WriteNow! and µISP

In-System Programmers

Programmer's Manual

Rev. 2.00 - February 2022

algocraft

UM00070000EN

Copyright Information

Disclaimer

The material contained in this document is provided "as is", and is subject to being change, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Algocraft disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Algocraft shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Algocraft and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Whilst every effort has been made to ensure that programming algorithms are correct at the time of their release, it is always possible that programming problems may be encountered, especially when new devices and their associated algorithms are initially released. It is Algocraft's policy to endeavor to rectify any programming issues as quickly as possible after a validated fault report is received.

It is recommended that high-volume users always validate that a sample of a devices has been programmed correctly, before programming a large batch. Algocraft can not be held responsible for any third party claims which arise out of the use of this programmer including 'consequential loss' and 'loss of profits'.

Algocraft Warranty Information

Algocraft warrants that this product will be free from defects in materials and workmanship for a period of one (1) year from the date of shipment. If any such product proves defective during this warranty period, Algocraft, at its option, either will repair the defective product without charge for parts and labor, or will provide a replacement in exchange for the defective product. Parts, modules and replacement products used by Algocraft for warranty work may be new or reconditioned to like new performance. All replaced parts, modules and products become the property of Algocraft. In order to obtain service under this warranty, Customer must notify Algocraft of the defect before the expiration of the warranty period and make suitable arrangements for the performance of service. Customer shall be responsible for packaging and shipping the defective product to the service center designated by Algocraft, with shipping charges prepaid. Algocraft shall pay for the return of the product to Customer if the shipment is to a location within the country in which the Algocraft service center is located. Customer shall be responsible for paying all shipping charges, duties, taxes, and any other charges for products returned to any other locations. This warranty shall not apply to any defect, failure or damage caused by improper use or improper or inadequate maintenance and care. Algocraft shall not be obligated to furnish service under this warranty a) to repair damage resulting from attempts by personnel other than Algocraft representatives to install, repair or service the product; b) to repair damage resulting from improper use or connection to incompatible equipment; c) to repair any damage or malfunction caused by the use of non-Algocraft supplies; or d) to service a product that has been modified or integrated with other products when the effect of such modification or integration increases the time or difficulty of servicing the product.

THIS WARRANTY IS GIVEN BY ALGOCRAFT WITH RESPECT TO THE PRODUCT IN LIEU OF ANY OTHER WARRANTIES, EXPRESS OR IMPLIED. ALGOCRAFT AND ITS VENDORS DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ALGOCRAFT'RESPONSIBILITY TO REPAIR OR REPLACE DEFECTIVE PRODUCTS IS THE SOLE AND EXLUSIVE REMEDY PROVIDED TO THE CUSTOMER FOR BREACH OF THIS WARRANTY. ALGOCRAFT AND ITS VENDORS WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER ALGOCRAFT OR THE VENDOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Technical Support

Please e-mail any technical support questions about this product to: support@algocraft.com.

Table of Contents

1. Introduction 7
Important Notice to Users 7
Technical Support 7
Required Skills 7
µISP Series 8
Programming algorithms 8
Programming Drivers and Licenses 8
Filename Restrictions 8
2. Getting Started 9
Install Software 9
Recommended Software System Requirements 9
Launch the Project Generator 10
Connect the programmer to the PC 10
Create a Project 10
Connect to Target Device 11
Program the Target Device 11
Where to Go from Here 12
3. WriteNow! Project Generator GUI 13
Overview 13
Main window 14
Top Toolbar 14
Left Toolbar 15
Right Control Box Panel 16
Create a New Project 17
Project Creation Wizard (Step 1 of 3) 17
Project Creation Wizard (Step 2 of 3) 18
Project Creation Wizard (Step 3 of 3) 20
Advanced Project Settings 21

Manual Project Editing 22
Model Selection 22
Communication settings 23
USB 23
COM (RS232) 23
LAN / Ethernet 24
Hardware Settings 26
Firmware Upgrade 26
Hardware Test 26
4. Image file Creation 27
Overview 27
Adding a Source File (File Formats) 28
Intel-Hex formats 28
Motorola S-Rec formats 29
Binary formats 30
JEDEC format 30
POF Format 31
HEX Dump format 31
Infineon Tricore Hex/SRec format 31
TI-TXT format 31
Espressif Binary format 32
Atmel XCFG format (ATMXT641T device) 32
NXP PMIC CFG format (PF0100A device) 33
Elmos E522 CFG format (E522.49 device) 33
XDPL AHEX format (XDPL821x devices) 33
Micronas HAL format (HAL37xx devices) 34
Editing a Source File 34
Const Data Pattern 34
Exclude Range 34
Variable Data 35
Conversion Report and Analysis 35
Fill Unused Location 36
Command Line conversion utility 36
5. Commands 37

Overview 37
Command Syntax 37
OK Answer 37
ERR Answer 37
BUSY Answer 38
WriteNow! Terminal 38
Command Reference 38
Data In/Out Commands 39
Execution Command 40
File System Commands 41
Programming Commands 43
Status Commands 44
System Commands 46
Time Commands 48
Volatile Memory Commands 49
6. Programming Variable Data and Serial Number 51
Define the device address range 51
Generate and set the variable memory 51
Start the programming process 52
Example 52
7. WriteNow! File System 55
Overview 55
File System Structure 56
8. Standalone Mode 57
Overview 57
Project Assignment 57
9. Protected Mode and Data Encryption 59
Protected Mode 59
Data Encryption 61
10. WriteNow! API 62
Overview 62
Including the API in Your Application 62
Function Reference (C++ Library) 63
WN_CloseCommPort() 64

WN_ExeCommand() 65
WN_GetFrame() 66
WN_GetLastErrorMessage() 67
WN_ReceiveFile() 68
WN_SendFile() 69
WN_SendFrame() 70
WN_OpenCommPort() 71
Function Reference (C++/CLI Library) 71
Function Reference (C# Library) 75
11. Troubleshooting 77
USB Driver issues (Windows 7 and earlier) 77
Get full access to WriteNow! folder on Windows 10 77
Diagnostic Test 79

1. Introduction

Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, Algocraft 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.

Technical Support

Algocraft is continuously working to improve WriteNow! firmware and to release programming algorithms for new devices. The latest version of WriteNow! system software is always available from our website: http://www.algocraft.com.

To get in touch with Algocraft, please email to support@algocraft.com

Trademarks

Names mentioned in this manual may be trademarks of their respective companies. Brand and product names are trademarks or registered trademarks of their respective holders.

Required Skills

In order to beneficially use the WriteNow! programmers, you should be familiar with certain skills, ranging from hardware design to software design. In particular, it is assumed a good knowledge of the following:

- Microcontroller systems.
- Programming knowledge (C, C++).
- The target architecture knowledge.
- The software tools used for building your application.

µISP Series

The µISP programmer is based on WriteNow! technology. µISP series and WriteNow! series are fully software compatible.

Programming algorithms

WriteNow! supports different devices (microcontrollers, serial and parallel memories, CPLD/FPGA, programmable sensors and ICs, etc). In order to program a specific device, the following software components are needed:

- The programming algorithm driver (.**wnd**). The programming driver is a library running on WriteNow! hardware platform, that contains routines needed to program a specific device family.
- The image file (.**wni**). The image file is a binary WriteNow! proprietary file format used to store customer firmwares.
- The project file (.**wnp**). The project file is a text file that contains device specific settings and programming commands.
- The license file (.**wnl**). The license file is a binary file linked to the serial number of the hardware unit, that contains the list of all the devices the programmer can use.

Programming Drivers and Licenses

WriteNow! comes with preinstalled programming drivers (algorithms) that support common microcontrollers and memories. When you purchase a new programming driver, you are supplied with a new driver file (.wnd) and an updated license file (.wnl). The license file enables the use of all of your purchased drivers on your specific WriteNow! unit.

You must copy these files to the unit's internal memory: the driver file must be copied to the unit's **\drivers** folder, and the license file to the unit's **\sys** folder.

Filename Restrictions

Characters not allowed:

* / \ | ? * . , : ; < >

and "space"

Filename (*.wni, and *.wnp) is limited to 39 characters, including extension.

2. Getting Started

The following tutorial will guide you through the steps required to set up your WriteNow! programmer and create your first programming project.

Note: it is highly recommended to install all the required software first, so that the WriteNow! USB driver will be automatically found by Windows once the programmer is connected (earlier versions than Windows 10 only).

Install Software

Install the WriteNow! software. Follow the on-screen instructions. The latest version of WriteNow! system software is always available from our website: http://www.algocraft.com.

Note: to install the WriteNow! software, you must log in as Administrator.

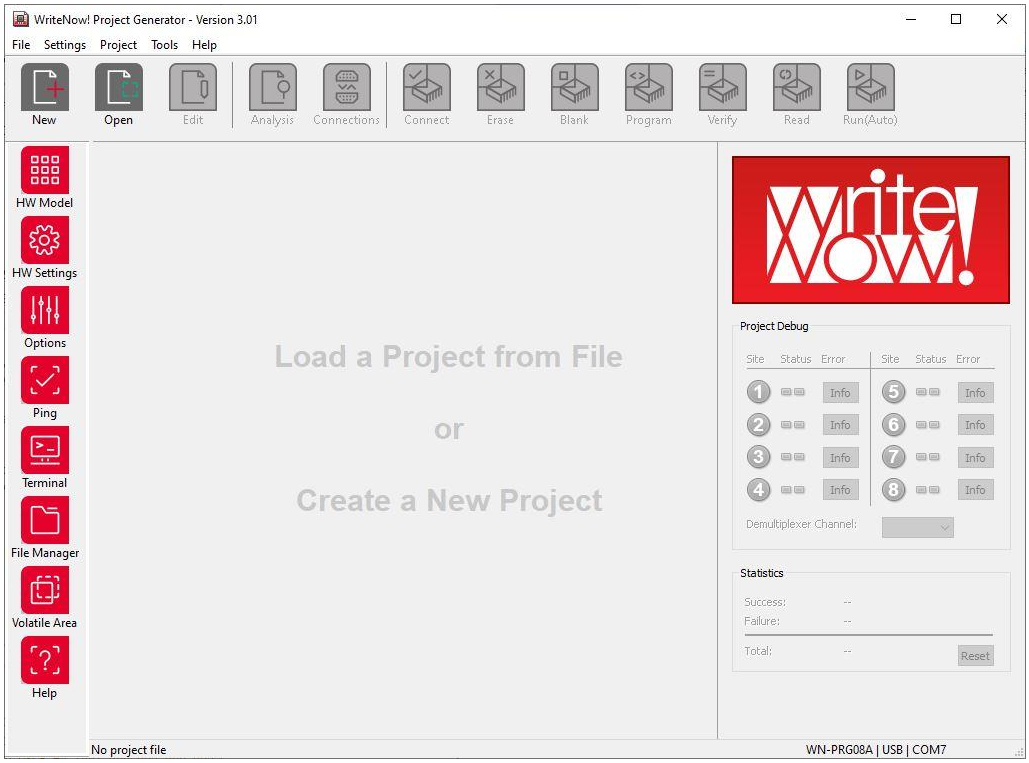
Recommended Software System Requirements

Microsoft Windows 7, Windows 8, Windows 10 (both 32-bit and 64-bit editions)

Microsoft .NET Framework 4.0 or later.

Launch the Project Generator

Launch the Project Generator application, that is located under **Programs > Algocraft > WriteNow! Software > Project Generator.**



Connect the programmer to the PC

Connect the programmer to the host PC through the available ports: USB, LAN or RS232. The "Model and Communication Settings" section describes the procedure to configure your instrument and the USB, LAN and COM (RS232) communication setup.

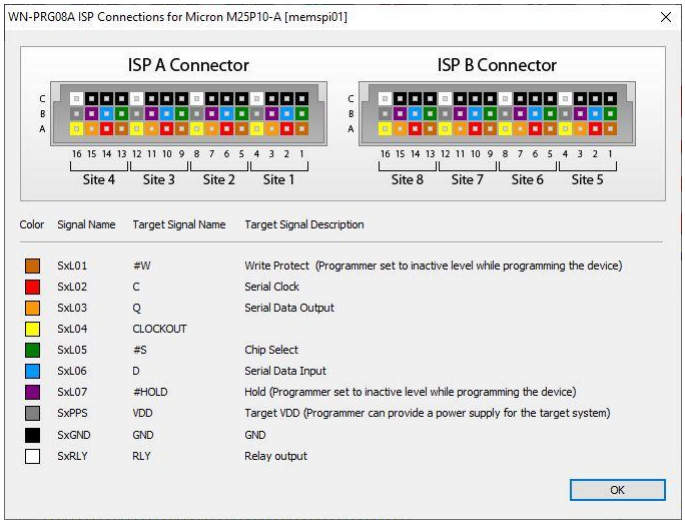
Once completed the communication setup procedure, press the  icon to verify the PC can establish a connection with the programmer.

Create a Project

A project must be created prior to starting a programming session. The "Project Setup" section describes the main steps to follow during the creation of the project.

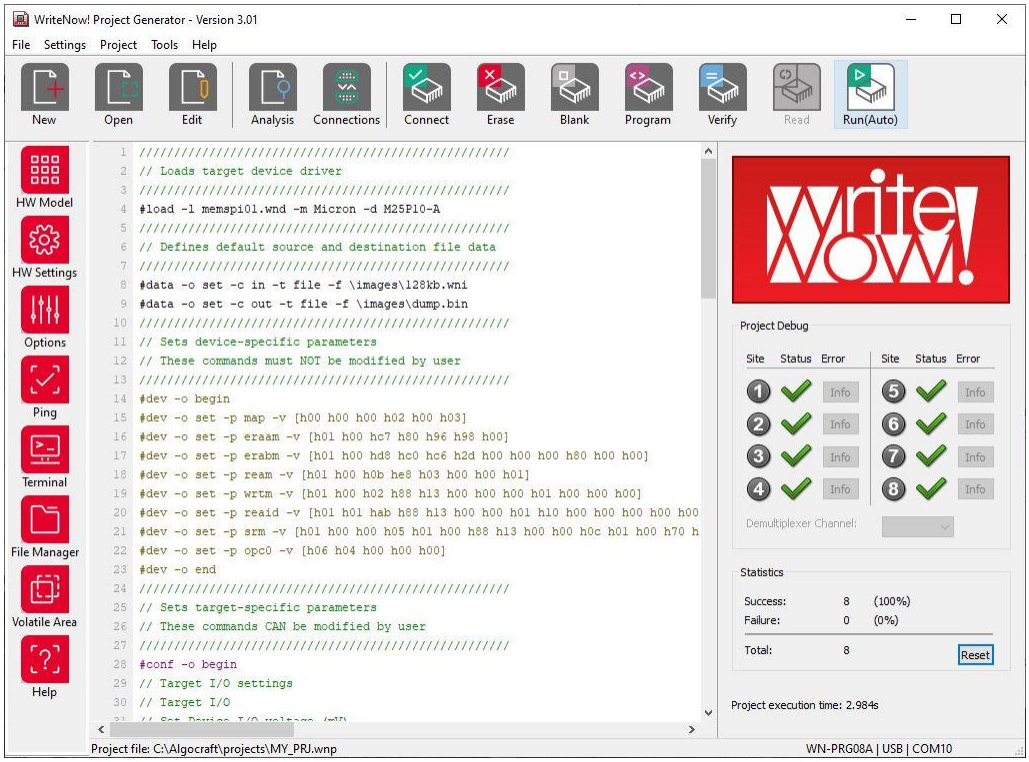
Connect to Target Device

Connect WriteNow! to your target system through the ISP connector(s). To view the connections for your selected target device, select **Project > Show ISP Connections**.



Program the Target Device

Select **Project > Run** Project. The Project file (.wnp) and Image file (.wni) will be automatically uploaded to WriteNow! and the project will be executed. Your target device(s) will be programmed.



In case of programming errors, or to change programming parameters/operations, you can relaunch the Project Wizard and review the project settings.

Where to Go from Here

In this chapter, you have learnt how to use the Project Generator to create and execute a typical programming project. Additionally, WriteNow! can be controlled in three other ways:

1. By manually sending commands and receiving answers, using the Project Generator Terminal or any other terminal application (for more information, see Commands);
2. By configuring the instrument so that it can work in standalone, that is without a connection to a PC (for more information, see Standalone Mode);
3. By building your own PC software that interfaces to the instrument (for more information, WriteNow! API).

3. WriteNow! Project Generator GUI

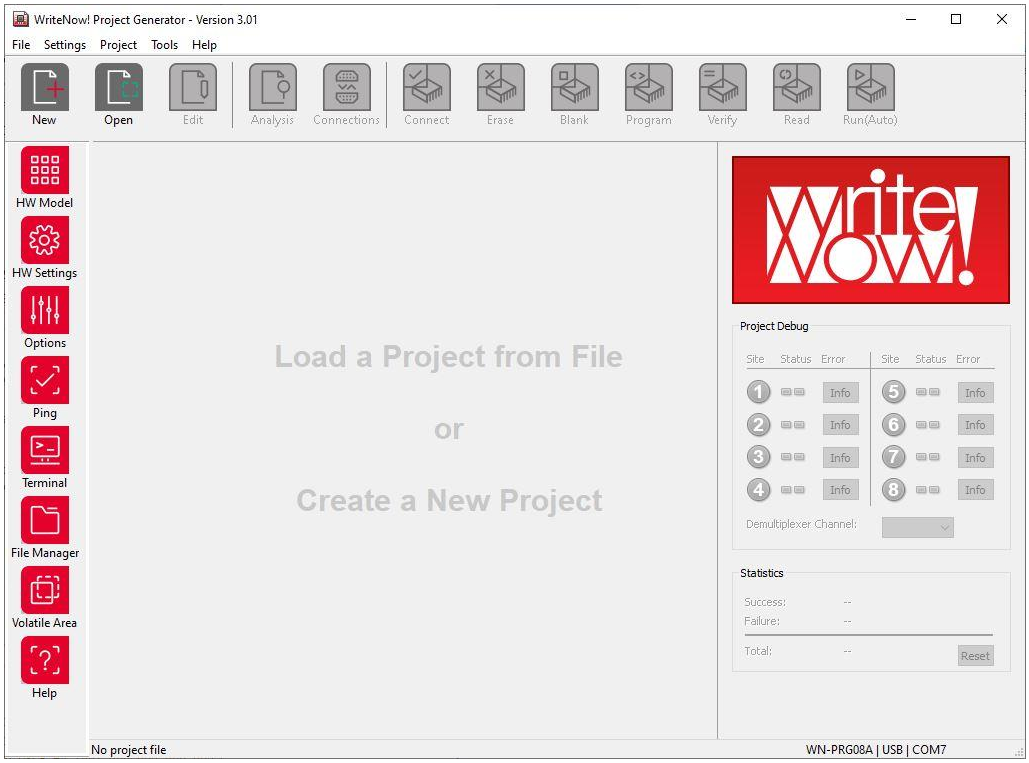
Overview

The WriteNow! Project Generator is a programming software for PC which is able to communicate with the programmers based on WriteNow! technology. The Project Generator guides you through the creation and debugging of a programming Project in few guided steps:

- Instrument settings;
- Device selection;
- Source file creation;
- Board parameter settings;
- Programming flow options;
- Upload and run of the Project;
- Memory view;
- Blank check/erase/program/verify/read operations;

Main window

Launch the Project Generator application, that is located under **Programs > Algocraft > WriteNow! Software > Project Generator.** The following dialog box will appear.



Top Toolbar

The top toolbar shows the most frequently used commands to create and debug a project – it is a subset of the menu bar commands.

Icon	Functions
	Create a new project
	Open an existing project
	Edit a project (wizard)
	View the image file segmentations and memory map of the device

	View the ISP connections
	Check the serial communication between device and programmer
	Erase the device memory
	Check the blank state of the device
	Program the device memory
	Verify the device memory
	Read the device memory
	Execute a complete programming cycle

Left Toolbar

The left toolbar shows the most frequently used commands to configure and set the programmer unit – it is a subset of the menu bar commands.

Icons	Features
	Select the hardware model
	Open the hardware settings window
	Edit miscellaneous settings

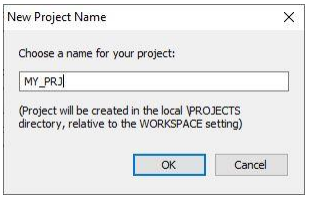
	Check the communication between PC and programmer (ping)
	Open the terminal window
	Open the File Manager window
	View the volatile memory
	Open the hardware manual

Right Control Box Panel

The right Control Box panel displays the site status errors and statistics.

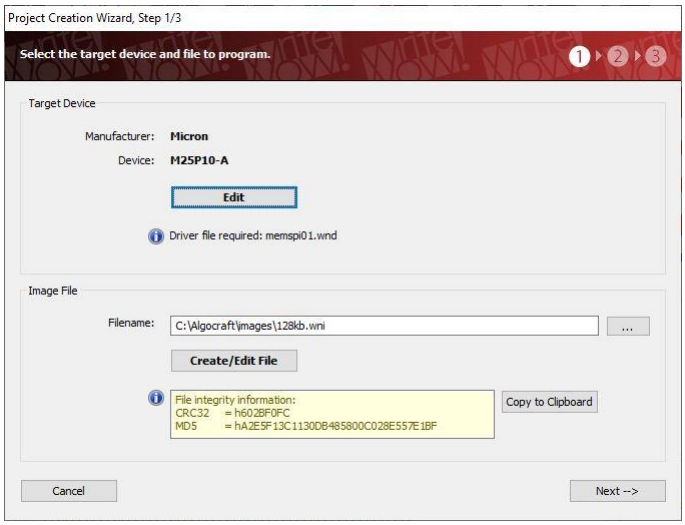
Create a New Project

Select **File > New Project**, give a name to your programming project, and then follow the Project Creation Wizard steps.



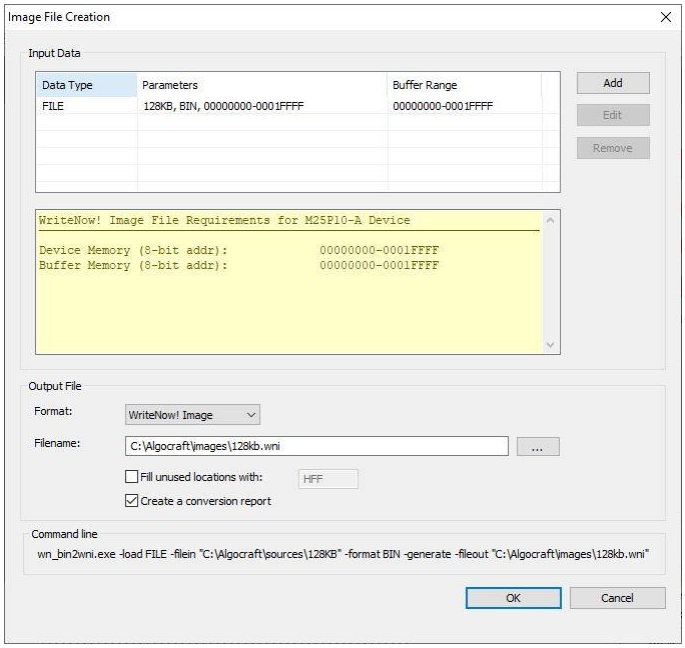
Project Creation Wizard (Step 1 of 3)

First Wizard step: specify the target device, by clicking the **"Edit"** button.



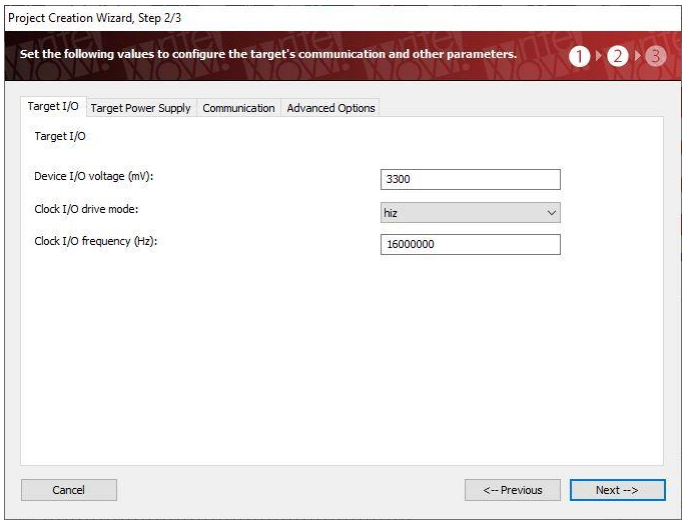
Then, specify the image file to be programmed. To create an image file, click the **"Create/Edit File"** button. A dedicated window will open.

In the Output File section, choose the output filename by clicking the **"..."** button. Use the **"Add"** button to compose the data that will be included in the image file. Use the **"..."** button to specify the name of the Image file. When done, click **"OK"** to return to the Wizard, and proceed to Step 2.



Project Creation Wizard (Step 2 of 3)

In this step, specify target parameters and connection values. The Wizard will automatically fill all data with typical values for the selected target device.

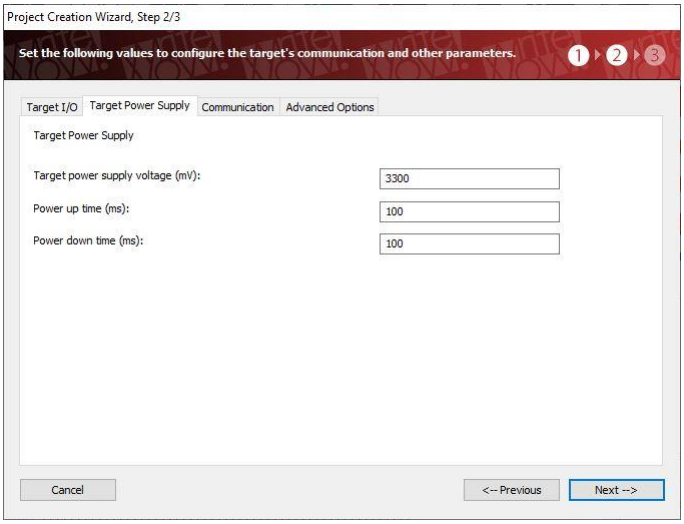


The number of tabs displayed in this window depends on the selected target device; however, three tabs (**"Target I/O", "Target Power Supply"** and **"Communication"**) are always present and will be briefly discussed below.

The first tab is **"Target I/O".** The **"Device I/O voltage"** setting specifies the voltage of the ISP lines. You should check the target board schematics, or ask the board developer about this value. The allowed voltage also depends on the selected target device.

The **"Clock I/O drive mode"** setting allows you to decide how the SxL04 ISP line is driven (the x index refers to the programming site). This line can be used as an auxiliary ISP line (to provide a clock to the target device), as a generic I/O line, or as a high-impedance output (no electrical driving). When used as output line (set to high or low), it could be used, for example, to disable the external watchdog circuit in the target

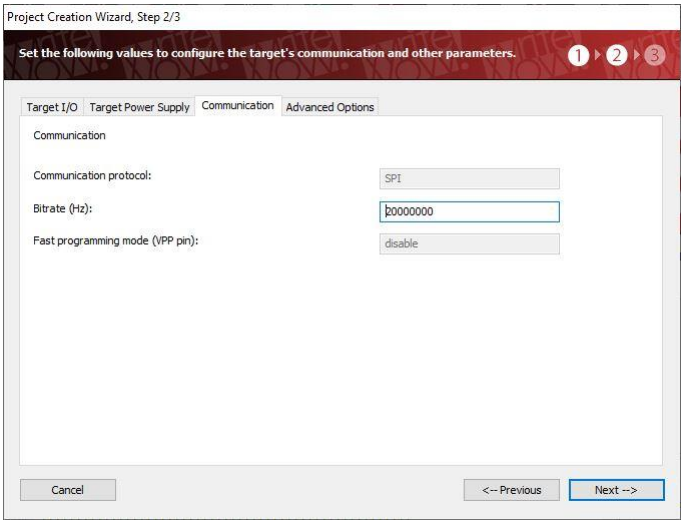
board. When used as clock out, you can specify the output frequency in the **"Clock I/O frequency"** field. We suggest leaving this line floating (HiZ) when not used, in order to decrease electrical noise on other ISP lines.



If you decide to power the target board through the WriteNow! power supply line (SxPPS), specify in the **"Target Power Supply"** tab the electrical and timing parameters of the target power supply line. WriteNow! is able to power the target board through a dedicated programmable power supply output line per site. The voltage of the programmable power supply line (**"Target power supply voltage"** setting) can be in the range 1700mV to 13000mV. Each programmable power supply line features an internal voltage limiter that cuts the voltage output in case of short circuits or overloads.

The **"Power up time"** setting specifies the delay between the programmable power supply line turning on and the first operation on the ISP lines. The purpose of this parameter is to wait for the power supply to become stable, before starting ISP programming. This parameter is useful when large capacitors are mounted in the target board's power line.

The **"Power down time"** setting acts in similar way: it sets the delay between the programmable power supply line turning off and subsequent operations.

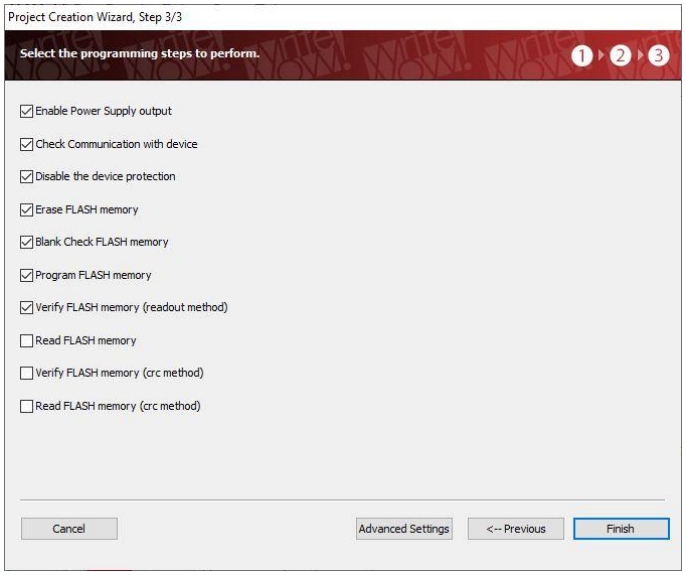


The content of the **"Communication"** tab depends on the selected target device. It allows you to select the communication protocol that will be used for programming (some target devices may provide more than one communication protocol) and its related settings, such as the communication speed and other parameters. Usually, the higher the communication speed, the shorter/better the ISP cabling must be.

After carefully checking all of the parameter values, proceed to Step 3.

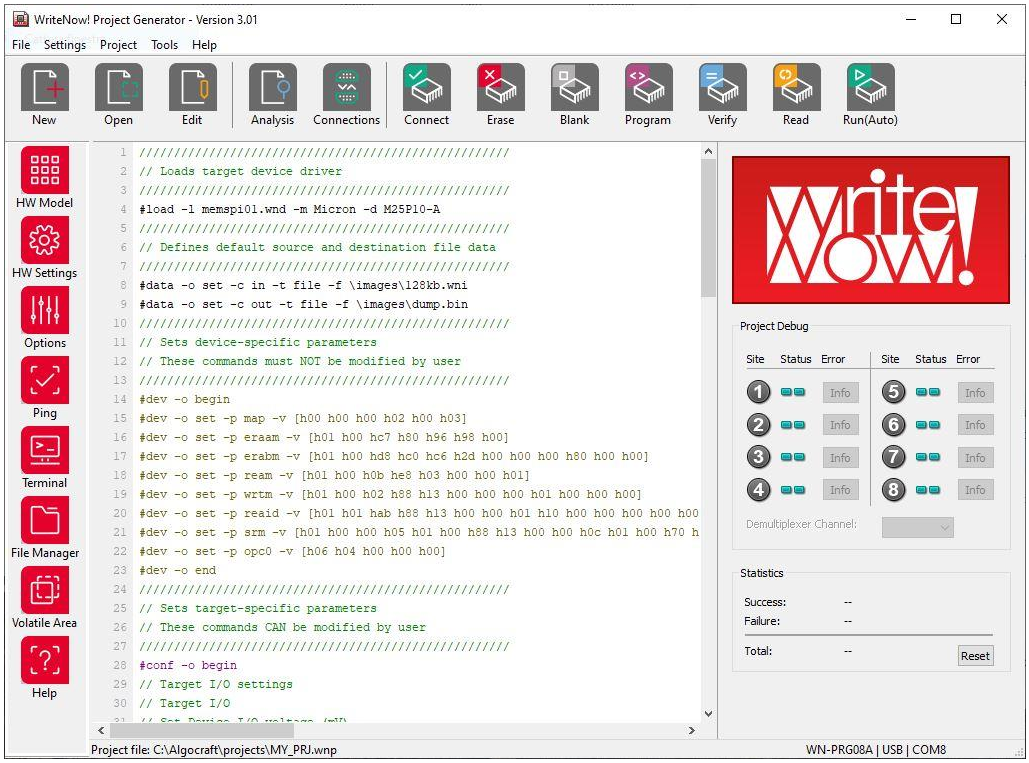
Project Creation Wizard (Step 3 of 3)

Select the programming operation(s) to be performed on the target.



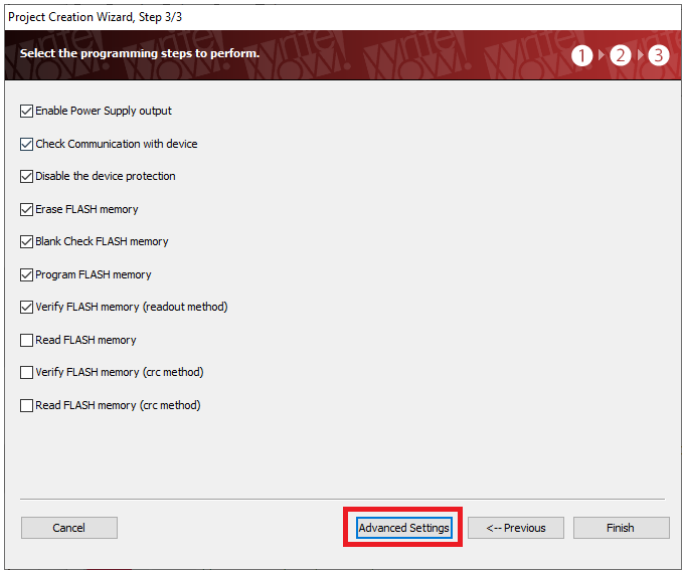
The read FLASH memory will create a dump file, located in the **\images** folder. It can be performed one site at once.

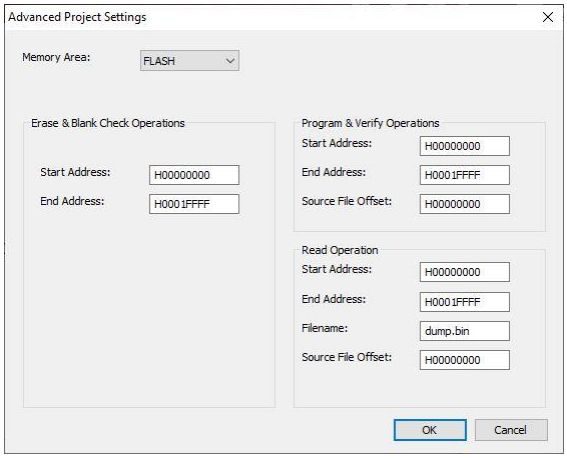
Click **"Finish"** to end the Wizard. At this point, a WriteNow! Programming Project will be created in the **\Projects** directory, relative to the Project Generator workspace location.



Advanced Project Settings

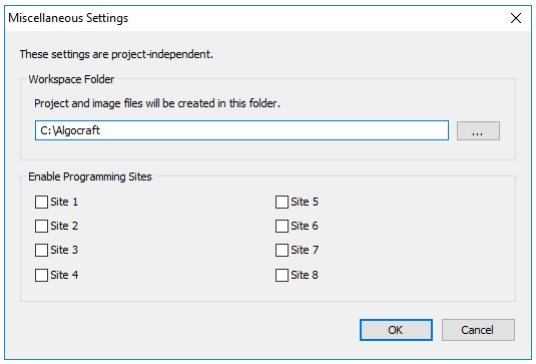
In most of our programming algorithm, it is possible to change some of the default project settings without editing the project manually.





Manual Project Editing

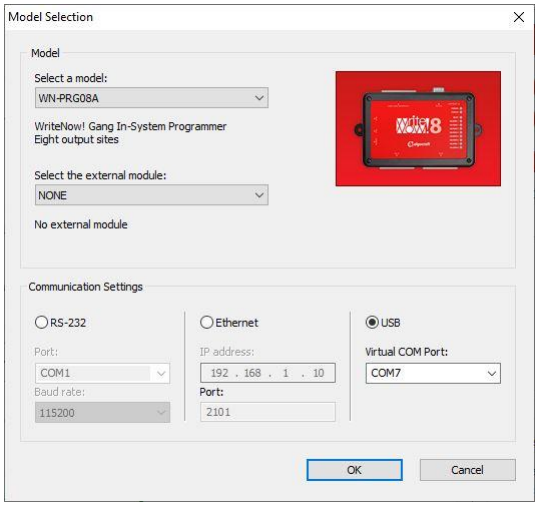
The Project file created by the Project Wizard is located, by default, in the **\Projects** folder, relative to the Project Generator application location (it can be changed any time by specifying a different "workspace" path: to do so, in the Project Generator, select **Settings > Edit Miscellaneous Settings** and modify the **Workspace** directory).



The generated project file is an ASCII text file and, if necessary, can be edited using any text editor. Please note, however, that once the file is modified by the user, it can be opened by the Project Generator but the Project Wizard will not be available.

Model Selection

Go to Settings > Select Hardware Model, and specify your instrument model and communication settings with the PC. In addition, select the external module if present.



Communication settings

This section describes the simple procedure for the USB, COM and LAN setup.

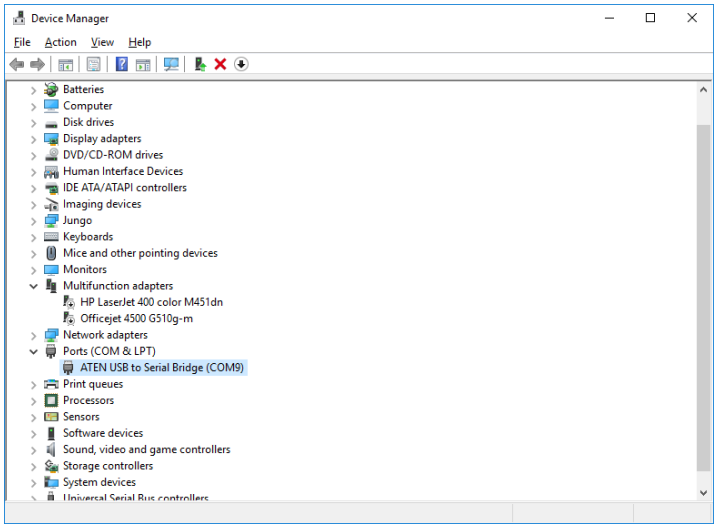
USB

The USB driver is designed as a virtual COM. In this way, the instrument will be detected automatically on Windows 10 OS (no additional driver is required). For earlier Windows versions, the setup software will install the driver, or, if needed, it can be installed manually through the WNUSB.inf located under the "...\developer\USB driver" directory.

Once connected the USB cable, please check on Device Manager → Ports (COM & LPT) if the USB Serial Port (COMX) is present. Where X is an integer number.

COM (RS232)

WriteNow! communicates at **115,200** bps by default.
On Windows system, check the proper COM port number in the **Device Manager** window:



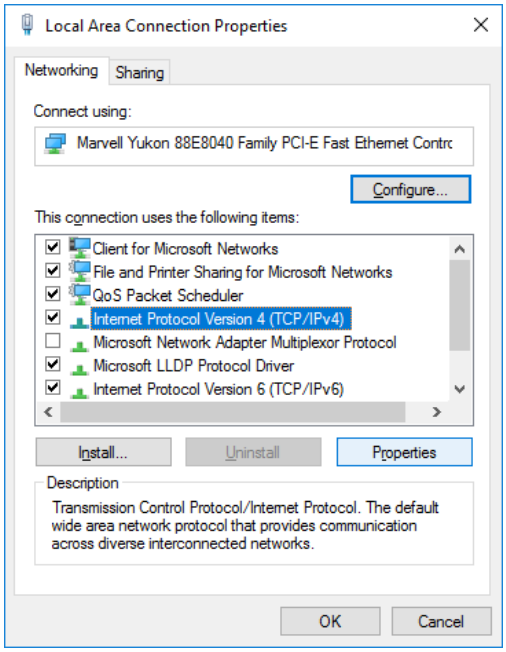
LAN / Ethernet

The WriteNow! programmer default LAN parameters are:

- IP address: **192.168.1.10**
- Netmask: **255.255.255.0**
- Gateway: **192.168.1.1**

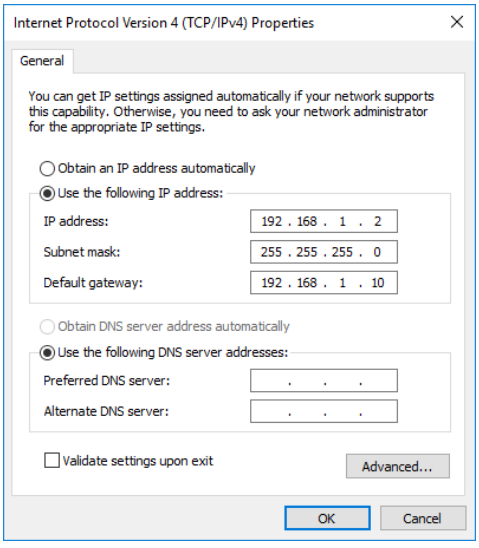
Connect WriteNow! to your LAN through the ethernet cable. If you are using a router, make sure that the configuration settings match.
WriteNow! could also be connected directly to your PC. In this case or if you want to configure your LAN manually, set a static IP:

- On Windows system, go to the network card properties window and choose the "Internet Protocol Version 4" item:

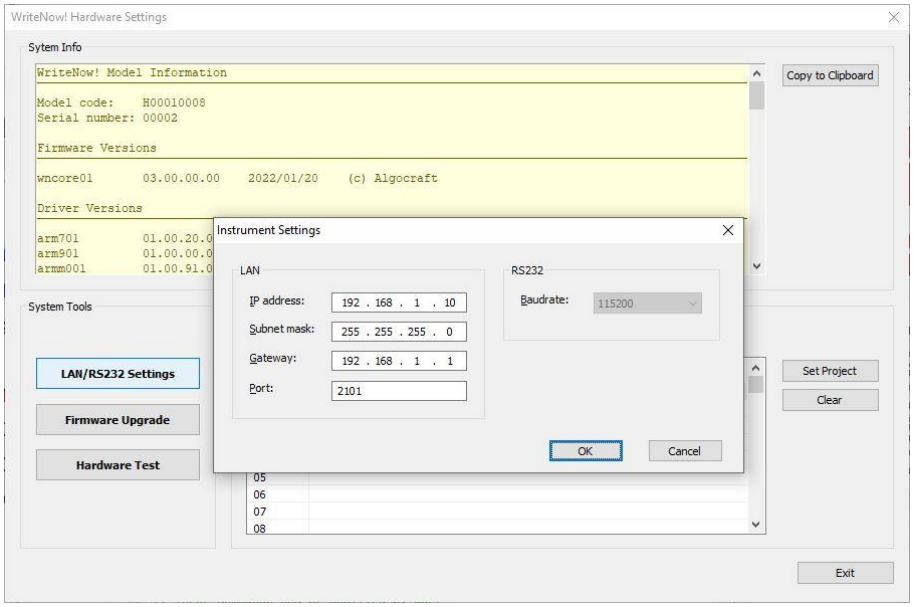


- Set the proper IP, subnet mask and the default gateway:

Once connected to the instrument, change the current LAN parameters if needed. To



do this, open the "WriteNow! Hardware Settings" and then select "LAN/RS232 Settings". The new configuration will be active after the programmer reboots.

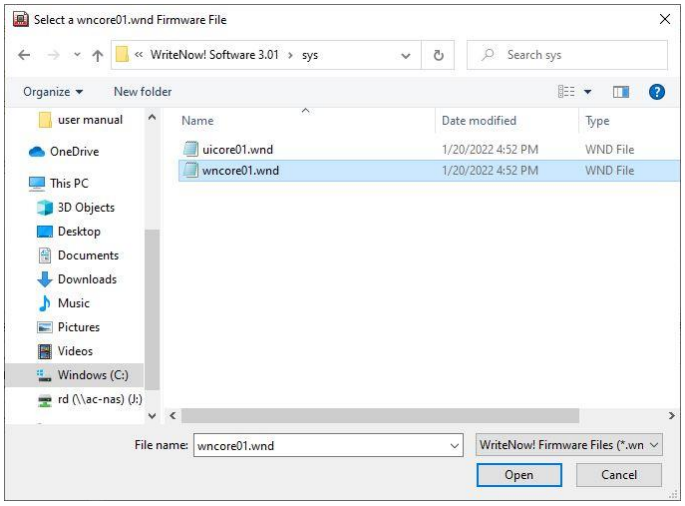


Hardware Settings

Firmware Upgrade

To update the firmware, click on the **Firmware Upgrade** button. Then browse the proper programmer firmware version.

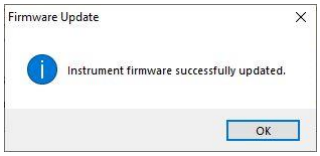
It is located in **Programs > Algocraft > WriteNow! Software > sys.**



For WriteNow! programmer the firmware filename is wncore01.wnd

For µISP programmer the firmware filename is uicore01.wnd.

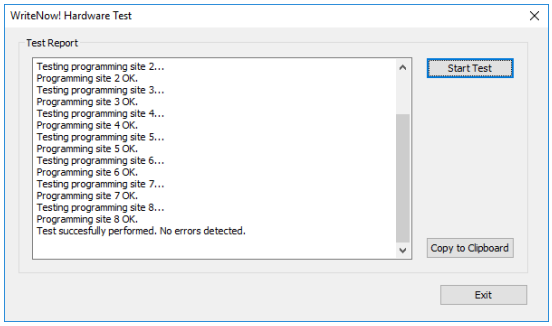
Wait until the firmware has been updated successfully:



At this time the new firmware is already working. It's not necessary to reboot the system.

Hardware Test

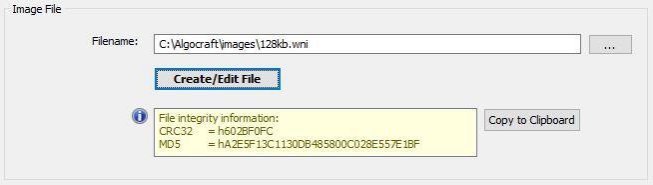
The diagnostic procedure is a very easy way to verify if some hardware faults occur.
Use the provided test-board to check the connections to all the ISP sites.



4. Image file Creation

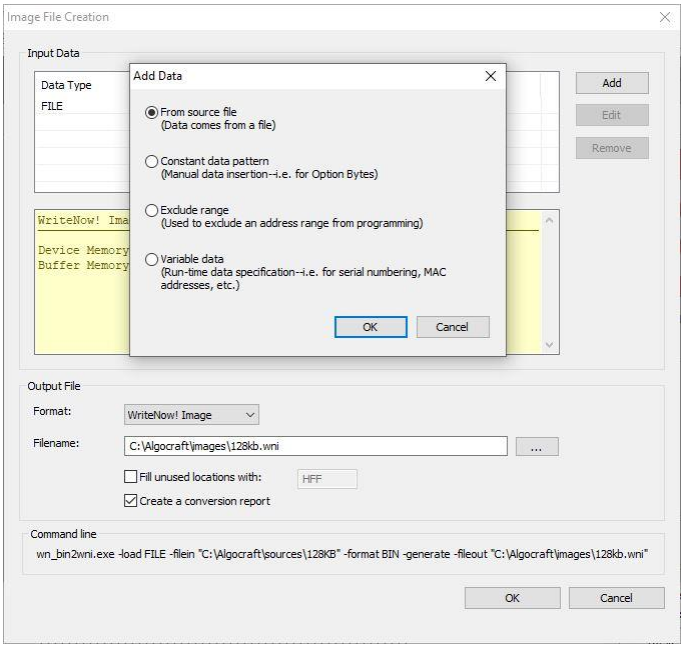
Overview

This chapter describes all the features which could be used to create your image file.
The standard output format is the WriteNow! image (.wni), optimized for reading and writing data.
When the image file is created, the CRC32 and the MD5 values are shown in order to check the file integrity information:



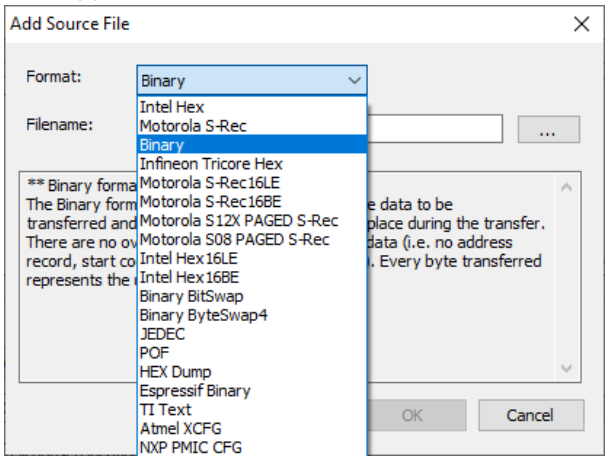
The CRC32 and MD5 values are not calculated on the entire file, but it skips the first **512 bytes.**

In the following sections will be explained all the input data types.



Adding a Source File (File Formats)

The supported file formats are:



Intel-Hex formats

The Intel HEX file is an ASCII text file. A sample HEX is shown below:

```
:020000040030CA
:20000000FE067C45974496432B4602001E425CAD9243DC825CA47640CF1996465200ED1D82
:20001000762256CF0014884502OB07428AA97EC456CF0013884502OC07428AA97EC456CF71
:20002000000C56BF024656CF00089244ECED5201ECF256CFFFF8761A9246FF69FE860006D4
:20003000FE06974496432B4502001E425CAD9243DC825CA47640CF1996455200ED06762294
:200040008A42924496EC761A9245FF69FE860006FE067C45974496432B4602001E425CADCE
:200050009243DC825CA47640CF1996465200ED1F56CF0018762288450209O7428AA97EC41A
:2000600076lA56CF0015762288450209A07428AA97EC4761A56CF000C56BF014656CF00083D
:200070009244ECE95202ECF056CFFFF89246FF69FE860006FE06974496432B4502001E4225
:20008000SCAD9243DC825CA47640CF1996455200ED117622021907428AA992449001FF8244
:2000900088A9CDC4FFF799A602190742BAA97EC4761A9245FF69FE860006FE06974496436F
:2000A0002B4502001E425CAD9243DC825CA47640CF1996455200ED107622021A07428AA93A
:2000B0009244900388A9CDC4FFFC99A6021A07428AA97EC4761A9245FF69FE860006FE0892
:2000C0007D467C45974496432B4702001E425CAD9243DC825CA47640CF1996477622020FB4
:2000D00007428AA99245900388A9CDC4FFFC99A6020F07428AA97EC456CF001C020F0742C4
:2000E0008AA992449003FF8188A9CDC4FFF399A6020F07428AA97EC4884602110742BAA9C5
:2000F0007EC4884602100742BAA97EC456CF000A9245EC075201ECE35202ECE15203ECDFB4
:20010000761A9247FF69FE880006FE069645A8422B4602001E445CAD9245DC845CA47640EE
:20011000CF19964676228A44C54292EC96C7020907448AA9834292C496D5020A07448AA92A
:20012000834292C496DD020207448AA9020407421EA7F6012684770077008A44834292C423
```

The format for this file is:

First character (:): Start of a record

Next two characters: Record length (in this example, 10h)

Next four characters: Load address (in this example, 0080h)

Next two characters: Record type (see below)

Remaining characters: Actual data

Last two characters: Checksum (i.e., sum of all bytes + checksum = 00)

HEX record types are shown below:

00 = Data record

01 = End of file record

02 = Extended segment address record

03 = Start segment address record

04 = Extended linear address record

05 = Start linear address record

The Intel Hex16LE and Intel Hex16BE formats are very similar to the Intel-Hex format, except that the addresses are word addresses. The count field is a word count.

Intel Hex16LE (little-endian byte orders) and Intel Hex16BE (big-endian byte orders) are Intel Hex file format with 16 bits data word for TMS320F devices.

Motorola S-Rec formats

The Motorola S-record is a file format that conveys binary information in ASCII hex text form. This file format may also be known as SRECORD, SREC, S19, S28, S37. A sample Motorola S-Rec is shown below:

```
S0110000000048657876696577205631 2E3009
S315800000002000008060005 9B3F0FF0180FFFF01806F
S315800000108 5BBC1767A443E897EEE50EE8111AF1162
S315800000200D0040034DC0E1FF6E035E1C3C147B1047
S3158000003000F71B0F20FD60FA0D0080041D00830C65
S315800000403C0A00A0913000FFD9FF20164CF0B71F64
S3158000005081F0680F3C00820200908 21F910000F7B9
S3158000006 0E9FFCDB200900100FFFFFFFFFFFFFFFF1A
S3158000007 0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0A
S3158000008 0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA
```

S-records are character strings of five fields: record type, record length, memory address, data and checksum.

The record type as follows:

S0 = Header record

S1 = Data record for 16-bit addresses

S2 = Data record for 24-bit addresses

S3 = Data record for 32-bit addresses

S5 = Count record

S7 = Terminator record for 32-bit addresses

S8 = Terminator record for 24-bit addresses

S9 = Terminator record for 26-bit addresses

The Motorola S-Rec16LE and Motorola S-Rec16BE format formats are very similar to the standard Motorola S-record format, except that the addresses are word addresses. The count field is a word count.

Note: Motorola SREC16LE (little-endian byte orders) and Motorola SREC16BE (big-endian byte orders) are Motorola S-record file format with 16 bits data word for TMS320F devices.

Note: Motorola S12X PAGED S-Rec format must be selected for NXP HCS12(X) MCU family only. It converts a Motorola S-Rec file remapping the address UNPAGED map (0x4000 - 0xFFFF) to PAGED map (0xFD8000 - 0xFF8000).

Note: Motorola S08 PAGED S-Rec format must be selected for NXP S08 MCU with more than 64KB. It converts an Motorola S-Rec file remapping the extended address memory map: ex: 0x4000-0x7FFF to PAGE1 0x8000-0xBFFF.

Binary formats

The Binary format is a literal representation of the data to be transferred and no translation of the data takes place during the transfer. There are no overhead characters added to the data (i.e. no address record, start code, end code, nulls, or checksum). Every byte transferred represents the user's data.

The Binary BitSwap format is very similar to the Binary format, except that the bits are swapped. This format can be also used to load a .rpf file. The .rpd (Raw Programming Data) file is a binary file containing configuration data for external serial memory like EPCS or EPCQ serial configuration devices. Data written to a serial configuration device should be shifted so that the least-significant bit is loaded into the device first.

For example, if the .rpd contains the byte sequence 02 1B EE 01 FA, the serial data programmed into the configuration device must be 100-0000 1101-10000111-0111 1000-0000 0101-1111.

JEDEC format

A JED format is a Xilinx JEDEC hardware configuration file.

It is used for programming complex programmable logic devices (CPLDs/FPGAs).

The JEDEC format consists of a start-of-text character (STX), various fields, an end-of-text character (ETX), and a transmission checksum.

Here is an example file taken from the reference below:

```
QP48*
QF17200*
G0*
F0*
L00000
11011010110001000010000100001000011010100001100011000100000100
0010000100001111111
01101011001010000100001000010000110101000011010011000100000100
0010000100001111111
```

```
10110001010000010000100001101010000110001100010000100111000000
0010000100001111111
*
CF5F2*
```

POF Format

The POF (Programmable Object File) format provides a highly compact data format to enable translation of high bit count logic devices efficiently. This format currently applies to Altera/Intel devices. The information contained in the file is grouped in packet. The POF is composed of a header and a list of packets.

HEX Dump format

The HEX Dump format is a human readable hexadecimal dump.

Each data byte is represented as 2 hex characters, and is separated by white space. The address is set by using a sequence of 8 hex characters.

Here is an example of ASCII HEX file.

It contains the data Hello, World to be loaded at address 0x1000:

0001000 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 0A

Infineon Tricore Hex/SRec format

This format must be selected for Infineon Tricore MCU family only. It converts an Intel-Hex or Motorola S-Rec file remapping the address from h80000000 to hA0000000.

TI-TXT format

The TI-TXT format is used by the Texas Instruments MSP430 devices.

The TI-TXT hex format supports 16-bit hexadecimal data.

It consists of one or more sections, followed by the end-of-file indicator.

Each section consists of an at (@) sign followed a start address (in hexadecimal)

and then data bytes (in hexadecimal).

The end-of-file indicator is the letter Q followed by a newline.

Here is an example TI-TXT file taken from the reference below:

```
@F000
31 40 00 03 B2 40 80 5A 20 01 D2 D3 22 00 D2 E3
21 00 3F 40 E8 FD 1F 83 FE 23 F9 3F
@FFFE
00 F0
Q
```

Algocraft has introduced an optional new field (#) to represent the data size (8,16 or 32 bits)

#8 = data byte

#16 = data word

#32 = data longword

This format is used when you need to specify few bytes/words value for configuration memory. An example:

```
#16
@8
0000
0000
0000
0000
0010
0000
0000
@1D
0000
01FF
q
```

Espressif Binary format

This format must be selected for Espressif ESP32 devices only.

The binary file loaded is converted and compressed using gzip deflate algorithm for improving programming times.

Atmel XCFG format (ATMXT641T device)

A XCFG format is a Microchip configuration file for ATMXT641T

Here is an example file taken from the reference below:

```
[VERSION_INFO_HEADER]
FAMILY_ID=164
VARIANT=61
VERSION=16
BUILD=170
VENDOR_ID=0x0
PRODUCT_ID=0x0
CHECKSUM=0x3E7598
INFO_BLOCK_CHECKSUM=0x75CEF3
[APPLICATION_INFO_HEADER]
NAME=maXTouchStudio
VERSION=1.6.828
[DEBUG_DIAGNOSTIC_T37 INSTANCE 0]
OBJECT_ADDRESS=214
OBJECT_SIZE=130
0 1 MODE=0
1 1 PAGE=0
2 1 DATA[0]=0
3 1 DATA[1]=0
4 1 DATA[2]=0
5 1 DATA[3]=0
6 1 DATA[4]=0
```

NXP PMIC CFG format (PF0100A device)

A PMIC CFG format is a NXP configuration file for some PMIC devices

Here is an example file taken from the reference below:

```
DEVICE:PF0100A
ADDR:E3:DATA:04 // OTP_SYS_PWRON_CFG|OTP_SYS_DVS_CLK|OTP_SYS_SEQ_CLK
ADDR:E7:DATA:02 // OTP_SYS_TBB_MODE|OTP_SYS_IS_PROGRAMMED
ADDR:E8:DATA:00 // OTP_SYS_PGOOD
ADDR:FF:DATA:08 // OTP_SYS_I2C_DEVICE_ADDRESS
ADDR:A0:DATA:2B // SW1AB VOUT
ADDR:A1:DATA:05 // SW1AB SEQ
ADDR:A2:DATA:01 // SW1AB CFG
ADDR:A8:DATA:2B // SW1C VOUT
ADDR:A9:DATA:05 // SW1C SEQ
ADDR:AA:DATA:01 // SW1C CFG
ADDR:AC:DATA:27 // SW2 VOUT
ADDR:AD:DATA:05 // SW2 SEQ
ADDR:AE:DATA:01 // SW2 CFG
ADDR:B0:DATA:26 // SW3A VOUT
```

Elmos E522 CFG format (E522.49 device)

A E522 CFG format is an Elmos configuration file for the E522.49 Led Driver.

Here is an example file taken from the reference below:

```
E522.48/49 - Config Wizard v1.1.4
SoftwareVersion=v1.1.4
##########
BUS_CONFIG_BUS_PULSE_0=0
BUS_CONFIG_BUS_PULSE_1=0
BUS_CONFIG_BUS_PULSE_2=0
BUS_CONFIG_BUS_PULSE_3=0
BUS_CONFIG_BUS_PULSE_4=0
BUS_CONFIG_BUS_PULSE_5=0
BUS_CONFIG_BUS_PULSE_6=0
BUS_CONFIG_BUS_PULSE_7=0
BUS_CONFIG_BUS_PULSE_8=0
BUS_CONFIG_BUS_PULSE_9=0
BUS_CONFIG_BUS_PULSE_10=0
BUS_CONFIG_BUS_PULSE_11=0
```

XDPL AHEX format (XDPL821x devices)

A AHEX format is an Infineon configuration file for XDPL8218/ XDPL8219 devices

Here is an example file taken from the reference below:

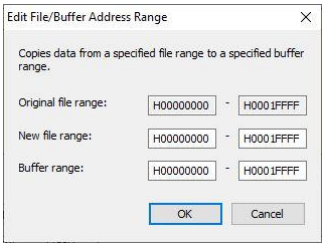
```
008082:2D7D
008083:0D46
008084:4400
008085:1096
008086:06C6
008087:00C7
008088:188E
008089:DDF9
00808A:4000
```

Micronas HAL format (HAL37xx devices)

This file format must be selected for Micronas HAL/HAR devices only.

Editing a Source File

When the source file loading is completed, you can change the image file settings by clicking on the **Edit** button:

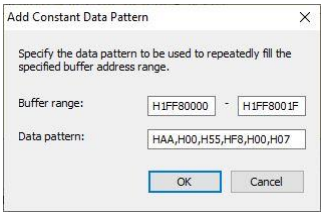


The **original file range** values show the start address and the end address of your source file. It is possible to remove data from the programming through the **new file range** section.

The **buffer range** values are used to set the offset for the start address and the end address according to the device memory map.

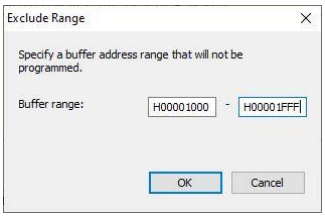
Const Data Pattern

This option is useful if you want to program some bytes which are not included in the firmware. For example, if you want to program the option bytes of STM32 devices, you can set the proper RDP level, BOR level ecc.



Exclude Range

This option can be used to skip a memory area from being programmed (typically for debug purpose).

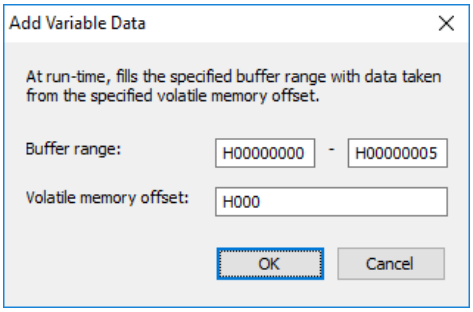


Variable Data

WriteNow! has built-in, dedicated memory banks for each programming site. This memory can be used to temporarily store variable data that will be written to the target device during programming. This is useful for serial numbering and for any other variable data that needs to be written to the target device at programming time.

To implement variable data programming:

- Set the target device address range to be programmed and the offset of the memory bank that will contain the variable data (the maximum data length available is **512 bytes**).



- Proceed to the end of the Project Creation wizard. Your programming project is now ready to accept variable data.
- Before executing the project, you must supply the variable data to each of the programming sites. To do so, send the **#volatile -o write** command through the WriteNow! Terminal (for more information, see "Volatile Memory Commands" on page 49). For example, if you want to program a MAC address:

```
#volatile -o write -s 1 -a h0 -l 6 -d [h00 h90 h96 h90 h48 h85]
#volatile -o write -s 2 -a h0 -l 6 -d [h00 h90 h96 h90 h48 h86]
```

Alternatively, you can skip steps 1 to 3, but you must manually edit your programming project by inserting an appropriate **#data -o set -c out -t volatile** command and subsequent appropriate programming commands (for more information, see "Data In/Out Commands).

Conversion Report and Analysis

The conversion report is useful to check if the source file conversion will be performed successfully. It is named as the image file and it is located in the same directory.

Here's an example:

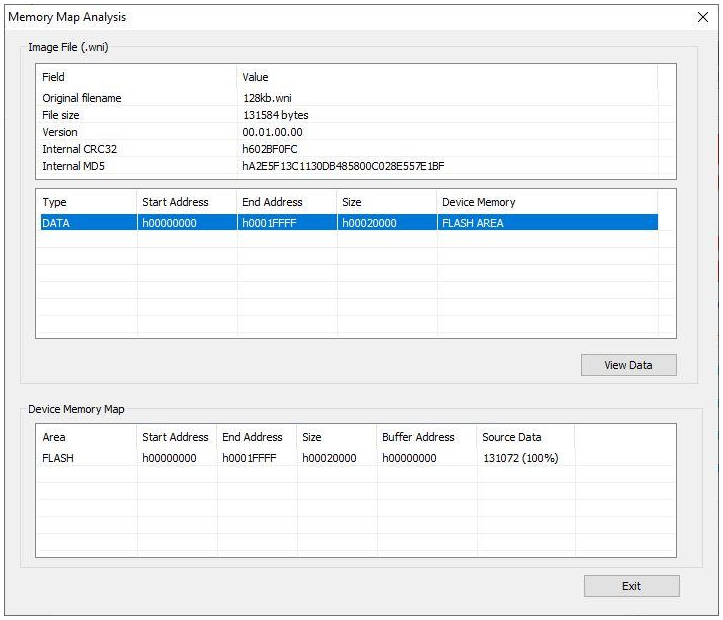
```
***** File Creation Report *****
Block Type    Start Addr  End Addr
==================================
```

Skip Area: *00000000 - 0000FFFF*

Data Area: *00010000 - 0001FFFF*

The "Skip Area" is not included in the memory programming as "Data Area" and "Fill Area" (see the following section).

The same report is also available on a dedicated window "Memory Map Analysis". This window contains a lot of useful information about the image file segmentation (.wni) and memory map of the device.



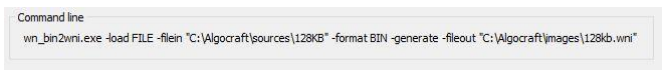
Fill Unused Location

This option allows to program the skip areas included in the source file with the specified value. The main reason to use this option is to avoid unexpected programming errors (e.g. data unaligned error).

It is very important to make sure that the fill value matches the blank value of the device.

Command Line conversion utility

It's also possible to create the image file using a command line utility (wn_bin2wni.exe). Here is an example from Image File Creation window:



5. Commands

Overview

WriteNow! is a slave unit and is always waiting for a new command coming from the master (PC).

When the programmer receives a SOF (Start Of Frame) character (**#**), indicating the start of a new command, it loads all incoming characters in a buffer until the reception of the return character (**\n**, ASCII code **h0A**). The maximum command length is 256 characters.

After reception of the return character, the programmer interprets and executes the received command; depending on the execution of the received command the protocol will answer to the master in three different ways.

1. If the command is correctly executed, the programmer answers with an OK frame.
2. If the command execution generates errors, the programmer answers with an ERR frame.
3. If the command takes long to execute, the programmer periodically answers with a BUSY frame, until command execution is over and an OK or ERR frame appears.

All commands and answers are case-insensitive.

Command Syntax

A WriteNow! command begins with the SOF character (**#**), followed by the command name, then followed by zero or more command switches, and terminates with the return character (**\n**).

This is an example of a WriteNow! valid command:

```
#status -o ping{\n}
```

OK Answer

An OK answer is composed of zero or more characters, followed by the **>** character, followed by the return character (**\n**).

This is an example of a WriteNow! OK answer:

```
pong>{\n}
```

ERR Answer

An ERR answer is composed of zero or more characters (usually the hexadecimal error code), followed by the **!** character, followed by the return character (**\n**).

This is an example of a WriteNow! ERR answer:

```
h40000103!{\n}
```

BUSY Answer

A BUSY answer is sent by the programmer to the PC if a command takes some time to execute. A BUSY answer is sent at most every 3 seconds. If no OK, ERR or BUSY answer is sent within 3 seconds from the last command sent to the programmer, a communication error has probably occurred.

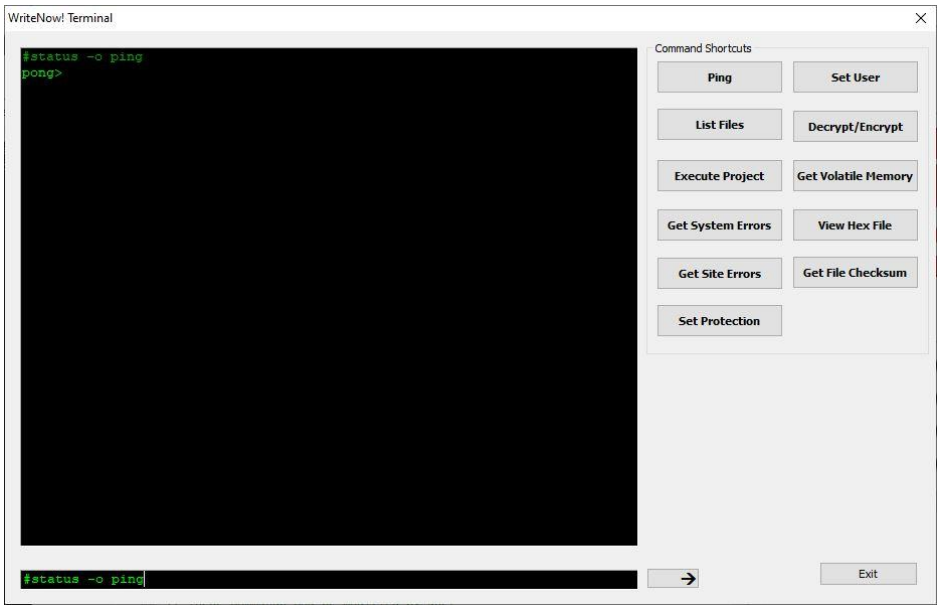
A BUSY answer is composed of zero or more characters, followed by the ***** character, followed by the return character (**\n**).

This is an example of a WriteNow! BUSY answer:

***{\n}**

A valid answer always ends with two characters: **>{\n}**, **!{\n}** or ***{\n}**, depending on whether an OK, ERR or BUSY frame is sent to the host. Additional return characters (**\n**) may be present in the answer, but they don't signal the end of the answer.

WriteNow! Terminal



Commands can be sent (and answers received) using any terminal application. For your convenience, the Project Generator application includes a Terminal window that will simplify the communication with the instrument. Just select **Tools > WriteNow! Terminal** to open the Terminal window.

Command Reference

The following pages list all of the WriteNow! commands, grouped by function, together with their syntax and usage examples.

Data In/Out Commands

Syntax

```
#data -o set -c <direction> -t file -f <filename> --fill <value>
#data -o set -c <direction> -t volatile
#data -o set -c in -t file -f \images\myimage.wni -s <MD5/CRC32>
```

Parameters

<direction>	**in** or **out**.
<filename>	Filename on the instrument's file system.

Description

Specify the source and destination of the programming data.

Examples

- Sets the input image file to be programmed, and subsequently programs it:

```
#data -o set -c in -t file -f \images\myfile.wni
>
#prog -o cmd -c program -m flash -s h8000 -t h8000 -l h8000
>
```

If you add the "-s MD5 | CRC32" flag, the programmer checks the full-integrity of the wni file and returns an error in case of wrong integrity.

- Sets the output file to receive binary data, and subsequently reads data from the target device:

```
#data -o set -c out -t file -f \images\dump.bin
>
#prog -o cmd -c read -m flash -s h8000 -t h8000 -l h8000
>
```

If you add the --fill flag, the programmer will fill the unused location with the <value> data.

Execution Command

Syntax

```
#exec -o prj -f <project> -s <sites>
#exec -o prj -f <project> -s <sites> --dmux <channel>
```

Parameters

<project>	The Project filename to execute.
<sites>	A 8-bit value indicating the programming sites to be enabled.

Description

Executes the specified Project over the specified programming sites. In case of error, a 32-bit value is returned. This value indicates whether the error is site-specific (bit 29 = 1) or system-specific (bit 29 = 0). If the error is site-specific, the 8 least significant bits (bits from 7 to 0) signal whether programming in the corresponding programming site (bit 7 = programming site 8, bit 0 = programming site 1) was successful (bit = 0) or not (bit = 1).

To retrieve error messages, use the **#status -o get -p err -v <site> -l <errlevel>** command, where **<site>** is **1** to **8** to retrieve a specific programming site error, or **0** to retrieve a system error. **<errlevel>** is the error detail information that is returned and can be **1**, **2**, **3**.

Examples

- Executes the Project "myprj.wnp" on programming sites 1, 2, 3, 4:

```
#exec -o prj -f \projects\myprj.wnp -s h0f
h20000003!
```

In this case, the returned error indicates that there are site-specific errors (bit 29 = 1) and that the sites where errors occurred are sites 1 and 2. To retrieve detailed error information about site 1, for example, the following command can be sent:

```
#status -o get -p err -v 1 -l 2
h5000001,23,"Error: Timeout occurred"
>
```

The answers indicates that Project line 23 issued a **h5000001** error, and the text between quotes explains the error.

File System Commands

Syntax

```
#fs -o rmdir -d <directory>
#fs -o mkdir -d <directory>
#fs -o dir -d <directory>
#fs -o del -f <filename>
#fs -o send -d <filename>
#fs -o receive -d <filename>
#fs -o enc --if images\myfile --of images\myfile.wnef --password <password>
#fs -o dec --if images\myfile.wnef --of images\myfile --password <password>
#fs -o get -f images/myimages.wni -p <info/calc> -r MD5
#fs -o get -f images/myimages.wni -p <info/calc> -r CRC32
#fs -o dump -f <filename> -a <offset> -l <len>
```

Parameters

<directory>	Full path of a directory.
<filename>	Full path of a filename.
<offset>	Address offset of the filename
<len>	Length of data

Description

Allow to perform various operations on the programmer's file system.

Examples

- Shows the contents of the programmer's root directory:

```
#fs -o dir -d \
2010/06/21  16:35:06  [DIR]              projects
2010/06/21  16:35:16  [DIR]              sys
2010/06/21  16:35:20  [DIR]              images
2010/06/21  16:35:26  [DIR]              drivers
>
```

- CRC32/MD5 values:

```
#fs -o get -f images/myimages.wni -p info -r MD5
"D0B8E00F64710AFB14EAE012D2225C8D">

#fs -o get -f images/myimages.wni -p info -r CRC32
h160AB585>
```

These commands return the MD5/CRC32 values without checking the data integrity. The CRC32 and MD5 values are not calculated on the entire file, but it skips the first **512 bytes.**

The similar commands with the option -p calc, return the calculated MD5/CRC32 values of the entire file.

```
#fs -o get -f images/myimages.wni -p info -r MD5
"3AED69753C360B5B9615DDDF453D4249">
#fs -o get -f images/myimages.wni -p info -r CRC32
hBFAED26D>
```

- Dumps the contents of any f:

```
#fs -o dump -f \images\dump.bin -a 0 -l 128
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
>
```

Programming Commands

Syntax

```
#load -l <driver> -m <manufacturer> -d <device>
#dev -o begin
#dev -o end
#dev -o set -p <parameter> -v <value>
#conf -o begin
#conf -o end
#conf -o set -p <parameter> -v <value>
#prog -o begin
#prog -o end
#prog -o cmd -c pps -v <pps value>
#prog -o cmd -c connect
#prog -o cmd -c disconnect
#prog -o cmd -c unprotect
#prog -o cmd -c erase -m <mem type> -t <tgt addr> -l <len>
#prog -o cmd -c blankcheck -m <mem type> -t <tgt addr> -l <len>
#prog -o cmd -c program -m <mem type> -s <src addr> -t <tgt addr> -l <len>
#prog -o cmd -c verify -v <ver mode> -m <mem type> -t <tgt addr> -l <len>
#prog -o cmd -c read -m <mem type> -s <dst addr> -t <tgt addr> -l <len>
#delay -o set -p <us/ms> -v <value>
```

Parameters

<driver>	Filename of the .wnd driver.
<manufacturer>	Target device's silicon manufacturer.
<device>	Target device code.
<parameter>	Target parameter to set.
<value>	Value of the corresponding parameter.
<pps value>	**on** or **off**.
<mem type>	Target memory type.
<tgt addr>	Target start address.
<len>	Data length.
<src addr>	Source (buffer) start address.
<ver mode>	Verify mode: **read** or **chks**.
<dst addr>	Destination start address.

Description

Perform various programming settings and operations on the target device.

Status Commands

Syntax

```
#status -o ping
#status -o get -p err -v <site> -l <errlevel>
```

Parameters

<site> **1** to **8** to get programming site errors. Use **0** to return system errors.

<errlevel> **1** to **3**.

Description

Get instrument status or error information.

When retrieving error information, one or more error lines (depending on the **<errlevel>** parameter) are returned. Each line begins with a 32-bit code, which codifies the following information:

Bit 31:	Reserved
Bit 30:	If 1, an error message in text format is available.
Bit 29:	If 1, the error is programming site specific.
Bit 28:	If 1, the error is driver (programming algorithm) specific.
Bit 27:	If 1, the error is a system fatal error.
Bits 26 to 24:	Reserved.
Bits 23 to 0:	Error code. If bit 29 is 1, then bits 7 to 0 signal whether programming in the corresponding programming site (bit 7 = programming site 8, bit 0 = programming site 1) was successful (bit = 0) or not (bit = 1).

Examples

- Pings the instrument to check if communication is OK:

```
#status -o ping
pong>
```

Retrieves the last generated errors, on programming site 1, with different error levels:

```
#status -o get -p err -v 1 -l 1
H50000023
>
#status -o get -p err -v 1 -l 2
H50000023,71,"Connection Error."
>
#status -o get -p err -v 1 -l 3
H50000023,71,"Connection Error.","algo_api",337
H10000000,71,"","st701_cmds",432
```

```
H10000000,71,"","st701_entry",287
H10000000,71,"","st701_icc",208
H10000001,71,"","hal_icc1",144
>
```

System Commands

Syntax

```
#sys -o get -p sn
#sys -o get -p ver -v <code>
#sys -o set -p lliop -s <prj sel> -f <prj filename>
#sys -o get -p lliop -s <prj sel>
#sys -o set -p ip -v <192.168.1.10>
#sys -o get -p ip
#sys -o set -p nm -v <255.255.255.0>
#sys -o get -p nm
#sys -o set -p gw -v <192.168.1.1>
#sys -o get -p gw
#sys -o set -p clip -v <2101>
#sys -o get -p clip
#sys -o set -p protection --password <password> --enable <yes/no>
#sys -o set -p user --name <admin/operator> --password <password>
#sys -o rst
```

Parameters

<code>	**sys** or **driver**.
<prj sel>	Project number, as selected by the PRJ_SEL[5..0] lines on the Low-Level Interface connector.
<prj filename>	Project file associated to **<the prj sel>** setting.

Description

Set or get instrument's internal parameters.

Examples

- Retrieves the instrument's serial number:

```
#sys -o get -p sn
00100>
```

- Associating the project for Standalone Mode:

Associates the project **test.wnp** to the project number 1:

```
#sys -o set -p lliop -s 1 -f \projects\test.wnp
>
```

- Sets a new LAN configuration:

IP:

```
#sys -o set -p ip -v 10.0.0.10
>
```

Netmask:

```
#sys -o set -p nm -v 255.255.255.0
>
```

Gateway:

```
#sys -o set -p gw -v 10.0.0.1
>
```

The new configuration will be added after the programmer reboot.

- Reset via software:

```
#sys -o rst
>
```

The programmer will be restarted after about 5 seconds.

Time Commands

Syntax

```
#time -o set -p date -d <date>
#time -o set -p time -d <time>
#time -o get -p date
#time -o get -p time
```

Parameters

<date>	A date in the format **yyyy/mm/dd**.
<time>	A time in the format **hh:mm:ss**.

Description

Set or get the instrument's date and time. Once set, the date and time are maintained even when the instrument is powered off.

Examples

- Sets the date/time to February 1st, 2011, at noon:

```
#time -o set -p date -d 2011/02/01
>
#time -o set -p time -d 12:00:00
>
```

- Retrieves the instrument's date and time:

```
#time -o get -p date
2011/02/01>
#time -o get -p time
12:02:05>
```

Volatile Memory Commands

Syntax

```
#volatile -o write -s <site> -a <start address> -l <len> -d <data>
#volatile -o read -s <site> -a <start address> -l <len>
```

Parameters

<site>	Programming site. **1** to **8** to set specific site data, **0** to set the same data for all sites.
<start address>	Volatile memory starting address.
<len>	Data length.
<data>	A data array.

Description

Read and write data from/to the instrument's volatile memory.

Examples

- Uses the volatile memory on site 1 to store the target board's MAC address:

```
#volatile -o write -s 1 -a h0 -l 6 -d [h00 h90 h96 h90 h48 h85]
>
```

- Retrieves data from site 1 volatile memory:

```
#volatile -o read -s 1 -a h0 -l 6
1,[h00 h90 h96 h90 h48 h85]>
```

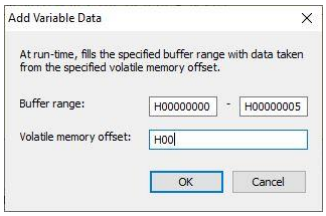
6. Programming Variable Data and Serial Number

Thanks to its built-in volatile memory, WriteNow! is able to store variable parameters during the programming process. These parameters can change at every programming cycle and, in case of a multi-site programmer, different values for each site.

To program a variable data (ex. Serial number) to the target device memory, follow these steps:

Define the device address range

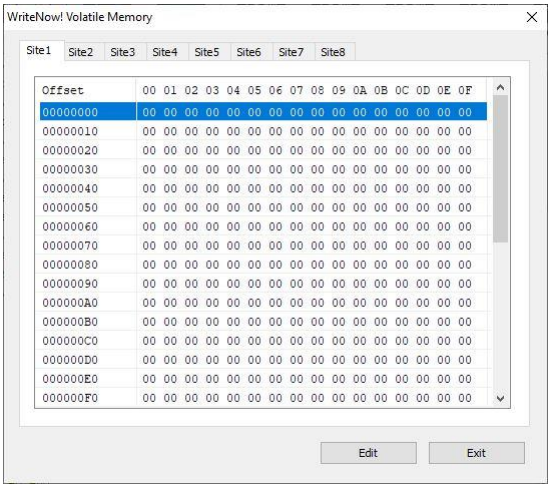
During the Image file creation process, set the target device address range to be programmed and the offset of the volatile memory bank that will contain the variable data, as follows:



Generate and set the variable memory

Before any programming cycle, the host system can send the variable data and then it transfers these values to the WriteNow! volatile memory by using **the "#volatile -o write"** commands.

You can view or edit the content of the built-in volatile memory from the following window:



Start the programming process

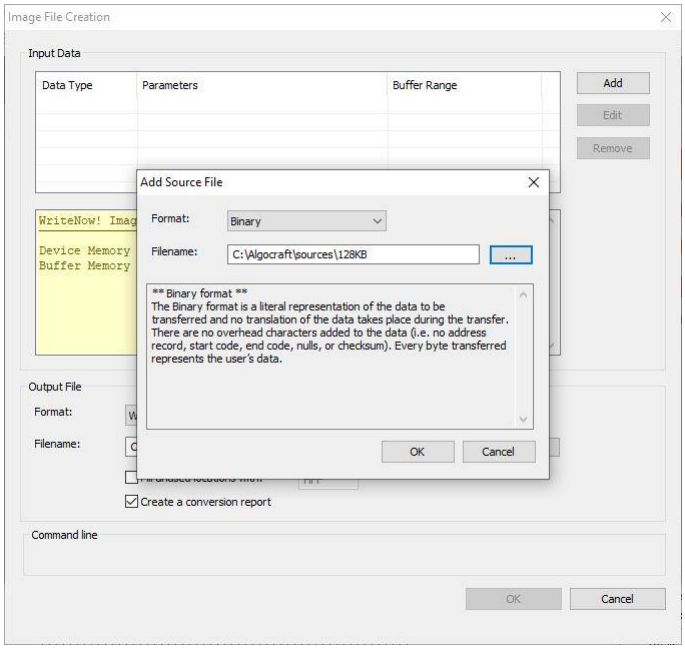
The content of the volatile memory is then programmed into the target device during the project execution.

Example

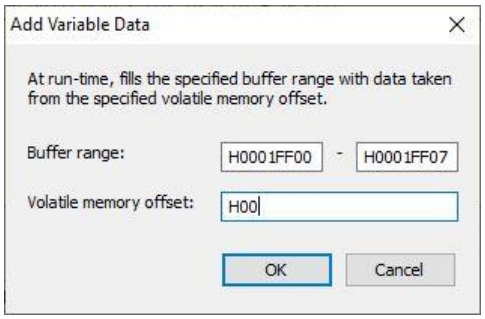
Let's say the internal memory of the target device is mapped in the range 0x00000 - 0x1FFFF (128KB), which is programmed completely by using a binary file. In addition, a serial number must be set from 0x1FF00 to 0x1FF07 (8 bytes) and a production batch code from 0x1FFFC to 0x1FFFF (4 bytes). The serial number is different for each board while the production batch code is common for all of them.

Step1: Create the image file

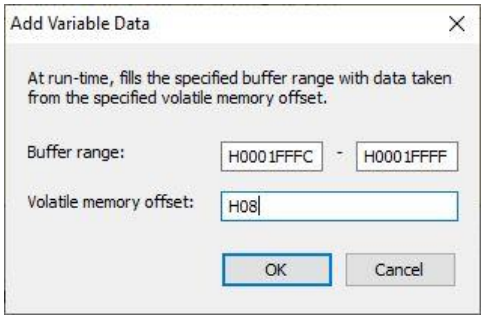
Add the binary file:

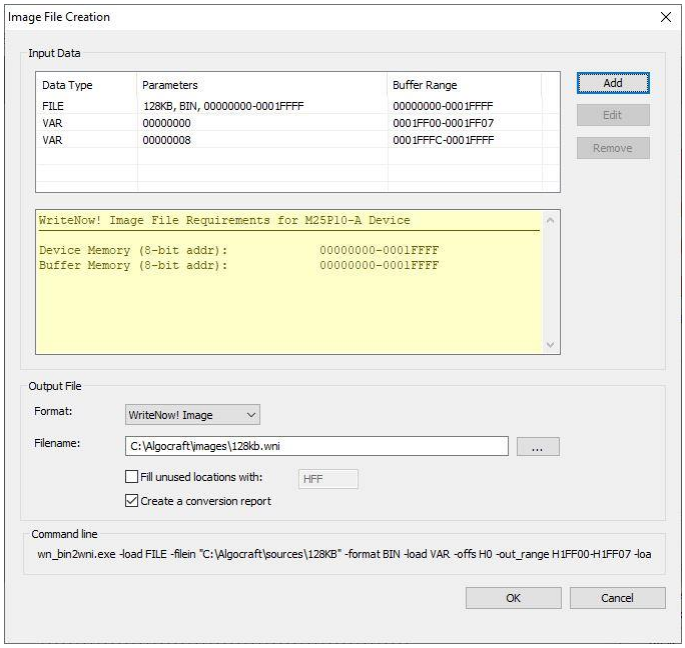


Then define a variable buffer range for the serial number. The data will be loaded at offset 0x00 of the volatile memory.



And finally define the variable buffer range for the production batch code. The data will be loaded at offset 0x08 of the volatile memory.





Step2: Set the variable memory

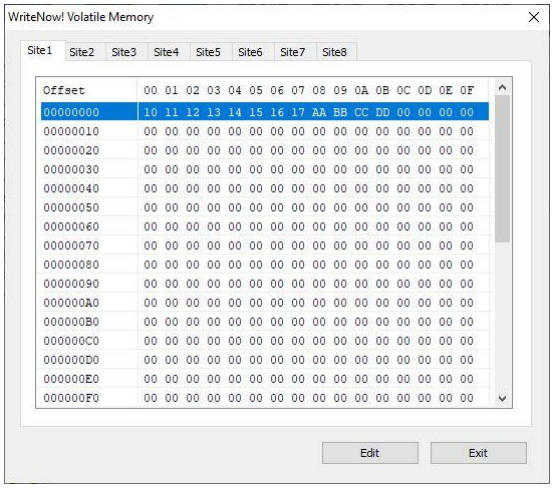
The serial numbers are:

```
Site1 = h10 h11 h12 h13 h14 h15 h16 h17
Site2 = h20 h21 h22 h23 h24 h25 h26 h27
Site3 = h30 h31 h32 h33 h34 h35 h36 h37
Site4 = h40 h41 h42 h43 h44 h45 h46 h47
```

The production batch code is hAA hBB hCC hDD

Before starting the project execution, set the volatile memory area for each site by using a sequence of "#volatile -o write" commands, like this:

```
#volatile -o write -s 1 -a h00 -l 8 -d [h10 h11 h12 h13 h14 h15 h16 h17]
>
#volatile -o write -s 2 -a h00 -l 8 -d [h20 h21 h22 h23 h24 h25 h26 h27]
>
#volatile -o write -s 3 -a h00 -l 8 -d [h30 h31 h32 h33 h34 h35 h36 h37]
>
#volatile -o write -s 4 -a h00 -l 8 -d [h40 h41 h42 h43 h44 h45 h46 h47]
>
#volatile -o write -s 1 -a h08 -l 4 -d [hAA hBB hCC hDD]
>
#volatile -o write -s 2 -a h08 -l 4 -d [hAA hBB hCC hDD]
>
#volatile -o write -s 3 -a h08 -l 4 -d [hAA hBB hCC hDD]
>
#volatile -o write -s 4 -a h08 -l 4 -d [hAA hBB hCC hDD]
>
```



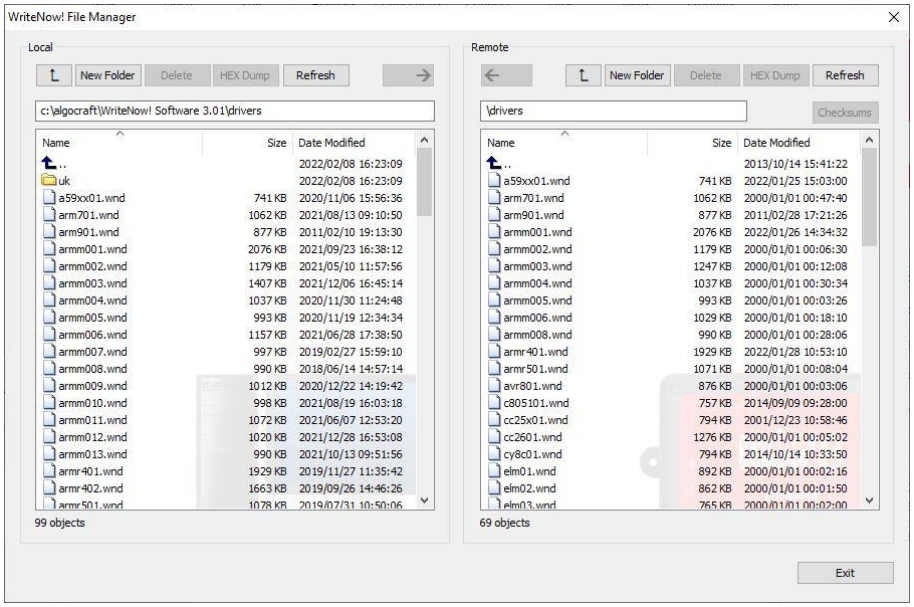
Step3: Start Programming

Run the project.

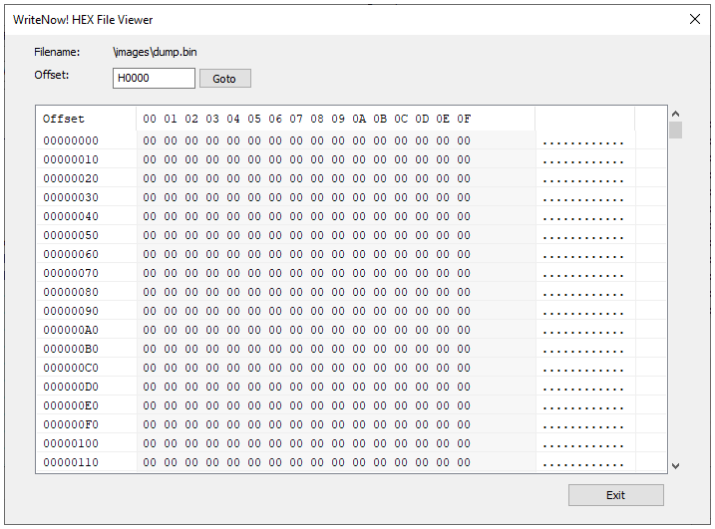
7. WriteNow! File System

Overview

WriteNow! has a large, built-in non-volatile memory, used to store the various files required by the instrument: programming projects, image files, etc. This memory is organized by a file system. You can explore the WriteNow! files either by using a Terminal application and sending file-system related commands, or (more simply) by using the File Manager window of the Project Generator application. The File Manager window allows you to easily see the instrument file structure and transfer files with the PC. To open the File Manager, choose **Tools > WriteNow! File Manager** from the Project Generator menu.



From this window you can also view the content of any file present in the programmer memory or PC. Just press "HEX Dump" button and the following window will appear:



File System Structure

The files required by the instrument are organized in various folders, as explained below:

- **\drivers folder**: contains programming algorithms (.wnd files).
  These files are provided by Algocraft.
- **\sys folder:** contains systems files, such as programming licenses, firmware files, etc.
  These files are provided by Algocraft.
- **\project folder:** contains programming projects (.prj files).
  You can create programming projects using the Project Generator application.
- **\images folder:** contains WriteNow! image files to be programmed to the target (.wni files). WriteNow! image files contain all the information needed to program a target device memory. These files are created by the Project Generator application.

You can create additional folders, but the four folders listed above must always be present on the WriteNow! file system and must not be removed. Additionally, do not remove or rename the contents of the \SYS folder.

8. Standalone Mode

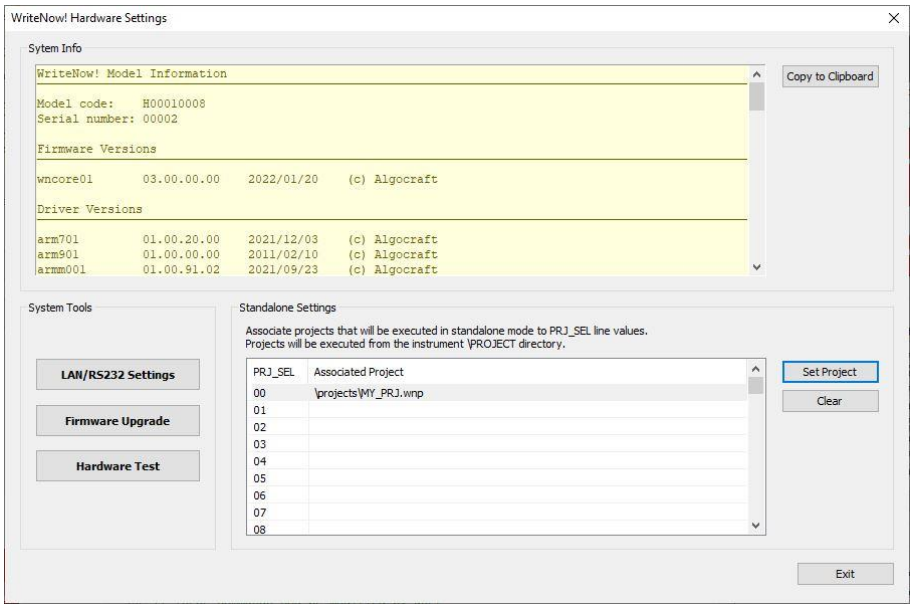
Overview

WriteNow! can work with no connection to a PC (standalone mode). In standalone mode, the instrument is controlled through a low-level connection interface.

Project Assignment

Before working in standalone mode, you must link the project(s) to be executed with the corresponding PRJ_SELx pattern.

To do so, in the WriteNow! Project Generator application select **Settings > Hardware Settings**. In the window that will appear, assign a PRJ_SEL value to the project filename by clicking the **"Set Project"** button for each PRJ_SEL configuration you wish to set up. For example, choose PRJ_SEL00 in case all the PRJ_SELx lines are driven low, PRJ_SEL01 if only PRJ_SEL0 line is driven high etc. (refer to the hardware manual of your product for further details).



Note: A project can be linked by using the following serial commands:

```
#sys -o set -p lliop -s <prj sel> -f <prj filename>
#sys -o get -p lliop -s <prj sel>
```

9. Protected Mode and Data Encryption

Protected Mode

A security feature has been designed to protect the intellectual property of the embedded firmware code. This is possible by locking the programmer with a unique private password, which will be used to encrypt all the files to be sent or received from the host PC. Care must be taken when choosing to lock the programmer. In fact, if the password is lost, the WriteNow! memory cannot be written anymore. The protection state is referred as follows:

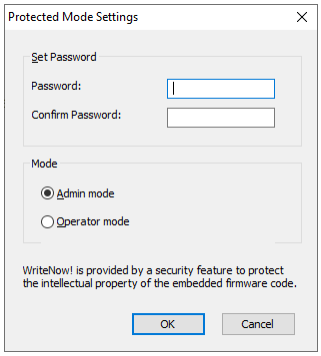
- Protection mode disable -> Level 00
- Protection mode enable -> Level 02

When the protection is active, two user modes are available:

- Operator mode
- Admin mode

In operator mode, the user can transfer only encrypted data to/from the programmer. In admin mode, the user can transfer any file to/from the programmer. Only in admin mode it is allowed to change the protection level. The operator mode will be reactivated starting from the next reboot.

To get access to the security settings, select "Tools->Protected Mode" and the following window will appear:



The same can be achieved by using WriteNow! commands. Especially, the following command is used to activate the protected mode:

```
#sys -o set -p protection --password ALGOCRAFT --enable yes
>
```

Now the current protection level is 02 and the user mode is operator.
To enter in admin mode use the following command:

```
#sys -o set -p user --name admin --password ALGOCRAFT
>
```

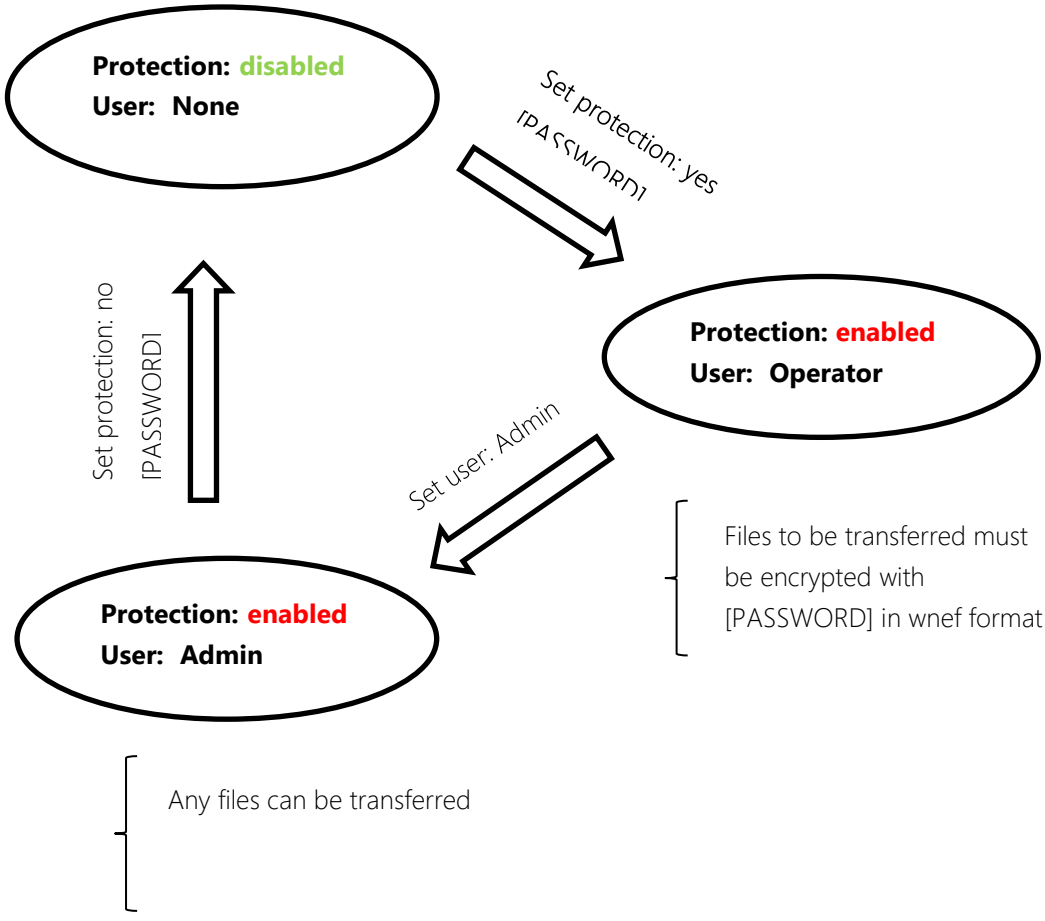
(Optional) Make sure that the password matched by checking the current user mode:

```
#sys -o get -p user
Admin
>
```

Now that you have the admin rights, send the following command if you want to disable the protected mode:

```
#sys -o set -p protection --password ALGOCRAFT --enable no
>
```

Protection model.



Data Encryption

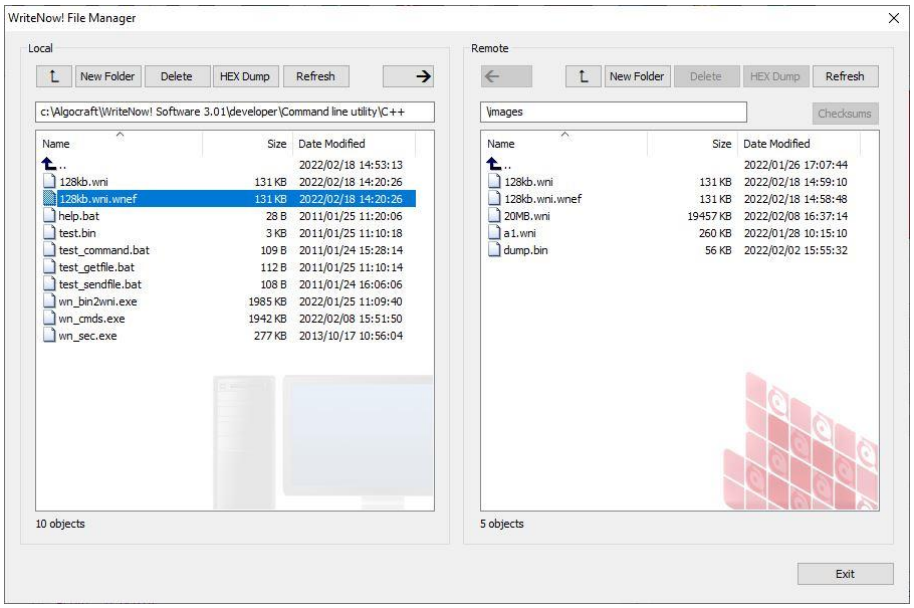
Encrypting the files can be done through the wn_sec.exe command line tool, located in the \developer\C++ directory, as follows:

1. Encrypt your files with the same password used to lock the programmer:

```
wn_sec.exe -o -enc -if ".\myprj.wnp" -of ".\myprj.wnp.wnef" -password ALGOCRAFT
```

```
wn_sec.exe -o -enc -if ".\myimage.wni" -of ".\myimage.wni.wnef" -password ALGOCRAFT
```

2. Transfer the encrypted files to the programmer:



3. In WriteNow! terminal, type the following command to decrypt the file (the password is NOT mandatory):

```
#fs -o dec --if \images\myimage.wni.wnef --of \images\myimage.wni
```

10. WriteNow! API

Overview

You can build your own PC software that interfaces to the instrument, by using the provided WriteNow! Application Programming Interface (API). The WriteNow! API consists of a series of functions, contained in the **wn_comm** DLL (Dynamic-Link Library), which allow you to set up and control the programmer.

The **wn_comm** DLL is located in the **\Developer** folder, relative to the WriteNow! software installation path. In the same folder you can find the source code of sample applications, in various programming languages, that use the **wn_comm** DLL.

The `wn_comm` DLL is available in Visual C++ and in Visual C# (COM Interop class library based on Microsoft .NET).

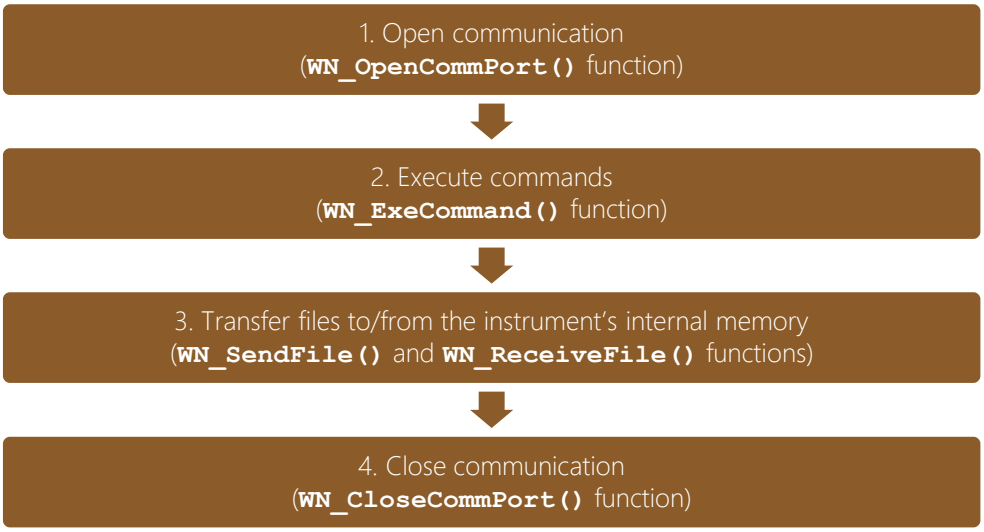
Additionally, a command line application (**wn_cmds.exe**) is provided, which reads a programming command from the stdin, sends the command to the instrument, and writes the command answer on the stdout.
For example, using the following commands to send a file through COM, USB or Ethernet ports:

Including the API in Your Application

To use the WriteNow! API, you must:

- Include the **"wn_comm.lib"** and **"wn_comm.h"** files in your application project (only needed for Visual C++ projects);
- Include the **"wn_comm.tlb"** file in your application project (only for Visual for C++/CLI projects);
- Copy the **"wn_comm.dll"** file in the same folder of your application executable (this file must also be redistributed with your application).

The typical programming flow to interface with WriteNow! is the following:



Function Reference (C++ Library)

API functions are listed and explained alphabetically in the following pages.

WN_CloseCommPort()

Prototype

ASCII version:

```
WN_COMM_ERR WINAPI WN_CloseCommPortA (WN_COMM_HANDLE handle);
```

Unicode version:

```
WN_COMM_ERR WINAPI WN_CloseCommPortW (WN_COMM_HANDLE handle);
```

Description

Closes the communication channel with the instrument.

Return Value

0	The function call was successful.
!=0	The function call was unsuccessful. Call the **WN_GetLastErrorMessage()** function to get error information.

Parameters

handle	Communication handle returned by the **WN_OpenCommPort()** function.

WN_ExeCommand()

Prototype

ASCII version:

```
WN_COMM_ERR WINAPI WN_ExeCommandA (WN_COMM_HANDLE handle, const char
*command, char *answer, unsigned long maxlen, unsigned long timeout_ms,
WN_ANSWER_TYPE *type);
```

Unicode version:

```
WN_COMM_ERR WINAPI WN_ExeCommandW (WN_COMM_HANDLE handle, const wchar_t
*command, wchar_t *answer, unsigned long maxlen, unsigned long timeout_ms,
WN_ANSWER_TYPE *type);
```

Description

Executes a WriteNow! command. This function automatically sends a command to the instrument and returns the answer read back from the instrument. This function combines the **`WN_SendFrame()`** and **`WN_GetFrame()`** function in a single call.

Return Value

`0`	The function call was successful.
`!=0`	The function call was unsuccessful. Call the **`WN_GetLastErrorMessage()`** function to get error information.

Parameters

`handle`	Communication handle returned by the **`WN_OpenCommPort()`** function.
`command`	A valid WriteNow! command.
`answer`	The answer read back from the instrument in response to the command sent.
`maxlen`	Maximum length, in characters, of the answer buffer.
`timeout_ms`	Time (in milliseconds) before the function times out.
`type`	Type of answer received: can be: **`WN_ANSWER_ACK`** (an OK frame was received); **`WN_ANSWER_NACK`** (an ERR frame was received); **`WN_ANSWER_TOUT`** (command timed out before an answer could be received).

WN_GetFrame()

Prototype

ASCII version:

```
WN_COMM_ERR WINAPI WN_GetFrameA (WN_COMM_HANDLE handle, char *answer,
unsigned long maxlen, unsigned long timeout_ms);
```

Unicode version:

```
WN_COMM_ERR WINAPI WN_GetFrameW (WN_COMM_HANDLE handle, wchar_t *answer,
unsigned long maxlen, unsigned long timeout_ms);
```

Description

Reads the answer to the command sent by the **WN_SendFrame()** function.

Return Value

0	The function call was successful.
!=0	The function call was unsuccessful. Call the **WN_GetLastErrorMessage()** function to get error information.

Parameters

handle	Communication handle returned by the **WN_OpenCommPort()** function.
answer	The answer read back from the instrument in response to the command sent.
maxlen	Maximum length, in characters, of the answer buffer.
timeout_ms	Time (in milliseconds) before the function times out.

WN_GetLastErrorMessage()

Prototype

ASCII version:

```
void WINAPI WN_GetLastErrorMessageA (char *error_msg, unsigned long
tring_len);
```

Unicode version:

```
void WINAPI WN_GetLastErrorMessageW (wchar_t *error_msg, unsigned long
string_len);
```

Description

Returns a string containing the last WriteNow! error message.

Parameters

error_msg	The string that will receive the error message.
msg_len	Length, in characters, of the error message buffer.

WN_ReceiveFile()

Prototype

ASCII version:

```
WN_COMM_ERR WINAPI WN_ReceiveFileA (WN_COMM_HANDLE handle, const char
*protocol, const char *src_filename, const char *dst_path, bool
force_transfer, WN_FileTransferProgressProc progress);
```

Unicode version:

```
WN_COMM_ERR WINAPI WN_ReceiveFileW (WN_COMM_HANDLE handle, const wchar_t
*protocol, const wchar_t *src_filename, const wchar_t *dst_path, bool
force_transfer, WN_FileTransferProgressProc progress);
```

Description

Receives a file from the instrument's internal memory and saves it to the PC.

Return Value

0	The function call was successful.
!=0	The function call was unsuccessful. Call the **WN_GetLastErrorMessage()** function to get error information.

Parameters

handle	Communication handle returned by the **WN_OpenCommPort()** function.
protocol	Transfer protocol. Must be **"ymodem"**.
src_filename	The full filename, including path, of the remote file.
dst_path	The PC path where to store the file.
force_transfer	If **TRUE**, file transfer will be executed even if a file with the same name and CRC exists on the PC; if **FALSE**, file transfer will be executed only if necessary.
progress	Address of a callback function that will receive progress information, or **0** if not used.

WN_SendFile()

Prototype

ASCII version:

```
WN_COMM_ERR WINAPI WN_SendFileA (WN_COMM_HANDLE handle, const char
*protocol, const char *src_filename, const char *dst_path, bool
force_transfer, WN_FileTransferProgressProc progress);
```

Unicode version:

```
WN_COMM_ERR WINAPI WN_SendFileW (WN_COMM_HANDLE handle, const wchar_t
*protocol, const wchar_t *src_filename, const wchar_t *dst_path, bool
force_ transfer, WN_FileTransferProgressProc progress);
```

Description

Sends a file to the instrument's internal memory.

Return Value

0	The function call was successful.
!=0	The function call was unsuccessful. Call the **WN_GetLastErrorMessage()** function to get error information.

Parameters

handle	Communication handle returned by the **WN_OpenCommPort()** function.
protocol	Transfer protocol. Must be **"ymodem"**.
src_filename	The source full filename.
dst_path	The remote instrument file system path where to store the file.
force_transfer	If **TRUE**, file transfer will be executed even if a file with the same name and CRC exists on the instrument; if **FALSE**, file transfer will be executed only if necessary.
progress	Address of a callback function that will receive progress information, or **0** if not used.

WN_SendFrame()

Prototype

ASCII version:

```
WN_COMM_ERR WINAPI WN_SendFrameA (WN_COMM_HANDLE handle, const char
*command);
```

Unicode version:

```
WN_COMM_ERR WINAPI WN_SendFrameW (WN_COMM_HANDLE handle, const wchar_t
*command);
```

Description

Sends a command to the instrument. Use the **WN_GetFrame()** function to retrieve the answer.

Return Value

0	The function call was successful.
!=0	The function call was unsuccessful. Call the **WN_GetLastErrorMessage()** function to get error information.

Parameters

handle	Communication handle returned by the **WN_OpenCommPort()** function.
command	A valid WriteNow! command.

WN_OpenCommPort()

Prototype

ASCII version:

```
WN_COMM_HANDLE WINAPI WN_OpenCommPortA (const char *com_port, const char *com_settings);
```

Unicode version:

```
WN_COMM_HANDLE WINAPI WN_OpenCommPortW (const wchar_t *com_port, const wchar_t *com_settings);
```

Description

Opens a RS-232, Ethernet or USB communication channel with the instrument.

Return Value

>0	Valid communication handle to use in subsequent functions.
NULL	The function call was unsuccessful. Call the **WN_GetLastErrorMessage()** function to get error information.

Parameters

com_port	Communication port. Can be **"COM"**, **"LAN"** or **"USB"**.
com_settings	RS-232 settings for **"COM"** port (e.g.: **"COM1,115200"**); Ethernet settings for **"LAN"** port (e.g.: **"192.168.1.100:2101"**); Virtual COM port (e.g. **"COM7")**

Function Reference (C++/CLI Library)

This new DLL version has been written in C# language by using the .NET Runtime Framework.

Once imported the **"wn_comm.tlb"** in a Visual C++/CLI environment, the .NET DLL is handled as a COM class within a WN_COMM namespace with the following methods:

```
    virtual HRESULT __stdcall WN_GetLastErrorMessageW (
        /*[in,out]*/ BSTR * error_msg,
        /*[in]*/ unsigned long string_len ) = 0;
      virtual HRESULT __stdcall WN_OpenCommPortW (
        /*[out]*/ VARIANT * handle,
        /*[in]*/ BSTR port,
        /*[in]*/ BSTR settings,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_CloseCommPortW (
        /*[in]*/ VARIANT handle,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_SendFrameW (
        /*[in]*/ VARIANT handle,
        /*[in]*/ BSTR command,
```

```
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_GetFrameW (
        /*[in]*/ VARIANT handle,
        /*[out]*/ BSTR * answer,
        /*[in]*/ unsigned long maxlen,
        /*[in]*/ unsigned long timeout_ms,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_ExeCommandW (
        /*[in]*/ VARIANT handle,
        /*[in]*/ BSTR command,
        /*[out]*/ BSTR * answer,
        /*[in]*/ unsigned long maxlen,
        /*[in]*/ unsigned long timeout_ms,
        /*[out]*/ unsigned char * type,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_ExeCommand_CBW (
        /*[in]*/ VARIANT handle,
        /*[in]*/ BSTR command,
        /*[out]*/ BSTR * answer,
        /*[in]*/ unsigned long maxlen,
        /*[in]*/ unsigned long timeout_ms,
        /*[out]*/ unsigned char * type,
        /*[in]*/ long progress,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_SendFileW (
        /*[in]*/ VARIANT handle,
        /*[in]*/ BSTR src_filename,
        /*[in]*/ BSTR dst_path,
        /*[in]*/ VARIANT_BOOL force_filetransfert,
        /*[in]*/ long progress,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_ReceiveFileW (
        /*[in]*/ VARIANT handle,
        /*[in]*/ BSTR src_filename,
        /*[in]*/ BSTR dst_path,
        /*[in]*/ VARIANT_BOOL force_filetransfert,
        /*[in]*/ long progress,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_GetLastErrorMessageA (
        /*[in,out]*/ SAFEARRAY * * error_msg,
        /*[in]*/ unsigned long string_len ) = 0;
      virtual HRESULT __stdcall WN_OpenCommPortA (
        /*[out]*/ VARIANT * handle,
        /*[in]*/ SAFEARRAY * port,
        /*[in]*/ SAFEARRAY * settings,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_CloseCommPortA (
        /*[in]*/ VARIANT handle,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_SendFrameA (
        /*[in]*/ VARIANT handle,
        /*[in]*/ SAFEARRAY * command,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_GetFrameA (
        /*[in]*/ VARIANT handle,
        /*[out]*/ SAFEARRAY * * answer,
        /*[in]*/ unsigned long maxlen,
        /*[in]*/ unsigned long timeout_ms,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_ExeCommandA (
        /*[in]*/ VARIANT handle,
```

```
        /*[in]*/ SAFEARRAY * command,
        /*[out]*/ SAFEARRAY * * answer,
        /*[in]*/ unsigned long maxlen,
        /*[in]*/ unsigned long timeout_ms,
        /*[out]*/ unsigned char * type,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_ExeCommand_CBA (
        /*[in]*/ VARIANT handle,
        /*[in]*/ SAFEARRAY * command,
        /*[out]*/ SAFEARRAY * * answer,
        /*[in]*/ unsigned long maxlen,
        /*[in]*/ unsigned long timeout_ms,
        /*[out]*/ unsigned char * type,
        /*[in]*/ long progress,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_SendFileA (
        /*[in]*/ VARIANT handle,
        /*[in]*/ SAFEARRAY * src_filename,
        /*[in]*/ SAFEARRAY * dst_path,
        /*[in]*/ VARIANT_BOOL force_filetransfert,
        /*[in]*/ long progress,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
      virtual HRESULT __stdcall WN_ReceiveFileA (
        /*[in]*/ VARIANT handle,
        /*[in]*/ SAFEARRAY * src_filename,
        /*[in]*/ SAFEARRAY * dst_path,
        /*[in]*/ VARIANT_BOOL force_filetransfert,
        /*[in]*/ long progress,
        /*[out,retval]*/ enum WN_COMM_RET * pRetVal ) = 0;
```

Note: refer to Function Reference (C++ Library) documentation for the explanation of each parameter.

Function Reference (C# Library)

This new DLL version has been written in C# language by using the .NET Runtime Framework.

Once added as reference in a Visual C# environment, the .NET DLL is handled as a usual class within a WN_COMM namespace with the following methods:

```
void WN_GetLastErrorMessageA(ref byte[] error_msg, uint string_len)

void WN_GetLastErrorMessageW(ref string error_msg, uint string_len)

WN_COMM_RET WN_OpenCommPortA(out object handle, byte[] port, byte[]
 settings)

WN_COMM_RET WN_OpenCommPortW(out object handle, string port, string
 settings)

WN_COMM_RET WN_CloseCommPortA(object handle)

WN_COMM_RET WN_CloseCommPortW(object handle)

WN_COMM_RET WN_SendFrameA(object handle, byte[] command)

WN_COMM_RET WN_SendFrameW(object handle, string command)

WN_COMM_RET WN_GetFrameA(object handle, out byte[] answer, uint max
len, uint timeout_ms)

WN_COMM_RET WN_GetFrameW(object handle, out string answer, uint max
len, uint timeout_ms)

WN_COMM_RET WN_ExeCommandA(object handle, byte[] command, out byte[
] answer, uint maxlen, uint timeout_ms, out WN_ANSWER_TYPE type)

WN_COMM_RET WN_ExeCommandW(object handle, string command, out strin
g answer, uint maxlen, uint timeout_ms, out WN_ANSWER_TYPE type)

WN_COMM_RET WN_ExeCommand_CBA(object handle, byte[] command, out by
te[] answer, uint maxlen, uint timeout_ms, out WN_ANSWER_TYPE type,
WN_ExeCommandProgressProc progress)

WN_COMM_RET WN_ExeCommand_CBW(object handle, string command, out st
ring answer, uint maxlen, uint timeout_ms, out WN_ANSWER_TYPE type,
WN_ExeCommandProgressProc progress)

WN_COMM_RET WN_SendFileA(object handle, byte[] src_filename, byte[]
 dst_path, bool force_filetransfert, WN_FileTransferProgressProc pr
ogress);

WN_COMM_RET WN_SendFileW(object handle, string src_filename, string
 dst_path, bool force_filetransfert, WN_FileTransferProgressProc pr
ogress);

WN_COMM_RET WN_ReceiveFileA(object handle, byte[] src_filename, byt
```

```
e[] dst_path, bool force_filetransfert, WN_FileTransferProgressProc
 progress);

WN_COMM_RET WN_ReceiveFileW(object handle, string src_filename, str
ing dst_path, bool force_filetransfert, WN_FileTransferProgressProc
 progress);
```

Note: refer to Function Reference (C++ Library) documentation for the explanation of each parameter.

11. Troubleshooting

This section reports some common problems that may arise during the typical usage. Please be aware, however, that working with a specific target device

may cause device-specific issues.

USB Driver issues (Windows 7 and earlier)

If the PC cannot establish a communication with the instrument, the USB driver may not have been correctly installed on your system.

To restore the USB driver, perform the following steps:

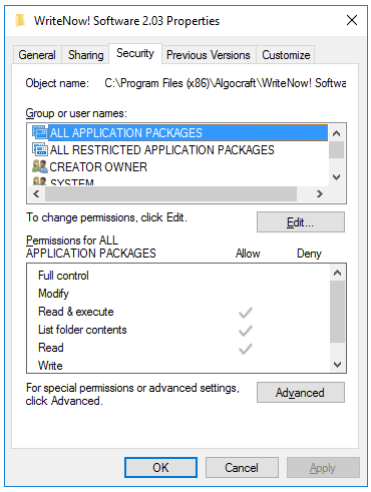
1. Connect Writenow! to the PC.
2. Open the Control Panel ("Start > Settings > Control Panel").
3. Open the "System" options.
4. Select the "Hardware" tab.
5. Click the "Device Manager" button.
6. The "Writenow! Programmer" device will be shown with an exclamation mark next to it. Double click on this device.
7. In the "General" tab, click the "Reinstall Driver" button. Follow the on-screen instructions.

Note: The USB support is available from 03.00.00.00 version of WriteNow! firmware.

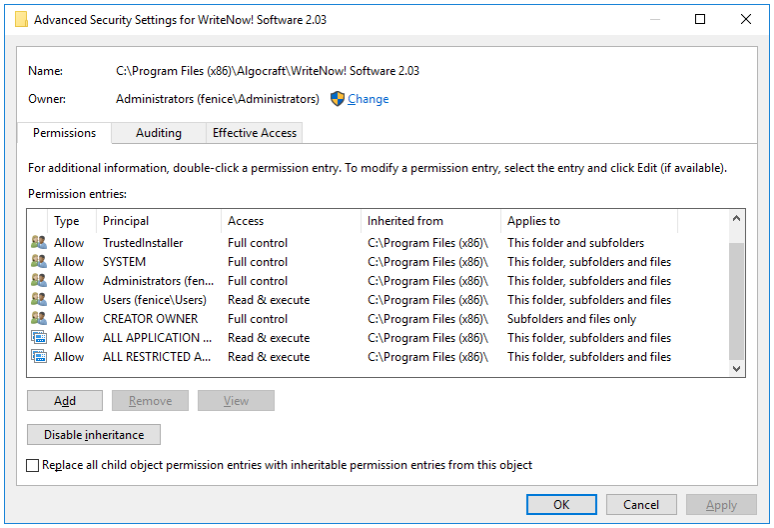
Get full access to WriteNow! folder on Windows 10

After the Software installation, if you choose to save it under the default directory (**C:\Program Files (x86)\Algocraft\WriteNow! Software X.XX**), make sure you have full control of it.

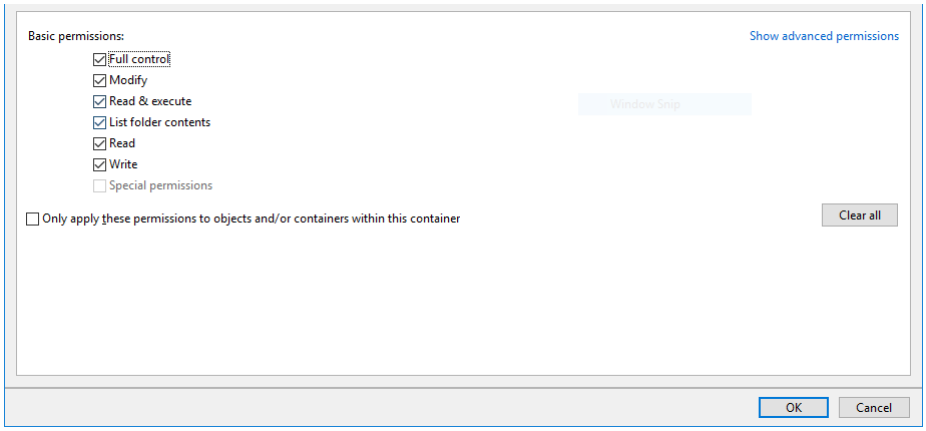
1. Open File Explorer and then locate the command utility line folder.
2. Right-click on the folder, click properties and then the Security tab.



3. Click the Advanced button. The "advanced Security Settings" windows will appear.



4. Click the Add button. The "Permission Entry" window will appear on the screen. Click "Select a principal" and select your account.
5. Set permissions to "Full control".



Diagnostic Test

WriteNow! has built-in self-test capabilities. This means that you can verify by yourself, at any time, the correct operation of the instrument hardware.

To perform the diagnostic test select "WriteNow! Hardware Settings > Hardware Test.