FlashRunner 2.0 Series

High-Performance, Standalone In-System Programmers

Programmer's Manual

Revision 2.23 — December 2023

SMH Technologies®

UNIVERSAL PRODUCTION IN-SYSTEM PROGRAMMING

SMH Technologies S.r.l.
via Giovanni Agnelli, 1
33083 Villotta di Chions (PN) Italy
E-mail (general information): info@smh-tech.com
E-mail (technical support): support@smh-tech.com
Web: http://www.smh-tech.com

Contents

Note: *an updated version of FlashRunner 2.0 System Software is available on the SMH Technologies website (www.smh-tech.com). Please check it before continuing to read this documentation.*

1.1 Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, SMH Technologies assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accidents, or any other cause.

1.2 Getting Technical Support

Note: *Keep FlashRunner 2.0 always in a well-ventilated area in order to prevent product overheating, which could affect product performance and, if maintained for a long time, it could damage product hardware components.*

SMH Technologies is continuously working to improve FlashRunner firmware and to release programming algorithms for new devices. SMH Technologies offers fast and knowledgeable technical support to all of its customers and it is always available to solve specific problems or meet specific needs.

To get in touch with SMH Technologies, please refer to the contact information below.

Phone: +39 0434 421111
Fax: +39 0434 639021
Technical Support: support@smh-tech.com

2 System Setup/Upgrade

2.1 Software Setup

The FlashRunner system software setup installs all required components to your hard drive. These components include:

- FlashRunner 2.0 Workbench software (GUI).
- Command-line utilities and Interface Library (DLL).
- Documentation in PDF format.

To install the FlashRunner system software:

- Check the latest **"System Software"** package for FlashRunner 2.0 on SMH Technologies website.
- Follow the on-screen instructions in order to install the System Software.

Note: *to install the FlashRunner system software you must log in as Administrator.*

To launch FlashRunner 2.0 Workbench under Microsoft Windows®, select **Start → Programs → SMH Technologies → FlashRunner 2.0 → FlashRunner 2.0 Workbench**.
For more details on the functionalities of the Workbench GUI, please refer to chapter 3.

2.2 What you need to start

FlashRunner 2.0 supports thousands of devices. In order to program a specific one, you will need the following:

- **Driver file** (**.so**): dynamic library which contains routines needed to program a set of specific devices. SMH Technologies releases continuous updates to support new devices; when you request a new device, you'll often receive also an updated version of the driver.
- **License file** (**.lic**): text file which contains a CRC key that binds together your specific FlashRunner 2.0 (by using its unique serial number) with your target device. There are different license types, Please ask SMH Technologies Sales Team for more information.

    a) License for a single target device.
    b) License for a single-family.
    c) License for a silicon manufacturer.

- **FRB file** (**.frb**): FlashRunner proprietary file format used to store customers' firmware. There is a specific tool available on FlashRunner 2.0 Workbench, called FRB Manager, described in ch 3.16 which allows the conversion from the customer file, to the FRB file.
- **Project file** (**.prj**): text file containing all the necessary information for setting your programming session. They contain some static information regarding the device, all user-configurable parameters and all commands which will be executed on the target device. FlashRunner 2.0 Workbench has a tool, Project Wizard, described in chapter **Errore. L'origine riferimento non è stata trovata.** which allows creating a project from scratch only using graphical

items. Once created, a project could be modified by simply editing it with a text editor.

On the SMH Technologies website (www.smh-tech.com) you can check the full supported device list.

In order to program a specific device identified by its part number, a dedicated license file for the couple "device and programmer" (identified by its serial number) must be purchased.

In addition, you can order a shared license, which binds a specific device to more FlashRunner (up to 10 programmers can be included inside a license). By doing this, a single file could be installed in more programmers and enable all of them to program a specific target device.

You can purchase a license through our direct channel by writing to our Sales Office: sales@smh-tech.com. If you instead bought FlashRunner from an SMH distributor, please contact him. Once ordered a license, you'll receive a package with a license file and a driver file, which must be copied to your FlashRunner 2.0 product.

2.3 Connection setup

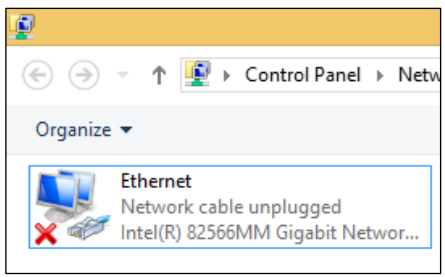
FlashRunner 2.0 Workbench can control the programmer in **Host mode** (via USB or Ethernet connection), or in **Standalone mode** (via Control Connector) which can select and run a specific project stored in its internal storage memory. For first use and to connect the programmer to FlashRunner 2.0 Workbench, you'll have to use FlashRunner 2.0 in Host mode.

2.3.1 Ethernet LAN connection settings

By default, FlashRunner 2.0 IP address is *192.168.1.100*, with Subnet Mask *255.255.255.0* and gateway *192.168.1.1*. After the first time connection, you will be able to change this setting using *SETIP* command.

Note: *LAN connector area reaches more than 50° degrees when connected to the host. Keep FlashRunner 2.0 always in a well-ventilated area to prevent product overheating, which could affect product performance and, if maintained for a long time, could damage product hardware components.*

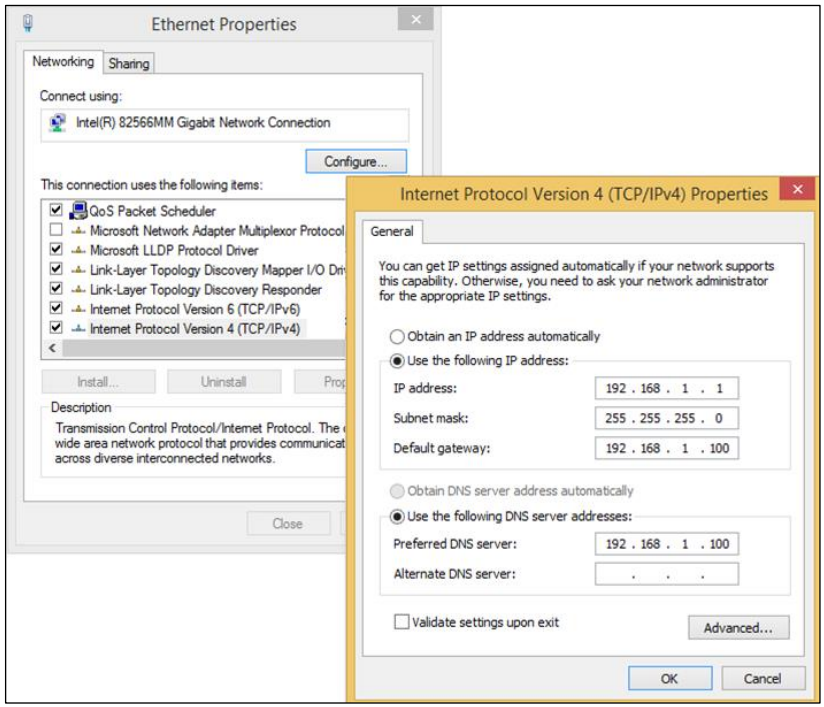
Use the ethernet cable included in FlashRunner 2.0 box and connect it to your switch or directly to your host pc. Once connected, the red cross in the network connections icon related to your network card should disappear.



If host pc and FlashRunner 2.0 are connected through a router, please be sure that they are in the same subnet: host pc IP address must be included between 192.168.1.1 and 192.168.1.254 address range.

If your pc and FlashRunner 2.0 are directly connected, you'll need to set a static IP on the network card used for connecting host pc with FlashRunner 2.0. Please open the network card settings window and use the following:

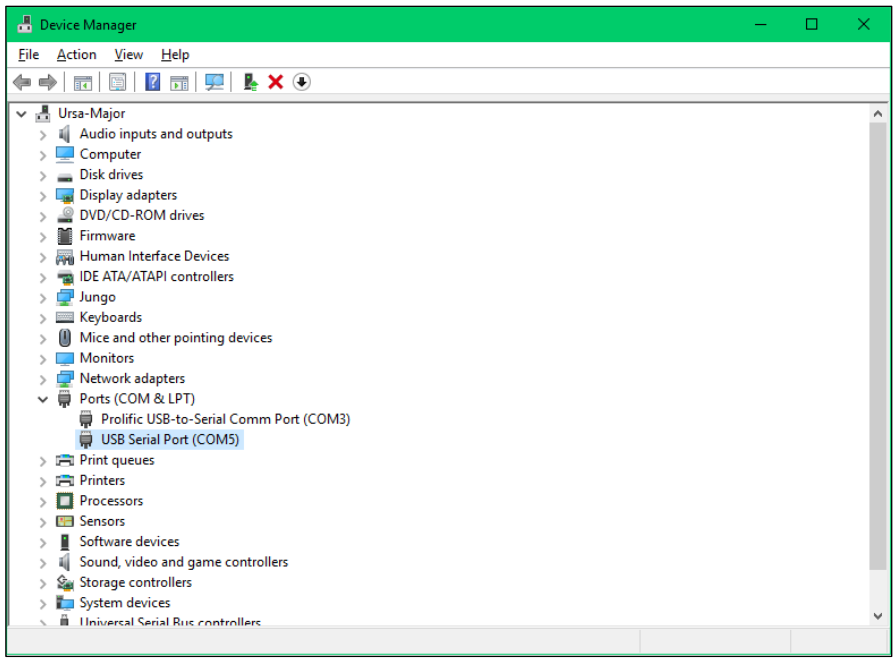
- **IP ADDRESS**: 192.168.1.X (where X is whatever number from 1 up to 254 except 100, which is FlashRunner IP).
- **SUBNET MASK**: 255.255.255.0
- **GATEWAY**: 192.168.1.100



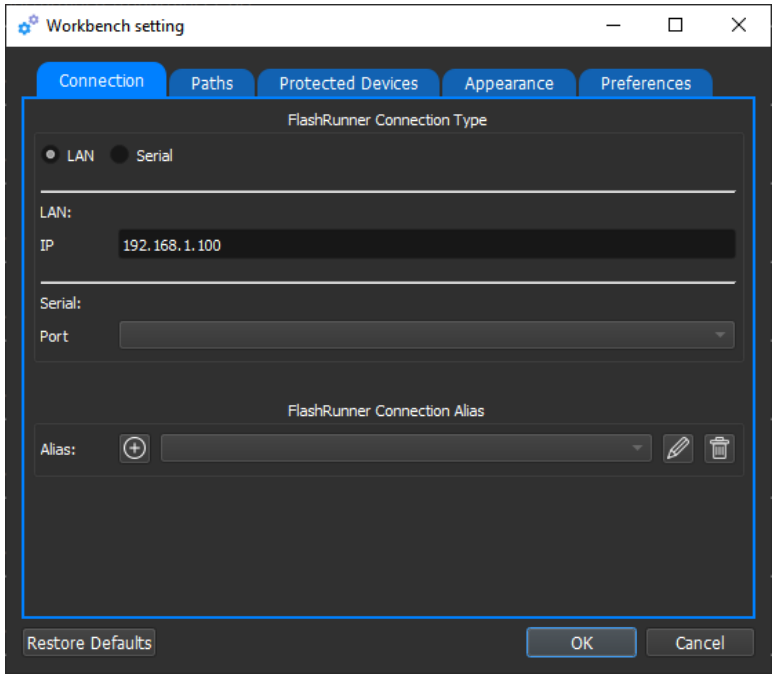
FlashRunner 2.0 Workbench is configured by default to connect to 192.168.1.100 FlashRunner 2.0 IP address. If need to change FlashRunner 2.0 IP address, you can easily update also FlashRunner 2.0 Workbench on connection settings.

2.3.2 **USB CONNECTION – WINDOWS®**

Once connected USB cable, please check on "*Device Manager → Ports (COM & LPT)*" if you can find USB Serial Port (COMX); where X is an integer number. If not, please click "*Action → Scan*" to update the hardware detected:



Once found this item, please use the same COM port to setup FlashRunner 2.0 Workbench software. Click on "*Settings->Connection->Serial*" put COMX value inside "*port textbox*".



2.3.3 **USB CONNECTION – LINUX**

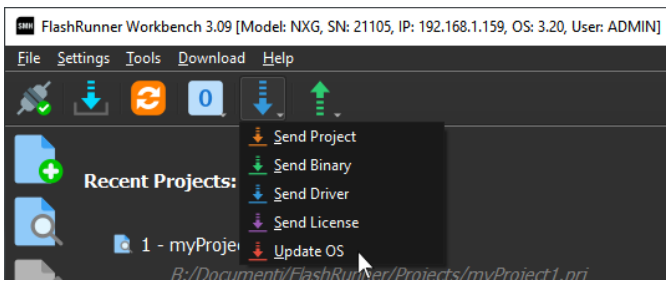
Please check with *dmesg* command which device node has been assigned to FlashRunner 2.0. Usually, Linux assigns *ttyUSBX* (where X is an integer number) device node. Please check under */dev* folder if your user has write/read privileges on */dev/ttyUSBX* device node. If not, please add it through *chmod*. Then open FlashRunner 2.0 Workbench "*Settings->Connection->Serial*"and fill "*port textbox*" with */dev/ttyUSBX*.

2.4 OS Update

Please, note that the procedure below is referred to the latest version of GUI Workbench.

In order to update FlashRunner 2.0 simply follow these steps:

1. Please connect to FlashRunner 2.0 using the "*Connect*" button at the top left of GUI Workbench.
2. Click on *"Download -> FlashRunner OS"* to get the latest FlashRunner 2.0 firmware, or download it directly from SMH website.
3. Click to "*Update OS*" in the GUI Workbench, like in the image below.



4. Then select the file "*update.tgz*" that you just downloaded. The GUI Workbench will transfer the file and the FlashRunner will reboot.
5. Please, connect again to FlashRunner using "*Connect*" button at the top left of GUI Workbench.
6. Check the OS version by sending, using the *Terminal*, on "*Master channel*" the "SGETVER" command. Or you can read it on the top of the Workbench (in the picture above: OS:3.19)

3 FlashRunner Workbench

3.1 Overview

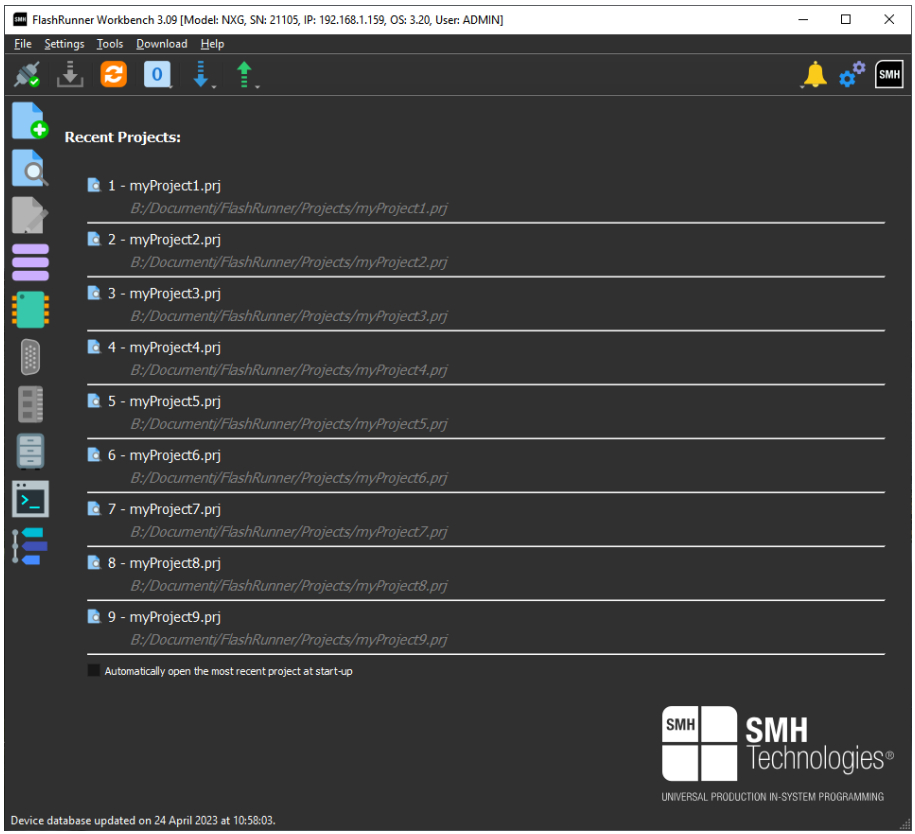
FlashRunner Workbench is a simple application for PC that is able to communicate with FlashRunner 2.0, FlashRunner LAN NXG and FlashRunner HS. It performs the following operations:

1. Create new projects;
2. Run projects and monitor programmer status;
3. Create FRB binary files;
4. Send and receive files to / from the programmer (such as projects, FRB, drivers and licenses);
5. Update OS and Drivers;
6. Retrieve log.

FlashRunner Workbench is compatible with all Microsoft Windows® operating systems and with Linux operating systems.

3.2 Opening window

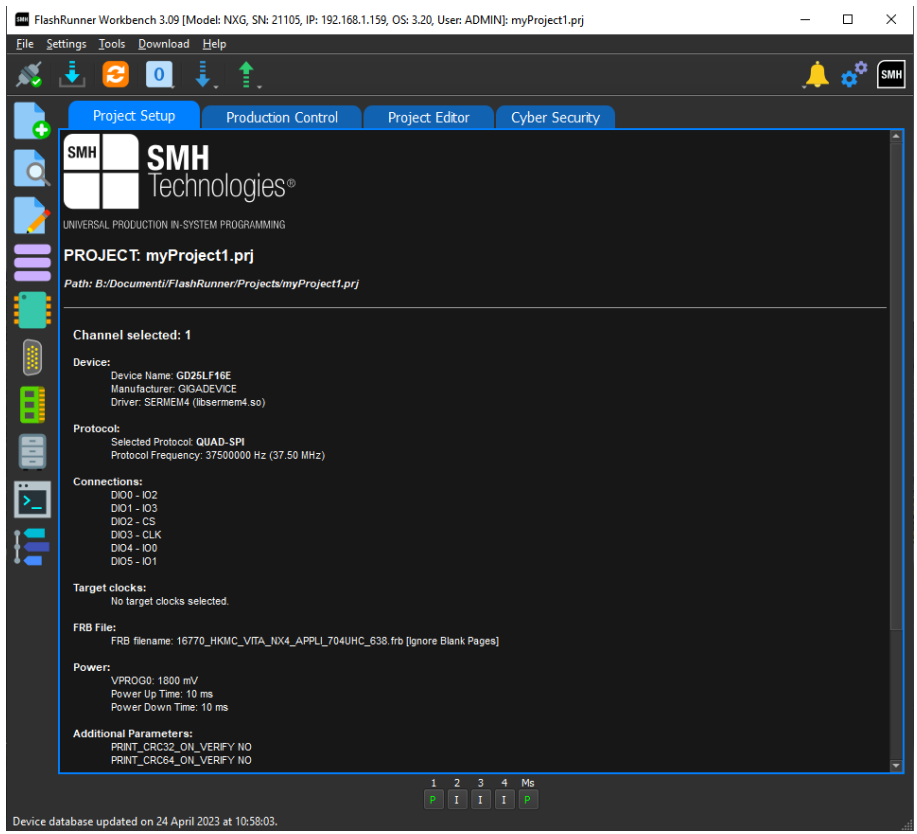
Once you run FlashRunner Workbench you'll see a window like the one below. It's designed with a top toolbar, a left toolbar and a central area that contains the recent projects.
From this window, you can create a new project or open an existing one.



After opening a project, the opening window will change and you will see the project details. The new window will be like the one in the figure below.

This window has still the same toolbars and a central area composed of 3 tabs:

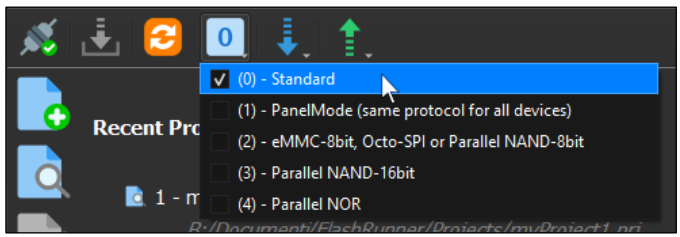
1. **Project Setup**: this tab gives a review of all settings of the current project**.**
2. **Production Control**: this tab monitors the on-going programming session.
3. **Project Editor**: this tab allows the user to manually edit the project from an advanced text editor.
4. **Cyber Security**: this tab allows the user to manage cyber security features.



3.3 Top toolbar

From left to right, the top toolbar provides the following features:

1. **Connect button**: connect/disconnect from FlashRunner and review connection status.
2. **Send configuration button**: send project and FRB to FlashRunner.
3. **Update database**: download the latest version of the Devices.smh file, which contains all the info of the supported devices.
4. **Working mode**: set the working mode of FlashRunner (this command is not available if the unit connect is a FlashRunner HS).



5. **Send button**: click to send projects, FRBs, drivers, licenses and OS updates.
6. **Get button**: click to get projects, FRBs, drivers, licenses and logs.

3.4 Left toolbar

The left toolbar shows the most important features of FlashRunner Workbench at a sight.

 Create project wizard. See ch **Errore. L'origine riferimento non è stata trovata.**

 Edit actual / existing project

 Load project

 Advanced FRB manager to create / edit an FRB file

 Show device list

 Pin map of the devices selected in the project

 Memory map of the devices selected in the project

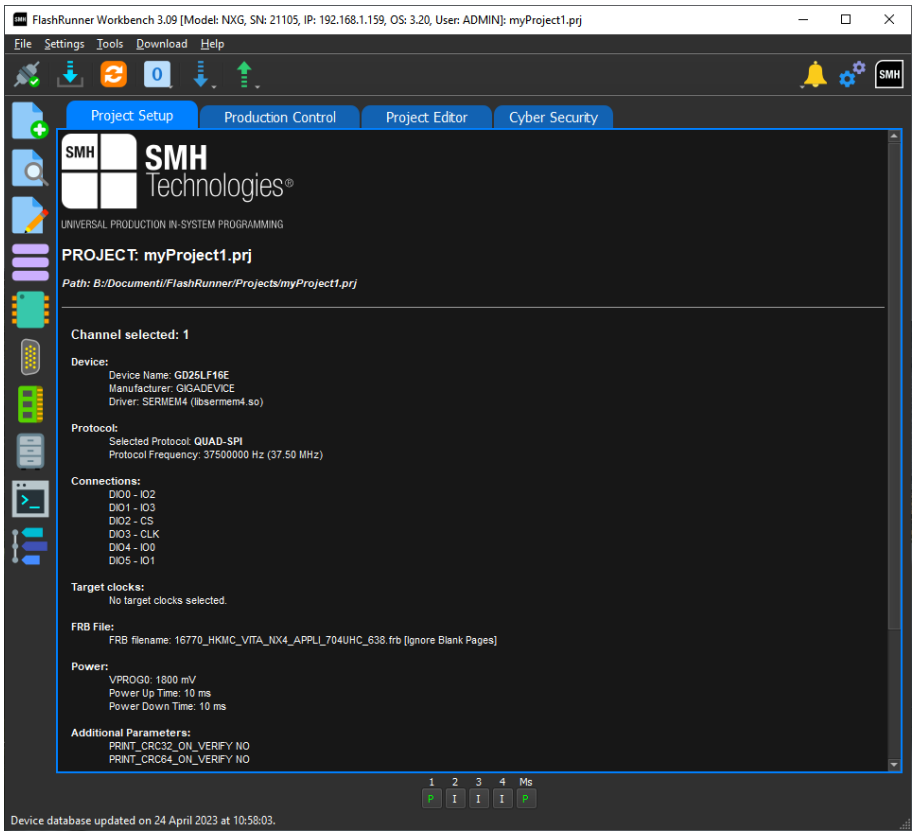
 Advanced File Manager. See ch 3.11

 Terminal. See ch 3.12

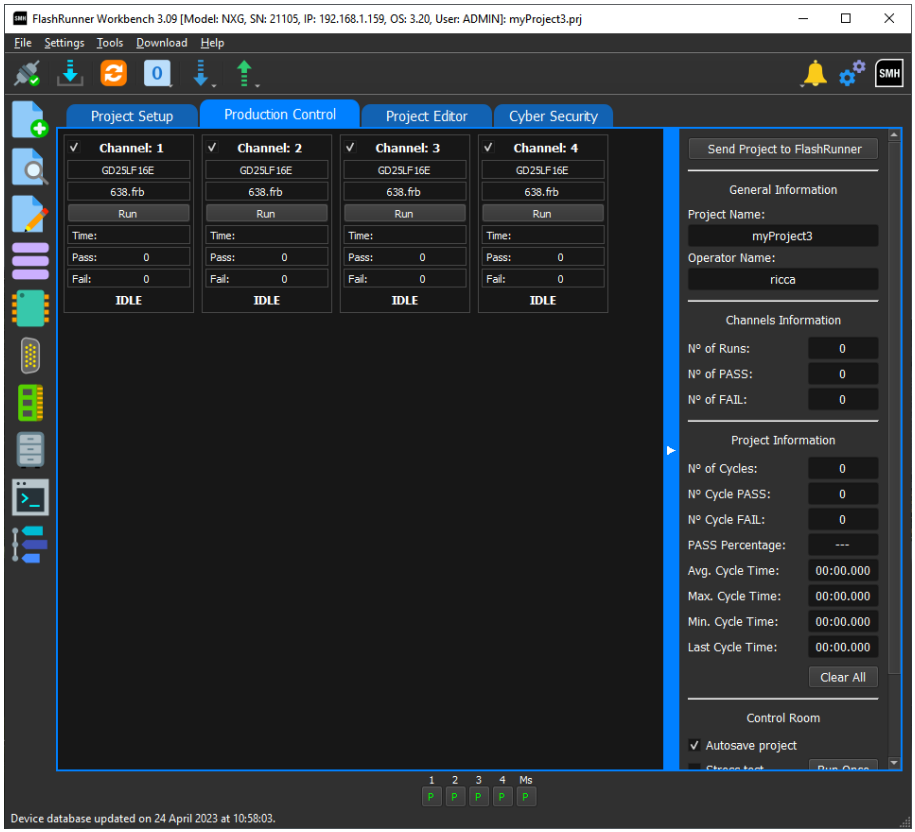
 Log. See 3.13

3.5 Project setup

After creating or opening a project, you will see a review of all the project settings. Moreover, you will get also information about connections and wirings, they are also available on the Pin Map Tool described in ch 3.15.
It is also possible to export this page in PDF.



3.6 Production Control



After opening a project, into the Production Control tab will be loaded a widget for each channel defined inside the project. Each widget contains the following labels:

1. **Device**: shows the target device name defined for that channel.
2. **Binary File**: shows FRB file defined for that channel.
3. **Run button**: the button which starts the project only on that single channel.
4. **Prog. Time**: shows the total execution time for that channel.

5. **N° of PASS**: shows the number of successful project executions for that channel.
6. **N° of FAIL**: shows the number of failed project executions for that channel.
7. **Status**: label which reports actual channel status. There are four possible states:
    a. **Pass**: last project execution completed successfully and the channel is idle.
    b. **Fail**: last project execution failed and the channel is idle.
    c. **Idle:** the channel is waiting for project execution.
    d. **Busy:** The channel is running a project.

On the right side of Production Control there are 5 sections:

1. **Send Project to FlashRunner**: this button sends the PRJ file and FRB files to FlashRunner.

2. **General Information:**
    a. **Project Name**: shows the project name currently loaded.
    b. **Operator Name**: shows the operator name (the user can insert it there).

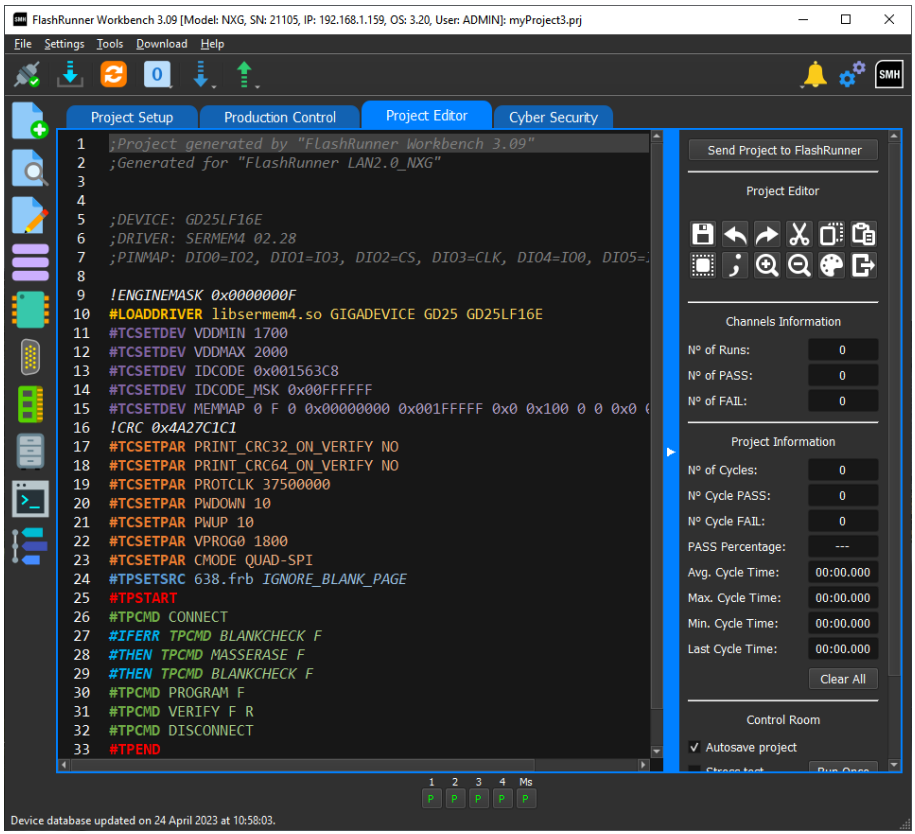
3. **Channels Information:**
    a. **N° of Runs:** shows the total number of executions considering each channel separately.
    b. **N° of PASS**: shows the total number of successful executions considering each channel separately.
    c. **N° FAIL**: shows the total number of failed executions considering each channel separately.

4. **Project Information:**
    a. **N° of Cycles:** shows the total number of project executions.
    b. **N° Cycles PASS**: shows the total number of successful project executions.
    c. **N° Cycles FAIL**: shows the total number of failed project executions.
    d. **PASS Percentage**: shows the actual pass percentage over the total number of project executions.
    e. **Avg. Cycle Time**: shows the average time of project executions.
    f. **Max. Cycle Time**: shows the maximum time of project executions.
    g. **Min. Cycle Time**: shows the minimum time of project executions.
    h. **Last Cycle Time**: shows the time of the last project execution.
    i. **Clear All**: this button will reset all the shown values.

5. **Control Room:** this section lets the user control the project executions. It is possible to launch a single project execution or to launch a stress test with multiple consecutive executions. Stress test mode can be launched with some additional settings:
    a. **Sync. Channels**: this option, if enabled, synchronize the start of the project on all the channels (default case), otherwise each channel will run separately.
    b. **Stop on Failure**: this option, if enabled, stops the stress test if a channel fails.
    c. **Limited to**: this option sets a limit to the number of project executions.

3.7 Project Editor

Into the Project Editor tab, the user can find a built-in text editor which can be used to manually edit the project file.
This editor has a syntax analyzer that helps the user to avoid mistakes and simplify the recognition with different colors. When saving a project, a warning could appear if there are some unrecognized commands and they can be easily noticed because these commands are underlined in red.



3.8 Cyber Security

Into the Cyber Security tab, the user can easily manage all cyber security features that are described in the chapter Data Protection System, such as user management and firmware encryption.

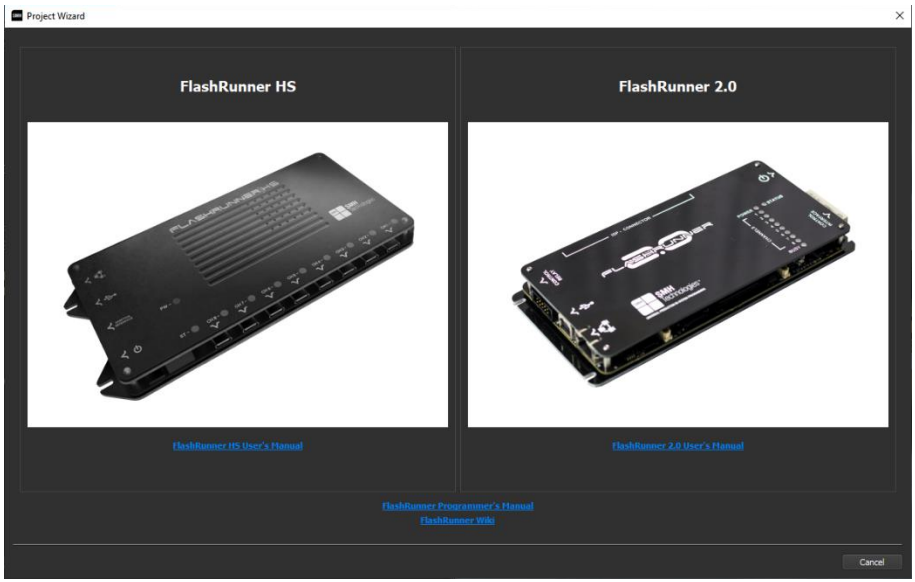
3.9 Wizard

FlashRunner collects all the user settings related to the programming sessions in text files called “projects”. Inside each project, you'll find a set of commands (all rows beginning with “#” character are commands, see ch. 4) which, of course, could be sent one by one through our interface library, through the serial port or through “Terminal” tool of FlashRunner Workbench. Having a single file including all these settings however brings several benefits to users, which they could save on a single file all the settings needed to program a specific device and running a complete programming cycle with only one click.

Wizard tool is one of the most innovative features of FlashRunner Workbench and lets users create a complete working project using only graphic items. A set of wizard pages will guide users toward all the specific device settings. Once completed, a project file will be created inside the FlashRunner data folder (which can be found or changed on Tools → Settings menu items, “Paths” tab) and must be uploaded to FlashRunner before executing it.

3.9.1 FlashRunner selection page

You can create a new project using File → New Project. If the FlashRunner Workbench is not connected to the FlashRunner the first wizard page will let you select the FlashRunner for which you want to create the project.

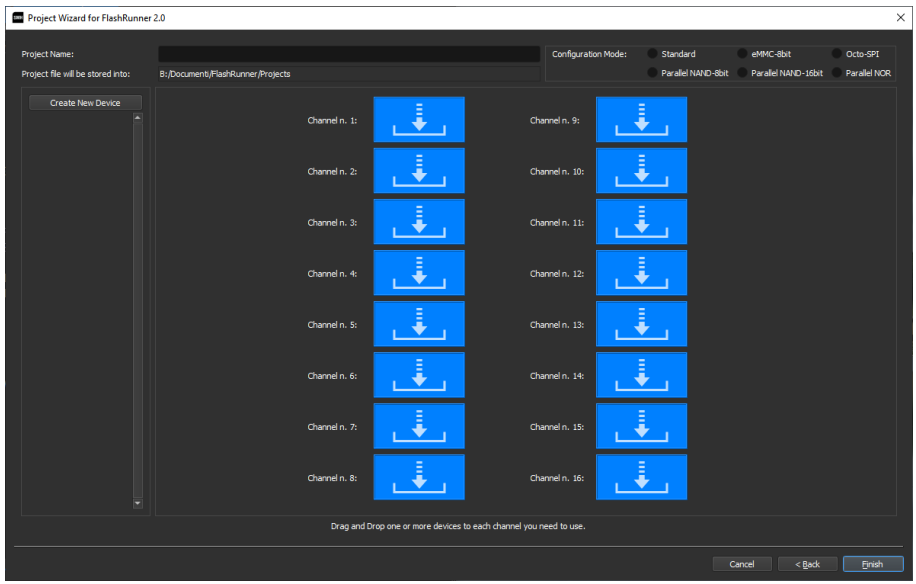


If the FlashRunner Workbench is already connected to the FlashRunner the wizard will show you the main page (see next chapter).

3.9.2 **Main page**

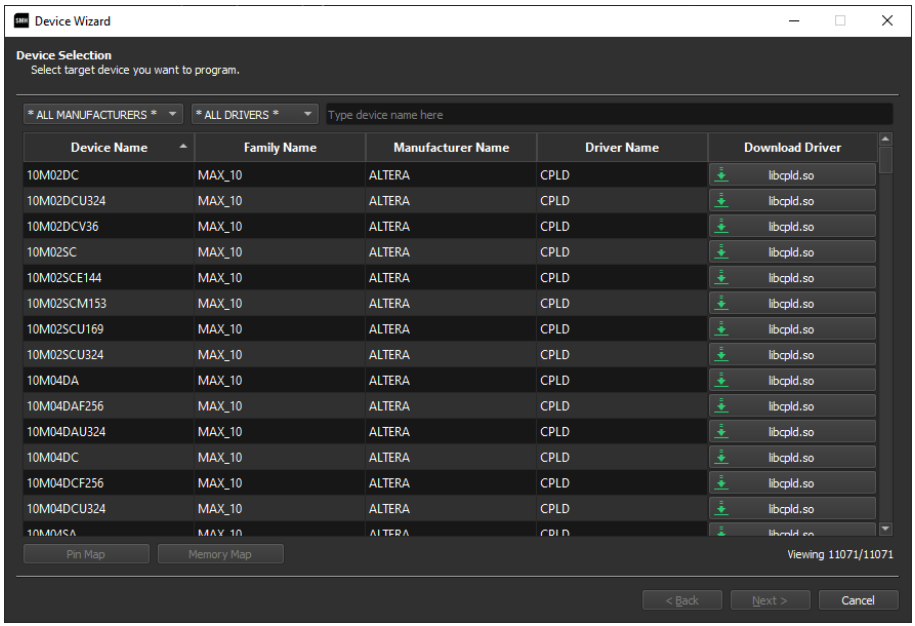
This is the main page of the wizard to create/modify a project. If FlashRunner Workbench is connected to the FlashRunner, you'll have a set of checkboxes enabled depending on how many channels are enabled/available on the FlashRunner.
The page is structured as follow:

- On the top a name for the project can be inserted.
- On the right-top it can be selected the *Configuration Mode* which depends on the device you want to program.
- In the centre there are the available channels.
- On the left can be created a device's project.

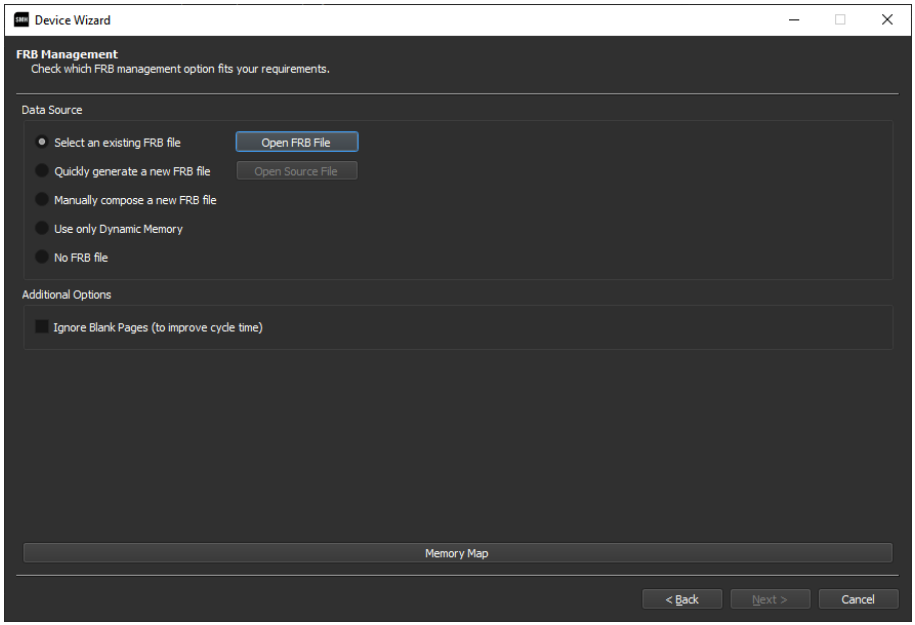


3.9.3 **Device selection page**

Clicking on “Create New Device” you can select which target device you want to program. Remember that each device needs its library, written in “Driver Name” column; make sure to have this library. You can download it locally on your PC the latest version by just clicking on the driver on the “Download Driver via FTP” column.



3.9.4 FRB Management page



On the FRB management page, there are some options for FRB creation and usage. First, you can choose the source:

- **Select an existing FRB file**: select an already created FRB file.
- **Quickly generate a new FRB file**: select an FRB source file and convert it with just a single click. This is the fastest way to convert a source file to FRB. The FRB is created and saved in the standard user data folder with the same filename as the selected source file (some special characters like '&' are not allowed).
- **Manually compone a new FRB file**: open a new window to access advanced features about FRB file creation. See the chapter "Advanced FRB Manager" for more details.

- **Use only Dynamic Memory**: this doesn't create any FRB file, it only uses dynamic memory. See chapter "Serial Numbering".
- **No FRB file**: this will set no FRB files.

In this window it is also possible to set the advanced option "**Ignore blank page**": this allows FlashRunner to skip pages without any data different from the blank value. Sometimes this feature can improve flashing times, according to the device's characteristics. Be careful that this option can not be used for all devices. If you are not sure please contact our Support Team.

At the bottom of the page, the user can also open and check the memory map of the selected device. The Memory map tool is described in detail on ch 3.14.

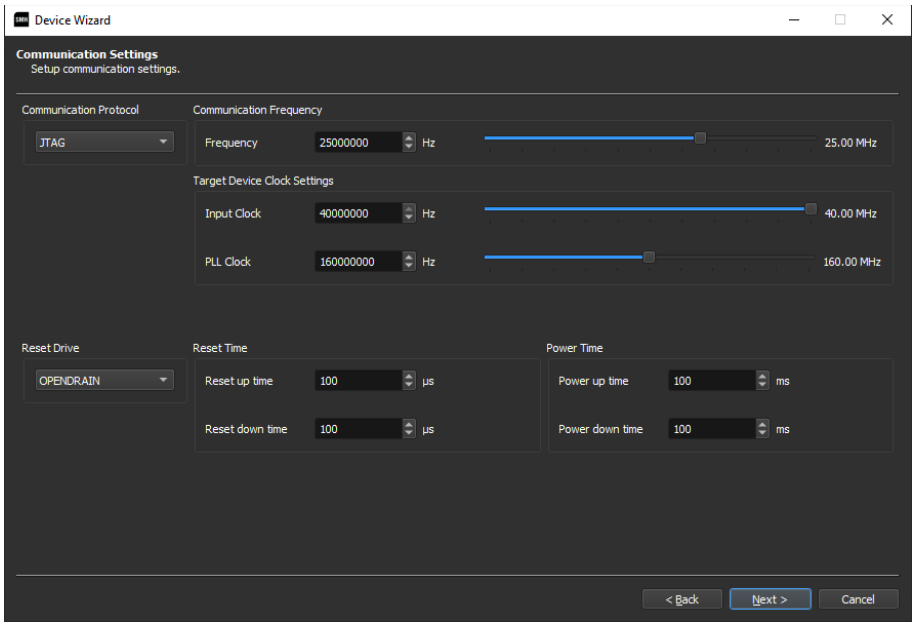
3.9.5 Communication settings page

This page has several configurations about the communication setting:

- **Communication Protocol**: JTAG, SWD, UART, SPI…
- **Communication Frequency**: in MHz
- **Input Clock**: frequency of the External Oscillator of the device. This field is not always present.
- **PLL Clock**: frequency of the PLL of the device. This field is not always present.
- **Reset Drive and Reset Time**: select the reset-up and reset-down time. The reset drive selects how the FlashRunner manages the reset line. OPENDRAIN means that the FlashRunner, after the reset, does not drive the line. PUSHPULL means the FlashRunner always drives the line. The choice is based on the hardware setup of the board.

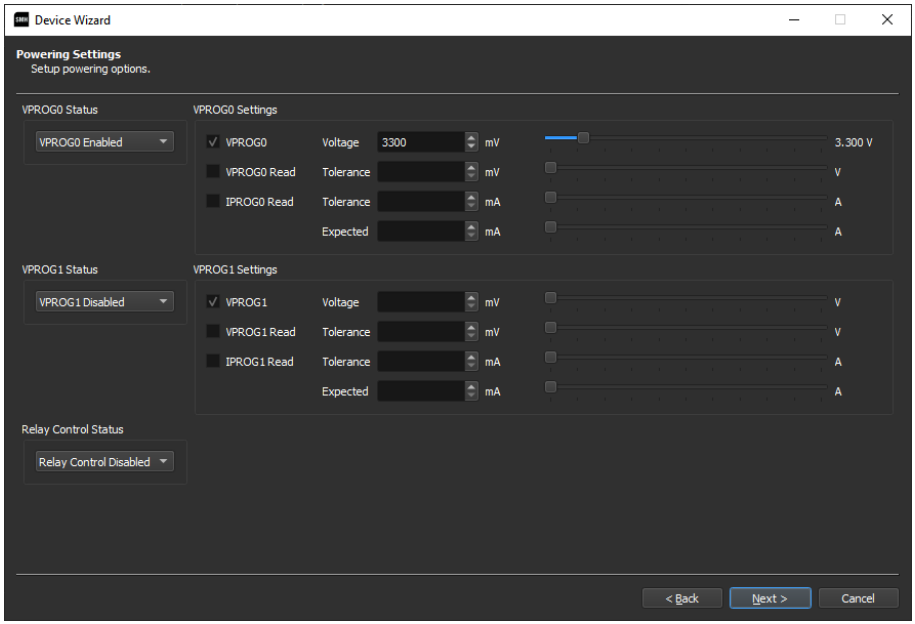
- **Power Time**: the FlashRunner can be used also as a power supply for the board. If there are big capacitors, it may be necessary to increase the power-up time. It is also possible to reduce this time to save a few milliseconds.

All these settings will enter as #TCSETPAR in the final project.



3.9.6 Powering settings page

This page allows the user to set the values of VPROG0 and VPROG1 and their tolerance values (#TCSETPAR values). The VPROG0 is also the logical voltage of the DIO signals. VPROG1, instead, can only be used as a power supply. The tolerance for the VPROG and IPROG monitoring can be set.
On this page, it is also possible to set the relay barrier usage, to manage automatically the opening and closing of the relays.

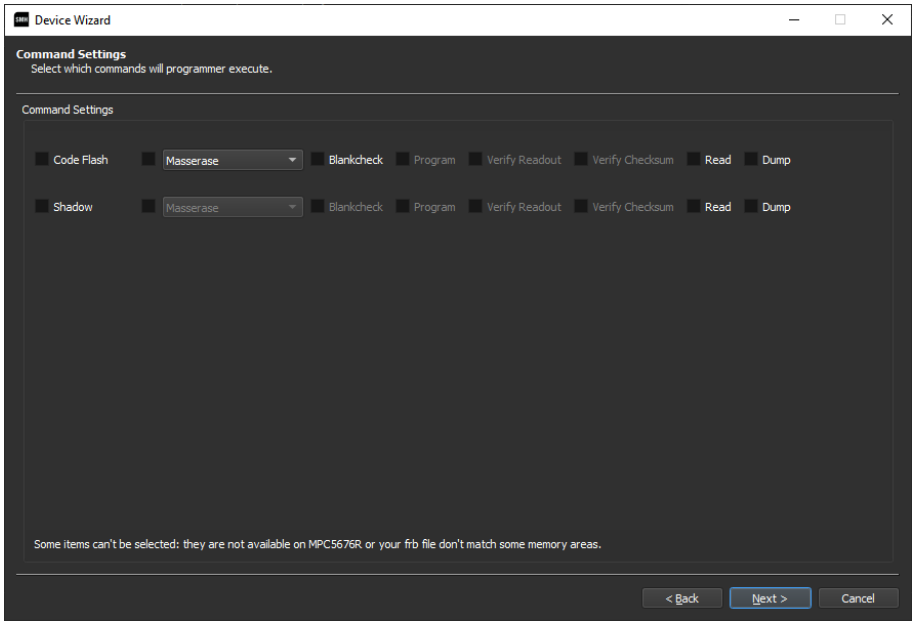


3.9.7 **Additional parameters page**

This page contains some additional parameters related to the device. This page is driver and device-dependent. To know more about the settings here presented, the WIKI of the driver can be consulted.

3.9.8 **Command settings page**

This page contains the standard commands related to the memory regions of the device. Some commands may be disabled according to the FRB file chosen. If there is no data present for some area, the Wizard does not allow to enable the Program and Verify operations. Moreover, the Checksum, Read and Dump operations may be not available for some devices.

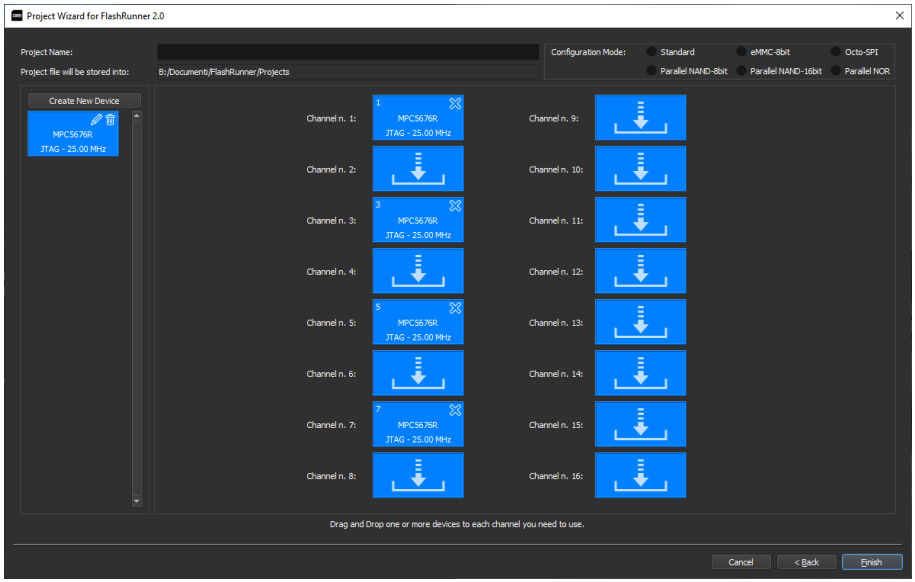


3.9.9 **Additional commands page**

This page contains some additional commands related to the device. This page is driver and device-dependent. To know more about the settings here presented, the WIKI of the driver can be consulted.

3.9.10 **Add the project to a channel**

Once the device is created, on the main page the user will see on the left the device. With drag and drop the user can insert the device into the desired channels.

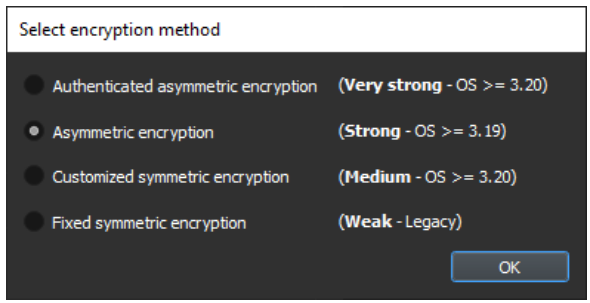


The user can create new devices and add them to a channel. Once the project creation is ended, the user can give a name to the project and click on finish.

3.10 Encrypt FRB (FRS)

An existing FRB could be encrypted through FlashRunner Workbench software. You simply have to click on the “Encrypt FRB” button from the tools menu and choose the FRB file you want to encrypt.
Otherwise, it is also possible to check the “Encrypt FRB” option while creating the FRB from the “Advanced FRB Manager”, this will directly generate the encrypted file without creating any additional unencrypted FRB file.

When encrypting FRB, it is possible to choose between four encryption methods as shown in the image below.



Authenticated asymmetric encryption
The first option is the strongest one and uses asymmetric encryption (available for OS versions higher than 3.20) and includes an authentication check to prevent data alterations. This encryption method requires some key exchange with the FlashRunner, so be sure to connect it to your computer. In case the FlashRunner cannot be connected to the computer that has to encrypt the file, you can get the keys from another computer connected to the FlashRunner, then you can export the keys and import them to your computer.

During the encryption process, you will be asked to choose for which FlashRunner SN you want to encrypt data, you can choose more than one SN and only those FlashRunners will be able to decrypt the data contained in the encrypted FRB.
Moreover, you will be asked to insert a secret key used to calculate the CMAC value which is used to authenticate the data.

Asymmetric encryption
The second option is still very strong and uses asymmetric encryption like the previous one, but it does not include the authentication check with CMAC (available for OS versions higher than 3.19).

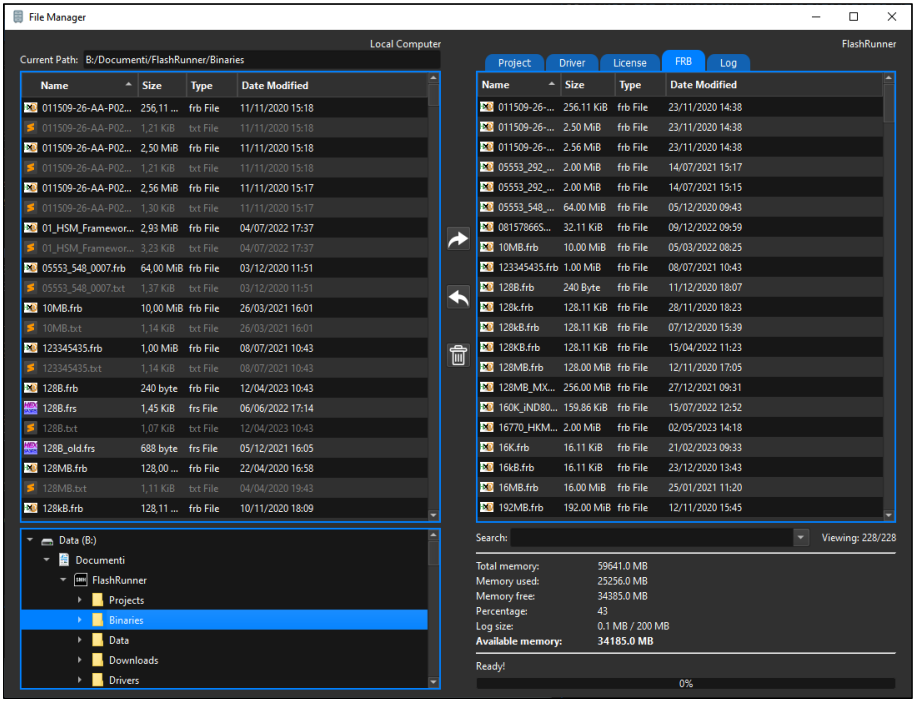
Customized symmetric encryption
The third option is still quite strong but it is also easier to use. In this case, the data are encrypted using a symmetric key generated from a password inserted by the user. Since it uses a symmetric key, the same password must be shared with FlashRunner. FlashRunner can only memorize one password at a time, so all the FRB files must be encrypted using the same password.

Fixed symmetric encryption
This is the legacy method that is discouraged for new applications and it is not compatible with the higher cyber security standards that are applied nowadays.
In case you are using this type of encryption, you can upgrade your FRS with a couple of clicks. In fact, You simply have to click on the “Encrypt FRB” button from the tools menu and choose the FRS file you want to upgrade.

It is also possible to generate or upgrade an encrypted FRB file from the command-line tool “FRB Converter”

Anyway, if you have a project which uses the original FRB file and you want to substitute it with its encrypted version, please modify the project file with the project editor at the #TPSETSRC command line. Then send both the project and FRS file to FlashRunner.

3.11 Advanced file manager



Advanced File Manager is an easy tool for updating or retrieving files to/from connected FlashRunner. On the left side you'll find your local resources, on the right side you'll find FlashRunner resources, in which only five folders are available and are shown as tabs.

As the names suggest, project files (.prj) must be copied in “Project” folder, drivers (.so) must be copied in “Drivers” folder, licenses (.lic) must be copied in “License” folder, FRB files must be copied in “FRB” folder, the log file is available in “Log” folder.

Once clicked a file from your local resources, please select a destination folder and then click “Send” button. Vice versa, select a file from FlashRunner folder and click “Get” button.

On the bottom of the right side you can also see the memory usage of your FlashRunner:

- **Total memory**: the amount of memory contained on the partition of the SD card.

- **Memory used**: the amount of memory that is currently used by user data.

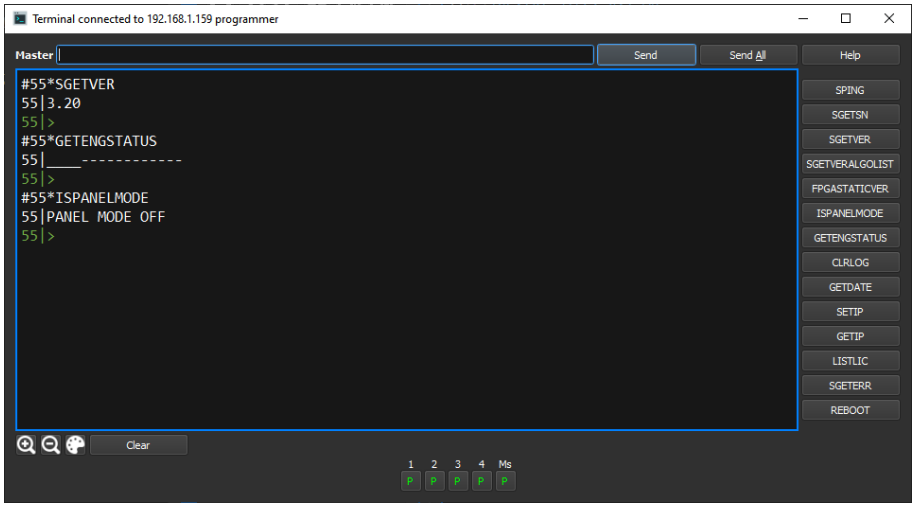
- **Memory free**: the amount of memory that is unused.

- **Percentage**: percentage of memory used by user data.

- **Log.txt size**: size of the log file, this can grow up to 200MB, then it will be automatically resized, but 200MB are always pre-allocated.

- **Available memory**: the amount of memory that can be used by user data. This is different from “Memory free” because it also considers the 200MB of the log file.

3.12 Terminal



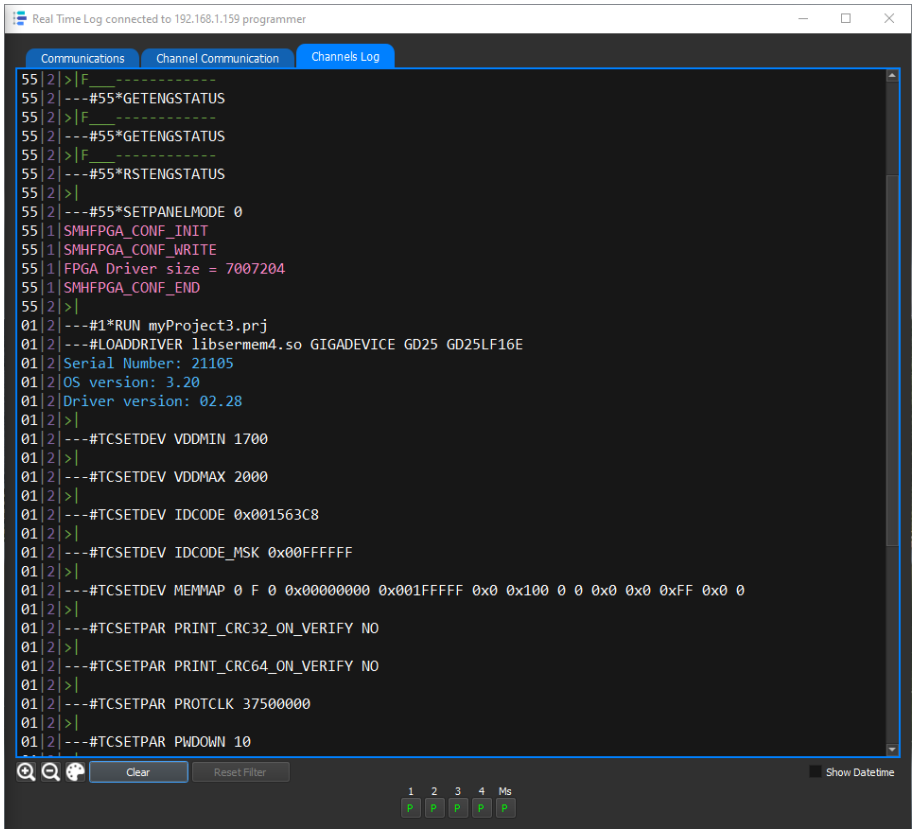
Host pc interacts with FlashRunner via synchronous serial communication. Host send commands and receive answers, for detailed information regarding communication syntax and available commands please see ch 4.

On the top left side of the window a label will show you which channel is selected. To send a command, write it inside the editable combo box at its right, finally, click the “Send” button. If you want to send a command to all channels simply click the “Send all” button. If you want to change the channel, please, select it with the button toolbar at the bottom right side.

Please note that the “**#**” character will be automatically added, if not entered.

On the left side, you have a list of buttons to quickly send the most common commands.

3.13 Log



The Real-Time Log feature shows the complete tracking of FlashRunner activity.
“Communication” tab will show full communication based on received commands, while “Channel communication” will filter out communication by single channel. You can select a channel by using the bottom right toolbar. “Log” tab will show all operation executed by FlashRunner, including commands included in project files.

Each row is composed with the following syntax:

```
<channel>|<log level>|<timestamp>|---<command sent>
<channel>|<log level>|<timestamp>|<command answer>
```

Example:

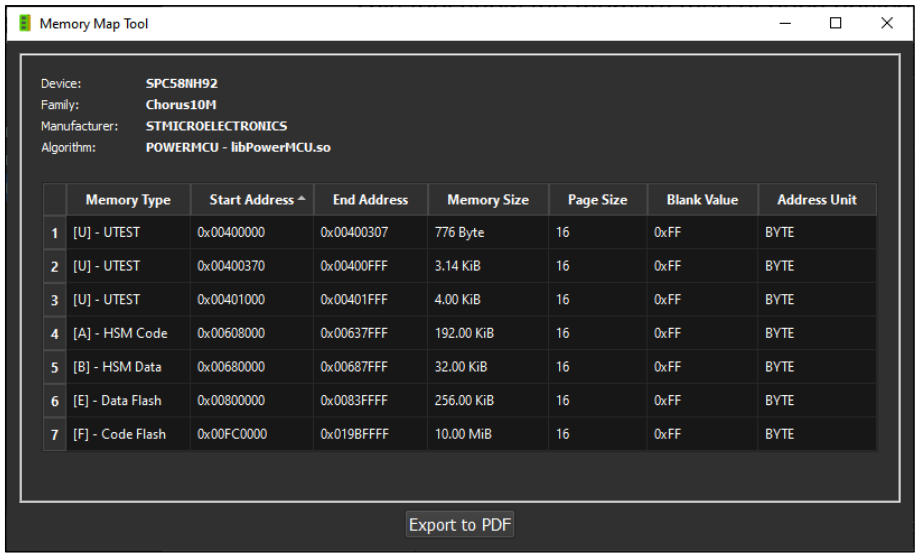
```
01|2|200331-16:28:10.437|---#TPCMD VERIFY F S
01|1|200331-16:28:12.306|Time for VERIFY F S: 1.87 s
01|2|200331-16:28:12.306|>|
```

Log Level is a number from 1 up to 6 and defines logging verbosity level. Level 1 is the more verbose, level 6 is the most coincise. You can change log verbosity with SETLOGLEVEL command (check ch 4.4.64).

Timestamp shows in which moment a command has been executed. Syntax used for timestamp is:
<year><month><day>-<hour>:<min>:<sec>.<millisec>
For each command sent there could be one or more answer lines.
It is also possible to hide timestamp by unticking the “Show Datetime” check box.

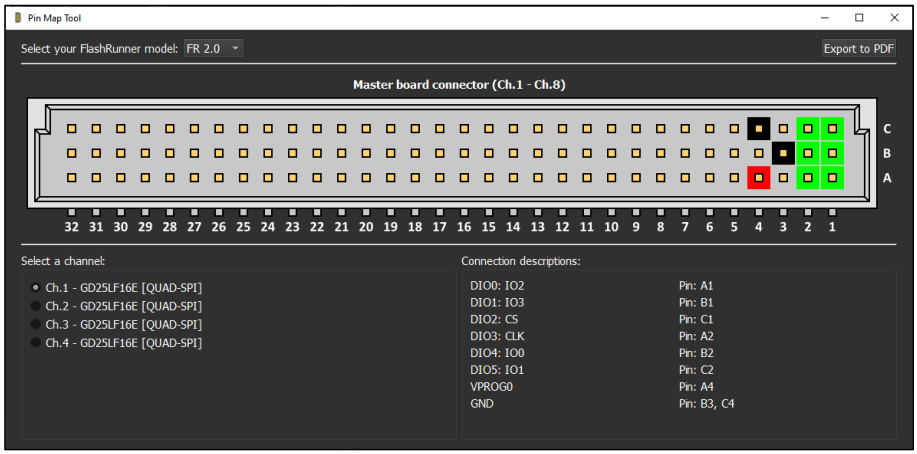
3.14 Memory Map tool

This tool show the memory map of each device included into the project. The interface is very simple and contains a lot of useful information about the memory of the device.



	Memory Type	Start Address	End Address	Memory Size	Page Size	Blank Value	Address Unit
1	[U] - UTEST	0x00400000	0x00400307	776 Byte	16	0xFF	BYTE
2	[U] - UTEST	0x00400370	0x00400FFF	3.14 KiB	16	0xFF	BYTE
3	[U] - UTEST	0x00401000	0x00401FFF	4.00 KiB	16	0xFF	BYTE
4	[A] - HSM Code	0x00608000	0x00637FFF	192.00 KiB	16	0xFF	BYTE
5	[B] - HSM Data	0x00680000	0x00687FFF	32.00 KiB	16	0xFF	BYTE
6	[E] - Data Flash	0x00800000	0x0083FFFF	256.00 KiB	16	0xFF	BYTE
7	[F] - Code Flash	0x00FC0000	0x019BFFFF	10.00 MiB	16	0xFF	BYTE

3.15 Pin Map Tool



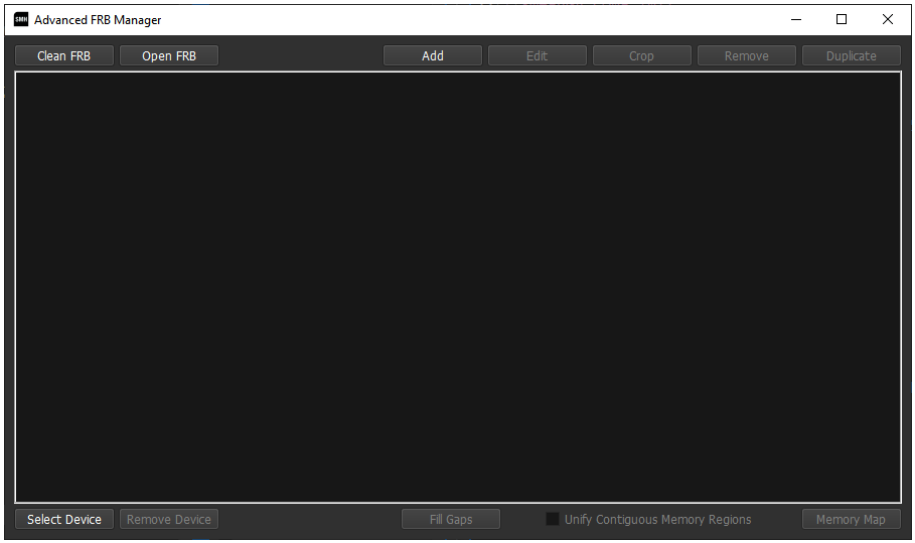
PinMap tool is a handy feature that helps users to do cable wirings from the target device to FlashRunner ISP connector. On the top you can select the FlashRunner (2.0, NXG or HS) and see the corresponding PinMap. Clicking on one of the channels available in list will load a table on the right side of the window, which lists all signals involved for device connection on that specific channel. Once clicked, related pins will become coloured and clicking on one of them will highlight the related signal in the signals table. Please note that FlashRunner has one or two ISP connectors based on product version: FlashRunner versions with 8 or less active channels will have only one ISP connector, FlashRunner with more than 8 active channels will have two ISP connector. Please pay attention to the connector indication on top of signals table: first 8 channels are related to the master board connector, channel 9 up to 16 are related to the slave board connector.

3.16 Advanced FRB Manager

The Advanced FRB Manager is a tool to create an FRB file (i.e. FlashRunner Binary) that contains all the source files (more than one is allowed) needed to program the target device. You can find this tool via Project Wizard or by selecting Tools → FRB Manager.
Attention: converting an FRB through the Advance FRB Manager allows you to create an FRB without a device Memory Map as a reference; hence the data position of the source file can not be checked.

FRB Manager can convert the most common source file formats: RAW Binary; Intel Hex and Motorola SREC.
Advanced FRB setup will enable full features to users to let them compose their FRB file. Users can import multiple source files, edit single blocks start address and size, remove blocks and add “fill” or “variable data” blocks.

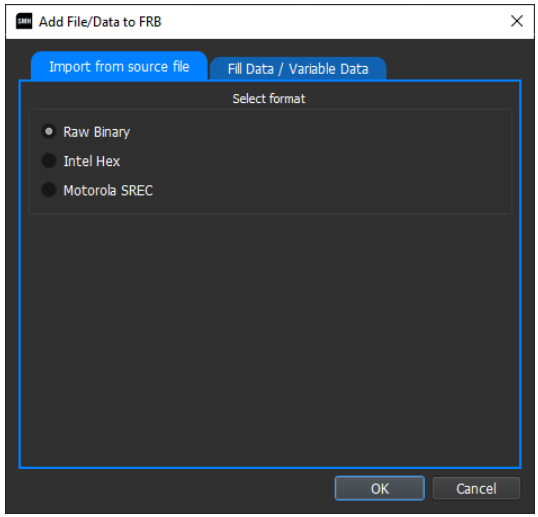
After opening the window (see the image above), the user can decide to create a new FRB by clicking the “New FRB” button or to edit an existing FRB by clicking the “Open FRB” button.
After that, the buttons on the left side will be activated and the user will be able to: add, edit, duplicate or delete a block of the FRB. The operations to edit, duplicate or delete a block will be active only after selecting a block from the list.



At the bottom of the window, the user can set the destination file and launch the conversion when the work on the FRB file is completed.

It is also possible to Select a Device (bottom left) to enable the check of the source file position according to the memory map of the device. This operation can be skipped.

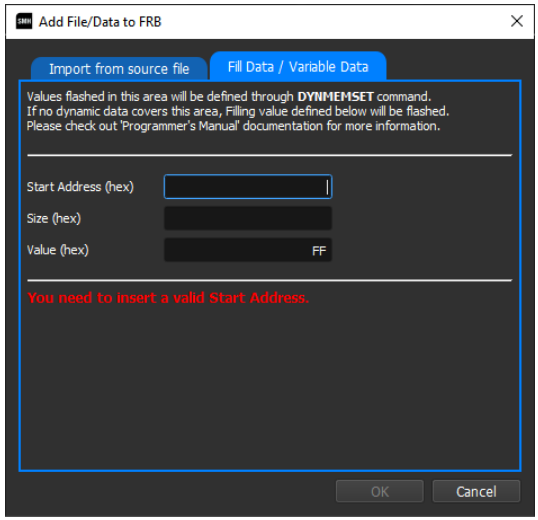
3.16.1 **Add data to FRB: import from source file**



Multiple source files can be added (also of different formats). Clicking on "**Add**" source file can be selected, choosing the format (binary, Intel Hex …). The tool automatically recognizes if the file selected is compatible with the memory map of the device. If one or more blocks are read, it means that they have data outside the programmable address of the device. If two or more blocks are blue, it means that there is an overlap of data in the same addresses (more blocks have data in the same places). Green means that everything is correct.

When choosing Motorola SREC or Intel Hex, the user should also choose the encoding type: if data has been defined by words or by bytes. If you are not sure about what to select, just use the "Byte encoding" option.
Data parsing will be achieved by reading and merging all the source file rows which define adjacent data areas, each disjointed block will define a new data area and will be placed in a new row (new block).

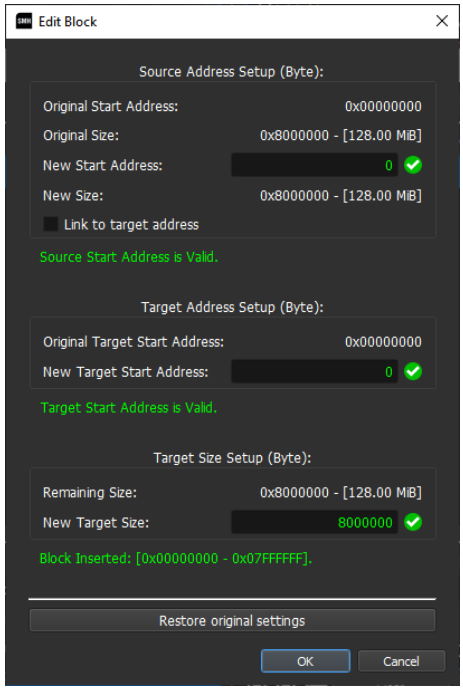
3.16.2 **Add data to FRB: Fill Data / Variable Data**



On the “Add “ window there is a second tab called “Fill Data / Variable Data”, the user can add a new block to FRB which contains the same value for each byte.
As you can see in the figure above, the user can set the start address, the size and the fill value of the block.
The new block will not impact total FRB size and could also overlap existing data.

The same procedure is valid also for variable data, in fact, the user should just choose the value that corresponds with the blank values of the device memory.
This will be used for dynamic content definition during target device programming (please check ch 6 for detailed information).

3.16.3 **Edit FRB block**



Once the user adds some data inside the new FRB file, some data rows inside the input data table will appear. If a data block overlapping occurs, two blocks involved are highlighted and the user should solve the conflict or explicitly decide to leave this conflict unresolved.
In order to modify a single data block, you need to select it on the input data table and then click on the “Edit” button, a new window will appear, like in the image above.

Data block overlapping conflicts will be solved following this rule: the last data block (in row order) will overwrite overlapping data of the first data block.

From the new window, the user will be able to edit the source start address, the target start address and the size.

If you use have selected a device, the memory map will appear at the bottom of the window. This helps to place the block in a proper memory region.
If the chosen settings don’t fit any device memory regions, a warning will appear. As a result, data blocks that don't fit any device memory region will not be programmed at all on target device flash memory.

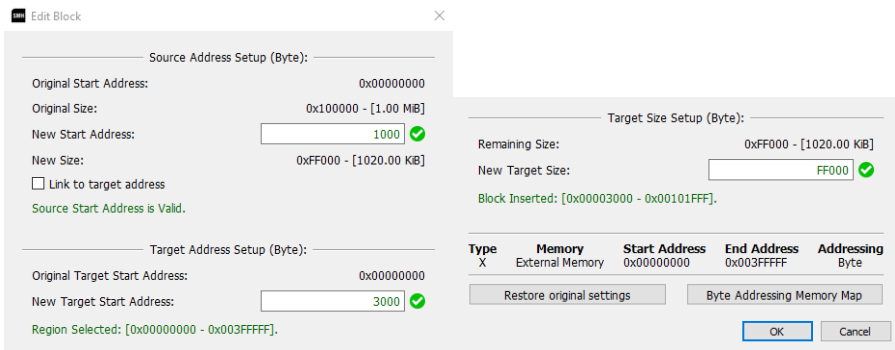
Source address Setup
This text field defines the address of the source file from which will start the block. This is only related to the source file.
The default value is the first address of the block.

Target address Setup
This text field defines from which target device address will start block. This is the actual address from which the FlashRunner will start programming the target device.
The default value corresponds with the source address.

Target Size Setup
This text field defines how many bytes will compose the block. This corresponds to the number of bytes which will be programmed on the target device by FlashRunner.
The default value is the full block length.

***Example* (see image below):**
The block as default starts from 0x00 and flashes data into the device from 0x00.
Setting 0x1000 in the “Source Address Setup” means that the data from 0x1000 of the source file are going to be flashed from address 0x00 of the target device.

The last operation changes the data available for the block. Originally they are 0x100000, now 0xFF000. For this reason, the “Target Size Setup” has to be changed to 0xFF000. This field means that the bytes to be considered for that block are 0xFF000.
The ”Target Address Setup”, instead, changes the target device address. 0x3000 means that the data are going to be flashed from address 0x3000 of the device, instead of 0x00.



3.16.4 Other Options

Other options are present in the FRB Manager window:

1. **Crop**: the start address and the size of the block can be changed. Changing the start address, all the data before that address are going to be erased. Changing the size, all the data after are going to be erased. This operation is irreversible.
2. **Remove**: the block can be removed from the creation of the FRB.
3. **Duplicate**: the block is duplicated. An overlap will be formed.
4. **Fill Gaps:** merge source blocks where the distance between them is less than the program page size of the memory selected; the value used to fill the gap is the blank value (see Memory Map). This can optimize FlashRunner performances when too many blocks are present. This operation is irreversible.
5. **Unify Contiguous Memory Regions:** treats two or more contiguous memory regions as a unique region. The FRB creator by default doesn't accept a block to cross different memory regions and it is highlighted in red. This option removes this limit.

4 FlashRunner Commands

4.1 How to control FlashRunner

FlashRunner is set up and controlled via ASCII-based commands. FlashRunner can receive and execute commands in two ways:

- Over a USB or Ethernet connection **(Host mode)**;
- Via signals received on its “Control connector” which can select and run a specific project stored in its internal storage memory **(Standalone mode)**.

In the first case, FlashRunner is controlled by a host system; in the latter case, FlashRunner works in standalone mode and it is fully autonomous inside an integrated production system.

4.1.1 Host Mode

In Host mode, commands are sent from the host system to FlashRunner:

- By using a TCP/IP or Serial command-line utility (like Termite© on Microsoft Windows©);
- By using any programming language that can send and receive data to/from a host system COM port or Ethernet port (i.e. Microsoft Visual C++/Visual Basic, National Instrument LabView/LabWindows, etc.). An Interface Library is available upon which you can build your application.
- You can use the FlashRunner Workbench software to send commands to the programmer.

Note for TCP/IP:
FlashRunner factory IP address is 192.168.1.100 and data is exchanged on port 1234.

4.1.2 Standalone Mode

In Standalone mode, FlashRunner does not need a connection to a host system. A group of control lines (*SEL[4..0]* in the "*CONTROL*" Connector) determines which of the 32 available projects stored in FlashRunner memory must be executed.
You can check if the project execution has succeeded or not and you can check the failed channels in case of failure.

4.2 Command Syntax

4.2.1 Sending a Command

Each command, except project-specific directives shown in table 5.2, must start with the **#** character (FlashRunner Terminal tool automatically adds this character). A command can be sent to:

- Master engine
- A single-site engine
- All engines (Master engine and site engines)
- All site engines
- A subset of site engines

Each command has different "coverage", described in chapter 4.3. For example, some commands can be sent only to the master (like ***#SPING***), and others only to the site engines (like ***#RUN***).

Each command is mainly composed of the following two parts:

1. Command name. Example: ***#RUN***
2. One or more parameters, each separated by a space. Example: ***#RUN example.prj example.frb***

Note:
The length of each command's parameter is at maximum 40 characters. All parts of the command are case sensitive.

When sending a command, the **#** character is always used as the first character of the string.

Single Site Command:
A command sent to a single engine begins with **#** character followed by **<channel number>** (decimal value of the channel), followed by ***** character, followed by the command, a Carriage Return character and a final Line Feed character. Channels' number starts from 1 up to 16, the master engine is 55.

Example:

Send a command to channel 7:

```
#7*RUN example.prj
```

Send command to the master:

```
#55*SPING
```

All Site Command (site engines and master):
A command sent to all engines in parallel begins with **#** character, followed by the command, a Carriage Return character and a final Line Feed character:

Example:

```
#RUN example.prj
```

Subset of site engines:

A command sent to a subset of site engines begins with **#** character followed by **<engine mask>**, followed by **|** character, followed by the command, a Carriage Return character and a final Line Feed character. The **<engine mask>** is a decimal number that identifies bitwise channels on which command must be executed.

```
Example:
  Send a command to channels: 8, 5, 3, 2, 1.
  Engine Mask: 0b10010111 = 151
  #151|RUN example.prj

  Send a command to all channel, but not the
master.
  Engine Mask: 0b11111111 = 255
  #255|RUN example.prj
```

FlashRunner Workbench software can send commands via the Terminal tool, which automatically adds **#<channel number>***. Before sending a command, please click on the bottom right side of the window the channel for which you want to send the command. See chapter 3.11 for more details.

Project files contain ENGINEMASK pseudo-command which already defines which engines will be involved for the following commands. For this reason, commands inside a project file don't need channel prefix. Thus, inside a project a command will be **#** character, followed by the command, a Carriage Return character and a final Line Feed character.

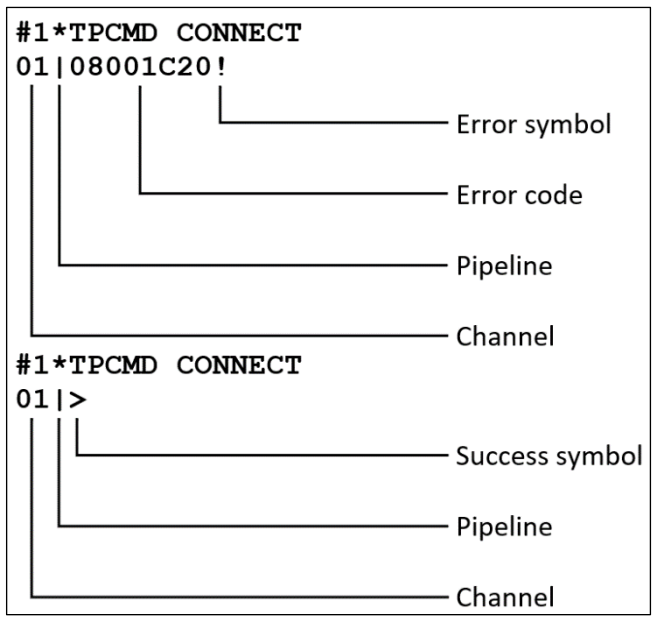
```
Example:
  #TPSTART
  #CONNECT
```

4.2.2 **Receiving the Answer**

After receiving a command from the host system and executing it, FlashRunner responds with an answer string. The answer string is composed of zero or more response characters, followed by one result character, followed by a final Line Feed. The character of the result is:

- > if the command has been executed successfully or
- ! if the command generated an error.

Below are two examples of answer (with and without error):



When a FlashRunner command executes successfully, FlashRunner typically answers just with the engine number followed by **|** character, followed by **>** character, see figure above, (unless the command requires data to be returned).

When a FlashRunner command generates an error, FlashRunner answers with an eight-digit hexadecimal error code followed by the "**!**" character (see figure above).

4.2.3 **Numeric Parameters**

Every numeric command parameter can be expressed either in decimal or hexadecimal format. Hexadecimal numbers must be preceded by the "**0x**" symbol. The figure below shows three examples of usage of the **DYNMEMSET** command to write two bytes on FlashRunner dynamic memory. These two examples below are equivalent:

```
#DYNMEMSET 0x8E0400 0x2 0x00 0x0F
#DYNMEMSET 9307136 2 0 15
```

Numeric parameters returned by FlashRunner as command answer (CRC, memory data, error codes, etc.) are expressed in hexadecimal or decimal format, depending on the case.

4.3 Command Summary

The following table summarizes all of the FlashRunner commands available. Each command is fully described in the "*Command Reference*" section in the next chapter. The columns are the following:

- **Command Name**: the name of the command.
- **Description**: a very brief description of the command.
- **Scriptable**: describes if the command can be added inside a project file or not.
- **Site**: describes if the command will work on channel engines ("*S*"), for the master engine only ("*M*") or both ("*M+S*");
- **Permission**: describes the default permission level of the command when the Admin/Guest management is enabled. This level can be changed with **SETCMDLEVEL** command or using the latest version of Workbench software. In "*Command Reference*" you can check if the permission level is changeable.

Command Name	Description	Scriptable	Type	Permission
System Commands				
CLRERR	Clear the errors stack	NO	M+S	GUEST
CLRLOG	Clear the log file	NO	M	ADMIN
ECHO	Echo a string on the log	YES	M+S	GUEST
FSCMAC	Return the CMAC value of a file	NO	M	GUEST
FSCOUNT	Count the number of files in a folder	NO	M	GUEST
FSCRC	Return the CRC32 value of a file	NO	M	GUEST
FSEXIST	Check if a file does exist	NO	M	GUEST
FSGETCONTROL	Read control interface value	NO	M	GUEST
FSLS	List files	NO	M	GUEST
FSLS2	List files with more details	NO	M	GUEST
FSRM	Remove file	NO	M	ADMIN
FSSETCONTROL	Set control interface value	NO	M	ADMIN
GETDATE	Return the FlashRunner date/time	NO	M	GUEST
GETFREEMEM	Show details about memory usage	NO	M	GUEST
GETIP	Return FlashRunner IP information	NO	M	GUEST
GETLOGLEVEL	Get the log verbosity level	NO	M	GUEST
GETVPROG	Read a power line value	NO	S	GUEST
HSMEMFORMAT	Erase all data on HS Memory	NO	M	ADMIN
HELP	Show driver help table	NO	S	GUEST
ISMEMENOUGH	Check if there is enough memory	NO	M	GUEST
ISPANELMODE	Return FlashRunner working mode	NO	M	GUEST
REBOOT	Reboot programmer	NO	M	GUEST
SGETAMSN	Return Active Module serial number	NO	S	GUEST
SGETSN	Return FlashRunner serial number	NO	M	GUEST
SGETVER	Get OS version	NO	M	GUEST
SGETVERALGO	Return driver version	NO	M	GUEST
SGETVERALGOLIST	Get the entire driver list with the version	NO	M	GUEST
SETDATE	Get the actual FlashRunner date/time	NO	M	ADMIN
SETDIO	Set output state of DIO	YES	S	GUEST
SETIP	Set FlashRunner IP information	NO	M	ADMIN
SETLOGLEVEL	Set log verbosity level	NO	M	ADMIN
SETPANELMODE	Change FlashRunner working mode	NO	M	GUEST
SETSERIALBAUDRATE	Change Serial communication speed	NO	M	GUEST

Command Name	Description	Scriptable	Type	Permission
SHA256	Calculate sha256 of a file	YES	M+S	GUEST
TESTVPROG	Set up a defined value on VPROG lines	NO	S	GUEST
System Security Commands				
GETADMINTIMEOUT	Get Admin session timeout	NO	M	GUEST
GETCMDLEVEL	Get command permission level	NO	M	GUEST
GETCOUNTER	Get flashing counter	NO	M	GUEST
GENCRYPTOKEY	Generate keys used for encryption	NO	M	PROTECTED
GETPUBKEY	Get the public key used to encrypt data	NO	M	GUEST
LOGIN	Login into Admin account	NO	M	GUEST
LOGOUT	Logout from Admin account	NO	M	ADMIN
SETADMINPWD	Set Admin password	NO	M	PROTECTED
SETADMINTIMEOUT	Set Adnub session timeout	NO	M	PROTECTED
SETCMDLEVEL	Set command permission level	NO	M	PROTECTED
SETCOUNTER	Set flashing counter	NO	M	PROTECTED
SETFRSPWD	Set the password to decrypt FRS	NO	M	PROTECTED
UNSETADMINTIMEOUT	Remove Admin session timeout	NO	M	PROTECTED
WHOAMI	Get current logged user	NO	M	GUEST
System Status Commands				
GETENGSTATUS	Get actual engine status	NO	M	GUEST
RSTENGSTATUS	Reset engine status	NO	M+S	GUEST
SGETERR	Return detailed error information	NO	M+S	GUEST
SPING	Ping FlashRunner	NO	M	GUEST
Dynamic Memory Commands				
DYNMEMCLEAR	Clears dynamic memory	YES	S	GUEST
DYNMEMCLEARHEADER	Clears dynamic memory crypto header	YES	S	GUEST
DYNMEMREAD	Read dynam memory	NO	S	NONE
DYNMEMSETHEADER	Defines dynamic data crypto header	YES	S	GUEST
DYNMEMSET	Defines dynamic data	YES	S	GUEST
DYNMEMSET2	Defines dynamic data	YES	S	GUEST
DYNMEMSETW	Dynamic data (word addressing)	YES	S	GUEST
DYNMEMSETW2	Dynamic data (word addressing)	YES	S	GUEST
FRB Management Commands				
FRBREADCMAC	Check and return FRB CMAC value	NO	M+S	GUEST
FRBREADCRC	Check and return FRB CRC value	NO	M+S	GUEST

Command Name	Description	Scriptable	Type	Permission
FRBREAD	Read FRB content	NO	S	NONE
License Management				
LISTLIC	FlashRunner licenses list	NO	M	GUEST
LISTLICAM	Active Module installed license list	NO	S	GUEST
LICERASE	Erases all Active Module licenses	NO	S	ADMIN
LICINSTALL	Install a license in an Active Module	NO	S	ADMIN
Project Programming Commands				
CRC	CRC of TCSETDEV section	NO	S	GUEST
DELAY	Stop engine operation for an interval	YES	S	GUEST
FORCEDRIVER	Force a specific driver name	YES	S	GUEST
GETPROGRESSBAR	Return the programming percentage	NO	M	GUEST
LOADDRIVER	Set target device	YES	S	GUEST
PROGRESSBAR	Set Progress Bar	YES	S	GUEST
RLYCLOSE	Closes the specified relay	YES	S	GUEST
RLYOPEN	Opens the specified relay	YES	S	GUEST
SETMUX	Drive demuliplexer	NO	M	GUEST
SHUFFLEDIO	Change logic/physical DIO map	YES	S	GUEST
SHUFFLEDIO_GETMAP	Get the actual DIO Map	YES	S	GUEST
TCSETDEV	Set target device information	YES	S	GUEST
TCSETPAR	Set target device parameter	YES	S	GUEST
UNFORCEDRIVER	Remove the forced driver	YES	S	GUEST
UNLOADDRIVER	Reset target before updating a driver	YES	S	GUEST
VOLTAGEMONITOR	Set Voltage Monitor	YES	S	GUEST
WATCHDOGFEED	Set square wave on selected channel	YES	S	GUEST
Target Programming Commands				
TPCMD	Executes programming command	YES	S	GUEST
TPEND	Ends programming sequence	YES	S	GUEST
TPSETDUMP	Set data destination	YES	S	GUEST
TPSETSRC	Set data source	YES	S	GUEST
TPSTART	Starts programming sequence	YES	S	GUEST
TPUNSETDUMP	Unset data destination	YES	S	GUEST
TPUNSETSRC	Unset data source	YES	S	GUEST
Script Execution Commands				
RUN	Executes the specified script	NO	S	GUEST

Command Name	Description	Scriptable	Type	Permission
Pseudo commands				
`!ENGINEMASK`	Select an engine subset	YES	S	-
`!CRC`	CRC calculation	YES	S	-

4.4 Command Reference

Each FlashRunner command is listed alphabetically and explained in the following pages.

The following conventions are used in the documentation of FlashRunner commands:

- Uppercase text indicates a command name or a command option that must be entered as shown.
  E.g. **SGETVER**
- Lowercase text between **<>** indicates a command parameter name.
  E.g. **TPSETDUMP <filename>**
- Lowercase text between **[]** indicates an optional command parameter.
  E.g. **TPCMD <command> [par1] [par2] ... [parn]**
- A vertical bar indicates a choice between two or more command options.
  E.g. **TPCMD MASSERASE F|E|C**

Please note that, except from examples, all the commands are provided without the **#<ch>*** prefix.

4.4.1 CRC

Command syntax:

`CRC <crc>`

Scriptable: No

Available on: Site engines

Permission:

Default: Guest
Changeable**:** No

Parameters:

crc: crc value

Answer data:

Success: none
Error: the error code

Description:

Set the CRC value of the precedent TCSETDEV section. The value can be taken from the project CRC pseudocode. This command can be used by the customer when he is using the DLL to send each command one by one.

Example:

```
#1*CRC 0x47546395
01|>
```

4.4.2 **CLRERR**

Command syntax:

CLRERR

Scriptable: No

Available on: Master and site engines

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Clears the error stack.

Example:

```
#55*CLRERR
55|>
```

4.4.3 CLRLOG

Command syntax:

```
CLRLOG
```

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

None.

Answer data:

Success: none
Error: the error code

Description:

Clears the log file.

Example:

```
#55*CLRLOG
55|>
```

4.4.4 DELAY

Command syntax:

DELAY <ms>

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

ms**:** milliseconds to wait

Answer data:

Success: none
Error: the error code

Description:

Insert a <ms> delay between FlashRunner operations during the flashing of a device.

Example:

```
#1*DELAY 2000
01|>
```

4.4.5 DYNMEMCLEAR

Command syntax:

`DYNMEMCLEAR [start_addr] [len]`

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

`start_addr`: address to start clearing dynamic memory data
`len`: bytes number to clear

Answer data:

Success: none
Error: the error code

Description:

Clears the data set on the dynamic memory area. In case no parameters are set, then all dynamic memory is cleared.

Example:

Clear all the dynamic memory:

```
#1*DYNMEMCLEAR
01|>
```

Clear only selected dynamic memory:

```
#1*DYNMEMCLEAR 0x0 0x10
01|>
```

4.4.6 DYNMEMCLEARHEADER

Command syntax:

DYNMEMCLEARHEADER

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Clears the crypto header of the dynamic data. All dynamic data sent after this command are meant to be unencrypted until another crypto header is set.

Example:

```
#1*DYNMEMCLEARHEADER
01|>
```

4.4.7 DYNMEMREAD

Command syntax:

`DYNMEMREAD <address> <length> [page_size] [fill_val]`

Scriptable: No

Available on: Site engines only

Permission:

Default: None
Changeable**:** Yes

Parameters:

address: the address where to start reading
length: length of the read, max 128 bytes
page_size: align data to a page size
fill_value: fill value when there are no data in the Memory

Answer data:

Success: the data inside the Dynamic Memory
Error: the error code

Description:

Read the data inside the loaded Dynamic Memory (with *DYNMEMSET* or equivalent commands). Max 128 bytes for each read. The data can be aligned to a specific page size and a fill value can be set (default is *0xFF*). This command is disabled by default for security reasons, since using this you could also read decrypted dynamic data. To enable it the User Management has to be enabled and the permission level changed. Please, refer to the chapter “Data Protection System”.

Example:

```
#1*#DYNMEMSET 0x00 0x05 0x00 0x01 0x02 0x03 0x04
01>
#1*DYNMEMREAD 0x02 0x03
01|[Addr=0x00000002] 0x02 0x03 0x04
01>
```

4.4.8 DYNMEMSET

Command syntax:

`DYNMEMSET <start_addr> <len> <data_0> ... <data_n>`

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

start_addr**:** address of the target device to start writing data to
len**:** bytes number to write (max. 16)
data**:** bytes to write

Answer data:

Success: none
Error: the error code

Description:

Writes **`len`** bytes to the dynamic memory starting at address **`start_addr`**. For devices that define size in words (check it out on the Memory Map tool of FlashRunner Workbench), use the command *DYNMEMSETW*.
Dynamic memory is a special memory area that retains its contents only as long as FlashRunner is powered. Both hexadecimal and decimal digits are accepted. More *DYNMEMSET* can be sent defining different memory areas.
Please refer to chapter 6 for a detailed description.

Example:

```
#1*DYNMEMSET 0x0000 4 0x00 0x01 0x02 0x03
01|>

Note: Address 0x00 -> value 0x00
      Address 0x01 -> value 0x01
      …
```

4.4.9 DYNMEMSET2

Command syntax:

`DYNMEMSET2 <start_addr> <len> <data stream>`

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

start_addr**:** address of the target device to start writing data to
len**:** number of bytes to write, max 500 (see below)
data stream**:** bytes stream to write defined by hexadecimal digits

Answer data:

Success: none
Error: the error code

Description:

Writes **`len`** bytes to the dynamic memory starting at address **`addr`**.
Devices which defines size in words (check it out on Memory Map tool of FlashRunner Workbench), see the command *DYNMEMSETW2*.
Dynamic memory is a special memory area that retains its contents only as long as FlashRunner is powered. More *DYNMEMSET* can be sent defining different memory areas.
Like all commands, the maximum number of characters for a line is 1024. This means that, depending on the first part of the command, **`len`** cannot be higher than 500.
Please refer to chapter 6 for a detailed description.

Example:

```
#1*DYNMEMSET2 0x0000 4 AB123402
01|>

Note: Address 0x00 -> value 0xAB
      Address 0x01 -> value 0x12
      …
```

4.4.10 DYNMEMSETHEADER

Command syntax:

DYNMEMSETHEADER <crypto_header>

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

crypto_header: bytes stream of the crypto header to use to decrypt the dynamic data defined by hexadecimal digits. The crypto header is 768 digits long (384 bytes)

Answer data:

Success: none
Error: the error code

Description:

Sets the crypto header of dynamic data. Dynamic data sent after this command are meant to be encrypted until the crypto header is cleared with *DYNMEMCLEARHEADER*.

Example:

```
#1*DYNMEMSETHEADER
736ADE84C5A1B1D9D27749E1D01ED3C67895170B5461243A2D771BA6E
8E655C907551C306D1CCCC99445159EB21324D543E0510C2C7783D332
E7F36283E55097F2A8681C5C4D890573F89000583504C47F9FC1BE945
7F88D236A7720C5E5996A6E50AD524715F997992D513F12409B29BADA
6EE736CF201D56FDFF1BF5965B37F91CBFD1992E9FB2BFE0D7804EA45
DF417EB8FEF20DF88A740E8C435D35E8540680DF2C7D5F3778E35903D
E7E1F055B1B2CDCE52F3FE73FE97952467E85D5B890957CE31DDB58D4
46E8BFFF06343ACBAFB0B51A74B43782889EBA49B9E795FB656B197D2
169E8105435EE54E6EE42801DC9C1F422AB90AE237AAD35D6BAD4CD39
8ABC3BF6DF97ACB42106B02A996128E2B4065421308F209C88AEEC2A7
2CF8CF95BBEC2C87A226B40B8D6257FF00D0EFC61FA686B4E61CC319D
D317FCF9C9376A6467D1AD1BA9B505A1F62A580B974AC7397172D1013
0896E032D6491F8CFF1040EFD06FAA2A8E228C323284141AB8A601998
148B0AC8871416A727083D3E93D
01|>
```

4.4.11 **DYNMEMSETW**

Command syntax:

`DYNMEMSETW <start_addr> <len> <data_0> ... <data_n>`

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

start_addr**:** address of the target device to start writing data to
len**:** words number to write (max. 16)
data**:** words to write

Answer data:

Success: none
Error: the error code

Description:

Writes **`len`** words to the dynamic memory starting at address **`start_addr`**. This command is only for devices that define size in words (check it out on the Memory Map tool of FlashRunner Workbench), for other devices see the command *DYNMEMSET*. More *DYNMEMSETW* can be sent defining different memory areas.
Please refer to chapter 6 for a detailed description.

Example:

`#1*DYNMEMSETW 0x0000 4 0x2301 0x6745 0xAB89 0xEFCD 01|>`

Note:
```
Address 0x00 -> value 0x01
Address 0x01 -> value 0x23
…
```

4.4.12 **DYNMEMSETW2**

Command syntax:

`DYNMEMSETW2 <start addr> <len> <data stream>`

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

start_addr**:** address of the target device to start writing data to
len**:** number of words to write (max. 500)
data_stream**:** words stream to write defined by hexadecimal digits

Answer data:

Success: none
Error: the error code

Description:

Writes **`len`** words to the dynamic memory starting at address **`start_addr`**. This command is only for devices that define size in words (check it out on the Memory Map tool of FlashRunner Workbench), for other devices see the command *DYNMEMSET2*. More *DYNMEMSET* can be sent defining different memory areas.
Like all commands, the maximum number of characters for a line is 1024. This means that, depending on the first part of the command, **`len`** cannot be higher than 500.
Please refer to chapter 6 for a detailed description.

Example:

```
#1*DYNMEMSETW2 0x0000 4 0123456789ABCDEF
01|>

Note: Address 0x00 -> value 0x01
      Address 0x01 -> value 0x23
      …
```

4.4.13 **ECHO**

Command syntax:

`ECHO <string>`

Scriptable: Yes

Available on: Master and site engines

Permission:

Default: Guest
Changeable**:** No

Parameters:

string: the string to *ECHO*

Answer data:

Success: the string
Error: the error code

Description:

ECHO the whole command on the log and on the terminal. The maximum length of the command is 1024 characters. This command is always printed on the log, independently from the log level selected (see *SETLOGLEVEL* command).

Example:

```
#1*#ECHO This is a dummy string.
01|#1*ECHO This is a dummy string.
01>
```

4.4.14 **FORCEDRIVER**

Command syntax:

`FORCEDRIVER <driver_name>`

Scriptable: Yes, before *#LOADDRIVER*

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

driver_name: the driver to load

Answer data:

Success: none
Error: the error code

Description:

Loads a different driver in place of the one specified in the *#LOADDRIVER* command. It can be used to have different versions of the same driver in the FlashRunner and call the desired one.

After the project is executed, this configuration is reset. In fact, the suggested usage of this command is to place it at the beginning of the project file between the *!ENGINEMASK* and the *#LOADDRIVER* command. There is also the command *#UNFORCEDRIVER* to manually reset this configuration, but this is actually redundant.

Example:

```
#1*FORCEDRIVER <libespressif_101.so>
01>
```

4.4.15 FRBREAD

Command syntax:

```
FRBREAD <address> <length> [page_size] [fill_value]
```

Scriptable: No

Available on: Site engines only

Permission:

Default: None
Changeable**:** Yes

Parameters:

address: the address where to start reading
length: length of the read, max 128 bytes
page_size: align data to a page size
fill_value: fill value when there are no data in the FRB

Answer data:

Success: the data inside the FRB
Error: the error code

Description:

Read the data inside the loaded FRB (with *TPSETSRC* command). Max 128 bytes for each read. The data can be aligned to a specific page size and a fill value can be set (default is *0xFF*). This command is disabled by default for security reasons, since using this you could also read decrypted data from an FRS. To enable it the User Management has to be enabled and the permission level changed. Please, refers to the chapter “Data Protection System”.

Example:

```
#1*#TPSETSRC 128_512.frb
01>
#1*FRBREAD 0x00 0x03
01|[Addr=0x00000000] 0x00 0x01 0x02
01>
```

4.4.16 **FRBREADCMAC**

Command syntax:

FRBREADCMAC

Scriptable: No

Available on: Master and site engines

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: the CMAC of the FRB
Error: the error code

Description:

Calculates CMAC of the previously set FRB file. CMAC value is calculated based on every FRB byte. Must be preceded by *TPSETSRC* command

Example:

```
#1*TPSETSRC 128B.frs
01>
#1*FRBREADCMAC
01|A145745E72467E58865DBD8843E2BE4C
01|>
```

4.4.17 **FRBREADCRC**

Command syntax:

FRBREADCRC

Scriptable: No

Available on: Master and site engines

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: the CRC of the FRB
Error: the error code

Description:

Calculates CRC of the previously set FRB file. CRC value is calculated based on every FRB byte. Must be preceded by *TPSETSRC* command

Example:

```
#1*#TPSETSRC 128_512.frb
01>
#1*FRBREADCRC
01|CE95C071
01>
```

4.4.18 **FSCMAC**

Command syntax:

FCSMAC <type> <filename> <key_length> <encrypted_key>

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

type**:** filetype you want to analyse: could be PRJ, LIB, FRB, LIC or LOG
filename**:** file to be used to calculate the CMAC
key_length**:** key length which can be: 16, 24 or 32 bytes (optional)
encrypted_key**:** data stream containing the secret key encrypted with the public key (optional)

Answer data:

Success: the CMAC value
Error: the error code

Description:

Calculate and return the CMAC of a file.
For FRS files which includes the CMAC, it just reads the value from its header and the parameters key_length and encrypted_key are not needed.

Example:

```
#55*FSCMAC FRB 128B.frs
55|CMAC = A145745E72467E58865DBD8843E2BE4C
55|>

#55*FSCMAC PRJ test.prj 16
c06ebdd1f29f07df37550e4b2fff865698c992d8296ef38c124
44ed023d52a48d23aea6ca6aee069111781c454d2004be53113
4038352233362c0d215a51e3afccfc81acff892e380181b98a7
e3a92d58a1ae1a35634dc445442da4bae8bdf8d5b79048e2edf
1b19ea256d6401c086b4c3b2141a2be40ce903aa534fc5e205c
```

```
9054827a74aa3b48a4f241668cbae9096e22fe15459ee918053
1eabd7a9839b5085d373f79b9be71561973fcb9e87f08e596b4
bcfa29f69830a8c6ed68dce37d9d86dfe7908b5cf36ea424fc1
bec204d833b17230002b721c923236bde869d9301eee2c87fa4
907b52628f4ec55ec3b95def91d52999a52352c2e3448f82202
a4b1ff95d0dd25c3e31c1a0947eaa0066222443e7050f421dc1
9d6deb7e77239440d81822fd5cc0a18766f9a2f0a78fc5fbd86
c14b67f9fd102fcd07f481c15d99173f5cbb69be72121c42835
31623bb2167229830e6e0ad70cd2601073d0aea631cf0149ea1
da7476ef4152ae1ea2d4d16e68982f818eb858ea7ee3dbfe399
c2b
55|CMAC = 97DD6E5A882CBD564C39AE7D1C5A31AA
55|>
```

4.4.19 **FSCOUNT**

Command syntax:

`FSCOUNT <type>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

type**:** filetype you want to analyse: could be PRJ, LIB, FRB, LIC or LOG

Answer data:

Success: the number of files contained in the specified folder
Error: the error code

Description:

Count the number of files contained in the specified folder.

Example:

```
#55*FSCOUNT LIC
55|10146
55|>
```

4.4.20 **FSCRC**

Command syntax:

`FSCRC <type> <filename>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

type**:** filetype you want to analyse: could be PRJ, LIB, FRB, LIC or LOG
filename**:** file to be used to calculate the CRC32

Answer data:

Success: the CRC32 value
Error: the error code

Description:

Calculate and return the CRC32 of a file.
For FRB files it just reads the value from its header.

Example:

```
#55*FSCRC LIB libdefault.so
55|CRC = 0x39153D78
55|>
```

4.4.21 **FSEXIST**

Command syntax:

```
FSEXIST <type> <filename>
```

Scriptable: No

Available on: Master engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

type**:** filetype you want to check: could be PRJ, LIB, FRB, LIC or LOG
filename**:** file to retrieve

Answer data:

Success: none
Error: the error code

Description:

Check if a file of a specific file type does exist in FlashRunner storage memory or not.

Example:

```
#55*FSEXIST PRJ test.prj
55|>
```

4.4.22 **FSGETCONTROL**

Command syntax:

FSGETCONTROL

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None.

Answer data:

Success: none
Error: the error code

Description:

Retrieves the read value from the lines belonging to the control connector.

Example:

```
#55*FSGETCONTROL
55|Start line read value is: 1
55|Control lines read value is: 31
55|>
```

4.4.23 **FSLS**

Command syntax:

`FSLS <type> <offset> <count>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

type**:** directory you want to list: could be PRJ, LIB, FRB, LIC or LOG

offset: starting point of the list of files to be returned (optional).

count: number of files to be returned (optional).

Answer data:

Success: the current directory content
Error: the error code

Description:

Lists the contents of the current directory in the FlashRunner and their size in bytes.

Example:

```
#55*FSLS PRJ
55|ATXMEGA128A4.prj - 1019
55|teridian.prj - 770
55|atxmega.prj - 1036
55|test.prj - 1067
55|>
```

4.4.24 **FSLS2**

Command syntax:

```
FSLS2 <type> <offset> <count>
```

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

type**:** directory you want to list: could be PRJ, LIB, FRB, LIC or LOG

offset: starting point of the list of files to be returned (optional).

count: number of files to be returned (optional).

Answer data:

Success: the current directory content
Error: the error code

Description:

Lists the contents of the current directory in the FlashRunner, their size in bytes and the timestamp (GMT) of their last change.

Example:

```
#55*FSLS PRJ 0 4
55|ATXMEGA128A4.prj - 1019 - 743849183
55|teridian.prj - 770 - 1334997983
55|atxmega.prj - 1036 - 1348562783
55|test.prj – 1067 - 1569746783
55|>
```

4.4.25 **FSRM**

Command syntax:

`FSRM <type> <filename>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

type**:** filetype you want to remove: could be PRJ, LIB, FRB, LIC or LOG
filename**:** file to remove

Answer data:

Success: none
Error: the error code

Description:

Removes a file stored in the host system to FlashRunner.
The user can also use the "*****" character as filename, this will remove all files from the selected folder.
To remove the log file, please use the command *CLRLOG*.

Example:

```
#55*FSRM PRJ test.prj
55|>
```

4.4.26 **FSSETCONTROL**

Command syntax:

`FSSETCONTROL <signal_name> <signal_value>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

signal_name**:** could be BUSY|CH1|CH2...|CH16
signal_value**:** could be OFF|ON for BUSY signal or OFF|PASS|FAIL for CH1...|CH16 channels

Answer data:

Success: none
Error: the error code

Description:

Sets a signal belonging to control connector to a defined value.
PASS is low logic level, *FAIL* is high logic level.

Example:

```
#55* FSSETCONTROL CH1 PASS
55|>
```

4.4.27 **GENCRYPTOKEY**

Command syntax:

GENCRYPTOKEY

Scriptable: No

Available on: Site engines only

Parameters:

None

Permission:

Default: Protected
Changeable**:** Yes

Answer data:

Success: none
Error: the error code

Description:

Generates the private and public keys used for the encryption and decryption process.
This operation requires up to one minute to be executed.
In case another pair of keys were already present, they will be overwritten and all the data encrypted with the old keys cannot be decrypted anymore.

Example:

```
#55* GENCRYPTOKEY
55|>
```

4.4.28 GETADMINTIMEOUT

Command syntax:

GETADMINTIMEOUT

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** No

Parameters:

None

Answer data:

Success: *ADMIN* timeout in seconds
Error: the error code

Description:

Get the ADMIN session timeout in seconds. Check the chapter “Data Protection System” for more details.

Example:

```
#55*GETADMINTIMEOUT
55|60 s
55|>
```

4.4.29 GETCMDLEVEL

Command syntax:

GETCMDLEVEL <command>

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

command**:** the command to get the security level

Answer data:

Success: command level
Error: the error code

Description:

Get the command level when the User Management is active. Check the chapter “Data Protection System” for more details.

Example:

```
#55*GETCMDLEVEL DYNMEMSET
55|GUEST
55|>
```

4.4.30 **GETCOUNTER**

Command syntax:

GETCOUNTER

Scriptable: No

Available on: Master engine only

Parameters:

None

Permission:

Default: Guest
Changeable**:** No

Answer data:

Success: none
Error: the error code

Description:

Returns current flash counter status. That number represents the remaining flashing cycles available in **GUEST** mode. Check the chapter “Data Protection System” for more details.

Example:

```
#55*GETCOUNTER
55|16
55|>
```

4.4.31 GETDATE

Command syntax:

GETDATE

Scriptable: No

Available on: Master engine only

Parameters:

None

Permission:

Default: Guest
Changeable**:** No

Answer data:

Success: current date
Error: the error code

Description:

Returns the current date set on FlashRunner.
Date format is <sec> <min> <hour> <date> <month> <year>.
<hour> is in 24-hour time format settings.

Example:

```
#55*GETDATE
55|current date: 8 4 15, 18.39.22
55|>
```

4.4.32 GETENGSTATUS

Command syntax:

GETENGSTATUS

Scriptable: No

Available on: Site engine only

Parameters:

None

Permission:

Default: Guest
Changeable**:** No

Answer data:

Success: the status of the engine
Error: the error code

Description:

Returns the actual engine status when a RUN command is in execution. The answer is composed of 16 (or 32 for FlashRunner HS) characters, one for each channel starting from left, and the value could be “P”, “R”, “F” or “-”. “P” stays for *PASS* status and means that the last programming on this channel passed successfully. “R” stays for *RUN* status and means that the channel is still executing commands. “F” character stays for *FAIL* status and means that the last programming on this channel failed. “-” character means that on this product this channel is not enabled. At power-up, there is one more status, represented by the “_”, which means *IDLE*, so the selected channel never executed any command since power-up.

Example:

```
#55*GETENGSTATUS
55|P_______--------
55|>
```

4.4.33 GETFREEMEM

Command syntax:

GETFREEMEM

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: memory usage details
Error: the error code

Description:

This command shows the memory usage details.
The total size doesn't correspond to the SD memory, it's just the size of the partition dedicated to the user data.
Usable memory is the amount of memory available considering that the log.txt file can reach a maximum of 200MB. If the log file reaches that size, then it's cropped and the oldest logs are removed.

Example:

```
#55*GETFREEMEM
55|Total size: 1356.6 MB
55|Memory used: 677.1 MB
55|Memory free: 609.5 MB
55|Percentage: 53%
55|log.txt size: 0.9 MB
55|Usable memory: 410.3 MB
55|>
```

4.4.34 **GETIP**

Command syntax:

`GETIP`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None.

Answer data:

Success: the IP information
Error: the error code

Description:

Returns FlashRunner IP address, network and gateway

Example:

```
#55*GETIP
55|IP: 192.168.1.137
Netmask: 255.255.255.0
Gateway: 192.168.1.1
55|>
```

4.4.35 GETLOGLEVEL

Command syntax:

GETLOGLEVEL

Scriptable: No

Available on: Master

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success**:** log verbosity level. It's a number within [1-6] range
Error: the error code

Description:

Returns the log verbosity level. Lower numbers mean more verbosity on the log file.

Example:

```
#55*GETLOGLEVEL
55|1
55|>
```

4.4.36 GETPROGRESSBAR

Command syntax:

`GETPROGRESSBAR <channel_num>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

channel_num: number of the channel to get the progress bar

Answer data:

Success: operation and progress percentage
Error: the error code

Description:

Returns the progress percentage of the running operation of program/verify for the selected memories by the command *PROGRESSBAR*. See chapter 11 for more details.

Example:

```
#55*GETPROGRESSBAR 2
55|PROGRAM F: 1%
55|>
```

4.4.37 **GETPUBKEY**

Command syntax:

GETPUBKEY

Scriptable: No

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: public key
Error: the error code

Description:

Returns the public key that must be used to encrypt data that can be decrypted only by that specific FlashRunner.
In case the crypto key is not present, an error is returned.

Example:

```
#55*GETPUBKEY
55|-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAgceuZhUirL4gI
FXvFMTUMwycQIgWgFrytplUbI9t4RpL+SKIeqcqZkPOBJ2HLoEgKfhQls
Est8btGAxWgjgihwSeRZ/sFcyfNBq+MLm+Hvft0QRDDEKY0mh4cafJlBs
8GUdGo1/p1i6p133haIipaaGQrsOArTt/5TMNIGEywEx/SUbsks4SX5Dj
4CWhijqmL/PQhq9p2XqL29TMDxtNmJ2f/DrXHaUWrUCe6tgxCc2zt3h3Z
BLaVEPs+Ntf7UtvKIRC7fmeUr9hDz4QrOoV60gLMorDX3zmVGqEh6GpcW
YQPGcKt/v8ZeKklqdFc3h3jJma21h9E4+2R/zRerzOoE/h7+GGbw5uT+r
FxTOiFXmchnEaCSfPmHEL8k/h6q4R8KaM9bEbGT2VDR336saK1nN4OfLS
yx4x2kffC7WKuDPzDuwpRiNOX4tW6Mxg6VvEfBHGkvt5nqPBu5PdmIICi
scTdZoH3PZ87C5EuJIPDjG7P6aQSYZBlIFYTVDVbJflAgMBAAE=
-----END PUBLIC KEY-----
55|>
```

4.4.38 GETVPROG

Command syntax:

GETVPROG <vprog_line>

Scriptable: No

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

vprog_line: vprog line (0 or 1) to read for the selected channel

Answer data:

Success: current voltage read value
Error: the error code

Description:

Returns the read value for the selected VPROG line in mV.

Example:

```
#1*GETVPROG 0
01|VPROG0=3295
01|>
```

4.4.39 **HELP**

Command syntax:

HELP <lib_name.so>

Scriptable: No

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

lib_name.so**:** library name for which help table has to be shown

Answer data:

Success: help table
Error: the error code

Description:

Returns help table, which contains commands description

Example:

```
#1*HELP libpic16.so
TPCMD MASSERASE <F|E|C>
TPCMD ERASE <F> <start_addr> <size>
TPCMD BLANKCHECK <F|E|I|W> or BLANKCHECK <F|E|I|W> <start_addr>
<size>
TPCMD PROGRAM <F|E|I|W> or PROGRAM <F|E|I|W> <start_addr> <size>
TPCMD VERIFY <F|E|I|W> <R> or VERIFY <F|E|I|W> <R> <start_addr>
<size>
TPCMD READ <F|E|I|W> <start_addr> <size>
TPCMD DUMP <F|E|I|W> <start_addr> <size>
TPCMD RUN or TPCMD RUN <delay(sec)>
TPCMD CONNECT
TPCMD DISCONNECT
01|>
```

4.4.40 **HSMEMFORMAT**

Command syntax:

HSMEMFORMAT

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

None

Answer data:

Success: None
Error: the error code

Description:

Erase all files contained in High-Speed Memory (Only for FlashRunner High-Speed).

Example:

```
#55*HSMEMFORMAT
33|Delete all files on HSMEM.
33|>
```

4.4.41 **ISMEMENOUGH**

Command syntax:

`ISMEMENOUGH <size_kB>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

size_kB**:** Size (kB) of memory to be checked if it is available

Answer data:

Success: YES or NO
Error: the error code

Description:

Returns YES or NO if the size of memory asked is available.
Attention: the parameter must be expressed in kilobytes.

Example:

```
#55*ISMEMENOUGH 1024
YES
55|>

#55*ISMEMENOUGH 1048576
NO
55|>
```

4.4.42 ISPANELMODE

Command syntax:

ISPANELMODE

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: the status of panel mode: ON, OFF, 2, 3 or 4
Error: the error code

Description:

Returns the status of panel mode. Not available on FlashRunner HS model.

Example:

```
#55*ISPANELMODE
55|PANEL MODE OFF
55|>

#55*ISPANELMODE
55|PANEL MODE 2
55|>
```

4.4.43 **LICERASE**

Command syntax:

`LICERASE`

Scriptable: No

Available on: Site engines only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

This command is available only on FlashRunner HS model. It erases all Active Module's currently installed licenses.

Example:

```
#1*LICERASE
1|>
```

4.4.44 **LICINSTALL**

Command syntax:

`LICINSTALL <license_filename>`

Scriptable: No

Available on: Site engines only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

lic_filename**:** license file or can be **`'*'`**

Answer data:

Success: none
Error: the error code

Description:

This command is available only on FlashRunner HS model. It installs new licenses into an Active Module which is selected by sending this command to the Active Module related channel. Before applying this command you need first to download license file into FlashRunner HS License folder.

Example:

```
#1*LICINSTALL MTFC128GAP.lic
1|>
```

4.4.45 **LISTLIC**

Command syntax:

LISTLIC

Scriptable: No

Available on: Master engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: license list
Error: the error code

Description:

Returns the stored license list.

Example:

```
#55*LISTLIC
**************************************************
R7F7010274.lic
License type: DEVICE. Only R7F7010274 is activated
Serial Number: 20027
Creation Date: 14.04.2016
Expiration Date: 9999/12/31
Algorithm Name: librh850.so
Manufacturer: RENESAS
Device Code: R7F7010274
**************************************************
55|>
```

4.4.46 **LISTLICAM**

Command syntax:

`LISTLICAM`

Scriptable: No

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: Active Module license list.
Error: the error code.

Description:

This command is available only on the FlashRunner HS model. Returns Active Module stored license list. You can only install licenses matching the Active Module serial number

Example:

```
#2*LISTLICAM
#0: ADESTO - AT25Q - AT25QL641 - SERMEM4
02|>
```

4.4.47 **LOADDRIVER**

Command syntax:

LOADDRIVER <driver_name> <silicon_name> <family_name> <device_name>

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

driver_name**:** driver filename which supports the selected device.
silicon_name**:** silicon producer which supports the selected device.
family_name**:** family name which supports the selected device.
device_name**:** name of the selected device.

Answer data:

Success: none.
Error: the error code.

Description:

Load the driver and check the license.

Example:

```
#1*#LOADDRIVER libfsl_e.so STMICROELECTRONICS SPC58
SPC584B70
01|>
```

4.4.48 **LOGIN**

Command syntax:

`LOGIN <user> <password>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

user**:** username: *ADMIN* or *GUEST*
password**:** *GUEST* has dummy password (any value accepted). *ADMIN* has the password set with *SETADMINPWD* command

Answer data:

Success: none
Error: the error code

Description:

Login as *ADMIN* or *GUEST*. Check the chapter “Data Protection System” for more details.

Example:

```
#55*LOGIN ADMIN applepie
55|>
```

4.4.49 LOGOUT

Command syntax:

LOGOUT

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

It exits from *ADMIN* account and get back to *GUEST* account. Check the chapter “Data Protection System” for more details.

Example:

```
#55*LOGOUT
55|>
```

4.4.50 **PROGRESSBAR**

Command syntax:

`PROGRESSBAR ON <mem_type> <end_addr>`

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

mem_type**:** the memory to monitor (i.e: F, C, D…)
end_addr**:** address to stop the monitoring

Answer data:

Success: the program/verify progress is monitored
Error: the error code

Description:

It enables the monitoring of the program/verify process for the selected memory. It has to be used in combination with the DLL. With the new DLL in C# the user has to establish a connection with the port <FR_ip>:1236 where the FR will write the progress of the programming. From the DLL side, it is necessary to open a *FR_Logger()* and to read the communication to extract the programming/verify progress. With the old DLL in C++, the user can use the command *GETPROGRESSBAR*. See chapter 11 for more details.

Command Example:

```
#2*PROGRESSBAR ON F 0x100000
02|>
```

4.4.51 **REBOOT**

Command syntax:

REBOOT

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Reboot FlashRunner.

Example:

```
#55*REBOOT
55|>
```

4.4.52 **RLYCLOSE**

Command syntax:

RLYCLOSE

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Close the Relay Barrier of the specified channel to connect it to the target.

Example:

```
#1*RLYCLOSE
01|>
```

4.4.53 **RLYOPEN**

Command syntax:

RLYOPEN

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Open the Relay Barrier of the specified channel to disconnect it from the target.

Example:

```
#1*RLYOPEN
01|>
```

4.4.54 **RSTENGSTATUS**

Command syntax:

`RSTENGSTATUS`

Scriptable: No

Available on: Master and site engines

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Reset engine status internal value.
Sending it to the master will reset all engine statuses, while sending it to a single site engine will just reset that single engine status

Example:

```
#55*RSTENGSTATUS
55|>
```

4.4.55 **RUN**

Command syntax:

```
RUN <project_name> [frb_name]
```

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

project_name**:** project filename to run
frb_name**:** the frb to use

Answer data:

Success: none
Error: the error code

Description:

Starts a project stored inside FlashRunner and defined by its filename. When running a project on a channel not included in the project, the command will be successfully executed, but you see a warning message into the log because nothing is actually done by that channel.

Example:

```
#1*RUN test.prj
01|>
```

4.4.56 **SETADMINPWD**

Command syntax:

`SETADMINPWD <password>`

Scriptable: No

Available on: Master engines only

Permission:

Default: Protected
Changeable**:** No

Parameters:

password**:** new password for *ADMIN* user

Answer data:

Success: none
Error: the error code

Description:

Set up new password value for *ADMIN* user. Password can be up to 40 characters long. Check the chapter “Data Protection System” for more details.

Example:

```
#55*SETADMINPWD myPassword1234!
55|>
```

4.4.57 **SETADMINTIMEOUT**

Command syntax:

`SETADMINTIMEOUT <seconds>`

Scriptable: No

Available on: Master engines only

Permission:

Default: Protected
Changeable**:** No

Parameters:

seconds**:** session timeout for *ADMIN* user

Answer data:

Success: none
Error: the error code

Description:

Set up a session timeout for *ADMIN* user in seconds. After the time is elapsed, the *ADMIN* is logged out automatically. The value range between 3 seconds and 24 hours. Check the chapter “Data Protection System” for more details.

Example:

```
#55*SETADMINTIMEOUT 20
55|>
```

4.4.58 **SETCOUNTER**

Command syntax:

`SETCOUNTER <n_cycles>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Protected
Changeable**:** No

Parameters:

n_cycles**:** number of allowed cycles

Answer data:

Success: none
Error: the error code

Description:

Set up a flash counter. After it is set, *GUEST* mode will have **`n_cycles`** flashing cycles allowed. To stop it just use **`n_cycles = 0.`** Check the chapter “Data Protection System” for more details.

Example:

```
#55*SETCOUNTER 10
55|Counter has been successfully set. It will be
active when logged in GUEST user.
55|>
```

4.4.59 SETCMDLEVEL

Command syntax:

`SETCMDLEVEL <level> <command>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Protected
Changeable**:** No

Parameters:

level**:** *ADMIN*, *GUEST, PROTECTED* or *NONE*
command**:** the command to change the security level

Answer data:

Success: none
Error: the error code

Description:

Change the command level when the User Management is active. Check the chapter “Data Protection System” for more details.

Example:

```
#55*SETCMDLEVEL ADMIN DYNMEMSET
55|>
```

4.4.60 **SETDATE**

Command syntax:

```
SETDATE <sec> <min> <hour> <date> <month> <year>
```

Scriptable: No

Available on: Master engine only

Permission:

Default:	Admin
Changeable**:**	Yes

Parameters:

sec**:**	seconds
min**:**	minutes
hour**:**	hours in 24-hour time format.
date**:**	date
month**:**	month
year**:**	year (last two digits)

Answer data:

Success:	none
Error:	the error code

Description:

Sets the current date on FlashRunner.
Date format is <sec> <min> <hour> <date> <month> <year>.
<hour> is in 24-hour time format settings.

Example:

```
#55*SETDATE 51 46 21 30 11 15
55|>
```

4.4.61 SETDIO

Command syntax:

`SETDIO <DIO_num> <logic_state> <reference_mV>`

Scriptable: Yes

Available on: Site engine only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

DIO_num**:** the DIO, from 0 to 7.
logic_state**:** 1 = high level, 0 = low level, H = high impedance.
reference_mV**:** voltage in mV for high level. Optional if VPROG0 has been already set.

Answer data:

Success: none
Error: the error code

Description:

Sets *DIO_num* to the requested logic state. This command can be used to keep in reset a device during the programming.
In case the parameter **`reference_mV`** isn't set and VPROG0 hasn't been previously set, this command returns an error.
Otherwise, if the parameter **`reference_mV`** is set and VPROG0 has been previously set, the new voltage value is ignored.
This command doesn't enable the output of VPROG0 line, unless it has been previously enabled.
Attention: do not use this command for DIOs controlled by the driver. The driver may remove the setting during the *#TPSTART*, for this reason, it should be placed after it if used in a script.

Example:

In the script, i.e. to keep in reset a device during the programming:

```
#TPSTART
#SETDIO 7 1 3300
#TPCMD CONNECT
```

4.4.62 **SETFRSPWD**

Command syntax:

`SETFRSPWD <password>`

Scriptable: No

Available on: Master engines only

Permission:

Default: Protected
Changeable**:** No

Parameters:

password**:** new password used to decrypt FRS files

Answer data:

Success: none
Error: the error code

Description:

Set up new password value used to decrypt FRS files. Password can be up to 40 characters long.

Example:

```
#55*SETFRSPWD myPassword1234!
55|>
```

4.4.63 **SETIP**

Command syntax:

```
SETIP <IP> <netmask> <gateway>
```

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

IP**:** new programmer IP address
netmask**:** new programmer netmask
gateway**:** new programmer gateway

Answer data:

Success: none
Error: the error code

Description:

Sets the new network settings for LAN peripheral. Once executed, you must reboot FlashRunner in order to enable new settings.

Example:

```
#55*SETIP 192.168.1.128 255.255.255.0 192.168.1.1
55|>
```

4.4.64 **SETLOGLEVEL**

Command syntax:

`SETLOGLEVEL <level>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Admin
Changeable**:** Yes

Parameters:

`level`: log verbosity level. It's a number within [1-6] range

Answer data:

Success: none
Error: the error code

Description:

Sets the log verbosity level. Lower numbers mean more verbosity on log file.

Example:

```
#55*SETLOGLEVEL 1
55|>
```

4.4.65 **SETMUX**

Command syntax:

`SETMUX <level>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

level**:** 0: isolate all outputs, 1: enable first bank, 2: enable second bank

Answer data:

Success: none
Error: the error code

Description:

Sets demultiplexer. “0” value will isolate all outputs, “1” will enable the first bank and “2” value will enable the second bank. This command is used in combination with Demultiplexer tool, available only for FlashRunner 2.0 8 or 16 channel version.

Example:

```
#55*SETMUX 1
55|>
```

4.4.66 **SETPANELMODE**

Command syntax:

`SETPANELMODE <level>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

level**:** 0: standard mode, 1: panel mode, 2: eMMC 8bit mode, 3: NAND mode, 4: NOR mode

Answer data:

Success: none
Error: the error code

Description:

Enable panel mode. This command is not available for FlashRunner HS. If the programmer works in panel mode you could only load a single communication protocol for all channels. For eMMC 8bit, NAND and NOR this setting is necessary to program those devices which use a special configuration.

Example:

```
#55*SETPANELMODE 1
55|>
```

4.4.67 **SETSERIALBAUDRATE**

Command syntax:

`SETSERIALBAUDRATE <baudrate>`

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

baudrate: DEFAULT or HIGH

Answer data:

Success: none
Error: the error code

Description:

Change the speed of the Serial connection. DEFAULT is 115200, HIGH is 3000000. When powered the FlashRunner always starts with 115200. The answer to the command, if successful, is done at the new speed requested.

Example:

```
#55*SETSERIALBAUDRATE HIGH
55|>
```

4.4.68 **SGETAMSN**

Command syntax:

SGETAMSN

Scriptable: No

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: the Active Module serial number
Error: none

Description:

Returns the Active Module serial number. Only for FlashRunner High-Speed.

Example:

```
#01*SGETAMSN
01|GP123456
01|>
```

4.4.69 SGETENG

Command syntax:

SGETENG

Scriptable: No

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Returns the engine instance number for the requested engine.

Example:

```
#1*SGETENG
01|Engine N. 0>
```

4.4.70 **SGETERR**

Command syntax:

SGETERR

Scriptable: No

Available on: Master and site engines

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: the error code stack
Error: none

Description:

Returns the error stack of the last error occurred on the selected engine. Each line follows the rule:
ERR--> **<err num>**|**<desc>**|[**<src file>**, **<line num>**,**<func>**]

Example:

```
#1*SGETERR
01|ERR-->05000007|(null)|[file ../Src/pi-algo_api_rw.c,
line 165, funct API_FrbSet()]
01|ERR-->05000007|(null)|[file ../Src/pi-algo.c, line
350, funct cmd_TPSETSRC()]
01|ERR-->05000007|(null)|[file ../Src/cli-cmd.c, line
305, funct cmd_RUN()]
01|>
```

4.4.71 **SGETSN**

Command syntax:

SGETSN

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: the product serial number
Error: none

Description:

Returns the product serial number.

Example:

```
#55*SGETSN
55|1
55|>
```

4.4.72 **SGETVER**

Command syntax:

SGETVER

Scriptable: No

Available on: Master engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

None

Answer data:

Success: The Operating System version
Error: none

Description:

Returns the Operating System version.

Example:

```
#55*SGETVER
55|3.19
55|>
```

4.4.73 SGETVERALGO

Command syntax:

SGETVERALGO <algo_name>

Scriptable: No

Available on: Site engine only

Permission:

Default: Guest
Changeable**:** No

Parameters:

algo_name**:** algorithm name

Answer data:

Success: algorithm version
Error: none

Description:

Returns the version of the driver indicated as parameter. Usually answer is a 3-digit number: 2 less significant are the minor release, the other one is the major release.

Example:

```
#1*SGETVERALGO libsermem.so
01|04.02
01|>
```

4.4.74 **SGETVERALGOLIST**

Command syntax:

SGETVERALGOLIST

Scriptable: No

Available on: Master engine only.

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: driver version list
Error: none.

Description:

Returns the driver version of all drivers stored inside the programmer. Usually answer is a 3-digit number: 2 less significant are the minor release, the other one is the major release

Example:

```
#55*SGETVERALGOLIST
55|libsermem.so – 04.02
55|libinf_c.so – 02.03
55|libatxmega.so – 02.00
55|>
```

4.4.75 **SHA256**

Command syntax:

`SHA256 <type> <filename>`

Scriptable: Yes

Available on: Master and Sites engines

Permission:

Default**:** Guest
Changeable: No

Parameters:

type: PRJ | LIB | FRB | LIC | LOG | PRJ_FRB
filename: filename for which you want to calculate SHA256

Answer data:

Success: calculated SHA256value
Error: the error code

Description:

Returns the calculated SHA256 of the selected file.
If you choose **`PRJ_FRB`** type, first it returns the SHA256 of the PRJ file selected, then it returns the SHA256 for all FRBs defined in the project.

Example:

1.

```
#55*SHA256 FRB 1MB.frb
55|1MB.frb
bd69c6afcdec157f287c85849e1eeea684b02cb6e901d0424a8
fd5fb67393b98
55|>
```

2.

```
#55*SHA256 PRJ_FRB example.prj
55|example.prj
3e2399f4521a09e3226d3fa205f0c3eff48815537d1c79921a7
9b9349b7e1879
55|2MB.frb
bd961b1e6aa3395c990cc71dc2ab84260edef12ced7f712e1c1
a23f3df3a168b
55|>
```

4.4.76 **SHUFFLEDIO**

Command syntax:

```
SHUFFLEDIO <logic_DIO> <physical_DIO>
                Or
SHUFFLEDIO <phys0->DIOx> ... <phys7->DIOx>
```

Scriptable: Yes**,** after the *#TPSTART* and before the *#CONNECT*

Available on: Site engine only

Permission:

Default**:** Guest
Changeable: No

Parameters:

logic_DIO**:** the logical DIO (0 to 7), to move
physical_DIO: the physical DIO (0 to 7), where to move logical DIO
phys0->DIOx**:** the logical DIO wanted in the corresponding physical position

Answer data:

Success: none
Error: the error code

Description:

With the first command syntax, the logical DIO selected is swapped with the logical DIO in the physical DIO selected.
With the second syntax is possible to set the whole DIO map with a single command. On the first position, there is the physical DIO-0 and the user has to insert the desired logical DIO. On the second position there is the physical DIO-1… and so on.
Each logical DIO must have a unique physical position. In case this is not true, the DIO map is reset to the default value.
The DIO map is reset to the default value at the *#TPEND*.

Example:

1. Move the logical DIO-3 in the physical DIO-7. The logical DIO on physical DIO-7 is moved in the physical DIO-3**.**

```
#1*SHUFFLEDIO 3 7
01|>
```

The new DIO map of the example above is:

```
#1*SHUFFLEDIO_GETMAP
```

```
01|DIO MAP: 0 1 2 7 4 5 6 3
01|>
```

2. Set the whole new DIO map:

```
#1*SHUFFLEDIO 0 2 5 3 4 7 1 6
01|>
```

The new DIO map of the example above is:

```
#1*SHUFFLEDIO_GETMAP
01|DIO MAP: 0 2 5 3 4 7 1 6
01|>
```

4.4.77 **SHUFFLEDIO_GETMAP**

Command syntax:

SHUFFLEDIO_GETMAP

Scriptable: Yes

Available on: Site engine only

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: the pinout
Error: the error code

Description:

Shows the actual Pin Map of the channel selected. The first position indicates for the physical DIO-0 the corresponding logical DIO. The second position indicates for the physical DIO-1 the corresponding logical DIO… and so on.
The logical DIO-8 is the watchdogfeed DIO.

Example:

```
#1*SHUFFLEDIO_GETMAP
01|DIO MAP: 0 2 5 3 4 7 1 6
01|>
```

In the example above we have:

1. On physical DIO-0 the logical DIO-0.
2. On physical DIO-1 the logical DIO-2.
3. On physical DIO-2 the logical DIO-5
4. On physical DIO-3 the logical DIO-3
5. On physical DIO-4 the logical DIO-4
6. On physical DIO-5 the logical DIO-7
7. On physical DIO-6 the logical DIO-1
8. On physical DIO-7 the logical DIO-6

4.4.78 SPING

Command syntax:

SPING

Scriptable: No

Available on: Master engine only

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: **SPONG**
Error: the error code

Description:

Pings the instrument. Used to verify whether FlashRunner is connected to the host system and running correctly.

Example:

```
#55*SPING
55|SPONG
55|>
```

4.4.79 **TCSETDEV**

Command syntax:

`TCSETDEV <par_name> <par_value>`

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

par_name**:** parameter name
par_value**:** parameter value

Answer data:

Success: none
Error: the error code

Description:

Sets device-specific and programming algorithm-specific information. This command must be sent after the **`LOADDRIVER`** command and before a **`TPSTART`**/**`TPEND`** command block. Please note that *CRC* pseudo command is a *CRC* number based on *TCSETDEV* data and is used to prevent device info tampering. For this reason, you can't calculate the *CRC* but you only can copy it from a working project done with FlashRunner Workbench software.

Example:

```
#1*TCSETDEV VDDMIN 1600
01|>
```

4.4.80 TCSETPAR

Command syntax:

TCSETPAR <par name> <par value>

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

par_name**:** parameter name
par_value**:** parameter value

Answer data:

Success: none
Error: the error code

Description:

Sets device-specific and programming algorithm-specific device parameter. This command must be sent after the **LOADDRIVER** command and before a **TPSTART** / **TPEND** command block.

Example:

```
#1*TCSETPAR PWDOWN 20
01|>
```

4.4.81 **TESTVPROG**

Command syntax:

`TESTVPROG <vprog_line> <mV> <output>`

Scriptable: No

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

vprog_line**:** *VPROG* line to read. Could be *0|1*
mV**:** mV to set in output on selected *VPROG* line
output**:** defines if *VPROG* line is in output or internally as high reference value. Could be *ON|OFF*

Answer data:

Success: none
Error: the error code

Description:

Sets up a defined value on vprog lines.

Example:

```
#1*TESTVPROG 0 3300 ON
01|>
```

4.4.82 **TPCMD**

Command syntax:

`TPCMD <command> [par1] [par2] ... [parn]`

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

command**:** programming command
par**:** zero or more programming command parameters

Answer data:

Success: none
Error: the error code

Description:

Performs a programming operation (i.e. mass erase, program, verify, etc.) This command must be sent within a **`TPSTART`**/**`TPEND`** command block. Programming commands and their relative parameters are device-specific.

Example:

```
#1*TPCMD PROGRAM F
01|>
```

4.4.83 TPEND

Command syntax:

TPEND

Scriptable: Yes

Available on: Site engine only

Permission:

Default**:** Guest
Changeable: No

Parameters:

Success: none
Error: the error code

Answer data:

Success: the product serial number
Error: none

Description:

Ends a programming block. This command must be preceded by a **TPSTART** command. **TPCMD** commands must be sent within a **TPSTART/TPEND** command block.

TPSTART/TPEND command block must be preceded by the **TCSETPAR** commands required for your specific target device. The **TPEND** command resets any previously set device-specific and programming algorithm-specific parameters.

Example:

```
#1*TPEND
01|>
```

4.4.84 **TPSETDUMP**

Command syntax:

`TPSETDUMP <filename>`

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

filename: Name of the dump file

Answer data:

Success: none
Error: the error code

Description:

Set up the filename which will be created on FlashRunner storage memory once **`TPCMD DUMP`** command is executed. As FlashRunner executes the same project on several channels, each channel will have its dump file. For this reason, on the filename indicated with this command FlashRunner will apply the prefix "**`S<chN>_`**", where "*chN*" is the channel number to which the dump refers. Dump files are raw binary files.

Example:

```
#1*TPSETDUMP dumpfile.bin
01|>
```

4.4.85 **TPSETSRC**

Command syntax:

`TPSETSRC <filename> [IGNORE_BLANK_PAGE]`

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

filename**:** name of the file in the binaries folder inside FlashRunner

IGNORE_BLANK_PAGE:
avoid programming FRB pages which are filled with the blank value

Answer data:

Success: none
Error: the error code

Description:

Sets the source of data to be programmed and verified in subsequent **`TPCMD`** commands.
The user can also use "*DYNMEM*" as filename, this keyword will set the FlashRunner to use only dynamic memory instead of an FRB file.
The maximum length of <filename.frb> is 40 characters.

Example:

```
#1*TPSETSRC test.frb
01|>
```

4.4.86 **TPSTART**

Command syntax:

TPSTART

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Starts a programming block. To end a programming block, send the **TPEND** command. **TPCMD** commands must be sent within a **TPSTART**/**TPEND** command block.

The **TPSTART** command performs some internal initializations and prepares FlashRunner to execute subsequent **TPCMD** commands.

Example:

```
#01*TPSTART
01|>
```

4.4.87 **TPUNSETDUMP**

Command syntax:

`TPUNSETDUMP`

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Unsets the destination file of data used by the DUMP command (set with #*TPSETDUMP*).

Example:

```
#1*TPUNSETDUMP
01|>
```

4.4.88 **TPUNSETSRC**

Command syntax:

TPUNSETSRC

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Unsets the source of data (set with *#TPSETSRC*) to be programmed and verified in subsequent **TPCMD** commands.

Example:

```
#1*TPUNSETSRC
01|>
```

4.4.89 **UNFORCEDRIVER**

Command syntax:

`UNFORCEDRIVER`

Scriptable: Yes

Available on: Site engines only

Permission:

Default: Guest
Changeable**:** Yes

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Remove the driver forced with the *#FORCEDRIVER* command.

Example:

```
#55*UNFORCEDRIVER
55|>
```

4.4.90 **UNLOADDRIVER**

Command syntax:

`UNLOADDRIVER`

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: none
Error: the error code

Description:

Unload the driver to remove dependencies before updating a new one.

Example:

```
#1*UNLOADDRIVER
01|>
```

4.4.91 **UNSETADMINTIMEOUT**

Command syntax:

`UNSETADMINTIMEOUT`

Scriptable: No

Available on: Master engines only

Permission:

Default: Protected
Changeable**:** No

Parameters:

seconds**:** remove session timeout for *ADMIN* user

Answer data:

Success: none
Error: the error code

Description:

Remove session timeout for *ADMIN* user in seconds. Check the chapter “Data Protection System” for more details.

Example:

```
#55*UNSETADMINTIMEOUT
55|>
```

4.4.92 VOLTAGEMONITOR

Reference: For detailed information refer to chapter 10.

Command syntax:

VOLTAGEMONITOR <parameter>
VOLTAGEMONITOR <parameter> <value>

Scriptable: Yes

Available on: Site engines only

Permission:

Default**:** Guest
Changeable: Yes

Parameters:

parameter: For detailed information refer to chapter 10
value: For detailed information refer to chapter 10

Answer data:

Success: none
Error: the error code

Description:

Monitor the Voltage Level during the programming. For detailed information refer to chapter 10

Example:

```
#1*VOLTAGEMONITOR ON ERROR_CONTINUE    log the error
01|>
#1*VOLTAGEMONITOR CLEAR_AVERAGE        reset values
01|>                                   for both lines
#1*TPCMD MASSERASE F
Time for Masserase [...]
01|>
#1*VOLTAGEMONITOR READ_AVERAGE         print average
01|>                                   for MASSERASE
```

4.4.93 **WATCHDOGFEED**

Command syntax:

`WATCHDOGFEED <frequency> <DIO_num> <duty_cycle> [<reference_mV>]`

Scriptable: Yes**,** between *#TPSTART* and *#CONNECT*

Available on: Site engine only

Permission:

Default**:** Guest
Changeable: Yes

Parameters:

frequency**:** frequency of the square wave
DIO_num**:** the DIO, from 0 to 7
duty_cycle**:** duty cycle of the square wave
reference_mV**:** voltage expressed in mV as reference for high level. This parameter is optional and if not set the voltage would be the same set for the programming.

Answer data:

Success: the actual frequency of the square wave
Error: the error code

Description:

Outputs a square wave on the *DIO*, of the duty cycle and frequency set. If the **`reference_mV`** isn't set and *VPROG0* hasn't been previously set, the square wave won't be enabled until the programming flow enables it (i.e. during the connect). Otherwise, if the *VPROG0* has been previously set, the **`reference_mV`** is ignored.
This command doesn't enable the output of *VPROG0* line, it only enables the output of the square wave on the DIO selected.
Attention: this command can cause problems if used for DIO lines controlled by the driver, please check the PinMap of the driver.
The square wave is turned off at the *#TPEND* command. The user can turn off it manually by setting the frequency to 0. It's important to use the same DIO in the command to restore properly the pinout.

Example:

Turn on the square wave:

```
#1*WATCHDOGFEED 50 7 50 3300
```

```
01|Requested WD frequency: 50 - Actual: 50
01|>
```

Turn off the square wave. Physical DIO-7 returns to be the logic DIO-7:

```
#1*WATCHDOGFEED 0 7 50 3300
01|>
```

Script example, the wave starts in connect when there is the power on:

```
#TPSTART
#WATCHDOGFEED 50 5 50
#TPCMD CONNECT
```

Script example, the wave starts before the connect:

```
#TPSTART
#WATCHDOGFEED 50 5 50 3300
#TPCMD CONNECT
```

4.4.94 **WHOAMI**

Command syntax:

`WHOAMI`

Scriptable: Yes

Available on: Master engine only

Permission:

Default**:** Guest
Changeable: No

Parameters:

None

Answer data:

Success: Prints enabled modes and currently logged user
Error: the error code

Description:

It returns enabled modes and the currently logged user. Check the chapter “Data Protection System” for more details.

Example:

```
#55*WHOAMI
Active users listed below. User currently logged is
highlighted with * symbol:
   ADMIN
-> GUEST
55|>
```

5 Projects

Projects are sequences of commands collected in a text file. This is a handy way to store all the target device information and user settings needed for FlashRunner. Projects are usually created with the Project Wizard tool of the Workbench (see ch **Errore. L'origine riferimento non è stata trovata.** for more information) and stored in the user data path folder. Once created, a project could be edited with any text editor. Please check the example below:

```
;Project generated by "FlashRunner 2.0 Workbench 2.02"


;DEVICE: ATXMEGA32E5
;DRIVER: ATXMEGA 01.07

!ENGINEMASK 0x0000FFFF
#LOADDRIVER libatxmega.so ATMEL ATXMEGA ATXMEGA32E5
#TCSETDEV VDDMIN 1600
#TCSETDEV VDDMAX 3600
#TCSETDEV FOSCMIN 0
#TCSETDEV FOSCMAX 0
#TCSETDEV FPLLMIN 0
#TCSETDEV FPLLMAX 0
#TCSETDEV MCUID 0x2918
#TCSETDEV IDCODE 0x00000000
#TCSETDEV IDCODE_MSK 0x0FFFFFFF
#TCSETDEV CORE ATXMEGA
#TCSETDEV MEMMAP 0 F 0 0x00800000 0x00808FFF 0x00000080
0x00000080 0 0 0x0 0x0 0xFF 0x0 0
#TCSETDEV MEMMAP 1 E 0 0x008C0000 0x008C03FF 0x00000020
0x00000020 0 0 0x0 0x0 0xFF 0x0 0
#TCSETDEV MEMMAP 2 U 0 0x008E0400 0x008E040F 0x00000001
0x00000001 0 0 0x0 0x0 0xFF 0x0 0
#TCSETDEV MEMMAP 3 C 0 0x008E0200 0x008E020F 0x00000001
0x00000001 0 0 0x0 0x0 0xFF 0x0 0
#TCSETDEV MEMMAP 4 L 0 0x008F0020 0x008F002F 0x00000001
```

```
0x00000001 0 0 0x0 0x0 0xFF 0x0 0
!CRC 0x25CDA0E6
#TCSETPAR PROTCLK 15000000
#TCSETPAR PWDOWN 100
#TCSETPAR PWUP 100
#TCSETPAR RSTDOWN 100
#TCSETPAR RSTDRV OPENDRAIN
#TCSETPAR RSTUP 100
#TCSETPAR VPROG0 3300
#TCSETPAR CMODE PDI
#TPSETSRC vipcb6_test.frb
#DYNMEMSET 0x8E0400 7 0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0x00
#TPSTART
#TPCMD CONNECT
#TPCMD MASSERASE C
#TPCMD BLANKCHECK F
#TPCMD PROGRAM F
#TPCMD VERIFY F R
#TPCMD BLANKCHECK E
#TPCMD PROGRAM E
#TPCMD VERIFY E R
#TPCMD PROGRAM U
#TPCMD PROGRAM L
#TPCMD DISCONNECT
#TPEND
```

The example above shows a simple project example that configures a channel subset for a target device. There could be more than one target device configured inside the same project, requiring another command block (starting with **!ENGINEMASK** and finishing with **#TPEND**) which defines the new target device settings.

The channel subset involved for a specific target device is defined by **!ENGINEMASK** command: the following number defines bitwise in base 2 the channels involved.

```
Example:
  Send a command to channels: 8, 5, 3, 2, 1.
  Engine Mask: 0b10010111 = 151
```

The subsequent section defines the target device (through **#LOADDRIVER**) and all the specific device information (through the **#TCSETDEV** command). This section is closed by **!CRC** command: this number prevents altering the information above which are sensitive data and would compromise the programming routine.

The next section is composed mainly of **#TCSETPAR** and **#TPSETSRC** commands, which defines a set of user-defined parameters (the result of Project Wizard settings). These commands are editable and the order doesn't matter.
Common *TCSETPARs* are here listed:

- **CMODE:** name of the communication protocol.
- **PROTCLK:** frequency of the communication.
- **PWDOWN:** ms used to power down the board.
- **PWUP:** ms used to power up the board.
- **RSTDOWN:** us waited to reset-down the board.
- **RSTUP:** us waited to reset-up the board.
- **RSTDRV:** reset drive management. *PUSHPULL* or *OPENDRAIN*.
- **VRPOG0:** voltage level of the VPROG0, it is also the logical voltage of DIO signals.
- **VPROG1:** voltage level of the VPROG1.
- **FOSC:** frequency of the external oscillator of the device.
- **FPLL:** frequency of the PLL of the device.

Each driver can use custom *TCSETPAR* needed for a specific device type. Their description can be found in the Wiki of the driver.

The last section is enclosed between **#TPSTART** and **#TPEND** commands and defines the operations executed on the target device. These commands are editable, the order does matter and we suggest not changing it once Project Wizard compiles the file.

Commands related to single memory types have a double syntax:

- **`#TPCMD PROGRAM F:`** programs automatically the memory type '*F*' in the area defined by the loaded FRB file.
- **`#TPCMD PROGRAM F 0x0 0x100:`** programs memory type '*F*' in the area defined by the command parameters above. Target start address is 0x0, length 0x100. If loaded FRB doesn't contain any data in this area, the target device is not going to be programmed.

Usually, the double syntax is available for **`BLANKCHECK`**, **`PROGRAM`**, **`VERIFY`**, **`READ`** and **`DUMP`** commands.

Note:
The maximum length of commands, parameters, project names, driver names and frb names is 40 characters. All parts are case-sensitive.

5.1 Execution and Termination

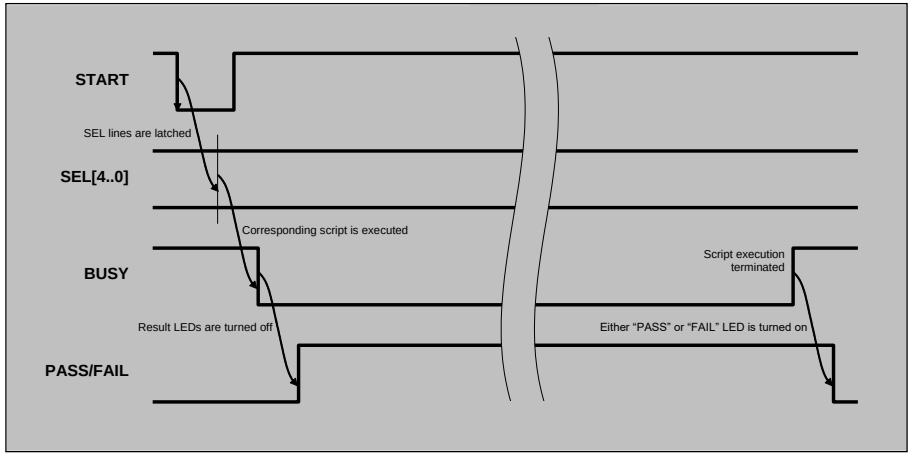
FlashRunner 2.0 execution can be controlled via Host mode (USB or Ethernet connection) or in Standalone mode (Control Connector).
Project execution ends either after FlashRunner 2.0 has executed the last project command or immediately after the first failing project command.

5.1.1 **Standalone project execution**

FlashRunner 2.0 has a control connector (for hardware details please refer to FlashRunner 2.0 User's Manual).

SEL[4..0] is a group of control lines that determine in binary logic a decimal number from 0 to 31. This number represents the project that is going to be executed, named as: *project0.prj ... project31.prj.*

Referring to the below diagram which illustrates the typical temporal relations between the various FlashRunner 2.0 control lines, the event that triggers script execution is the *START* control line becoming active (*LOW* value) while the *BUSY* line is not active (*HIGH* value). This line can be easily driven by an external control logic.
When FlashRunner 2.0 begins executing a project, the *BUSY* LED turns on and the line goes *LOW*.
When the *BUSY* line goes *HIGH* the LED turns off and it is possible to read the *PASS/FAIL* line. A *HIGH* value means failure, a *LOW* value means success.



5.1.2 **Remote projects execution**

A project can be manually executed in host mode. **RUN** command (see ch 4.2) executes a specified project.

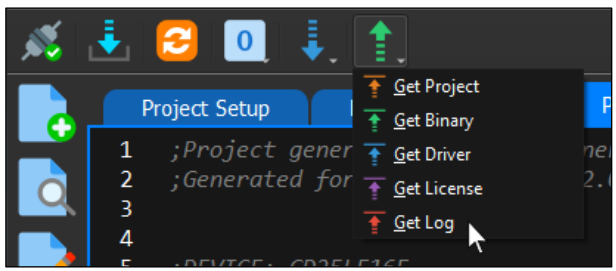
5.2 Project-Specific Directives

FlashRunner commands contained in a project are executed sequentially, exactly as they would be executed in Host mode. However, projects contain additional directives (not available in Host mode) indicated with ***"!"*** prefix which controls how projects are executed. The following table lists these directives:

Directive Syntax	Description
ENGINEMASK	**Defines bitwise which channels are involved for the following command section**
CRC	**Calculate CRC of the preceding commands to avoid specific target device data altering.**

5.3 Logging

On FlashRunner, project command execution is logged. You can check at the runtime log file (see ch 6) or download the log file just by clicking the quick button on the top toolbar.



5.4 Comments

A project line may contain a comment. A comment line starts with the ";" character, FlashRunner will completely ignore that line and so can be used as a comment.

5.5 Conditional scripting

With the aim of raising the flexibility and the customization of projects, FlashRunner implements low level commands able to control the flow of the script's commands.
The syntax used gets back to classical programming languages and shall be immediately clear to all the users who are familiars with them, because it reproduces *if, then, else* statement. In fact, in "C" programming language control flow syntax is as follows:

```
if (expression)
      statement1
else
      statement2
```

where the else part is optional. The expression is evaluated; if it is true (that is if the *expression* has a non-zero value), $statement_1$ is executed. If it is false (the expression is zero) and if there is an else part, $statement_2$ is executed instead.
In FlashRunner the same goal can be achieved using the syntax below inside any project file:

```
#IFERR expression
#THEN statement1
```

in which *expression* is ***TRUE*** when the command returns "**>**" character (meaning that command has been executed successfully), or it is ***FALSE*** if the command returns an error (with correspondent error code).

Notes:
1. Please note that syntax above can be used only inside a script file and it's not recognized on the command line
2. Control flows can't be nested
3. Only one *expression* can be evaluated

4. Multiple *statements* can be executed for each case
5. If *expression* evaluation returns false, the error stack will be traced in the log file. Anyway, if all the subsequent commands will return “>”, the project will not return with an execution error.
6. A syntax error will be returned in case the script has two consecutive IFERR, or if there is an IFERR without a THEN or vice versa.

Example:

The following example is an extract from a script where the MASSERASE operation is carried out only if blank check operation returns an error, that is the device to be programmed is not blank.

```
#IFERR TPCMD BLANKCHECK F
#THEN TPCMD MASSERASE F
```

With this approach it is often possible to reduce project execution time. This technique applies mostly to conditioning target device memory-erasing only if BLANKCHECK fails.

It is also possible to include a second statement to perform the BLANKCHECK operation one more time, just in case the first one failed. In this way it’s possible to be sure that MASSERASE worked, while two operations are skipped if the first BLANKCHECK doesn’t fail.

```
#IFERR TPCMD BLANKCHECK F
#THEN TPCMD MASSERASE F
#THEN TPCMD BLANKCHECK F
```

Please refer to your driver-specific commands before implementing conditional scripting it in your projects.

6 Serial Numbering

6.1 Introduction

Thanks to its built-in dynamic memory, FlashRunner provides you with the possibility of serial numbering during programming operations. During each programming cycle, a host system generates a serial number and transfers it to FlashRunner's dynamic memory. The content of the dynamic memory is then programmed into the target device.

6.2 Command syntax

The following example illustrates how serial numbering can be performed.
Let's assume that the serial number is composed of 4 bytes, must be programmed into target device connected to channel 1, flash starting from address 0x400, and that serial number to be programmed is 0x55 0xAA 0x22 0xFE (0x55 at address 0x400, 0xAA at address 0x401 … and so on).
Host system transfers this serial number to FlashRunner's dynamic memory with the following command:

```
#1*DYNMEMSET 0x400 4 0x55 0xAA 0x22 0xFE
```

or with the following command:

```
#1*DYNMEMSET2 0x400 4 55AA22FE
```

And FlashRunner will apply this "patch" over FRB data. You can define more than one patch, virtually without limits (physical limit

is FlashRunner 1 GB RAM), but defined data is 16 bytes for **DYNMEMSET**, and a total of 512 bytes for the entire **DYNMEMSET2** command.
You can overwrite data which have been previously set in the same addresses, FlashRunner will automatically remove what has been previously set and write the new data. Anyway, we suggest using the command **DYNMEMCLEAR** to clear all data before setting new data.

6.3 Example

```
...
#TCSETPAR RSTUP 100
#TCSETPAR VPROG0 3300
#TCSETPAR CMODE JTAG
#TPSETSRC APH_U27_varD.frb
#DYNMEMSET 0xA0604020 4 0x39 0x30 0x41 0x46
#DYNMEMSET 0xA06040A0 3 0x44 0x48 0x31
#TPSTART
#TPCMD CONNECT
#TPCMD MASSERASE D
#TPCMD BLANKCHECK D
#TPCMD PROGRAM D
#TPCMD VERIFY D R
#TPCMD MASSERASE F
#TPCMD BLANKCHECK F
#TPCMD PROGRAM F
#TPCMD VERIFY F R
#TPCMD DISCONNECT
#TPEND
```

APH_U27_varD.frb must contains defined region at start address 0xA0604020 for 10 bytes size and 0xA06040A0 for 8 bytes size. If your source file doesn't cover this region please use FRB Manager (see ch 3.16) to define it (use Advanced FRB setup feature → Add → Variable data option).

Once defined, this data will be programmed overwriting FRB original data, together with **PROGRAM** command in a single step. Typically, **DYNMEMSET** command is not contained inside a project but it's sent manually from connected PC host; after that PC host can run the project with **RUN** command: FlashRunner will remember **DYNAMIC** data table until **DYNMEMCLEAR** command execution or FlashRunner power-on reset.

Note: *until **#DYNMEMCLEAR** command, dynamic data will be maintained during the project execution loop*

6.4 Word Addressing

Most devices don't need this kind of commands, in fact, this section is reserved for the devices which have a word addressed memory.
If you intend to use dynamic memory with them, you shouldn't use the standard commands described in the previous sections because they use byte addressing. You must use the following commands which are specifically developed for this case (as before, 0x55 at address 0x200, 0xAA at address 0x401 … and so on):

```
#1*DYNMEMSETW 0x200 2 0xAA55 0xFE22
```

or with the following command:

```
#1*DYNMEMSETW2 0x200 2 55AA22FE
```

These commands are extremely similar to the standard ones, just pay attention to the length which is in words and to the endianness.

6.5 Using dynamic memory without FRB

Sometimes it is useful to have a very flexible solution, without using a dummy FRB just to define the addresses of memory where to set dynamic data. That's why you can directly set the dynamic memory as the source instead of an FRB file:

```
#TPSETSRC DYNMEM
```

Below you can see an example where we program and verify only the 12 bytes defined into the dynamic memory, without needing to generate any additional FRB file.

```
#TPSETSRC DYNMEM
#DYNMEMSET2 0x400120 12 E0391234348456807880 9A73
#TPSTART
#TPCMD CONNECT
#TPCMD PROGRAM F
#TPCMD VERIFY F R
#TPCMD DISCONNECT
#TPEND
```

7 Data Protection System

7.1 User management

User management has two modes: *ADMIN* and *GUEST* mode. This mechanism allows administrators to prepare FlashRunner unit with all the required settings and then drop-down privileges and allow *GUESTs* to limit functionalities preventing settings modifications, file download or upload, and so on.

By default, FlashRunner comes with *ADMIN* mode activated only. If you want to enable *GUEST* mode, you need first to create an *ADMIN* password (maximum 40 characters). You can do this by using **SETADMINPWD** command described in chapter 4.4 or using the Cyber Security tab of the Workbench.

```
#LOGIN ADMIN
#SETADMINPWD <new_password>
```

Once done, please remember that after reboot FlashRunner will always start in *GUEST* mode. If you want to disable the user management, you just need to execute **SETADMINPWD** with no password value or use the Cyber Security tab of the Workbench. This way only *ADMIN* will be available. You can see in which state FlashRunner is by using **WHOAMI** command. You can easily switch between users by using **LOGIN** / **LOGOUT** commands.

Warning: *It is important for customers to ensure that they do not misplace or forget the admin password. Anyway, in the unfortunate event this occurs, the customer will need to send the programmer to SMH Technologies for a factory reset.*

Note: *SMH Technologies suggests using always the latest version of OS when Cyber Security features are needed.*

7.1.1 **Command permission level**

When user management is enabled, each command has a default permission level which, for some of them, can be changed. This information is described in chapter 4.4 for each command.

The possible levels permission levels are:

1. **GUEST**: this command can be executed by all the users.
2. **ADMIN**: this command can be executed only by the admin.
3. **PROTECTED**: this command can be executed only if the admin user logged in within 3 seconds.
4. **NONE**: this command can't be executed by any user. These commands, for security reasons, are disabled by default also when the user management is not activated. To use such commands the user has to activate the user management, log in as *ADMIN* and change the command level within 3 seconds of the login.

To change command permission, if possible, the user can send the **`SETCMDLEVEL`** command or use the Workbench. **`GETCMDLEVEL`** is used to get the permission level of a command.

Note: *SMH Technologies strongly suggests using the Workbench GUI. Moreover, some permissions, like the ability to send/receive files, can be changed only via Workbench.*

7.1.2 Non-standard command permission level

The permission level typically is linked to a specific command, but there are some special permission settings:

- **TPMCD_READ**: execution of READ command.
  The default value for this permission is GUEST.

- **TPCMD_DUMP**: execution of DUMP command.
  The default value for this permission is GUEST.

- **CMDBYCMD_EXEC**: execution of scriptable commands when sent outside a project.
  The default value for this permission is GUEST.

- **MULTIPLE_LAN_CONN**: blocks more than one LAN connection to the port 1234, which is the port used to send commands and exchange data with the FlashRunner. The real-time log port 1235 is always available, so if the user has sensitive data on the log, he must hide them using SETLOGLEVEL or using encrypted dynamic data. Moreover, the USB port is always available as well.
  When it is set to NONE, multiple LAN connections are always allowed; when it is set to PROTECTED it blocks more than one LAN connection only during the PROTECTED session; when it is set to ADMIN it blocks more than one LAN connection only for ADMIN user; when it is set to GUEST it blocks more than one LAN connection for all users.
  If the FlashRunner is in GUEST mode with the LAN connection permission set to PROTECTED, and multiple LAN connections are active, changing to ADMIN mode will fail.
  The default value for this permission is NONE.

- **SEND_OS_UPDATE**: allows sending OS updates to the programmer. The default value for this permission is PROTECTED. SMH Technologies suggests setting this to NONE for maximum security. In fact, this will prevent users from installing any old OS that is not managing password and permissions.

7.1.3 **Flashing cycle limitation**

The *ADMIN* can set a maximum number of flashing cycles for the *GUEST* user. This operation can be done on the Cyber Security tab of the Workbench, or using the commands **`SETCOUNTER / GETCOUNTER.`**

For each programming success, the counter is decreased and when it arrives at 0 the *GUEST* can't run any project anymore.

7.1.4 **Admin session timeout**

It is possible to set a session timeout for the *ADMIN* user. The countdown starts when the login is executed. When the counter arrives at zero, at the next command received the *ADMIN* is logged out and the FlashRunner enters in *GUEST* mode.

The **`SETADMINTIMEOUT`** can be used to set the counter value in seconds. The **`UNSETADMINTIMEOUT`** to remove the counter. The **`GETADMINTIMEOUT`** to get the timeout value in seconds.

7.2 FRB encryption

Each FRB could be encrypted using the FlashRunner Workbench tool (See ch 3.10).
This feature will produce a new file, with .frs extension, which is the encrypted version of the original file. New .frs file can't be encrypted anymore outside of the specified FlashRunner.
To use it, please, upload .frs to FlashRunner (using Advanced File Manager, see ch 3.11) and change **#TPSETSRC** filename extension on the related project, finally upload the project to FlashRunner.

When using the highest level of encryption, you could face the need to transfer public keys from one computer to another one. In this case, you have to connect the FlashRunner to the first computer and go to the “Cyber Security” tab on the “Firmware Encryption” section. From there you can press the button “Get Public Key” to obtain the public key of your FlashRunner, then press the button “Export public keys” to generate a “pubKeys.frk” file containing all the public keys memorized by that computer.
On the second computer, while following the procedure to encrypt the firmware, you will be asked to choose the SN of the FlashRunner and, from that window, you can click the “Import public keys” button to import the “pubKeys.frk” file.

See also:

- #GENCRYPTOKEY
- #GETPUBKEY
- #SETFRSPWD

7.3 Managing big FRS on FlashRunner High-Speed

When encrypting big firmware files (such as firmware for an eMMC memory) and employing FlashRunner High-Speed, a decrypted version of the file is stored in the High-Speed Memory to guarantee the best performance. This memory is a non-volatile memory soldered in the programmer and the user cannot read its content using standard commands. The only possible vulnerability would be to steal the programmer and access the memory in a non-conventional way. To prevent this remote possibility, we recommend executing the HSMEMFORMAT command to clean the High-Speed Memory after completing the production batch.

See also:

- HSMEMFORMAT

7.4 Dynamic data encryption

Dynamic data (such as passwords, serial numbers and other keys that are different for each target) can be encrypted as well using the same method used to encrypt FRB.
After encrypting these data, they can only be decrypted by the specific FlashRunner selected while encrypting them.

The data can be encrypted using the DLLs. See chapter 8 for more details.

See also:

- #DYNMEMSETHEADER
- #DYNMEMCLEARHEADER

7.5 OS Certification

Starting from version 3.19 the procedure of installation of the OS changes and to downgrade (or upgrade) you have to use the new certified versions.

The previous certificated versions of the OS (starting from 3.17 included) can be downloaded from our website in the OS changelog page that can be found in the Wiki section.

For older versions than OS 3.17, please contact directly our Support Team (T:+39 0434 421111 or support@smh-tech.com).

8 FlashRunner Interface Library

8.1 Overview

This chapter deals with interfacing FlashRunner with PC applications written by the user. This chapter assumes you have already read the previous sections of this manual and got acquainted with the instrument.

8.2 FlashRunner Interface Library Overview

FlashRunner Interface Library is a DLL that includes all of the functions that allow you to set up a communication channel with the instrument and send commands to FlashRunner.
Dynamic-link libraries (DLL) are modules that contain functions and data. A DLL is loaded at run time by its calling modules (.exe or .dll). When a DLL is loaded, it is mapped into the address space of the calling process.
FlashRunner Interface Library contains Visual C++ written routines (version 1.0.x.x) that can be used to interface the instrument from within, for example, a Microsoft Visual C++ or Visual Basic application, as well as any other programming language that supports the DLL mechanism.
It also contains a Visual C# written COM Interop class library (version 2.0.x.x) that can be used to interface the instrument not only with the above-mentioned IDEs but also with Visual C#, Visual C++ CLI applications and graphical programming environments such as, for example, LabVIEW and TestStand.

For details on how to call DLL functions from within your application, please refer to your programming language's documentation.

8.3 Installation

Before to start working with the FlashRunner Interface Library, you must set up your system with all the required files and drivers. The files to be installed, into your application's directory, are:

For version 1.0:

- The **"FR_COMM.dll"** (this file must also be redistributed with your application);
- For Visual C++ only: the **"FR_COMM.lib" and "FR_COMM.h"** files (you must include these files in your project);
- For Visual Basic only: the **"FR_COMM.bas"** file (you must include this file in your project).

For version 2.0:

- **"FR_COMM_x86.dll"** or **"FR_COMM_x64.dll"** (these files must also be redistributed with your application);
- **"FR_COMM_x86.tlb** or **"FR_COMM_x64.tlb"** (these files become necessary only for plain C++ applications requiring COM Interop functionalities. These files must also be redistributed with your application);
- **.NET Runtime Library 3.5** (or higher) is requested for this new version of Interface Library to work.

These files are automatically installed by the System Software setup (in your installation path).

8.4 Interface Library Reference (version 1.0)

8.4.1 Using the Interface Library Functions

When you control FlashRunner within your own application, you will typically follow the steps indicated below:

- **Open a communication channel with the instrument.** The `FR_OpenCommunication()` function must be called prior to any other Interface Library function.
- **Send commands to the instrument and read answers back.**
  Use the `FR_SendCommand()` and `FR_GetAnswer()` functions to send a command and receive the answer sent back by the instrument, respectively.
  As the very first command, the user should always call a SPING command to check the communication and, optionally, also the SGETSN command to check that the connection has been established with the correct FlashRunner.
- **Transfer files to/from FlashRunner.**
  Two dedicated functions, `FR_SendFile()` and `FR_GetFile()`, allow you to transfer a file from the PC to FlashRunner and vice-versa, respectively.
  The `FR_SendFile()` function is typically used to upload a binary file to the instrument, while the `FR_GetFile()` function is typically used to download a log file to the PC.
- **Close the communication channel with the instrument.** This is done by the `FR_CloseCommunication()` function.

8.4.2 Return Values of the Interface Library Functions

Most of the FlashRunner Interface Library functions return an **`unsigned long`** value which indicates whether the function was successfully executed (return value = `0`) or not (return value other than 0). In the latter case it is possible to get extended error information by calling the function **`FR_GetLastErrorMessage()`**.

8.4.3 Unicode Functions

Every Interface Library function comes in two versions, an ASCII version and a Unicode version. ASCII function names end with A, while Unicode function names end with W. For example, the **`FR_SendCommand()`** function is available as an ASCII version as:

```
FR_COMM_ERR   WINAPI   FR_SendCommandA   (FR_COMM_HANDLE
handle, const char *command);
```

and as a Unicode version as:

```
FR_COMM_ERR   WINAPI   FR_SendCommandW   (FR_COMM_HANDLE
handle, const wchar_t *command);
```

8.4.4 FR_OpenCommunication

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_HANDLE WINAPI  FR_OpenCommunicationA
                       (const char *port,
                       const char *settings);
FR_COMM_HANDLE WINAPI  FR_OpenCommunicationW
                       (const wchar_t *port,
                       const wchar_t *settings);
```

Parameters:

port:	communication port. Must be **"LAN"** for Ethernet communication **"COMx"** for USB communication, where "x" is the number of the used port.
settings:	IP address and port for Ethernet communication (e.g. **"192.168.1.100:1234"**), baudrate for USB (e.g. **"115200"**)

Return value:

>0:	handle of the communication.
NULL:	an error occurred. Call the function **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Creates a communication link with the instrument. Returns a communication handle that must be used by successive FlashRunner Interface Library function calls.

Note:

After opening communication, the user should always call a SPING command to check the communication and, optionally, also the SGETSN command to check that the connection has been established with the correct FlashRunner.

8.4.5 FR_CloseCommunication

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_CloseCommunicationA
                   (FR_COMM_HANDLE handle);
FR_COMM_ERR WINAPI FR_CloseCommunicationW
                   (FR_COMM_HANDLE handle);
```

Parameters:

handle:	handle of communication. This is the value returned by the **FR_OpenCommunication()** function.

Return value:

0:	the function was successful.
Other than 0:	an error occurred. Call the **FR_GetLastErrorMessage()** function to get an extended error information.

Description:

Closes the communication link with the instrument.

8.4.6 FR_GetAnswer

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_GetAnswerA
                   (FR_COMM_HANDLE handle,
                   char *answer,
                   unsigned long maxlen,
                   unsigned long timeout_ms);
FR_COMM_ERR WINAPI FR_GetAnswerW
                   (FR_COMM_HANDLE handle,
                   wchar_t *answer,
                   unsigned long maxlen,
                   unsigned long timeout_ms);
```

Parameters:

handle:	handle of communication. This is the value returned by the **FR_OpenCommunication()** function.
answer:	the buffer that will receive the answer (\0 terminated) of the instrument.
maxlen:	maximum number of characters to receive (must be less than or equal to the answer buffer length).
timeout_ms:	timeout, in milliseconds, after which the function returns even if a complete answer has not been received.

Return value:

0:	the function was successful.
Other than 0:	an error occurred. Call the function **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Receives the answer sent by FlashRunner to the PC, in response to the **FR_SendCommand()** function. A **FR_GetAnswer()** function should always follow a **FR_SendCommand()** function.

8.4.7 FR_GetFile

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_GetFileA
                   (FR_COMM_HANDLE handle,
                   const char *protocol,
                   const char *src_filename,
                   const char *dst_path,
                   const char *filetype,
                   FR_FileTransferProgressProc
                   progress);
FR_COMM_ERR WINAPI FR_GetFileW
                   (FR_COMM_HANDLE handle,
                   const wchar_t *protocol,
                   const wchar_t *src_filename,
                   const wchar_t *dst_path,
                   const wchar_t *filetype,
                   FR_FileTransferProgressProc
                   progress);
```

Parameters:

handle:	handle of the communication. This is the value returned by the **FR_OpenCommunication()** function.
protocol:	transfer protocol. Must be "YMODEM".
src_filename:	name of the file to be retrieved from FlashRunner, e.g. "test.prj.
dst_path:	local path where to save the file.
filetype:	could be **FRB|PRJ|LIC|LOG|LIB.**
progress:	address of a callback function which will receive the progress status of the file transfer operation. If not used, set this parameter to NULL.

Return value:

0:	the function was successful.

`Other than 0:` an error occurred. Call the function **`FR_GetLastErrorMessage()`** to get an extended error information.

Description:

Retrieves a file from FlashRunner and stores it in a specified local path.

8.4.8 FR_GetLastErrorMessage

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
void WINAPI FR_GetLastErrorMessageA
            (char *error_msg,
            unsigned long string_len);
void WINAPI FR_GetLastErrorMessageW
            (wchar_t *error_msg,
            unsigned long string_len);
```

Parameters:

error_msg:	buffer that will receive the error message.
string_len:	length of the buffer.

Return value:

none.

Description:

Most of the FlashRunner Interface Library functions return an **unsigned long** value which indicates whether the function was successfully executed (return value = **0**) or not (return value other than 0). In the latter case it is possible to get extended error information by calling the function **FR_GetLastErrorMessage()** function. After the function is called, the **error_msg** buffer is cleaned

8.4.9 FR_SendCommand

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_SendCommandA
                   (FR_COMM_HANDLE handle,
                   const char *command);
FR_COMM_ERR WINAPI FR_SendCommandW
                   (FR_COMM_HANDLE handle,
                   const wchar_t *command);
```

Parameters:

handle:	handle of the communication. This is the value returned by the **FR_OpenCommunication()** function.
command:	string containing the FlashRunner command.

Return value:

0:	the function was successful.
Other than 0:	an error occurred. Call the function **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Sends a command to FlashRunner. To get the command answer, use the **FR_GetAnswer()** function.

8.4.10 FR_SendFile

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_SendFileA
             (FR_COMM_HANDLE handle,
             const char *protocol,
             const char *src_filename,
             const char *dst_path,
             FR_FileTransferProgressProc progress);
FR_COMM_ERR WINAPI FR_SendFileW
             (FR_COMM_HANDLE handle,
             const wchar_t *protocol,
             const wchar_t *src_filename,
             const wchar_t *dst_path,
             FR_FileTransferProgressProc progress);
```

Parameters:

handle:	handle of the communication. This is the value returned by the **FR_OpenCommunication()** function.
protocol:	transfer protocol. Must be **"YMODEM".**
src_filename:	name of the file (inclusive of the path) to be sent to FlashRunner, e.g. **"C:\\MYBINARIES\\FLASH1.FRB"**.
dst_path:	could be **FRB|PRJ|LIC|LOG|LIB**.
progress:	address of a callback function which will receive the progress status of the file transfer operation. If not used, set this parameter to NULL.

Return value:

0:	the function was successful.
Other than 0:	an error occurred. Call the function **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Sends a file from the PC to a specified path of FlashRunner.

8.4.11 FR_GetPublicKey

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_GetPublicKey
                    (FR_COMM_HANDLE handle);
```

Parameters:

handle:	handle of the communication. This is the value returned by the **FR_OpenCommunication()** function.

Return value:

0:	the function was successful.
Other than 0:	an error occurred. Call the function **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Get the public key, for internal use of the DLL, for the encryption of the dynamic data. This is like an initialization for the successive **FR_EncryptData()** operations.

8.4.12 FR_EncryptData

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_EncryptData
                (FR_COMM_HANDLE handle,
                const unsigned char* data_to_encrypt,
                unsigned int len_data,
                unsigned char* header,
                unsigned char* data_encrypted);
```

Parameters:

handle:	handle of the communication. This is the value returned by the **FR_OpenCommunication()** function.
data_to_encrypt:	buffer containing the data to encrypt.
len_data:	length of the data to encrypt.
header:	buffer containing the encrypted header in Hex format. The len is 384 bytes.
data_encrypted:	buffer with the data encrypted in Hex format. The len is **len_data** aligned to 16.

Return value:

0:	the function was successful.
Other than 0:	an error occurred. Call the function **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Receive as input a buffer of data (**data_to_encrypt**). Returns the header, in Hex format, to be sent with the **#DYNMEMSETHEADER** command and the data encrypted, in Hex format, to be used with a **#DYNMEMSET** command. Both the **header** and **data_encrypted** buffers have to be convereted to ASCII format. The **FR_HexToAsciiStream()** utilty can be used for this task.

8.4.13 FR_HexToAsciiStream

Include file:

```
#include "FR_COMM.h"
```

Function prototypes:

```
FR_COMM_ERR WINAPI FR_HexToAsciiStream
                   (const unsigned char* data_hex,
                   unsigned int len_data,
                   char* data_ascii);
```

Parameters:

data_hex:	buffer containing the data in Hex format to be converted to ASCII.
len_data:	the length of the data in input.
data_ascii:	buffer containing the data in ASCII format

Return value:

0:	the function was successful.
Other than 0:	an error occurred. Call the function **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Converts a buffer of Hex values in the corresponding Ascii string. The output (**data_ascii**) has a dimension double of the original buffer (**data_hex**).

8.5 Interface Library Reference (version 2.0)

8.5.1 Using the C# Interface Library Class

When you control FlashRunner within your own application, you will typically follow the steps indicated below:

- **Open a communication channel with the instrument.**
  The `FR_OpenCommunication()` method must be called prior to any other Interface Library method.

- **Send commands to the instrument and read answers back.**
  Use the `FR_SendCommand()` and `FR_GetAnswer()` methods to send a command and receive the answer sent back by the instrument, respectively.
  As the very first command, the user should always call a SPING command to check the communication and, optionally, also the SGETSN command to check that the connection has been established with the correct FlashRunner.

- **Transfer files to/from FlashRunner.**
  Two dedicated methods, `FR_SendFile()` and `FR_GetFile()`, allow you to transfer a file from the PC to FlashRunner and vice-versa, respectively.
  The `FR_SendFile()` method is typically used to upload a binary file to the instrument, while the `FR_GetFile()` method is typically used to download a log file to the PC.

- **Close the communication channel with the instrument.**
  This is done by the `FR_CloseCommunication()` method.

- **Open a communication channel with the instrument for real-time logging**
  This is done by the **`FR_GetLogger()`** function and logger read method.

- **Close the communication channel with the instru0ment for real-time logging**
  This is done by the **`FR_DisposeLogger()`** method.

8.5.2 Return Values of the Interface Library Methods

Most of FlashRunner Interface Library methods return an **`FR_COMM_ERRORS`** enumerative value which indicates whether the function was successfully executed (return value = **`RET_OK`**) or not (return value other than **`RET_OK`**). In the latter case, it is possible to get extended error information by calling the **`FR_GetLastErrorMessage()`** method.

Below a list of actual FR_COMM_ERRORS entries:

```
public enum FR_COMM_ERRORS
{
    RET_OK = 0,
    RET_ERR_INVALID_HANDLE,
    RET_ERR_INVALID_PORT,
    RET_ERR_INVALID_FORMAT,
    RET_ERR_LOGGER_INVALID_FORMAT,
    RET_ERR_INVALID_IP,
    RET_ERR_INVALID_BAUDRATE,
    RET_ERR_OPEN_CHANNEL,
    RET_ERR_CLOSE_CHANNEL,
    RET_ERR_CHANNEL_CLOSED,
    RET_ERR_SEND_BUFFER,
    RET_ERR_GET_BUFFER,
    RET_ERR_RECEIVE_TIMEOUT,
    RET_ERR_SEND_CHAR,
    RET_ERR_GET_CHAR,
    RET_ERR_SEND_COMMAND,
    RET_ERR_GET_ANSWER,
    RET_ERR_SEND_FILE,
    RET_ERR_GET_FILE,
    RET_ERR_YMODEM_SEND,
    RET_ERR_YMODEM_GET,
    RET_ERR_FAST_SEND,
    RET_ERR_FAST_GET,
    RET_ERR_FILE_OPEN,
    RET_ERR_INVALID_DEST_PATH,
    RET_ERR_INVALID_SOURCE_PATH,
    RET_ERR_FILE_NOT_FOUND,
    RET_ERR_EMPTY_FILE,
    RET_ERR_INVALID_COMMAND,
    RET_ERR_UNKNOWN,
    RET_ERR_INVALID_LOGGER,
}
```

8.5.3 Method Reference for FR 2.0

Before calling the methods it is necessary to instantiate a **ComManage**r class object. After that it will be possible to use its methods whose descriptions follow.

8.5.4 FR_OpenCommunication

Signature:

```
FR_COMM_ERRORS FR_OpenCommunication
(out object handle, string port, string settings);
```

Parameters:

handle	handle of the communication.
port:	communication port. Must be **"LAN"** for Ethernet communication **"COMx"** for USB communication, where "x" is the number of the used port.
settings:	IP address and port for Ethernet communication (e.g. **"192.168.1.100:1234"**), baudrate for USB (e.g. **"115200"**)

Return value:

== RET_OK:	the method call was successful..
<> RET_OK:	an error occurred. Call the **FR_GetLastErrorMessage()** method to get extended error information.

Description:

Creates a communication link with the instrument. If successful it returns as output parameter a communication handle that must be used by successive FlashRunner Interface Library methods calls.

Note:

After opening communication, the user should always call a SPING command to check the communication and, optionally, also the SGETSN command to check that the connection has been established with the correct FlashRunner.

8.5.5 FR_CloseCommunication

Signature:

```
FR_COMM_ERRORS FR_CloseCommunication
(object handle);
```

Parameters:

handle:	handle of communication. This is the object obtained by the **FR_OpenCommunication()** method.

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the **FR_GetLastErrorMessage()** method to get extended error information.

Description:

Closes the communication link with the instrument.

8.5.6 FR_SendCommand

Signature:

```
FR_COMM_ERROR FR_SendCommand
(object handle, string  command);
```

Parameters:

handle: handle of the communication. This is the object obtained by the **FR_OpenCommunication()** method.

command: string containing the FlashRunner command (carriage return and line feed characters are added by the DLL).

Return value:

== RET_OK: the method call was successful.

<> RET_OK: an error occurred.
Call the **FR_GetLastErrorMessage()** method to get an extended error information.

Description:

Sends a command to FlashRunner.
According to command prefix (see 4.2.1) the number of expected answers are evaluated. To get the command answer (a unique string with all the involved channels answers), use the **FR_GetAnswer()** function.

8.5.7 FR_GetAnswer

Signature:

```
FR_COMM_ERRORS FR_GetAnswer
(object handle, out string answer, int timeout_ms);
```

Parameters:

handle:	handle of communication. This is the object obtained by the **FR_OpenCommunication()** method.
answer:	the unique string containing all the expected answers from the instrument returned as an output parameter.
timeout_ms:	timeout, in milliseconds, after which the method returns even if a complete answer has not been received.

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the **FR_GetLastErrorMessage()** method to get an extended error information.

Description:

Receives the answer (or the answers) sent by FlashRunner to the PC, in response to a **FR_SendCommand()** method call.

Normally a **FR_GetAnswer()** method should always follow a **FR_SendCommand()** method.

8.5.8 FR_GetLastErrorMessage

Signature:

```
string FR_GetLastErrorMessage(void);
```

Parameters:

None**.**

Return value:

a string containing the error message.

Description:

Most of the FlashRunner Interface Library methods return a **FR_COMM_ERRORS** value which indicates whether the function was successfully executed (return value = **RET_OK**) or not (return value other than **RET_OK**). In the latter case it is possible to get extended error information by calling the **FR_GetLastErrorMessage()** method. After the call, the error is cleaned.

8.5.9 FR_GetDllVersion

Signature:

```
string FR_GetDllVersion(void);
```

Parameters:

None.

Return value:

a string containing the current DLL version (e.g. 2.0.x.x).

Description:

Gets the current DLL assembly version.

8.5.10 FR_SendFile

Signature:

```
FR_COMM_ERRORS FR_SendFile
(object handle, string src_filename, string
dst_path, TransferProgressHandler progress)
```

Parameters:

handle:	handle of the communication. This is the object obtained by the **FR_OpenCommunication()** method.
src_filename:	name of the file (inclusive of the path) to be sent to FlashRunner, e.g. **"C:\\MYBINARIES\\FLASH1.FRB"**.
dst_path:	could be **FRB|PRJ|LIC|LOG|LIB**.
progress:	a delegate object which encapsulates a callback method which will receive the progress status of the file transfer operation. If not used, set this parameter to NULL. It must conform to the following declaration: **delegate void TransferProgressHandler (int progress)**

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the method **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Sends a file from the PC to a specified path of FlashRunner.

8.5.11 FR_GetFile

Signature:

```
FR_COMM_ERRORS FR_GetFile
(object handle, string src_filename, string
dst_path, string file_type, TransferProgressHandler
progress)
```

Parameters:

handle:	handle of the communication. This is the object obtained by the **FR_OpenCommunication()** method.
src_filename:	name of the file to be retrieved from FlashRunner, e.g. "test.prj.
dst_path:	local path where to save the file.
filetype:	could be **FRB|PRJ|LIC|LOG|LIB.**
progress:	a delegate object which encapsulates a callback method which will receive the progress status of the file transfer operation. If not used, set this parameter to NULL. It must conform to the following declaration: **delegate void TransferProgressHandler (int progress)**

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the method **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Retrieves a file from FlashRunner and stores it in a specified local path.

8.5.12 FR_RunProject

Signature:

```
FR_COMM_ERRORS FR_RunProject
(object handle, string project_name, int[]
channels, int timeout_ms, ProjectExecutionHandler
callback)
```

Parameters:

handle:	handle of the communication. This is the object obtained by the **FR_OpenCommunication()** method.
project_name:	name of the file to be executed by FlashRunner, e.g. "test.prj.
channels:	an array of channels we want the project to be executed on (e.g. int [] channels = {1 2 3 14 15 16}).
timeout_ms:	timeout, in milliseconds, after which the method returns even if not all the channels have completed project execution.
callback:	a delegate object which encapsulates a callback method which will receive the channel id and the execution result (PASS=true or FAIL=false). If not used, set this parameter to NULL. It must conform to the following declaration: **delegate void ProjectExecutionHandler(int channel, bool result);**

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the method **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Executes a project on a given set of FlashRunner's channels while receiving notifications upon individual channel project execution.

8.5.13 FR_GetLogger

Signature:

```
FR_COMM_ERRORS FR_GetLogger
(string ip_address, out FR_Logger logger)
```

Parameters:

ip_address:	IP address and port for Ethernet communication (e.g. **"192.168.1.100:1235"**).
logger:	**FR_Logger** class object used for the real-time logging.

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the method **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Creates a communication link with the instrument for the real-time logging. If successful it returns as output parameter a **FR_Logger** object handle that must be used to read from the network stream by using its **read()** methods.

8.5.14 FR_DisposeLogger

Signature:

```
FR_COMM_ERRORS FR_DisposeLogger(FR_Logger logger)
```

Parameters:

logger:	a real-time logging handle of communication. This is the object obtained by the **FR_GetLogger()** method.

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the method **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Closes the communication link with the instrument and dispose the **FR_Logger** object.

8.5.15 FR_GetPublicKey

Signature:

```
FR_COMM_ERRORS FR_GetPublicKey(object handle)
```

Parameters:

handle:	handle of the communication. This is the object obtained by the **FR_OpenCommunication()** method.

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the method **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Get the public key, for internal use of the DLL, for the encryption of the dynamic data. This is like an initialization for the successive **FR_EncryptData()** operations.

8.5.16 FR_EncryptData

Signature:

```
FR_COMM_ERRORS FR_EncryptData(object handle, byte[]
input, out byte[] output, out byte[] header)
```

Parameters:

handle:	handle of the communication. This is the object obtained by the **FR_OpenCommunication()** method.
input:	array containing the data to encrypt.
output:	array with the data encrypted in Hex format. The len is aligned to 16.
header:	array containing the encrypted header in Hex format. The len is 384 bytes.

Return value:

== RET_OK:	the method call was successful.
<> RET_OK:	an error occurred. Call the method **FR_GetLastErrorMessage()** to get an extended error information.

Description:

Receive as input an array of data (**input**). Returns the header, in Hex format, to be sent with the **#DYNMEMSETHEADER** command and the data encrypted, in Hex format, to be used with a **#DYNMEMSET** command. Both the **header** and **output** arrays have to be convereted to ASCII format. The **FR_HexToAsciiStream()** utilty can be used for this task.

8.5.17 **FR_HexToAsciiStream**

Signature:

```
string FR_HexToAsciiStream(byte[] input)
```

Parameters:

`input:`	array containing the data to convert from Hex to a Ascii string.

Return value:

`string:`	the ascii string of the Hex array in input. The length is double of the input.

Description:

Converts an array of Hex values in the corresponding Ascii string. The output (**`string`**) has a dimension double of the original array (**`input`**).

9 FRB Converter

This section explains how to use the **frbconverter.exe** tool from a terminal or a batch script. This tool is very powerful and allows you to create FRB or FRS (encrypted FRB) with almost the same features that you can find from the FlashRunner Workbench. It checks the source file addresses overlaps and if it fits the device's memory map in the case a device is selected (option -device).

You can refer to the **-help** to see the full description of the features and the parameters that can be used. This is the list of some of them:

- **-pathDevices** *devices_smh_path*
  which defines the path for Devices.smh file [Optional]. The default is frbconverter.exe directory.

- **-fillGaps** *<YES|NO>*
  which enables to fill the gaps of the source file [Optional].

- **-device** *part_number*
  which defines the device part number [Optional].

- **-input** *input_file_name*
  which defines the input file and path. It can be used multiple times to use multiple input files.

- **-format** *input_file_format*
  which defines the format of the input file. It must be used for each input file. Supported formats are:
  - **bin** – for binary files.

- **hex** – for Intel Hex files.
- **s19** – for Motorola SREC files.

- **-output** *output_file_name*
  which defines the output file name and path.

- **-offset** *offset_value*
  which defines an offset and that can be used only for binary files.

- **-pubKeys** *public_keys_path*
  which defines the file path for pubKeys.frk file where public keys to encrypt the file are stored [Optional]. This can be enabled only if the output file extension is ".frs", so only encrypted FRB files.

- **-secKey** *secret_key*
  which defines secret key to calculate CMAC [Optional]. This key must be 16, 24 or 32 bytes long.

- **-pwdFRS** *password*
  which defines custom password to encrypt FRB [Optional].

- **-emmcOptimize** *Y*
  which enable the "Remove Blank Pages" from suorce file during FRB conversion only for eMMC devices (Y/N default N) [Optional].

Some examples of typical usage below:

- **frbconverter.exe -input in.hex -format hex -output out.frb**

This simple command converts the `in.hex` file into `out.frb`.

- **`frbconverter.exe -input first.s19 -format s19 -input second.bin -format bin -output out.frb`**
  This command converts the `first.s19` and `second.bin` file into `out.frb`.

- **`frbconverter.exe -input input.bin -format bin -offset 0x200 -output out.frb`**
  This command converts the `input.bin` file with an offset of `0x200` into `out.frb`.

It is also possible to set zones with variable data into the FRB to be used for dynamic data. This can be done by setting as input **`variable`** and defining the parameters below:

- **`-start_addr`** *`address_value`*
  which defines the start address of the variable data.

- **`-size`** *`size_value`*
  which defines the size of the variable data.

- **`-value`** *`fill_byte`*
  which defines the byte to use to fill the variable data.

Some examples of typical usage with variable data below:

- **`frbconverter.exe -input variable -start_addr 0x1000 -size 0x10 -output out.frb`**
  This command defines a variable data from `0x1000` to `0x100F` into `out.frb`.

- **frbconverter.exe -input input.bin -format bin -offset 0x10 -input variable -start_addr 0x0 -size 0x10 -output out.frb**
  This command converts the `input.bin` file with an offset of `0x10` preceded by `0x10` bytes of variable data into `out.frb`.

A simple batch file can be created with the following code:

```
set FRBCONVERTER=C:\Program Files (x86)\SMH
Technologies\FlashRunner2\frbconverter.exe

set INPUT_FILE=C:\Users\rertolupi\Desktop\myFile.s19
set OUTPUT_FILE=C:\Users\rertolupi\Desktop\myFile.frb

call "%FRBCONVERTER%" -input "%INPUT_FILE%" -format s19
-output "%OUTPUT_FILE%"
```

From this command line tool it is also possible to encrypt an existing FRB file or to upgrade an FRS file.

Input parameters are requested in order:

- **-input** *input_file_name*
  which defines the input file and path. It must be an FRB (.frb) or FRS (.frs) according to what you need to do.

- **-pubKeys** *public_keys_path*
  which defines the file path for pubKeys.frk file where public keys to encrypt the file are stored.

- **`-secKey`** *`secret_key`*
  which defines secret key to calculate CMAC [Optional]. This key must be 16, 24 or 32 bytes long.

- **`-pwdFRS`** *`password`*
  which defines custom password to encrypt FRB [Optional].

Example:

- **`frbconverter -input myOldFile.frs -pubKeys C:\...\pubKeys.frk`**
  This command upgrade the `myOldFile.frs` file and create a new encrypted FRB file with the same name. The old file is renamed as `myOldFile.frs.old`.

10 Voltage Monitor

10.1 Introduction

Voltage Monitor is a *new* operative system feature implemented starting from version 2.32/3.02 of the OS that keeps constantly measured the voltage level of the two **VPROG** lines available for each channel and runs in the background regardless of driver, device or number of channels in use.

The basic operating principle is that if an *under-voltage* or *over-voltage* level is detected caused by exceeding both the negative or positive boundary threshold any ongoing flashing operation can be interrupted.

Options to control operations are available therefore the monitoring can be paused or resumed by user commands that can be inserted in the file script, as well as the error can be detected to exit immediately or continue the overall flashing process and log.

Voltage Monitor can be activated without specifying any type of command or parameter. The process starts checking the power level after the activation of the **VPROG** line just after ending the *Power-up delay* defined during the *Project Wizard Creation* and stops before the power is turned off.

If any voltage error is identified, the monitor sends a signal to the operating system which will immediately disable both **VPROG** lines and terminate the execution of the running procedure. After disabling **VPROG** lines digital lines will stop also, resulting in a

variable timeout error return during the currently executed command.

10.2 Command syntax

Voltage monitor is enabled by setting voltage limits control check of the two **VPROG** lines **(0 or 1)** via Workbench software or scripting parameters as described below:

#TCSETPAR PROG(x)LIMITS <thr> <prm2> <prm3>

parameters explanation:

(x) **0** or **1:** specifies the **VPROG** line

<thr> threshold in mV of the error detection for **VPROG**.

Threshold must be equal or greater than 1% of **VPROG**(x)

Example: VPROG0 = 3300mV

minimum threshold value allowed: 33mV

Note: parameter <prm2> and parameter <prm3> are not involved with Voltage Monitor.

#TCSETPAR PROG0LIMITS <thr> 0 0 *VPROG0 threshold limit*
#TCSETPAR PROG1LIMITS <thr> 0 0 *VPROG1 threshold limit*

#TCSETPAR VPROG0 <mV> *VPROG0 Output Level*
#TCSETPAR VPROG1 <mV> *VPROG1 Output Level*

The under-voltage error is detected using the formula:

UVerr = Is Vsampled < (VprogSet *minus* Vthreshold)

The over-voltage error is detected using the formula:

OVerr = Is Vsampled > (VprogSet plus Vthreshold)

The error detected is reported in the *Real-Time log* of the channel in which it occurs.
Error types are described later in the paragraph 10.5.

Optional commands:

#VOLTAGEMONITOR DYN_SAMPLE <value>

Parameter/values explanation:

<value> ENABLED **default*

Dynamic Sampling mode is enabled by default and the time of the sampling point of each channel is dynamically adjusted to always achieve the best available sampling rate.

If the measurement is paused for any channel, the dynamic sampling algorithm (if not disabled by the user) compensates by increasing the sampling time in the other channels to reach the maximum frequency available. The sampling sequence may change due to internal task scheduling but all the channels are equally sampled.

<value> DISABLED

Fixed sampling time is obtained by disabling the *Dynamic Sampling Algorithm*, and can be calculated multiplying the minimum sampling time per channel (300uS) with the number of channels in which the monitor is activated and the number of the power supplies to control.

S.T. = 300uS * 8 channels * (vprog0=1) = 2.4mS ~ 400Hz

(continued)

#VOLTAGEMONITOR ON <value>

<value> ERROR_CONTINUE

The voltage monitor is enabled and keeps constantly monitored the subsequent operation. If an error is detected it is logged and the flashing process continues.

<value> ERROR_EXIT *(default)*

Monitoring is restarted for the current operation and forces an exit of the current command execution if an error is detected.

#VOLTAGEMONITOR OFF

Monitoring can be paused *(if not necessary for the next operation)*

#VOLTAGEMONITOR CLEAR_AVERAGE <value/no value>

<no value>

reset the average value already calculated for both lines

<value> VPROG0
<value> VPROG1

clear data for the selected line only.

#VOLTAGEMONITOR READ_AVERAGE <value/no value>

<no value>

print in the *Realtime Log terminal* both **VPROG0** and **VPROG1** *average values* of the sampled data starting from the beginning of operations or the last **CLEAR_AVERAGE** command.

<value> VPROG0
<value> VPROG1

print the read average value for the selected line

Usage:

```
#VOLTAGEMONITOR CLEAR_AVERAGE
[…]
#VOLTAGEMONITOR READ_AVERAGE
```

Commands can be added to the script to read the voltage value measured during the same operation.

Script Example:

```
[…]
#TCSETPAR PROG0LIMITS 50 0 0
#TCSETPAR VPROG0 3300
[…]
#VOLTAGEMONITOR ON ERROR CONTINUE        log only
#VOLTAGEMONITOR CLEAR_AVERAGE            reset measure
#TPCMD MASSERASE F                       start operation
#VOLTAGEMONITOR READ_AVERAGE             log measure
#VOLTAGEMONITOR OFF                      no monitoring
#TPCMD BLANKCHECK F                      start operation
#VOLTAGEMONITOR ON ERROR_EXIT            error detection
#VOLTAGEMONITOR CLEAR_AVERAGE            reset measure
#TPCMD PROGRAM F                         start operation
#VOLTAGEMONITOR READ_AVERAGE             log measure
#VOLTAGEMONITOR OFF                      no monitoring
#TPCMD VERIFY F R                        start operation
[…]
```

10.3 Computational load

Voltage Monitoring has a computation load that may reflect in 5% - 7% increase of the overall programming time measured on a 16 channels system.

10.4 Measurement Process

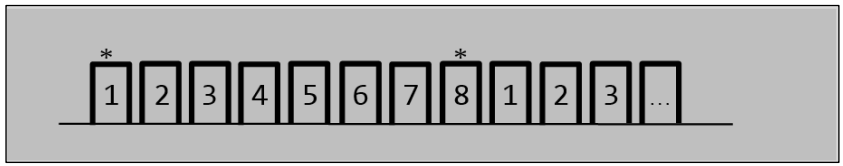
The measurement process starts as soon as **VPROG** is activated and stable in the output line and continues until **VPROG** is shut down.

The sampling frequency is proportional to the number of channels currently active and its value is approximately 3.3KHZ when only 1 channel of **VPROG0** is monitored.

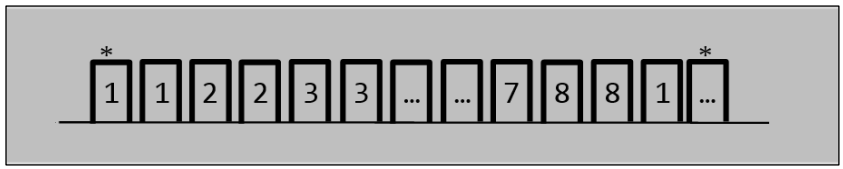
If both **VPROG0** and **VPROG1** lines are monitored simultaneously the sampling time increases to 600us and the sampling frequency is approximately 1.6KHz.

For 8 channels of **VPROG0** monitored only, the sampling frequency is about 400Hz and for 16 channels it is about 200Hz. If **VPROG0 and VPROG1** are both monitored, the sampling rate for 16 channels is approximately 100Hz per channel.

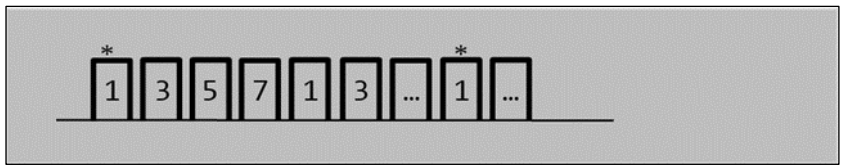
Sampling sequence for 8ch of VPROG0, 300uS per sample:



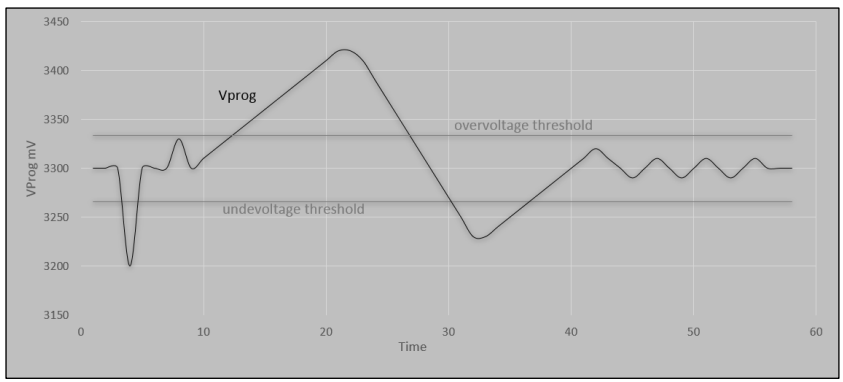
Sampling sequence for 8ch of VPROG0+VROG1, 300uS per sample:



Sampling sequence for 8ch of VPROG0, 300uS per sample, only odd channels are monitored:



Threshold limits:



10.5 Error Types

```
#TCSETPAR PROG0LIMITS 50 0 0
#VOLTAGEMONITOR ON ERROR_CONTINUE
```

Example of under-voltage detection and log:

```
[VoltageMonitorPoll] ch:1, * VProg0 Under Voltage ERROR:
2061mV->3300mV, [@ms: 1224]
- 2061mV: the level measured,
- 3300mV: the reference
- [@ms: 1224]: elapsed time from start of operation
```

→ *task continue.*

```
[VoltageMonitorPoll] ch:1, * VProg0 Over Voltage ERROR:
4180mV->3300mV, [@ms: 1551]
- 4180mV: the level measured,
- 3300mV: the reference
- [@ms: 1551]: elapsed time from start of operation
```

→ *task continue.*

```
#VOLTAGEMONITOR ON ERROR_EXIT

[VoltageMonitorPoll] ch:1, * VProg0 Under Voltage ERROR:
2148mV->3300mV, [@ms: 56075]

!*! -> Exit Signal detected [10]: VMError -6 Address
0x000002dc. Process expiring...
!*! -> Disabling VPROG0...
!*! -> Disabling VPROG1...

[VMErrorStatusCond] threadStatusCond[0] = TD_ERROR
VoltageMonitor has terminated the execution of command:
#TPCMD MASSERASE F

|ERR--0400001D|Voltage Monitor Error
detected|[file ../Src/voltageMonitor.c, line 456, funct
VMSignalError()]
```

11 Progress Bar

11.1 Introduction

Progress Bar is a new operative system feature implemented starting from version 2.39/3.09 of the OS. The aim is to give to the user a tool to keep monitored the programming/verify progress process.

The operating principle is to keep track of how much data of the FRB have been processed and to return a percentage value to the user. Therefore, this operation can't be used to monitor masserase or blankcheck.

This new feature is meant to be integrated using the FlashRunner DLL and to allow the user to create his own progress bar to monitor the progress of the program/verify processes. This way the user has a feedback of the operation status when this takes a long time due to the huge amount of data to program.

This chapter explains how to use this feature and its limitations.

11.2 Command Syntax

From OS 2.39/3.09 to 2.47/3.17

#PROGRESSBAR <num_memories> <start_addr_1> <size_1>

Parameters explanation:

num_memories: this is the number of memories to monitor.

Start_addr_1: start address of the first memory to monitor

Size_1: size of the first memory to monitor

. . .

Example of usage:

A device has a Flash (from 0x0 to 0xFFFF) and an EEPROM (from 0xF1000 to 0xF1FFF); to monitor both the memories, the command will be:

#PROGRESSBAR 2 0x0 0x10000 0xF1000 0x1000

Otherwise, to monitor only one of the two memories, the command will be:

#PROGRESSBAR 1 0x0 0x10000

Script example:

```
[…]
#PROGRESSBAR 2 0x0 0x10000 0xF1000 0x1000
#TPSTART
#TPCMD CONNECT
#TPCMD MASSERASE C
#TPCMD BLANKCHECK C
#TPCMD PROGRAM C
#TPCMD VERIFY C R
#TPCMD MASSERASE D
#TPCMD BLANKCHECK D
```

```
#TPCMD PROGRAM D
#TPCMD VERIFY D R
#TPCMD DISCONNECT
#TPEND
```

Starting from OS 2.48/3.18

Starting from OS 2.48/3.18 the progress bar has been updated to provide better performances and an easier syntax to the user.

```
#PROGRESSBAR ON <mem_type> <end_address>
```

Parameters explanation:

mem_type: character of the memory to monitor: F, D, C…

end_address: end address to monitor.

Example of usage with the device used in the previous chapter:

```
[…]
#TPSTART
#TPCMD CONNECT
#TPCMD MASSERASE C
#TPCMD BLANKCHECK C
#PROGRESSBAR ON C 0xFFFF
#TPCMD PROGRAM C
#TPCMD VERIFY C R
#TPCMD MASSERASE D
#TPCMD BLANKCHECK D
#PROGRESSBAR ON D 0xF1FFF
#TPCMD PROGRAM D
#TPCMD VERIFY D R
```

```
#TPCMD DISCONNECT
#TPEND
```

11.3 Progress Bar and DLL

To get the progress percentage can be used the **GETPROGRESSBAR** command. This command can be sent only to the Master with the following syntax:

```
#55*GETPROGRESSBAR <channel>
```

Where <channel> is the number of the channel to get the process percentage. There are two possible answers:

1. Progress percentage:
```
#55*GETPROGRESSBAR 2
55|VERIFY F R: 45%
55|>
```

2. When the run is ended (success/fail) or before the progress bar gets any data, the answer is:
```
#55*GETPROGRESSBAR 2
55|No operation: 0%
55|>
```

Using standard send/receive functions available in the DLL it's possible to loop this command and get the progress (please refer to chapter 8). It's suggested to introduce an appropriate timeout between two requests in order to not overload the FlashRunner and affect too much the programming performances.

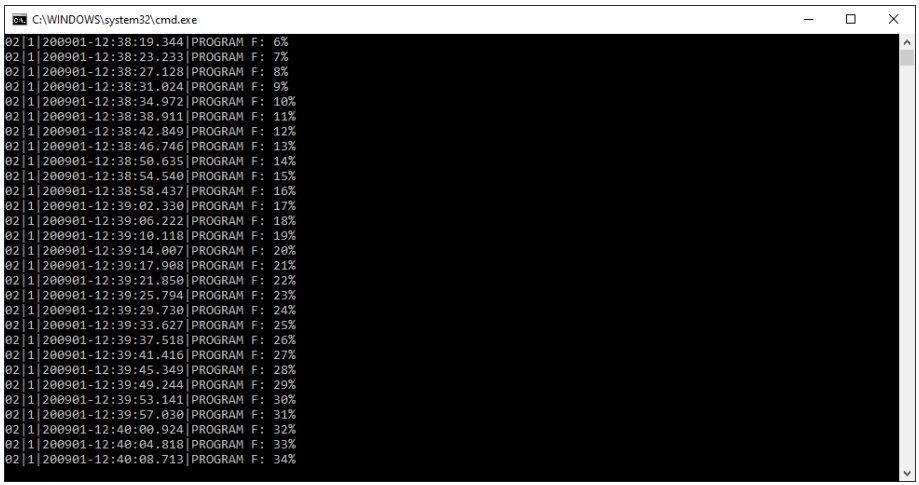
The new C# DLL can be used to get the progress percentage by using the dedicated communication channel on address <FR_ip>:1236 which can be opened using the **`FR_GetLogger`** and then to loop the `Read` command to get the stream.

Example of usage DLL side (pseudo-code):

```
ComManager myComManager = new ComManager();
FR_Logger progress_bar;
myComManager.FR_GetLogger("192.168.1.152:1236", out progress_bar)

while (...condition...)
{
    ... Operation ...
    progress_bar.Read(out buffer, out len);
    ... Operation on the buffer ...
}
```

Example of the channel communication:



If the run fails, 100% is returned from version 2.48/3.18. In the previous versions -1 was returned.

11.4 Limitations

- The use of the Progress Bar, by its nature, generates an increase in cycle time equal to about 15% of the total.
- The progress bar is meant to be used with devices with big memory. Using it with small devices will results in a lot of percentage jumps (i.e: from 0% to 15% and so on).
- The use of the **IGNORE_BLANK_PAGE** and/or fragmented FRB will result in percentage jumps.

From OS 2.39/3.09 to 2.47/3.17

- The progress bar can't be used if the **IGNORE_BLANK_PAGE** option is set on the **TPSETSRC** command.
- Progress bar can be used only with automatic programming/verify, not with manual commands.

FlashRunner Internal Memory

FlashRunner has an internal memory storage which collects all the data, files, information regarding your projects. Its memory is an SD card which comes by default with 64GB size.
This value can be increased up to 256GB.
If you need to increase the memory size of an already purchased product please contact your distributor
If you want to purchase a new product with an already increased memory storage, please notify that to your distributor at ordering time.
Approved SD cards for FlashRunner products are signed below:

2GB	Class10
64GB	microSDXC
128 GB	MicroSDXC
256 GB	MicroSDHC

12 Troubleshooting

This section collects a set of troubleshooting techniques to program successfully your device with FlashRunner.

Note: *Keep FlashRunner always in a well-ventilated area in order to prevent product overheating, which could affect product performance and, if maintained for a long time, it could damage product hardware components.*

12.1 Project execution failures

If you are executing a project and FlashRunner answers to project execution with FAIL please open the Real Time Log tool, described in chapter 3.13 Click on the Log tab, click on the Clear button, Run again project and check related error description. Usually a failure on “Connect” command execution means that FlashRunner and target device are not correctly communicating.

1. Please check that project is set for the exact device mounted on your board
2. Please check cable wirings using the PinMap tool described in chapter 3.15.
3. Verify you are running the correct channel
4. Verify that all connections have been wired correctly using a tester:
    a. check which test point/connector pin implements function described on the PinMap tool and verify the continuity test point/connector pin and FlashRunner

ISP connector pin. You may find useful target board schematics and target board test point map.

b. Did you confuse RX signal with TX signal? Is the soldering rugged?
c. Check which device pin is connected to each test point/connector pin. Check continuity between the device pin and FlashRunner ISP connector.
d. Does each signal they have passive components in between that could cause interference? If capacitance or resistor are needed on some lines (check it on device datasheet) verify that they have been designed on your board under specification.

5. Is the board powered up correctly? If you are using FlashRunner VPROG1, please try with an external power supply. Does current absorption reach a realistic value? (at least 30mA)
6. If you are using an external power supply, be sure that FlashRunner GND line is coupled with the external supplier GND line.
7. If you are using FlashRunner VPROG0 line together with an external supply, be sure that the VPROG0 reference is the same as the one defined by target board design reference.
8. If you are using FlashRunner VPROG1 line, you must be sure that board current absorption is less than FlashRunner model maximum current level supported. Please check FlashRunner User's Manual to get maximum current absorption on VPROG0 and VPROG1
9. Has this board been already programmed? Firmwares could affect device startup, please try always with a device in erased state.
10. Is there a watchdog active on the board? If yes please check how to disable it.
11. Try slowing down communication frequency to the lowest value accepted (100kHz usually is available)

12. Try increasing PWUP, PWDOWN, RSTUP, RSTDOWN values
13. GND reference must not float
14. Please use an oscilloscope to check if signals are affected by “glitches”, if they are present try to compensate by putting a small capacitance between this signal and GND
15. Signals must have a specific time frame for rising edge and falling edge. Check on datasheet which are these constraints and check if they are satisfied. If not, put a power-up resistor (resistor between GND and VPROG0) or a power-down resistor.
16. Remember that cable wirings must be the shortest as possible. Try reducing their length, especially if they are more than 30 cm long and always use twisted and shielded cables.

In case of assistance need please open the Real Time Log tool, described in chapter 3.12. Click on the Log tab, click on the Clear button, Run again project, check related error description. Contact support@smh-tech.com attaching this error log in your email together with SGETVER command answer (please check chapter 4.4.72 for more information)