WriteNow! Series

Single and Parallel In-System Programmers

User's Manual

Rev. 1.02

algocraft

UM00010102EN

Copyright Information

Disclaimer

The material contained in this document is provided "as is", and is subject to being changed, 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 profit'.

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 EXCLUSIVE 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. WriteNow!—At a Glance

Overview

Congratulations for purchasing a WriteNow! In-System Programmer. Based on the proprietary WriteNow! Technology, the WriteNow! Series of In-System Programmers are a breakthrough in the Programming industry. The programmers support a large number of devices (microcontrollers, memories, CPLDs and other programmable devices) from various manufacturers and have a compact size for easy ATE/fixture integration. They work in standalone or connected to a host PC (RS-232, LAN and USB connections are built-in), and are provided with easy-to-use software utilities.

WN-PRG08A

Features

- Support of microcontrollers, serial and parallel memories, CPLDs and other programmable devices
- High-speed, parallel programming
- Compact size (fixture friendly)
- Standalone operations or host controlled
- Designed for easy ATE interfacing
- Robust and reliable
- Support of several programming interfaces (JTAG, BDM, SPI, I^2C, UART, etc.)
- Large built-in internal memory for projects, images, etc.
- Programmable power supply output (1.5-13V)
- Programmable I/O voltage (1.6-5.5V)
- High-speed I/O
- USB, LAN (isolated), RS-232 (isolated) and low-level interface (isolated)
- ISP I/O relay barrier (only available on the single-site model)
- I/O protection
- Wide range power supply (12-25V)

The shortest possible programming times are guaranteed due to a combination of highly optimized programming algorithms, local storage of programming data and high slew rate line driver circuitry.

Model Comparison

The following table summarizes the main features of the various WriteNow! family models.

WriteNow! Model Comparison

Feature	WN-PRG01A	WN-PRG04A	WN-PRG08A
General Features			
Programming Sites	1	4	8
Power Supply	12-25V	12-25V	12-25V
Device Type Support	Microcontrollers, CPLDs, Serial Memories	Microcontrollers, CPLDs, Serial Memories	Microcontrollers, CPLDs, Serial Memories, Parallel Memories
Programming Protocols	UART, SPI, JTAG, I²C, BDM, SWD, etc.	UART, SPI, JTAG, I²C, BDM, SWD, etc.	UART, SPI, JTAG, I²C, BDM, SWD, etc.
Relay Barrier	Yes	No	No
ISP Lines			
Adj. Voltage Range	1.6-5.5V	1.6-5.5V	1.6-5.5V
Adj. Voltage Resolution	100mV	100mV	100mV
Bidirectional Lines	6	24	48
Prog. Clock Out Lines	1	4	8
Programmable Power Supply (PPS)			
Range	1.5-15V	1.5-15V	1.5-15V
Resolution	100mV	100mV	100mV
Channels	1	4	8
Host Interface			
RS-232 (Isolated)	Yes	Yes	Yes
LAN (Isolated)	Yes, 100Mbit/s	Yes, 100Mbit/s	Yes, 100Mbit/s
USB	Yes, Full Speed	Yes, Full Speed	Yes, Full Speed
Low-Level Interface (Isolated)	START, OK/ERR, BUSY, PRJ_SEL[0..5]	START, START_ENA[1..4], OK/ERR[1..4], BUSY, PRJ_SEL[0..5]	START, START_ENA[1..8], OK/ERR[1..8], BUSY, PRJ_SEL[0..5]

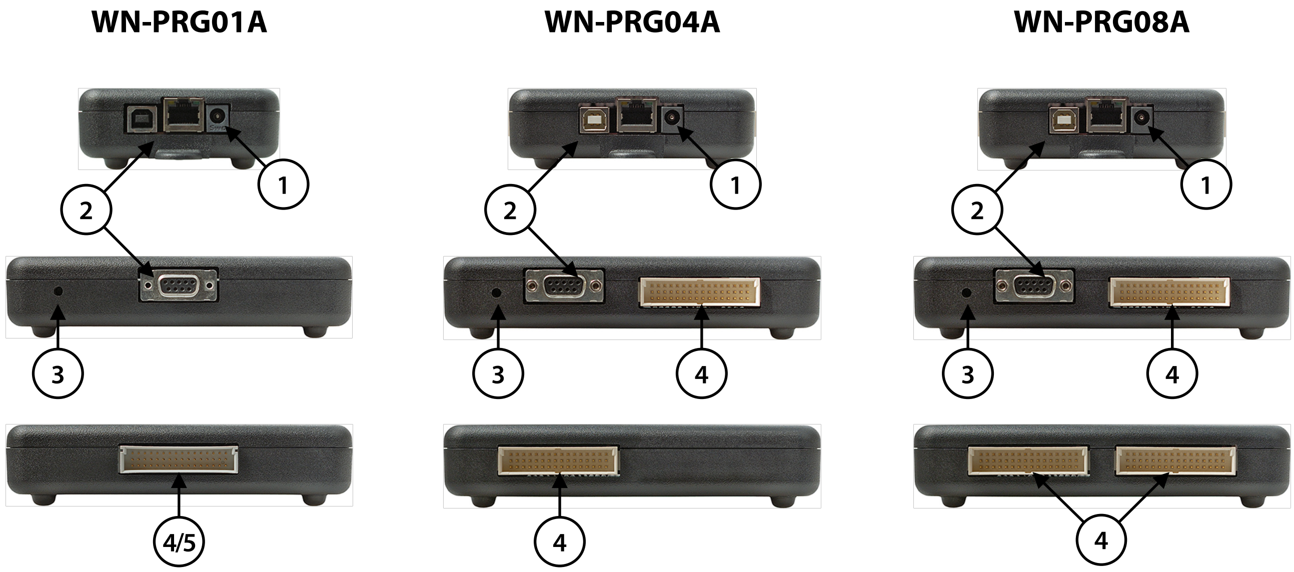
Package Checklist

The WriteNow! package includes the following items:

1. WriteNow! unit.
2. 15V power supply.
3. Serial and USB cables.
4. WriteNow! test board.
5. 48-way, female wire-wrap DIN41612 connector.
6. Software CD.

Connectors Overview

WriteNow! has several connectors for interfacing to a host PC, to an Automatic Test Equipment (ATE), and to the target system(s) to be programmed. The following pictures show where, depending on the model, the various connectors are located.



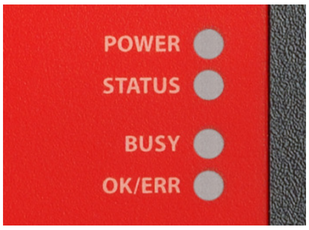
1. The POWER connector accepts a DC voltage between 12V and 25V.
2. The USB connector, LAN, and RS-232 connectors are used to interface the instrument to a PC.
3. The LAN RESET push button is used to reset LAN settings to their factory settings.
4. The ISP connector(s) are used to interface to the target system(s) to be programmed.
5. The LOW-LEVEL INTERFACE connector (which is merged with the ISP connector on the WN-PRG01A model) is used to interface the instrument to an ATE or other systems.

For details and pinout of the various connectors, see the “Connectors” chapter on page 51.

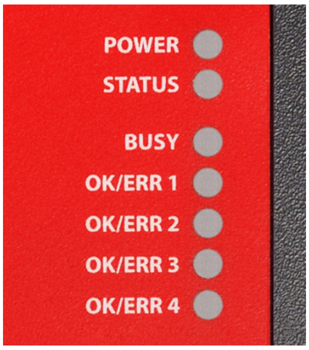
LEDs

The LEDs on the top cover of the instrument, from top to bottom, indicate:

1. POWER: the instrument is turned on.
2. STATUS: indicates system warnings. Normally off, blinks if the system needs user action (to retrieve detailed error information, see “Status Commands” on page 26)
3. BUSY: turns on when programming (when a programming project is being executed).
4. OK/ERR: result of programming. Each programming site has an OK/ERR LED, which turns green if programming on that site has been successful, or red otherwise.



WN-PRG01A LEDs



WN-PRG04A LEDs

WN-PRG08A LEDs

2. Getting Started

Guided Tutorial

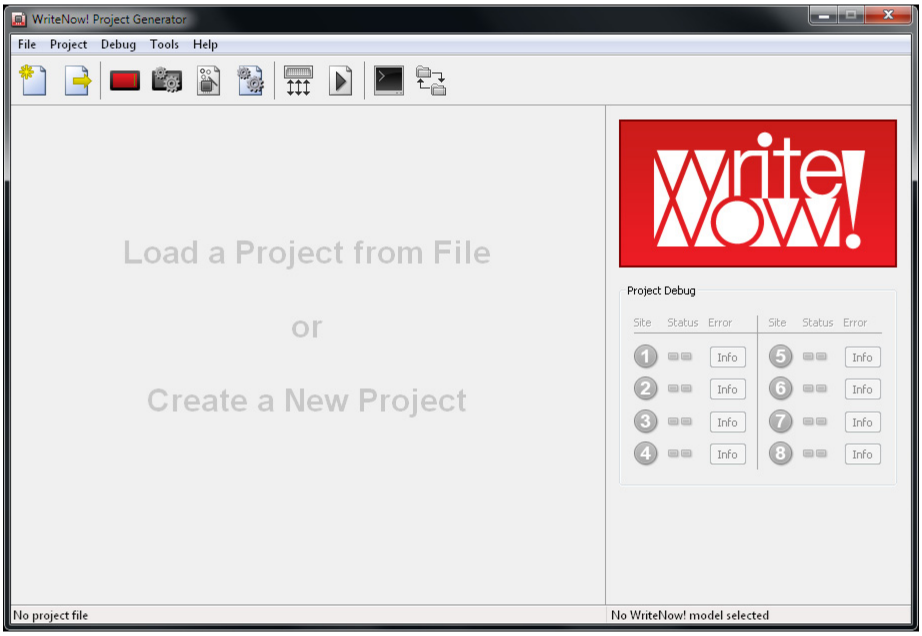
The following tutorial will guide you through the steps required to set up your WriteNow! programmer and create your first programming project.

1. Install Software

Insert the Setup CD into your PC and install the WriteNow! software.

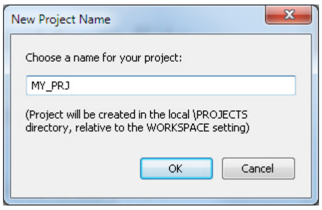
2. Launch the Project Generator

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



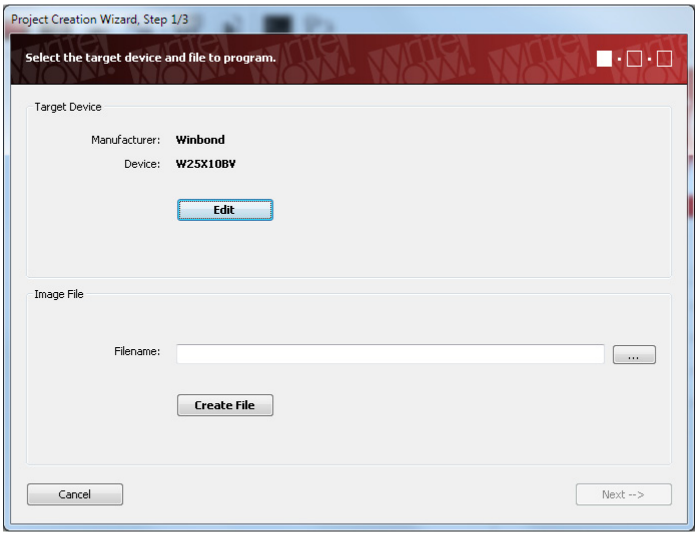
3. Create a New Project

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

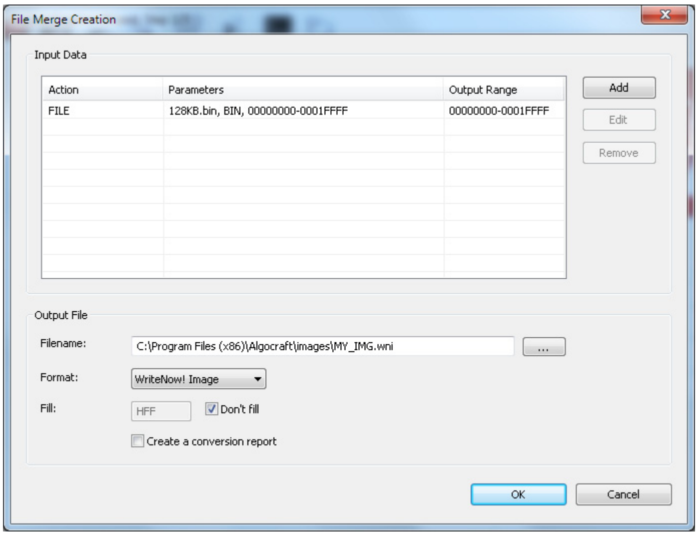


4. Create a New Project, Step 1 of 3

In the first Wizard step, specify the target device, by clicking the **"Edit"** button.



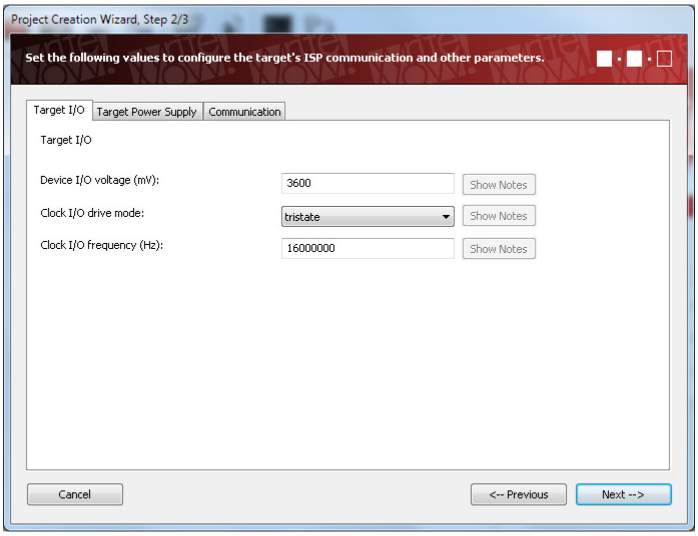
Next, specify the file to be programmed (image file). To create an image file, click the **"Create File"** button. A dedicated window will open.



Use the **"Add"** button to compose the data that will compose 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.

5. Create a New Project, Step 2 of 3

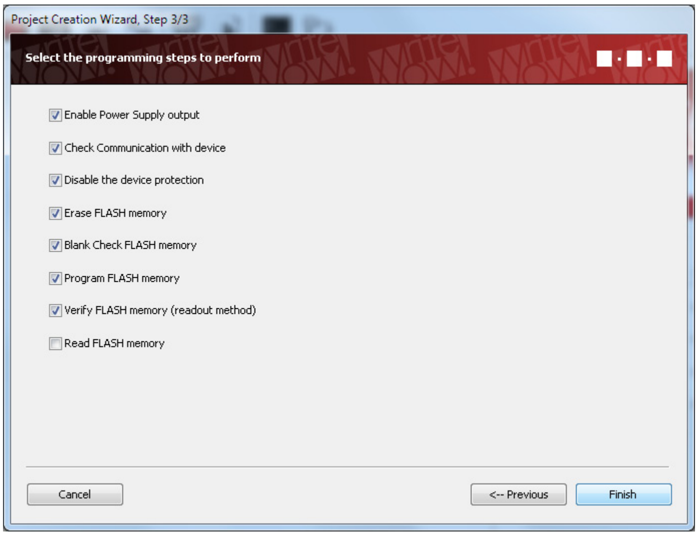
In this step, specify target parameters and connection values. The Wizard will automatically fill all data with typical values for the selected target device.



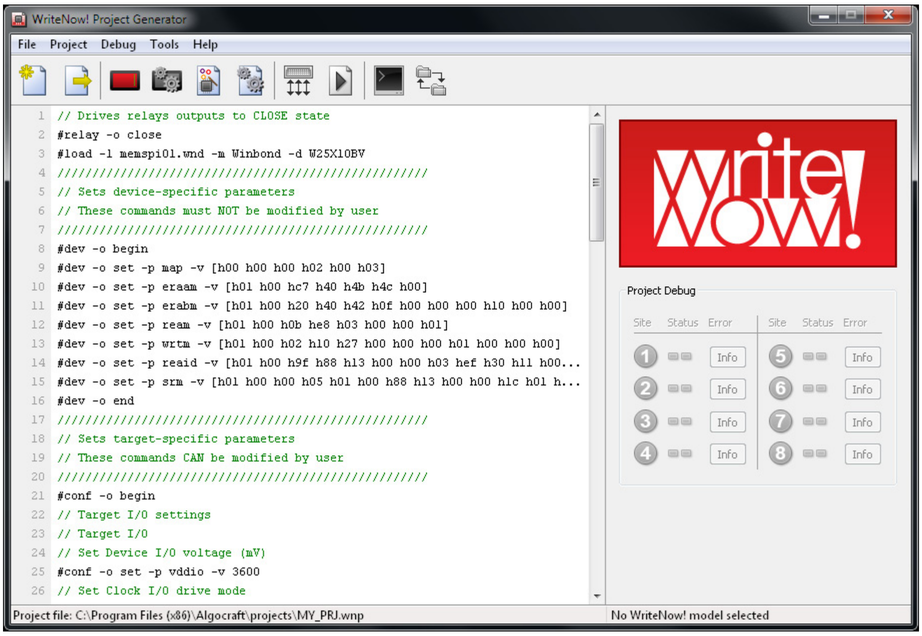
After carefully checking all of the parameters values, proceed to Step 3.

6. Create a New Project, Step 3 of 3

In this step you select which programming operation to perform on the target.

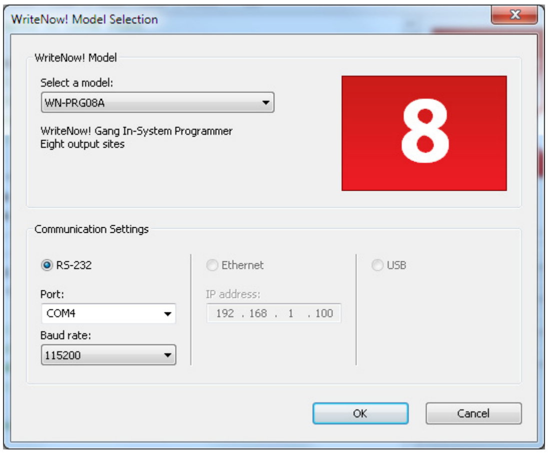


Click **"Finish"** to end the Wizard. At this point, a WriteNow! Programming Project will be created in the **\Projects** directory, relative to the Project Generator application location.



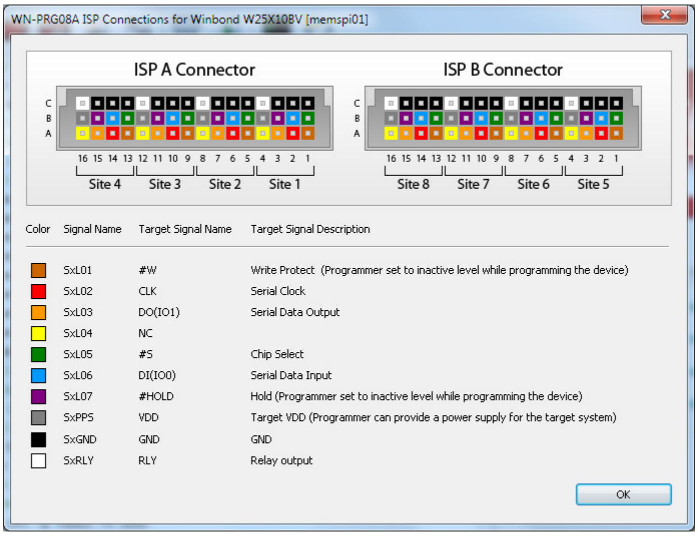
7. Configure your WriteNow! Instrument

Choose **Project > Select WriteNow! Model**, and specify your WriteNow! model and communication settings with the PC. Currently, WriteNow! can be connected only through a serial port. WriteNow! communicates at **115,200** bps by default. LAN and USB connections will be supported soon through a free software upgrade.



8. 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 **Debug > Show ISP Connections**.

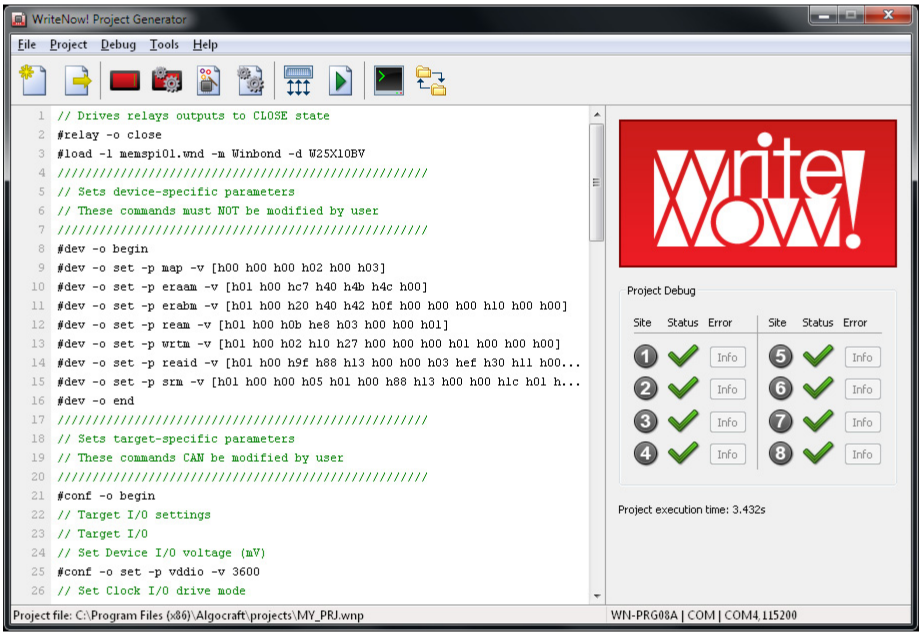


9. Startup WriteNow!

Connect WriteNow! to your PC through the provided serial cable. Finally, power up WriteNow! using the provided power supply.

10. Program the Target Device

Select **Debug > 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 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.

Manual Project Editing

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

The generated project file is a 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.

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" on page 19);
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" on page 31);
3. By building your own PC software that interfaces to the instrument (for more information, see "" on page 35).

3. Commands

Overview

WriteNow! is a slave unit and is always awaiting for a new command incoming 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**). 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 answers 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 is answered.

All commands and answers are case-insensitive.

Command Syntax

A WriteNow! command begins with the SOF character (**#**), followed by the command name, followed by zero or more command switches, and ends 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 take 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**).

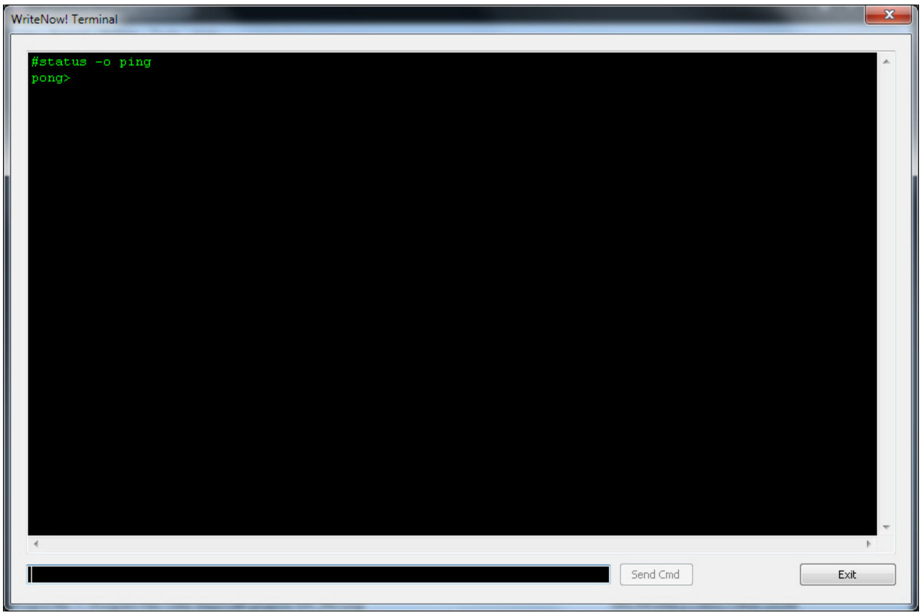
This is an example of a WriteNow! BUSY answer:

`*{\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>
#data –o set –c <direction> -t volatile
```

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
>
```

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
>
```

Execution Command

Syntax

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

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>
```

Parameters

`<directory>`	Full path of a directory.
`<filename>`	Full path of a filename.

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
>
```

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>
```

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 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 set -p br -d <baud rate>
#sys -o get -p br
#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>
```

Parameters

`<baud rate>`	`9600`, `19200`, `38400`, `57600`, `115200`, or `230400`.
`<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

Sets a new serial baud rate:

```
#sys -o set -p br -d 115200
>
```

Retrieves the instrument’s serial number:

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

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

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

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, 2010, at noon:

```
#time -o set -p date -d 2010/02/01
>
#time -o set -p time -d 12:00:00
>
```

Retrieves the instrument's date and time:

```
#time -o get -p date
2010/02/01>
#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]>
```

4. 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.

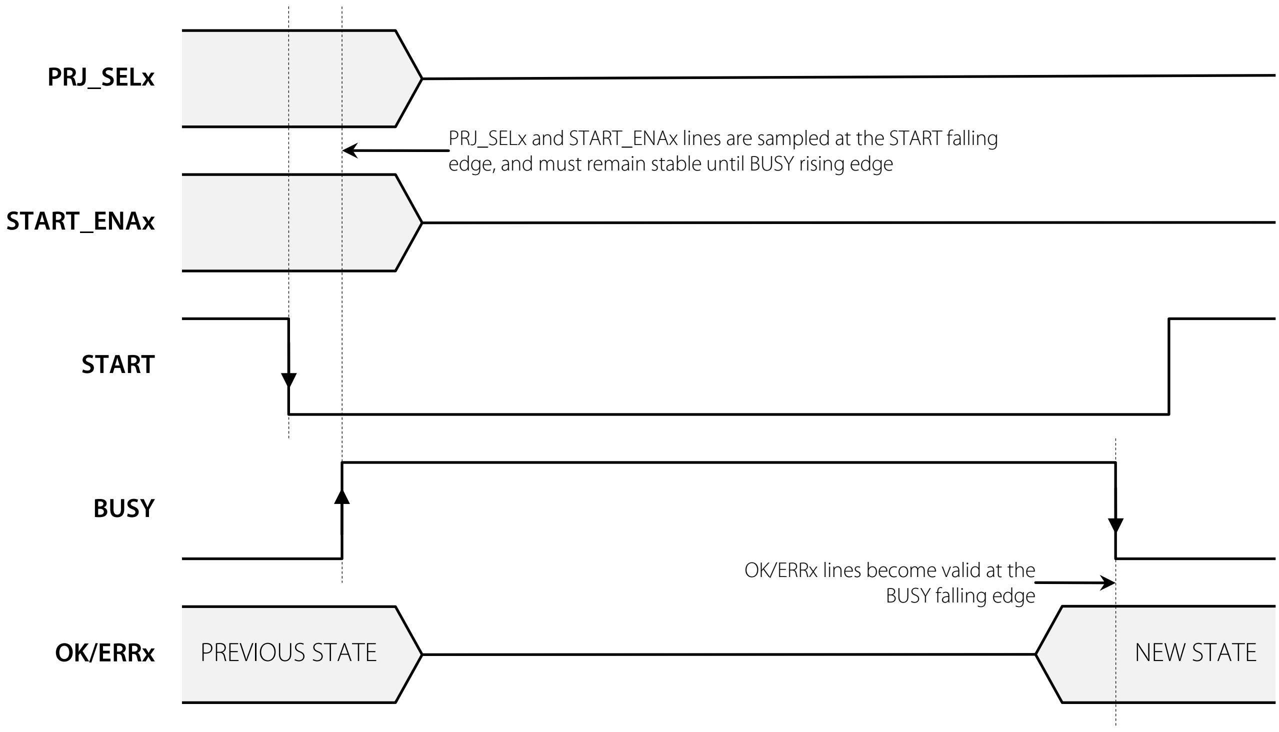
Signals

Signals needed to control the instrument in standalone mode are located in the "Low-Level Interface" connector (see "Connectors" on page 51 for the connector pinout on the various WriteNow! models) and are explained below.

Signal level is 0-5V. All lines are isolated (referenced to GNDI).

PRJ_SELx lines (input):	Define which project to execute (see "Project Assignment" later on this chapter).
START_ENAx lines (input):	Select which programming site(s) to enable. Active low.
START line (input):	Executes the project specified by PRJ_SELx lines on the programming site(s) enabled by START_ENAx lines. Active low.
BUSY line (output):	Indicates that a project is being executed. Active high.
OK/ERRx lines (output):	Valid at the end of project execution (when BUSY is low). Indicate, for each programming site(s), the success state of the programming project. (OK = high, ERR = low).

The following diagram illustrates the timing for the Low-Level Interface signals.

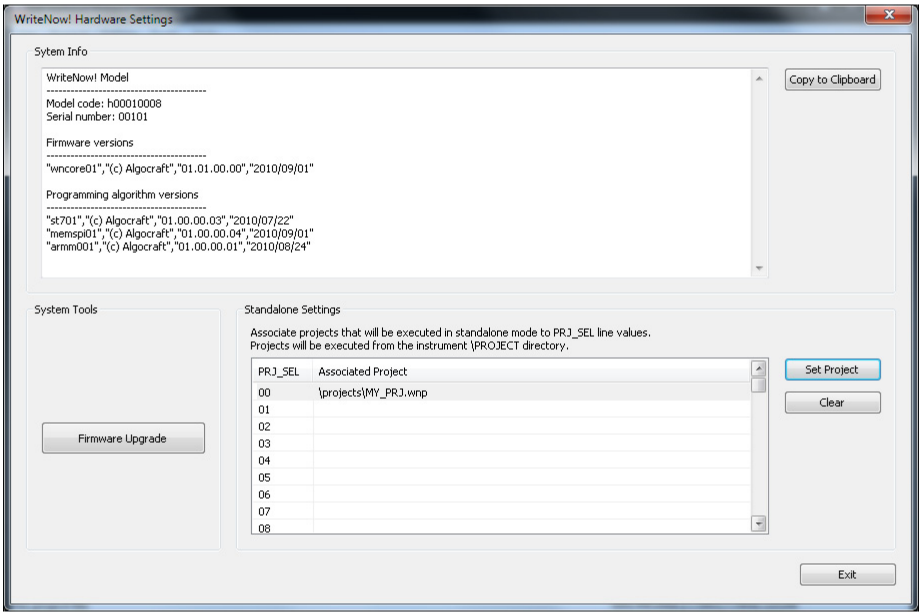


Low-Level Interface Signals Timing

Project Assignment

Before working in standalone mode, you must associate PRJ_SELx lines to a Project filename to execute.

To do so, in the WriteNow! Project Generator application select **Project** > **Hardware Settings**. In the window that will appear, associate PRJ_SEL values to project names by clicking the **"Set Project"** button for each PRJ_SEL configuration you wish you setup.


WriteNow! Hardware Settings
Sytem Info
WriteNow! Model
Model code: h00010008
Serial number: 00101
Firmware versions
"wncore01","(c) Algocraft","01.01.00.00","2010/09/01"
Programming algorithm versions
"st701","(c) Algocraft","01.00.00.03","2010/07/22"
"memspi01","(c) Algocraft","01.00.00.04","2010/09/01"
"armm001","(c) Algocraft","01.00.00.01","2010/08/24"
Copy to Clipboard
System Tools
Firmware Upgrade
Standalone Settings
Associate projects that will be executed in standalone mode to PRJ_SEL line values.
Projects will be executed from the instrument \PROJECT directory.
PRJ_SEL
Associated Project
00
\projects\MY_PRJ.wnp
01
02
03
04
05
06
07
08
Set Project
Clear
Exit

5. 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 (located in the **\Developer** folder, relative to the WriteNow! software installation path), which allow you to set up and control the programmer.

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;
- 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 program flow for interfacing with WriteNow! is the following:

1. Open communication
(**`WN_OpenCommPort()`** function)

↓

2. Execute commands
(**`WN_ExeCommand()`** function)

↓

3. Transfer files to/from the instrument's internal memory
(**`WN_SendFile()`** and **`WN_ReceiveFile()`** functions)

↓

4. Close communication
(**`WN_CloseCommPort()`** function)

Function Reference

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.

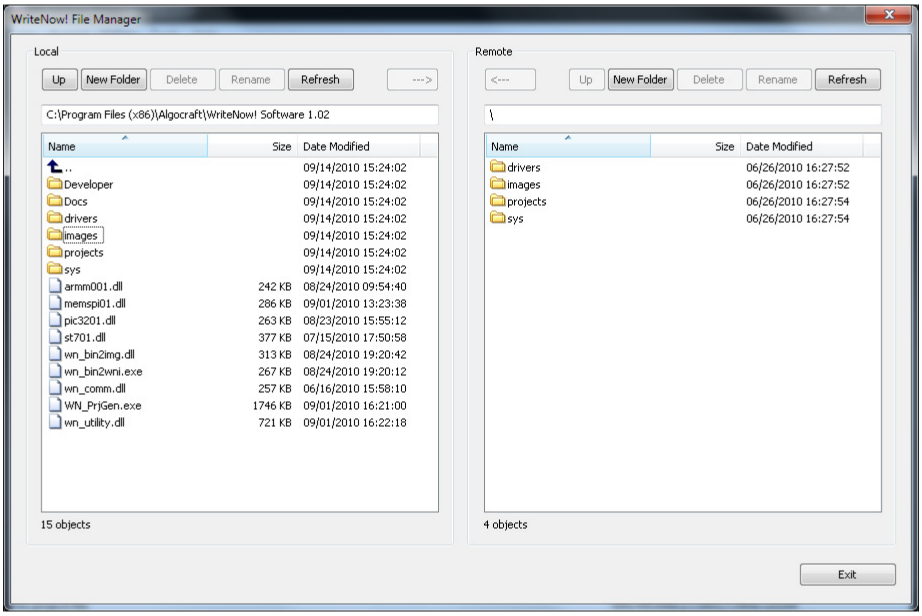
Parameters

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"**); Empty string for **"USB"** port.

6. 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.



File System Structure

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

- **\DRIVERS folder:** contains programming algorithms (.drv 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 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.

7. Variable Data Programming

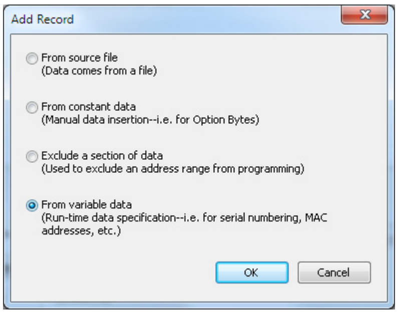
Overview

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.

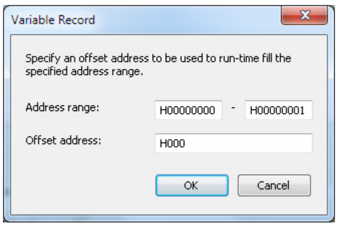
Usage

To implement variable data programming:

1. Use the Project Creation wizard of the Project Generator application to create your programming project. When creating the WriteNow! Image file, add a variable data record to the output file, as shown below.



2. You will then be asked for the target device address range to be programmed and the offset of the memory bank that will contain the variable data.



3. Proceed to the end of the Project Creation wizard. Your programming project is now ready to accept variable data.

4. 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 (for more information, see "Volatile Memory Commands" on page 29).

   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" on page 22).

8. Power and Relay Options

Power Supply Options

WriteNow! can be powered in two ways:

1. With the provided power supply (which supplies 15V DC);
2. By providing a power supply to the PWR pin of the Low-Level Interface connector (see "Low-Level Interface Connector" on page 52).

> WriteNow! accepts a DC power voltage between 12V and 25V. Please note, however, that the SxPPS line on each programming site, if used, can provide a maximum voltage of power voltage minus about 2V.

Relays

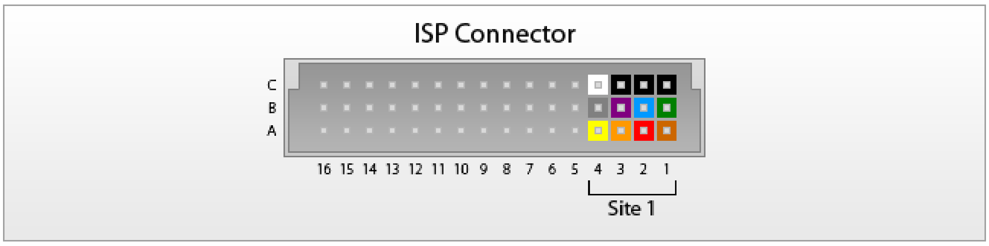
On the single-site WriteNow! model (WN-PRG01A), a relay barrier is provided on the ISP signals. When you create a programming project using the Project Generator application, relays are by default closed at the beginning of the project (with **`the #relay –o close`** command) and opened at the end (with **`the #relay –o open`** command).

On all WriteNow! models, a special signal (SxRLY) is present (on the "ISP" connector), on every programming site. If the programming site is enabled, this signal is driven to 0V when **`a #relay –o close`** command is executed, and driven to 5.5V when a **`the #relay –o open`** command is executed). This is useful for driving an external relay barrier.

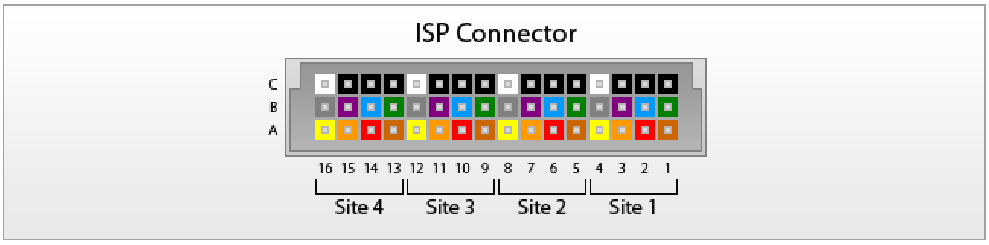
9. Connectors

ISP Connectors

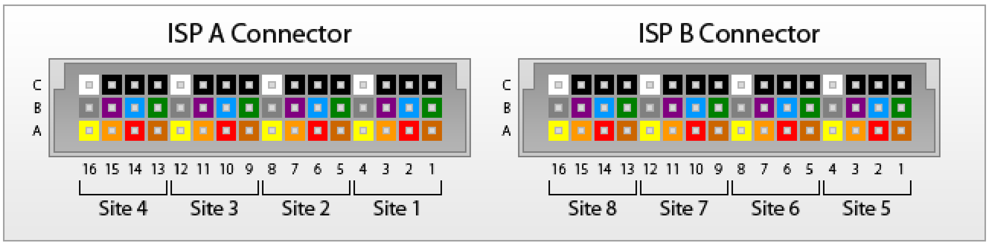
WriteNow! WN-PRG01A and WN-PRG04A models have one ISP connector; the WN-PRG08A model has two ISP connectors. Furthermore, in the WN-PRG01A model, the ISP connector also includes low-level interface signals.



WN-PRG01A ISP Connector



WN-PRG04A ISP Connector



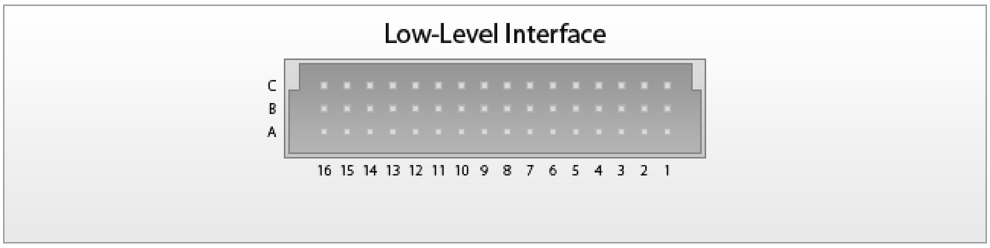
WN-PRG08A ISP Connectors

ISP Signal Definitions

Color	Signal	Description
(brown)	SxL01	Site x line 1
(red)	SxL02	Site x line 2
(orange)	SxL03	Site x line 3
(yellow)	SxL04	Site x line 4
(green)	SxL05	Site x line 5
(blue)	SxL06	Site x line 6
(purple)	SxL07	Site x line 7
(grey)	SxPPS	Site x programmable power supply
(white)	SxRLY	Site x relay output
(black)	SxGND	Site x GND

Low-Level Interface Connector

In the WN-PRG01A model, low-level interface signals are included in the ISP connector, which is called “ISP & LOW-LEVEL INTERFACE” connector.



Low-Level Interface Connector

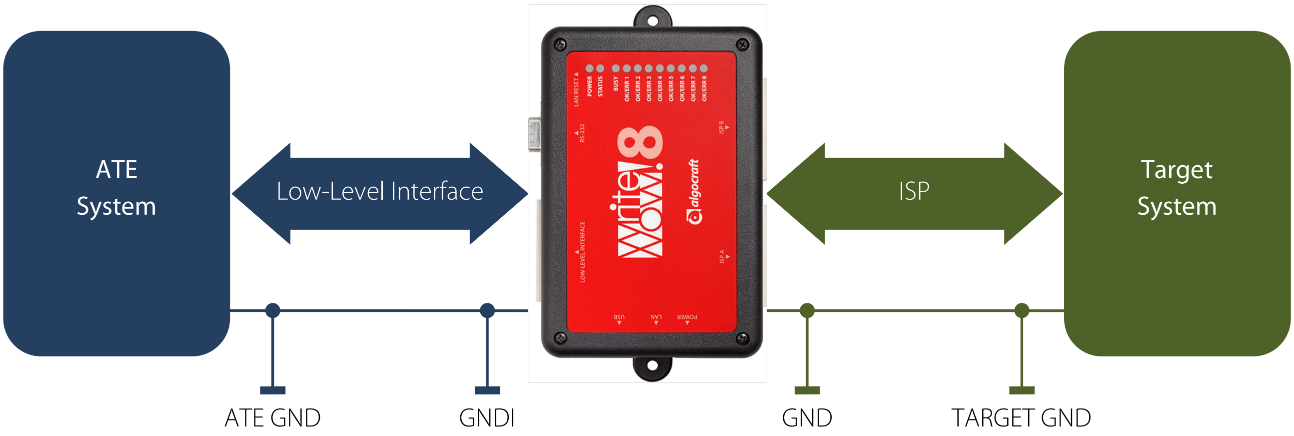
Low-Level Interface Signals

Signal	Description	WN-PRG01A Pin	WN-PRG04A Pin	WN-PRG08A Pin
PWR	Input Power Supply (12-25V)	A5/B5	A5/B5	A5/B5
GND	Power Supply Ground	C5	C5	C5
GNDI	Low-Level Interface Ground	A10/B12/C15/C16	A10/B12/C15/C16	A10/B12/C15/C16
TX_RS232	RS-232 TX (Output)	A16	A16	A16
RX_RS232	RS-232 RX (Input)	B16	B16	B16
PRJ_SEL0	Project Selector 0 (Input)	B10	B10	B10
PRJ_SEL1	Project Selector 1 (Input)	C10	C10	C10
PRJ_SEL2	Project Selector 2 (Input)	A11	A11	A11
PRJ_SEL3	Project Selector 3 (Input)	B11	B11	B11
PRJ_SEL4	Project Selector 4 (Input)	C11	C11	C11
PRJ_SEL5	Project Selector 5 (Input)	A12	A12	A12
START	Project Start (Input)	A7	A7	A7
START_ENA1	Site 1 Project Start Enable (Input)	-	B7	B7
START_ENA2	Site 2 Project Start Enable (Input)	-	C7	C7
START_ENA3	Site 3 Project Start Enable (Input)	-	A8	A8
START_ENA4	Site 4 Project Start Enable (Input)	-	B8	B8
START_ENA5	Site 5 Project Start Enable (Input)	-	-	C8
START_ENA6	Site 6 Project Start Enable (Input)	-	-	A9
START_ENA7	Site 7 Project Start Enable (Input)	-	-	B9
START_ENA8	Site 8 Project Start Enable (Input)	-	-	C9
BUSY	Busy (Output)	C12	C12	C12
OK/ERR1	Site 1 OK/ERR (Output)	A13	A13	A13
OK/ERR2	Site 2 OK/ERR (Output)	-	B13	B13
OK/ERR3	Site 3 OK/ERR (Output)	-	C13	C13
OK/ERR4	Site 4 OK/ERR (Output)	-	A14	A14
OK/ERR5	Site 5 OK/ERR (Output)	-	-	B14
OK/ERR6	Site 6 OK/ERR (Output)	-	-	C14
OK/ERR7	Site 7 OK/ERR (Output)	-	-	A15
OK/ERR8	Site 8 OK/ERR (Output)	-	-	B15

All low-level interface lines are isolated from system GND (and are referenced to GNDI), except for the PWR line, which is referenced to GND.

Ground Domains

The following diagram illustrates the two ground domains of the programmer.



ATE and Target Ground Domains

In order to avoid undesired current paths between the programmer and the target board, we suggest to use a power supply with a floating output (ground not referenced to the Earth potential).

10. Specifications

Feature	Value
Maximum Ratings	
Power supply voltage	30V
ISP SxL0[1..7] voltage	-0.7-6.5V
ISP SxL0[1..7] current	±60mA
ISP SxPPS voltage	-0.7-18V
ISP SxPPS current(*)	380mA
ISP SxRLY voltage	-1.0-30V
Low level interface PRJ_SELx, START, START_ENAx, BUSY, OK/ERRx voltage	-0.7-6.0V
Operating Ranges	
Power supply voltage	12-25V
ISP SxL0[1..7] voltage	0-5.5V
ISP SxPPS voltage	1.5-15V
ISP SxPPS current	300mA
ISP SxRLY voltage	0-28V
Low level interface PRJ_SELx, START, START_ENAx, BUSY, OK/ERRx voltage	0-5.0V
Physical and Environmental	
Operating conditions	0-40°C, 90% humidity max (without condensation)
Storage conditions	-10-60°C, 90% humidity max (without condensation)
EMC (EMI/EMS)	CE, FCC

(*) Current limited, recovers automatically after fault condition is removed.