

FlashRunner 2.0 Series

**High-Performance,
Standalone In-System
Programmers**

Programmer's Manual

Revision 2.8 — July 2020



Copyright © 2017 SMH Technologies

We want your feedback!

SMH Technologies is always on the lookout for new ways to improve its Products and Services. For this reason, feedback, comments, suggestions or criticisms, however small, are always welcome.

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>

Important

SMH Technologies reserves the right to make improvements to FlashRunner, its documentation and software routines, without notice. Information in this manual is intended to be accurate and reliable. However, SMH Technologies assumes no responsibility for its use; nor for any infringements of rights of third parties which may result from its use.

SMH TECHNOLOGIES WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

Trademarks

SMH Technologies is the licensee of the SofTec Microsystems trademark.
All other product or service names are the property of their respective owners.

Contents

| | | |
|----------|--|-----------|
| 1 | BEFORE STARTING | 8 |
| 1.1 | IMPORTANT NOTICE TO USERS | 8 |
| 1.2 | GETTING TECHNICAL SUPPORT | 9 |
| 2 | SYSTEM SETUP/UPGRADE | 10 |
| 2.1 | SOFTWARE SETUP | 10 |
| 2.2 | WHAT YOU NEED TO START | 11 |
| 2.3 | CONNECTION SETUP | 12 |
| 2.4 | FIRMWARE UPDATE | 17 |
| 3 | FLASHRUNNER 2.0 WORKBENCH | 19 |
| 3.1 | OVERVIEW | 19 |
| 3.2 | OPENING WINDOW | 20 |
| 3.3 | TOP TOOLBAR | 22 |
| 3.4 | LEFT TOOLBAR..... | 23 |
| 3.5 | PROJECT SETUP | 24 |
| 3.6 | PRODUCTION CONTROL | 25 |
| 3.7 | PROJECT EDITOR..... | 28 |
| 3.8 | WIZARD..... | 29 |
| 3.8.1 | <i>Introduction page</i> | 30 |
| 3.8.2 | <i>Channel selection page</i> | 31 |
| 3.8.3 | <i>Device selection page</i> | 32 |
| 3.8.4 | <i>FRB Management page</i> | 33 |
| 3.8.5 | <i>Communication settings page</i> | 34 |
| 3.8.6 | <i>Delay settings page</i> | 35 |
| 3.8.7 | <i>Powering settings page</i> | 35 |
| 3.8.8 | <i>Additional parameters page</i> | 35 |
| 3.8.9 | <i>Command settings page</i> | 35 |
| 3.8.10 | <i>Additional commands page</i> | 35 |
| 3.8.11 | <i>Finish page</i> | 35 |
| 3.9 | ENCRYPT FRB | 36 |
| 3.10 | ADVANCED FILE MANAGER | 37 |
| 3.11 | TERMINAL | 39 |
| 3.12 | LOG..... | 40 |
| 3.13 | MEMORY MAP TOOL..... | 42 |
| 3.14 | PIN MAP TOOL..... | 43 |

FlashRunner 2.0 Workbench

| | | |
|----------|---|-----------|
| 3.15 | ADVANCED FRB MANAGER..... | 44 |
| 3.15.1 | <i>Add data to FRB: import from source file.....</i> | 46 |
| 3.15.2 | <i>Add data to FRB: Fill Data / Variable Data</i> | 47 |
| 3.15.3 | <i>Edit FRB block.....</i> | 48 |
| 4 | FLASHRUNNER 2.0 COMMANDS | 50 |
| 4.1 | OVERVIEW | 50 |
| 4.1.1 | <i>Host Mode</i> | 50 |
| 4.1.2 | <i>Standalone Mode.....</i> | 51 |
| 4.2 | COMMAND SYNTAX..... | 51 |
| 4.2.1 | <i>Sending a Command.....</i> | 51 |
| 4.2.2 | <i>Receiving the Answer.....</i> | 53 |
| 4.2.3 | <i>Numeric Parameters.....</i> | 54 |
| 4.3 | COMMAND SUMMARY | 54 |
| 4.4 | COMMAND REFERENCE | 58 |
| 4.4.1 | <i>Command Documentation Conventions</i> | 58 |
| 4.4.2 | <i>CLRERR.....</i> | 59 |
| 4.4.3 | <i>CLRLOG</i> | 60 |
| 4.4.4 | <i>DELAY.....</i> | 61 |
| 4.4.5 | <i>DYNMEMCLEAR</i> | 62 |
| 4.4.6 | <i>DYNMEMSET.....</i> | 63 |
| 4.4.7 | <i>DYNMEMSET2.....</i> | 64 |
| 4.4.8 | <i>DYNMEMSETW</i> | 65 |
| 4.4.9 | <i>DYNMEMSETW2</i> | 66 |
| 4.4.10 | <i>FRBREADCRC.....</i> | 67 |
| 4.4.11 | <i>FSCRC</i> | 68 |
| 4.4.12 | <i>FSEXIST.....</i> | 69 |
| 4.4.13 | <i>FSGETCONTROL.....</i> | 70 |
| 4.4.14 | <i>FSL.....</i> | 71 |
| 4.4.15 | <i>FSL2</i> | 72 |
| 4.4.16 | <i>FSRM.....</i> | 73 |
| 4.4.17 | <i>FSSETCONTROL</i> | 74 |
| 4.4.18 | <i>GETDATE</i> | 75 |
| 4.4.19 | <i>GETENGSTATUS.....</i> | 76 |
| 4.4.20 | <i>GETIP.....</i> | 77 |
| 4.4.21 | <i>GETFREEMEM</i> | 78 |
| 4.4.22 | <i>GETLOGLEVEL.....</i> | 79 |
| 4.4.23 | <i>GETVPROG</i> | 80 |
| 4.4.24 | <i>HELP.....</i> | 81 |

Command Reference

| | | |
|----------|-----------------------------------|------------|
| 4.4.25 | ISMEMENOUGH..... | 82 |
| 4.4.26 | ISPANELMODE..... | 83 |
| 4.4.27 | LISTLIC..... | 84 |
| 4.4.28 | LOADDRIVER..... | 85 |
| 4.4.29 | LOGIN..... | 86 |
| 4.4.30 | LOGOUT..... | 87 |
| 4.4.31 | REBOOT..... | 88 |
| 4.4.32 | RLYCLOSE..... | 89 |
| 4.4.33 | RLYOPEN..... | 90 |
| 4.4.34 | RUN..... | 91 |
| 4.4.35 | RSTENGSTATUS..... | 92 |
| 4.4.36 | SETADMINPW..... | 93 |
| 4.4.37 | SETDATE..... | 94 |
| 4.4.38 | SETDIO..... | 95 |
| 4.4.39 | SETIP..... | 96 |
| 4.4.40 | SETLOGLEVEL..... | 97 |
| 4.4.41 | SETMUX..... | 98 |
| 4.4.42 | SETPANELMODE..... | 99 |
| 4.4.43 | SGETENG..... | 100 |
| 4.4.44 | SGETERR..... | 101 |
| 4.4.45 | SGETSN..... | 102 |
| 4.4.46 | SGETVER..... | 103 |
| 4.4.47 | SGETVERALGO..... | 104 |
| 4.4.48 | SGETVERALGOLIST..... | 105 |
| 4.4.49 | SPING..... | 106 |
| 4.4.50 | TCSETDEV..... | 107 |
| 4.4.51 | TCSETPAR..... | 108 |
| 4.4.52 | TESTVPROG..... | 109 |
| 4.4.53 | TPCMD..... | 110 |
| 4.4.54 | TPEND..... | 111 |
| 4.4.55 | TPSETDUMP..... | 112 |
| 4.4.56 | TPSETSRC..... | 113 |
| 4.4.57 | TPSTART..... | 114 |
| 4.4.58 | UNLOADDRIVER..... | 115 |
| 4.4.59 | VOLTAGEMONITOR..... | 116 |
| 5 | PROJECTS..... | 118 |
| 5.1 | EXECUTION AND TERMINATION..... | 121 |
| 5.1.1 | Standalone project execution..... | 121 |

FlashRunner 2.0 Workbench

| | | |
|-----------|--|------------|
| 5.1.2 | Remote projects execution | 121 |
| 5.1.3 | Projects Termination..... | 122 |
| 5.2 | PROJECT-SPECIFIC DIRECTIVES | 122 |
| 5.3 | LOGGING | 122 |
| 5.4 | COMMENTS | 123 |
| 5.5 | CONDITIONAL SCRIPTING | 123 |
| 6 | SERIAL NUMBERING | 126 |
| 6.1 | INTRODUCTION..... | 126 |
| 6.2 | COMMAND SYNTAX | 126 |
| 6.3 | EXAMPLE | 127 |
| 6.4 | WORD ADDRESSING | 128 |
| 6.5 | USING DYNAMIC MEMORY WITHOUT FRB | 129 |
| 7 | DATA PROTECTION SYSTEM..... | 130 |
| 7.1 | USER MANAGEMENT..... | 130 |
| 7.2 | FRB ENCRYPTION..... | 131 |
| 8 | FLASHRUNNER 2.0 INTERFACE LIBRARY | 132 |
| 8.1 | OVERVIEW | 132 |
| 8.2 | FLASHRUNNER 2.0 INTERFACE LIBRARY | 132 |
| 8.3 | INSTALLATION..... | 133 |
| 8.4 | INTERFACE LIBRARY REFERENCE | 133 |
| 8.4.1 | <i>Using the Interface Library Functions.....</i> | <i>133</i> |
| 8.4.2 | <i>Return Values of the Interface Library Functions.....</i> | <i>134</i> |
| 8.4.3 | <i>Unicode Functions.....</i> | <i>134</i> |
| 8.4.4 | <i>Application examples.....</i> | <i>135</i> |
| 8.4.5 | <i>Function Reference for FR 2.0.....</i> | <i>136</i> |
| 8.4.6 | <i>FR_CloseCommunication</i> | <i>136</i> |
| 8.4.7 | <i>FR_GetAnswer</i> | <i>137</i> |
| 8.4.8 | <i>FR_GetFile.....</i> | <i>138</i> |
| 8.4.9 | <i>FR_GetLastErrorMessage</i> | <i>140</i> |
| 8.4.10 | <i>FR_OpenCommunication</i> | <i>141</i> |
| 8.4.11 | <i>FR_SendCommand.....</i> | <i>142</i> |
| 8.4.12 | <i>FR_SendFile.....</i> | <i>143</i> |
| 9 | FRB CONVERTER | 145 |
| 10 | VOLTAGE MONITOR..... | 148 |

Command Reference

| | | |
|-----------|--|------------|
| 10.1 | INTRODUCTION | 148 |
| 10.2 | COMMAND SYNTAX | 149 |
| 10.3 | COMPUTATIONAL LOAD | 153 |
| 10.4 | MEASUREMENT PROCESS | 153 |
| 10.5 | ERROR TYPES | 155 |
| 11 | FLASHRUNNER 2.0 INTERNAL MEMORY | 156 |
| 12 | TROUBLESHOOTING | 157 |
| 12.1 | PROJECT EXECUTION FAILURES | 157 |

1 Before Starting



Note: *an updated version of FlashRunner 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 a fast and knowledgeable technical support to all of its customers and 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;
- Command-line utilities and Interface Library;
- 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**.

Then click on File → Connect menu item in order to connect to FlashRunner 2.0. If the icon will change to “plugged state”, your product has been connected successfully.

2.2 What you need to start

FlashRunner 2.0 supports several devices. In order to program a specific device, you will need the following:

- A driver file (.so);
- A license file (.lic);
- An FRB file (.frb);
- A project file (.prj);

Driver files are dynamic libraries that contain routines needed to program a set of specific devices. SMH Technologies releases daily updates in order to support new devices, so when you request a new device, you'll often receive also an updated version of the driver.

License files are text files that contain 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: license for a single target device, license for a single-family, license for a silicon manufacturer. Please ask SMH Technologies Sales Team for more information.

FRB file is the FlashRunner proprietary file format used to store customer firmware. There is a specific tool available from FlashRunner 2.0 Workbench, called FRB Manager, described on ch 3.15.

Project files are text files 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 3.7 which allows users to create a project from scratch only using graphical

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

All files are stored in the user data path which can be found or changed on Tools → Settings menu items, “Paths” tab.

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

In order to program a specific device a specific 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 serial numbers (up to 10 programmers can be included inside a license). Doing this, a single file could be installed in more programmers and enable them to program a specific target device.

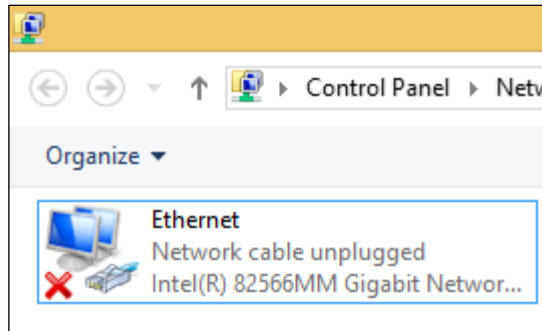
You can purchase a license through our direct channel by writing to our Sales Office: sales@smh-tech.com or, if you bought FlashRunner from an SMH distributor, please contact him. Once bought a license you'll receive a package with a license file and a driver file, which must be copied to your FlashRunner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

2.3 Connection setup

FlashRunner 2.0 Workbench can control 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 it to FlashRunner 2.0 Workbench, you'll have to use FlashRunner 2.0 in Host mode.

Command Reference

Ethernet LAN connection settings:



Example of disconnected network card

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 (see ch 4.4.37).



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, it could damage product hardware components.

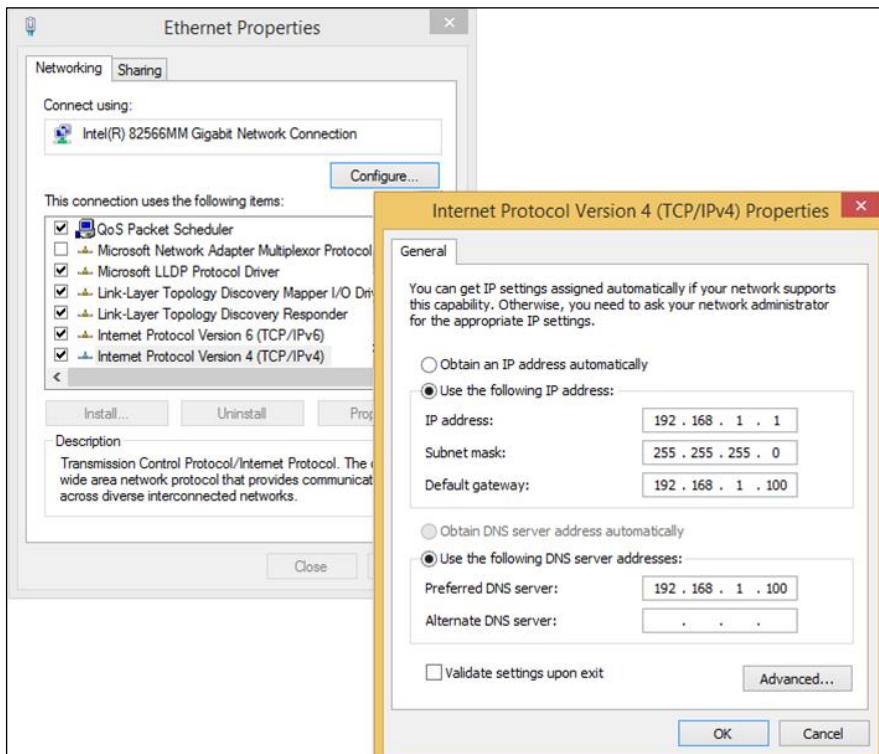
Please use ethernet cable included in FlashRunner 2.0 box and connect it to your switch or directly to your host pc. Once connected, the red cross in the network connections icon related to your network card should disappear.

If host pc and FlashRunner 2.0 are connected through a router, please be sure that they are running 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 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

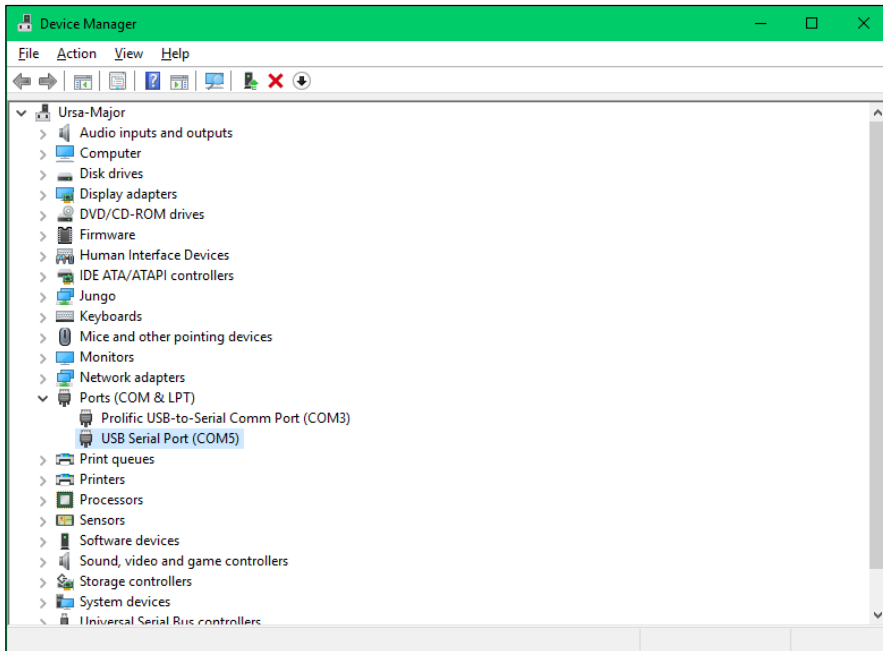


Command Reference

FlashRunner 2.0 Workbench is configured by default to connect to 192.168.1.100 FlashRunner 2.0 IP address. If you'll need to change FlashRunner 2.0 IP address you can easily update also FlashRunner 2.0 Workbench settings using Tool → Settings menu item.

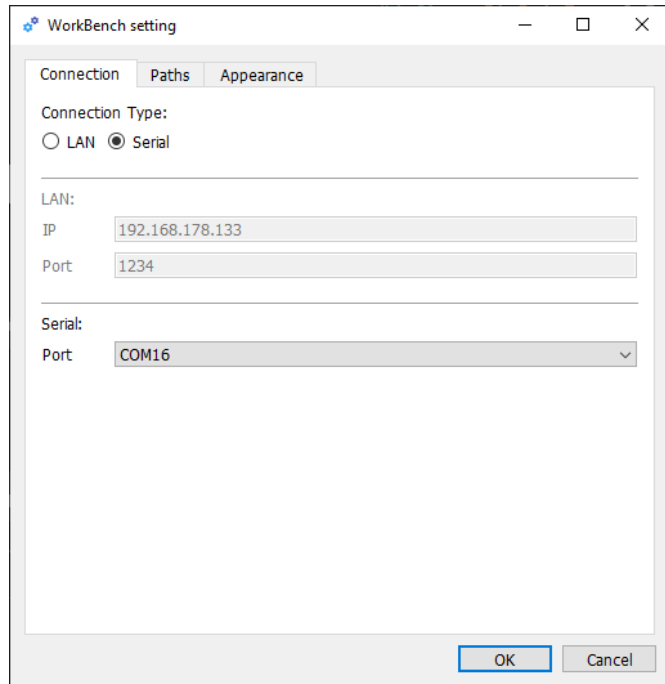
USB CONNECTION SETTINGS – WINDOWS® PROCEDURE:

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 for hardware changes.



Once found this item, please sign which COM port has been assigned to FlashRunner and use it to setup FlashRunner 2.0 Workbench software: please click on Tools → Settings, click on

“Serial” connection type and put COMX value inside Port textbox.



USB CONNECTION SETTINGS – LINUX PROCEDURE:

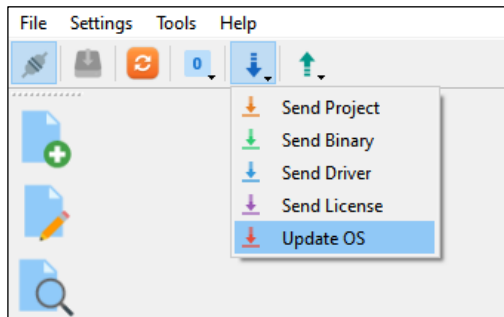
Please check with `dmesg` command which device node has been assigned to FlashRunner 2.0. Usually Linux assigns `ttyUSBX` (where X is an integer number) device node. Please check under `/dev` folder if your user has write/read privileges on `/dev/ttyUSBX` device node. If not, please add it through `chmod`. Please open FlashRunner 2.0 WorkBench Tools → Settings, select Serial Connection Type and fill Port textbox with `/dev/ttyUSBX`.

2.4 Firmware 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 [here](#) to get the latest FlashRunner 2.0 firmware.
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 it will ask to reboot the FlashRunner 2.0.
5. Please, connect again to FlashRunner 2.0 using "Connect" button at the top left of GUI WorkBench.
6. Open Terminal tool available on GUI WorkBench and send on "Master" channel (selectable by toolbar on the bottom-right side) "FPGASTATICVER" command.

7. If FPGASTATICVER command answer is less than “8”, you need to manually reboot the FlashRunner 2.0 two more times by doing a power cycle.
8. Please, check [here](#) to get the latest FlashRunner 2.0 setup, to get the latest GUI WorkBench version, updated documentation and related tools. Please remember to uninstall the previous FlashRunner 2.0 setup before installing the new one.

3 FlashRunner 2.0 Workbench

3.1 Overview

FlashRunner 2.0 Workbench is a simple application for PC which is able to communicate with FlashRunner 2.0 and perform the following operations:

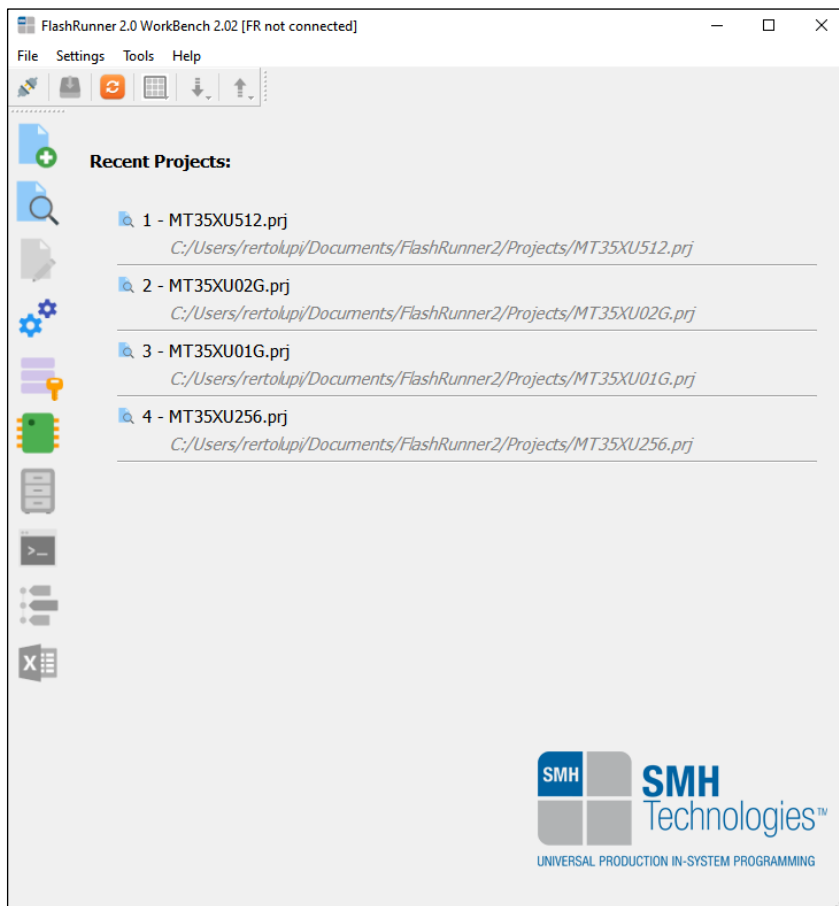
1. Create new projects;
2. Run projects and monitor programmer status;
3. Create FRB binary files;
4. Copy projects, FRB, drivers and licenses from/to programmer;
5. Update OS;
6. Retrieve log.

FlashRunner 2.0 Workbench is compatible with all Microsoft Windows® operating systems and with Linux operating system.

3.2 Opening window

Once you run FlashRunner 2.0 Workbench you'll see a window like the one in the figure 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.

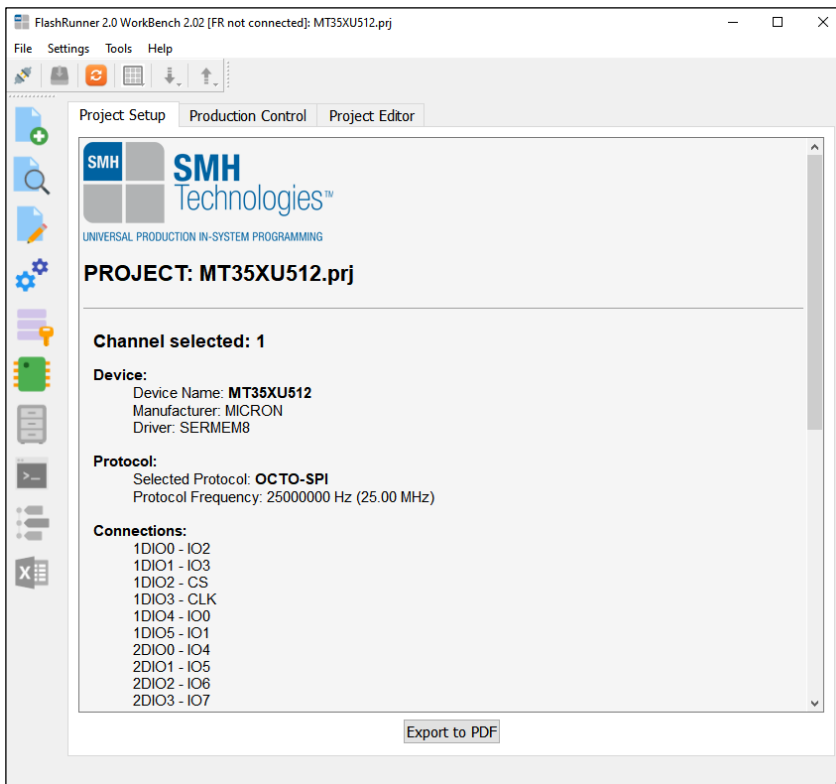


Command Reference

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.

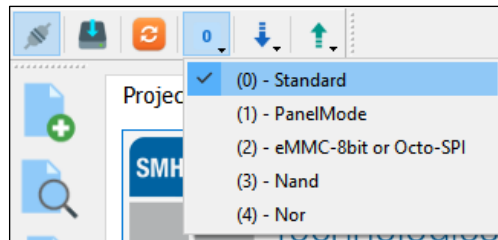


3.3 Top toolbar

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



1. **Connect button:** connect/disconnect from FlashRunner 2.0 and review connection status.
2. **Send configuration button:** send project and FRB to FlashRunner 2.0.
3. **Update database:** download the latest version of the data.xml file, which contains all the info of the supported devices.
4. **Working mode:** set the working mode of FlashRunner 2.0.



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 2.0 Workbench at a sight.



Create project wizard. See ch 3.7



Edit actual / existing project



Load project



Settings



FRB encryption. See ch 3.9



Show device list



Advanced File Manager. See ch 3.10



Terminal. See 3.11



Log. See 3.12

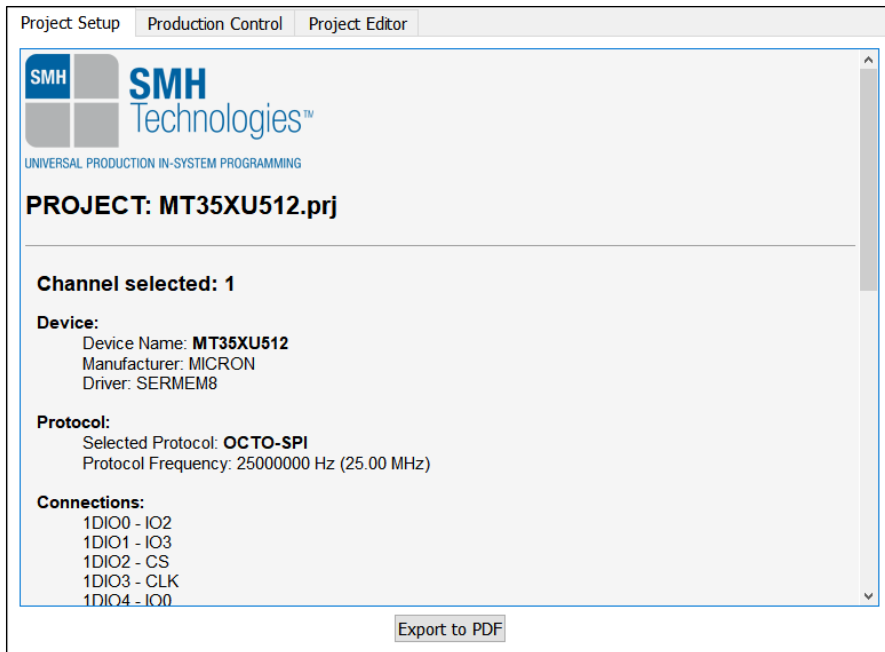


Download Production Report

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 3.14.

It is also possible to export this page in PDF.



3.6 Production Control

| Project Setup | Production Control | Project Editor |
|---|--------------------|----------------|
| <div style="display: flex; justify-content: space-around;"> <div style="width: 45%;"> <h3 style="text-align: center;">Channel: 1</h3> <p>Device: <input type="text" value="AT25QF641"/></p> <p>Binary File: <input type="text" value="8MB.frb"/></p> <p style="text-align: center;"><input type="button" value="Run"/></p> <p>Prog. Time: <input type="text"/></p> <p>N° of PASS: <input type="text" value="0"/></p> <p>N° of FAIL: <input type="text" value="0"/></p> <p>Status: IDLE</p> </div> <div style="width: 45%;"> <h3 style="text-align: center;">Channel: 2</h3> <p>Device: <input type="text" value="AT25QF128A"/></p> <p>Binary File: <input type="text" value="16MB.frb"/></p> <p style="text-align: center;"><input type="button" value="Run"/></p> <p>Prog. Time: <input type="text"/></p> <p>N° of PASS: <input type="text" value="0"/></p> <p>N° of FAIL: <input type="text" value="0"/></p> <p>Status: IDLE</p> </div> </div> | | |
| <div style="display: flex; justify-content: space-around;"> <div style="width: 45%;"> <h3 style="text-align: center;">Channel: 3</h3> <p>Device: <input type="text" value="AT25QL641"/></p> <p>Binary File: <input type="text" value="8MB.frb"/></p> <p style="text-align: center;"><input type="button" value="Run"/></p> <p>Prog. Time: <input type="text"/></p> <p>N° of PASS: <input type="text" value="0"/></p> <p>N° of FAIL: <input type="text" value="0"/></p> <p>Status: IDLE</p> </div> <div style="width: 45%;"> <h3 style="text-align: center;">Channel: 4</h3> <p>Device: <input type="text" value="AT25QL128A"/></p> <p>Binary File: <input type="text" value="16MB.frb"/></p> <p style="text-align: center;"><input type="button" value="Run"/></p> <p>Prog. Time: <input type="text"/></p> <p>N° of PASS: <input type="text" value="0"/></p> <p>N° of FAIL: <input type="text" value="0"/></p> <p>Status: IDLE</p> </div> </div> | | |
| <div style="display: flex; justify-content: space-around;"> <div style="width: 45%;"> <h3 style="text-align: center;">Channel: 5</h3> <p>Device: <input type="text" value="AT25QL641"/></p> <p>Binary File: <input type="text" value="8MB.frb"/></p> <p style="text-align: center;"><input type="button" value="Run"/></p> <p>Prog. Time: <input type="text"/></p> <p>N° of PASS: <input type="text" value="0"/></p> <p>N° of FAIL: <input type="text" value="0"/></p> <p>Status: IDLE</p> </div> <div style="width: 45%;"> <h3 style="text-align: center;">Channel: 6</h3> <p>Device: <input type="text" value="AT25QL128A"/></p> <p>Binary File: <input type="text" value="16MB.frb"/></p> <p style="text-align: center;"><input type="button" value="Run"/></p> <p>Prog. Time: <input type="text"/></p> <p>N° of PASS: <input type="text" value="0"/></p> <p>N° of FAIL: <input type="text" value="0"/></p> <p>Status: IDLE</p> </div> </div> | | |
| <div style="border: 1px solid gray; padding: 5px;"> <p style="text-align: center;"><input type="button" value="Send Project to FlashRunner"/></p> <hr/> <p style="text-align: center;">General Information</p> <p>Project Name: <input type="text" value="at25q.prj"/></p> <p>Operator Name: <input type="text"/></p> <hr/> <p style="text-align: center;">Channels Information</p> <p>N° of Runs: <input type="text" value="0"/></p> <p>N° of PASS: <input type="text" value="0"/></p> <p>N° of FAIL: <input type="text" value="0"/></p> <hr/> <p style="text-align: center;">Project Information</p> <p>N° of Cycles: <input type="text" value="0"/></p> <p>N° Cycle PASS: <input type="text" value="0"/></p> <p>N° Cycle FAIL: <input type="text" value="0"/></p> <p>PASS Percentage: <input type="text" value="---"/></p> <p>Avg. Cycle Time: <input type="text" value="00:00.000"/></p> <p>Max. Cycle Time: <input type="text" value="00:00.000"/></p> <p>Min. Cycle Time: <input type="text" value="00:00.000"/></p> <p>Last Cycle Time: <input type="text" value="00:00.000"/></p> <p style="text-align: right;"><input type="button" value="Clear All"/></p> <hr/> <p style="text-align: center;">Control Room</p> <p><input type="checkbox"/> Stress Test <input type="button" value="Run Once"/></p> <p><input type="checkbox"/> Sync. Channels</p> <p><input type="checkbox"/> Limited to: <input type="text" value="100"/></p> </div> | | |

After opening a project, into 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 2.0.
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. **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.

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

;DEVICE: MT35XU512
;DRIVER: SERMEM8 01.00

!ENGINEMASK 0x00000001
#LOADDRIVER libsermem8.so MICRON MT35X MT35XU512
#TCSETDEV VDDMIN 1700
#TCSETDEV VDDMAX 2000
#TCSETDEV FOSCMIN 0
#TCSETDEV FOSCMAX 0
#TCSETDEV FPLLMIN 0
#TCSETDEV FPLLMAX 0
#TCSETDEV MCUID 0x0000
#TCSETDEV IDCODE 0x001A5B2C
#TCSETDEV IDCODE_MSK 0x00FFFFFF
#TCSETDEV CORE DUMMY
#TCSETDEV MEMMAP 0 F 0 0x00000000 0x03FFFFFF 0x00000000 0x00000100 0 0 0x0 0x0 0xFF 0x0 0
#TCSETDEV SWAP_BYTES NO
!CRC 0x9C7C1E7D
#TCSETPAR ENABLE_DDR YES
#TCSETPAR PROTCCLK 25000000
#TCSETPAR PWDOWN 100
#TCSETPAR PWUP 100
#TCSETPAR RSTDOWN 100
#TCSETPAR RSTDRV OPENDRAIN
#TCSETPAR RSTUP 100
#TCSETPAR VPROG0 1800
#TCSETPAR CMODE OCTO-SPI
#TPSETSRC 64MB.frb
#TPSTART
#TPCMD CONNECT
#TPCMD MASSERASE F
#TPCMD BLANKCHECK F
#TPCMD PROGRAM F
#TPCMD VERIFY F R
#TPCMD DISCONNECT
#TPEND
```

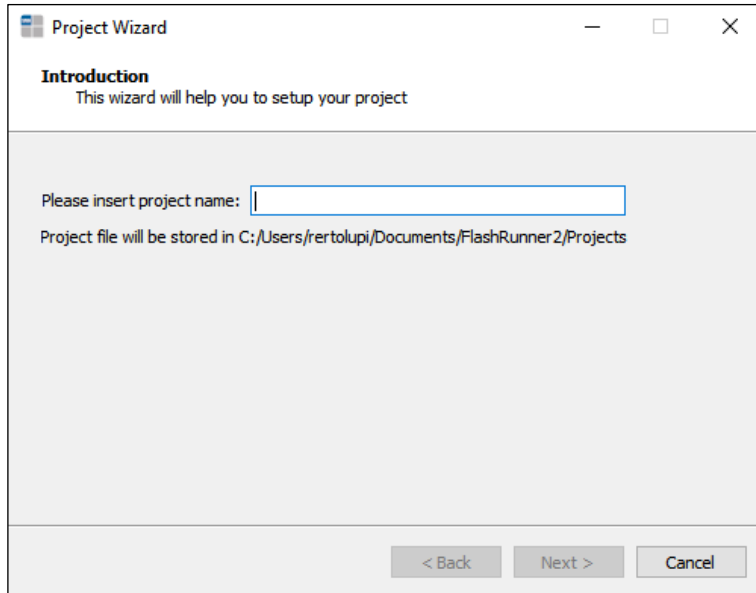
3.8 Wizard

FlashRunner 2.0 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 4.4.58) which, of course, could be sent one by one through our interface library, through the serial port or through “Terminal” tool of FlashRunner 2.0 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 2.0 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 2.0 data folder (which can be found or changed on Tools → Settings menu items, “Paths” tab) and must be uploaded to FlashRunner 2.0 before executing it.

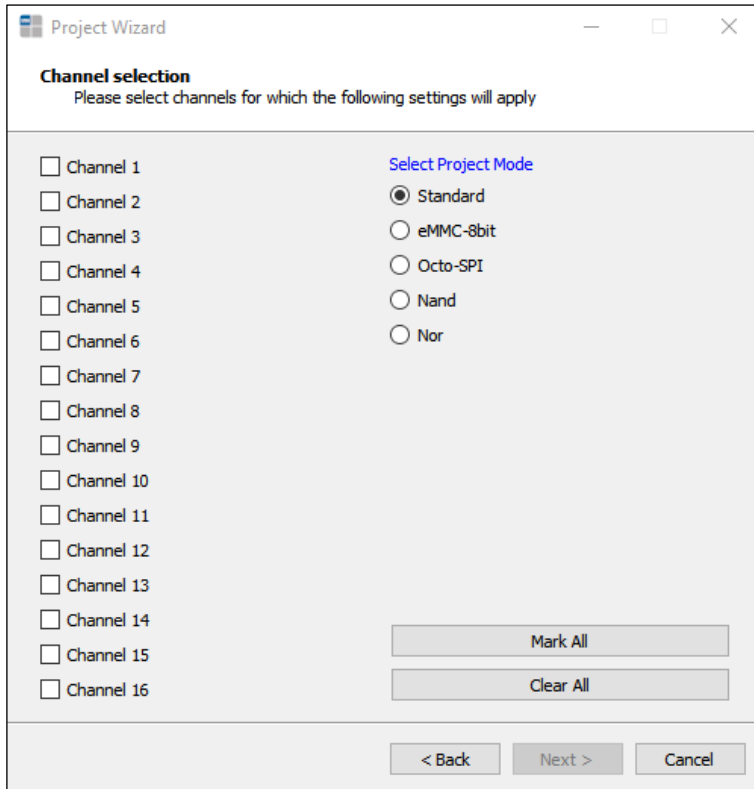
3.8.1 Introduction page

You can create a new project using File → New Project. The first wizard page will let you set the project file name.



3.8.2 Channel selection page

Next page, Channel selection, will let you define channels set for which the following settings will apply.

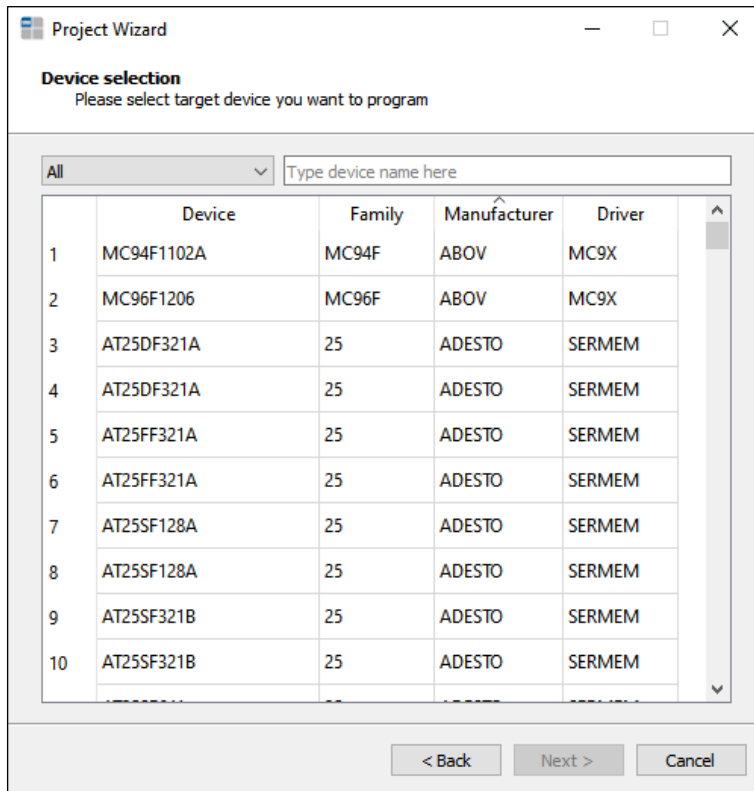


If FlashRunner 2.0 Workbench is connected to the PC, you'll have a set of checkboxes enabled depending on how many channels are enabled on FlashRunner 2.0.

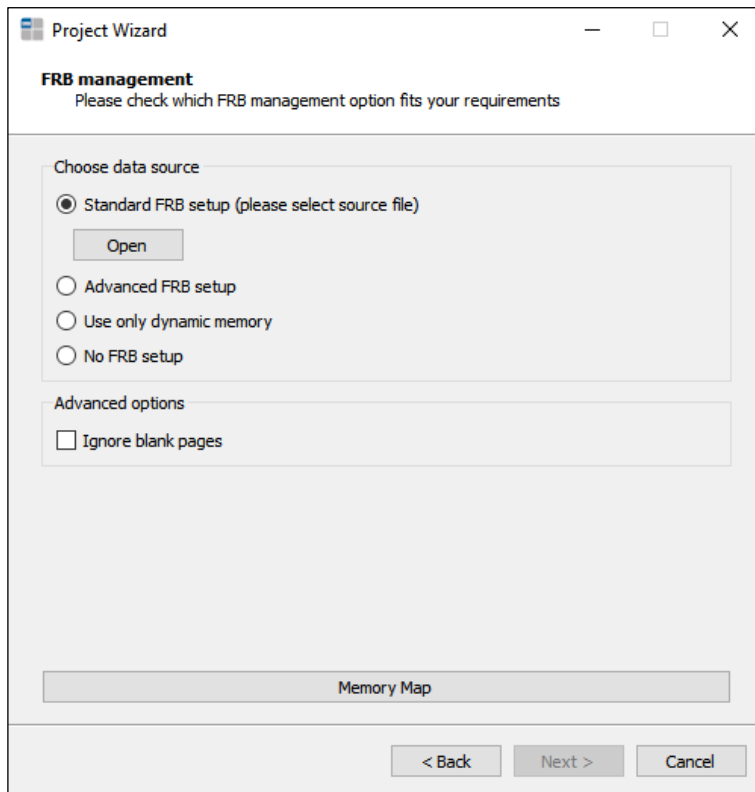
If you leave unticked some checkboxes, at the end of the wizard you'll have the chance to start over and set up all the remaining undefined channels.

3.8.3 Device selection page

This page will let you select which target device you want to program for channels defined on the last page. Remember that each device needs its library, wrote in “Driver” column, make sure to have this library, in case ask SMH Technologies technical support at support@smh-tech.com for an updated version.



3.8.4 FRB Management page



As you can see in the figure, in the FRB management page, you'll find some options about FRB usage. First, you can choose the source:

- **Standard FRB setup:** this will let you select a source file and convert it with just a single click.
- **Advanced FR setup:** this will open a new window that will let you manage advanced features about FRB file.
- **Use only dynamic memory:** this won't create any FRB file, it will only use the dynamic memory.
- **No FRB setup:** this will set no FRB files.

The “**Standard FRB setup**” is the fastest way of achieving source file to FRB conversion: simply select the source. FRB will be created and saved in the standard user data folder with the same filename as the selected source file.

You can also select an existent FRB: in this case, in place of conversion, the tool will read FRB and load its content.

The “**Advanced FRB setup**” is a powerful tool to create an FRB file (i.e. FlashRunner Binary file) which will contain all the source files (more than one are allowed) needed to program target device with your firmware. You can reach this tool via Project Wizard or by selecting Tools → FRB Manager. There's an important difference between these two paths: converting an FRB through the Project Wizard will let you create an FRB using selected target device memory addresses as reference, using it through Tools → FRB Manager it doesn't. Moreover, once created the FRB file, Project Wizard will define an address map which will filter programming commands available. FRB management is described in detail on ch 3.15.

It is also possible to set the advanced option “**Ignore blank page**”: this allows FlashRunner 2.0 to skip pages without any data different from the blank value. Sometimes this feature can improve flashing times, according to the device characteristics.

On 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 3.13.

3.8.5 Communication settings page

This page allows the user to choose the communication protocol, the protocol frequency and other settings about the target device's internal clock.

3.8.6 Delay settings page

This page allows the user to set the power-up and power-down timings and the timings about the signals on the reset line. On this page the user can also choose the working mode of the reset line between open-drain and push-pull.

3.8.7 Powering settings page

This page allows the user to set the values of VPROG0 and VPROG1 and their tolerance values. On this page it's also possible to set the relay barrier usage.

3.8.8 Additional parameters page

This page contains some additional parameters related to the device which can be set by the user.

3.8.9 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.

3.8.10 Additional commands page

This page contains some additional commands related to the device which can be set by the user.

3.8.11 Finish page

From this page, the user can choose if the project is concluded or if there are other target devices to add to the project.

3.9 Encrypt FRB

An existing FRB could be encrypted through FlashRunner 2.0 Workbench software. You simply have to click on the Encrypt FRB button of the left toolbar (see ch 3.4) and choose the FRB file you want to encrypt. FlashRunner 2.0 will provide a new file in the same folder, with the same filename and .frs extension, which is the encrypted version of the original FRB.

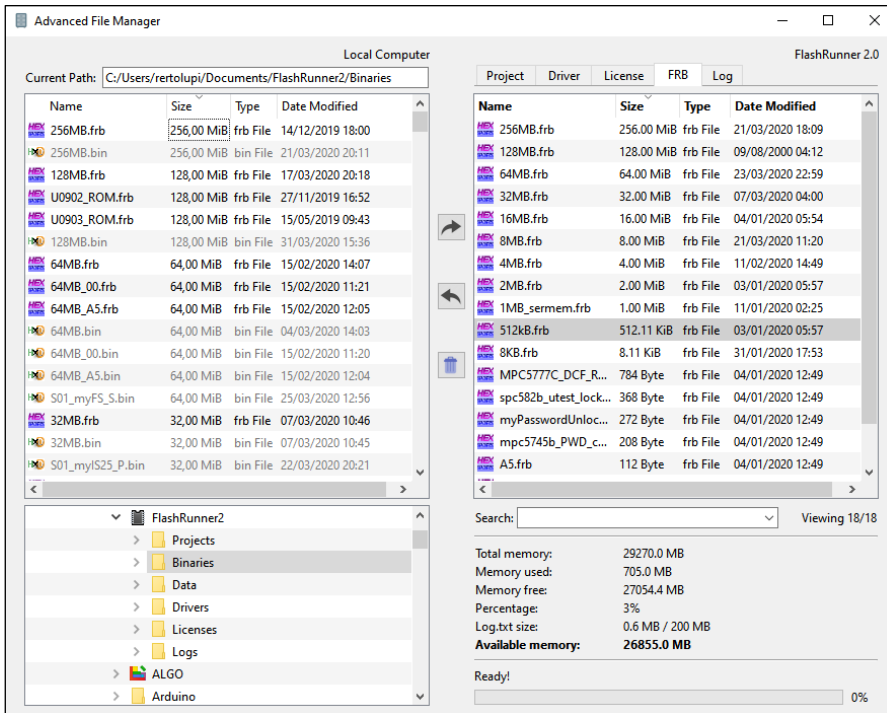
If you have a project which uses the original FRB file and you want to substitute it with its encrypted version, please modify the project file with the project editor at the #TPSETSRC command line. Then send both the project and FRS file to FlashRunner 2.0.

The encryption method implemented is AES256.



Note: *once encrypted, the FRB file can't be decrypted anymore.*

3.10 Advanced file manager



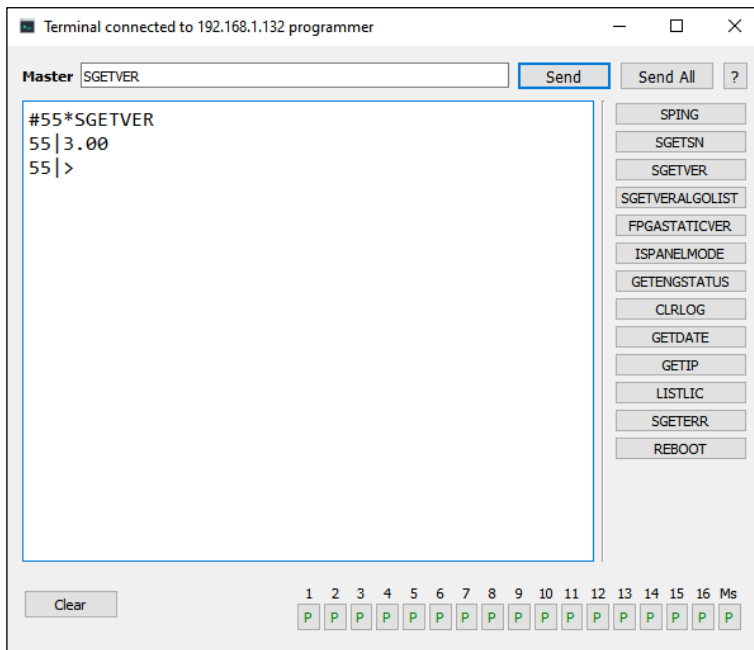
Advanced File Manager is an easy tool for updating or retrieving files to/from connected FlashRunner 2.0. On the left side you'll find your local resources, on the right side you'll find FlashRunner 2.0 resources, in which only five folders are available and are shown as tabs.

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 2.0 folder and click "Get" button.

On the bottom of the right side you can also see the memory usage of your FlashRunner 2.0:

- **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.

3.11 Terminal



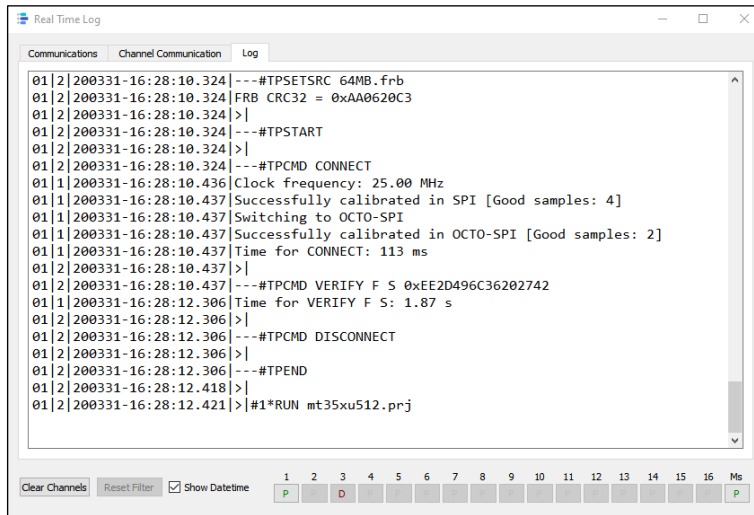
Host pc interacts with FlashRunner 2.0 via synchronous serial communication. Host send commands and receive answers, for detailed information regarding communication syntax and available commands please see ch 4.

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.

3.12 Log



The Real-Time Log feature shows the complete tracking of FlashRunner 2.0. 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 2.0, 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|>|

```


Command Reference

Log Level is a number from 1 up to 6 and define logging verbosity level. Level 1 is the most concise, level 6 is the more verbose. You can change log verbosity with SETLOGLEVEL command (check ch 3.12).

Timestamp shows in which moment a command has been executed. Syntax used for timestamp is:

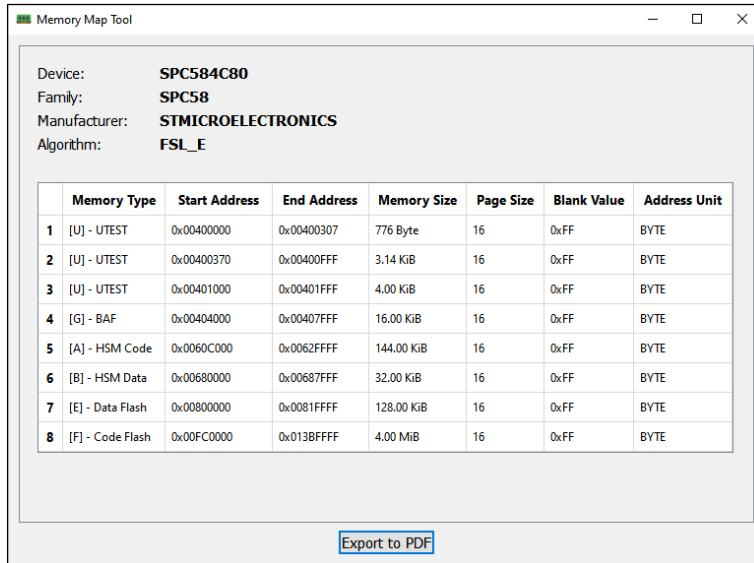
`<year><month><day>-<hour>:<min>:<sec>.<millisec>`

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.

3.13 Memory Map tool

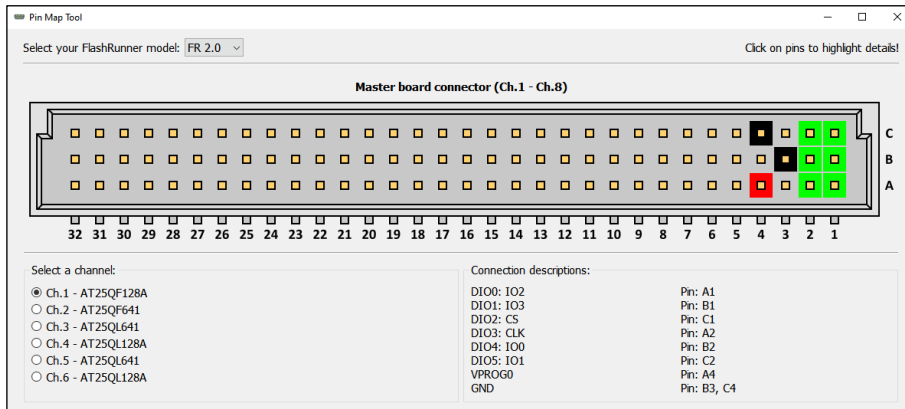
This tool show the memory map of each device included into the project. The interface is very simple and contains a lot of useful informations about the memory of the device.



The screenshot shows the 'Memory Map Tool' window. At the top, it displays device information: Device: SPC584C80, Family: SPC58, Manufacturer: STMICROELECTRONICS, and Algorithm: FSL_E. Below this is a table with 8 rows of memory regions. Each row includes a number, a memory type in brackets, start and end addresses, memory size, page size, blank value, and address unit. At the bottom of the window, there is an 'Export to PDF' button.

| | 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 | [G] - BAF | 0x00404000 | 0x00407FFF | 16.00 KiB | 16 | 0xFF | BYTE |
| 5 | [A] - HSM Code | 0x0060C000 | 0x0062FFFF | 144.00 KiB | 16 | 0xFF | BYTE |
| 6 | [B] - HSM Data | 0x00680000 | 0x00687FFF | 32.00 KiB | 16 | 0xFF | BYTE |
| 7 | [E] - Data Flash | 0x00800000 | 0x0081FFFF | 128.00 KiB | 16 | 0xFF | BYTE |
| 8 | [F] - Code Flash | 0x00FC0000 | 0x013BFFFF | 4.00 MiB | 16 | 0xFF | BYTE |

3.14 Pin Map Tool



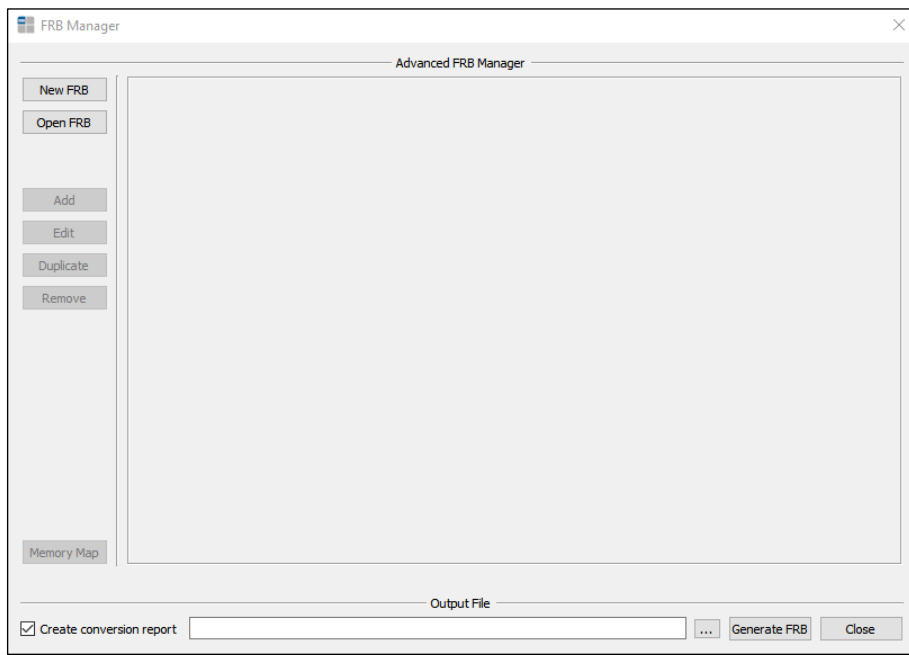
PinMap tool is a handy feature that helps users to do cable wirings from the target device to FlashRunner 2.0 ISP connector. Clicking on one of the channels available in list will load a table on the right side of the window, which lists all signals involved for device connection on that specific channel. 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 2.0 has one or two ISP connectors based on product version: FlashRunner 2.0 versions with 8 or less active channels will have only one ISP connector, FlashRunner 2.0 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.

3.15 Advanced FRB Manager

The Advanced FRB Manager is a powerful tool to create an FRB file (i.e. FlashRunner Binary) which contains all the source files (more than one are 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 Project Wizard will let you create an FRB using selected target device memory addresses as reference, using it through Tools → FRB Manager instead, it doesn't. Moreover, once created the FRB file, Project Wizard will define an address map which will filter programming commands available.

FRB Manager can convert the most common source file formats: RAW Binary; Intel Hex and Motorola SREC.

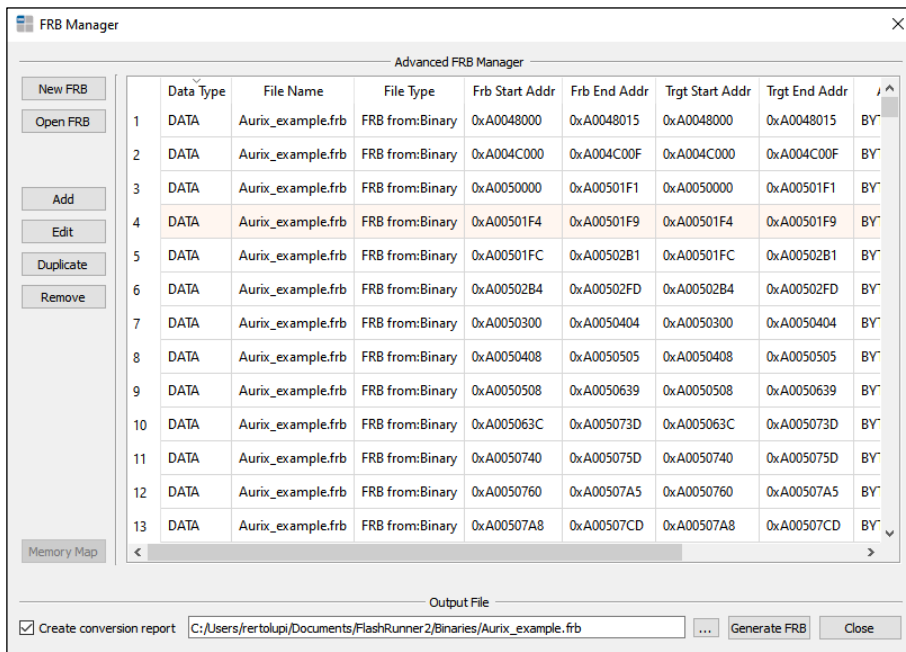


Command Reference

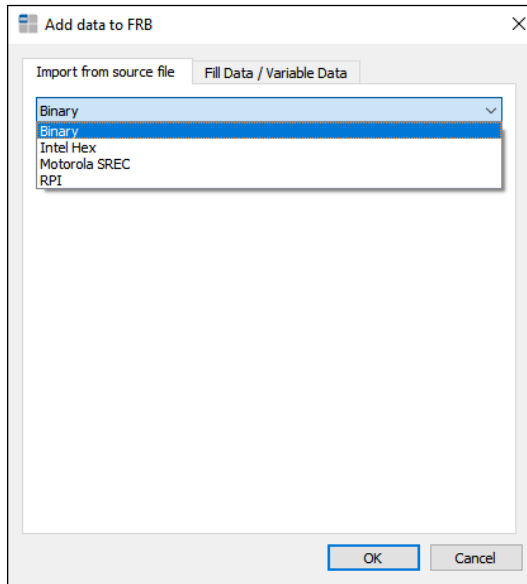
Advanced FRB setup will enable full features to users in order to let them compose their own FRB file. Users can import multiple source files, edit single blocks start address and size, remove blocks and add “fill” or “variable data” blocks.

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.



3.15.1 Add data to FRB: import from source file



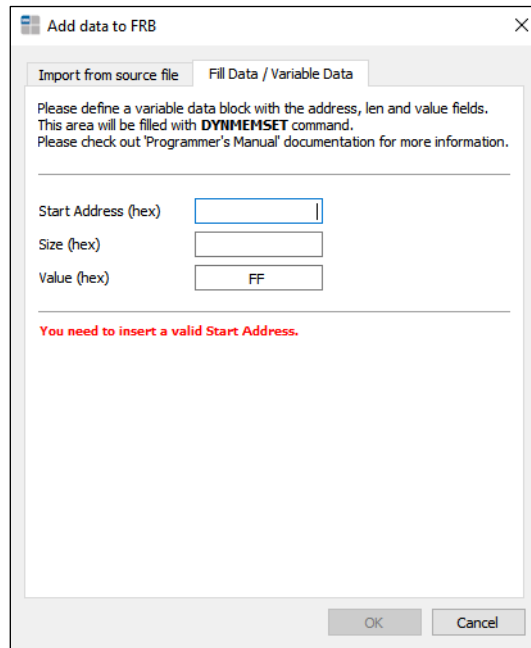
When the user clicks the “Add” button a new window will appear, this new window contains two tabs.

The first one is used to import data from a source file and, by choosing this option, the user can import a source file defined by the format selected from the list.

When choosing 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).

3.15.2 Add data to FRB: Fill Data / Variable Data



From the “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).

3.15.3 Edit FRB block

Source Address Setup (Byte):

Original Start Address: 0xA00501F4

Original Size: 0x6 - [6 Byte]

New Start Address: ✓

New Size: 0x6 - [6 Byte]

Link to target address

Source Start Address is Valid.

Target Address Setup (Byte):

Original Target Start Address: 0x00000000

New Target Start Address: ✓

Target Start Address is Valid.

Target Size Setup (Byte):

Remaining Size: 0x6 - [6 Byte]

New Target Size: ✓

Block Inserted: [0xA00501F4 - 0xA00501F9].

Restore original settings

OK Cancel

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.

Command Reference



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 FRB Manager through the Project Wizard, the memory map of the device 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 2.0 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 2.0. The default value is the full block length.

4 FlashRunner 2.0 Commands

4.1 Overview

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 by its “Control connector” which are able to 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 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 command-line utility (like Terminate© on Microsoft Windows©);
- By using any programming language that is able to 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 own application (see “Projects” chapter).

Alternatively, you can use the FlashRunner 2.0 Workbench software to send commands to the instruments.



Note (for TCP/IP command-line utilities):

FlashRunner 2.0 factory IP address is 192.168.1.100 and data is exchanged on port 1234.

4.1.2 Standalone Mode

In Standalone mode, FlashRunner 2.0 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 2.0 memory must be executed.

A project is simply a text file containing a sequence of FlashRunner 2.0 interface commands, plus some project-specific directives. Projects are explained in detail in the ch 4.4.58.

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 2.0 Terminal tool automatically adds this character). Each command has different coverage, described in chapter 4.3, but at first glance, a command could be sent to:

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

A command sent to a single engine begins with # character followed by <channel number>, followed by * character, followed by the command name, followed by a space, followed by zero, one or more parameters (separated by a space), a Carriage Return character and a final Line Feed character. All parts of the command are case sensitive. Channels' number starts from 1 up to 16 plus the master engine which is 55.

A command sent to all engines in parallel begins with # character, followed by the command name, followed by a space, followed by zero, one or more parameters (separated by a space), a Carriage Return character and a final Line Feed character.

A command sent to a subset of site engines begins with # character followed by <engine mask>, followed by | character, followed by the command name, followed by a space, followed by zero, one or more parameters (separated by a space), a Carriage Return character and a final Line Feed character, where <engine mask> is a decimal number which identifies bitwise channels on which command must be executed.

FlashRunner 2.0 Workbench software can send commands via the Terminal tool, which is automatically adding #<channel number>*, or #<channels subset bitwise mask>| prefix. Before sending a command, please click on the bottom right side of the window the channel for which you want to send the command. 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 name, followed by a space, followed by zero, one or more parameters (separated by a space), a Carriage Return character and a final Line Feed character.

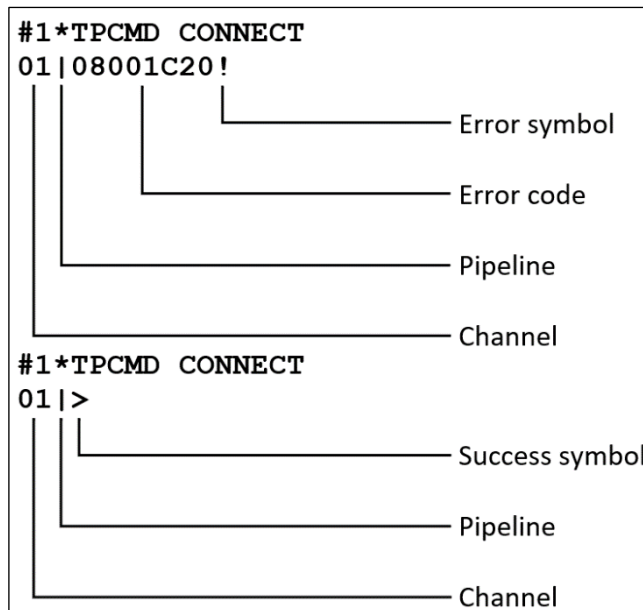
Command Reference

4.2.2 Receiving the Answer

After receiving a command from the host system and executing it, FlashRunner 2.0 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 2.0 command executes successfully, FlashRunner 2.0 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 2.0 command generates an error, FlashRunner 2.0 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 2.0 dynamic memory. These two examples below are equivalent:

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

Numeric parameters returned by FlashRunner 2.0 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 2.0 commands. Each command is fully described in the “Command Reference” section. The “Type” column describes if the command will work on channel engines (“S”) or for the master engine only (“M”).

Command Reference

| Command Syntax | Description | Scriptable | Type |
|-----------------------------|---|------------|-------|
| File System Commands | | | |
| CLRERR | Clears the errors stack | NO | M + S |
| CLRLOG | Clears the log file | NO | M |
| DELAY | Stop every operation for an interval | YES | M |
| FSEXIST | Check if a file does exist inside FlashRunner memory | NO | M |
| FSCRC | Return the CRC32 value of a file | NO | M |
| FSGETCONTROL | Read control interface value | NO | M |
| FSGETFILE | Gets file from FlashRunner | NO | M |
| FSLs | Lists files | NO | M |
| FSLs2 | Lists files with more details | NO | M |
| FSRM | Remove file | NO | M |
| FSSETCONTROL | Set control interface value | NO | M |
| GETDATE | Returns the actual FlashRunner date/time | NO | M |
| GETFILE | Returns file from FlashRunner | NO | M |
| GETFREEMEM | Show details about memory usage | NO | M |
| GETLOGLEVEL | Gets the log verbosity level | NO | M + S |
| GETIP | Returns the FlashRunner IP address, netmask and gateway | NO | M |
| GETVPROG | Read a power line value | NO | S |
| HELP | Shows help table for a driver | NO | S |
| ISMEMENOUGH | Check if there is enough memory | NO | M |
| ISPANELMODE | Returns FlashRunner working mode | NO | M |
| LISTLIC | Returns licenses list | NO | M |
| LOGIN | Login a user account | NO | M |
| LOGOUT | Logout a user account | NO | M |
| REBOOT | Reboot programmer | NO | M |
| SETADMINPWD | Set administrator password | NO | M |
| SETDATE | Gets the actual FlashRunner date/time | NO | M |

FlashRunner 2.0 Workbench

| Command Syntax | Description | Scriptable | Type |
|--|--|------------|-------|
| SETDIO | Set the output state of DIO | YES | S |
| SETIP | Sets up FlashRunner IP address | NO | M |
| SETLOGLEVEL | Sets the log verbosity level | NO | M + S |
| SETMUX | Drives demultiplexer | NO | M |
| SETPANELMODE | Change FlashRunner working mode | NO | M |
| TESTVPROG | Sets up a defined value on VPROG lines | NO | S |
| Status Commands | | | |
| GETENGSTATUS | Gets actual engine status | NO | M |
| SGETENG | Returns the activated engines number | NO | S |
| SGETERR | Returns detailed error information | NO | M + S |
| SGETSN | Returns the FlashRunner serial number | NO | M |
| SGETVER | Gets version | NO | M |
| SGETVERALGO | Returns driver version | NO | M |
| SGETVERALGOLIST | Gets entire driver list with version | NO | M |
| SPING | Pings instrument | NO | M |
| RSTENGSTATUS | Resets engine status | NO | M + S |
| Dynamic Memory Commands | | | |
| DYNMEMCLEAR | Clears dynamic memory | YES | S |
| DYNMEMSET <start addr> <len> <data> <data> ... | Defines dynamic data | YES | S |
| DYNMEMSET2 <start addr> <len> <data stream> | Defines dynamic data | YES | S |
| DYNMEMSETW <start addr> <len> <data> <data> ... | Defines dynamic data (word addressing) | YES | S |
| DYNMEMSETW2 <start addr> <len> <data stream> | Defines dynamic data (word addressing) | YES | S |
| FRB Management Commands | | | |
| FRBREADCRC | Read FRB CRC value | NO | M + S |
| Target Configuration Commands | | | |
| TCSETDEV <dev setting name> <dev setting value> | Sets target device information | YES | S |

Command Reference

| Command Syntax | Description | Scriptable | Type |
|---|--|------------|------|
| TCSETPAR <par name> <par value> | Sets target device parameter | YES | S |
| LOADDRIVER <driver> <silicon> <family> <device> | Sets target device | YES | S |
| UNLOADDRIVER | Reset target before updating a driver | YES | S |
| RLYCLOSE | Closes the specified relay | YES | S |
| RLYOPEN | Opens the specified relay | YES | S |
| VOLTAGEMONITOR <parameter> <value> | Sets working mode of the voltage monitor | YES | S |
| Target Programming Commands | | | |
| TPCMD <command> [par1] [par2] ... [parn] | Executes programming command | YES | S |
| TPEND | Ends programming sequence | YES | S |
| TPSETDUMP <filename> | Sets data destination | YES | S |
| TPSETSRC <filename> | Sets data source | YES | S |
| TPSTART | Starts programming sequence | YES | S |
| Script Execution Commands | | | |
| RUN <script file> | Executes the specified script | NO | S |
| Pseudo commands | | | |
| ENGINE | Select an engine | YES | S |
| 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.

4.4.1 Command Documentation Conventions

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.

4.4.2 CLRERR

Command syntax:

CLRERR

Scriptable: No

Available on: Master and site engines

Parameters:

None.

Answer data:

Success: none.

Error: none.

Description:

Clears the error stack.

Example:

```
#55 *CLRERR
```

```
55 |>
```

4.4.3 CLRLOG

Command syntax:

CLRLOG

Scriptable: No

Available on: Master engine only

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

Parameters:

`ms:` milliseconds to wait

Answer data:

Success: none.

Error: the error code.

Description:

Insert a <ms> delay between FlashRunner 2.0 operations.

Example:

```
#1*DELAY 2000
```

```
1|>
```

4.4.5 DYNMEMCLEAR

Command syntax:

```
DYNMEMCLEAR <start addr> <len>
```

Scriptable: Yes

Available on: Site engines only

Parameters:

start addr: (optional) address of the dynamic memory to start clearing data to.
len: (optional) 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:

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

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

4.4.6 DYNMEMSET

Command syntax:

```
DYNMEMSET <start addr> <len> <data> ... <data>
```

Scriptable: Yes

Available on: Site engines only

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 **addr**. For devices which defines size in words (check it out on Memory Map tool of FlashRunner WorkBench), see the command DYNMEMSETW. Dynamic memory is a special memory area (embedded in the FlashRunner electronics) which is typically used for storing temporary, variable data (e.g. serial numbers) before programming it to the target device. Dynamic memory 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|>
```

4.4.7 DYNMEMSET2

Command syntax:

```
DYNMEMSET2 <start addr> <len> <data stream>
```

Scriptable: Yes

Available on: Site engines only

Parameters:

start addr: address of the target device to start writing data to.
len: number of bytes to write (See the description below for the maximum value supported).
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 (embedded in the FlashRunner electronics) which is typically used for storing temporary, variable data (e.g. serial numbers) before programming it to the target device. Dynamic memory 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|>
```


Command Reference

4.4.8 DYNMEMSETW

Command syntax:

```
DYNMEMSETW <start addr> <len> <data> ... <data>
```

Scriptable: Yes

Available on: Site engines only

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 **addr**. This command is only for devices which defines size in words (check it out on Memory Map tool of FlashRunner WorkBench), for other devices see the command DYNMESET. More DYNMEMSET 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|>
```

4.4.9 DYNMEMSETW2

Command syntax:

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

Scriptable: Yes

Available on: Site engines only

Parameters:

start addr: address of the target device to start writing data to.
len: number of words to write (See the description below for the maximum value supported).
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 **addr**. This command is only for devices which defines size in words (check it out on Memory Map tool of FlashRunner WorkBench), for other devices see the command DYNMESET2. 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|>
```

Command Reference

4.4.10 FRBREADCRC

Command syntax:

FRBREADCRC

Scriptable: No

Available on: Master and site engines

Parameters:

None.

Answer data:

Success: none.
Error: the error code.

Description:

Returns CRC value stored inside 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>
```

4.4.11 FSCRC

Command syntax:

```
FSCRC <type> <filename>
```

Scriptable: No

Available on: Master engine only

Parameters:

type: filetype you want to transfer: could be PRJ|LIB|FRB|LIC|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.
The settings used to calculate the CRC32 are:

- Input reflected = off;
- Result reflected = off;
- Initial value = 0;
- Final xor value = 0.

For FRB files it just read the value from its header.

Example:

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

4.4.12 FSEXIST

Command syntax:

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

Scriptable: No

Available on: Master engines only

Parameters:

type: filetype you want to transfer: could be PRJ|LIB|FRB|LIC|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|>
```

4.4.13 FSGETCONTROL

Command syntax:

`FSGETCONTROL`

Scriptable: No

Available on: Master engine only

Parameters:

None.

Answer data:

Success: none.

Error: the error code.

Description:

Retrieves the read value from the lines belonging to control connector

Example:

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

Command Reference

4.4.14 FSLs

Command syntax:

FSLs <type>

Scriptable: No

Available on: Master engine only

Parameters:

type: directory you want to list: could be PRJ|LIB|FRB|LIC|LOG.

Answer data:

Success: the current directory contents.

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|>
```

4.4.15FSL2

Command syntax:

`FSL2 <type>`

Scriptable: No

Available on: Master engine only

Parameters:

type: directory you want to list: could be PRJ|LIB|FRB|LIC|LOG.

Answer data:

Success: the current directory contents.

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*FSL2 PRJ
55|ATXMEGA128A4.prj - 1019 - 743849183
55|teridian.prj - 770 - 1334997983
55|atxmega.prj - 1036 - 1348562783
55|test.prj - 1067 - 1569746783
55|>
```


Command Reference

4.4.16FSRM

Command syntax:

`FSRM <type> <filename>`

Scriptable: No

Available on: Master engine only

Parameters:

type: filetype you want to transfer: could be PRJ|LIB|FRB|LIC.

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|>
```

4.4.17 FSSETCONTROL

Command syntax:

```
FSSETCONTROL <signal name> <signal value>
```

Scriptable: No

Available on: Master engine only

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|>
```

4.4.18 GETDATE

Command syntax:

GETDATE

Scriptable: No

Available on: Master engine only

Parameters:

None.

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|>
```

4.4.19 GETENGSTATUS

Command syntax:`GETENGSTATUS`**Scriptable:** No**Available on:** Master engine only**Parameters:**

None.

Answer data:

Success: current date.

Error: the error code.

Description:

Returns the actual engines status. The answer is composed by 16 characters, one for each channel starting from left, and value could be "P", "R", "F" or "-". "P" character stays for PASS status and means that last programming on this channel passed successfully. "R" character stays for RUN status and means that channel is still executing commands. "F" character stays for FAIL status and means that last programming on this channel failed, "-" character means that on this product, this channel is not enabled. At power up state, there is one more status, represented by "_" character, which means "idle state", so selected channel never executed any command since power up.

Example:

```
#55*GETENGSTATUS
55|P_____
55|>
```

Command Reference

4.4.20 GETIP

Command syntax:

GETIP

Scriptable: No

Available on: Master engine only

Parameters:

None.

Answer data:

Success: none.

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|>
```

4.4.21 GETFREEMEM

Command syntax:

GETFREEMEM

Scriptable: No

Available on: Master engine only

Parameters:

None.

Answer data:

Success: memory usage details.

Error: the error code.

Description:

This command shows memory usage details.

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 at maximum 200MB. If the log file reach 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|>
```

4.4.22 GETLOGLEVEL

Command syntax:

```
GETLOGLEVEL
```

Scriptable: No

Available on: Master and site engines

Parameters:

None.

Answer data:

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

Description:

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

Example:

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

4.4.23 GETVPROG

Command syntax:

GETVPROG

Scriptable: No

Available on: Site engines only

Parameters:

vprog line: vprog line to read for the selected channel. Could be 0|1.

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=50
```

```
01|>
```


Command Reference

4.4.24 HELP

Command syntax:

```
HELP <lib_name.so>
```

Scriptable: No

Available on: Site engines only

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|>
```

4.4.25 ISMEMENOUGH

Command syntax:

`ISMEMENOUGH <size_kB>`

Scriptable: No

Available on: Master engine only

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|>
```

Command Reference

4.4.26 ISPANELMODE

Command syntax:

ISPANELMODE

Scriptable: No

Available on: Master engine only

Parameters:

None.

Answer data:

Panel mode: the status of panel mode. It can be ON, OFF, 2, 3 or 4.

Description:

Returns the status of panel mode.

Example:

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

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

4.4.27 LISTLIC

Command syntax:

LISTLIC

Scriptable: No

Available on: Master engines only

Parameters:

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
*****
STM32F103CB.lic
License type: DEVICE. Only STM32F103CB is activated
Serial Number: 20058 20059
Creation Date: 21.06.2018
Expiration Date: 9999/12/31
Algorithm Name: CORTEX
Manufacturer: STMICROELECTRONICS
Device Code: STM32F103CB
*****
55|>
```

Command Reference

4.4.28 LOADDRIVER

Command syntax:

```
LOADDRIVER <driver name> <silicon name> <family name> <device name>
```

Scriptable: Yes

Available on: Site engines only

Parameters:

driver name: driver filename which supports the selected device.
silicon name: silicon producer name 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|>
```

4.4.29 LOGIN

Command syntax:

```
LOGIN <user> <password>
```

Scriptable: No

Available on: Master engine only

Parameters:

user: username, you can choose between ADMIN|USER
password: USER has dummy password (any value accepted),
ADMIN has dummy password until changed with
SETADMINPW command

Answer data:

Success: none
Error: the error code

Description:

Login a user, which has different command set enabled.

Example:

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

Command Reference

4.4.30 LOGOUT

Command syntax:

LOGOUT

Scriptable: No

Available on: Master engine only

Parameters:

Answer data:

Success: none

Error: none

Description:

It exits from ADMIN account and get back to USER account

Example:

```
#55*LOGOUT
```

```
55|>
```

4.4.31 REBOOT

Command syntax:

REBOOT

Scriptable: No

Available on: Master engine only

Parameters:

Answer data:

Success: none

Error: the error code

Description:

Reboot FlashRunner 2.0

Example:

```
#55*REBOOT
```

```
55|>
```


Command Reference

4.4.32 RLYCLOSE

Command syntax:

RLYCLOSE

Scriptable: Yes

Available on: Site engines only

Parameters:

None.

Answer data:

Success: none.

Error: the error code.

Description:

Drives related channel signal on the Relay control connector in order to close the circuit. Putting this command inside a project will drives signals related to channel subset defined by ENGINEMASK pseudo-command.

Example:

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

4.4.33 RLYOPEN

Command syntax:

RLYOPEN

Scriptable: Yes

Available on: Site engines only

Parameters:

None.

Answer data:

Success: none.

Error: the error code.

Description:

Drives related channel signal on the Relay control connector in order to open the circuit. Putting this command inside a project will drives signals related to channel subset defined by ENGINEMASK pseudo-command.

Example:

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

Command Reference

4.4.34 RUN

Command syntax:

```
RUN <project name>
```

Scriptable: Yes

Available on: Site engines only

Parameters:

project name: project filename to run.

Answer data:

Success: none.

Error: the error code.

Description:

Starts a project stored inside FlashRunner 2.0 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|>
```

4.4.35 RSTENGSTATUS

Command syntax:

RSTENGSTATUS

Scriptable: No

Available on: Master and site engines

Parameters:

Answer data:

Success: none.

Error: none.

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|>
```

Command Reference

4.4.36 SETADMINPW

Command syntax:

```
SETADMINPW <password>
```

Scriptable: No

Available on: Master engines only

Parameters:

password: new password for ADMIN user

Answer data:

Success: none.

Error: none.

Description:

Set up new password value for ADMIN user. This command is available only if you are logged as ADMIN account.

Example:

```
#55*SETADMINPW newpassword  
55|>
```

4.4.37 SETDATE

Command syntax:

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

Scriptable: No

Available on: Master engine only

Parameters:

sec: set seconds.
min: set minutes.
hour: set hours in 24-hour time format.
date: set date.
month: set month.
year: set 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|>
```

Command Reference

4.4.38 SETDIO

Command syntax:

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

Scriptable: Yes

Available on: Site engine only

Parameters:

DIO_num: the number which indicates the DIO, from 0 to 7.
logic_state: 1 to indicate high level, 0 to indicate low level.
reference_mV: the voltage expressed in mV to be used as reference for high level. This parameter is optional if VPROG0 has been already set.

Answer data:

Success: none.
Error: the error code.

Description:

Sets the current DIO to output at the indicated voltage level. 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. In any case, this command doesn't enable the output of VPROG0 line, unless it has been previously enabled. Attention: this command can cause problems if used for DIO lines controlled by the driver.

Example:

```
#1*SETDIO 7 1 3300  
01|>
```

4.4.39 SETIP

Command syntax:

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

Scriptable: No

Available on: Master engine only

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|>
```


Command Reference

4.4.40 SETLOGLEVEL

Command syntax:

```
SETLOGLEVEL <level>
```

Scriptable: No

Available on: Master and site engines

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 |>
```

4.4.41 SETMUX

Command syntax:

SETMUX <level>

Scriptable: No

Available on: Master engine only

Parameters:

level: 0 to isolate all outputs, 1 to enable first bank, 2 to 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 only in combination with Demultiplexer tool, available only for FlashRunner 16 channel version.

Example:

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

Command Reference

4.4.42 SETPANELMODE

Command syntax:

```
SETPANELMODE <level>
```

Scriptable: No

Available on: Master engine only

Parameters:

level: 0 to work in standard mode, 1 to enable panel mode, 2 to enable eMMC 8bit mode, 3 to enable NAND mode, 4 to enable NOR mode

Answer data:

Success: none.
Error: the error code.

Description:

Enable panel mode. If 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 in order to program this kind of devices.

Example:

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

4.4.43 SGETENG

Command syntax:

SGETENG

Scriptable: No

Available on: Site engines only

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>
```

Command Reference

4.4.44 SGETERR

Command syntax:

SGETERR

Scriptable: No

Available on: Master and site engines

Parameters:
None.

Answer data:

Success: the error code stack.
Error: none.

Description:

Returns the error stack related to 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|>
```

4.4.45SGETSN

Command syntax:

SGETSN

Scriptable: No

Available on: Master engine only.

Parameters:

None.

Answer data:

Success: the product serial number.
Error: none.

Description:

Returns the product serial number.

Example:

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

Command Reference

4.4.46 SGETVER

Command syntax:

`SGETVER`

Scriptable: No

Available on: Master engine only.

Parameters:

None.

Answer data:

Success: The Operating System version.

Error: none.

Description:

Returns the Operating System version.

Example:

```
#55*SGETVER
```

```
55|2.31
```

```
55|>
```

4.4.47 SGETVERALGO

Command syntax:
SGETVERALGO

Scriptable: No

Available on: Site engine only.

Parameters:
None.

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 minor release, the other one is the major release

Example:
#1*SGETVERALGO libsermem.so
01|04.02
01|>

Command Reference

4.4.48 SGETVERALGOLIST

Command syntax:

`SGETVERALGOLIST`

Scriptable: No

Available on: Master engine only.

Parameters:

None.

Answer data:

Success: algorithm version list

Error: none.

Description:

Returns algorithm version of all drivers stored inside programmer. Usually answer is a 3-digit number: 2 less significant are 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|>
```

4.4.49 SPING

Command syntax:

SPING

Scriptable: No

Available on: Master engine only.

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|>
```

Command Reference

4.4.50 TCSETDEV

Command syntax:

```
TCSETDEV <par name> <par value>
```

Scriptable: Yes

Available on: Site engines only.

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 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|>
```

4.4.51 TCSETPAR

Command syntax:

```
TCSETPAR <par name> <par value>
```

Scriptable: Yes

Available on: Site engines only.

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|>
```

Command Reference

4.4.52 TESTVPROG

Command syntax:

TESTVPROG

Scriptable: No

Available on: Site engines only.

Parameters:

vprog line: vprog line to read for the selected channel. Could be 0|1.
mV: mV to set in output on selected vprog line for the selected channel.
output: defines if selected vprog line is in output or only defined internally as high reference value. Could be ON|OFF.

Answer data:

Success: ok.
Error: none.

Description:

Sets up a defined value on vprog lines.

Example:

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

4.4.53 TPCMD

Command syntax:

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

Scriptable: Yes

Available on: Site engines only.

Parameters:

command: programming command.

par: zero or more programming command parameters.

Answer data:

Success: programming command specific.

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|>
```

Command Reference

4.4.54 **TPEND**

Command syntax:

TPEND

Scriptable: Yes

Available on: Site engine only.

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|>
```

4.4.55 TPSETDUMP

Command syntax:

```
TPSETDUMP <filename>
```

Scriptable: Yes

Available on: Site engines only.

Parameters:

filename Name of the dump file

Answer data:

Success: none

Error: the error code.

Description:

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

Example:

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


Command Reference

4.4.56 TPSETSRC

Command syntax:

```
TPSETSRC <filename> IGNORE_BLANK_PAGE
```

Scriptable: Yes

Available on: Site engines only.

Parameters:

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

IGNORE_BLANK_PAGE: optional parameter, avoid to program 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 special keyword will set the FlashRunner to use only dynamic memory instead of an FRB file.

Example:

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

4.4.57 TPSTART

Command syntax:

TPSTART

Scriptable: Yes

Available on: Site engines only.

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|>
```

Command Reference

4.4.58 UNLOADDRIVER

Command syntax:

```
UNLOADDRIVER
```

Scriptable: Yes

Available on: Site engines only

Parameters:

None.

Answer data:

Success: none.

Error: the error code.

Description:

Unload the driver to remove dependencies before updating the driver.

Example:

```
#1 *#UNLOADDRIVER
```

```
01 |>
```

4.4.59 VOLTAGEMONITOR

Reference: For detailed information refer to chapter 10.

Command syntax:

```
VOLTAGEMONITOR <parameter>
VOLTAGEMONITOR <parameter> <value>
```

Scriptable: Yes

Available on: Site engines only

Parameters:

| | | |
|-------------------|-----------------------|---|
| parameter: | OFF | <i>pause monitoring</i> |
| value: | none. | - |
| parameter: | ON | <i>start/resume monitoring</i> |
| value: (*) | ERROR_EXIT | <i>exit operations on error</i> |
| value: | ERROR_CONTINUE | <i>log and continue</i> |
| parameter: | DYN_SAMPLE | <i>dynamic sampling mode</i> |
| value: (*) | ENABLED | <i>based on currently active channels</i> |
| value: | DISABLED | <i>constant sampling rate</i> |
| parameter: | READ_AVERAGE | <i>print both VPROG0 and VPROG1 Average Values.</i> |
| value: | VPROG0 | <i>print the selected line</i> |
| | VPROG1 | <i>print the selected line</i> |
| parameter: | CLEAR_AVERAGE | <i>reset both VPROG0 and VPROG1 average values</i> |
| value: | VPROG0 | <i>clear the selected line</i> |
| | VPROG1 | <i>clear the selected line</i> |

(*) default value, if a parameter is omitted

Answer data:

Success: none.

Error: the error code.

Command Reference

Description:

Voltage monitor is enabled by default setting **VPROG** (x) limits:

```
#1*TCSETPAR PROG0LIMITS 50 0 0
```

```
#1*TCSETPAR PROG1LIMITS 100 0 0
```

- Threshold value must be greater than 1% of **VPROG**
- All the described parameters below can be omitted.

Example:

```
#1*VOLTAGEMONITOR DYN_SAMPLE ENABLED      *default
01|>
#1*VOLTAGEMONITOR DYN_SAMPLE DISABLED     user choice
01|>
#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 Masserese [...]
01|>
#1*VOLTAGEMONITOR READ_AVERAGE           print average
01|>                                     for MASSERASE
#1*VOLTAGEMONITOR OFF                    no monitoring
01|>
#1*TPCMD BLANKCHECK F
Time for Blankcheck [...]
01|>
#1*VOLTAGEMONITOR ON_ERROR_EXIT           exit if error
01|>                                     is detected
#1*VOLTAGEMONITOR CLEAR_AVERAGE         reset values
01|>
#1*TPCMD PROGRAM F
Time for Program [...]
01|>
#1*VOLTAGEMONITOR READ_AVERAGE           print average
01|>                                     for PROGRAM
```

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 to FlashRunner 2.0. Projects are usually created with the Project Wizard tool (see ch 3.7 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 0xFFFFFFFF
#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
```

Command Reference

```
!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 commands 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 will bitwise define channels involved. Each number in base 2 defines one channel, starting from less significative. Number value (1 or 0) defines if the channel is selected or not. For example, !ENGINEMASK 0x1A, equals to 00011010 in binary and it means that channels 2, 8 and 9 are selected.

The following section will define the target device (through #LOADDRIVER) and all the specific device information (through #TCSETDEV command). This section will be closed by !CRC command: this number will prevent from altering the information above which contain sensitive data and would compromise the programming operation.

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 order doesn't matter.

The last section is enclosed between #TPSTART and #TPEND commands and defines which operation will be executed on the target device. These commands are editable, the order does matter and we suggest not changing it once Project Wizard will compile this file.

Commands related to single memory types have the double syntax:

#TPCMD PROGRAM F

Will program automatically memory type areas defined by loaded FRB file.

#TPCMD PROGRAM F 0x0 0x100

Will program memory type areas defined by command parameters above. Target start address is 0x0, length 0x100. If loaded FRB doesn't contain any data in this area target device will not be programmed.

Usually double syntax is available for **PROGRAM**, **VERIFY**, **BLANKCHECK** commands.

5.1 Execution and Termination

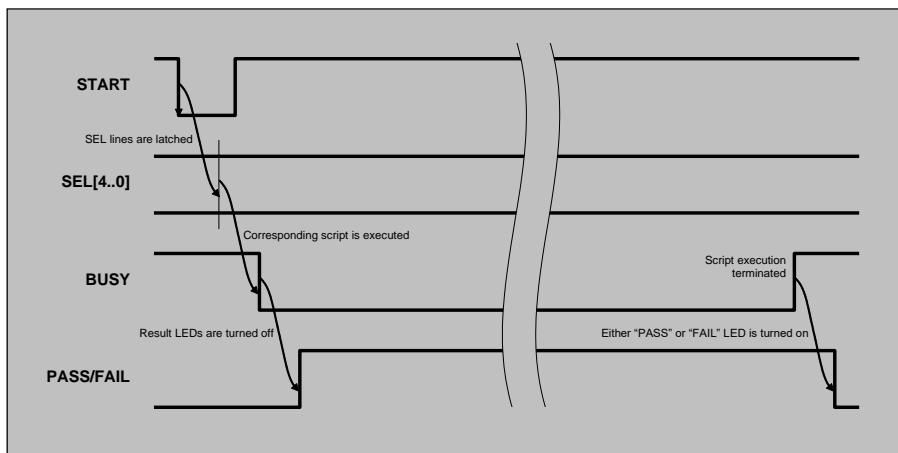
5.1.1 Standalone project execution

FlashRunner 2.0 has a control connector, a group of control lines (SEL[4..0] in the “CONTROL” Connector, for hardware details please refer to FlashRunner 2.0 User's Manual) determines in binary logic a decimal number from 0 to 31 which will execute project named project0.prj....project31.prj.

The event that triggers script execution is the START control line becoming active (while the BUSY line is not active). This line can be easily driven by the ATE control logic.

When FlashRunner 2.0 begins executing a project, “BUSY” LED turns on.

The following diagram illustrates the typical temporal relations between the various FlashRunner 2.0 control lines.



5.1.2 Remote projects execution

Additionally, projects can be manually executed in host mode. `RUN` command (see ch 4.4.34) executes a specified project.

5.1.3 Projects Termination

Project execution ends either after FlashRunner 2.0 has executed the last project command or immediately after the first failing project command.

5.2 Project-Specific Directives

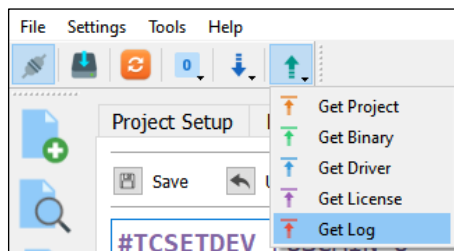
FlashRunner 2.0 commands contained in a project are executed sequentially, exactly as they would be executed in Host mode. However, projects contain additional directives (not available in Host mode) indicated with “!” prefix which controls how projects are executed. The following table lists these directives.

Each directive is valid from its line forward.

| 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 2.0, project command execution is logged. You can check at the runtime log file (see ch 3.12) 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 2.0 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 2.0 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 familiar 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), statement₁ is executed. If it is false (the expression is zero) and if there is an else part, statement₂ is executed instead.¹

In FlashRunner 2.0 the same goal can be achieved using the syntax below inside any project file:

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

¹ “The ANSI C Programming Language” 2nd ed., Brian W. Kernighan and Dennis M. Ritchie, Prentice Hall

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.

Command Reference

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 2.0 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 2.0'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.

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 2.0 will apply this "patch" over FRB data. You can define more than one patch, virtually without limits (physical limit is FlashRunner 2.0 1 GB RAM), but defined data is 16

Command Reference

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 2.0 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 3.15) 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 2.0 will remember DYNAMIC data table until DYNMEMCLEAR command execution or FlashRunner 2.0 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:

```
#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 E03912343484568078809A73
#TPSTART
#TPCMD CONNECT
#TPCMD PROGRAM F
#TPCMD VERIFY F R
#TPCMD DISCONNECT
#TPEND
```

7 Data Protection System

7.1 User management

User management lets users switch between two modes: Administrator and User modes. The Administrator mode is enabled to execute all commands, User instead has access to a subset of all commands.

By default, FlashRunner 2.0 starts in Administration mode, if you want to change the account, you must use the `#LOGIN` command (see ch 4.4.29).

While the User account doesn't have a password (so any value as password parameter for `#LOGIN` command is accepted), the Administrator account could have a password. By default, also Administrator account accepts any value as `#LOGIN` password parameter unless you decide to set it. If you want to set the Administrator password, please use `#SETADMINPWD` (See ch 4.4.36) with the following procedure:

1. Send `#LOGIN ADMIN dummy`
2. Send `#SETADMINPWD new_password`

From this moment on, if you want to login as Administrator you must enter `new_password`. Once the user sets a password for Administration mode, FlashRunner 2.0 will start in user mode. The table below lists the FlashRunner commands available for each user.

| Command | Unsecured Mode | Secured Mode (Administrator) |
|-------------|----------------|------------------------------|
| SETADMINPWD | - | • |
| CLRLOG | - | • |

7.2 FRB encryption

Each FRB could be encrypted using the FlashRunner 2.0 Workbench tool (See ch 3.97.2).

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.

To use it, please, upload .frs to FlashRunner 2.0 (using Advanced File Manager, see ch 3.10) and change #TPSETSRC filename extension on the related project, finally upload the project to FlashRunner 2.0.

8 FlashRunner 2.0 Interface Library

8.1 Overview

This chapter deals with interfacing FlashRunner 2.0 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 2.0 Interface Library

FlashRunner 2.0 Interface Library is a DLL which includes all of the functions that allow you to set up a communication channel with the instrument and send commands to FlashRunner 2.0. 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 2.0 Interface Library contains Visual C++ written routines and 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. 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:

- 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).

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

8.4 Interface Library Reference

8.4.1 Using the Interface Library Functions

When you control FlashRunner 2.0 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.

- **Transfer files to/from FlashRunner 2.0.**
Two dedicated functions, `FR_SendFile()` and `FR_GetFile()`, allow you to transfer a file from the PC to FlashRunner 2.0 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 2.0 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 Application examples

Application examples for Visual C and Visual Basic are provided in the local installation path.

8.4.5 Function Reference for FR 2.0

8.4.6 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 extended error information.

Description:

Closes the communication link with the instrument.

Command Reference

8.4.7 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 `FR_GetLastErrorMessage()` function to get extended error information.

Description:

Receives the answer sent by FlashRunner 2.0 to the PC, in response to the `FR_SendCommand()` function. A `FR_GetAnswer()` function should always follow a `FR_SendCommand()` function.

8.4.8 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 2.0, 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.

Command Reference

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

Description:

Retrieves a file from FlashRunner 2.0 and stores it in a specified local path.

8.4.9 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 2.0 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.

8.4.10 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 **FR_GetLastErrorMessage()** function to get extended error information.

Description:

Creates a communication link with the instrument. Returns a communication handle that must be used by successive FlashRunner 2.0 Interface Library function calls.

8.4.11 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 `FR_GetLastErrorMessage()` function to get extended error information.

Description:

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

8.4.12 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 `FR_GetLastErrorMessage()` function to get extended error information.

Description:

Sends a file from the PC to a specified path of FlashRunner 2.0.

9 FRB Converter

This section explains how to use the `frbconverter.exe` tool from a terminal or a batch script.

The parameters that can be used are:

- **-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.

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.

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`.

Command Reference

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

```
set FRBCONVERTER=C:\Program Files (x86)\SMH  
Technologies\FIashRunner2\frbconverter.exe  
  
set INPUT_FILE=C:\Users\reitolupi\Desktop\myFile.s19  
set OUTPUT_FILE=C:\Users\reitolupi\Desktop\myFile.frb  
  
call "%FRBCONVERTER%" -input "%INPUT_FILE%" -format s19  
-output "%OUTPUT_FILE%"
```

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

Command Reference

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: $VPROG0 = 3300\text{mV}$
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:

$$UVerr = I_s V_{\text{sampled}} < (V_{\text{progSet}} \text{ minus } V_{\text{threshold}})$$

The over-voltage error is detected using the formula:

$$OVerr = I_s V_{\text{sampled}} > (V_{\text{progSet}} \text{ plus } V_{\text{threshold}})$$

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

Command Reference

(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:

```
#TPCMD VOLTAGEMONITOR CLEAR_AVERAGE
[...]
#TPCMD 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
#TPCMD VOLTAGEMONITOR CLEAR_AVERAGE         reset measure
#TPCMD MASSERASE F                           start operation
#TPCMD VOLTAGEMONITOR READ_AVERAGE          log measure
#VOLTAGEMONITOR OFF                          no monitoring
#TPCMD BLANKCHECK F                          start operation
#VOLTAGEMONITOR ON ERROR_EXIT                error detection
#TPCMD VOLTAGEMONITOR CLEAR_AVERAGE         reset measure
#TPCMD PROGRAM F                             start operation
#TPCMD 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.

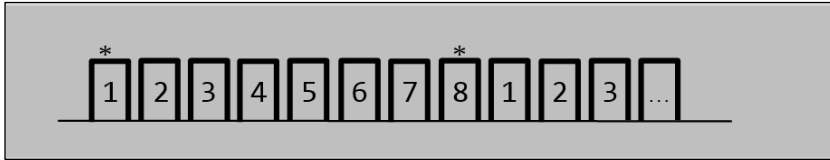
The sampling frequency is proportional to the number of channels currently active and its value is approximately 3.3KHZ when only 1 channel of `vPROG0` is monitored.

If both `vPROG0` and `vPROG1` lines are monitored simultaneously the sampling time increases to 600us and the sampling frequency is approximately 1.6KHz.

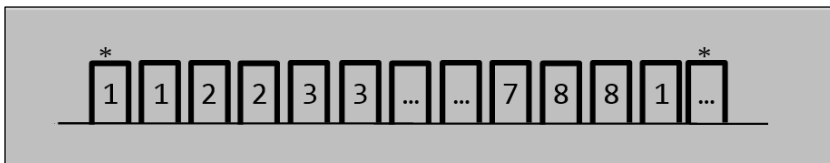
For 8 channels of `vPROG0` monitored only, the sampling frequency is about 400Hz and for 16 channels it is about 200Hz. If `vPROG0` and `vPROG1` are both monitored, the sampling rate for 16 channels is approximately 100Hz per channel.

FlashRunner 2.0 Workbench

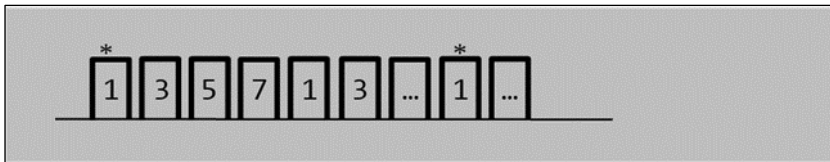
Sampling sequence for 8ch of VPROG0, 300uS per sample:



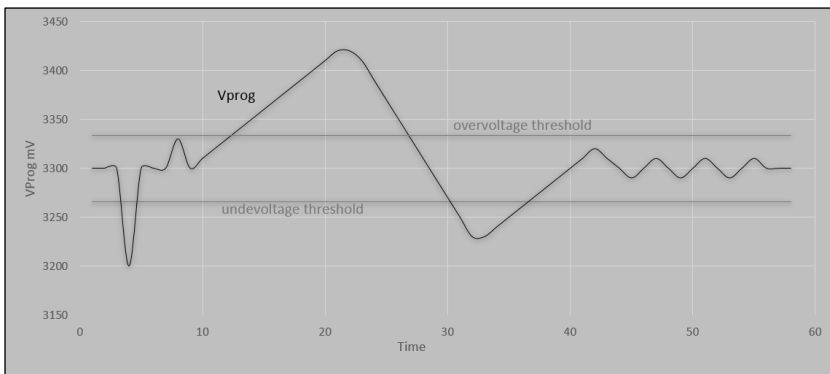
Sampling sequence for 8ch of VPROG0+VROG1, 300uS per sample:



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



Threshold limits:



10.5 Error Types

```
#TCSETPAR PROGOLIMITS 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 FlashRunner 2.0 Internal Memory

FlashRunner 2.0 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 2.0 products are signed below:

| | |
|--------|--|
| 64GB | Lexar microSDXC. Cod: LSDMI64GBBEU633A |
| 128 GB | Kingston MicroSDXC. Cod: DCG2/64GB |
| 256 GB | Micron MicroSDHC. Cod: MTSD256AHC6MS-1WT |

12 Troubleshooting

This section collects a set of troubleshooting techniques to program successfully your device with FlashRunner 2.0.



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.*

12.1 Project execution failures

If you are executing a project and FlashRunner 2.0 answers to project execution with FAIL 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 and check related error description. Usually a failure on “Connect” command execution means that FlashRunner 2.0 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 3.14.
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 2.0 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 2.0 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 2.0 GND line is coupled with the external supplier GND line.
 7. If you are using FlashRunner 2.0 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 2.0 VPROG1 line, you must be sure that board current absorption is less than FlashRunner model maximum current level supported. Please check FlashRunner 2.0 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.

Command Reference

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 3.11 and 4.4.46 for more information)