Version N° 1.0

LaurTec

LTboot

Quick Guide

Author : *Mauro Laurenti*

ID: PJ11008-EN

License

The usage of "LTboot SDK" (Software Development Package) given by Mauro Laurenti (the Author) imply the acceptance of the following license:

- The Software must not be used before reading the documentation.

The binary code within the SDK can be redistributed respecting the following points:

- The official documentation must be distributed with the Software.
- LTboot SDK cannot be used for developing and testing military equipment.
- LTboot SDK can be used for commercial and non commercial applications.
- The license associated with it must be preserved.

"LTboot SDK" is made of different components:

- GUI (Graphical User Interface)
- Dynamic libraries (xxx.dll) to support the GUI
- LTboot Firmware xxx.hex file (compiled for different MCU)

The dynamic libraries can be used and distributed only with LTboot SDK and cannot be integrated and linked within other projects without a written agreement given by the Author.

THE SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS MATERIAL. THE AUTHOR SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

Indice

Introduction

LTboot is a bootloader for microcontrollers. It is made of different components, thus using it requires an understanding of the different parts.

LTboot is made of:

- LTboot Firmware
- LTboot GUI
- LTboot libraries

In the following sections some additional details are provided for the Firmware and GUI applications. The libraries details are outside the scope of this document.

LTboot software environments

LTboot has been developed using different IDEs and compilers, as shown below:

LTboot Firmware

Developed using MPLAB X and XC8 compiler.

LTboot GUI

Developed using Visual Studio 2017.
.NET Framework 4.0
Windows 10 64 bits.

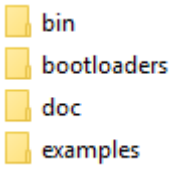
LTboot libraries

Developed using Visual Studio 2017.
.NET Framework 4.0
Windows 10 64 bits.

The software tests have been done within the development environments only, thus if a different one will be used, it may need some fine tuning.

LTboot package

LTboot is distributed as Software Development Kit (SDK) which contains all the required software, firmware, examples and documentation to get started. The SDK folder can be downloaded by www.laurtec.it and can be copied in any working directory.
The SDK folder structure is:



The content of each folder is:

bin

It contains the GUI executable.

bootloaders

It contains the bootloader firmware variants compiled for the different microcontrollers ad options.

doc

It contains the documents associated with the software

examples

It contains compiled examples that can be loaded directly via the GUI within the MCU where the bootloader got already installed.

LTboot Firmware

The core of the LTboot bootloader is the Firmware. The Firmware is written in C and must be compiled for the microcontroller that is used. The Firmware uses LTlib as internal peripheral library so that some section of the code are transparent to the specific microcontroller.
The Firmware must be compiled so that it will get stored and executed right after the MCU reset vector, so it will be the first code that will be executed after a POR (Power On Reset) or an Hardware or Software reset. The bootloader entry point procedure can be selected within the Firmware, but at each code change a new compilation is required. The LTboot Firmware, (hex file) can be flashed within the microcontroller using directly the IDE and programmer, indeed it gets stored right after the reset vector.
The memory range that is reserved to the bootloader is till address 0x1FFF, so the new Reset Vector for the application is shifted to the address 0x2000. The bootloader does not use the entire allocated memory range, allowing flexibility to add additional code without the need of using any special compiler code optimization.
The code that the bootloader has to flash within the MCU, using the GUI, must be properly compiled. Indeed the code that must be loaded within the microcontroller must be compiled to make sure that the allocated flash memory does not overlap the bootloader section. This requires some special settings for the Linker to make sure that the right memory offset is applied for the flash memory, while the Reset and Interrupt Vectors are properly shifted to the new offset memory range. Depending on the working environment (IDE) and compiler version, the settings might be slightly different. For the PIC microcontrollers under MPLAB X is required to adjust both:

- Code offset
- ROM ranges

These configurations can be found under the Linker project options, within the sub configuration:

- Additional Options
- Memory Model

In particular the code offset must be set to *0x2000* as shown in Figure 1, while the ROM range must be set as *default,-2000-FFFF* as shown in Figure 2. After compiling the code with the new settings, the hex code is generated starting from address 0x2000, and the Reset vectors are shifted accordingly.
The .hex file generated by compiling the code should not be loaded directly within the microcontroller via the IDE. The code can be uploaded within the microcontroller by using the LTboot GUI, making sure that the microcontroller already has LTboot Firmware installed.

LTboot entry sequence

LTboot, while is the first code that the MCU executes, will not let the MCU stay in bootloader mode, unless the entry sequence is found. Indeed the bootloader, upon an MCU reset, searches for the entry sequence, if it is found it remains in bootloader mode,

otherwise it jumps to the new Reset Vector *0x2000*. The default entry sequence is pressing a button on PORTB RB4 to ground. So before doing a power cycle or a reset you have to press the button connected to RB4 (BT1 by using the evaluation board Freedom II and Freedom III) and then, by keeping it pressed, reset the MCU. After the reset you can release the button and the bootloader will stay in bootloader mode. The code 0xAA is shown on the PORTD while the bootloader is running.

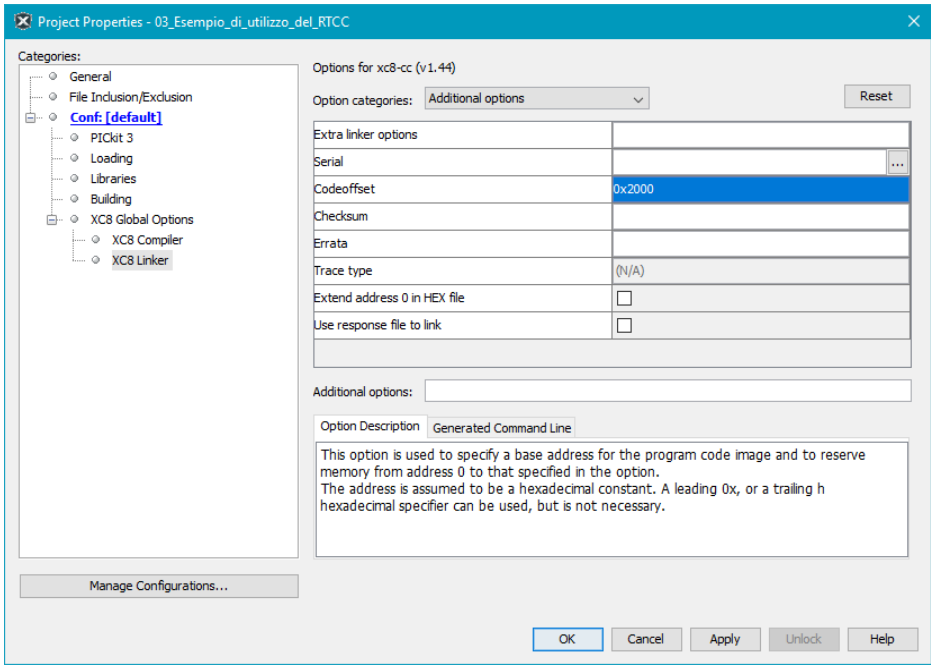


Figure 1: *Code Offset setting.*

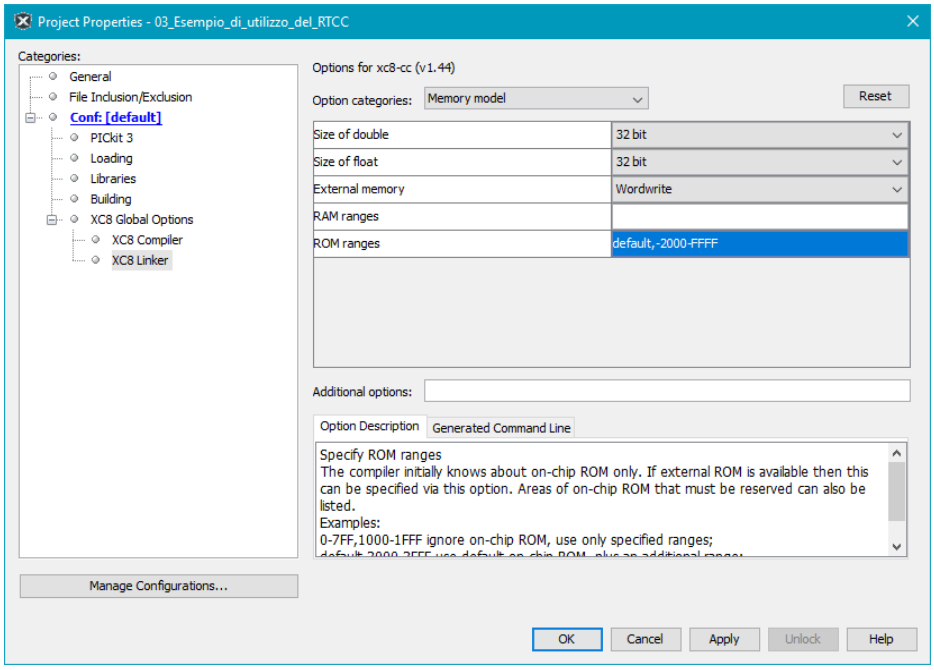


Figure 2: *ROM ranges setting.*

LTboot GUI

The GUI has been designed using .NET environment, making it quite user friendly. The GUI uses ltboot.dll library, offering an easy user interface that exploits all the features made available via the library. In Figure 3 is shown the main GUI window once the application is running.

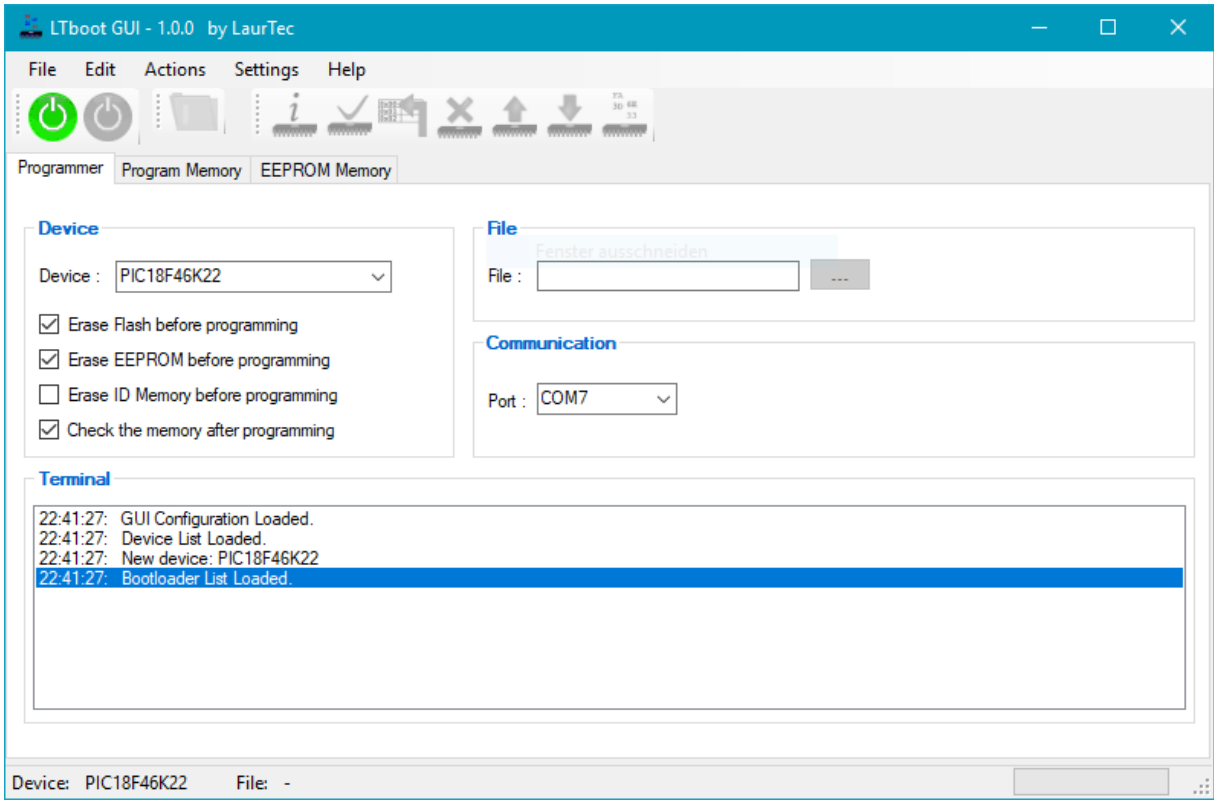


Figure 3: *GUI main Window.*

Before activating the communication via the green button in the Toolbar, you need to make sure that the hardware that provides a serial port and bootloader, is connected to the PC. In Figure 3 is possible to see that the hardware is connected and provides COM 7 as communication port. Once the connection is established the Toolbar gets activated and different services get enabled depending on the specific state, as shown in Figure 4.

Figure 4: *Toolbar once the connection is activated.*

The GUI allows loading .hex files generated for the LTboot bootloader by clicking the Import Hex File icon.
Once the file is selected it requires few seconds before in gets loaded within the GUI. All the memory ranges within the .hex file are loaded within the respective memory Tab.
By default only the Program Memory Tab and EEPROM Memory Tab are shown. If there is interest to see the Configuration and ID memory ranges, it is possible to enable it via the following menu:

`Settings → Advanced Settings`

Once the Advanced Settings are enabled, the additional memory range tabs are enabled as shown in Figure 5.

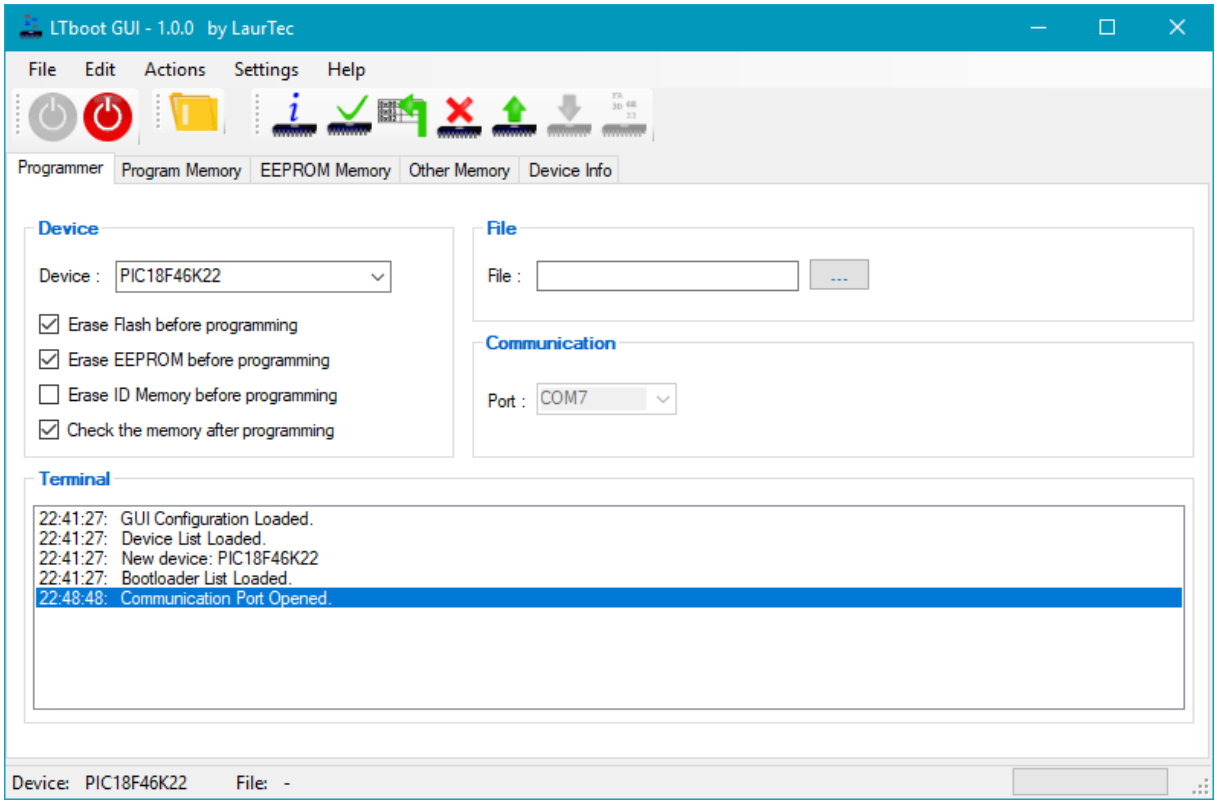


Figure 5: *Advanced settings with the additional memory ranges and Device Info tabs.*

In addition to the memory ranges, also the Device Info tab is enabled. This tab allows special settings and shows also the proprieties of the current active device as shown in Figure 6. Beside the Device Info, the tab shows and allows changing the bootloader settings.

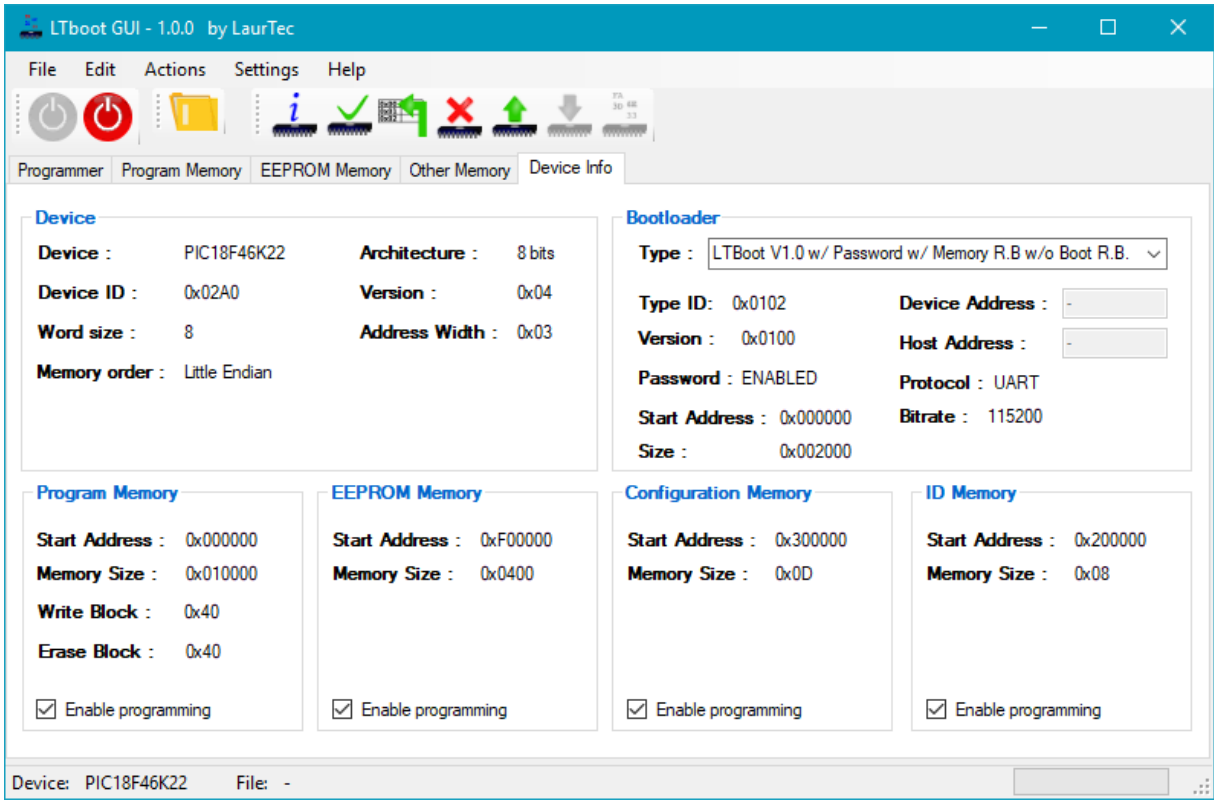


Figure 6: *Device Info tab.*

In case you do not want to write special memory ranges, you can use the Device Info tab to disable the specific memory range. For instance if you do not want to write the EEPROM memory range, you can disable it via the *Enable programming* checkbox within the EEPROM Memory group.

Programming Steps

To program a new software within a microcontroller, by using LTboot, you need to follow these steps:

- Program LTboot Firmware within the MCU, by using a programmer.
- Connect the hardware mounting the microcontroller to the PC.
- Depending on the bootloader, bring the system in the bootloader mode (entry sequence).
- Run LTboot GUI.
- Connect the GUI by pressing the green button.
- Select the device of interest.
- Load the application .hex file you want to download inside the MCU.
- Press the "Write Device" button.
- Wait until the programming is complete.

Before executing the programming command, make sure that the "Erase Flash before programming" check box (Programmer Tab) is selected. If it is not selected, by doing a

memory check after the programming, you may get errors.
Some bootloaders, depending on the compilation modes, may require a password.
The default one is: 00010203.

Depending on the command that must be executed, writing a wrong password may create time out errors or return invalid data.

Writing wrong passwords for too many times (selected within the bootloader code), as safety feature, all the flash content will be erased.

Firmware auto-reload

LTboot GUI allows monitoring the date and time of the .hex file you have selected and reload it automatically if any change occurred. This feature is handy in case you want to load the code any time you do a change and recompile it. During the design phase, this may happen quite often, so a reload can save you time by reloading the new code automatically. This feature can be enabled by selecting the following option:

```
Settings → Enable .hex reload
```

Adding a new bootloader

If you have compiled your own version of LTboot you may need to add it within the bootloader.ini file. After rebooting the LTboot GUI, you can than select it within the Device Info tab, that can be enables via the Advanced Settings. For instance you can change the entry point sequence, the need for password or the memory read back features. Any of this change requires to compile the code, creating a new bootloader. To distinguish the different bootloader options, new ones can be added within the bootloader.ini file. Each Bootloader has the following information that must be added:

```
BOOTLADER
 TYPE: LTBoot V1.0 w/ Password w/ Memory R.B w/o Boot R.B.
 TYPE_ID: 0x0102
 VERSION: 0x0100
 START_ADDRESS: 0x000000
 SIZE: 0x002000
 PROTOCOL: UART
 BITRATE: 115200
 PASSWORD: ENABLED
 DEVICE_ADDRESS: 0x00
 DEVICE_ADDRESS_ENABLE: DISABLED
 HOST_ADDRESS: 0xA0
 HOST_ADDRESS_ENABLE: DISABLED
 MEMORY_READBACK_ENABLE: ENABLED
END
```

To add the bootloader you can just copy and paste the information block above and change the required info. Each bootloader block start with the keyword BOOTLOADER and ends with the keyword END.

Adding a new device

LTboot GUI supports multiple devices and adding a new one can be dynamically done overtime. The available devices shown within the combo list, are loaded any time LTboot GUI is launched. The information is read out from the device.ini file. Each device information is written between the keyword DEVICE and the keyword END. All the information that must be written are shown below.

```
DEVICE
 PN: PIC18F46K22
 ID: 0x02A0
 VERSION: 0x04
 ARCHITECTURE: 8 bits
 ADDRESS_BYTE_WIDTH: 0x03
 WORD_SIZE: 8
 MEMORY_ORDER: Little Endian

 PROGRAM_MEMORY_ENABLE: ENABLED
 PROGRAM_MEMORY_START_ADDRESS: 0x000000
 PROGRAM_MEMORY_SIZE: 0x010000
 PROGRAM_MEMORY_WRITE_BLOCK:0x40
 PROGRAM_MEMORY_ERASE_BLOCK:0x40

 EEPROM_MEMORY_ENABLE: ENABLED
 EEPROM_MEMORY_START_ADDRESS: 0xF00000
 EEPROM_MEMORY_SIZE: 0x0400

 ID_MEMORY_ENABLE: ENABLED
 ID_MEMORY_START_ADDRESS: 0x200000
 ID_MEMORY_SIZE: 0x08

 CONFIGURATION_MEMORY_ENABLE: ENABLED
 CONFIGURATION_MEMORY_START_ADDRESS: 0x300000
 CONFIGURATION_MEMORY_SIZE: 0x0D
END
```

Adding one device is just a matter of adding this block within the device.ini file. Similar information are also contained within the LTboot firmware, thus, also the Firmware that would be compiled for a new device must be changed accordingly. Nevertheless this is a Firmware change and not related to the GUI.

Software Debug

All the activities are shown within the Terminal. The details shown in the Terminal depends on the Debugging level that is activated. By Default the Debugging level is 1 (minimal feedback) but it can be changed up to 3 via the menu:

```
Settings → Debug → Debugging Level
```

By Debugging Level 1, the amount of information that are shown are limited, avoiding jamming the Terminal with information that are typically not needed.

For instance if you run a Device Check command, within the Terminal you can read the following info:

```
15:38:22:   The Bootloader requires a password.
15:38:24:   Unlock Password has been sent out.
15:38:24:   Reset Vector Empty: FAILED
15:38:24:   Device Check: PASSED
```

If you run the same command using the Debug Level 2 you will get the following information:

```
15:40:14:   The Bootloader requires a password.
15:40:15:   Unlock Password has been sent out.
15:40:15:   Reset Vector Empty: FAILED
15:40:15:   Device ID Match: PASSED
15:40:15:   Device Version: PASSED
15:40:15:   Bootloader Type ID: PASSED
15:40:15:   Bootloader Version : 0x0100
15:40:15:   Device Check: PASSED
```

While if you run the command with the Debug Level 3 you get the following Info.

```
15:38:35:   The Bootloader requires a password.
15:38:37:   Unlock Password has been sent out.
15:38:37:   Checking Reset Vector...
15:38:37:   Reset Vector Empty: FAILED
15:38:37:   Send: Command_GetDeviceID
15:38:37:   Device ID : 0x02A0
15:38:37:   Device ID Match: PASSED
15:38:37:   Send: Command_GetDeviceVersion
15:38:37:   Device Version : 0x04
15:38:37:   Device Version: PASSED
15:38:37:   Send: Command_GetBootloaderTypeID
15:38:37:   Bootloader Type ID: 0x0102
15:38:37:   Bootloader Type ID: PASSED
15:38:37:   Send: Command_GetBootloaderVersion
15:38:37:   Bootloader Version : 0x0100
15:38:37:   Device Check: PASSED
```

During the programming phase, either by reading or writing the memory, using the Debugging Level 3, is possible to see the information related to each specific memory block.

The following FAIL means that the device program memory is not empty so it contains an application program beside the bootloader Firmware.

```
15:38:24:   Reset Vector Empty: FAILED
```

Code Editing

Once the .hex file has been opened, it is possible to modify it and eventually export it again or flash the memory with the new code or data.

Writing directly in the memory is handy in case you need to write data within the EEPROM or ID memory space.

To modify a specific address you need first select the memory tab related to the one you want to modify. Afterward you need to double click on the blue address on the left side. By doing that the memory editor window gets opened, as shown in Figure 7.

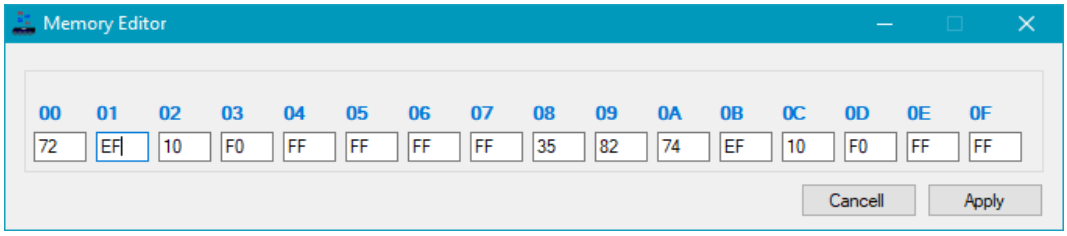


Figure 7: *Memory Editor.*

Within the memory address are shown the 16 bytes that belong to the blue memory address block that has been clicked out.

At this point is possible to change the address of interest and press Apply.

Index

Bibliografy

[1] www.LaurTec.it: official site where you can download LTboot and the documentation.

History

Date	Version	Author	Revision	Description
23.09.18	1.0	Mauro Laurenti	Mauro Laurenti	Original version.