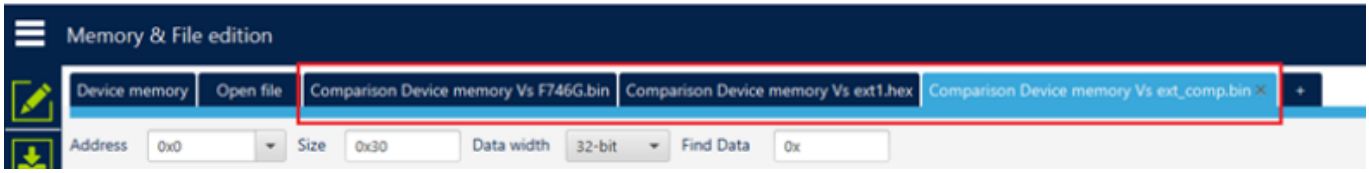


Fig. 3.72: Multiple comparisons

Note

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

Comparison between two files

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

This operation does not need a connected board.

The used files can be of different sizes and types.

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

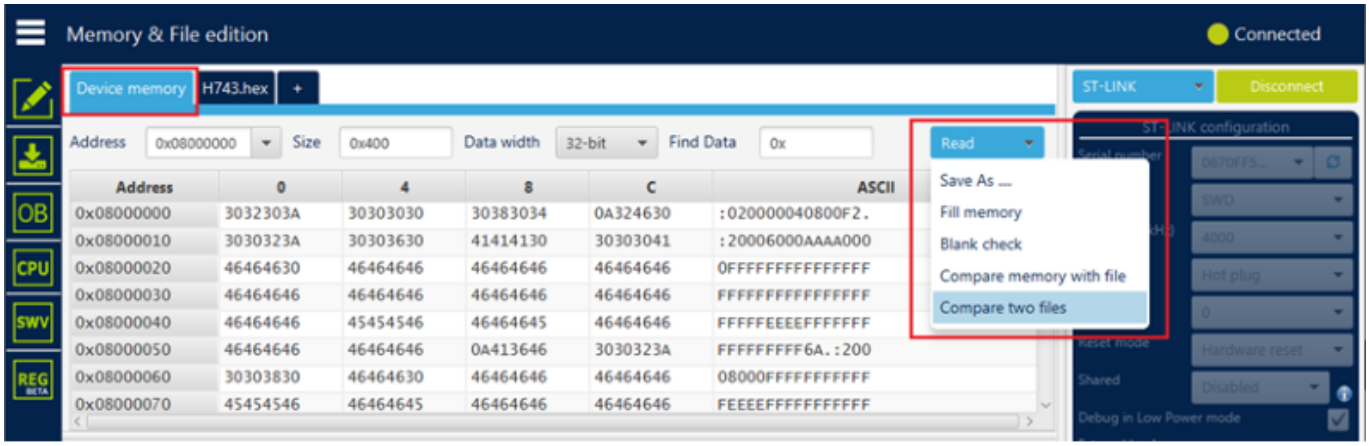


Fig. 3.73: Sub-menu displayed from Read combo-box in device memory tab

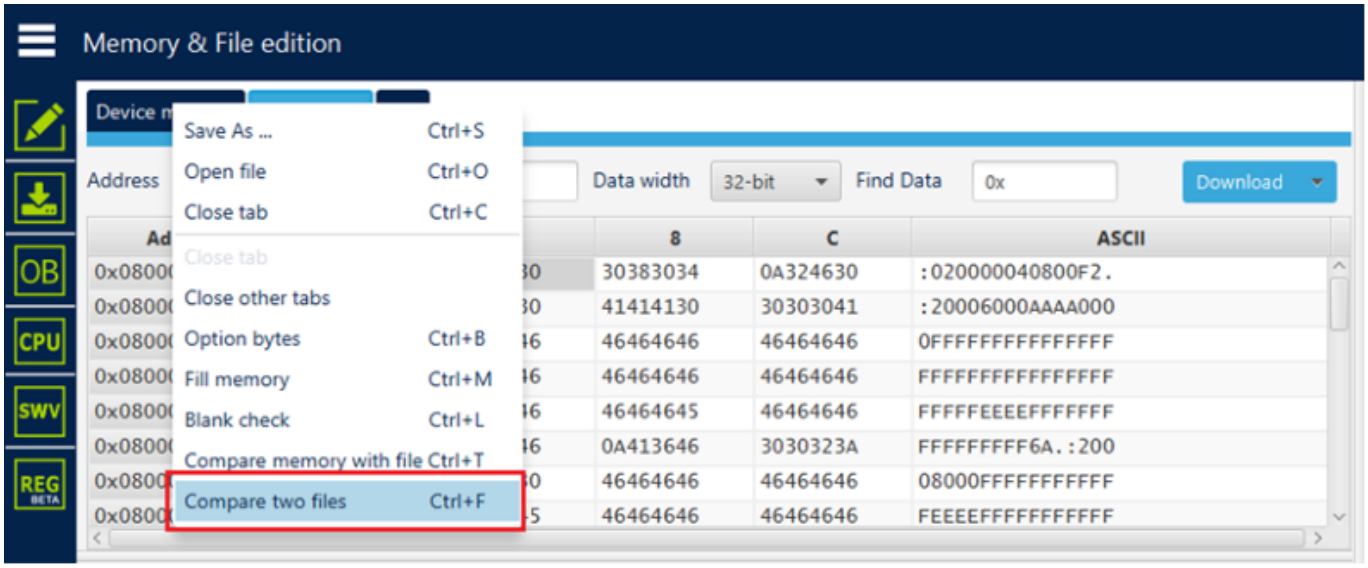


Fig. 3.74: Sub-menu displayed with right click on Device memory tab

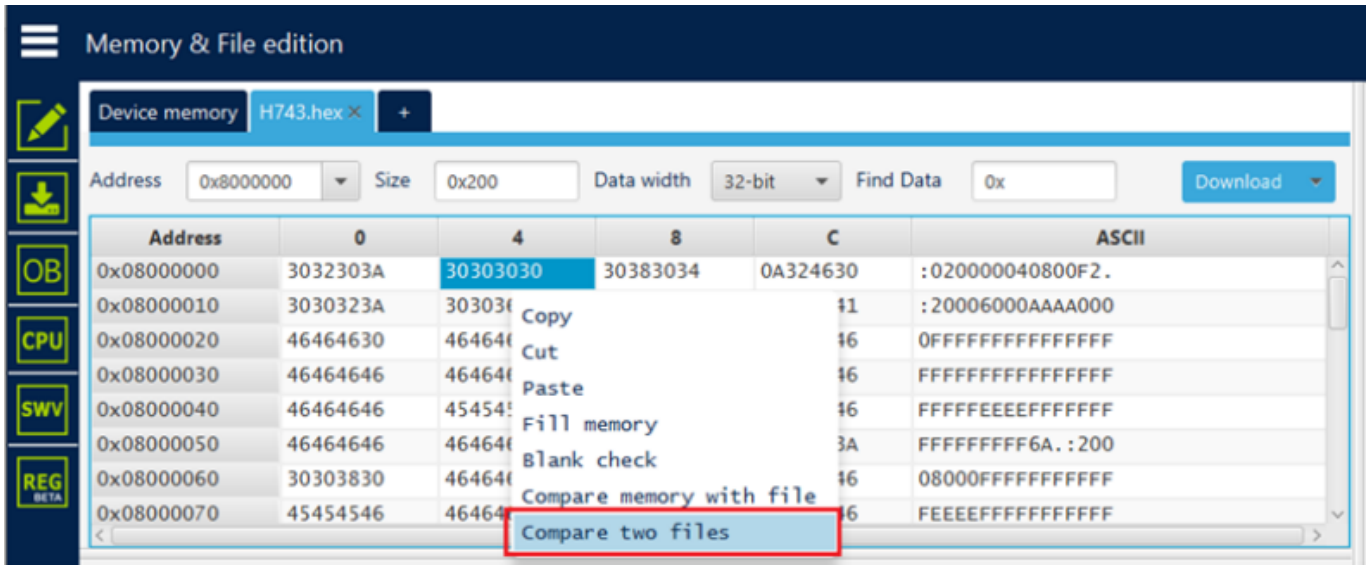


Fig. 3.75: Sub-menu displayed with right click on the cell of grid

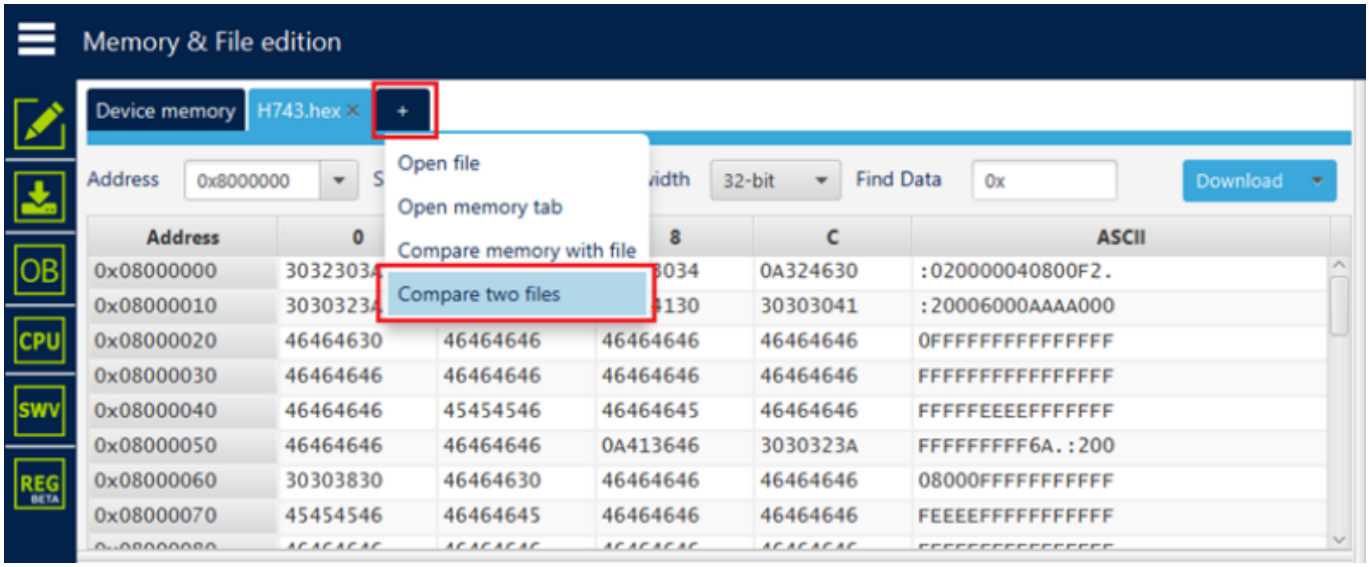


Fig. 3.76: Sub-menu displayed with add tab button

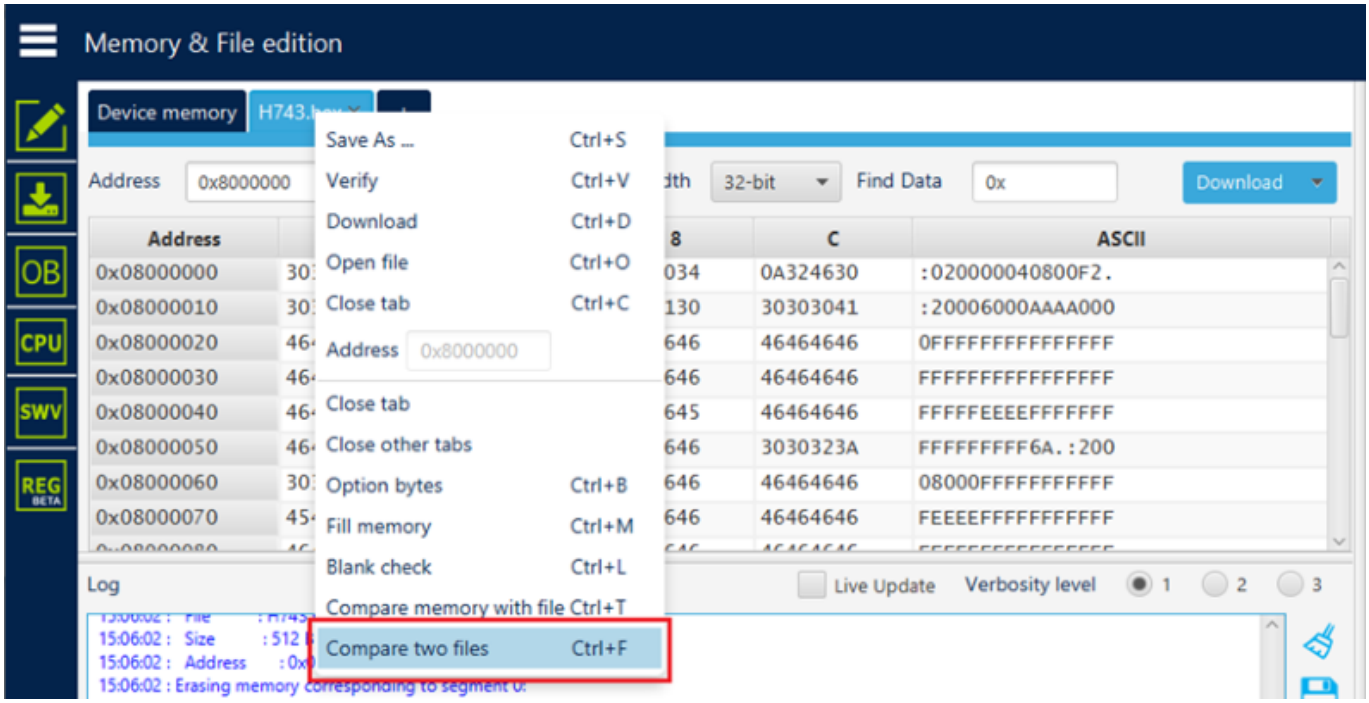


Fig. 3.77: Sub-menu displayed with right click on the opened file tab

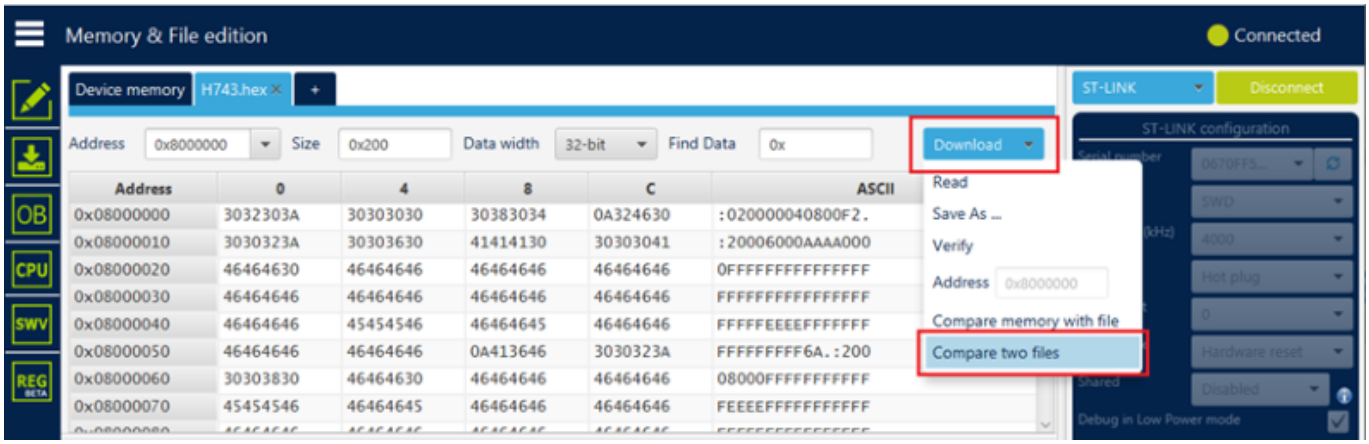


Fig. 3.78: Sub-menu displayed from Download combo-box displayed in file tab

Note

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

Example: Difference between two files of the same type and different sizes

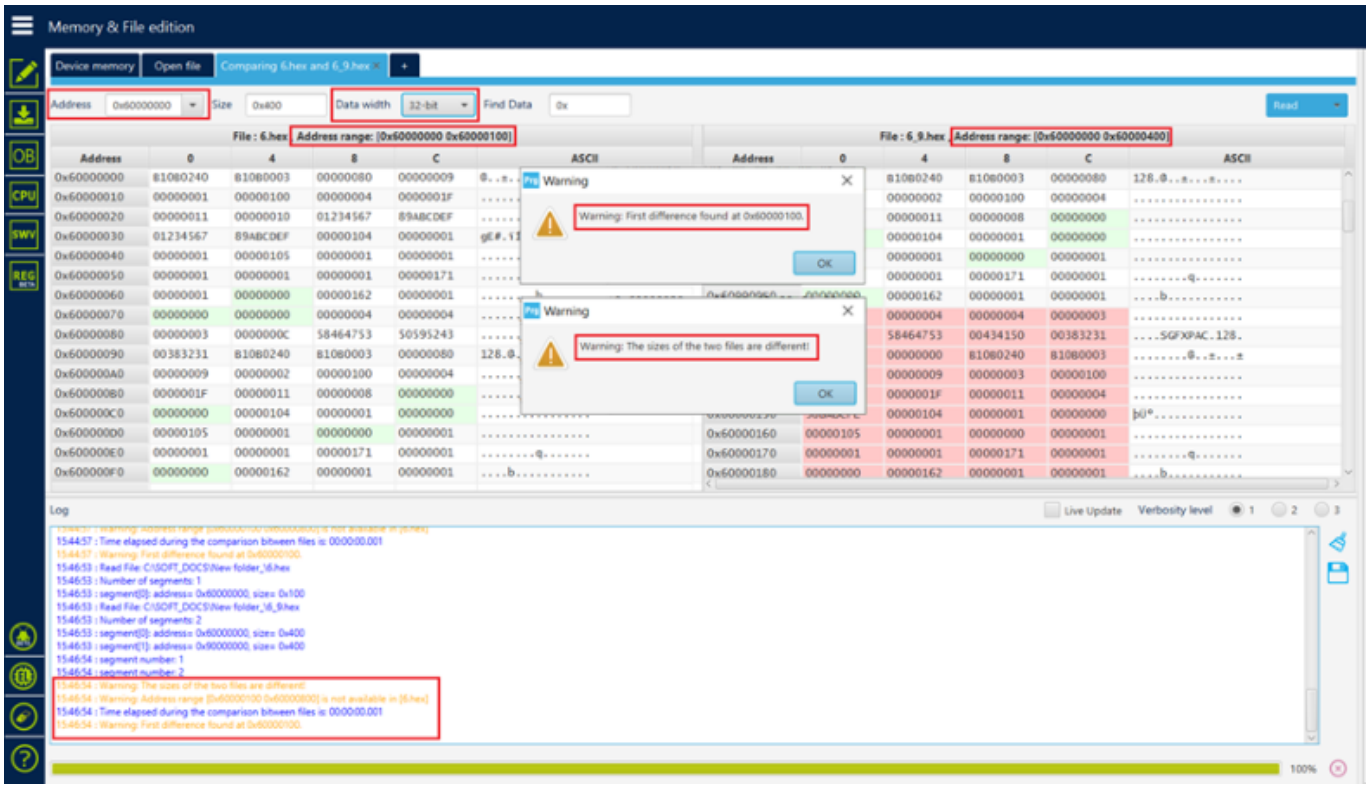


Fig. 3.79: Data width: 32 bits

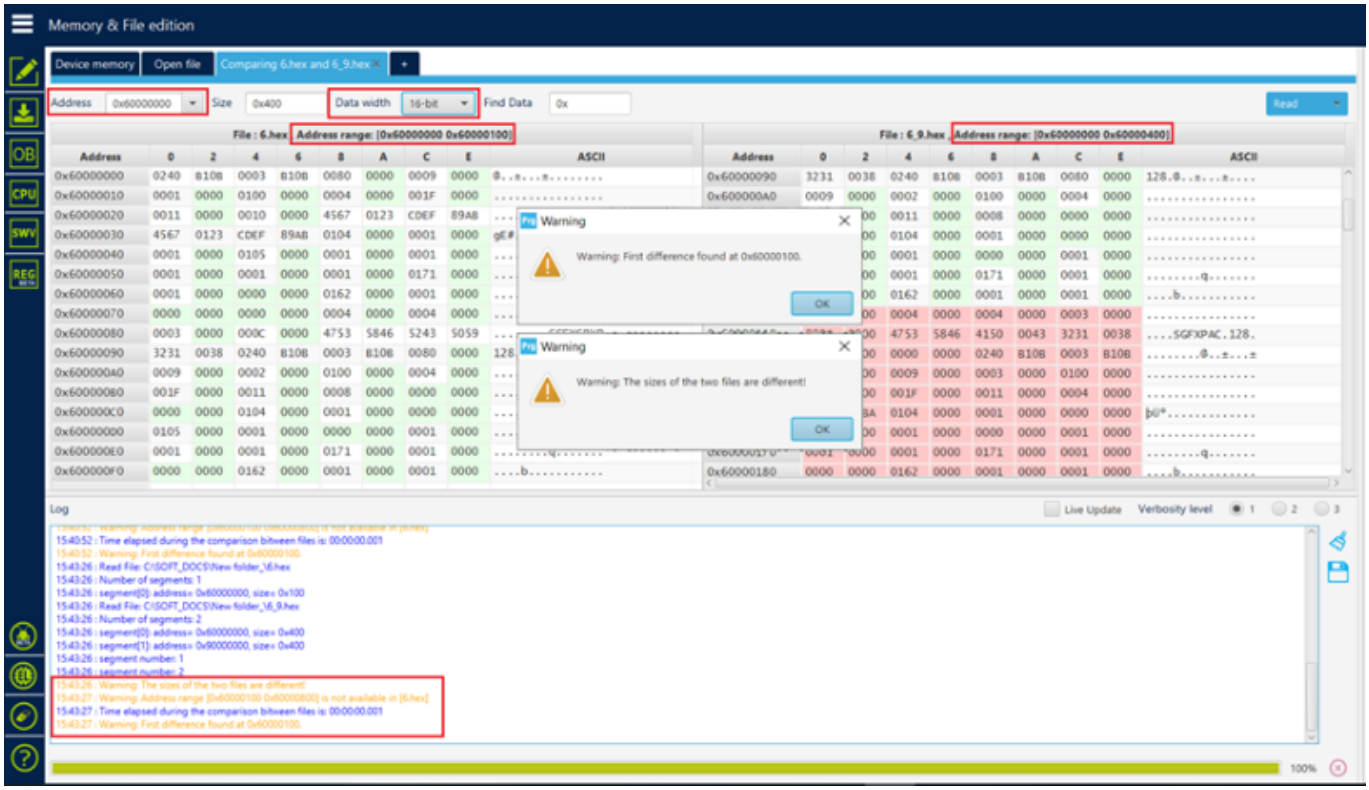


Fig. 3.80: Data width: 16 bits

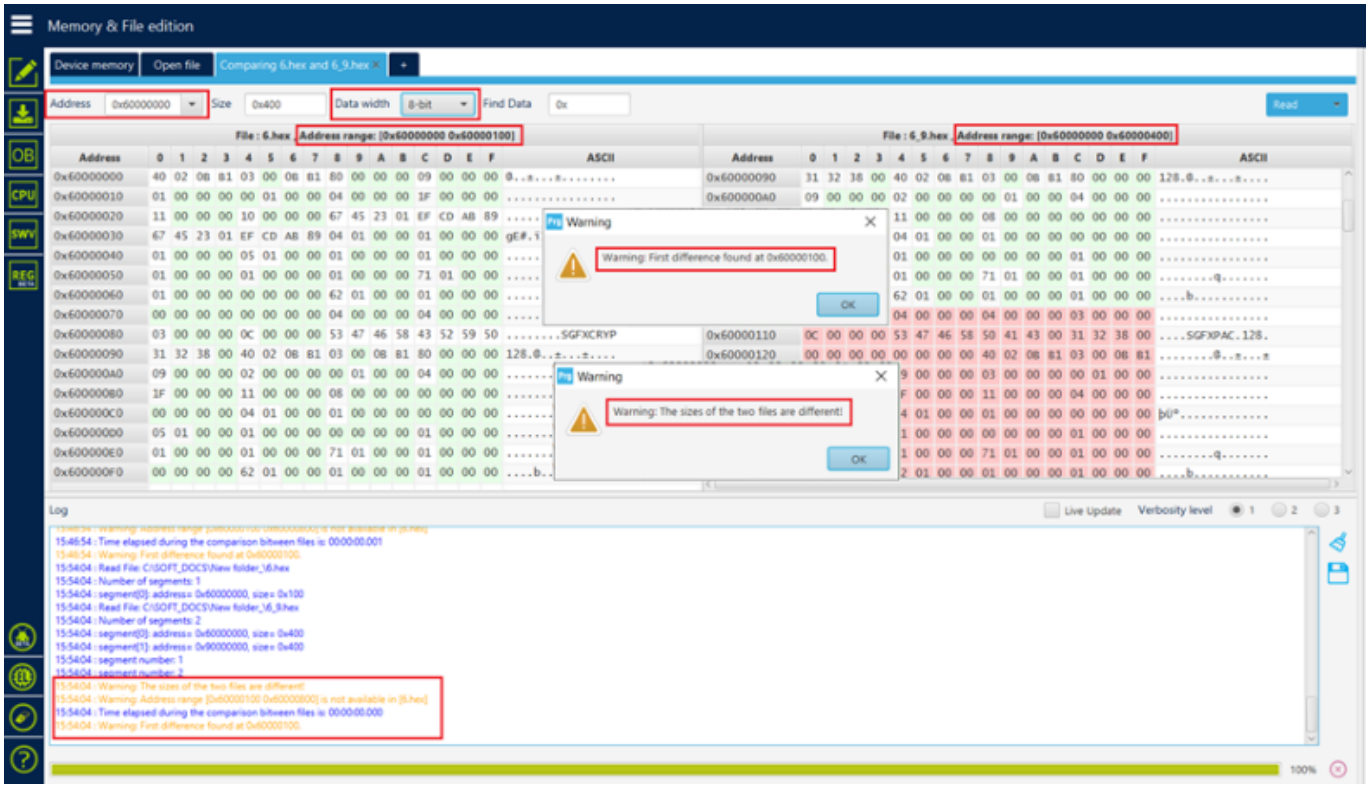


Fig. 3.81: Data width: 8 bits

Note

The user can make multiple comparisons between files.

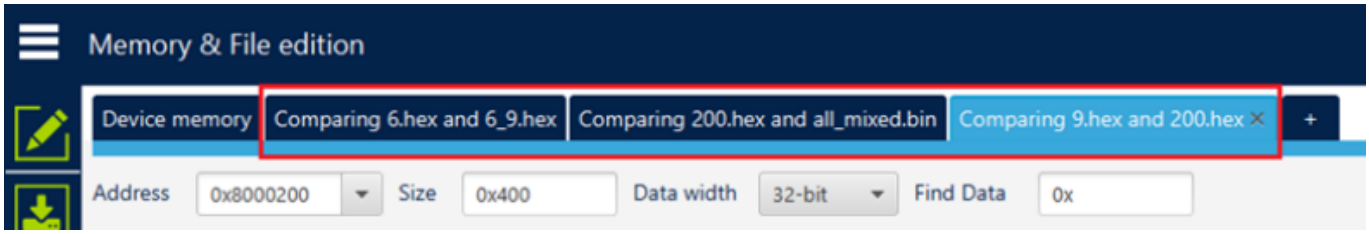


Fig. 3.82: Multiple comparisons

LiveUpdate feature

`-liveUpdate` checkbox

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

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

Note

Live update illustration is not available in this repository snapshot.

Calculator

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

1. Number converter

   Composed of several graphical components, to facilitate the number conversion between decimal, hexadecimal, and binary formats.

   Use the 32 check boxes, representing a word of 32 bits, to activate or deactivate the relevant bit or bits.

   Use the `Reset` button to reinitialize the number to 0.

   Any time a bit is changed, the number value is updated.

2. Checksum calculation

   To calculate the checksum value, based on addition algorithm applicable on the file content or on the flash memory.

   File Checksum: choose your binary file and click on `Calculate` button to display the corresponding result on the log panel.

   Flash Memory Checksum: calculates the checksum value of a region, based on start address and size of the desired region, once the device is connected.

   To calculate the full flash memory checksum retrieve the memory size, displayed in the `Flash size` field.

3. Memory programming

   To expose the generic memory edition options.

   Fill memory: see Section 2.16.

   Blank check: see Section 2.18.

Note

Calculator window illustration is not available in this repository snapshot.

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

Import/Export project settings

The user can load or store the settings below from or to a `.prog` project file:

- Connection settings, only if the board is connected.
- Firmware's list.
- External flash loaders.
- Settings used for the automatic mode.
- Settings of STM32WB stack install, only when connecting STM32WB devices.
- RSSe binaries.
- Option bytes values, only if the board is connected.

To import the settings, choose the settings file path and check at least one option.

> **Note**
>
> Import settings interface illustration is not available in this repository snapshot.

If the device ID saved in the imported project settings does not match the connected device, a warning message appears.

After importing the option bytes settings from the `Import/Export Settings` tab, apply them from the Option Bytes tab by clicking the `Take OB from Project Settings` button.

To export the settings, choose at least one option.

OTP programming window for STM32N6

For generic usage of this window, refer to Section 4.3. STM32N6 devices support 367 OTP words, and STM32CubeProgrammer allows users to program these OTP with the same commands as MPU or the user interface.

- **Via Debug interfaces: use the external loader `OTP_FUSES_STM32N6xx` from** the `ExternalLoader` panel, selected by default when you connect via ST-Link.
- **Via BootROM: STM32CubeProgrammer needs the TSV file that includes the** OpenBootloader for OTP programming. For more information refer to STMicroelectronics/stm32-mw-openbl.

Example of OTP programming

The tool loads the OpenBootloader as requested in the TSV file using the embedded BOOTROM. Once the OpenBootloader is running, you can manipulate the entire OTP using the same commands as the MPU.

Command for launching the OpenBootloader: `-c port=USB1 -d file.tsv`

Example of TSV file to launch the OpenBootloader:

```
#Opt Id  Name        Type  IP    Offset  Binary
P    0x1 FSBL Binary none  0x0    OpenBootloader_STM32N6-DK_OTP_Cut2-Signed.stm32
```

External flash memory window for STM32N6

STM32N6 devices can be connected via ST-Link (JTAG/SWD) and via BootROM (USB/UART). The external flash memory can be programmed in two ways:

1. **Via ST-Link: select the corresponding external flash loader from the** `ExternalLoader` GUI panel to perform programming, write, erase, and read operations with an external memory.
2. **Via BootROM: this is used to sequentially load the partitions requested by** the BootROM. To achieve this, STM32CubeProgrammer requires the TSV file, which contains the OpenBootloader for the external memory programming, the corresponding external flash loader, and the data to be loaded. Refer to GitHub for more details.

STM32V8 NVM programming constraints

STM32V8 PCM `Power pulse` programming mode presents extra programming constraint: only four attempts are allowed to program the internal memory, including OTP programming.

When setting the Option Byte PM=1, the internal memory can be programmed only five times, four times for OTP programming, beyond this limit there is a risk of loosing the V8 devices. The tool automatically blocks this operation and raises a confirmation popup. The user must click `OK` to continue the operation or cancel it.

Note

STM32V8 PM activation illustration is not available in this repository snapshot.

Export MCU memory map

Description: the Export Memory Map window is designed to facilitate the creation and export of memory content in HEX format, only in GUI mode.

The tool allows users to:

- **Define one or more memory segments by specifying their base addresses and** sizes.
- **Select additional options such as option byte (OB) and one-time** programmable (OTP) memory regions for export.

Access the Export Memory Map window using any of the following methods:

- Keyboard shortcut `Ctrl + E`.
- **Right-click `Device Memory` from the `Memory & editing` window and choose** `Export memory`.
- Click the `Read` combo box to display items and select `Export memory`.

Note

Export memory map window illustration is not available in this repository snapshot.

Users can set the number of memory segments. For each segment, specify any of the following:

- Base Address: in hexadecimal format.
- Size: in decimal or hexadecimal format, multiple of 4 Bytes.
- Type: choose between:
  - `Memory` to export memory data.
  - `Pattern` to fill the segment with specific values.
- **Pattern: a value in hexadecimal format, needed only when the type is** `Pattern`.

Users can include OB and OTP memory regions in the export, in addition to the user-defined segments.

The `Clear` button allows users to quickly reset the export memory map data to its default state.

The output path for the HEX file can be specified manually or selected using a file browser dialog.

Note

When OB is selected, the export process may include multiple segments corresponding to the option bytes defined by the current device configuration. The number of these OB segments depends on the specific device and its configuration. Each OB segment is exported as a separate segment in the HEX file, ensuring that all relevant option byte data are included. The tool prevents data corruption and ensures that each memory segment occupies a unique, non-overlapping region in the exported file. If any overlap is detected, the export process is stopped, and the user is notified with an error message indicating which segments are in conflict.

3.1.2 MPU graphical user interface

Main window MPUs

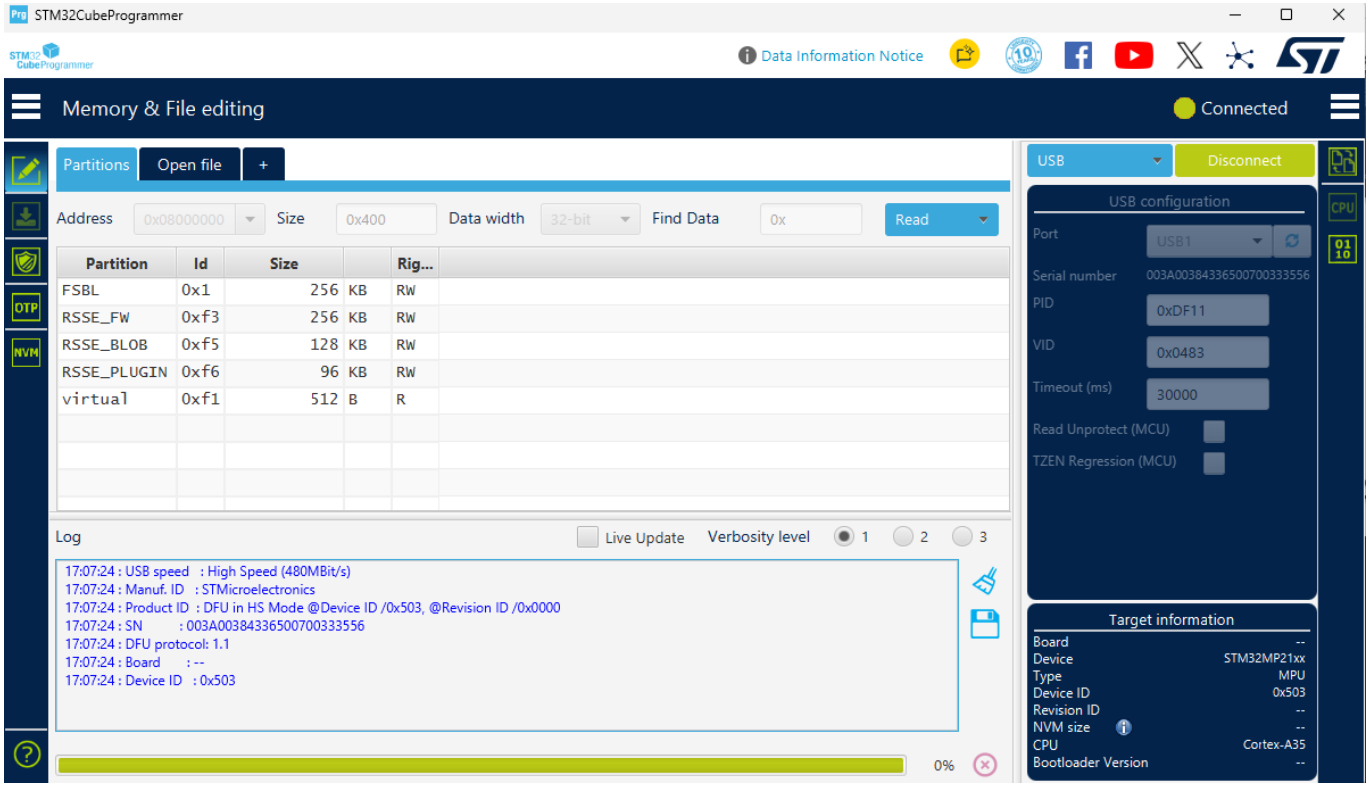


Fig. 3.83: STM32CubeProgrammer main window

The main window allows the user to select the interface used to connect to STM32MPxx BootROM. Possible interfaces are USB-DFU and UART, since programming through STLink interface is not possible with STM32MPxx devices.

Once connected using the `Connect` button, available partitions are displayed, and the user is able to open a TSV file for programming.

Programming windows

TSV programming window

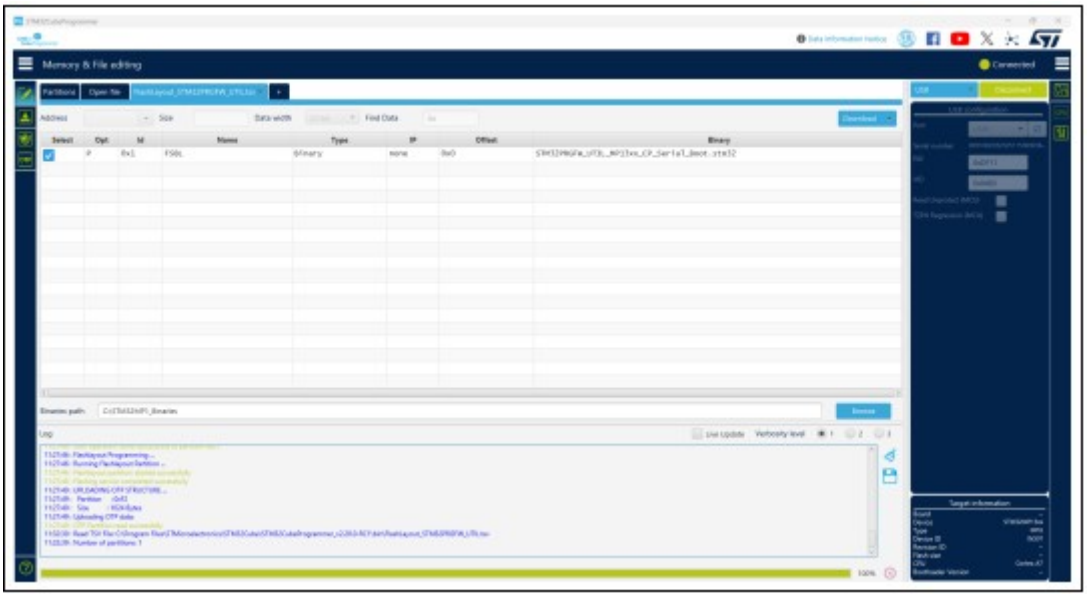


Fig. 3.84: TSV programming window

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

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

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

OTP programming window

The OTP window is available exclusively for MPUs and STM32N6 devices. It extracts the OTP partition `[ID 0xF2]` using the STMPRGFW-UTIL interface to read, display, and fuse the OTP registers.

STM32MP1xx devices have 3072 OTP (one-time programmable) bits, which can be read-accessed in 96 words: `OTPx` with `x = 0` to `95`. STM32MP2xx and STM32N6 devices have 368 words: `OTPx` with `x = 0` to `367`.

Some OTP words are programmed during manufacturing for product differentiation or keys.

> **Note**
>
> The OTP window is accessible through USB/UART interfaces for STM32MP. For STM32N6 it is available via Debug/USB/UART interfaces.

The UI has six main goals:

1. Get OTP structure information.
2. Read and display words in table view format.
3. Edit and fuse OTP registers.
4. Lock specific or all words.

5. Program binary files, frequently for key fuse operations.
6. Save the OTP partition in an output binary file for debug.

Open the OTP window by pressing the `OTP` button from the main window to start reading the OTP partition. When this is correctly executed, the words are displayed in table view.

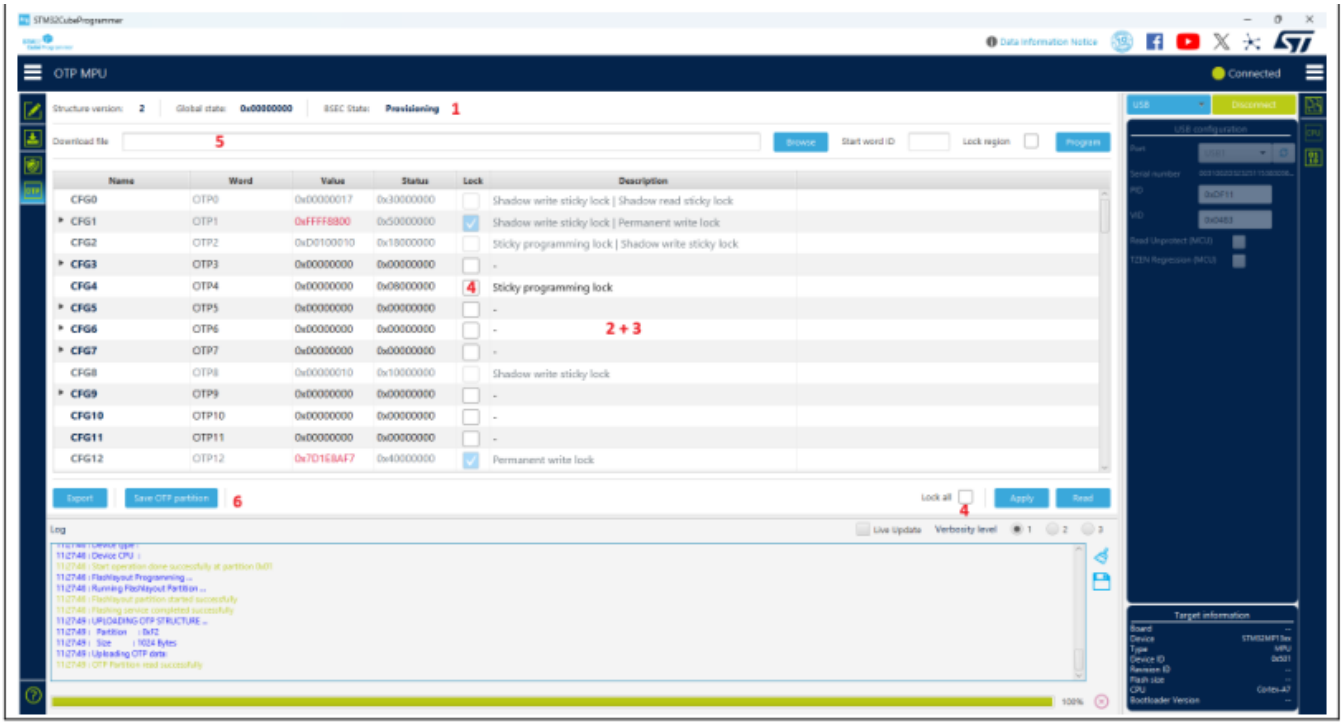


Fig. 3.85: OTP MPU window

Get OTP structure information

This section displays an overview of the connected device.

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

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

Read and display words

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

- Name: OTP word name.
- Word: OTP word ID as a tree component.
- Value: hexadecimal value, shown in red for locked words, as editable field.
- Status: hexadecimal value.
- Lock: indicates the lock state of the considered word as a checkbox component. It is checked if the item is locked, otherwise it is unchecked. The column is disabled if the item is in permanent write lock state.

- Description: decodes the status of the OTP word and displays a brief description for children items.

Note

Press the `Read` button to refresh all table fields or discard changes.

Edit and fuse words

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

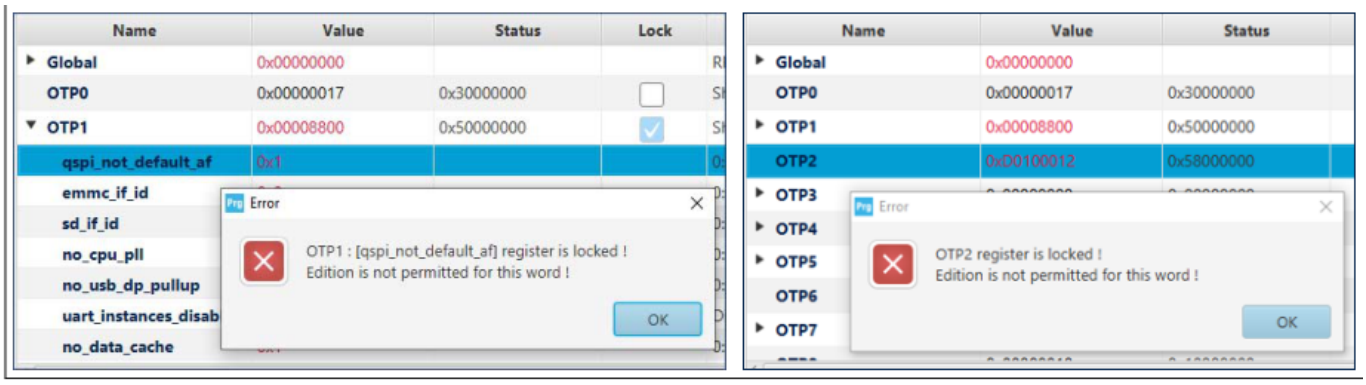


Fig. 3.86: Edit denial for locked words

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

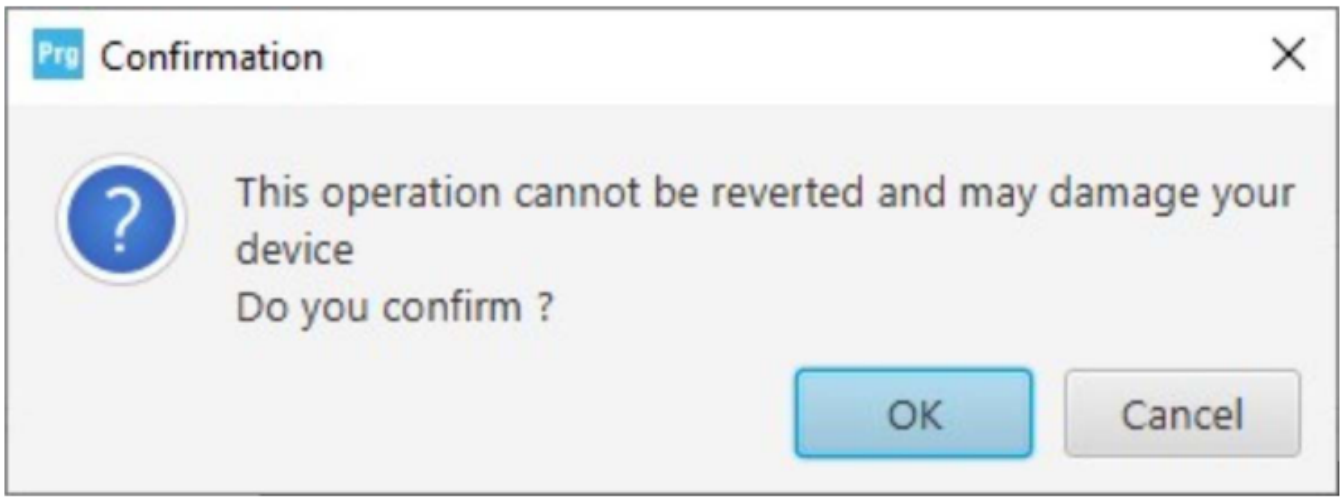


Fig. 3.87: Program Apply confirmation

Lock specific/all words

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

Note

To go faster to the initial OTP lock state, it is recommended to press the `Read` button.

It is possible to lock all words at once by setting the `Lock all` checkbox. A message pops up asking to confirm the start of the procedure, which results in all words being closed and no further changes allowed.

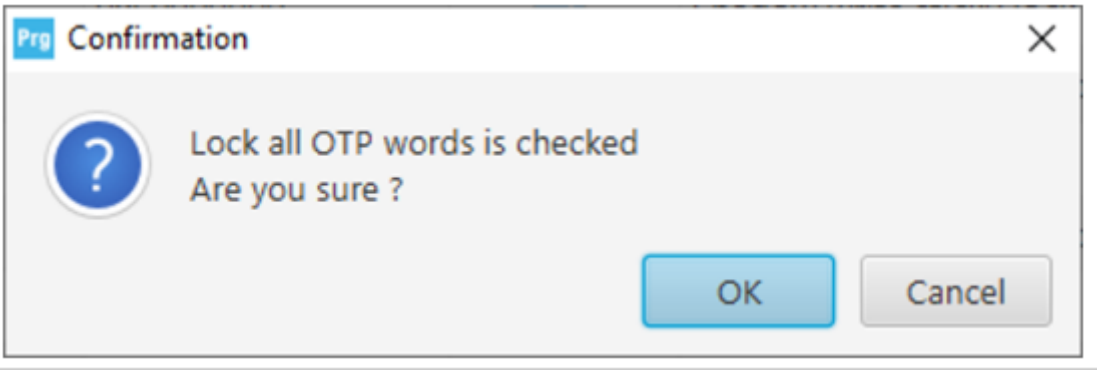


Fig. 3.88: Lock all words confirmation

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

Program binary file

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

1. **Choose the adequate binary file with `.bin` extension by clicking** `Browse`.
2. Mention the start word ID in decimal format from 0 to 95.
3. **Check or uncheck the `Lock region` checkbox to indicate the operation** type, update or permanent write lock.
4. Press the `Program` button to start the download flow.

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

Note

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

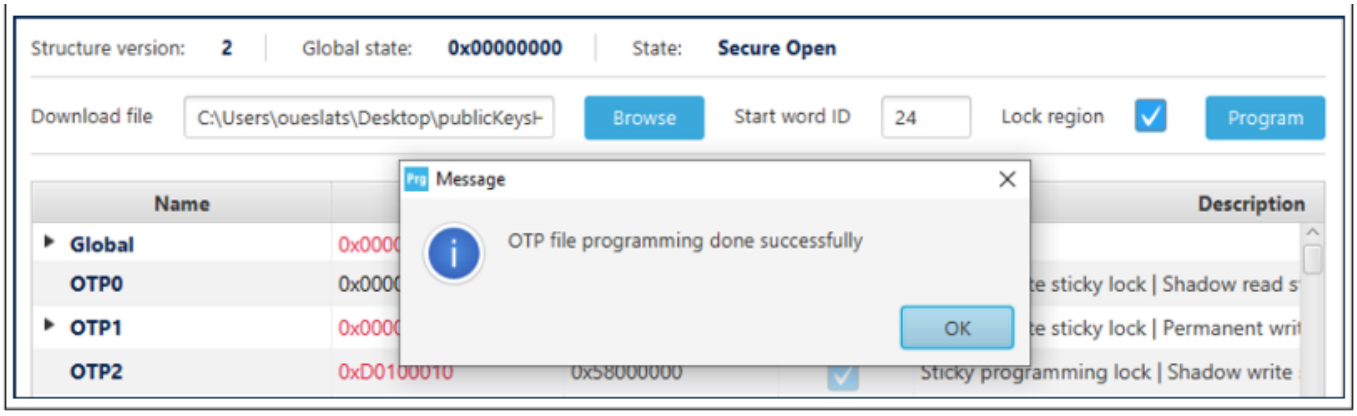


Fig. 3.89: Download binary file

Save OTP partition

The user can save the whole current OTP partition in a binary file with `.bin` extension, which can be used for future analysis or to archive the current device configuration.

Press `Save OTP partition` and choose the desired output name and directory. If the save is completed correctly, an informational dialog appears to confirm that the operation is completed.



Fig. 3.90: Save OTP partition

Note

The size of the output file must be 1024 bytes. U-Boot program must be installed before launching the OTP window, which is necessary to expose the OTP partition. Words 32 to 95 do not have child fields, can be edited only once, and must be permanently locked after programming. Word editing and lock operation can be performed at the same time after clicking `Apply`.

PMIC NVM programming

The STM32CubeProgrammer PMIC NVM window is available exclusively for MPU devices. It extracts the PMIC partition `[ID 0xF4]` using the PRGFW-UTIL interface to read, display, and program the PMIC NVM registers. `STPMIC1` has 8 registers and `STPMIC25` has 40 registers, each register with a 1-byte size.

PRGFW-UTIL can be downloaded from ST GitHub: STMicroelectronics/STM32PRGFW-UTIL.

The UI has five main features:

1. Get the PMIC information.
2. Read and display PMIC registers.
3. Edit and program PMIC registers.
4. **Program a binary file, only for the entire PMIC registers, since partial** register programming is not supported.
5. Save or export the PMIC registers, NVM partition, in an output binary file.

To begin reading the PMIC partition, open the PMIC NVM window by pressing the `NVM` button located in the main window. Once the partition is successfully read, data are displayed in a table format.

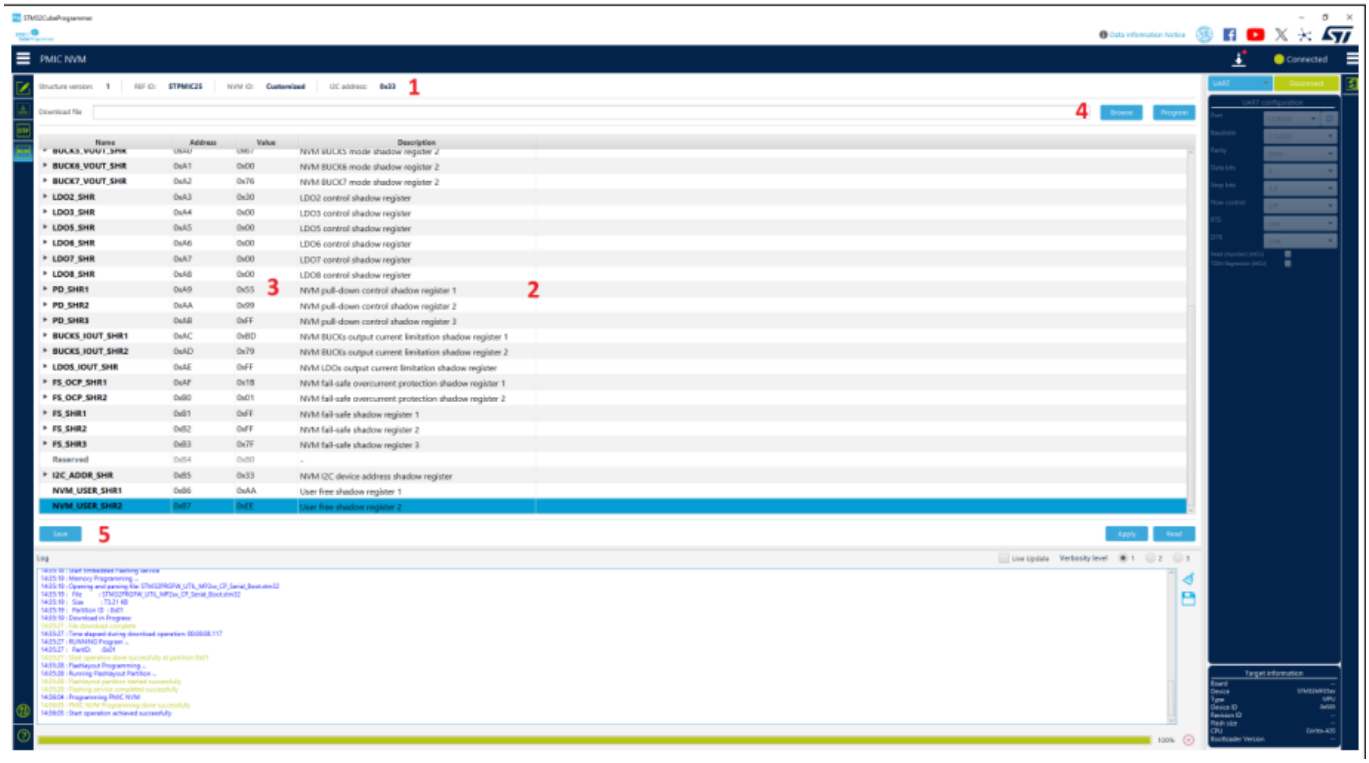


Fig. 3.91: PMIC NVM window

Get PMIC NVM structure information

This section displays an overview of the device.

- Structure version: version 1 so far.
- Ref ID: `STPMIC1`, `STPMIC25`, `STPMIC2L`, or `STPMIC1L`.
- NVM ID: Customized, A, B, or C.
- I2C address: the address that PRGFW-UTIL uses to communicate with PMIC.

Read and display words

After checking the PMIC partition, the tool decodes and shows the words in table view format including four columns:

- Name: NVM register name.
- Address: hexadecimal value representing the address in the NVM.
- Value: hexadecimal value, editable.
- Description: displays the description of the register and its fields.

> **Note**
>
> Press the `Read` button to refresh all table fields or discard changes.

The value and name cells expose a tooltip message to add description, name, hex value, and binary value representation.

Edit and program registers

There are two ways to edit the value of the register:

1. Use the combo box with a list of possible values.
2. **Directly write the register value in the text field and press `Enter`** after each change.

The modified element is highlighted in red.

After modification, it is mandatory to press the `Apply` button and confirm the operation to start the update and refresh the table view.

Program binary file

To program a binary file into the PMIC NVM register, it is necessary to take care about the following points:

1. Select the adequate binary having a `.bin` extension.
2. **The file size must have the exact size of the PMIC NVM registers partition,** for example STPMIC1 8 Bytes and STPMIC25 40 Bytes.
3. Using an incorrect file may risk damaging the board.
4. Press the `Program` button to start the download sequence.

Note

If the binary file contains the same content as the NVM registers, the tool takes no action.

Save/Export PMIC NVM partition

This capability allows saving the whole content of the PMIC NVM partition into a binary file with `.bin` extension or into an output text file in human-readable format, which can be used for future analysis or to archive the current device configuration.

Press `Save` or `Export` and choose the desired output file name and directory. If the save operation is completed successfully, an informational dialog appears to confirm the completion of the process.

Note

PRGFW-UTIL firmware must be installed before launching the PMIC NVM window, which is necessary to expose the PMIC partition. It is mandatory to use an adequate version of PRGFW-UTIL that supports the new PMIC structure format, version 1 or later. Otherwise the tool shows an error message.

3.2 Command lines

3.2.1 Command-line usage

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

To launch the command-line interface, use the corresponding command for your platform:

- **macOS x86_64**:

```
STM32CubeProgrammer.app/Contents/MacOs/bin/STM32_Programmer_CLI
```

- **macOS aarch64**:

```
STM32CubeProgrammer.app/Contents/Resources/bin/STM32_Programmer_CLI
```

- **Windows**:

```
..\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_Programmer_CLI.exe
```

- **Linux**:

```
../STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin/STM32_Programmer_CLI
```

MCU command lines

Generic commands

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

Long Command	Short Command	Description
–connect	-c	Establishes the connection to the device. This command allows the host to open the selected device port (UART/USB/JTAG/SWD/J-LINK/SPI/CAN/I2C).

Long Command	Short Command	Description
–erase	-e	According to the given arguments, this command can be used to erase specific sectors or the whole flash memory. This operation can take a second or more to complete, depending on the involved size.
–download,–write	-d, -w	Downloads the content of the specified binary file into the memory of the device. The download operation is preceded by the erase operation. A write address is needed to download binary files.
–verify,–verify fast	-v, -v fast	Used for validation of data integrity with two mechanisms: legacy uses a bit-by-bit comparison, while fast uses a sector-by-sector comparison. If the device does not support checksums, the legacy mechanism is used instead.
-w32	-w32	Downloads the specified 32-bit data into flash memory starting from a specified address.
–read, –upload	-r, -u	Reads and uploads the device memory content into a specified binary file starting from a specified address.
–go, –start	-g, -s	This command enables execution of the device memory starting from the specified address.
-rst	-rst	Executes a software system reset.
-hardRst	-hardRst	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.
–power	-pwr	Controls the target board power through an ST-LINK probe that supports this capability. The command can be used even without the -c connect command.
-halt	-halt	Halts the core.
-step	-step	Executes one instruction.
-score	-score	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”
-coreReg	-coreReg	Read/write Cortex-M core registers. The core is halted before a read/write operation.
–list	-l	This command lists all available UART, DFU and STLink interfaces.
-getTargetSelList	-getTargetSelList	Lists all the devices currently connected via ST-LINK on the same SWD bus supporting the SWD multidrop.
–quietMode	-q	This command disables the progress bar display during download and read commands
-rstbl	-rstbl	Enables a reset function compatible with all bootloader interfaces. The connection must be established while performing the operation.
–verbosity	-vb	This command makes it possible to display more messages, to be more verbose.
-log	–log	This traceability command makes it possible to store the whole traffic (with maximum verbosity level) into a log file.
-el	-el	This command allows entering the path of one or more external memory loaders to perform programming, write, erase, and read operations with an external memory.
-elbl	-elbl	With this command, the user can provide the path of an external memory loader used to perform programming, write, erase, and read operations using bootloader interfaces.
–readunprotect	-rdu	This command removes the memory read protection by changing the RDP level from level 1 to level 0.

Long Command	Short Command	Description
–tzenregression	-tzenreg	This command removes TrustZone protection by disabling TZEN from 1 to 0.
–optionbytes	-ob	This command allows the user to manipulate the device option bytes by displaying or modifying them.
–safelib	-sl	This command allows a firmware file to be modified by adding a load area (segment) containing the computed CRC values of the user program.
–sfi	-sfi	Programs an sfi file.
–ssp	-ssp	Programs an SSP file. For MCUs, this command is supported only on STM32N6.
–rsse	-rsse	This command allows the user to select the root secure services extension library (RSSe).
–abort	-a	This command allows the user to clean a not properly finished process. The currently ongoing operation stops and the system returns to idle state.
–mcsv	-mcsv	This command allows the user to select the modules file including the list of modules to be installed with SFI. This option is mandatory when the SFI image integrates at least one module (area of type m).
–sfi	-sfi	Programs an sfix file
–rsse	-rsse	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.
–abort	-a	This command allows the user to clean a not properly finished process. The ongoing operation stops and the system returns to idle state.
–mcsv	-mcsv	This command allows the user to select the modules file including the list of modules to be installed with SFI. This option is mandatory when the SFI image integrates at least one module (area of type m).
–ecsv	-ecsv	This command is used for Secure Manager Secure Modules install in external flash. It allows the user to select the modules file including the list of external modules to be installed with SFIx. This option is mandatory when the SFI image integrates at least one external module (area of type e).
–hsmgetinfo	-hsmgetinfo	Reads the HSM available information.
-hsmgetcounter	-hsmgetcounter	Reads the current value of the license counter.
-hsmgetfwid	-hsmgetfwid	Reads the Firmware/Module identifier.
-hsmgetstatus	-hsmgetstatus	Reads the current card life-cycle state.
-hsmgetlicense	-hsmgetlicense	Gets a license for the current chip if counter is not null.
-hsmgetlicensefromcertifbin	-hsmglfcb	Gets a license for the current certificate binary file if counter is not null.
-antirollback	-antirollback	Perform the antirollback operation.
-startfus	-startfus	Start the FUS.
-getuid64	-getuid64	Read the device unique identifier (UID).
-fusgetstate	-fusgetstate	Read the FUS state.
-fwdelete	-fwdelete	Delete the BLE stack firmware.
-fwupgrade	-fwupgrade	Upgrade of BLE stack firmware or FUS firmware.
-startwirelessstack	-startwirelessstack	Start the wireless stack.
-authkeyupdate	-authkeyupdate	Authentication key update.
-authkeylock	-authkeylock	Authentication key lock.
-wusrkey	-wusrkey	Write user key.

Long Command	Short Command	Description
-swv	-swv	Accesses serial wire viewer console mode.
-startswv	-startswv	Accesses serial wire viewer console mode.
-dsecurity	-dsecurity	Disables security on STM32WL.
-setdefaultob	-setdefaultob	Configures option bytes to their default values on STM32WL.
-ob unlockchip	-ob unlockchip	Unlocks STM32WL when bad option bytes are programmed.
-ssigfoxc	-ssigfoxc	Saves the SigFox chip certificate to a binary file.
-wsigfoxc	-wsigfoxc	Writes the SigFox chip certificate.
-regdump	-regdump	Reads and dumps core and MCU registers.
-hf	-hf	Starts hard fault analysis.
–file-checksum	-fchecksum	Calculates the checksum value for the entire file content.
–checksum	-checksum	Calculates the checksum value for accessible memory.
-lockRDP1, -setOEM1KEY	-lockRDP1, -setOEM1KEY	Locks RDP regression from level 1 with a password.
-lockRDP2, -setOEM2KEY	-lockRDP2, -setOEM2KEY	Locks RDP regression from level 2 with a password.
-unlockRDP1	-unlockRDP1	Unlocks RDP regression from level 1 with a password.
-unlockRDP2	-unlockRDP2	Unlocks RDP regression from level 2 with a password.
-verifyrdp2	-verifyrdp2	Validates the provisioned OEM password before setting the RDP level 2.
-lockbs	-lockbs	Provisions the boundary-scan key.
-unlockbs	-unlockbs	Performs the boundary-scan RDP transition.
-gc	-gc	Reads the chip certificate.
-w32dbgmcu	-w32dbgmcu	Writes 32-bit data to the DBGMCU AUTH HOST register.
-sdp	-sdp	Programs OBKey content.
-pwd	-pwd	Provisions a password and generates a password.bin file.
debugauth	debugauth	Performs debug-auth discovery, authentication, or debug close, posssible value: 1|2|3.
–force_no_da	–force_no_da	Forces product-state OB programming without configured debug authentication.
-installipmodule	-installipmodule	Installs a Secure Manager IP module.
–updateipmodule	-updateipmodule	Updates a Secure Manager IP module.
–skiperase	–skiperase	Skips sector erase before programming.
-storekeyotp	-storekeyotp	Stores an authentication public key in OTP.
-rssekw	-rssekw	Selects RSSeKW and executes private-key wrapping.
-r32fast	-r32fast	Fast read 32-bit data memory.
–yes	-y	Confirms STM32V8 programming when the tool blocks the risky operation.
-blankcheck	-blankcheck	Verifies that the STM32 flash memory is blank. If this is not the case, the first address containing data is highlighted in a message.
-fillmemory	-fillmemory	This command allows the user to fill memory with a given pattern from the chosen address.

Command details

-connect

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

`port=<Portname>`: Interface identifier, for example `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 bit, value in {0, 1}, default is 0. `noinit=1` can be used if a previous connection

is active.

ST-LINK options

Option	Description
[freq=<frequency>]	Frequency (in kHz) used for the connection. Default value is 4000 kHz for SWD port, and 9000 kHz for JTAG port.
[reset=<mode>]	Reset mode. Possible values are {SWrst/HWrst/Crst}. The default value is `SWrst`. When using `UR` connection mode, the reset mode is `HWrst`.
[HWrst]	Hardware reset. Performed by the ST-LINK, which physically drives the NRST pin of the MCU. The NRST pin of the ST-LINK must be connected to the MCU.
[SWrst]	Software reset (a system reset via Cortex registers).
[Crst]	Core reset (via Cortex registers).
[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 for which you know the serial number. Do not use this option together with the `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 instruction. This is useful, for example, when the target contains code that disables the JTAG/SWD pins.
[HOTPLUG]	The *Hot Plug* mode enables connection to the target without a halt or reset. This is useful for updating the RAM addresses or the IP registers while the application is running.
[POWERDOWN]	Allows putting the target in debug mode, even if the application has not started since the target power-up. The hardware reset signal must be connected between ST-LINK and the target. This feature might not be fully effective on some boards (MB1360, MB1319, MB1361, MB1355) with the STMPS2141 power switch.
[hwRstPulse]	The tool generates a reset pulse and then connects to the target. This connection mode does not prevent application launch before connection. It is used on devices where under-reset mode is not available, such as STM32WB0x and STM32WL33.
[ap=<accessPort>]	Access port index. Default access port value is 0.
[speed=<speed_mode>]	Connection speed. Default is `Reliable`. Available only for Cortex-M33.
[Reliable]	Allows the user to connect in slow mode.
[Fast]	Allows the user to connect in fast mode.
[shared]	Enables shared mode, allowing connection of two or more instances of STM32CubeProgrammer or other debuggers to the same ST-LINK probe.
[tcpport=<Port>]	Selects the TCP port to connect to an ST-LINK server. The `shared` option must be selected. Default value is 7184.
[dLPM / LPM]	Disable/enable debug in low-power mode (default configuration is enabled for the supported devices (STM32U5/WB/L4 series)).
[getAuthID]	Get device identification (only for STM32U5 series). This is a 32-bit device-specific quantity that can be read through the JTAG port. This 32-bit information is used to derive the expected OEM password keys to unlock this specific device. This command is not applicable when RDP level = 0 (MCU constraint).

Note

Shared mode is supported only on Windows.

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

SPI options

Option	Description
[br=<baudrate>]	Baudrate (for example 187, 375, 750). Default is 375.
[cpha=<cpha_val>]	1Edge or 2Edge. Default is 1Edge.
[cpol=<cpol_val>]	Low or High. Default is Low.
[crc=<crc_val>]	Enable or disable (0/1). Default is 0.
[crcpol=<crc_pol>]	CRC polynomial value.
[datasize=<size>]	8- or 16-bit. Default is 8-bit.
[direction=<val>]	2LFullDuplex / 2LRxOnly / 1LRx / 1LTx.
[firstbit=<val>]	MSB or LSB. Default is MSB.
[frameformat=<val>]	Motorola or TI. Default is Motorola.
[mode=<val>]	Master or Slave. Default is Master.
[nss=<val>]	Soft or Hard. Default is Hard.
[nsspulse=<val>]	Pulse or NoPulse. Default is Pulse.
[delay=<val>]	Delay/NoDelay, default Delay

Note

To use SPI at high speed, the hardware infrastructure must comply with the requirements to ensure proper connection on the bus.

Direct GPIO bootloader programming on Raspberry Pi:

This mode programs an STM32 target from a Raspberry Pi or compatible ARM Linux host without ST-LINK V3. The STM32 target must already be in bootloader mode.

SPI direct GPIO command

SPI prerequisites:

- Enable SPI on the Linux host.
- Check nodes with: `ls /dev/spidev*`
- Connect NSS, SCK, MISO, MOSI, and GND.

```
STM32_Programmer_CLI -c port=spidevX.X -d "firmware.bin" "<start_address>"
```

Example:

```
STM32_Programmer_CLI -c port=spidev0.0 -d "application.bin" 0x08000000
```

J-Link options

Option	Description
[reset=<mode>]	Reset mode. Possible values are {SWrst/HWrst/Crst}. The default value is SWrst. When using UR connection mode, the reset mode is HWrst.
[HWrst]	Hardware reset. Performed by the STLink, which physically drives the NRST pin of the MCU. The NRST pin of the STLink must be connected to the MCU.
[SWrst]	Software reset (a system reset via Cortex registers)
[Crst]	Core reset (via Cortex registers)
[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.
[mode=<mode>]	Connection mode. Value in {NORMAL/UR/HOTPLUG}. Default value is NORMAL.
[Normal]	With “Normal” connection mode, the target is reset, then halted. The type of reset is selected using the “Reset Mode” option.
[UR]	The “Connect Under Reset” mode enables connection to the target using a reset vector catch before executing any instructions. This is useful in many cases, for example when the target contains a code that disables the JTAG/SWD pins.
HOTPLUG	The “Hot Plug” mode enables connection to the target without a halt or reset. This is useful for updating the RAM addresses or the IP registers while the application is running.
[ap=<accessPort>]	Access port index. Default access port value is 0.
[speed=]	Connection speed. Default is Reliable. Available only for Cortex-M33.
Reliable	Allows the user to connect with a slow mode.
Fast	Allows the user to connect with a fast mode.

I2C options

Option	Description
[add=<ownadd>]	Slave address: address in hex format
[br=<sbaudrate>]	Baudrate: 100 or 400 kbps, default 400 kbps.
[sm=<smode>]	Speed Mode, STANDARD or FAST, default FAST.
[am=<addmode>]	Address Mode: 7 or 10 bits, default 7.
[af=<afilter>]	Analog filter: ENABLE or DISABLE, default ENABLE.
[df=<dfilter>]	Digital filter: ENABLE or DISABLE, default DISABLE.
[dnf=<dnfilter>]	Digital noise filter: 0 to 15, default 0.
[rt=<rtime>]	Rise time: 0-1000 (STANDARD), 0-300 (FAST), default 0.
[ft=<ftime>]	Fall time: 0-300 (STANDARD), 0-300 (FAST), default 0.

Note

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

CAN options

Option	Description
[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.

Option	Description
[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 the following figure.

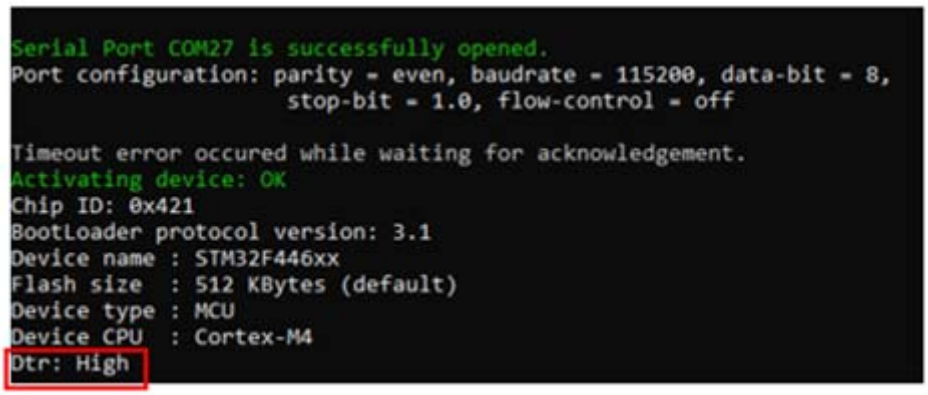


Fig. 3.92: Connect operation using RS232

STM32CubeProgrammer provides the possibility to configure RTS and DTR pins:

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

Example:

```
STM32_Programmer_CLI.exe -c port=COM27 dtr=high
```

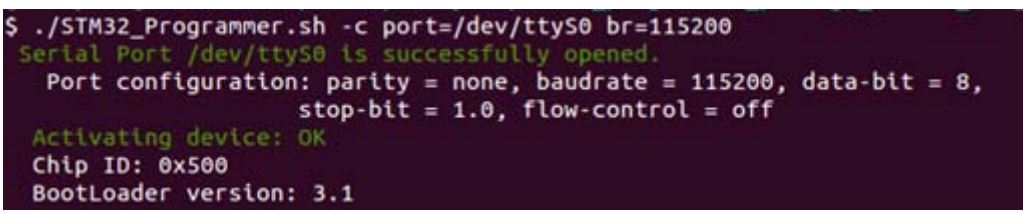


Fig. 3.93: Enabling COM DTR pin

Example using USB

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

The result of this example is shown in the following figure.

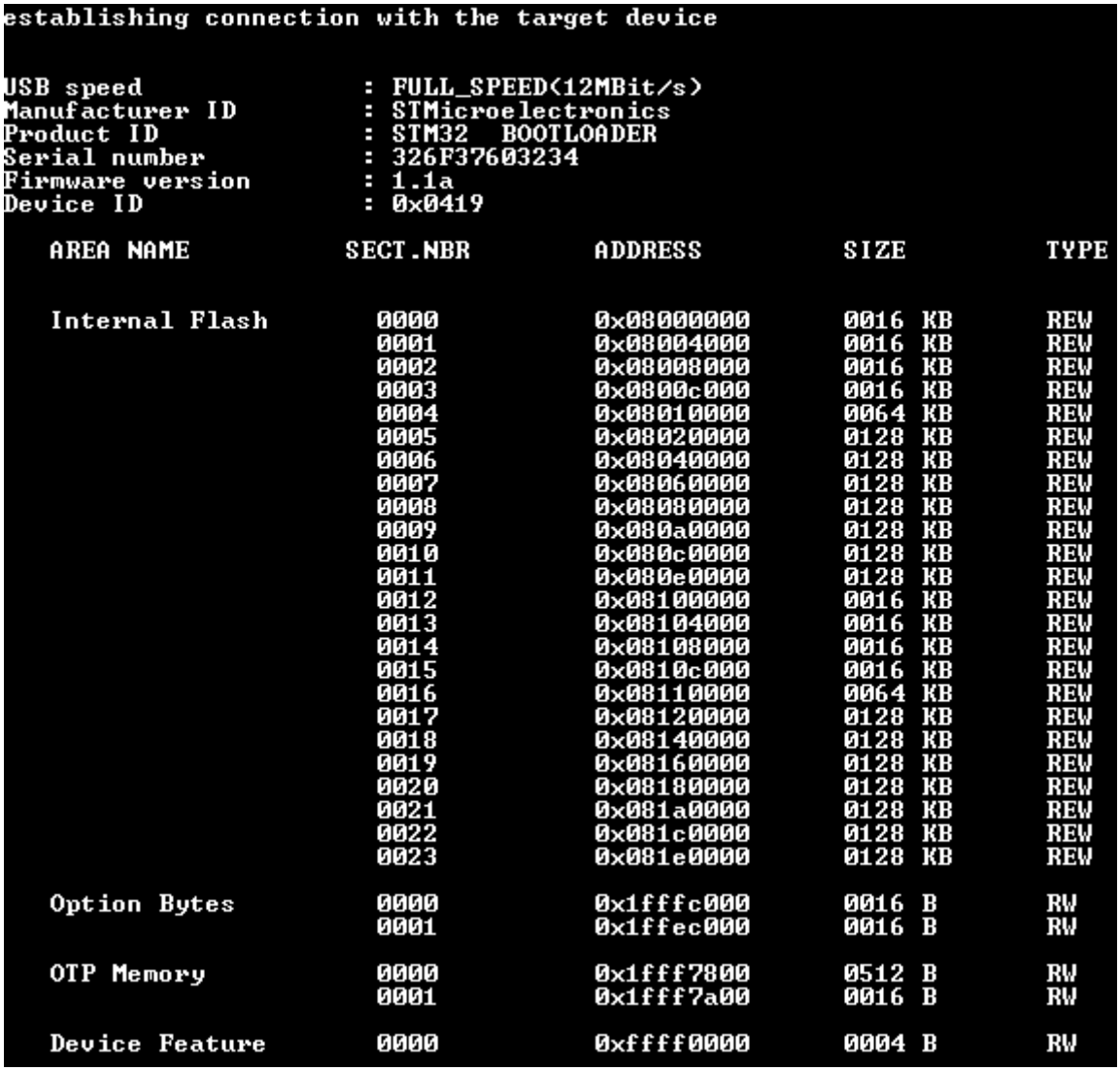


Fig. 3.94: Connect operation using USB

Note

When using a USB interface, all the configuration parameters such as baudrate, parity, data-bits, frequency, and index are ignored. To connect using a UART interface, the port configuration baudrate, parity, data-bits, stop bits, and flow-control must have a valid combination depending on the used device.

Example using DFU IAP/USBx options

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

The result of this example is shown in the following figure.

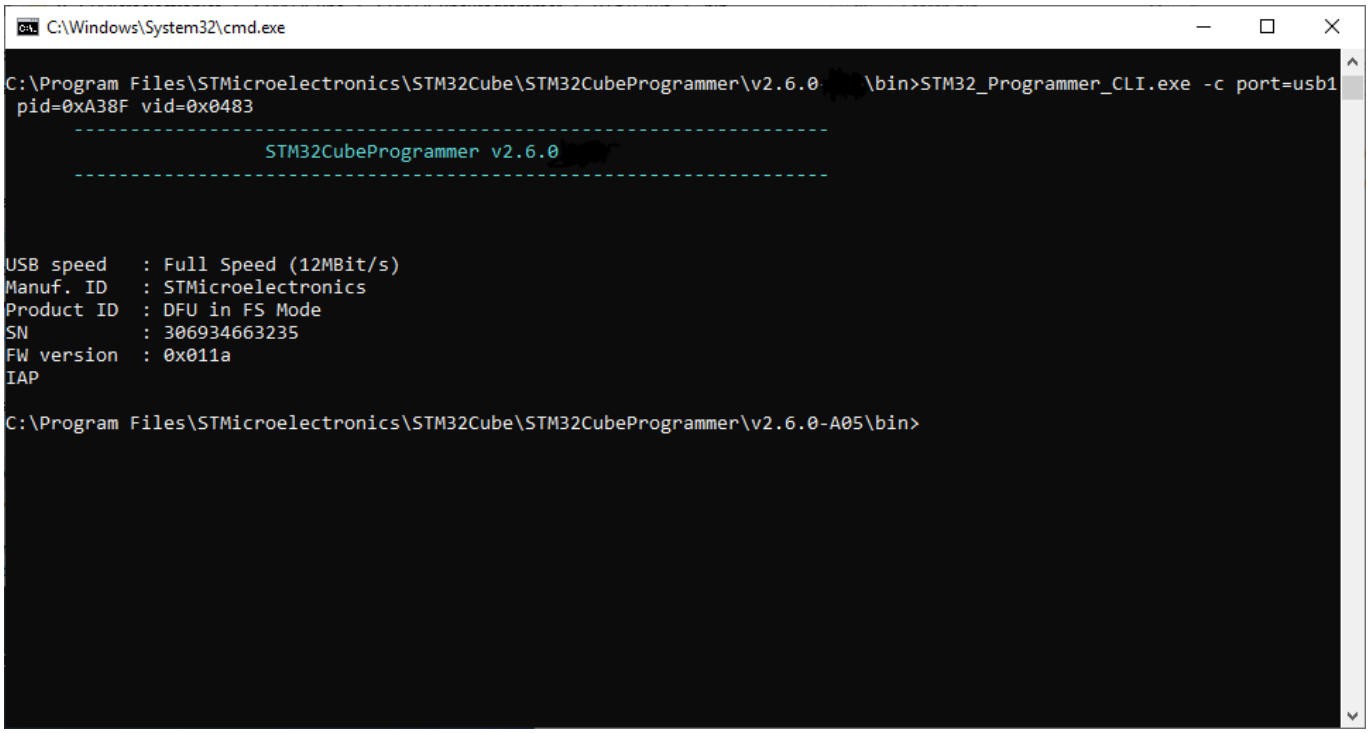


Fig. 3.95: Connect operation using USB DFU options

Note

The default product ID and vendor ID values are ST product values: PID = 0xDF11 and VID = 0x0483.

Example using JTAG/SWD debug port

To connect using port connection mode with an ST-LINK probe, it is necessary to mention the port name with at least the connect command, for example:

```
-c port=JTAG
```

Note

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

There are other parameters used in connection with JTAG/SWD debug ports that have default values. See the tool help for more information about those default values.

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

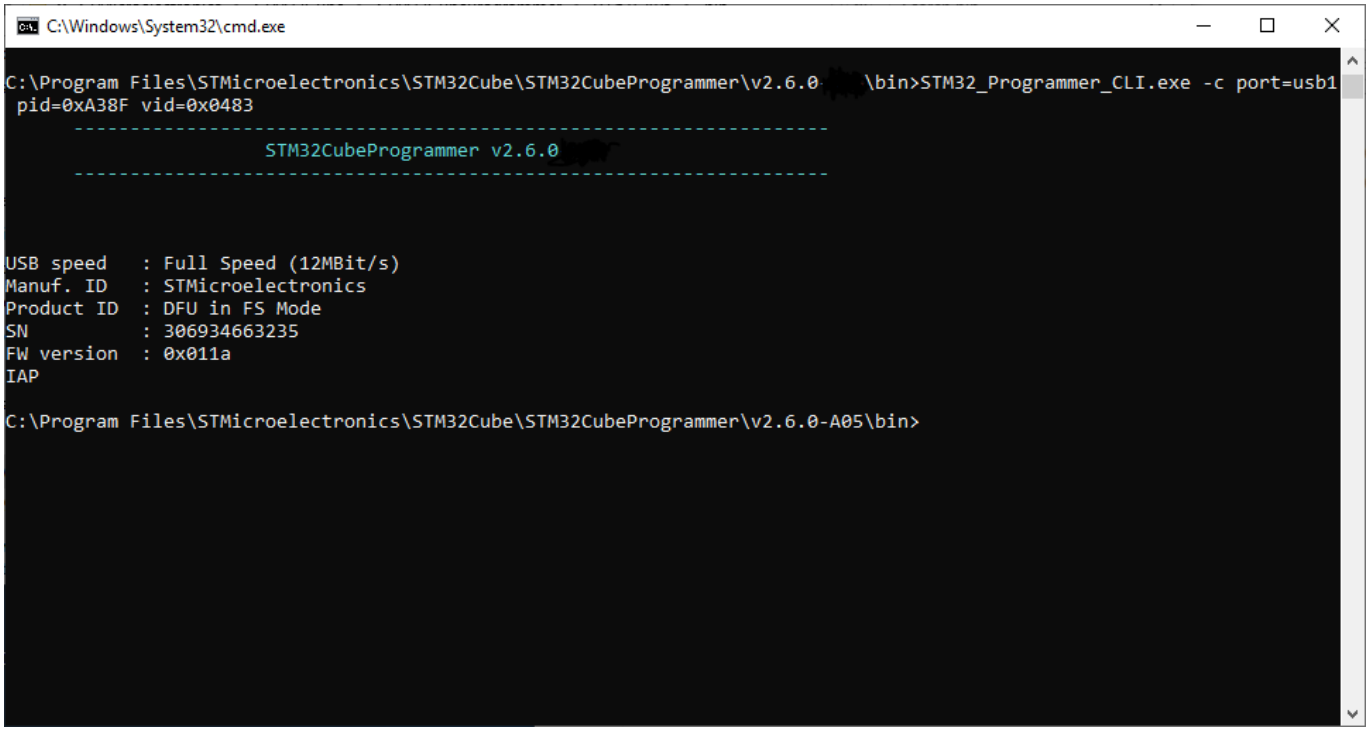


Fig. 3.96: Connect operation using SWD debug port

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 at 4000 kHz. This is because the 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, and 5 kHz
  - JTAG: 9000, 4500, 2250, 1125, 562, 281, and 140 kHz
- ST-LINK V3:
  - SWD: 24000, 8000, 3300, 1000, 200, 50, and 5 kHz
  - JTAG: 21333, 16000, 12000, 8000, 1777, and 750 kHz

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

- SWD: ST-LINK V2: 4000 kHz, ST-LINK V3: 24000 kHz
- JTAG: ST-LINK V2: 9000 kHz, ST-LINK V3: 21333 kHz

Note

JTAG frequency selection is supported only 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. 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 J-Link debug port

To connect using port connection mode with a J-Link probe, it is necessary to mention the port name with at least the connect command, for example:

```
-c port=JLINK
```

Only Flash and Option Bytes (OB) programming are supported via J-Link. Other features are not supported.

Note

There are other parameters used in connection with the J-Link debug port that have default values. See the tool help for more information about those default values.

The example below shows a connection example with an STM32 with device ID 0x480.

```
Connecting to J-Link Probe
Device=Cortex-M7
Device ID   : 0x480
Voltage     : 3.32V
Flash size  : 2 MBytes
```

Fig. 3.97: Connect operation using J-Link debug port

The corresponding command line for this example is `-c port=JLINK ap=0`.

In the connect command (`-c port=JLINK ap=0`):

- The `port` parameter is mandatory.
- The default frequency value is 4000.

Note

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.

Example using SPI

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

The result of this example is shown in the following figure.



Fig. 3.98: Connect operation using SPI port

Note

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

There are other parameters used in connection with the SPI port that have default values, and some others must have specific values. See the tool help 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 the following figure.



Fig. 3.99: Connect operation using CAN port

Note

Not all devices implement this feature. Make sure the device you are using supports a CAN bootloader.

There are other parameters used in connection with the CAN port that have default values, and some others must have specific values. See the tool help for more information.

Example using I2C

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

In the connect command:

- The `add` parameter changes from one device to another. Refer to AN2606 to determine the correct value. In this case, the STM32F42xxx has a bootloader address equal to `0x38`.
- The baudrate parameter `br` 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 the following figure.



Fig. 3.100: Connect operation using I2C port

Note

For each I2C connection operation, the address parameter is mandatory. Not all devices implement this feature. Make sure that the device supports an I2C bootloader.

There are other parameters used in connection with the I2C port that have default values, and some others must have specific values. See the tool help for more information.

Note

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

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

I2C direct GPIO command

I2C prerequisites:

- Enable I2C on the Linux host.
- List buses with: `i2cdetect -l`
- Detect address with: `i2cdetect -y <bus_number>`
- Connect SDA, SCL, and GND.

```
STM32_Programmer_CLI -c port=i2c-X add=0xYY -d "firmware.bin" "<start_address>"
```

Example:

```
STM32_Programmer_CLI -c port=i2c-1 add=0x48 -d "application.bin" 0x08000000
```

Warnings and notes

- Follow AN2606 to enter STM32 bootloader mode.
- SPI and I2C pins depend on the selected STM32 device.
- Linux users may need permission to access `/dev/spidev*` and `/dev/i2c-*`.

-erase

Syntax:

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

Note

In the case of multiplicity of external loaders, the first selected loader is the one taken into account during erasing of the external memory. In macOS, erasing a range of sectors is not possible with the zsh terminal interpreter. The user must switch to bash to get this command working.

For STM32V8, erasing sectors in the High-Cycle region is not supported via the command line.

-write

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 is shown in the following figure.

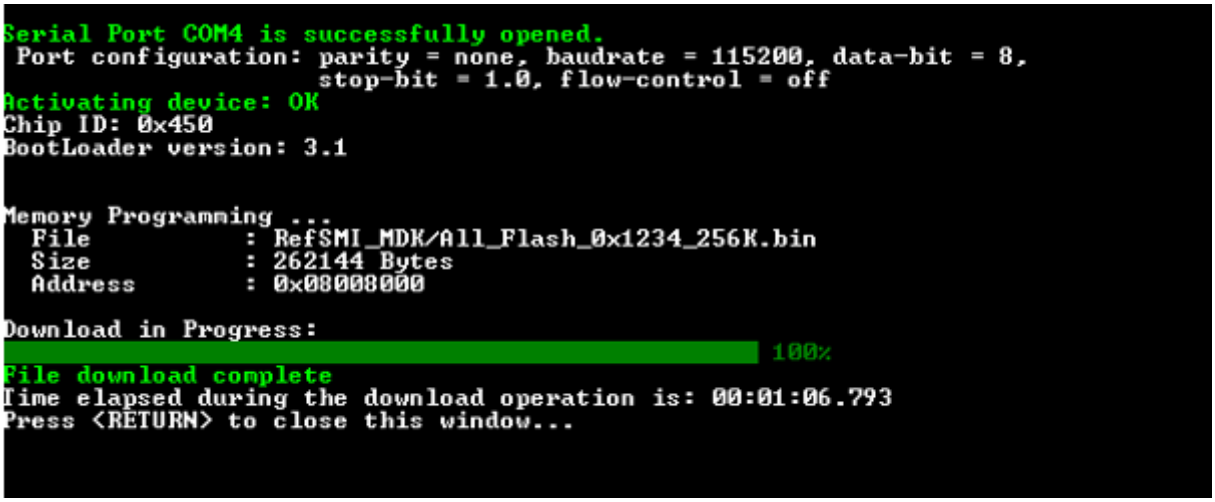


Fig. 3.101: Download operation

Note

To verify that the download has been successful, call the verify option (`-v` or `--verify`) just after the write command. Otherwise, the verify option is ignored. STM32CubeProgrammer can write on aligned memory regions. Flash memory imposes a data alignment described in reference manuals. As an example, for STM32U5 devices, the reference manual indicates that this MCU supports `137 bits` wide data read and write (128 effective bits plus 9 ECC bits), which means that data must be aligned on 16 bytes.

Mechanisms for programming command

The programming command supports two mechanisms, legacy and incremental, each with its own approach.

In the legacy mechanism, STM32CubeProgrammer reprograms all the flash memory sectors. This process involves erasing and reprogramming the whole memory.

[file_path]
Path of the file to be downloaded.

[start_address]
Start address of download.

The incremental mechanism programs only the modified sectors. This process involves calculating checksums for each sector of the new firmware and the flash memory, comparing them to identify modified sectors, and then selectively erasing and programming only those sectors. This approach significantly reduces the programming time.

Example:

```
-c port=SWD -w 1MG.bin 0x08000000 incremental
```

This command programs the binary file `1MG.bin` at address `0x08000000` using the incremental mechanism. If the new firmware already exists on the flash memory, no flashing is done.

-verify

Description

Used for validation of data integrity. It has two mechanisms:

- Legacy mechanism: uses a bit-by-bit comparison to verify data integrity.
- Fast mechanism: uses a sector-by-sector comparison. If the device does not support checksums, the legacy mechanism is used instead.

-download

`-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 space.

Example:

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

This command makes it possible to write the 32 data bits `0x12345678`, `0xAABBCCFF`, and `0x12AB34CD` into the flash memory starting from address `0x08000000`.

> **Note**
>
> STM32CubeProgrammer can write on aligned memory regions. Flash memory imposes data alignment described in reference manuals. As an example, STM32U5 devices support `137 bits` wide data read and write (128 effective bits plus 9 ECC bits), which means that data must be aligned on 16 bytes. To write into the flash memory, it must already be erased. The `-w32` command does not erase the flash memory.

-read

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>`

Example:

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

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

@0x08000000 :  0x20000600  0x08006BA9  0x08005ADD  0x08005ADD
@0x08000010 :  0x08005AAA  0x08005ADD  0x08005ADD  0x00000000
@0x08000020 :  0x00000000  0x00000000  0x00000000  0x08005ADD
@0x08000030 :  0x08005ADD  0x00000000  0x08005AEB  0x080066E3
@0x08000040 :  0x08005B0D  0x08005B0D  0x08005B0D  0x08005AF9
@0x08000050 :  0x08005B0D  0x08005B0D  0x08005AF9  0x08005AF9
@0x08000060 :  0x08005AF9  0x08005AF9  0x08005AF9  0x08003AB9
@0x08000070 :  0x08003ACB  0x08003ADD  0x08003AF1  0x08003B05
@0x08000080 :  0x08003B19  0x08003B2D  0x08005B0D  0x08005B0D
@0x08000090 :  0x08005B0D  0x08005B0D  0x08005BBB  0x08005ABB
@0x080000A0 :  0x08005AF9  0x08004689  0x08005AF9  0x08005B0D
@0x080000B0 :  0x08005AF9  0x08005AF9  0x0800469F  0x08005B0D
@0x080000C0 :  0x08005B0D  0x08005B0D  0x08005B0D  0x08005B0D
@0x080000D0 :  0x08005B0D  0x080040AB  0x08005AF9  0x08005AF9
@0x080000E0 :  0x08005AF9  0x08005B0D  0x08005B0D  0x08005AF9
@0x080000F0 :  0x08005AF9  0x08005AF9  0x08005B0D  0x08005B0D
```

Fig. 3.102: Read 32-bit operation

Note

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

-start

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.

Debug commands

-rst

-rst

Syntax: -rst

-hardRst

Syntax: -hardRst

–power

Description: Controls the target board power through an ST-LINK probe that supports this capability.

The command can be used even without the -c connect command.

Syntax: `-pwr, --power <on/off> [index=<number>]`

Parameters

- `<on/off>`: Selects the power state to apply to the target board.
- `index=<number>`: Index of the debug probe. Optional when only one probe is connected.

Notes

- Available only on ST-LINK probes supporting power control.
- If several probes are connected, specify `index=<number>`.

Examples

Using `--power` with `-c`

```
STM32_Programmer_CLI.exe -c port=swd --power off index=0 --power on index=0
```

Using `--power` without `-c`

```
STM32_Programmer_CLI.exe --power off index=0 --power on index=0
```

-halt

Syntax: -halt

-step

Syntax: -step

-score

Syntax: -score

-coreReg

Syntax: `-coreReg [<core_register>]`

Supported core registers:

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`.

Note

The following commands are available only with the JTAG/SWD debug port: -rst, -halt, -step, -score, -coreReg.

-list

Syntax: `-l, --list`

Example:

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

The result of this example is shown in the following figure.

```
=====  DFU Interface   =====

No STM32 device in DFU mode connected

===== STLink Interface =====

-------- Connected ST-LINK Probes List --------

ST-Link Probe 0 :
   ST-LINK SN  : 002200144741500220383733
   ST-LINK FW  : V3J8M3
   Access Port Number  : 2
-----------------------------------------------

=====  UART Interface  =====

Total number of serial ports available: 2

Port: COM47
Location: \\.\COM47
Description: STMicroelectronics STLink Virtual COM Port
Manufacturer: STMicroelectronics

Port: COM3
Location: \\.\COM3
Description: Intel(R) Active Management Technology - SOL
Manufacturer: Intel
```

Fig. 3.103: List of available serial ports

-getTargetSelList

SWD Multidrop

SWD (Serial Wire Debug) Multidrop is an enhancement to the standard SWD protocol, commonly used to debug microcontrollers based on Cortex-M cores. This feature allows a single SWD interface, from a debugger or programmer, to communicate with multiple target devices connected on the same SWD bus.

Only flash and option bytes (OB) programming are supported.

Syntax: `-getTargetSelList`

The result of this command is shown in the following figure.

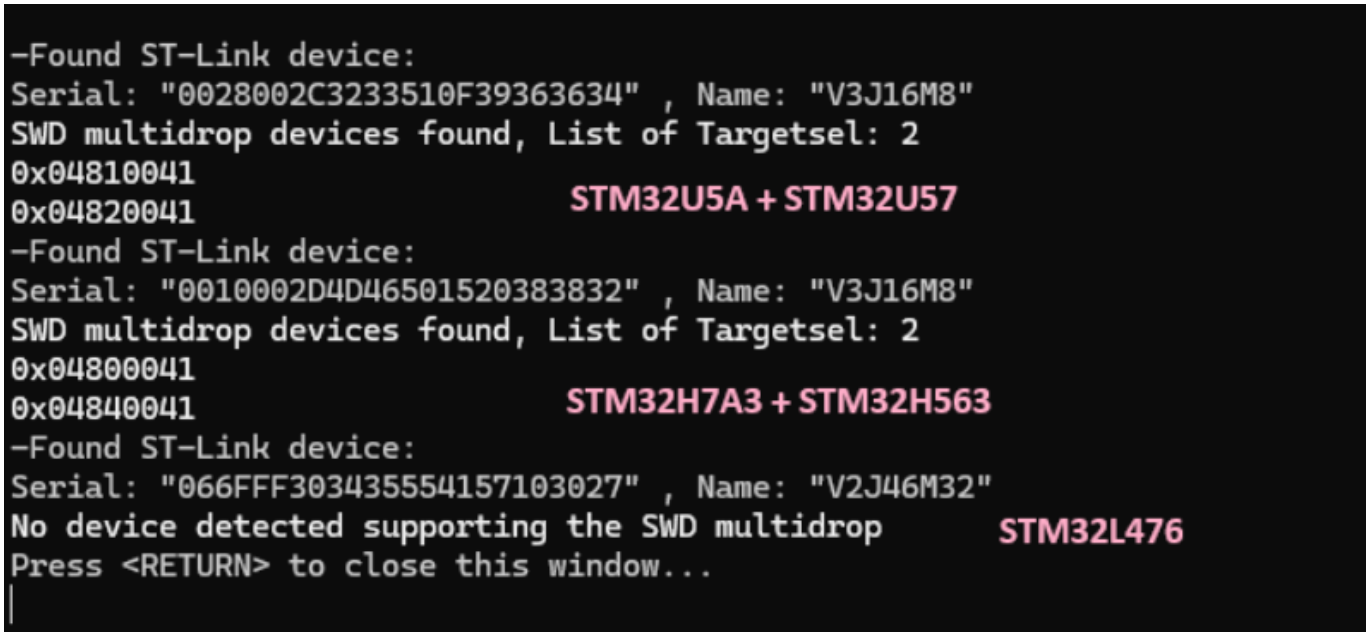


Fig. 3.104: List of connected devices via different ST-LINK

Note

If no board is detected, the command returns `No STM32 Target Found!`

If the connected boards use an outdated ST-LINK firmware, the command returns `Too Old Firmware, please upgrade your STLINK!`

Once the ST-LINK firmware is updated, all connected devices supporting the SWD Multidrop are correctly listed.

Limitation: only devices with different `TargetSel` can be connected simultaneously to the same ST-LINK.

New option in Connect command: `TargetSel`

Description: Specifies the target device by its `TargetSel` value.

Syntax: `TargetSel=<TargetSelValue>`

Example:

```
./STM32_Programmer_CLI -c port=swd sn="002F00373038510934333935" TargetSel=0x04540041 -d "C:\
↪Users\path.bin" 0x08000000
```

Note

If multiple boards are connected via the ST-LINK bus, you can perform read, write, and download operations on each board by selecting the appropriate `TargetSel`.

Switching between devices can be done without hardware intervention only by selecting the `TargetSel` value.

If multiple ST-LINKs are connected, you must specify the serial number (`sn`) in addition to `TargetSel` in the command line.

It is not possible to insert several targets that cannot be selected only by the SWD multi-drop selection sequence. This means that all devices on the lines must have different `TargetSel` values.

For more details, refer to AN6382, `Introduction to SWD multi-drop for STM32 with ST-LINK`, available on www.st.com.

-quietMode

QuietMode command

Syntax: `-q, --quietMode`

Example:

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

-rstbl

Bootloader reset command

Syntax: `-rstbl`

Example (for SPI):

```
-c port=spi -rstbl
```

-vb

Verbosity command

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 (result shown in the following figure):

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

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

Fig. 3.105: Verbosity command

-log

Log command

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
```

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

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

Fig. 3.106: Log command

The log file trace.log contains verbose messages:

-el

External loader command

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

Syntax: `-el [externalLoaderFilePath1.stldr]`

[externalLoaderFilePath1.stldr]
Absolute path of an external loader file.

Syntax: `-el [externalLoaderFilePath1.stldr] ... -el [externalLoaderFilePath10.stldr]`

[externalLoaderFilePath1.stldr] ... [externalLoaderFilePath10.stldr]
Absolute paths of external loader files.

Example 1:

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

Example 2:

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

Example 3:

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

Note

This command is supported only with SWD/JTAG ports. A maximum of ten external loaders can be used. It is recommended to use the normal connection mode when trying to download data in an external memory.

-elbl

External loader bootloader command

Description: With this command, the user can provide the path of an external memory loader used to perform programming, write, erase, and read operations using bootloader interfaces. The command is used for both SFI and external memory programming via bootloader interfaces. When accessing the external memory via bootloader, the open bootloader is loaded into RAM to perform all the operations using bootloader interfaces.

Syntax: `-elbl [externalLoaderFilePath.stldr]`

[externalLoaderFilePath.stldr]
Absolute path of external loader file.

Example 1:

```
STM32_Programmer_CLI.exe -c port=usb1 -elbl MX25LM51245G_STM32L552EEVALSFIX-BL.stldr -sfi out.
↪sfix hsm=0 license.bin -rsse RSSe\L5\enc_signed_RSSe_sfi_jtag.bin
```

Example 2 (external memory programming):

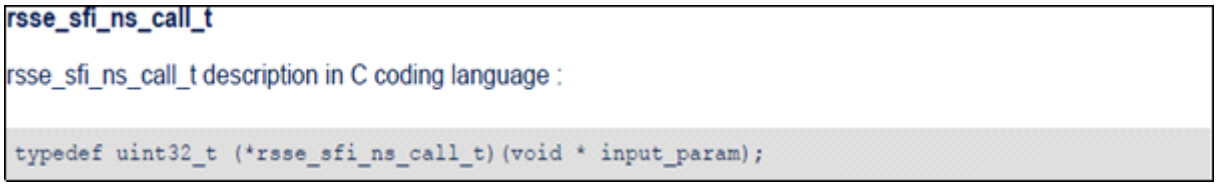
```
STM32_Programmer_CLI.exe -c port=usb1 -elbl MX66UW1G45G_STM32H7S78-DKSFIx.stldr -w file.bin␣
↪0x70000000
```

Note

This command is supported only with bootloader interfaces to program an external memory in SFIx or OpenBootloader-STM32H7R/S scenarios. To program an external memory via bootloader, each time the device is disconnected, perform a hardware reset. For STM32H7RS products, choose the external loaders ending with `-SFIx`.

External loader for SFIx

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



As a consequence the implementation of these function inside the external loader must be slightly modified to be synchronized with input parameters. Example of Sector erase function after modification:

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

-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
```

-tzenregression

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

Syntax: `--tzenregression`

Example:

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

> **Note**
>
> This command is only supported for bootloader interfaces and MCUs with trusted zone.

-ob

Option bytes command

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 device option bytes, refer to the dedicated sections in the programming manual and reference manual, both available on www.st.com.

-safelib

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

Supported formats: bin, elf, hex and Srec.

Syntax: `-sl <file_path> <start_address> <end_address> <slice_size> [pattern]`

`<file_path>`
: 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 (only `0x400` is supported).

`[pattern]`
: Optional pattern value from `0x00` to `0xFF` (default is `0x00`).

Example

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

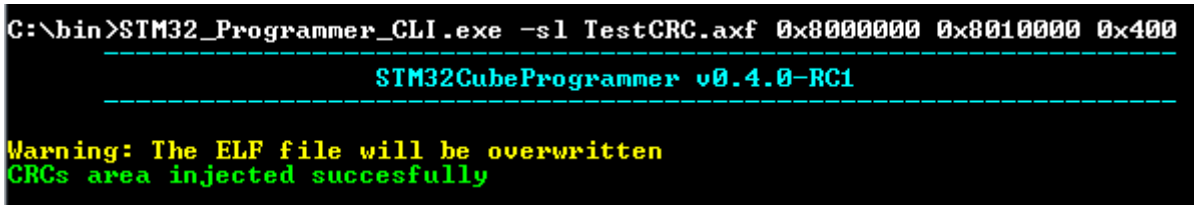


Fig. 3.107: Safety lib command

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

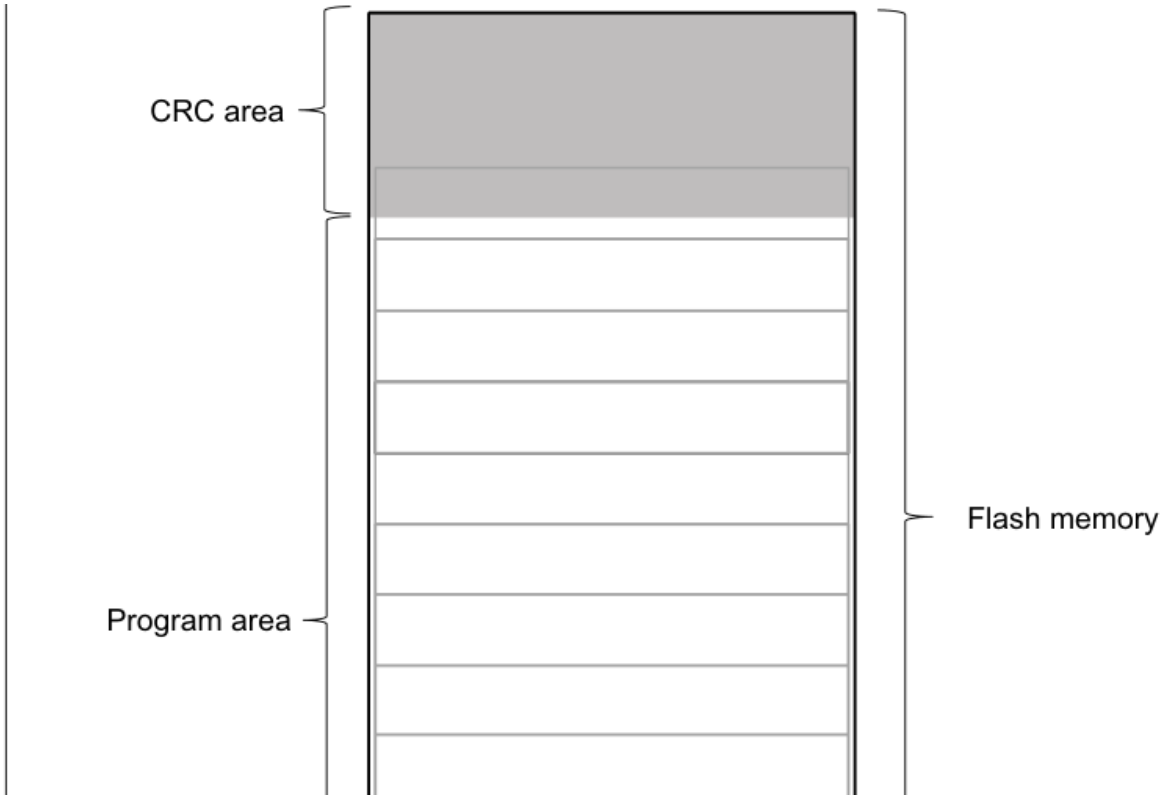


Fig. 3.108: Flash memory mapping

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

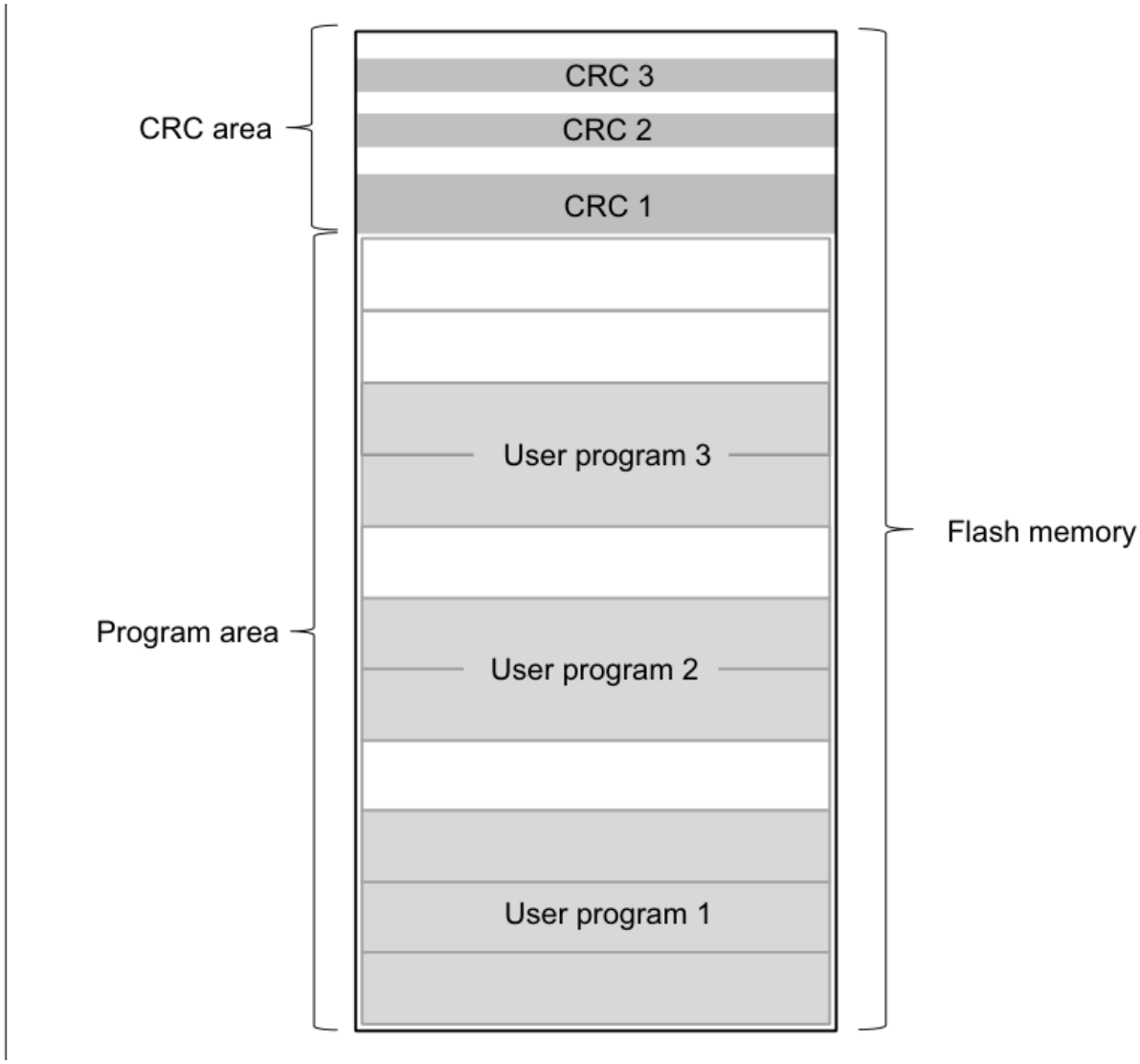


Fig. 3.109: Flash memory mapping example

The tool checks the syntax and the processing of the command, and marks a failure if there is a parameters mismatch or a problem in the CRC calculation by displaying various error message:

- "Invalid file path or wrong file extension" `<file_path>` has an unrecognized extension: `.bin .binary .hex .srec .s19 .elf .axf .out`.
- "The safety lib command is missing parameters" The passed command does not respect the parameters number.
- "Invalid flash start address" `<start_address>` is not correct (exceeds `0xFFFFFFFF` as max value) or is not in hexadecimal format.
- "Invalid flash end address" `<end_address>` is not correct (exceeds `0xFFFFFFFF` as max value) or is not in hexadecimal format.
- "Invalid slice size" `<slice_size>` is not correct (exceeds `0xFFFFF` as max value) or is not in hexadecimal format.
- "Invalid pattern" `<pattern>` is not in the range `[0x00 to 0xFF]` or is not in hexadecimal format.
- "The slice size is larger than the flash size" `<slice_size>` is larger than the dedicated flash size (`<end_address> - <start_address>`).
- "The slice size is invalid" The dedicated flash memory is not aligned (based on `<slice_size>`).
- "File is Read Only" The input file `<file_path>` is not editable.

- “No segments in this file” The input `<file_path>` does not contain any data segment.
- “The segment {x} of {y} does not start at the beginning of a flash slice” The current treated segment does not start at the beginning of the flash slice.

Secure programming SFI specific commands

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

The protected firmware installation can be performed using different communication channels, such as JTAG/SWD or bootloader interfaces (UART, SPI and USB). For more details refer to AN5054.

-sfi

SFI command

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).

-rsse

RSSe command

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 command

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

-mcsv

MCSV command

Description: This command allows the user to select the modules file including the list of modules to be installed with SFI. This option is mandatory when the SFI image integrates at least one module (area of type m).

Syntax: -mcsv <module_path.mcsv>

<module_path.mcsv> : Path of mcsv fil

Secure programming SFIx specific commands Secure firmware install (SFIx) is a feature supporting secure external firmware flashing, available on some STM32 devices with OTFDEC capability. The firmware provider has the possibility to protect its external firmware/data against any illegal access, and to control the number of devices that can be programmed. The SFIx secure programming can be carried out only with JTAG/SWD interface. For more details refer to AN5054

-sfi

SFIx command

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). -el –extload Selects a custom external memory-loader, only for the JTAG/SWD interfaces <file_path> External memory-loader file path -elbl –extloadblSelects a custom external memory-loader for the bootloader interface <file_path> External memory-loader file path

-rsse

RSSe x command

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 SFI command

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

Syntax: -a

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

-mcsv

MCSV SFI command

Description: This command allows the user to select the modules file including the list of modules to be installed with SFI. This option is mandatory when the SFI image integrates at least one module (area of type m).

Syntax: -mcsv <module_path.mcsv>

<module_path.mcsv> : Path of mcsv file

-ecsv

Description: This command is used for Secure Manager Secure Modules install in external flash. It allows the user to select the modules file including the list of external modules to be installed with SFIx. This option is mandatory when the SFI image integrates at least one external module (area of type e).

Syntax: -ecsv : Path of ecsv file

<exmodule_path.ecsv>: Path of ecsv file

HSM related commands

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

Two HSM versions are available:

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

Before using the HSM, it must be programmed using Trusted Package Creator. This tool can program both versions with some specific input configurations, as detailed in UM2238. For more details, refer to AN5054.

-hsmgetinfo

Description: Reads the HSM available information.

Syntax: -hsmgetinfo [slot=<SlotID>]

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

-hsmgetcounter

Description: Reads the current value of the license counter.

Syntax: -hsmgetcounter [slot=<SlotID>]

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

-hsmgetfwid

Description: Reads the Firmware/Module identifier.

Syntax: -hsmgetfwid [slot=<SlotID>]

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

-hsmgetstatus

Description: Reads the current card life-cycle state.

Syntax: -hsmgetstatus [slot=<SlotID>]

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

-hsmgetlicense

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

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

<file_path> File path into where the received license is stored [slot=<SlotID>] Slot ID of the smart card reader Default value: slot = 1 (the PC integrated SC reader) [<protocol=Ptype>] Protocol type to be used: static/live Only static protocol is supported so far Default value: static

–hsmglfcb

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

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

<certif_file_path.bin> File path from which the input certificate is read. <license_file_path.bin> File path where the received license is stored [slot=<SlotID>] Slot ID of the smart card reader. Default value: slot = 1 (the PC integrated SC reader)

STM32WB specific commands

-antirollback

Description: Perform the antirollback operation.

Syntax: -antirollback

-startfus

Description: Start the FUS.

Syntax: -startfus

-getuid64

Description: Read the device unique identifier (UID).

Syntax: -getuid64

-fusgetstate

Description: Read the FUS state.

Syntax: -fusgetstate

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

-fwdelete

Description: Delete the BLE stack firmware.

Syntax: -fwdelete

-fwupgrade

Description: Upgrade of BLE stack firmware or FUS firmware.

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

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

-startwirelessstack

Description: Start the wireless stack.

Syntax: -startwirelessstack

-authkeyupdate

Description: Authentication key update.

Syntax: -authkeyupdate <file_path>

-authkeylock

Description: Authentication key lock. Once locked, it is no longer possible to change it using -authkeyupdate command.

Syntax: -authkeylock

-wusrkey

Description: Write user key.

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

-swv

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

Syntax: -swv <freq=<frequency» <portnumber=0-31|all> [<file_path.log>] [-RA]

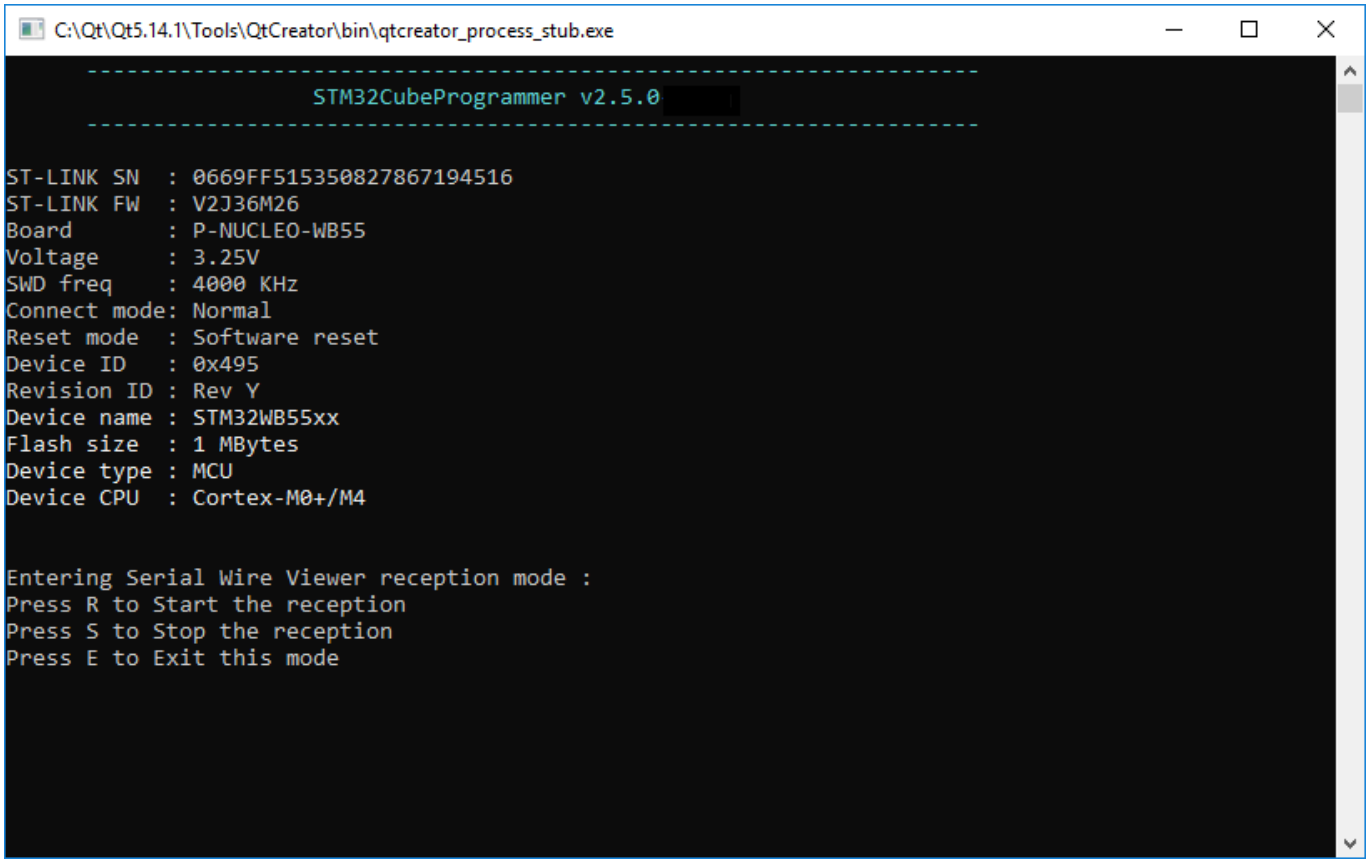


Fig. 3.110: SWV command

-startswv

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

Syntax: -startswv <freq=<frequency» <portnumber=0-31|all> [<file_path.log>]

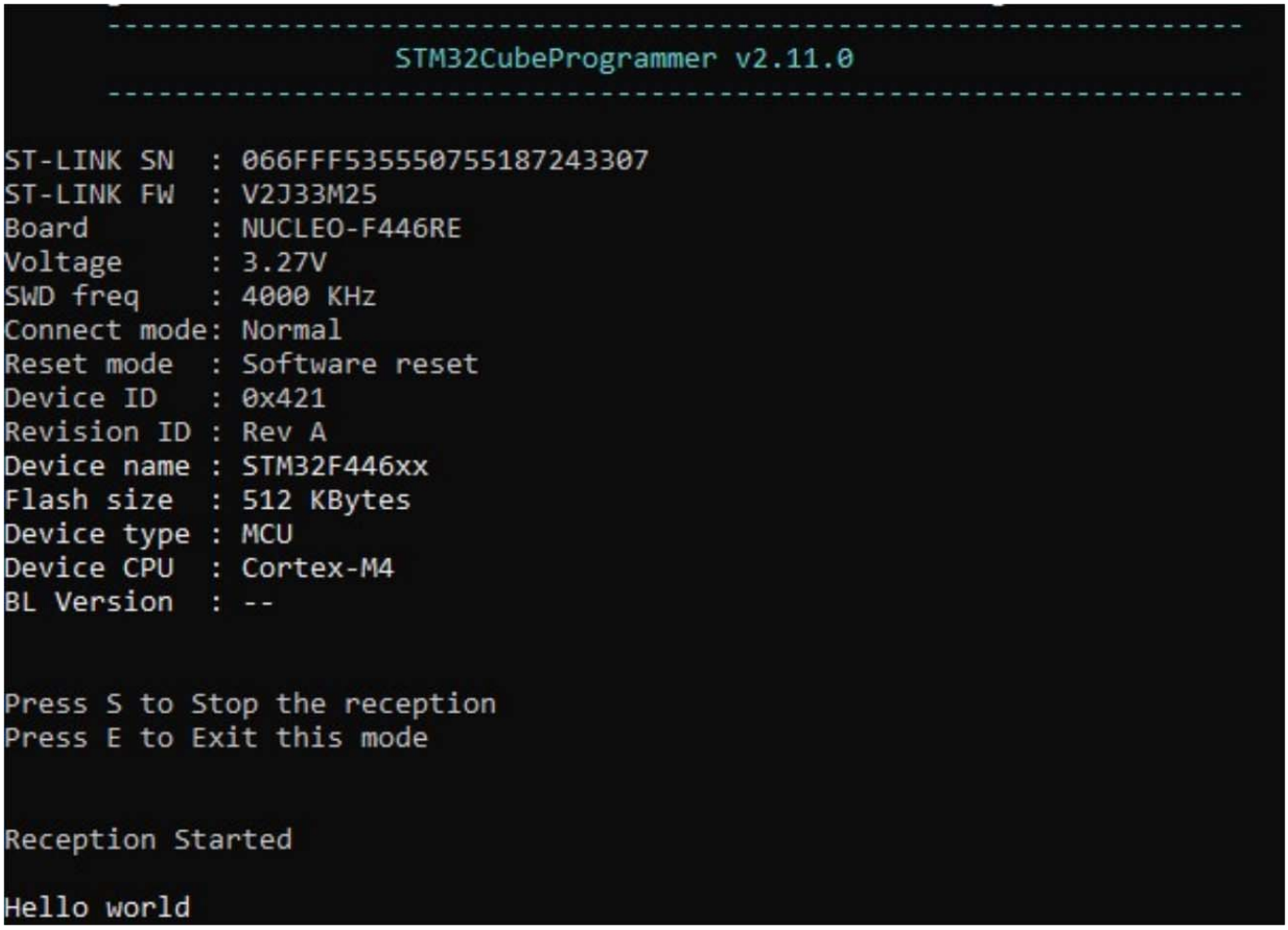


Fig. 3.111: startswv command

-dsecurity

Description: This command allows the user to disable security on STM32WL.

Syntax: -dsecurity

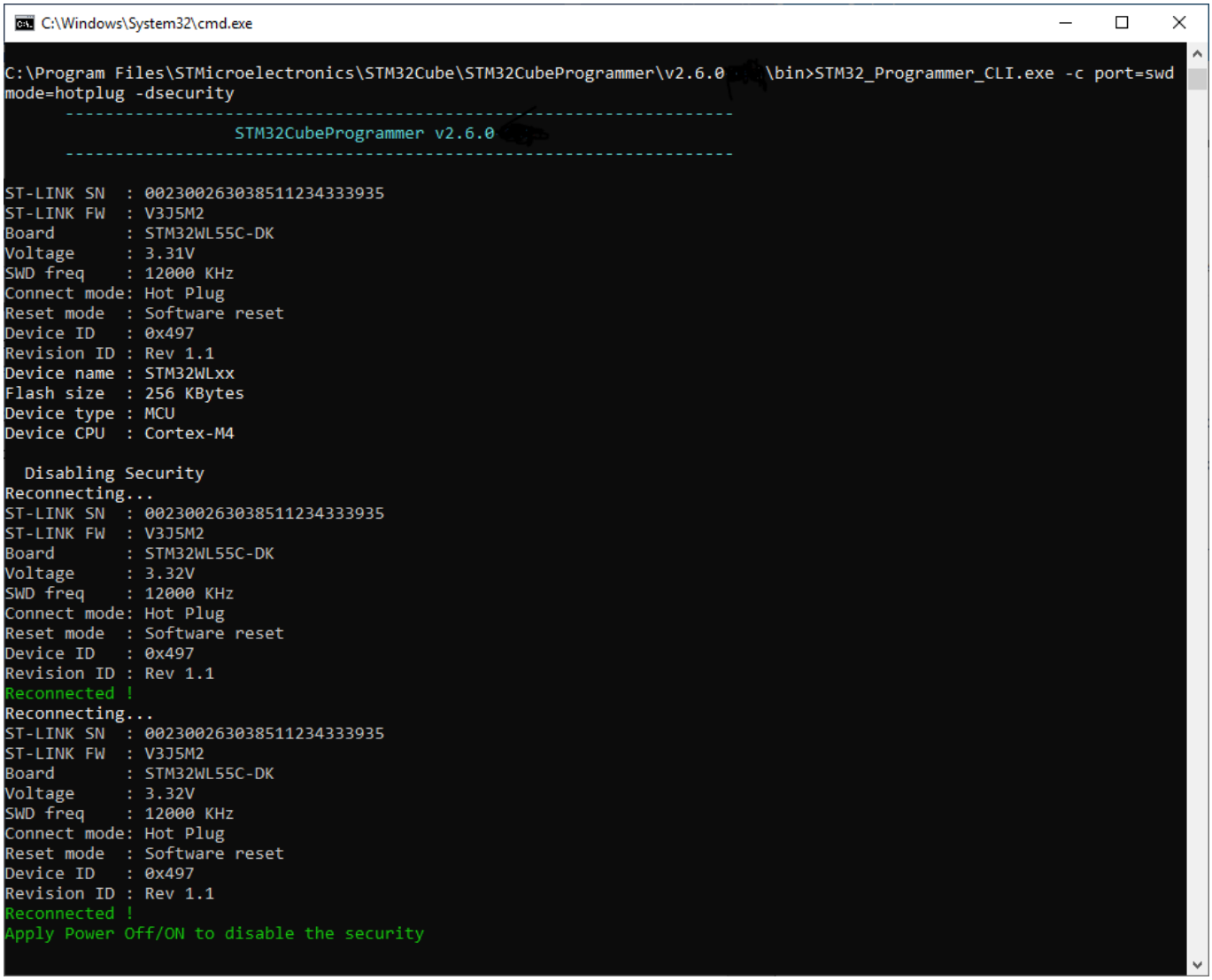


Fig. 3.112: Disable security

-setdefaultob

Description: This command allows the user to configure option bytes to their default values on STM32WL.

Syntax: -setdefaultob

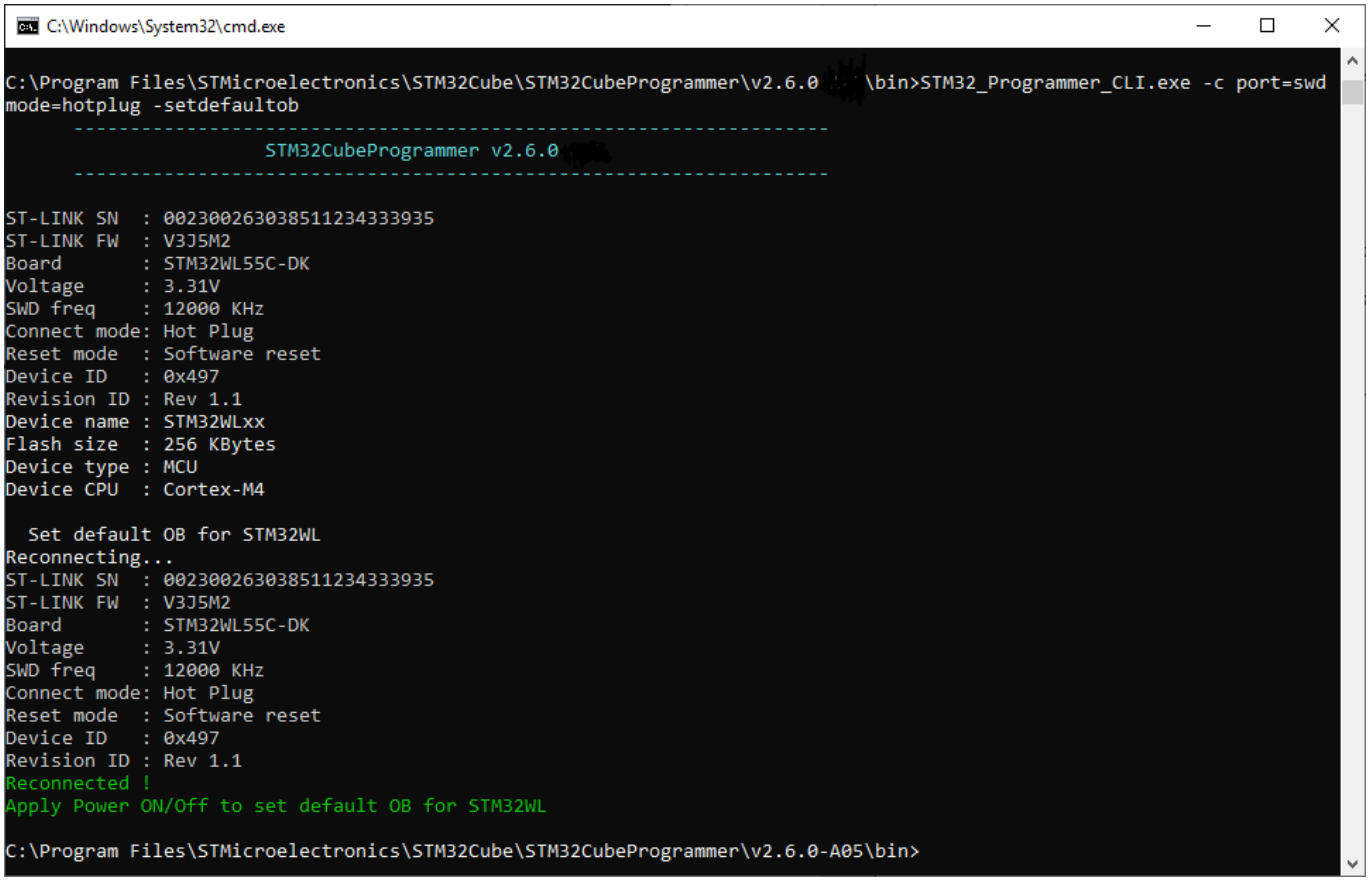


Fig. 3.113: Configure option bytes to their default values

-unlockchip

Description: This command allows the user to unlock the device if bad option bytes are programmed.

Syntax: -ob unlockchip

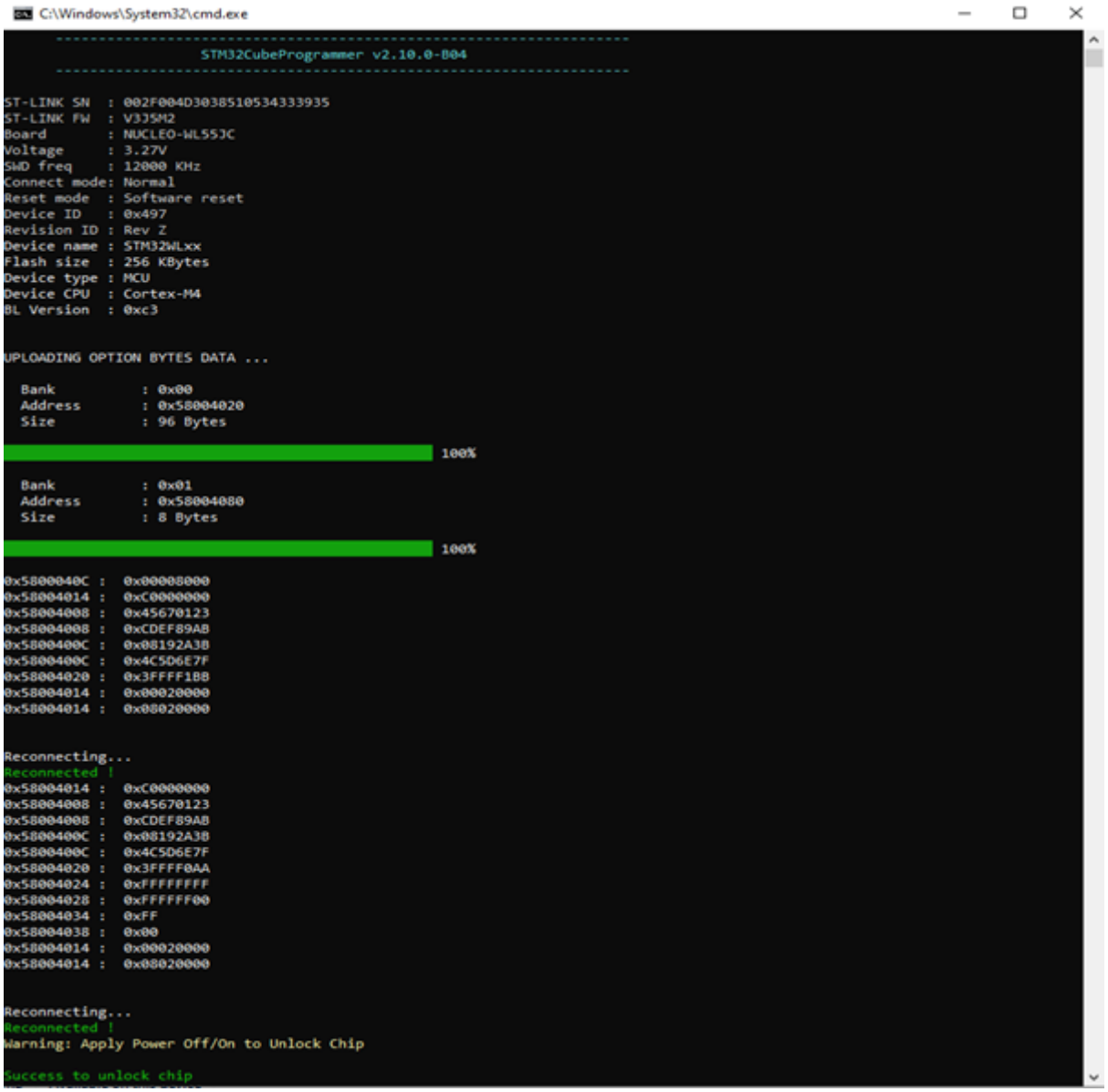


Fig. 3.114: Output of unlockchip command

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

-ssigfoxc

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

Syntax: -ssigfoxc <binary_file_path>

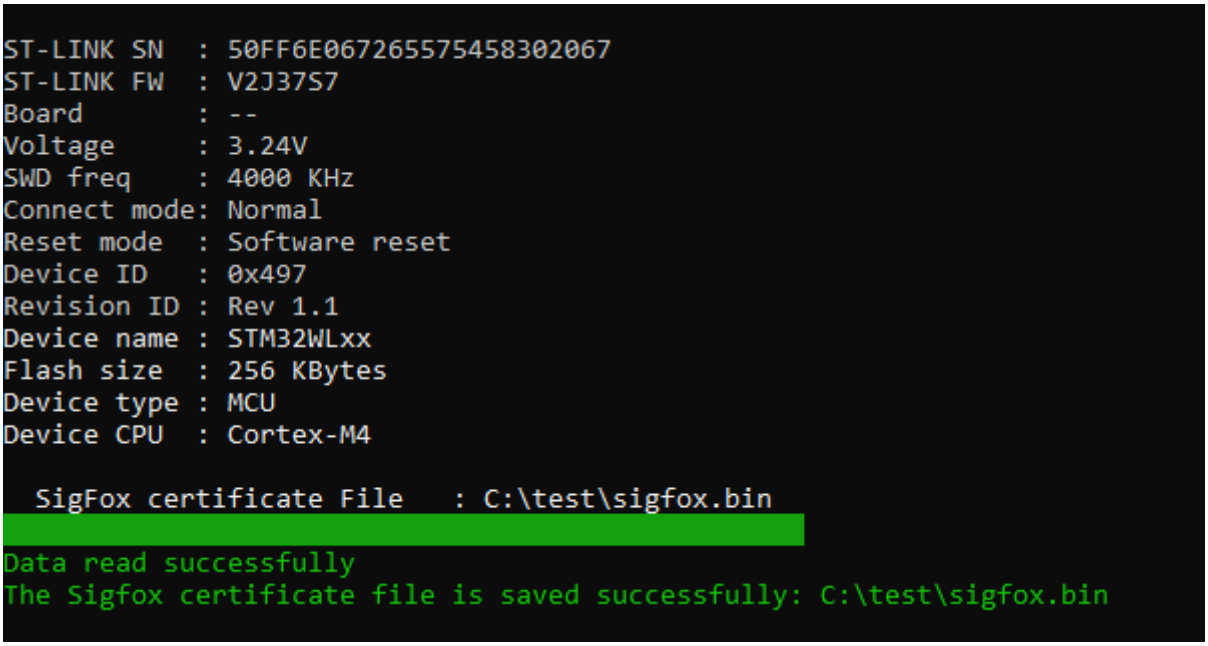


Fig. 3.115: Example of -ssigfoxc command

-wsigfoxc

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

Syntax: -wsigfoxc <binary_file_path> <address>

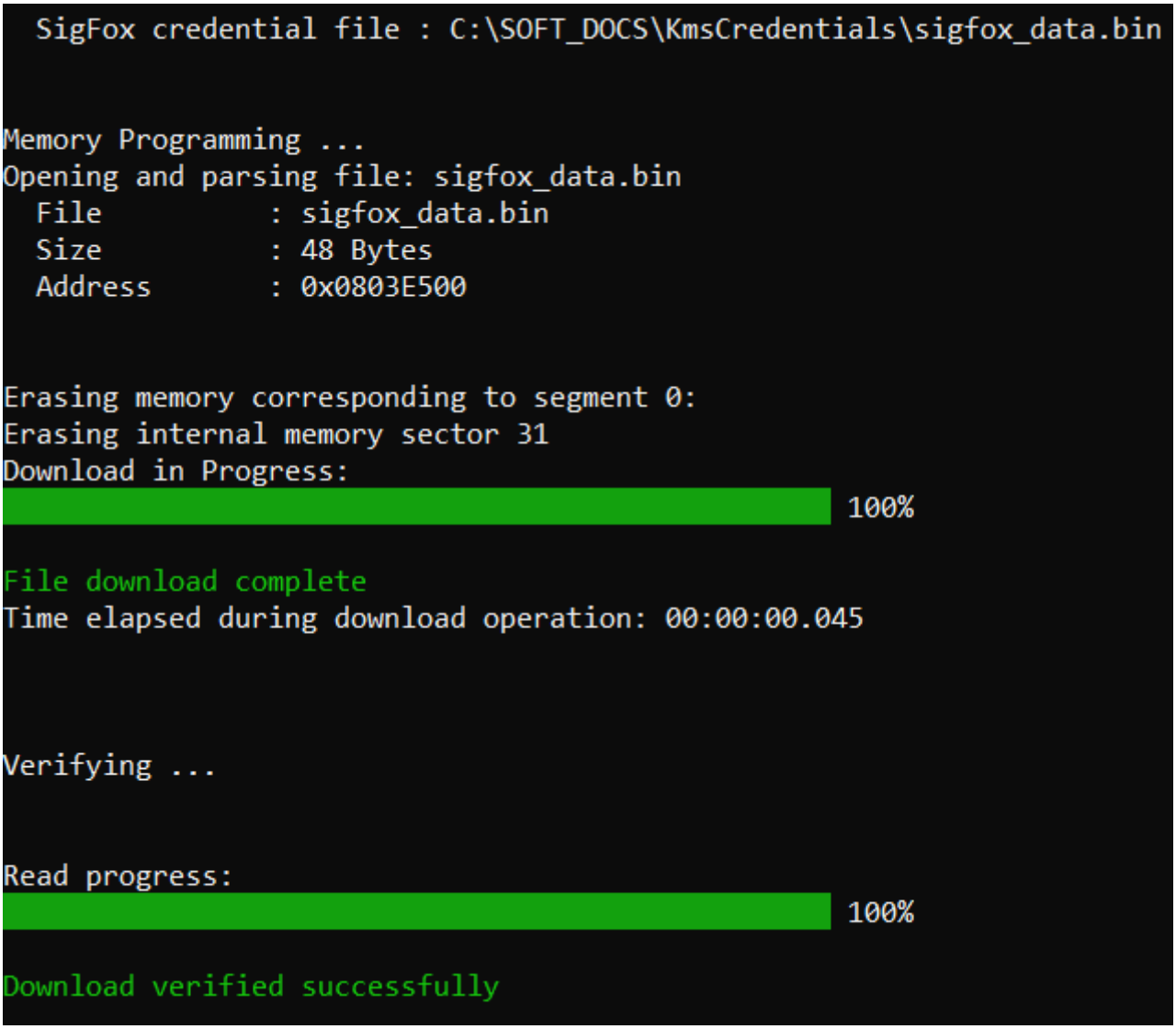


Fig. 3.116: Example 1 of -wsigfoxc command

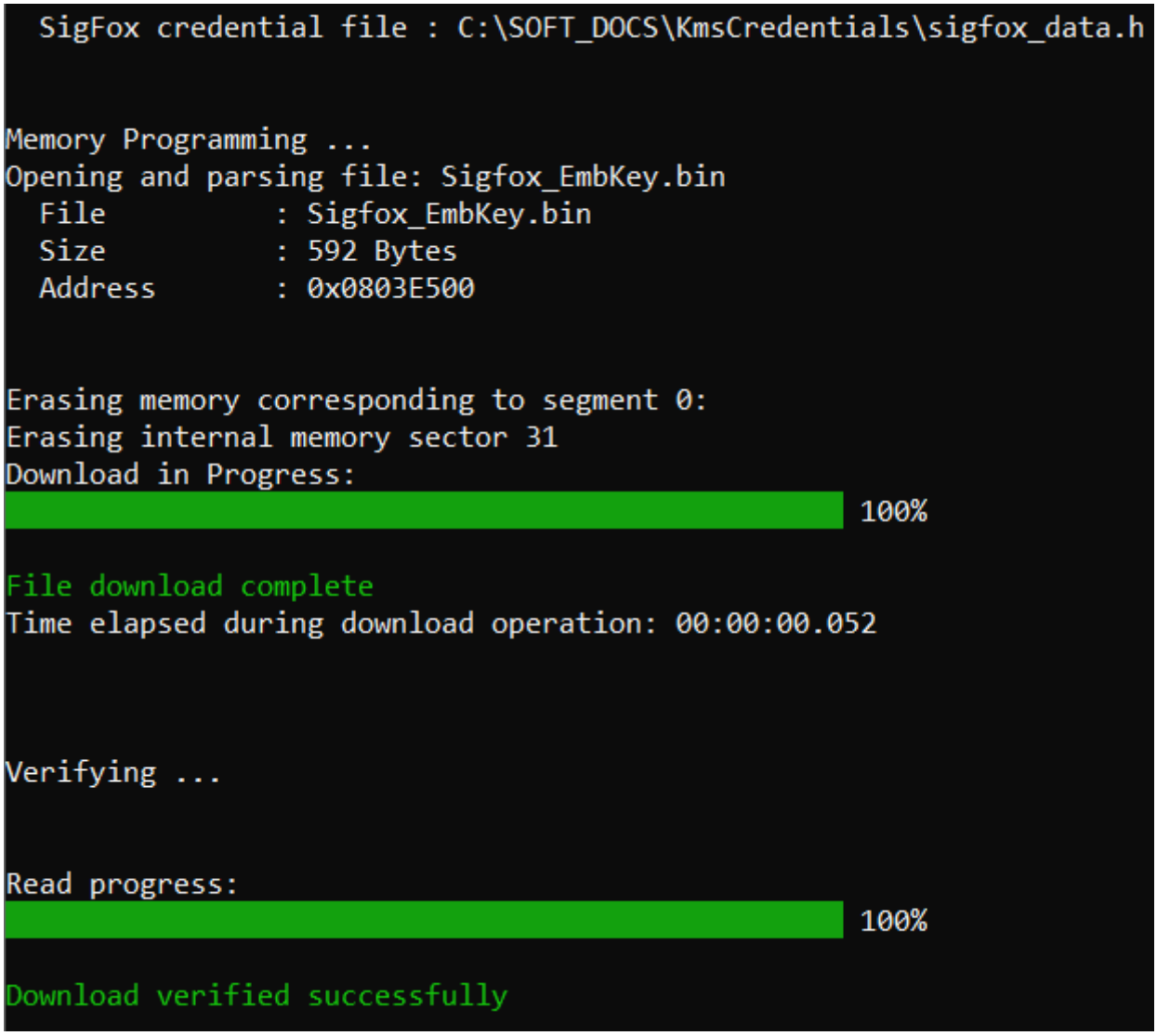


Fig. 3.117: Example 2 of -wsigfoxc command

-regdump

Description: Reads and dumps core and MCU registers.

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

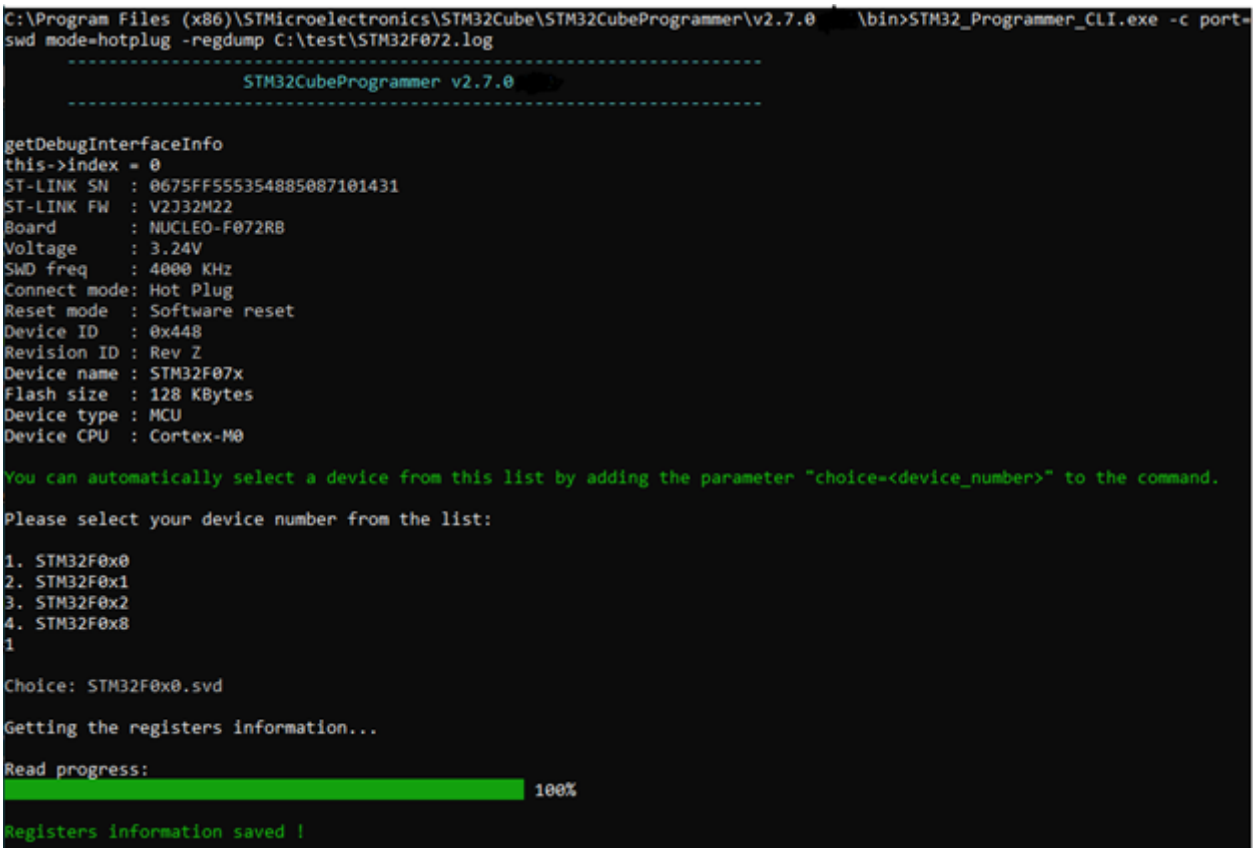


Fig. 3.118: Read core and MCU registers

-hf

Description: Starts hard fault analysis and displays the output trace needed to understand the fault reason.

Syntax: -hf

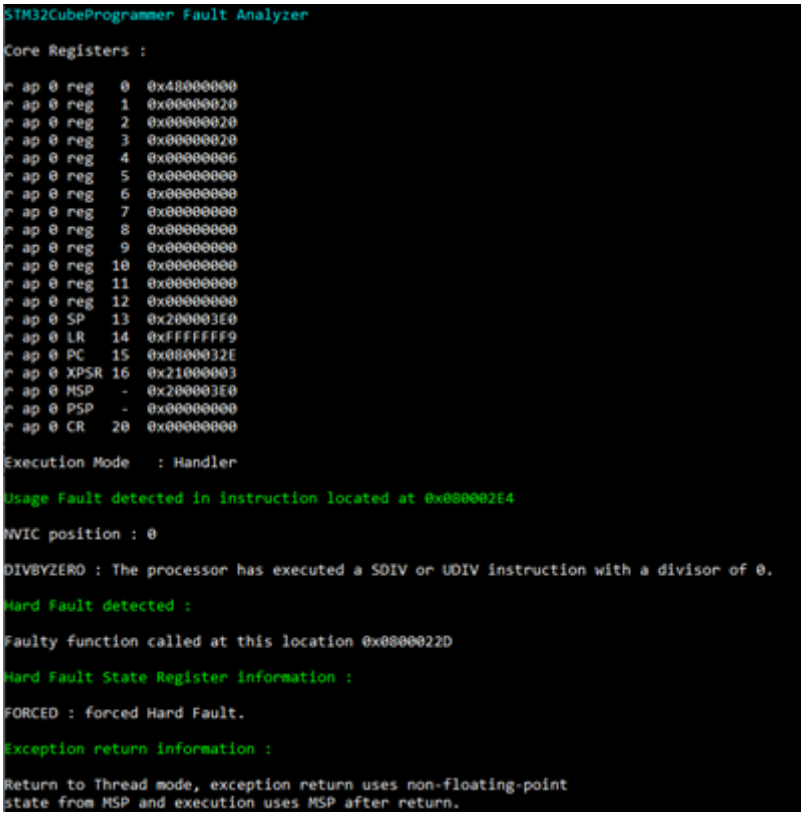


Fig. 3.119: Fault analyzer CLI view when Hard Fault is detected

-fchecksum

Description: Calculates the checksum value for the entire file content using an addition algorithm and displays the resulting value.

Syntax: -fchecksum, –file-checksum <file_path>

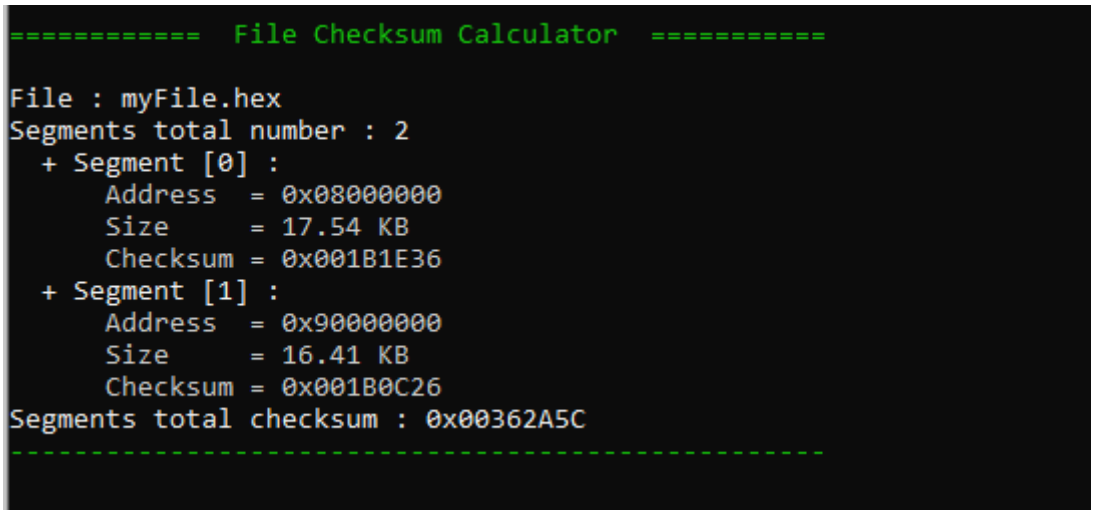


Fig. 3.120: Example of File checksum command

-checksum

Description: Calculates the checksum value for any accessible memory type, including internal flash memory, and displays the resulting output.

Syntax: -checksum, –checksum <address> <size>

```
ST-LINK SN  : 066AFF3133314642570419 20
ST-LINK FW  : V2J40M27
Board       : 32F769IDISCOVERY
Voltage     : 3.24V
SWD freq    : 4000 KHz
Connect mode: Normal
Reset mode  : Software reset
Device ID   : 0x451
Revision ID : Rev Z
Device name : STM32F76x/STM32F77x
Flash size  : 2 MBytes
Device type : MCU
Device CPU  : Cortex-M7
BL Version  : 0x93

Memory checksum calculating ...
Memory [0x08000000 : 0x08200000] - Checksum : 0x1FDFF4DC
```

Fig. 3.121: Checksum command output for the internal flash memory

```
ST-LINK SN  : 066AFF3133314642570419 20
ST-LINK FW  : V2J40M27
Board       : 32F769IDISCOVERY
Voltage     : 3.24V
SWD freq    : 4000 KHz
Connect mode: Hot Plug
Reset mode  : Software reset
Device ID   : 0x451
Revision ID : Rev Z
Device name : STM32F76x/STM32F77x
Flash size  : 2 MBytes
Device type : MCU
Device CPU  : Cortex-M7
BL Version  : 0x93

Memory checksum calculating ...
Memory [0x90000000 : 0x90000200] - Checksum : 0x0001FE00
```

Fig. 3.122: Checksum command output for an external memory

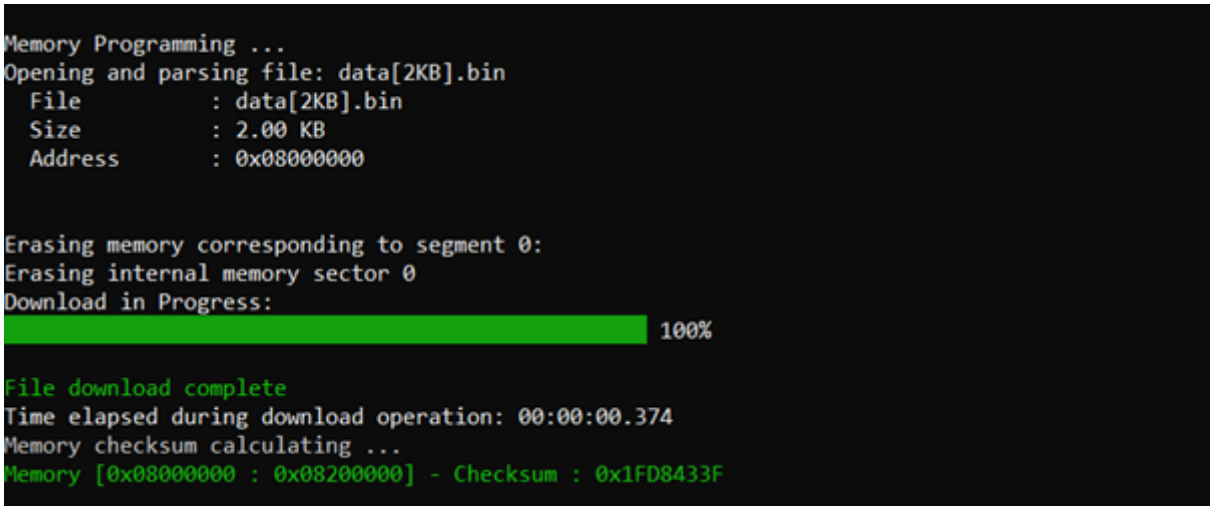


Fig. 3.123: Checksum command output at the end of file download

-lockRDP1

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

Syntax: -lockRDP1 <OEM1 least significant bytes key> <OEM1 most significant bytes key>

-lockRDP2

Description: Allows the user to lock the RDP regression from level 2 with a password.

Syntax: -lockRDP2 <OEM2 least significant bytes key> <OEM2 most significant bytes key>

-unlockRDP1

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

Syntax: -unlockRDP1 <OEM1 least significant bytes key> <OEM1 most significant bytes key>

-unlockRDP2

Description: Allows the user to unlock the RDP regression from level 2 with a password.

Syntax: -unlockRDP2 <OEM2 least significant bytes key> <OEM2 most significant bytes key>

-verifyrdp2

Description: The OEMKEY can be validated while in RDP L0, to ensure the provisioned password is validated before setting RDP 2.

RDP lock verification sequence:

- Set the OEM key values (Set password)
- Verify whether the OEM keys are correct (Verify)

Syntax: -verifyrdp2 <OEM2 R1> <OEM2 R2> <OEM2 R3> <OEM2 R4>

-lockbs

Description: This command provisions the BS Key, which is a 32-bit password allowing RDP transition from L2 with boundary scan to L2.

Syntax: -lockbs <BSKEY>

-unlockbs

Description: This command performs the RDP transition using the provisioned BSKEY or 0xAAAAAAAA if no BSKEY provisioning has been done.

Syntax: -unlockbs <BSKEY>

-gc

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

Syntax: -gc certification.bin

-w32dbgmcu

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

Syntax: -w32dbgmcu <32_data_bits>

-sdp

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

Syntax: -sdp [OBKey_File_Path.obk]



Fig. 3.124: OBKey provisioning example

-pwd

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

Syntax: -pwd value=<password> path=<PasswordOPath>

-debugauth

Description: Performs debug-auth discovery, authentication using a password or certificate, or a debug close operation on supported devices.

Syntax: `debugauth=2 / pwd=<password_path.bin> [magic=STDA|OEDA] debugauth=1 / per=<Permission> key=<Key_Path.pem> cert=<Certificate_Path.b64> [magic=STDA|OEDA] debugauth=1 / debugauth=3`

`magic=STDA|OEDA` selects the debug-authentication magic. The default is `STDA`. For STM32N6, set `magic=OEDA`; this value is mandatory.

```
Start Debug Authentication Sequence

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

open_comms                    :   438 : open       : Writing magic number

open_comms                    :   446 : open       : De-asserting target reset

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

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

Debug Authentication: Discovery Success
```

Fig. 3.125: Discovery log

```
Start Debug Authentication Sequence

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

open_comms                    :   438 : open       : Writing magic number

open_comms                    :   446 : open       : De-asserting target reset

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

[00%]   discovery command
[10%]   sending discovery command
[20%]   receiving discovery
response_packet_lock
[40%]   loading credentials
[50%]   sending challenge request
[60%]   receiving challenge
response_packet_lock
[70%]   signing token
SDMAuthenticate               :  1131 : client     : Found 3 certificates

response_packet_lock
response_packet_lock
response_packet_lock
[80%]   sending respose
[90%]   receiving status
response_packet_lock
SDMAuthenticate               :  1229 : client     : Authentication successful

[100%]  finished authentication

Debug Authentication Success
```

Fig. 3.126: Debug authentication with password

```
Start Debug Authentication Sequence

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

open_comms                    :   438 : open       : Writing magic number

open_comms                    :   446 : open       : De-asserting target reset

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

[00%]   discovery command
[10%]   sending discovery command
[20%]   receiving discovery
response_packet_lock
[40%]   loading credentials
[50%]   sending challenge request
[60%]   receiving challenge
response_packet_lock
SDMAuthenticate               :  1131 : client     : Found 1 certificates

[80%]   sending ST password
[90%]   receiving response
response_packet_lock
[100%]  authentication successful
SDMAuthenticate               :  1195 : client     : Authentication successful


Debug Authentication Success
```

Fig. 3.127: Debug authentication with certificate

–force_no_da

Description: This option forces the product state to OB programming even when debug authentication is not configured.

Syntax: –force_no_da

-installipmodule

Description: Installs a Secure Manager module using HSM or a global license.

Syntax: -installipmodule <file_path> [hsm=0|1] [slot=slotID] [address]

-updateipmodule

Description: Updates a Secure Manager module.

Syntax: -updateipmodule <file_path> <address>

-skiperase

Description: Skips sector erase before programming.

Syntax: –skiperase

-storekeyotp

Description: Stores the authentication public key in the OTP. This command applies only to STM32WB0x and STM32WL33 devices and can be performed only with UART.

Syntax: -storekeyotp <key_path> <start_address>

-rssekw

Description: Allows the user to select the RSSeKW library for key wrapping and execute wrapping of the private key provisioned or provided by the user.

Syntax: -rssekw <rssekw .bin file path> <wrapping output file path> [KeyType=eccchipfu|eccchiplu] [ExportPublicKey=yes|no] [WrappingKeySelect=dhuk] [KeyUsage=ecc_usage_sign|ecc_usage_scalar_mul] [SecAttr=secure|non_secure] [PrivAttr=privilege]

- `<.bin file_path>`: RSSe key wrapping file path.
- `<.bin file_path>`: Output `.bin` file path.
- `[KeyType=<eccchipfu|eccchiplu>]`: Private key to wrap.
- `[ExportPublicKey=<Yes|No>]`: Generate or not the public key.
- `[WrappingKeySelect=<DHUK>]`: Method used to wrap the private key.
- `KeyUsage=<ECDSA_USAGE_SIGN|ECC_USAGE_SCALAR_MUL>[]`: Usage of the private key.
- `[SecAttr=<SECURE|NON_SECURE>]`: Secure or nonsecure context.
- `[PrivAttr=<PRIVILEGE|NON_PRIVILEGE>]`: Privileged or unprivileged context. Value = `Privilege` for ECC chip.

For STM32H5 series:

Syntax: -rssekw <rssekw.bin file path> <wrapping output file path> [KeyType=eccchipfu|eccchiplu] [ExportPublicKey=yes|no] [WrappingKeySelect=dhuk] [KeyUsage=ecc_usage_sign|ecc_usage_scalar_mul] [SecAttr=secure|non_secure] [PrivAttr=privilege] [epochsel=0xFFF277D1]

[epochsel=0xFFF277D1] Epoch select value, for ECC chip private keys. Only one value is possible: 0xFFF277D1.

-r32fast

Description: Fast read 32-bit data memory.

Syntax: -r32fast

–yes

Description: STM32V8 PCM programming «Power pulse » mode presents extra programming constraint, only four trials are allowed for internal memory programming (including OTP programming). When setting the Option Byte PM=1, the internal memory can be programmed only five times, four times for OTP programming, beyond this limit there is a risk of loosing the V8 devices. The tool automatically blocks this operation and raises an error. The user should re-enter the command adding: -y/–yes.

Syntax: -y, –yes

Example:

```
./STM32_Programmer_CLI -c port=swd -d "C:\Users\path.bin" 0x08000000 -y
```

-blankcheck

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

Syntax:

```
-blankcheck
```

Example:

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

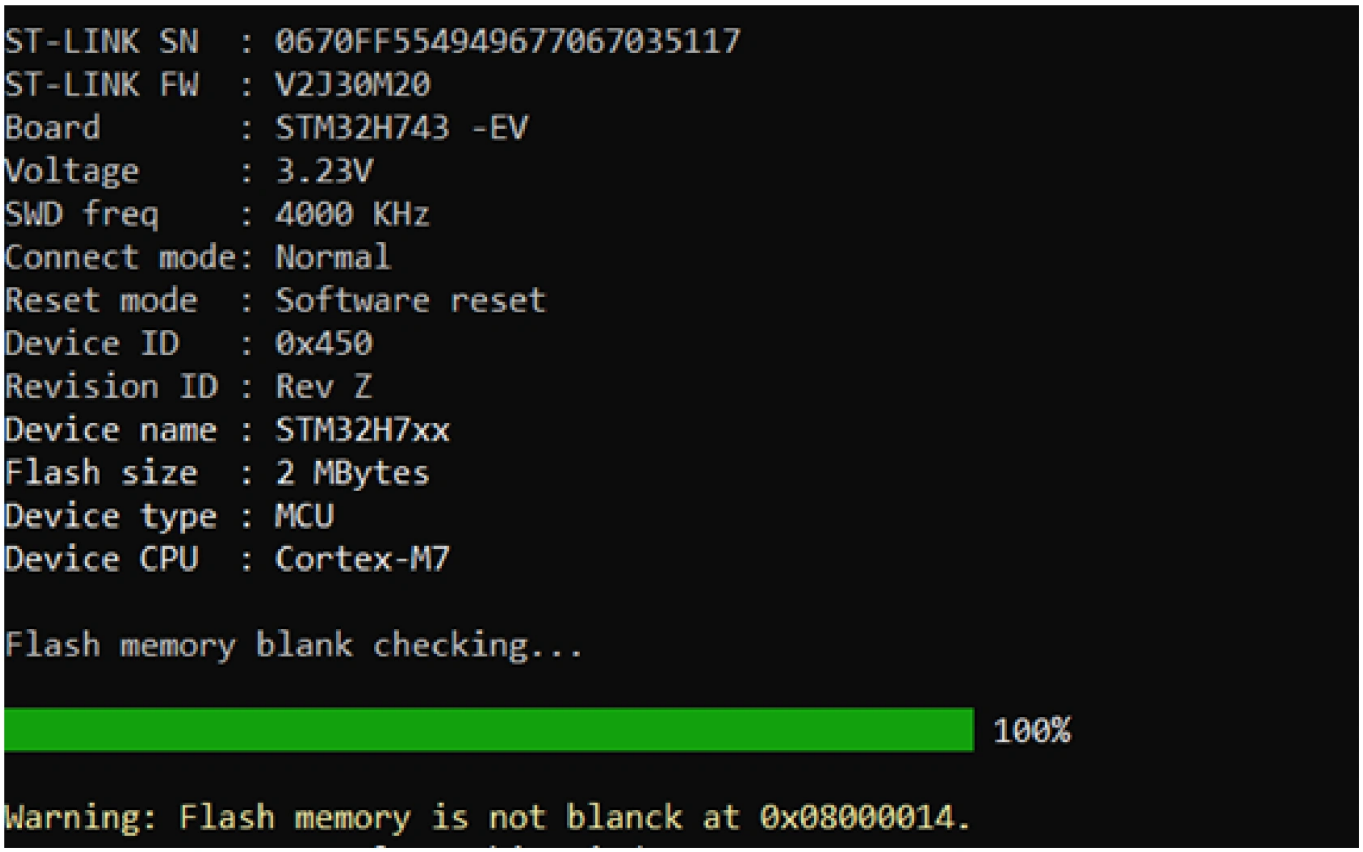


Fig. 3.128: Example 1: memory is not blank at address 0x08000014

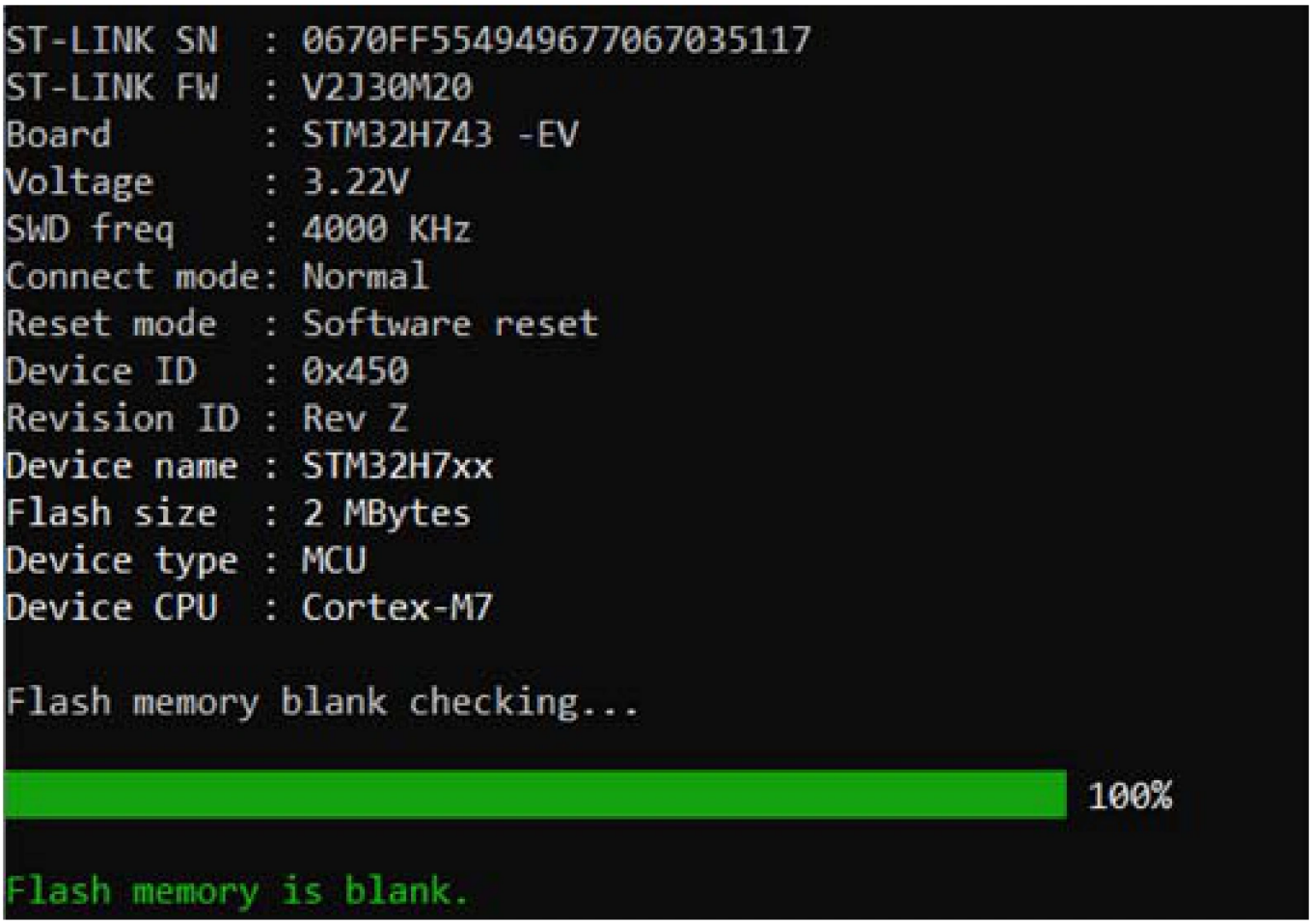


Fig. 3.129: Example 2: memory is blank

-fillmemory

Syntax:

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

Examples:

- Example 1:

```
STM32_Programmer_CLI.exe -c port=swd -fillmemory 0x08000000 size=0x10 pattern=0XAA␣
↪datawidth=16
```

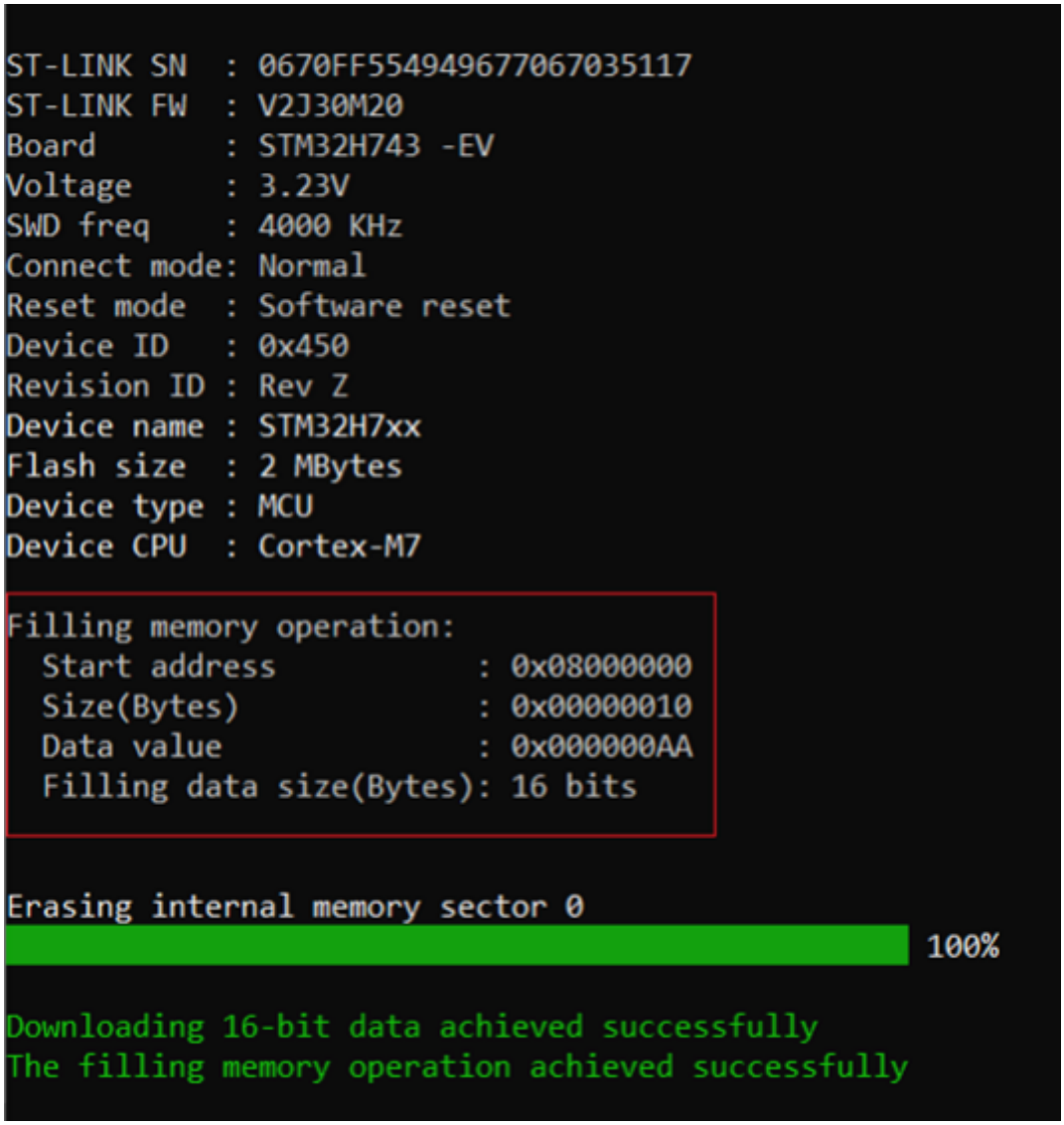


Fig. 3.130: Fill memory command - Example 1

- Example 2:

```
STM32_Programmer_CLI.exe -c port=swd -fillmemory 0x08000000 size=0x10 pattern=0XCC␣
↪datawidth=32
```

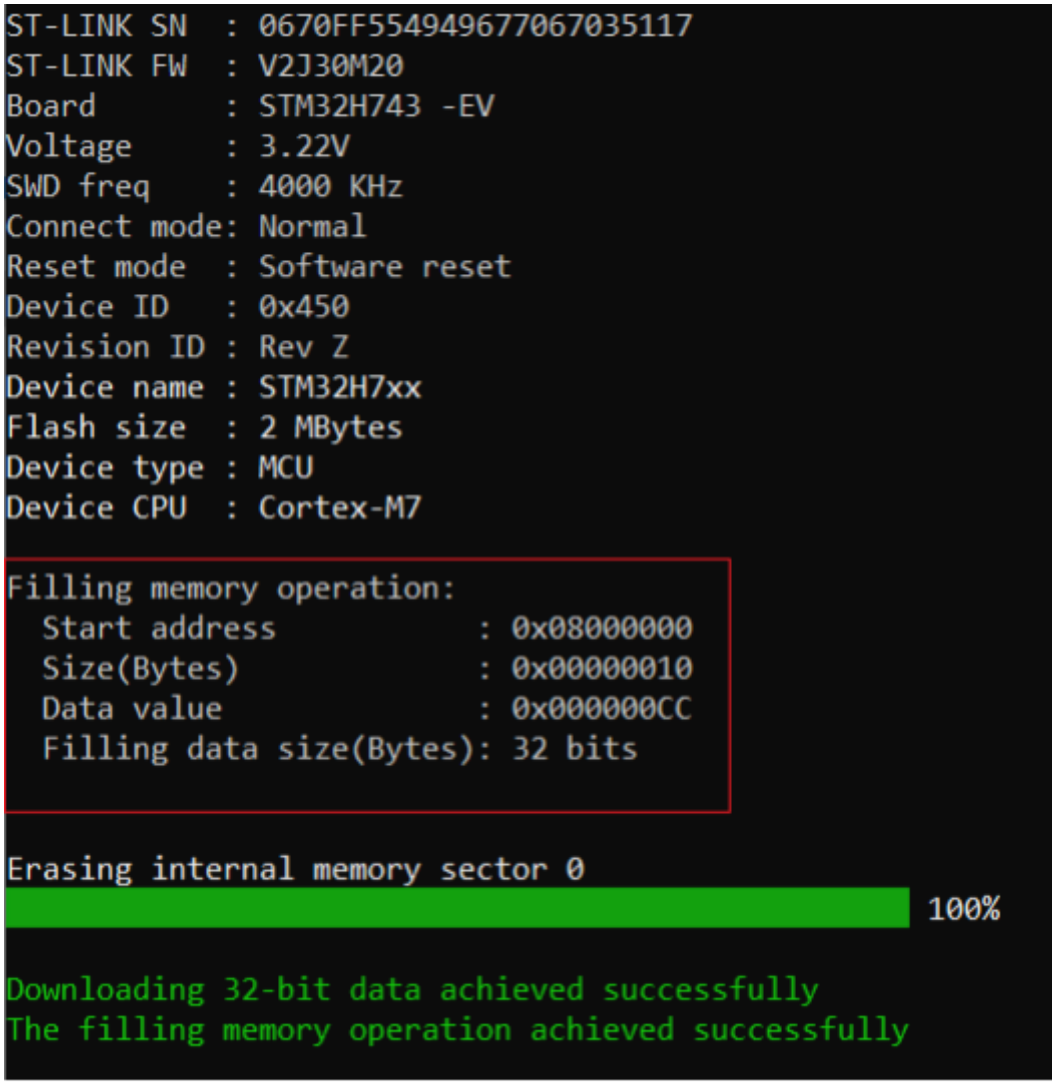


Fig. 3.131: Fill memory command - Example 2

`<start_address>`:

Start address for write. The address `0x08000000` is used by default.

`[size=<value>]`:

Size of the data to write.

`[pattern=<value>]`:

The pattern value to write.

`[datawidth=8|16|32]`:

Filling data size can be 8, 16, or 32 bits. The selected value by default is 8 bits.

MPU command lines

Available commands for STM32MPU

This section details the commands supported on STM32MP1 devices.

Long command	Short command	Description
–connect	-c	Establishes the connection to the device. This command allows the host to open the chosen device port (UART/USB)
–phaseID	-p	Allows the user to know the next partition ID to be executed.
–download,–write	-d, -w	Downloads the content of the specified binary file into a specific partition in the flash or SYSRAM memories.
-w	-w	Loads sequentially the partitions requested by the bootloader using a TSV file.
–go,–start	-g, -s	Executes the device memory starting from the specified address.
–readPart	-rp	Reads and uploads the specified partition content into a binary file.
–list	-l	Lists all available communication interfaces UART and USB.
–quietMode	-q	Disables the progress bar display during download and read partition commands.
–verbosity	-vb	Allows the user to display more messages and be more verbose.
–log	-log	Stores the whole traffic with maximum verbosity into a log file.
-otp	-otp	Manages OTP programming operations.
-detach	-detach	Sends a detach command to USB DFU.
-gc	-gc	Reads the chip certificate and saves it to a binary file.
-wb	-wb	Sends the blob containing secrets and license.
–ssp	-ssp	Programs an SSP file.

Command details

-connect

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

Option	Description
port=<Portname>	Interface identifier: ex COMx (for Windows), /dev/ttySx (for Linux), usbx for USB interface.
[noinit=<noinit_bit>]	Sets No Init bits, value in {0,1}, default 0. Noinit = 1 can be used if a previous connection is active (no need to send 0x7F).
[br=<baudrate>]	Baudrate, (for example 9600, 115200), default 115200.
[P=<Parity>]	Parity bit, value in (EVEN, NONE, ODD), default EVEN.
[db=<data_bits>]	Data bit, value in (6, 7, 8), default 8.
[sb=<stop_bits>]	Index of the debug probe. Default index value is 0.
[sn=<serialNumber>]	Stop bit, value in (1, 1.5, 2), default 1.
[fc=<flowControl>]	Flow control, value in (OFF, Software, Hardware). Software and Hardware flow controls are not yet supported for STM32MP1 series, default OFF.

Example

Using UART:

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

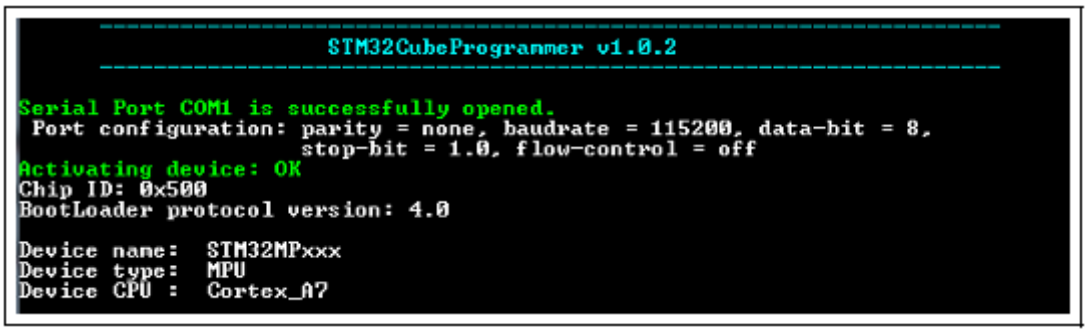


Fig. 3.132: Connect operation using RS232

Note

When using the USB interface, all the configuration parameters (such as baudrate, parity, data-bits, frequency, index) are ignored. To connect using UART interface, the port configuration (baudrate, parity, data-bits, stop-bits and flow-control) must have a valid combination.

–phaseID

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

Syntax: –phaseID

–download,–write

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

Syntax: -w <file_path> [partitionID]

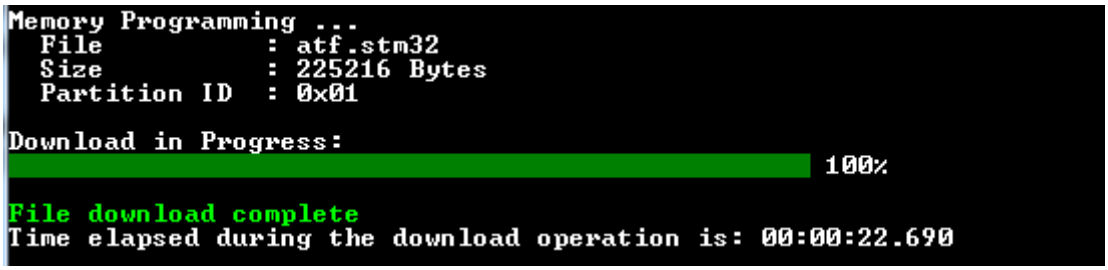


Fig. 3.133: Download operation

Note

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

-W

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

Syntax: -w <tsv file_path>

```
#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
```

Fig. 3.134: TSV file format

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>).

-go,-start

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

Syntax: –start [start_address/Partition_ID]

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.

-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>

-list

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

Syntax: -l, –list <interface_name>

-quietMode

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

Syntax: -q, –quietMode

-verbosity

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

Syntax: -vb <level>

-log

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

Syntax: -log [filePath.log]

```
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
```

Fig. 3.135: Log file content

-otp

Description: These commands allow the user to program and manage the OTP region from a host computer.

Syntax: -otp write {lock} {word=[index] value=[val]…} / -otp lock {word=[index]…} / -otp displ {word=[index]…} / -otp fwrite {lock} [path.bin] word=[index]

Note

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



Fig. 3.136: OTP write command for OTP structure v2

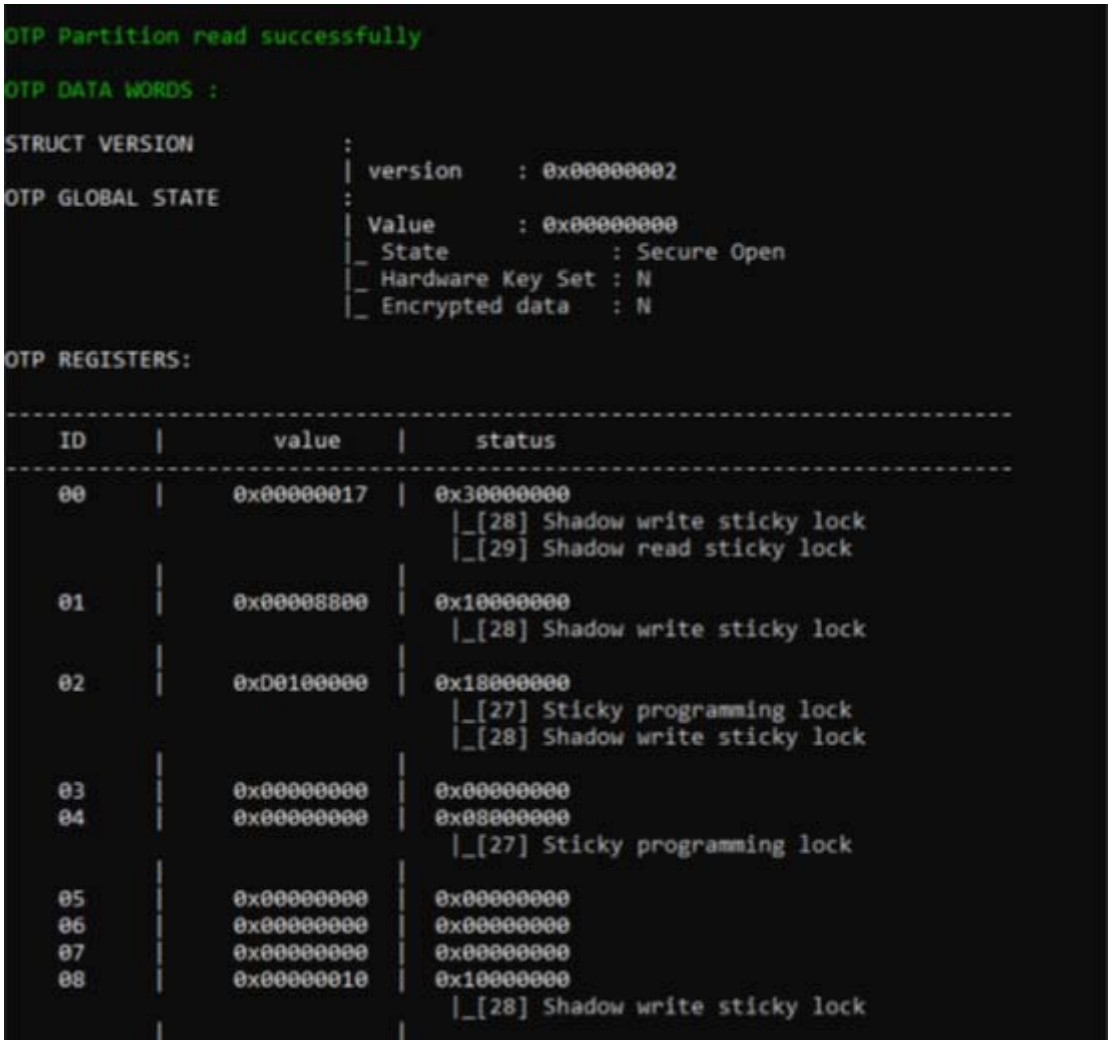


Fig. 3.137: OTP write command for OTP structure v2

-detach

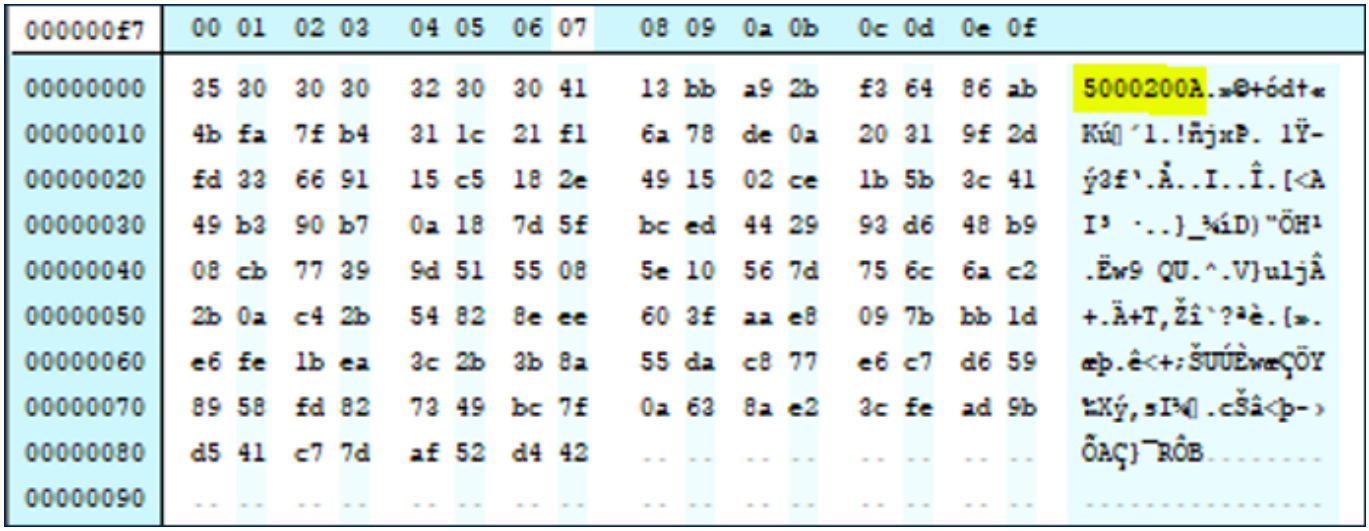
Description: This command allows the user to send detach command to USB DFU.

Syntax: -detach

-gc

Description: This command can be used to read the chip certificate and save the data to a binary file.

Syntax: -gc <Output_Path>



000000f7	00 01	02 03	04 05	06 07	08 09	0a 0b	0c 0d	0e 0f	
00000000	35 30	30 30	32 30	30 41	13 bb	a9 2b	f3 64	86 ab	5000200A.»©+ódt«
00000010	4b fa	7f b4	31 1c	21 f1	6a 78	de 0a	20 31	9f 2d	Kú´1.!ñjxÞ. 1Ÿ-
00000020	fd 33	66 91	15 c5	18 2e	49 15	02 ce	1b 5b	3c 41	ý3f‘.Å..I..Î.[<A
00000030	49 b3	90 b7	0a 18	7d 5f	bc ed	44 29	93 d6	48 b9	I³ ·..}_¼íD)"ÖH¹
00000040	08 cb	77 39	9d 51	55 08	5e 10	56 7d	75 6c	6a c2	.Ëw9 QU.^.V}uljÂ
00000050	2b 0a	c4 2b	54 82	8e ee	60 3f	aa e8	09 7b	bb 1d	+.Ä+T‚Žî`?ªè.{».
00000060	e6 fe	1b ea	3c 2b	3b 8a	55 da	c8 77	e6 c7	d6 59	æþ.ê<+;ŠUÚÈwæÇÖY
00000070	89 58	fd 82	73 49	bc 7f	0a 63	8a e2	3c fe	ad 9b	‰Xý‚sI¼.cŠâ<þ-›
00000080	d5 41	c7 7d	af 52	d4 42	.. ..	.. ..	.. ..	.. ..	ÕAÇ}¯RÔB........
00000090	.. ..	.. ..	.. ..	.. ..	.. ..	.. ..	.. ..	.. ..	....

Fig. 3.138: Get certificate output file

> **Note**
>
> This command can be used only if a specific firmware (tfa-ssp) is installed, as it is the basis to retrieve the stored certificate.

-wb

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

Syntax: -wb blob.bin

-ssp

Description: Program an SSP file.

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

```
Requesting Chip Certificate...

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62000000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x62062FD8
P11 lib initialization Success!

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Writing blob


Blob successfully written

Start operation achieved successfully
Send detach command
Detach command executed
SSP file out.ssp Install Operation Success
```

Fig. 3.139: SSP successfully installed

Note

All SSP traces are shown on the output console. If there is any faulty input the SSP process is aborted, and an error message is displayed to indicate the root cause of the issue.

4 Supported Interfaces

4.1 Introduction

This section presents the probes supported by STM32CubeProgrammer and the main connection options available for programming and debugging STM32 devices.

4.2 Debug Interfaces

4.3 Bootloader Interfaces

4.4 Additional article

STM32 microcontroller system memory boot mode

4.4.1 ST-Link connection

ST-LINK driver

To connect to an STM32 device through a debug interface using ST-LINK/V2, ST-LINK/V2-1, ST-LINK-V3, or ST-LINKV3Power, install the ST-LINK driver by running the `stlink_winusb_install.bat` file. This driver is provided with the release package and can be found under the `Driver/stsw-link009_v3` folder.

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
```

When using ST-LINK or J-Link probes, or USB DFU to connect to a target, copy the rules files located under the `Driver/rules` folder into `/etc/udev/rules.d/` on Ubuntu:

```
sudo cp *.* /etc/udev/rules.d
```

Note

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

Docker and Ubuntu

When using USB DFU in a Docker context, it is often necessary to have the device rules installed on the host machine and create the container using the following flags:

```
-v /dev:/dev --device /dev:/dev
--net=host
```

The first flag gives the container access to the devices on the host machine. The second lets the container communicate udev host events.

VirtualBox

For optimal performance when using STM32CubeProgrammer in a VirtualBox VM context, it is recommended to switch the USB controller to the USB 3.0 (xHCI) controller in Settings > USB.

Fedora

To install the libusb package, execute the following command:

```
sudo dnf install libusb1
```

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

DFU driver

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

If the DFUSE driver is already installed on your machine, first uninstall it, then reboot the machine and run the previously mentioned `.bat` file. When uninstalling, check the Delete the driver software for this device option to avoid reinstalling the old driver the next time a board is plugged in.



Fig. 4.1: Deleting the old driver software



Fig. 4.2: STM32 DFU device with DfuSe driver

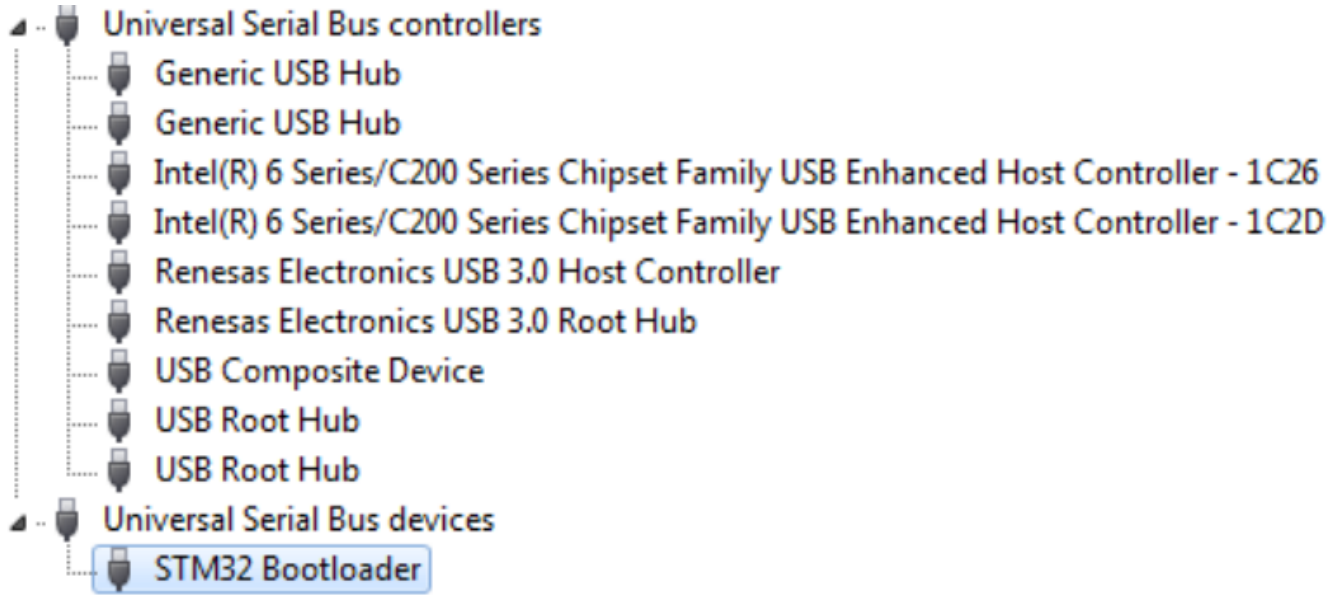


Fig. 4.3: STM32 DFU device with STM32CubeProgrammer driver

Note

When using the USB DFU interface or the ST-LINK interface on a Windows 7 PC, ensure that all USB 3.0 controller drivers are up to date. Older driver versions can contain bugs that prevent access or cause connection problems with USB devices.

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 an ST-LINK or J-Link debug probe, or the bootloader interface over UART, USB, SPI, CAN, or I2C.

With the refresh button, the user can check the available interfaces connected to the PC. If this button is pressed while the ST-LINK or J-Link interface is selected, the tool checks the connected ST-LINK or J-Link probes and lists

them in the **Serial numbers** combo box. If the UART interface is selected, it checks the available communication ports of the PC and lists them in the **Port** combo box. If the USB interface is selected, it checks the USB devices in DFU mode connected to the PC and lists them in the **Port** combo box. Each interface has its own settings to configure before connection.

ST-LINK settings

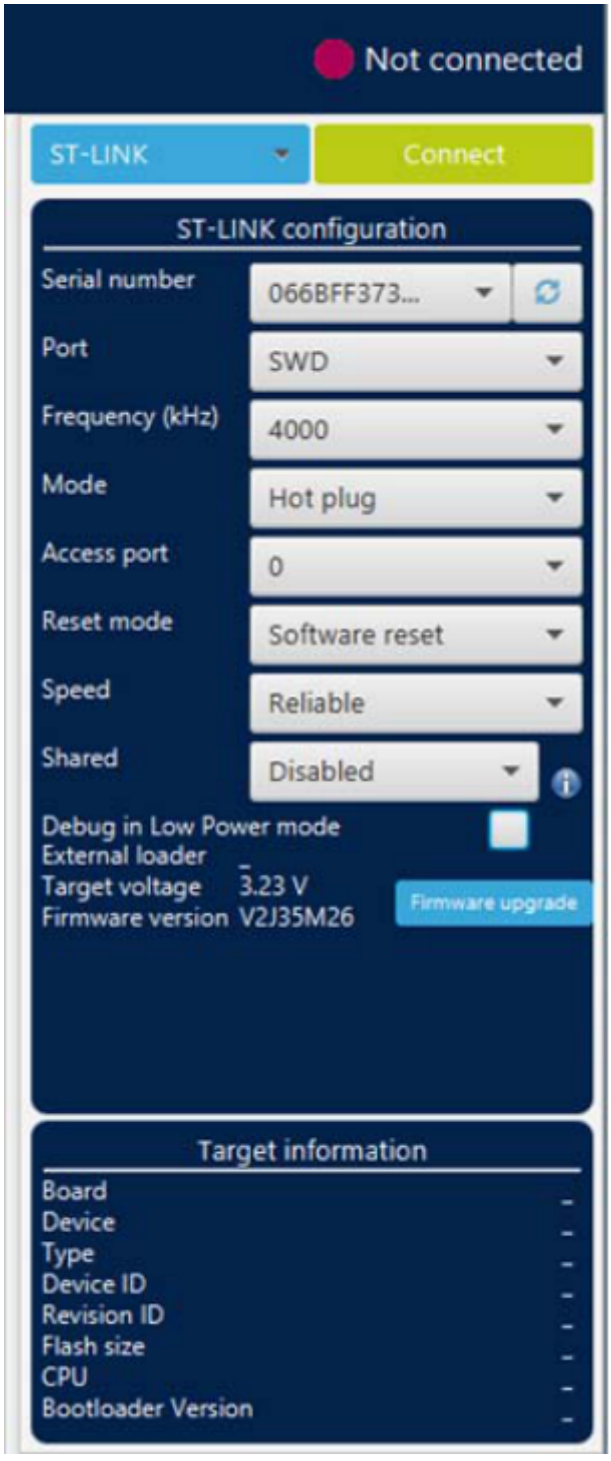


Fig. 4.4: ST-LINK configuration panel

- **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.

> **Note**
>
> STM32CubeProgrammer has been tested with up to 34 ST-LINK probes connected simultaneously to the same host, on both Linux and Windows machines.

- **Port:** ST-LINK probe supports two debug protocols, JTAG and SWD.

> **Note**
>
> JTAG is not available on all embedded ST-LINK interfaces on STM32 Nucleo or Discovery boards.

- **Frequency:** the JTAG or SWD clock frequency.
- **Access port:** selects the access port to connect to. Most 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 reset type 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, for example, when the target contains code that disables the JTAG or SWD pins.
  - **Hot plug:** enables connection to the target without a halt or reset. This is useful for updating RAM addresses or IP registers while the application is running.
  - **Power down:** used to put the target in debug mode, even if the application has not started since target power-up. The hardware reset signal must be connected between ST-LINK and the target. This feature may not be fully effective on some boards such as MB1360, MB1319, MB1361, and MB1355 with STMPS2141 power switch.
  - **hwRstPulse:** the tool generates a reset pulse, then connects to the target. This connection mode does not prevent application launch before connection. It is used on some devices where under-reset mode is not available, such as STM32WB0x and STM32WL33.
- **Reset mode:**
  - **Software system reset:** resets all STM32 components except the Debug block through the Cortex-M application interrupt and reset control register (AIRCR).
  - **Hardware reset:** resets the STM32 device through the `nRST` pin. The RESET pin of the JTAG connector (pin 15) must be connected to the device reset pin.
  - **Core reset:** resets only the Cortex-M through the AIRCR.
- **Speed (Cortex-M33 only):**
  - **Reliable:** allows the user to connect in slow mode.
  - **Fast:** allows the user to connect in fast mode.
- **Shared:** enables shared mode allowing connection of two or more instances of STM32CubeProgrammer or another debugger to the same ST-LINK probe.
- **Debug in low-power mode (STM32U5/WB/L4 series only):** sets the bits in `DBGMCU_CR` to `1`.
- **External loader:** displays the name of the external memory loader selected in the External loaders panel accessible from the main menu (Hamburger menu).
- **Target voltage:** target voltage is measured and displayed.
- **Firmware version:** displays the ST-LINK firmware version. The firmware upgrade button allows you to upgrade the ST-LINK firmware.

Log messages

- **Starting Automatic Mode...**

  Indicates that the system successfully entered the automatic process.

- **More than one ST-LINK probe detected! Keep only one ST-LINK probe!**

  Automatic mode cannot be used if more than one ST-LINK probe is connected to the computer when using JTAG or SWD interfaces. A message is displayed asking the user to keep only one ST-LINK probe connected to continue using this mode.

- **More than one ST-LINK Bridge detected! Keep only one ST-LINK Bridge!**

  Automatic mode cannot be used if more than one ST-LINK bridge is connected to the computer when using the SPI, CAN, or I2C bootloader interfaces. A message is displayed asking the user to keep only one ST-LINK bridge connected to continue using this mode.

- **More than one ST-LINK USB DFU detected! Keep only one USB DFU!**

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/JLINK/SPI/CAN/I2C).

Syntax:

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

Common parameters

- `port=<Portname>`: Interface identifier, for example `COMx` on Windows, `/dev/ttySx` on Linux, `usbx` for USB, or `SPI`, `I2C`, and `CAN` for the corresponding interfaces.
- `[noinit=<noinit_bit>]`: Sets No Init bits, value in `{0, 1}`, default `0`. `noinit=1` can be used if a previous connection is already active.

ST-LINK options

CLI location by platform:

- **macOS x86_64**: `STM32CubeProgrammer.app/Contents/MacOs/bin/STM32_Programmer_CLI`
- **macOS aarch64**: `STM32CubeProgrammer.app/Contents/Resources/bin/STM32_Programmer_CLI`
- **Windows**: `..\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_Programmer_CLI.exe`
- **Linux**: `../STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin/STM32_Programmer_CLI`

Supported connection options:

- `[freq=<frequency>]`: Frequency in kHz used for connection. Default is `4000` kHz for SWD and `9000` kHz for JTAG.
- `[reset=<mode>]`: Reset mode. Possible values are `SWrst`, `HWrst`, and `Crst`. The default value is `SWrst`. When using `UR` connection mode, the reset mode is `HWrst`.
  - `HWrst`: Hardware reset. Performed by the ST-LINK, which physically drives the MCU `NRST` pin. The ST-LINK `NRST` pin must be connected to the MCU.
  - `SWrst`: Software reset through Cortex registers.
  - `Crst`: Core reset through Cortex registers.

Note

Entered frequency values are rounded to the values supported by the ST-LINK probe.

- `[index=<index>]`: Debug probe index. Default value is `0`.
- `[sn=<serialNumber>]`: Serial number of the debug probe. Use this when you need to connect to a specific ST-LINK probe. Do not use this option together with `index` in the same connect command.
- `[mode=<mode>]`: Connection mode. Values are `NORMAL`, `UR`, `HOTPLUG`, `POWERDOWN`, and `hwRstPulse`. Default is `NORMAL`.
  - `NORMAL`: The target is reset, then halted. The reset type is selected using the Reset mode option.
  - `UR`: Connect Under Reset mode enables connection to the target using a reset vector catch before executing any instruction. This is useful, for example, when the target contains code that disables the JTAG or SWD pins.
  - `HOTPLUG`: Hot Plug mode enables connection to the target without a halt or reset. This is useful when updating RAM addresses or IP registers while the application is running.
  - `POWERDOWN`: Puts the target in debug mode even if the application has not started since power-up. The hardware reset signal must be connected between ST-LINK and the target. This feature might not be fully effective on boards such as MB1360, MB1319, MB1361, and MB1355 with STMPS2141 power switch.
  - `hwRstPulse`: The tool generates a reset pulse and then connects to the target. This mode does not prevent application launch before connection. It is used on devices where under-reset mode is not available, such as STM32WB0x and STM32WL33.
- `[ap=<accessPort>]`: Access port index. Default value is `0`.
- `[speed=]`: Connection speed. Default is `Reliable`. Available only for Cortex-M33.
  - `Reliable`: Allows the user to connect in slow mode.
  - `Fast`: Allows the user to connect in fast mode.
- `[shared]`: Enables shared mode, allowing connection of two or more instances of STM32CubeProgrammer or another debugger to the same ST-LINK probe.
- `[tcpport=<Port>]`: Selects the TCP port to connect to an ST-LINK server. The `shared` option must be selected. The default value is `7184`.

4.4.2 J-Link connection

Segger J-link/Flasher library

The Segger J-link/Flasher library is not mandatory for using STM32CubeProgrammer with other interfaces. If you want to build your own package without using it, the library can be deleted without impact on STM32CubeProgrammer.

Linux install

When connecting to an STM32 device over USB on Linux, install the libusb development package first:

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

If you use ST-LINK, J-Link, or USB DFU, copy the rule files from the `Driver/rules` directory into `/etc/udev/rules.d/` on Ubuntu so the host can access the device correctly.

```
sudo cp *.* /etc/udev/rules.d
```

Note

STM32CubeProgrammer requires libusb version 1.0.12 or later.

Docker and Ubuntu

For USB DFU inside Docker, install the device rules on the host machine and start the container with device access enabled so it can reach the host USB devices and receive udev events.

```
docker run -v /dev:/dev --device /dev:/dev --net=host
```

VirtualBox

For better behavior in a VirtualBox virtual machine, switch the USB controller to USB 3.0 (xHCI) in the VM USB settings.

Fedora

On Fedora, install libusb with:

```
sudo dnf install libusb1
```

To install STM32CubeProgrammer on Linux, download and extract the Linux package from the STM32CubeProgrammer website, then run the Linux setup program to complete installation. On Ubuntu 20, the application icon may be disabled by default; if needed, right-click it and choose Allow launching.

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/J-LINK/SPI/CAN/I2C).

J-Link options

- `[dLPM / LPM]`: Disable or enable debug in Low power mode. The default configuration is enabled for supported devices such as STM32U5, STM32WB, and STM32L4 series.
- `[getAuthID]`: Get device identification for STM32U5 series only. This is a 32-bit device-specific quantity that can be read through the JTAG port. This information is used to derive the expected OEM password keys to unlock the device. This command is not applicable when RDP level = 0.
- `[reset=<mode>]`: Reset mode. Possible values are `SWrst`, `HWrst`, and `Crst`. The default value is `SWrst`. When using `UR` connection mode, the reset mode is `HWrst`.
  - `HWrst`: Hardware reset performed by the ST-LINK, which physically drives the MCU `NRST` pin. The ST-LINK `NRST` pin must be connected to the MCU.
  - `SWrst`: Software reset through Cortex registers.
  - `Crst`: Core reset through Cortex registers.
- `[sn=<serialNumber>]`: Serial number of the debug probe. Use this option to connect to a specific ST-LINK probe when you know its serial number.
- `[mode=<mode>]`: Connection mode. Values are `NORMAL`, `UR`, or `HOTPLUG`. The default value is `NORMAL`.
  - `NORMAL`: The target is reset, then halted. The reset type is selected using the Reset mode option.
  - `UR`: Connect Under Reset mode enables connection to the target using a reset vector catch before executing any instructions. This is useful, for example, when the target contains code that disables the JTAG or SWD pins.

  - HOTPLUG: Hot Plug mode enables connection to the target without halt or reset. This is useful when updating RAM addresses or IP registers while the application is running.

- [ap=<accessPort>]: Access port index. The default access port value is 0.
- [speed=]: Connection speed. The default value is Reliable. This option is available only for Cortex-M33.
  - Reliable: Allows the user to connect in slow mode.
  - Fast: Allows the user to connect in fast mode.

Example using J-Link debug port

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

Note

Only Flash and Option Bytes (OB) programming are supported via J-Link. Other features are not supported, e.g RDP regression with password, Debug Authentication, Security features. There are other parameters used in connection with the J-Link debug port 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 0x480.

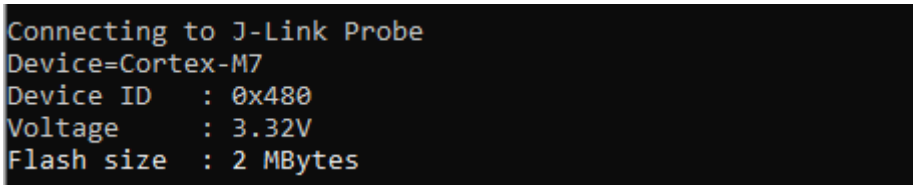


Fig. 4.5: Connect operation using J-Link debug port

The corresponding command line for this example is -c port=JLINK ap=0.

In the connect command (-c port=JLINK ap=0):

- The port parameter is mandatory.
- The default frequency value is 4000.

Note

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

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 an ST-LINK or J-Link debug probe, or the bootloader interface over UART, USB, SPI, CAN, or I2C.

With the refresh button, the user can check the interfaces available on the PC. If this button is pressed while the ST-LINK or J-Link interface is selected, the tool checks the connected ST-LINK or J-Link probes and lists them in the **Serial numbers** combo box. If the UART interface is selected, it checks the available communication ports on the PC and lists them in the **Port** combo box. If the USB interface is selected, it checks the USB devices connected in DFU mode and lists them in the **Port** combo box. Each interface has its own settings that must be configured before connection.

J-Link settings

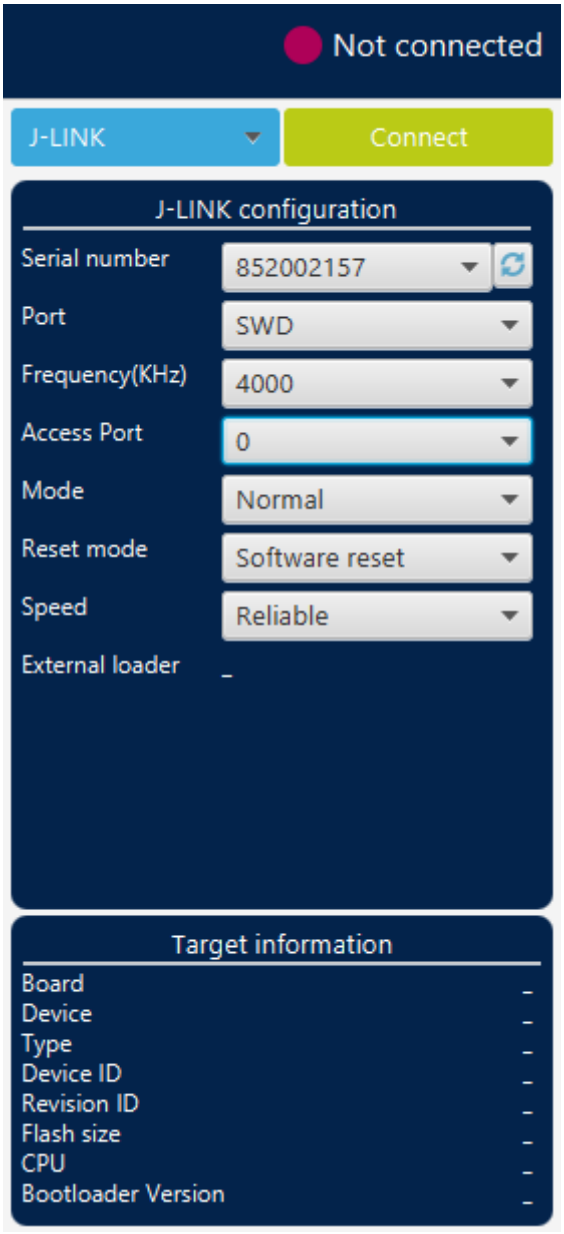


Fig. 4.6: J-Link configuration panel

- **Serial number:** this field contains the serial numbers of all connected J-Link probes. The user can choose one of them based on its serial number.
- **Port:** J-Link probe supports only the SWD debug protocol.
- **Frequency:** SWD clock frequency. Only `4000 kHz` is available.
- **Access port:** selects the access port to connect to. Most STM32 devices have only one access port, access port 0.
- **Mode:**
  - **Normal:** the target is reset, then halted. This is selected using the **Reset mode** option.
  - **Connect under reset:** enables connection to the target using a reset vector catch before executing any instructions. This is useful, for example, when the target contains code that disables the SWD pins.

  - **Hot plug:** enables connection to the target without a halt or reset. This is useful when updating RAM addresses or IP registers while the application is running.

- **Reset mode:**
  - **Software system reset:** resets all STM32 components except the Debug block through the Cortex-M application interrupt and reset control register (AIRCR).
  - **Hardware reset:** resets the STM32 device through the `nRST` pin. The RESET pin of the SWD connector (pin 15) must be connected to the device reset pin.
  - **Core reset:** resets only the Cortex-M through the reset control register.
- **Speed (Cortex-M33 only):**
  - **Reliable:** allows the user to connect in slow mode.
  - **Fast:** allows the user to connect in fast mode.
- **External loader:** displays the name of the external memory loader selected in the External loaders panel accessible from the main menu (Hamburger menu).

Additional articles

For more information about the Segger J-link/Flasher library, please refer to the Segger website.

4.4.3 USB connection

Linux install

When using a USB connection to an STM32 device on Linux, first install the libusb 1.0 development package:

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

For ST-LINK, J-Link, or USB DFU connections, copy the udev rules from the `Driver/rules` folder into `/etc/udev/rules.d/` on Ubuntu:

```
sudo cp *.* /etc/udev/rules.d
```

> **Note**
>
> STM32CubeProgrammer requires libusb version 1.0.12 or newer.

Docker and Ubuntu

When USB DFU is used inside Docker, it is often necessary to install the device rules on the host machine and start the container with access to host devices and host udev events:

```
docker run -v /dev:/dev --device /dev:/dev --net=host
```

VirtualBox

For better behavior in a VirtualBox virtual machine, switch the USB controller to USB 3.0 (xHCI) in the VM settings under USB.

Fedora

On Fedora, install libusb with:

```
sudo dnf install libusb1
```

To install STM32CubeProgrammer on Linux, download and extract the Linux package from the STM32CubeProg-Linux delivery on the website, then run `SetupSTM32CubeProgrammer-vx.y.z.linux` to complete setup. On Ubuntu 20, the STM32CubeProgrammer icon may be disabled by default; if needed, right-click it and choose Allow launching.

DFU driver

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

If the DFUSE driver is already installed on your machine, first uninstall it, then reboot the machine and run the previously mentioned `.bat` file. When uninstalling, select the Delete the driver software for this device option to avoid reinstalling the old driver when a board is plugged in later.



Fig. 4.7: Deleting the old driver software

Note

When using the USB DFU interface or the ST-LINK interface on a Windows 7 PC, ensure that all USB 3.0 controller drivers are updated. Older driver versions can contain bugs that prevent access or cause connection problems with USB devices.

USB settings

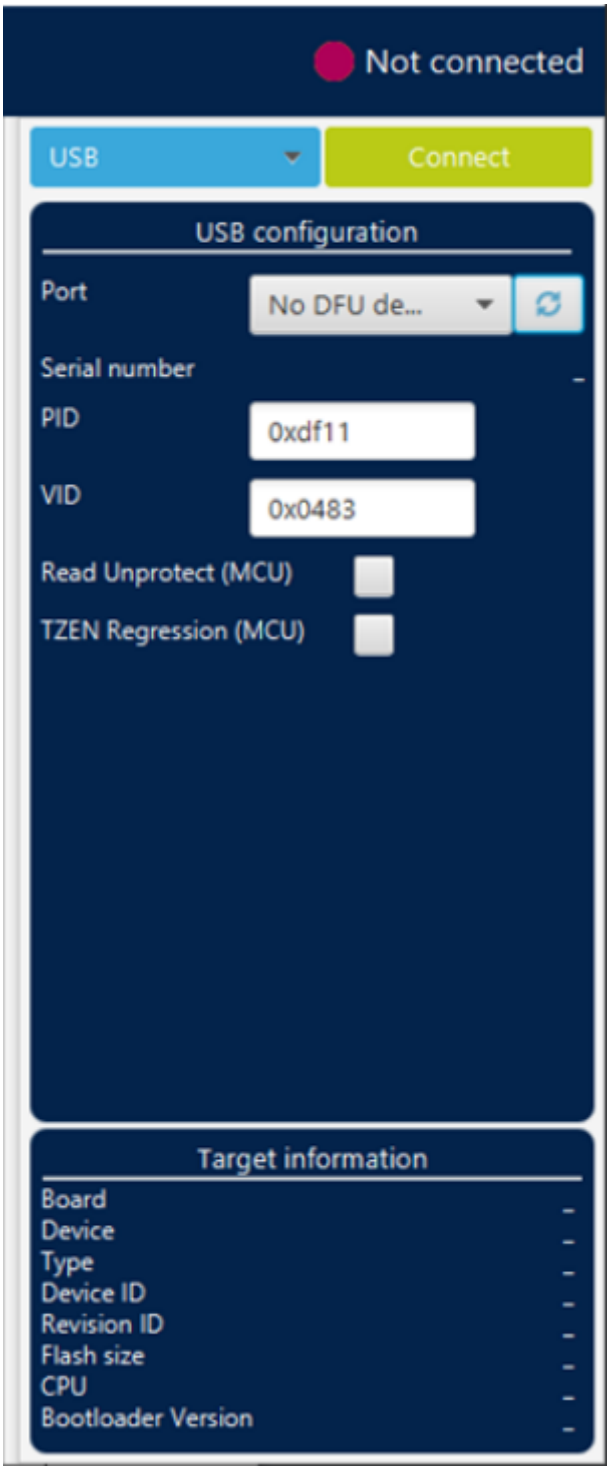


Fig. 4.8: USB configuration panel

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

Log messages

- **Starting Automatic Mode...**

  Indicates that the system successfully entered the automatic process.

- **More than one ST-LINK probe detected! Keep only one ST-LINK probe!**

  Automatic mode cannot be used if more than one ST-LINK probe is connected to the computer when using JTAG or SWD interfaces. A message is displayed asking the user to keep only one ST-LINK probe connected to continue using this mode.

- **More than one ST-LINK Bridge detected! Keep only one ST-LINK Bridge!**

  Automatic mode cannot be used if more than one ST-LINK bridge is connected to the computer when using SPI, CAN, or I2C bootloader interfaces. A message is displayed asking the user to keep only one ST-LINK bridge connected to continue using this mode.

- **More than one ST-LINK USB DFU detected! Keep only one USB DFU!**

  Automatic mode cannot be used if more than one USB DFU device is connected to the computer when using the USB bootloader interface. A message is displayed asking the user to keep only one USB DFU connected to continue using this mode.

- **More UART ports detected than last connection!**

  During the first connection, automatic mode calculates the number of available serial ports and uses that as a reference so it can correctly detect that only one UART port is used for each STM32 device.

- **Please disconnect device and connect the next...**

  If the system finishes the first process, whatever the result, disconnect the current device to prepare the next device connection.

- **Waiting for device...**

  Once the connection to the previous device is correctly lost, the system keeps searching for a new device.

- **Automatic Mode is stopped.**

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

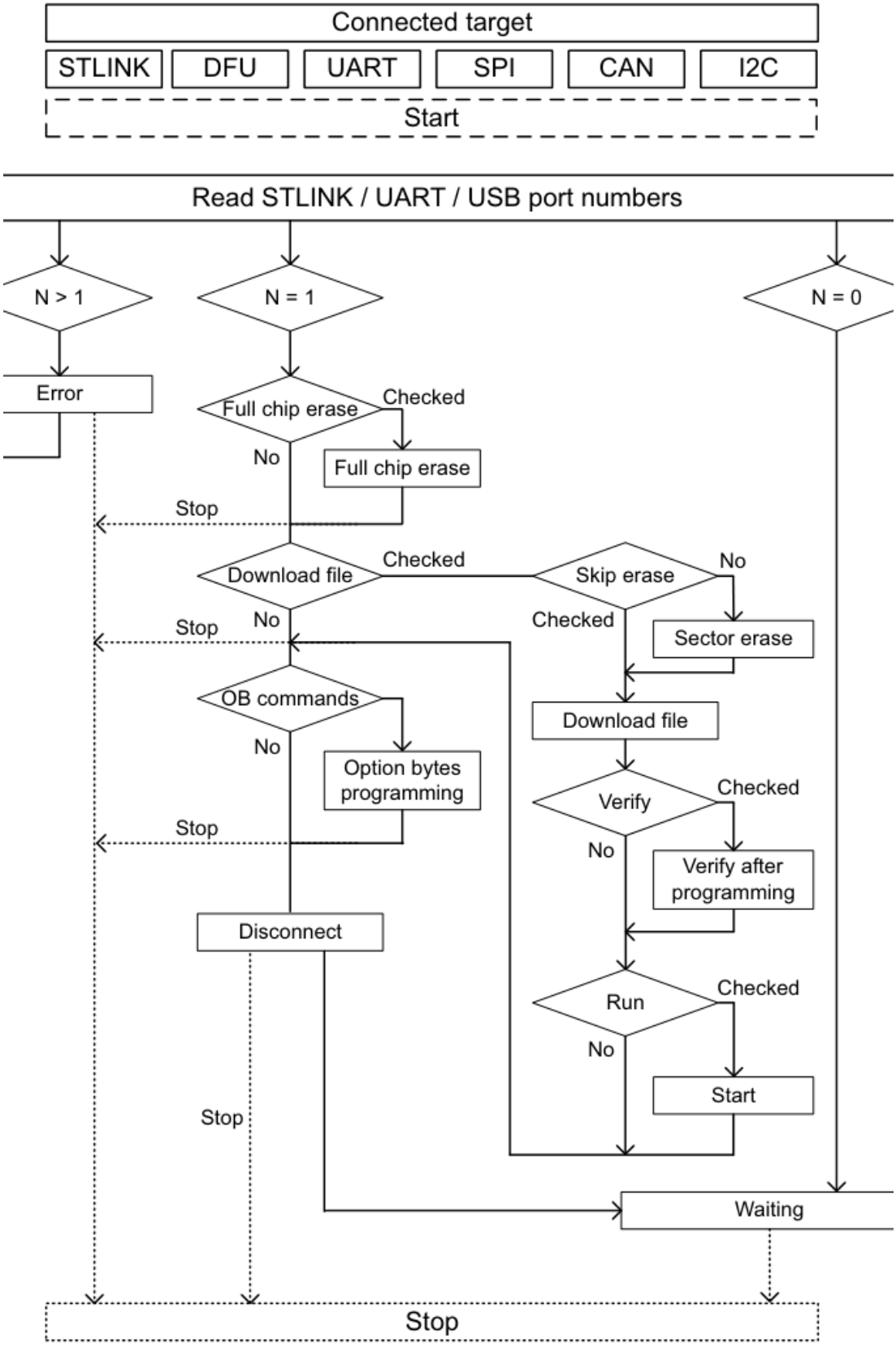


Fig. 4.9: Algorithm

In application programming (IAP/USBx)

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

> **Note**
>
> Option byte, memory edition, and sector erase are not available with IAP/USBx.

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

DFU IAP/USBx with custom PID and VID

STM32CubeProgrammer DFU IAP/USBx supports not only ST product IDs when connecting through DFU IAP.

Before starting the DFU connection with a new product ID, sign your USB driver. For more information, visit woshub.com.

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

To connect through the new USB DFU, follow this sequence:

1. Modify the default product ID.
2. Modify the default vendor ID.
3. Click the refresh button, then click the connect button.

> **Note**
>
> If the user does not enter a PID or VID value, STM32CubeProgrammer uses the default ST product values: `PID = 0xDF11` and `VID = 0x0483`.

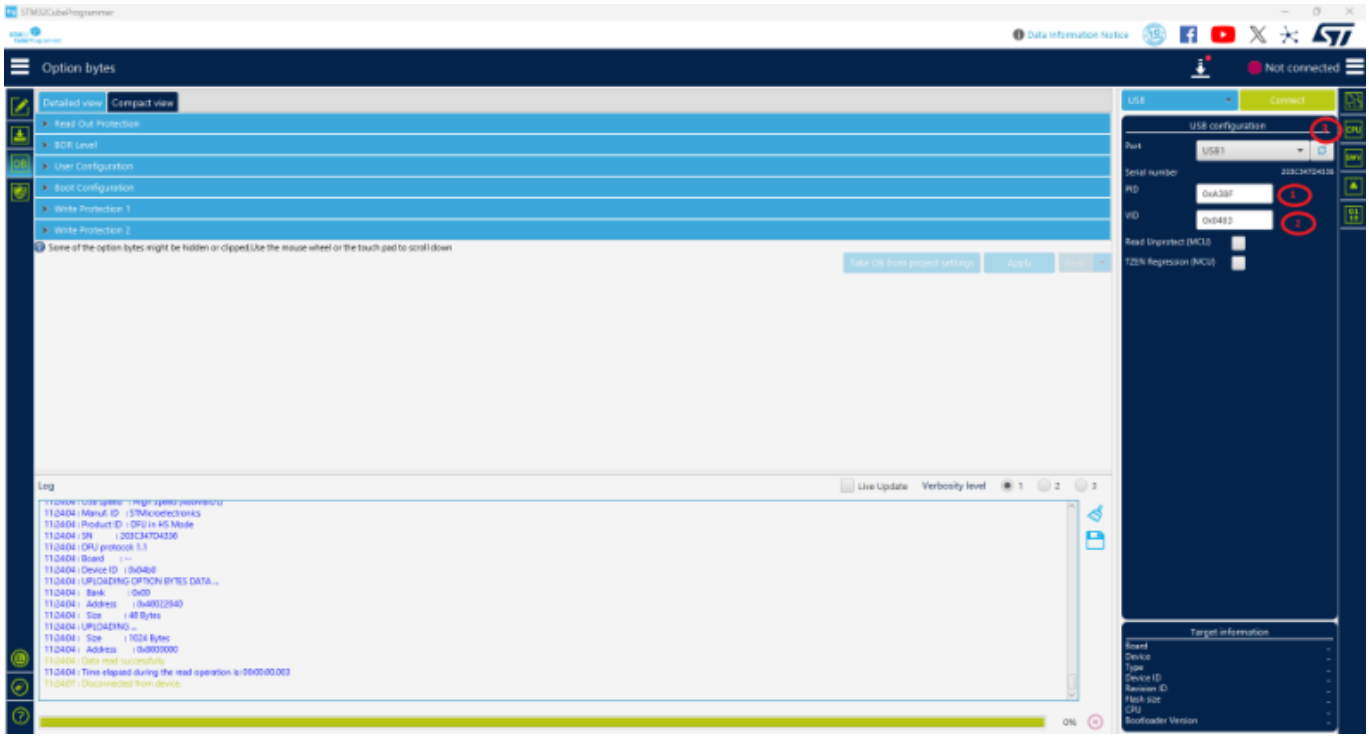


Fig. 4.10: Connect via USB DFU panel

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/JLINK/SPI/CAN/I2C).

Syntax:

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

- `port=<Portname>`: Interface identifier, for example `COMx` on Windows, `/dev/ttySx` on Linux, `usbx` for USB interface, and `SPI`, `I2C`, or `CAN` for the corresponding interfaces.

USB options

The connection under the DFU interface supports two options: product ID and vendor ID. The default values are `PID = 0xDF11` and `VID = 0x0483`.

Example using USB

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

```
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
```

Fig. 4.11: Connect operation using USB

Note

When using a USB interface, all configuration parameters such as baud rate, parity, data bits, frequency, and index are ignored. To connect using a UART interface, the port configuration, including baudrate, parity, data bits, stop bits, and flow-control, must use a valid combination for the target device.

Example using DFU IAP/USBx options

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

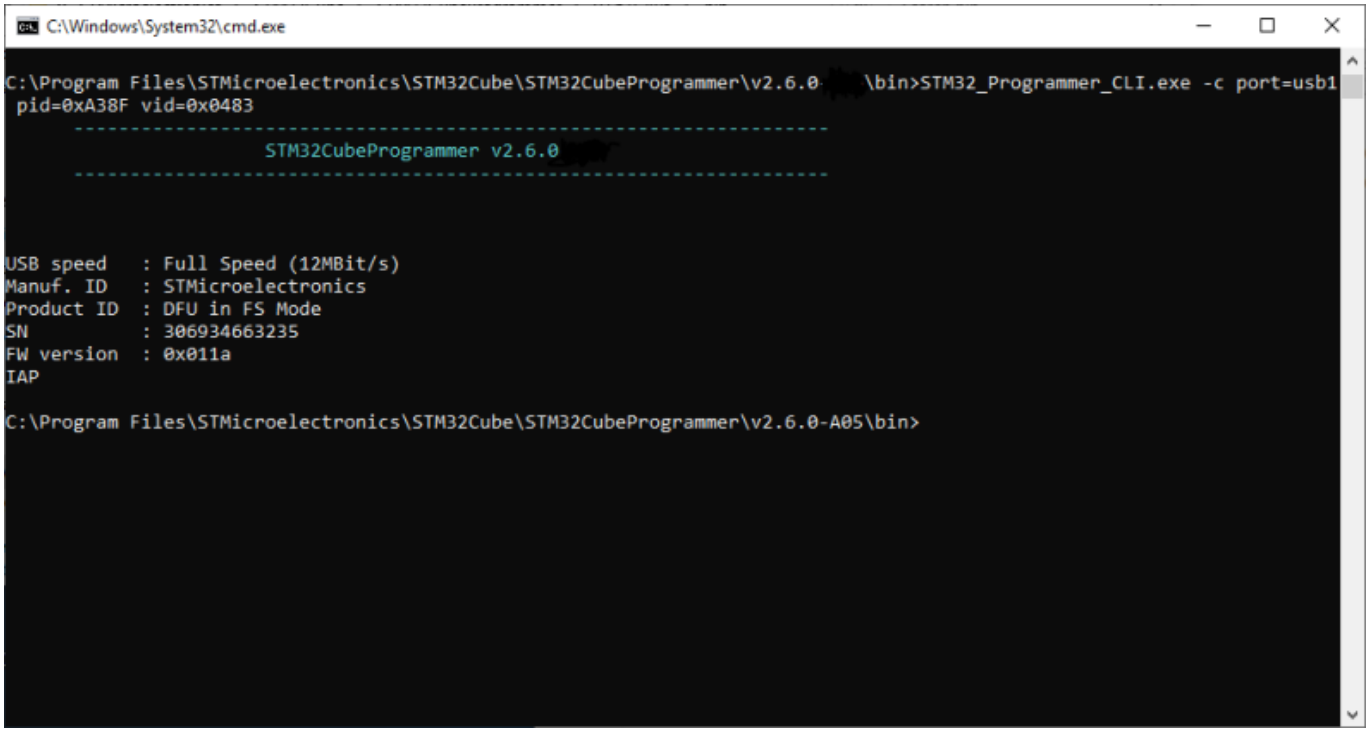


Fig. 4.12: Connect operation using USB DFU options

Note

The default value of product ID and vendor ID corresponds to ST product values: `PID = 0xDF11` and `VID = 0x0483`.

4.4.4 I2C connection

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 an ST-LINK or J-Link debug probe, or the bootloader interface over UART, USB, SPI, CAN, or I2C.

With the refresh button, the user can check the available interfaces connected to the PC. If this button is pressed while the ST-LINK or J-Link interface is selected, the tool checks the connected ST-LINK or J-Link probes and lists them in the **Serial numbers** combo box. If the UART interface is selected, it checks the available communication ports of the PC and lists them in the **Port** combo box. If the USB interface is selected, it checks the USB devices in DFU mode connected to the PC and lists them in the **Port** combo box. Each interface has its own settings to configure before connection.

I2C settings

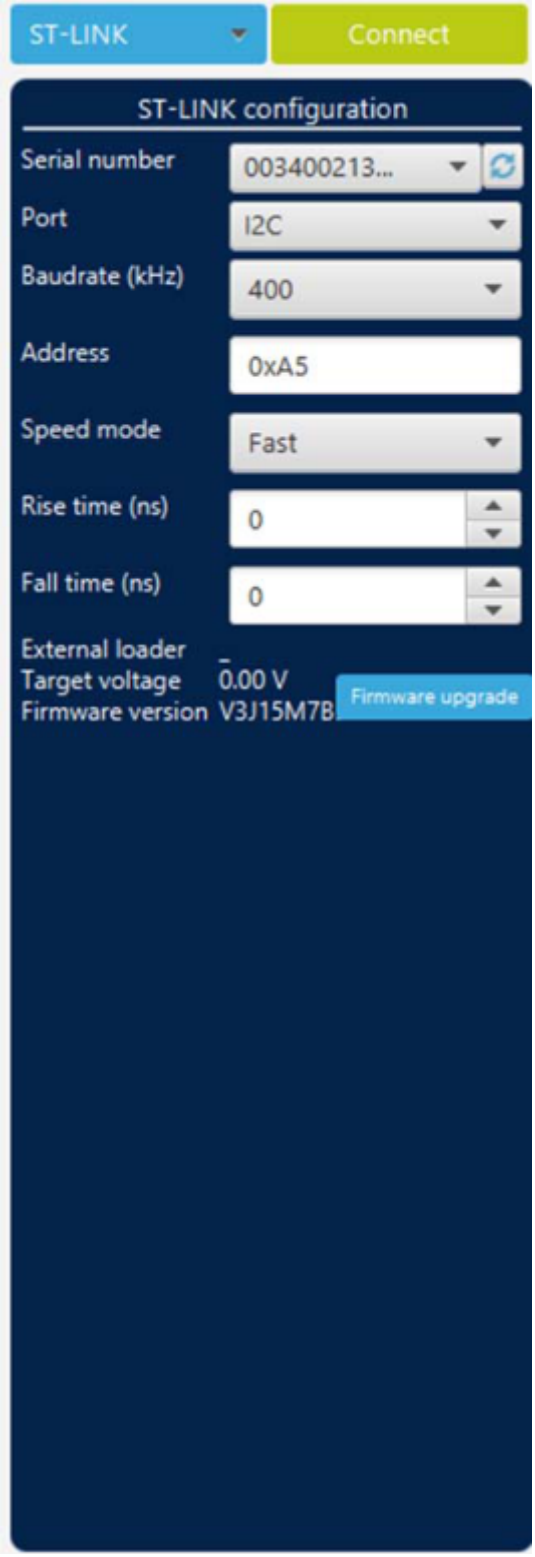


Fig. 4.13: I2C configuration panel

- **Serial number:** this field contains the serial numbers of all connected ST-LINK-V3 probes when using the I2C bootloader.
- **Port:** selects the I2C devices connected to the PC. You can use the refresh button to recheck the available

devices.

- **Baudrate:** selects the I2C baud rate.
- **Address:** adds the address of the slave bootloader in hexadecimal format.
- **Speed mode:** selects the transmission speed mode, Standard or Fast.
- **Rise time:** choose values according to the speed mode, `0-1000` for Standard and `0-300` for Fast.
- **Fall time:** choose values according to the speed mode, `0-300` for Standard and `0-400` for Fast.

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/JLINK/SPI/CAN/I2C).

Syntax:

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

- `port=<Portname>`: Interface identifier, for example `COMx` on Windows, `/dev/ttySx` on Linux, `usbx` for USB interface, and `SPI`, `I2C`, or `CAN` for the corresponding interfaces.
- `[noinit=<noinit_bit>]`: Sets No Init bits, value in `{0, 1}`, default `0`. `noinit=1` can be used if a previous connection is already active.

I2C options

- `[add=<ownadd>]`: Slave address in hexadecimal format.

> **Note**
>
> The I2C address option must always be provided, otherwise the connection is not established.

Example using I2C

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

In the connect command:

- The `add` parameter changes from one device to another. Refer to AN2606 to determine the correct value. In this case, the STM32F42xxx bootloader address is `0x38`.
- The `br` baudrate parameter depends directly on the `sm` speed mode parameter. For example, if `sm = standard`, the baudrate does not support the value `400`.



Fig. 4.14: Connect operation using I2C port

> **Note**
>
> For each I2C connection operation, the address parameter is mandatory. Not all devices implement this feature. Make sure that the target device supports an I2C bootloader.

There are other parameters used with the I2C port connection that have default values, while some others require specific values. See the help menu of the tool for more information.

4.4.5 UART connection

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 an ST-LINK or J-Link debug probe, or the bootloader interface over UART, USB, SPI, CAN, or I2C.

With the refresh button, the user can check the available interfaces connected to the PC. If this button is pressed while the ST-LINK or J-Link interface is selected, the tool checks the connected ST-LINK or J-Link probes and lists them in the **Serial numbers** combo box. If the UART interface is selected, it checks the available communication ports of the PC and lists them in the **Port** combo box. If the USB interface is selected, it checks the USB devices in DFU mode connected to the PC and lists them in the **Port** combo box. Each interface has its own settings to configure before connection.

UART settings

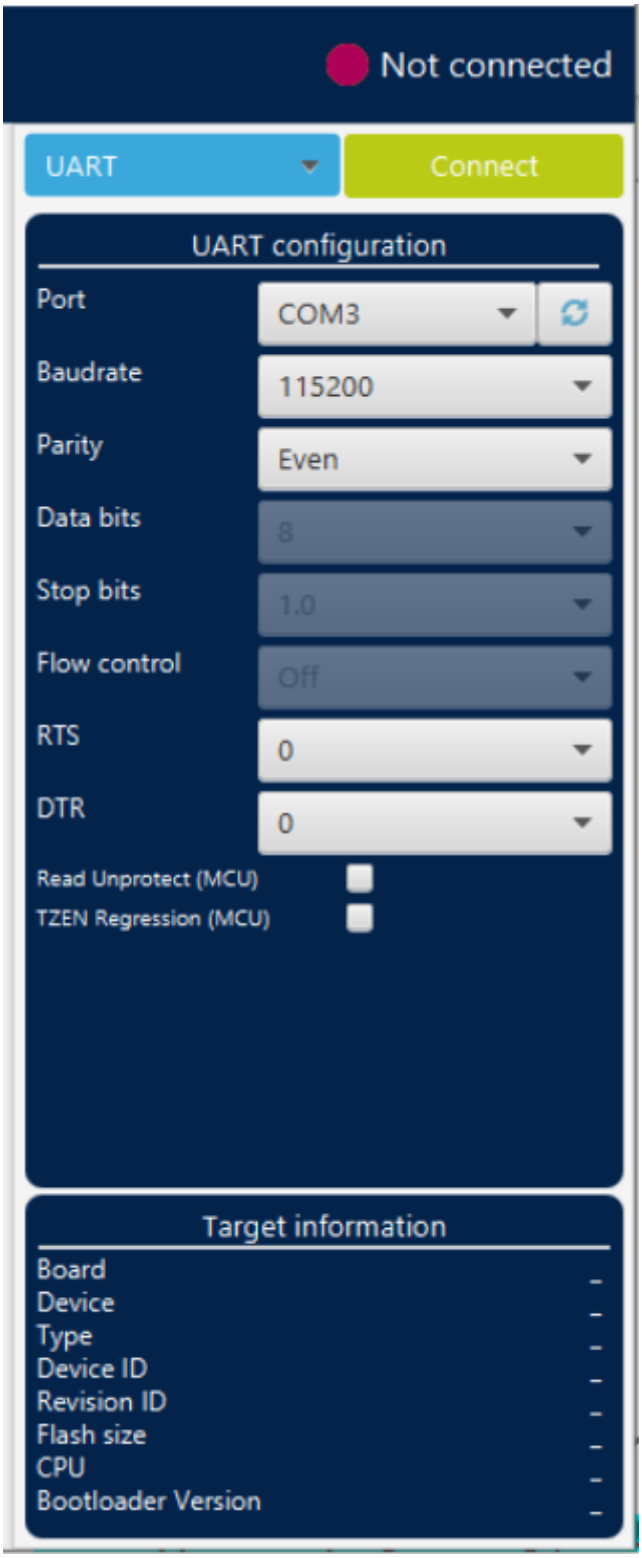


Fig. 4.15: UART configuration panel

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

> **Note**
>
> The STM32 must boot in bootloader mode using boot pins and or option bits. Refer to AN2606, STM32 microcontroller system memory boot mode, available on www.st.com , for more information about the STM32 bootloader.

- **Baudrate:** selects the UART baud rate.
- **Parity:** selects the parity, even, odd, or none. It must be **even** for STM32 devices.
- **Data bits:** must always be `8`. Only 8-bit data is supported by STM32.
- **Stop bits:** must always be `1`. Only 1-bit stop is supported by STM32.
- **Flow control:** must always be off.
- **RTS (request to send):** sets the COM RTS pin to either high or low level.
- **DTR (data terminal ready):** sets the COM DTR pin to either high or low level.

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/JLINK/SPI/CAN/I2C).

Syntax:

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

- `port=<Portname>`: Interface identifier, for example `COMx` on Windows, `/dev/ttySx` on Linux, `usbx` for USB interface, and `SPI`, `I2C`, or `CAN` for the corresponding interfaces.
- `[noinit=<noinit_bit>]`: Sets No Init bits, value in `{0, 1}`, default `0`. `noinit=1` can be used if a previous connection is already active.

Using UART

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

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

Fig. 4.16: Connect operation using RS232

STM32CubeProgrammer provides the possibility to configure RTS and DTR pins:

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

Example:

```
STM32_Programmer_CLI.exe -c port=COM27 dtr=high
```

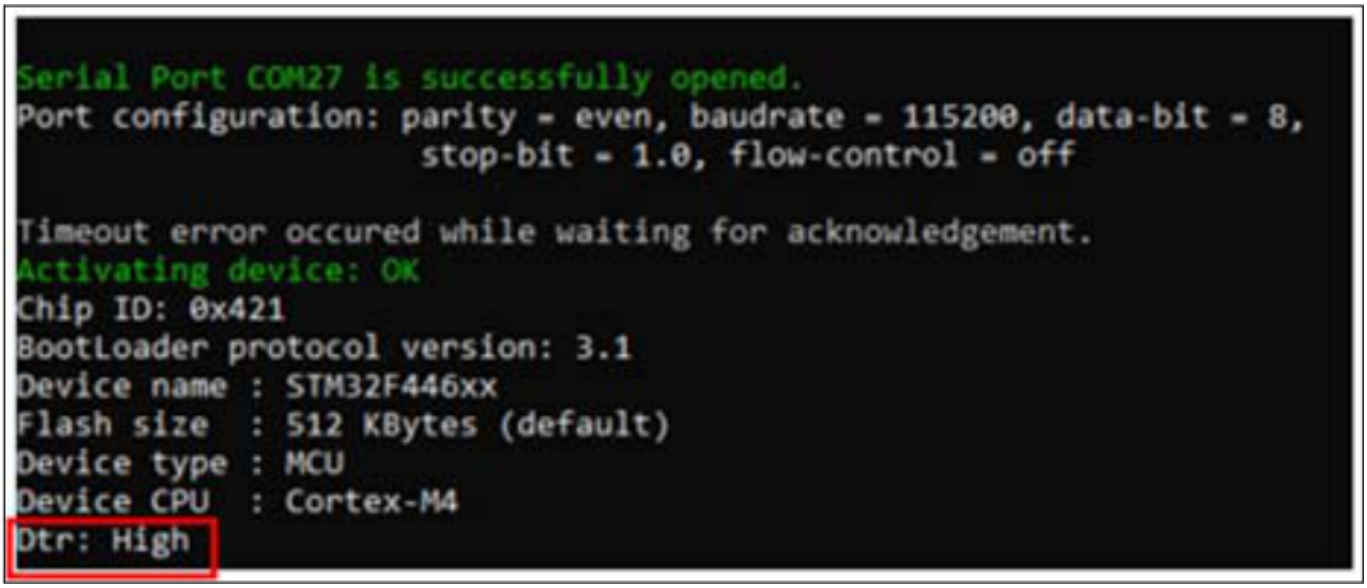


Fig. 4.17: Enabling COM DTR pin

4.4.6 SPI connection

Introduction

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/J-Link debug probe, or the bootloader interface over UART, USB, SPI, CAN, or I2C).

SPI settings

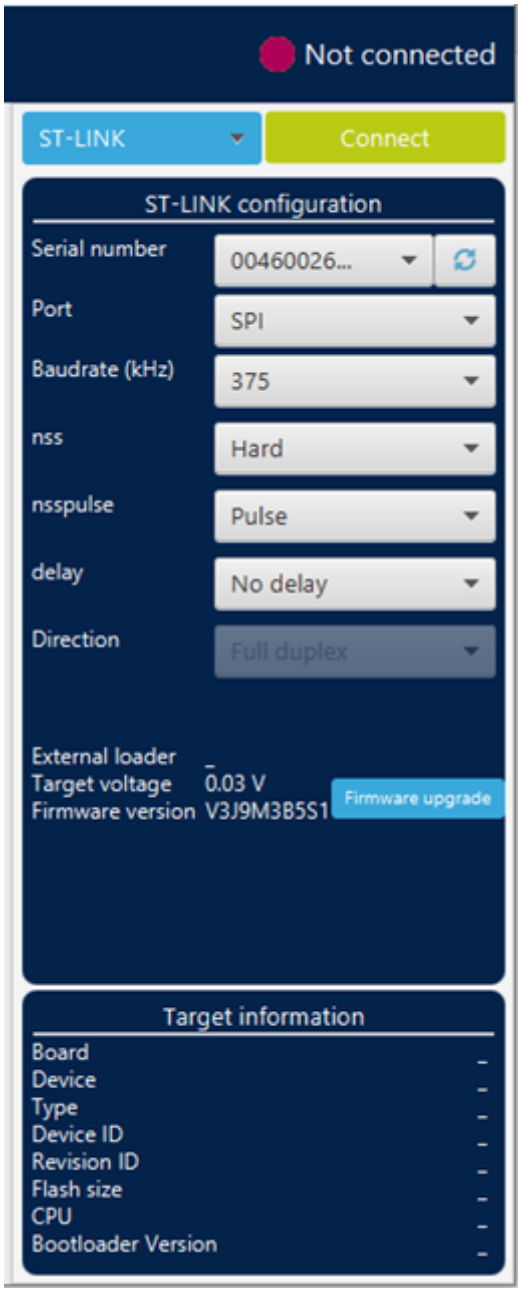


Fig. 4.18: SPI configuration panel

SPI bootloader fields

- Serial number: this field contains the serial numbers of all connected ST-LINK-V3 probes in case of use of SPI bootloader.
- Port: selects the SPI devices connected to the PC. Use the refresh button to recheck the available devices.
- Baudrate: selects the SPI baud rate.
- nss: slave select software or hardware.
- nsspulse: the slave selection signal can operate in a pulse mode, where the master generates pulses on nss output signal between data frames for a duration of one SPI clock period when there is a continuous transfer period.
- Delay: used to insert a delay of several microseconds between data.

- Direction: must be always Full-duplex, both data lines are used, and synchronous data flows in both directions.

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/JLINK/SPI/CAN/I2C).

Syntax:

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

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

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

SPI options

```
[br=<baudrate>]
[cpha=<cpha_val>]
[cpol=<cpol_val>]
[crc=<crc_val>]
[crcpol=<crc_pol>]
[datasize=<size>]
[direction=<val>]
[firstbit=<val>]
[frameformat=<val>]
[mode=<val>]
[nss=<val>]
[nsspulse=<val>]
[delay=<val>]
```

Note:
Baudrate (for example 187, 375, 750), default 375.

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

Option values and defaults:

- CRC polynomial value.
- 8- or 16-bit, default 8-bit.
- 2LFullDuplex/2LRxOnly/1LRx/1LTx.
- MSB/LSB, default MSB.
- Motorola/TI, default Motorola.
- Master/slave, default master.
- Soft/hard, default hard.
- Pulse/NoPulse, default Pulse.
- Delay/NoDelay, default Delay.

4.4.7 CAN connection

Introduction

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/J-Link debug probe, or the bootloader interface over UART, USB, SPI, **CAN**, or I2C).

CAN settings

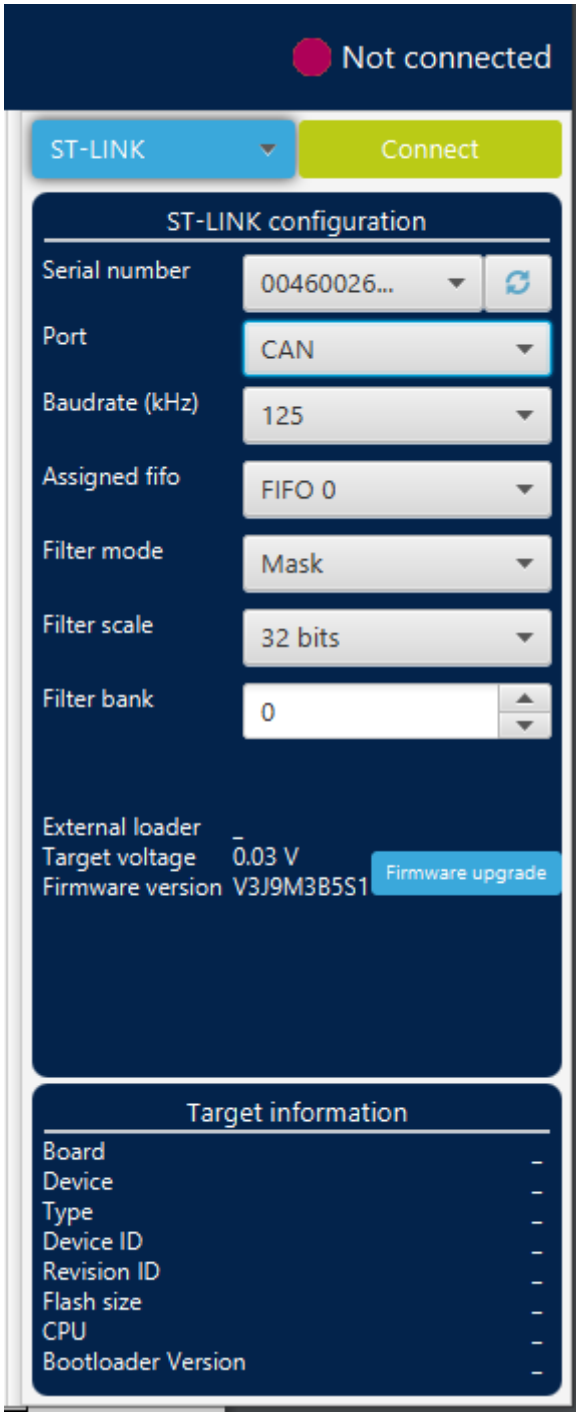


Fig. 4.19: CAN configuration panel

CAN bootloader fields

- 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.

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/JLINK/SPI/CAN/I2C).

Syntax:

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

port=<Portname>

Interface identifier, for example COMx (for Windows), /dev/ttySx (for Linux), usbx for USB interface, and SPI, I2C, or CAN for the corresponding SPI, I2C, and CAN interfaces.

[noinit=<noinit_bit>]

Set No Init bits, value in {0, 1}, default 0. noinit=1 can be used if a previous connection is active.

CAN options

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

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.

5 Release notes

5.1 Release details

5.1.1 Introduction

5.1.2 Release note updates

This release note is updated periodically to keep abreast of the STM32CubeProgrammer (STM32CubeProg) evolution, problems, and limitations. Check the STMicroelectronics website at www.st.com/stm32softwaretools for the latest version.

For the most recent release summary, see the table below.

Version	Summary
2.24.0	**Support for new devices:** • **STM32U396x/A6x:** – Internal memory programming through debug and bootloader interfaces – Option bytes programming through debug and bootloader interfaces – OTP programming through debug and bootloader interfaces – Core register programming through debug interfaces – Automatic mode – Serial Wire Viewer – Register viewer – SWD Multidrop – Incremental programming – Firmware encryption/signing – RDP regression with password – Key Wrapping via debug interfaces – SFI through debug interfaces • **STM32G55x/56x:** – Internal memory programming through debug and bootloader interfaces – Option bytes programming through debug and bootloader interfaces – OTP programming through debug and bootloader interfaces – Core register programming through debug interfaces – Automatic mode – Serial Wire Viewer – Register viewer – SWD Multidrop – Incremental programming – Firmware encryption/signing – OBkey generation/provisioning – Certificate generation – Debug authentication – Key Wrapping via debug interfaces – SFI through debug interfaces • **STM32G534/544:** – Internal memory programming through debug and bootloader interfaces – Option bytes programming through debug and bootloader interfaces – OTP programming through debug and bootloader interfaces – Core register programming through debug interfaces – Automatic mode – Serial Wire Viewer – Register viewer – SWD Multidrop – Incremental programming – Firmware encryption/signing – OBkey generation/provisioning – Certificate generation – Debug authentication • **STM32H7P5/7P4:** – Internal memory programming through debug and bootloader interfaces – Option bytes programming through debug and bootloader interfaces – OTP programming through debug and bootloader interfaces – Core register programming through debug interfaces – Automatic mode – Serial Wire Viewer – Register viewer – SWD Multidrop – Incremental programming – External memory programming via debug interfaces – Firmware encryption/signing – OBkey generation/provisioning – Certificate generation • **STM32C5Q/5P:** – Internal memory programming through debug and bootloader interfaces

5.1.3 Main Fixed issues for 2.24.0

ID	Summary
9447	CubeProgrammer REG dump doesn't work reliably
9686	Image generation xml file don't support a comment line using standard xml comment
9305	STM32CubeProg - Noreconnect option not working after OB programming
8992	Spurious secure slot erase during programming non secure one
9050	TPC GUI fails to open STiROT_Config.xml if it is updated by provisioning.bat
8926	The Fault instruction address indicated by Fault Analyzer is incorrect
9087	[CLI] Misleading warning displayed when erasing more than 3000 Sectors
9258	[DependsOnFusOp][STM32WB55] issue to deploy stack v1.10.0
8976	[F767] nDBOOT not showing on UART in the option bytes tab when nDBANK=0
9292	[GUI] Memory & File editing download button not working
9651	[H562] Failure to re-program a previously programmed high-cycle sector on STM32H5 1MB device
30	[H743/H753] Incorrect display of option bytes
9709	[H755] Incorrect Option bytes display
9391	[H7] Application hangs when changing BCM7 or BCM4
9191	[MacOS] STM32CubeProgrammer not supported on Apple Case sensitive File System
9772	[MacOS] Updater_Gui_Log.txt infinitely growing due to unsupported proxy type
9291	[N6] Unable to program Individual OTP Bits
9562	[STLinkDriver] Reconnect timeout issue
9640	[STM32CubeProgrammer] Wrong OTP16.bit0 description STM32N6
9493	[STM32L4][USB DFU] Start programming has no effect in v2.22.0 and v2.23.0
9062	[SWV] [WB55] SWV not reliably working on v2.22 on WB55 devices
7170	[U5] [JLINK] CHIPID reading issue
9525	[WL33] Value of RDP via the UART Bootloader interface problem
9546	[WL33] Enabling RDP takes a long time.

5.1.4 Customer support

For more information or assistance regarding **STM32CubeProgrammer**, contact the nearest STMicroelectronics sales office or use the STM32CubeProgrammer Community.

5.1.5 General information

Overview

STM32CubeProgrammer is provided as a standalone tool. To install the tool, see *Installing STM32CubeProgrammer*.

Host PC system requirements

STM32CubeProgrammer is supported on Windows, Linux, and macOS. For the latest information about supported operating systems and system requirements, see *System requirements*.

Licensing

STM32CubeProgrammer is delivered under the SLA0048 software license agreement and its Additional License Terms.

5.2 Release limitations

5.2.1 STM32 series limitations

STM32 series	Limitations
STM32G5 microcontrollers	• When TrustZone is enabled, the `Erase sector` operation with `Select all sectors` selected is not supported. Users must use mass erase instead.
STM32H7 microcontrollers	Support integrity check for SFI: • Only the devices with a bootloader version higher than `0x91` are supported. • STM32CubeProgrammer does not support devices with older bootloader versions. Users must use STM32CubeProgrammer v2.9.0 to use devices with old bootloader versions.
STM32H73xxx microcontrollers	Support integrity check for SFI/SFIx: • Only the devices with a bootloader version higher than `0x93` are supported. • STM32CubeProgrammer does not support devices with older bootloader versions. Users must use STM32CubeProgrammer v2.9.0 to use devices with old bootloader versions.
STM32H7Bxxx microcontrollers	Support integrity check for SFI/SFIx: • Only the devices with a bootloader version equal to or higher than `0x92` are supported. • STM32CubeProgrammer does not support devices with older bootloader versions. Users must use STM32CubeProgrammer v2.9.0 to use devices with old bootloader versions.
STM32H7 microcontrollers	• STM32CubeProgrammer does not support SFI. • STM32H7Axxx/Bxxx microcontrollers: download verification issue with OTP.
STM32H7Rx/7Sx microcontrollers	• Erasing the external flash memory is not supported using IWDG. • GUI: the external memory edition is not supported via USB and UART while connected to the OpenBootloader. • Option bytes are not supported via OpenBootloader interfaces.
STM32L0 microcontrollers	• Register access mode incorrectly displayed for register viewer.
STM32L4R5xx microcontrollers	• Issues occur when erasing sectors in bank 2 memory with the UART interface.
STM32U3 microcontrollers	• Limited support via JTAG (functional only with `mode = normal` and `reset = HWrst`). • STM32U3C5/STM32U3B5: unable to program binary or change OB when `IWDG_SW=0`. • STM32U335/STM32U345: the programming operation fails when Swap_Bank is enabled.

STM32 series	Limitations
STM32U5 microcontrollers	• Unable to read the chip ID via J-Link when RDP is set to Level 2. • OTP memory programming failure with CLI when TZEN is enabled. • STM32U535/545 microcontrollers: SFIx is not supported. • STM32U59xxx microcontrollers: SFIx only supports the area E installation with OTFDEC region 1 (region 0 on TPC). • STM32U5Fxxx and STM32U5Gxxx microcontrollers: unable to program SFIx via SPI/I2C on GUI. • STM32U5Fxxx and STM32U5Gxxx microcontrollers: SFIx programming with big size external firmware (63 MB) over SPI blocks on several areas and requires user intervention (`CTRL + C`) to resume. • STM32U575/STM32U585 microcontrollers: for CPNs with flash size 1M, erase selected sectors fails when `SWAP_BANK` is enabled. • STM32U575/STM32U585 microcontrollers: SFIx programming with big size external firmware (63 MB) over SPI blocks on several areas and requires user intervention (`CTRL + C`) to resume. • STM32U535/STM32U545 microcontrollers: for CPNs with flash size 2M, erase selected sectors fails when `SWAP_BANK` is enabled with the UART interface.
STM32L43 and STM32L45 microcontrollers	• Failed to erase a part of the last sector using the GUI with the SWD interface.
STM32WBA microcontrollers	• OpenBootloader via SPI: cannot connect to STM32CubeProgrammer when the baud rate is below 3000 Bd. • `Connect Under Reset` is not working when called by STM32CubeIDE.
STM32WBA2 microcontrollers	• Erase operation failed when the external loader is selected. Note: deselect the external flash to be able to perform the erase operation.
STM32WL33 microcontrollers	• Failed to connect with Hotplug mode on CLI. • STM32CubeProgrammer fails to program `RDP=0xABACABAD` on WL33.
STM32L5 microcontrollers	• Unable to read, write, or erase the external flash memory when TZEN is enabled. • SFI programming failed via UART on Linux.
STM32V8 microcontroller	• Erase NVM High Cycle data is not working; a workaround is to write `0xFFFFFFFF` in the register. • For High-Cycle memory, use address regions rather than sector numbers in the sectors table. Sector numbers identify sectors and avoid conflicts with internal flash sector numbering; they do not determine High-Cycle regions. • Bank swap for High-Cycle memory is not yet supported.
STM32H74x/5x/7x microcontrollers	• When connecting the target via J-LINK, the BCM4 option byte is removed from STM32CubeProgrammer if it has been disabled.

5.2.2 Additional limitations

- SFIx fails when programming:
  - Large flash memory content on Linux via the UART interface.
  - Small flash memory content on supported bootloader interfaces with all platforms.
- USB interoperability issue on macOS due to ST-LINK limitation.
- Incremental programming :
  - Is not supported when using bootloader interfaces.
  - Is not supported when TrustZone is enabled; the tool switches automatically to the normal programming procedure.
  - Is an experimental feature with limited test coverage:

    For internal flash programming, tested devices are: STM32C03x, STM32C05x, STM32C09x, STM32F101/F102/F103, STM32F37xx, STM32F411xC/E, STM32F72x/STM32F73x, STM32F76x/STM32F77x, STM32G03x/STM32G04x, STM32G0B0xx/B1xx/C1xx, STM32G491xx, STM32H50x, STM32H533/523, STM32H56x/573, STM32H72x/STM32H73x, STM32H7R/Sxx, STM32L05x/L06x/L010, STM32L4Pxxx/STM32L4Qxxx, STM32L4x1/STM32L475xx/STM32L476xx/STM32L486xx, STM32L5xx, STM32U0xx, STM32U3xx, STM32U535/STM32U545, STM32U575/STM32U585, STM32U59x/5Ax, STM32WB5x/35xx, STM32WBA52/54/55, STM32WBA6x, and STM32WLxx.

    For external flash programming, tested boards are: STM32H735G-DK, STM32H7S78-DK, STM32H573I-DK, STM32U575I-EV, STM32WBA65I-DK1.
- The automatic mode does not work with the UART interface when using an ST-LINK with dual VCP. The ST-LINK configuration must be changed to single VCP.
- In Windows 10 32-bit, STM32TrustedPackageCreator display can be cut when maximizing the window.
- OBKey provisioning with J-LINK is not supported. Currently, the CLI says it clearly, but the GUI message does not.
- In IAP mode, memory editing is not supported in the GUI.
- Memory map export supports a maximum of 9 segments.
- SWV feature is provided as a beta functionality for STM32V8 and STM32C5 devices. It is not fully validated.
- MCU factory reset is supported only via ST-LINK.
- RDP regression with password is supported only via ST-LINK.
- SFI feature:
  - For the full list of supported devices, please refer to AN4992.
  - Parallel SFI Programming with Shared HSM is not supported.
  - SFI is not working with SWD multidrop.
- Starting from STM32CubeProgrammer v2.21.0, the `-align` option when using `STM32_SigningTool_CLI` is mandatory with STM32N6.
- Write and erase operations using an external flash with a start address of `0x00000000` are not supported.
- Flash high cycle:
  - Programming is not supported at secure addresses.
  - A masse erase is required before programming the full flash high cycle area (bank1+bank2).
- The flash high cycle programming is not supported on secure address.
- The OTP memory programming is not supported when TZEN is enabled for CM33 SoCs.
- Display issues occur on Fedora when the system display scale is set to 125%; the recommended scale is 100%.

- Any file path provided as an input argument to the STM32CubeProgrammer CLI that contains spaces must be enclosed in quotation marks.
- Old ST-LINK/V2 firmware versions may not be properly detected or updated by STM32CubeProgrammer. In some cases, the probe must be unplugged and reconnected before the firmware update can proceed.
- On Ubuntu® 26, additional libraries need to be installed.
  - First update the package index:

```
sudo apt update
```

  - Then install the required libraries:

```
sudo apt install -y libxcb-icccm4 libxcb-image0 libxcb-keysyms1 \
  libxcb-render-util0 libxcb-xkb1 libxcb-cursor0 \
  libxkbcommon0 libxkbcommon-x11-0 libopengl0
```

6 Community

6.1 Support link

If you need assistance or have questions regarding STM32CubeProgrammer, support is available. Access a wide range of resources and receive help from the STMicroelectronics support team.

For more information and to get support, visit the STMicroelectronics Support Home.

6.2 Community link

Feedback is valuable and helps improve STM32CubeProgrammer. If you have suggestions, issues, or comments, share them with the community.

To provide feedback and participate in discussions, visit the ST Community.

6.3 Knowledge articles link

To find information on STM32 microcontroller programming, troubleshooting, ecosystem, and product details, visit the STM32 MCUs Knowledge Base.

6.4 Additional links

ST-Link connect

- How can I connect to my STM32 evaluation board using STM32CubeProgrammer - STMicroelectronics Community

STM32U5, RDP regression with password and TrustZone

- How to disable Trustzone on STM32U5 Series - STMicroelectronics Community
- How to regress from RDP level 2 to RDP level 0 on the STM32U5 series - STMicroelectronics Community
- Precautions to avoid lock states when developing with TrustZone - STMicroelectronics Community

STM32H5, product state and DA

- How to enable RDP-like product state flash protection in STM32H5 - STMicroelectronics Community
- Implementing debug authentication on STM32H5 using CLI tools - STMicroelectronics Community

STM32H7

- How to enable/disable cores on dual-core STM32H7 using STM32CubeProgrammer - STMicroelectronics Community
- How to unbrick an STM32H7 after setting the wrong power mode - STMicroelectronics Community

BL/OpenBL

- How to use the ST Open Bootloader for STM32 - STMicroelectronics Community
- How can I use STM32CubeProgrammer to access the USB-DFU bootloader on my STM32 board? - STMicroelectronics Community
- How to use STM32CubeProgrammer to access the USART bootloader on my STM32 board - STMicroelectronics Community
- How to use STM32CubeProgrammer and the STLINK-V3SET to access the SPI bootloader on my STM32 board - STMicroelectronics Community

- How to use STM32CubeProgrammer and the STLINK-V3 SET to access the I2C bootloader on my STM32 board - STMicroelectronics Community
- How to Modify USB DFU firmware examples to work with STM32CubeProgrammer - STMicroelectronics Community
- How to use STM32CubeProgrammer and the STLINK-V3 SET to access the I2C bootloader on my STM32 board - STMicroelectronics Community
- How to program an STM32 with another STM32 using an embedded bootloader - STMicroelectronics Community

Factory reset

- Restoring the STM32 option bytes to their factory settings - STMicroelectronics Community

Reg dump

- How can I use STM32CubeProgrammer to perform a reg dump - STMicroelectronics Community

External loaders

- Custom external loader “failed to download segment[0]” - STMicroelectronics Community
- How to add your SPI flash into the STM32CubeProgrammer’s external loader Part 1 - STMicroelectronics Community
- How to add your SPI flash into the STM32CubeProgrammer’s External Loader Part 2 - STMicroelectronics Community

STM32N6

- How to add the STM32N6’s header signature as post build output - STMicroelectronics Community
- How to connect the STM32N6 using STLINK and serial boot interfaces - STMicroelectronics Community
- How to create an STM32N6 FSBL load and run - STMicroelectronics Community
- How to program the OTP fuse bits in the STM32N6 - STMicroelectronics Community
- STM32N6 boot ROM explained - STMicroelectronics Community

7 FAQ

7.1 Basic questions

What is STM32CubeProgrammer?

STM32CubeProgrammer is STMicroelectronics' free programming tool for STM32 microcontrollers and STM32 microprocessors.

What is STM32 Trusted Package Creator?

STM32 Trusted Package Creator is a tool used to build secure STM32 firmware packages. It is delivered as part of the STM32CubeProgrammer package.

Is STM32CubeProgrammer free and open source?

STM32CubeProgrammer is a free tool publicly delivered by STMicroelectronics. However, it is not an open-source project.

How do I download and install STM32CubeProgrammer?

Download the installer for your operating system from the installer section, then run it and follow the installation steps. For detailed instructions, refer to *Installing STM32CubeProgrammer*.

Which operating systems and architectures are supported?

The following operating systems and architectures are supported:

- Windows 10 32-bit (x86) or 64-bit (x64), and Windows 11 64-bit (x64)
- Linux: Ubuntu LTS 22.04 and LTS 24.04, and Fedora 42
- macOS 15 (Sequoia) and macOS 26 (Tahoe), on x86_64 and ARM64 (aarch64)
- Raspberry Pi OS (Debian 12 "Bookworm" or later)

Does STM32CubeProgrammer support Raspberry Pi?

Yes. STM32CubeProgrammer can run on Raspberry Pi 64-bit devices by using the Linux ARM 64-bit package to program STM32 devices.

Raspberry Pi 32-bit devices are not supported.

Which STM32 series are supported?

STM32CubeProgrammer supports all STM32 microcontrollers (MCUs) and STM32 microprocessors (MPUs).

Which programming interfaces are supported?

STM32CubeProgrammer supports the debug interface through:

- ST-LINK for SWD and JTAG
- Segger J-Link for SWD

It also supports the bootloader interface through:

- UART
- USB DFU
- I2C
- SPI
- CAN

Which file formats are supported for programming?

STM32CubeProgrammer supports the standard programming file formats used across STM32 workflows, including binary files and common executable or hexadecimal formats depending on the target and programming flow.

Does STM32CubeProgrammer support secure programming and TrustZone?

Yes. STM32CubeProgrammer supports secure programming features on compatible devices, including secure firmware install workflows and device security-related operations. It also supports TrustZone-related programming flows on STM32 devices that provide TrustZone capabilities. For details, refer to the secure programming sections in the user manual and command-line documentation.

How do I get the latest updates and documentation?

Refer to the online release information and documentation delivered with STM32CubeProgrammer. For support resources and community content, see *Community*.

What is the STM32CubeProgrammer API?

STM32CubeProgrammer provides an API intended to let external applications reuse programming features programmatically instead of only through the graphical user interface or command-line executable.

7.2 Getting started questions

Do I need an ST-LINK probe to use this tool?

No. An ST-LINK probe is required for SWD or JTAG debug connections, but it is not required when using supported bootloader interfaces such as UART, USB DFU, I2C, SPI, or CAN.

How do I program my Nucleo or Discovery kit board?

Connect your board to the PC using the ST-LINK connector and make sure the ST-LINK selector is correctly set for power supply through ST-LINK.

If your board has two similar connectors, one is usually for ST-LINK and the other for USB USER. Check the board markings and use the ST-LINK connector.

Then follow these steps:

1. Open STM32CubeProgrammer.
2. Click **Connect**.
3. Open the **Erasing & Programming** panel from the left menu.
4. Select your firmware with **Browse**.
5. Click **Start Programming**.

My board is not detected. What should I check first?

First check the USB connection and cable, the board power supply, and the ST-LINK settings.

For additional guidance, refer to the connection and programming sections in the documentation, especially the user manual and connection-related chapters.

What if my question is not listed here?

If your question is not listed here, visit the online documentation for more guidelines, examples, and detailed articles.

You can also ask for help through the ST support and community resources listed in *Community*.

Note

See also: hardware debugger and programmer tools for STM32.

7.3 Security questions

What is the product life cycle of an STM32?

The STM32 product life cycle describes the different support phases of a device, from active development and full availability to long-term maintenance and eventual end of life.

From a security perspective, a device can be relatively open during development for testing and debugging, then locked in the field to protect firmware, data, and system integrity.

What is RDP vs. RDP with Password vs. DA?

These acronyms correspond to three security access models that reflect the evolution of STM32 device protection:

- **RDP (Readout Protection): the legacy protection scheme used mainly on older** STM32 series to prevent unauthorized reading of flash memory.
- **RDP with Password: a newer protection model on some STM32 devices, where** access is controlled by a password rather than only by protection levels.
- ***DA (Debug Authentication)*: a more advanced secure access flow that allows** controlled debug and unlock operations through an authentication process.

In short, RDP is the classic model, while RDP with Password and DA are newer approaches that provide more controlled and secure access on recent devices.

What is STM32 provisioning?

Provisioning is the process of preparing an STM32 device before deployment by configuring its security material and access rules.

For STM32 devices using RDP with Password, provisioning means defining the password-based protection used later to control access and regression.

For STM32 devices using DA, provisioning means preparing the device for trusted authentication so that only authorized tools or users can unlock it. This can include provisioning OBKey data, certificates, or platform authorization data.

Note

For more information on STM32 security topics, refer to the STM32 Security wiki.

7.4 Production question

Can I use STM32CubeProgrammer in a production environment?

No. STM32CubeProgrammer is a development programming tool and is not intended for production programming, as stated in the license and in the development versus production programming guidance:

Development programming and production programming.

7.5 History

What is STM32CubeProgrammer the evolution of?

STM32CubeProgrammer is the evolution of the following legacy tools:

- STM32 ST-LINK Utility

- DfuSe USB device firmware upgrade tool
- STM32 Flash loader demonstrator

STM32CubeProgrammer integrates their main functionalities and features. These three tools are now discontinued.

What is the country of origin of STM32CubeProgrammer?

STM32CubeProgrammer is developed by STMicroelectronics engineering teams located in Tunis, Tunisia, and in Le Mans, France.