Software Utility - NPA Terminal Utility

Table of contents

Title Page

COPYRIGHT

Revision History

Date	Revision	Firmware Compatibility	Changes Made
March 2020	2.0	All PMX & CMD	New Document for Software v. 2.0
September	2.1	All PMX & CMD	Added new Terminal display modes, additional delay commands, ASCII

2020			programming (AppProgram).
November 2020	2.2	All PMX & CMD	Update for Version 2.2 of software. Added Serial support, and AutoControl scripts.
January 2021	2.2a	All PMX & CMD	Changes for following version of software.
December 2021	2.43	All PMX & CMD	Update for Version 2.43 of the software. Installed portable version of the program. Added bit comparing, loops, AutoControl scripts.
October 2023	2.49	All PMX & CMD	Update for Version 2.49 of the software. Added ShellApps, ResizeApp, LoopOnFail.
May 2025			Update formatting to improve displaying of comments in example code on web page. Changed the footer.
			Update Bit-mask use -1 to clear Bit-mask

Last Build: 8/21/2026 11:15 AM

Cautions

The specifications of this software may be changed to improve performance or quality without prior notice.

Although this manual was produced with the utmost care, if you find any points that are unclear, wrong, or have inadequate descriptions, please let us know.

- If you find an error, the best way to do this is by clicking the Page Suggestions or Feedback? at the bottom of the page where you believe there is an issue. This will send NPA an e-mail with the link to the page, and you can describe the issue that you found and we can correct it quickly.

This guide is delivered subject to the following conditions and restrictions:

- The text and graphics included in this manual are for the purpose of illustration and reference only. The specifications on which they are based are subject to change without notice.
- Information in this document is subject to change without notice.

Nippon Pulse Terminal Program Overview

The Nippon Pulse Terminal Program is designed to help the user quickly configure, monitor, and develop programming for the Commander (CMD), and Performax (PMX) product lines.

Key features:

- Can quickly be connected or disconnected to any Commander or Performax controllers connected to the computer via USB or RS485, without needing to stop and restart the software.
- Any valid ASCII commands can be sent directly to the connected controller, and the response received is directly displayed.
    - The round-trip time is displayed for each command, from time sent to its reply being received.
- Any valid string of commands can be sent.
- ASCII commands or strings of commands can be attached to function keys.
- Multi-language support.
- Basic ASCII program support.
    - Pass/fail test for individual commands possible with ASCII program.

- Advanced auto control of ASCII programs.
    - Pass/fail system test with logging is possible with auto control.

Help with connecting a Commander or Performax controller

Help with buttons/functions on the Main Screen

Help with ASCII programs

Help with Auto Control

Help with Language Editor

Main Screen

The Nippon Pulse Terminal utility software is available in two versions.
RunNPTerminal.exe is the portable version
NPTerminal.exe is the version that must be installed with an installer app

The only difference between the two is where thefiles are stored. Everything else in the manual applies to both versions.
The Nippon Pulse Terminal utility software is available for downloaded from these links

> Portable V2.49-P 

> Installed V2.49 

The Nippon Pulse Terminal utility software has only one screen.
The background of the main screen will change colors based on the mode the Nippon Pulse Terminal is running in:

Gray: Normal Terminal mode

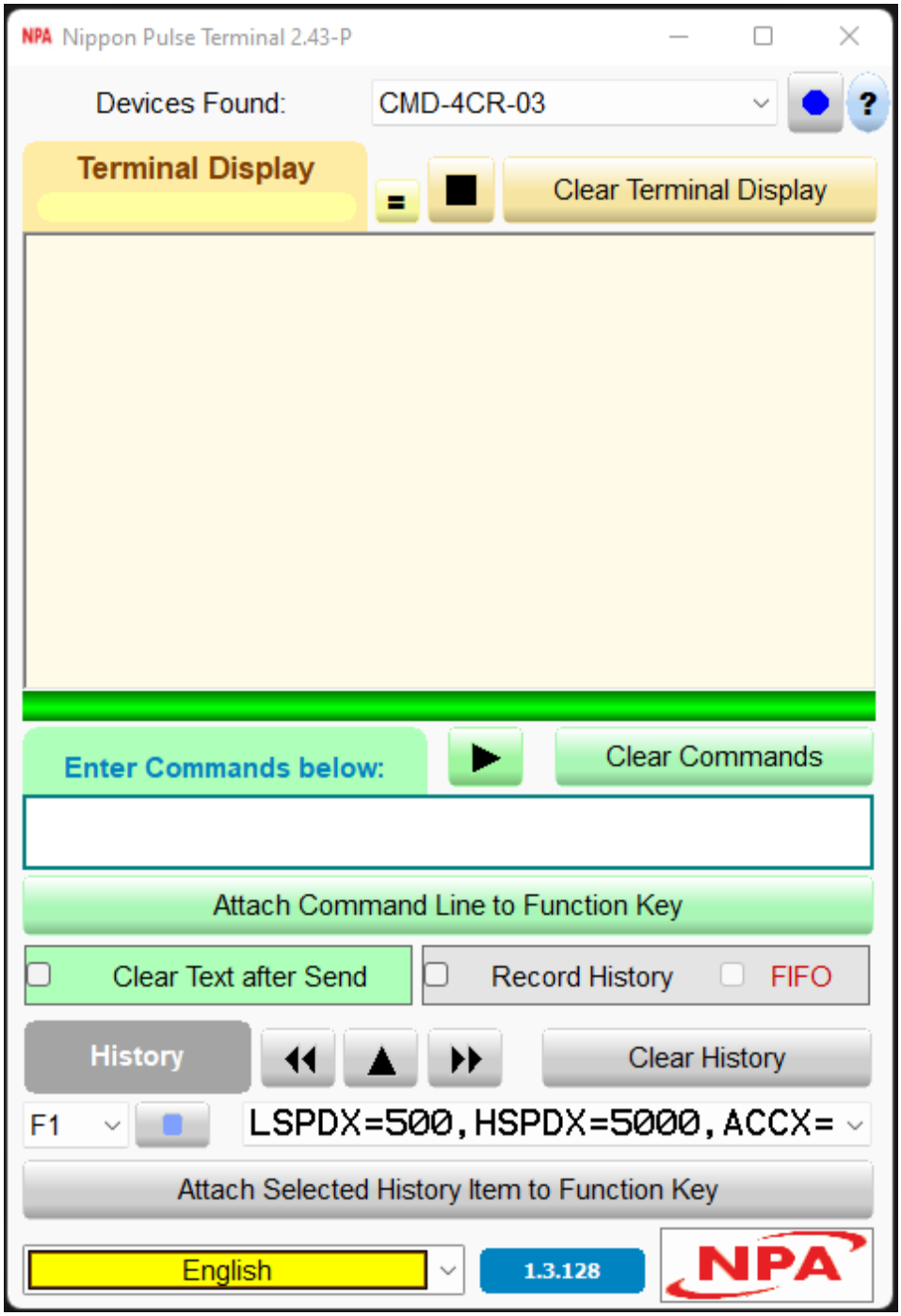
Pink: Sending ASCII commands or loading/parsing of the ASCII program

Orange: Running ASCII Program

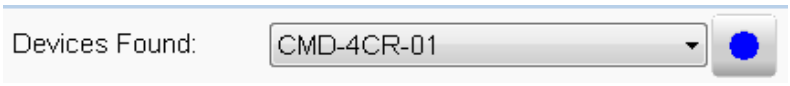
Yellow: Running Auto Control Script

Normal Terminal mode

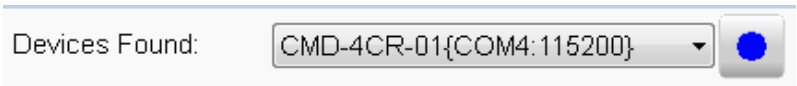
This screen is designed to give easy access to all the functions of the terminal program.
Click on a button or feature in the image below to learn more about it.


NPA Nippon Pulse Terminal 2.43-P
Devices Found:
CMD-4CR-03
?
Terminal Display
Clear Terminal Display
Enter Commands below:
Clear Commands
Attach Command Line to Function Key
Clear Text after Send
Record History
FIFO
History
Clear History
F1
LSPDX=500,HSPDX=5000,ACCX=
Attach Selected History Item to Function Key
English
1.3.128
NPA

Device selection



USB connected



Serial configured device

Device selection

- When started, the Nippon Pulse Terminal program will scan for all available USB-connected Commander and Performax controllers.
- These devices are listed, along with all configured serial devices, in a drop-down box that can be used to select the connected controller that commands should be sent to, and from which responses are received.
- The device can be changed any time a command is sent.

Main Screen

Refresh USB connected devices

Pressing the  (refresh) button will refresh the USB-connected device list.

For more information on DLLs and USB communication, see USB Communication.

Main Screen

Configure new serial devices/connections

For more information on adding, editing or removing a serial device, see Serial Communication.

Language selection

Language select (Yellow)

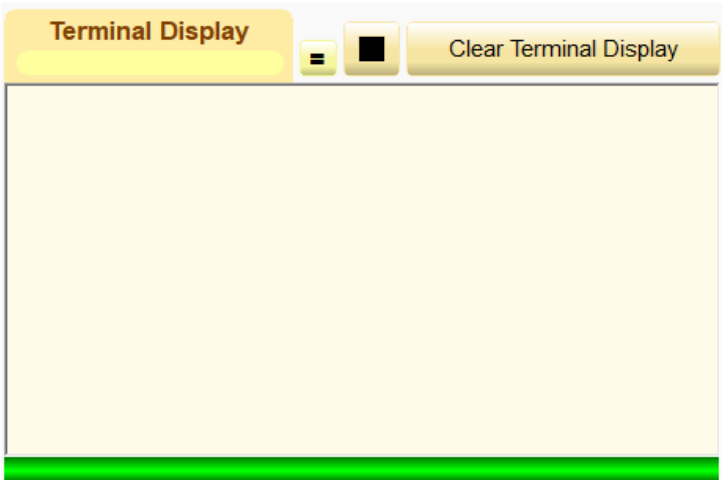
The Nippon Pulse Terminal utility software can support several different languages. The language that is displayed can be selected with this drop-down box.

DLL Version (Blue)

This box shows the version of the DLL installed and used by the Nippon Pulse Terminal utility software

Main Screen

Terminal Display area



Terminal Display area (Gold)

Underneath the text "Terminal Display" there is a small info box, which displays the current number of bytes of text stored in the Terminal Display box. (This is helpful when you have a long program or AutoControl script.)

The terminal display will echo all commands sent to the controller.

```
>> PX=0
     Reply: OK (1.8ms)
```

- The reply received from the controller will appear below the echoed command following the "Reply:" prompt.
- The time elapsed between the command being sent and the reply being received is then displayed in parentheses.

When running an ASCII program, the terminal will display an additional return when the reply received does not match the expected reply. No additional information is returned when the reply matches what is expected.

```
 >> PU=0
     Reply: OK (2ms)
!! Expected: DUMMY
 >> PX=0
     Reply: OK (1.8ms)
```

- For a Write command, the typical response will be OK.
- If the command cannot be executed, an error message will appear, preceded by a "?"
- For a Read command, the value requested will be returned based on the command syntax.
- If the command is not executable, an error message will be returned.

The controls for the Terminal Display are shown below:

Main Screen

Save

The text in the terminal display can be saved to a text file by pressing the  (save) button. By default, it

will save as a text file in the [.../TermDisplay] folder. You can name and save the file to any location.

Main Screen

Terminal Display mode

The button with the two horizontal stacked lines in it (next to the words Terminal Display) is used to select the mode that will be used to display text in the Terminal Display window.

The first three can be used for normal terminal mode or when running an ASCII program.

When the lines are horizontal this means the Terminal Display box displays a command, its reply and time elapsed. They will be on separate lines like this:

```
 >> PX=0
      Reply: OK (1.8ms)
 >> PY=0
      Reply: OK (2.3ms)
 >> PZ=0
      Reply: OK (1.9ms)
 >> PU=0
      Reply: OK (2.4ms)
```

When running an ASCII program, if the reply does not match the expected value, the reply looks like:

```
 >> PX=0
      Reply: OK (2.2ms)
!! Expected: DUMMY
 >> PY=0
      Reply: OK (1.9ms)
!! Expected: DUMMY
```

Main Screen

If the two lines are vertical, the Terminal Display box will display a command, its reply and time elapsed on the same line, like this:

```
 >> PX=0     Reply: OK (2.5ms)
 >> PY=0     Reply: OK (1.9ms)
 >> PZ=0     Reply: OK (1.8ms)
 >> PU=0     Reply: OK (1.7ms)
```

When running an ASCII program, if the reply does not match the expected value, the reply looks like:

```
 >> PX=0     Reply: OK (2ms)
!!        Expected: DUMMY
 >> PY=0     Reply: OK (1.6ms)
!!        Expected: DUMMY
```

Main Screen

When the button displays two dots, the Terminal Display box will display a command, its reply and time elapsed on the same line, like this:

```
! PX=0  (OK)  [DUMMY]  (1.8ms)
! PY=0  (OK)  [DUMMY]  (1.6ms)
! PZ=0  (OK)  [DUMMY]  (1.8ms)
! PU=0  (OK)  [DUMMY]  (2.3ms)
  PX=0  (OK)  (2.3ms)
  PY=0  (OK)  (1.9ms)
  PZ=0  (OK)  (2.3ms)
  PU=0  (OK)  (2.2ms)
```

Replies inside parenthesis ( ) are the actual reply.
When running an ASCII program, if the reply does not match the expected value, the expected value is enclosed in square brackets [ ].

Main Screen

The next two modes are used for ASCII program use.

When the button displays an exclamation point, the Terminal Display box will only display commands when the reply does not match the expected value.
The displayed line is:
! Program line number: Reply received [expected reply] (time elapsed)
Example:

```
! 000002: OK [DUMMY]  (1.9ms)
! 000003: OK [DUMMY]  (1.9ms)
! 000004: OK [DUMMY]  (2.4ms)
! 000005: OK [DUMMY]  (2.5ms)
```

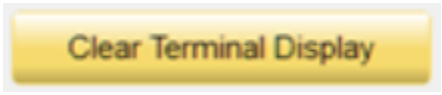
Main Screen

When the button displays an exclamation point with an apostrophe, the Terminal Display box will display program comment lines, and only display commands when the reply does not match expected value.
The displayed line is:
! Program line number: Reply received [expected reply] (time elapsed)

Example:

```
  <wrong reply to demonstrate how app flags it>
! 000002: OK [DUMMY] (2.1ms)
! 000003: OK [DUMMY] (2.5ms)
! 000004: OK [DUMMY] (2.5ms)
! 000005: OK [DUMMY] (2.4ms)
  <correct commands from here on>
```

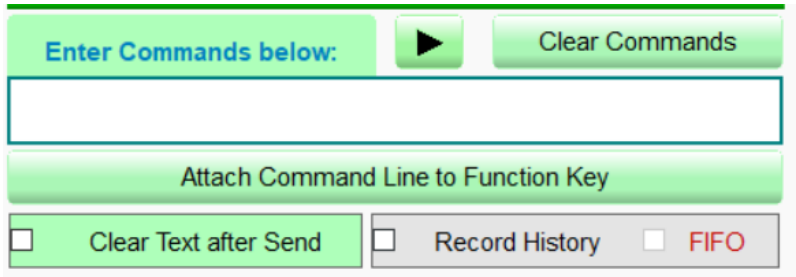
Main Screen



Clear Terminal Display – The terminal display can be cleared by selecting the Clear Terminal Display button.

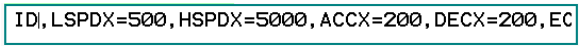
Main Screen

Command Entry



Command Entry (Green)
ASCII commands can be sent directly to the controller from the Command input box.

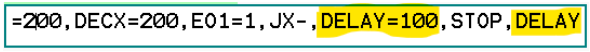
ASCII commands can be entered one at a time,



or ASCII commands can be entered as a string of commands. For a string, each command must be separated by a comma.

- Pressing a function key will enter the linked command into the Command input box.
- Pressing one of the insert keys will insert the selected command into the Command input box.
- See the Command Reference Manual for a full list of ASCII commands that are valid for the controller you are using.

Main Screen



Additionally, the Terminal program can accept the "DELAY" command. When a delay is sent, it only delays the following command from being sent to the controller. The delay commands are shown below:
DELAY
a 2000 ms (2 seconds) delay

DELAYS
delay short interval of 500 ms (1/2 second)
DELAYM
delay medium interval of 1500 ms (1.5 seconds)
DELAYL
delay long interval of 3000 ms (3 seconds)
DELAY=####
set delay to any value you want (where #### is the number of ms)

Main Screen

Send command
The ASCII commands entered in the Command input box will be sent after 1) the Enter key is pressed, or 2) the  button is pressed.

Main Screen

Clear Command input box
The text in the Command input box can be cleared by selecting the Clear Commands button.

If the Clear Text after Send checkbox is checked, the Command input box will be cleared automatically after the command is sent.

If not checked, the last command will remain in the Command input box after it is sent.

Main Screen

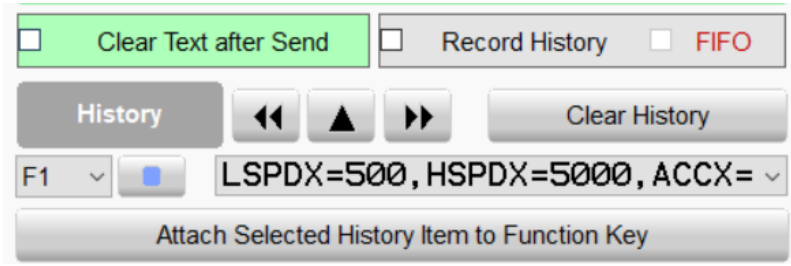
Delay progress bar
This progress bar shows how much more time is left until the next command is sent.

Main Screen

Attach to Function Key
Any ASCII command or string of commands can be attached to a function key. This is done by selecting the Attach Command Line to Function Key button. See Function Key in the History section for information on how to select the function key to which the ASCII command is attached.

Main Screen

History section



History section (Gray)
All functions related to recording and recalling the commands that have been sent to the controller are in this section.

Function Key
This drop-down is used to select the Function that you want to attach a command to. You can then select the Attach Command Line to Function Key button or select the Attach Selected History Item to Function Key button to attach a command.

Main Screen

Function Key List
Selecting the  button will display a list of all function keys that have a command attached to them. If no command is attached, the function key will not be displayed on the list.

Main Screen

Attach History Item to Function Key
Any ASCII command or string of commands can be attached to a function key. This is done by selecting the Attach Selected History Item to Function Key button.

Main Screen

Record History
If the Record History checkbox is checked, when an ASCII command is sent from the Command input box it will be added to the beginning of the History drop-down box.

FIFO
If the FIFO checkbox is also checked, when an ASCII command is sent from the Command input box it will be added to the end of the History drop-down box.

History List
This drop-down box is a list of commands you have chosen to record.

Clear History
By selecting the Clear History button, all information in the History list is deleted.

Insert buttons
These buttons will insert the selected command from the history list into the Command input box.

 – Inserts the command into the beginning of the Command input box.

 – Inserts the command into the cursor's location in the Command input box.

 – Inserts the command at the end of the Command input box.

ASCII programs

The Nippon Pulse Terminal program has the ability to run a list of ASCII commands and compare the reply to an expected reply. This feature supports all ASCII commands for the connected controller, as well as a few additional commands.

This allows for easy testing of command sequences along with the replies that are expected.

- If the actual reply matches the expected reply, the command passes.
- If the actual reply does not match the expected reply, the command fails.
- The actual reply must match exactly (including case) the expected result in order to pass. Anything else is a fail.
    - Numeric replies have additional options for compare. See Reply compare

Passed commands are displayed normally.
Failed commands are marked with an exclamation point, and the received and expected results are shown.
See Terminal Display area for more information and examples of the pass displays.

ASCII programs are stored in the [.../AppPrograms] sub folder.

The ASCII programs are designed to work with ASCII character languages. Please note that while it is possible to use Unicode character languages in the comments, it is not recommended as it has not been fully tested, and may cause unexpected results in the code execution.

Main Screen

Other Supported commands

The ASCII program function supports all ASCII commands for the connected controller, as well as a few additional commands/functions as shown below.

Command / Function	Description
Comments	Allows adding comments to the program
Delay	Sets the amount of time that the program waits before sending the next ASCII command
Reply Compare	Select how a numeric reply is compared

Repeat/Until Loop	A loop which can repeat until a defined condition is met

Return to ASCII programs

Comments

The program file can also contain lines that are comments. Such lines simply start with the single quote (') character followed by a space like this:

```
' wrong reply to demonstrate how app flags it
PX=0,DUMMY
PY=0,DUMMY
PZ=0,DUMMY
PU=0,DUMMY
```

The comments are designed to work with ASCII character languages. Please note that while it is possible to use Unicode character languages in the comments, it is not recommended as it has not been fully tested, and may cause unexpected results in the code execution.

Return to ASCII programs

Delay

The Terminal program can accept the "DELAY" command.

- When a delay is sent, it only delays the next command from being sent to the controller.

The delay commands are shown below:

DELAY
will cause a 2000 ms (2 seconds) delay

DELAYS
will cause a short interval delay of 500 ms (1/2 second)

DELAYM
will cause a medium interval delay of 1500 ms (1.5 seconds)

DELAYL
will cause a long interval delay of 3000 ms (3 seconds)

DELAY=####
will set a delay to any value you want (where #### is the number of ms)

When entering a "DELAY" command, it can be entered with or without a comma. If using a comma, only CR/LF should be entered.

- Acceptable entry:

DELAY

or

DELAY,

Return to ASCII programs

Reply compare

This function defines how the actual reply is compared to the expected reply to define a pass or fail.

- By default, the actual reply must match exactly (including case) the expected result in order to pass. Anything else is a fail.

Let's say the expected reply rather than being the number 1000, is to be in a range of values. You can define that range in many different ways:

Less than/equal to:
<=1000

Greater than/equal to:
>=1000

Range
*=1000|2000

Equal to expected value and force a math comparison
==1000

ANDed with current Bit Mask (BTMASK command) is equal to expected value
&=1000

- The comparison macro must always come first and is always 2 characters (>= , <=, *=, ==, =&). The range is always inclusive. There is no <, just <=

The first two macros are obvious. The third one means a range. So:

*=1000|2000

means:

>= 1000 and <= 2000

So values between 1000 and 2000 (inclusive of the 1000 and 2000) are valid reply values.

- When a math comparison is done (any above) the values are tested to see if they are using either BASIC or C notations for Binary or Hex numbers as well.
    - The app test for the following notations:
        - 0x for C++ Hexadecimal (just x can be used as well)
        - 0B for C++ Binary (just B can be used as well)
        - &H for Basic Hexadecimal
        - &B for Basic Binary

In case of multiple values replies

Some commands (PP, EP, etc) will return multiple values (i.e. for all axis) like this:

0:0:0:0

If you want to use the comparison macros, you can do this for all or any of the individual items like this:

=0|1:=0|1:0:0

The app test for the : character assumes multiple return values (i.e. one for each axis). It breaks up the reply and expected text into parts and then does the normal comparison for each item. If any value fails, the entire command is considered to have failed.

Return to ASCII programs

Repeat/Until Loop

The Nippon Pulse Terminal Program includes a Repeat/Until Loop macro, which can repeat until a defined condition is met.

These commands are micro scripts for use in the ASCII programs, and can also be used when typing in commands via the Command Entry, but you have to type in a complete Repeat/Until Loop code (start with REPEAT and end with UNTIL) all on one line separated by commas. ASCII programs loaded in from files are what it is primarily designed for though.

You can nest up to 3 levels deep of Repeat/Until Loops.

The testing loops are comprised of the following micro commands:

Command / Function	Description
BtMask	Sets the mask used for bit-wise operations
Repeat	Designates the start of the Repeat/Until Loop
Rsave	Designates a math comparison
Show	This converts and displays a reply from a command
Until	Designates the end of the Repeat/Until Loop, and defined condition to end loop

Here is an example of three REPEAT loops:

1) (nested counting) using just the default counter value for comparison:

```
REPEAT,
  PX=0,OK
  PY=0,OK
  PZ=0,OK
  PU=0,OK
  REPEAT,
    PX=0,OK
    PZ=0,OK
  UNTIL=3,
UNTIL=10,
```

The outside loop will loop 10 times. The inside loop will loop 3 times, but because the outside loop iterates 10 times it will actually run 30 times.

2) A Bit Mask comparison

```
Y6400,OK
BTMASK=7,
REPEAT,
```

```
  MSTY,
  RSAVE1,
UNTILM0,
```

The Y axis is commanded to position 6400. A bit mask for the first three bits is created (BTMASK=7). A loop is created that will call the MSTY command. The reply is saved to RSAVE1. The loop will continue until the first three bits of the reply are 0 (the motor is stopped).

3) A math comparison using a reply value would look like this:

```
JX-,OK
REPEAT,
  EP,
  RSAVE1,
UNTILR1000:2000,
STOP,OK
```

Assuming the EP values start at 0000:0000:0000:0000 the program above should loop until the first value (Encoder X) returned from EP is between 1000 and 2000 and then the loop ends and the STOP command is executed.

4) A comparison to a single value when multiple values are returned (PP, EP, BSTAT, etc) would look like this:

```
JZ-,OK
REPEAT,
  EP,
  RSAVE3,
UNTIL>1000,
STOP,OK
```

Assuming the EP values start at 0000:0000:0000:0000 the program above should loop until the third value (Encoder Z) returned from EP is greater then 1000 and then the loop ends and the STOP command is executed.

Return to ASCII programs

BtMask

BTMASK=#### (set current Bit Mask to new value)

- Sets the mask used for bit-wise operations. The syntax for Bit Mask value #### can be:
    - 32000 ' decimal
    - 0xFFFF ' hexadecimal (C syntax)
    - &HFFFF ' hexadecimal (Basic syntax)
    - 0b001010 ' Binary (C syntax)
    - &B001010 ' binary (Basic syntax)

RSAVE# commands also save the last reply value and can process any of the above numeric types.

To clear or reset the BTMASK value set BTMASK=-1

Return to Repeat/Until Loop

Repeat

REPEAT (starts a repeat cycle or loop)

- Designates the start of the Repeat/Until Loop, which repeats until a defined condition is met.

- Be default the repeat loop uses its own counter (kind of like a FOR NEXT loop) unless a math value is designated.

Return to Repeat/Until Loop

Rsave

RSAVE

To designate a math comparison use the commands:

RSAVE1
RSAVE2
RSAVE3
RSAVE4

to save the last reply value of the last command executed in a math comparison variable. The reason there are four save commands is if the reply value is like this:

1000:2000:3000:4000

which some commands can return (ie. PP, EP). You can save any one of the values based on the index. In this case:

RSAVE1 would save a value of 1000
RSAVE2 would save a value of 2000
RSAVE3 would save a value of 3000
RSAVE4 would save a value of 4000

Now when the UNTIL command is executed it will use this saved math value instead of the loop counter for the comparison.

The RSAVE# command will return a reply of what the actual value it stores in memory from the previous reply value.

RSAVE# commands also save the last reply value and can process any of the following numeric types.

```
32000        ' decimal
0xFFFF       ' hexadecimal  (C syntax)
&HFFFF       ' hexadecimal (Basic syntax)
0b001010     ' Binary (C syntax)
&B001010     ' binary (Basic syntax)
```

Return to Repeat/Until Loop

Show

SHOW##

This converts and displays a reply from a command. The ## sets the format the reply is returned as:

SHOWDC (displays last reply value of previous command as decimal)
SHOW32 (displays last reply value of previous command as 32 bit Binary)
SHOW16 (displays last reply value of previous command as 16 bit Binary)

SHOW8 (displays last reply value of previous command as 8 bit Binary)
SHOWHX (displays last reply value of previous command as 32 bit Hexadecimal)

alternate syntax for above SHOW commands:

SHOW32| ' breaks up Binary in multiple 8 bit blocks separated by | character
SHOW32|# ' breaks up Binary in multiple 8 bit blocks separated by | character and # (1-4) represents specific item if reply is ####:####:####:####

This is a different Show than the one used for Auto Control Script.

Return to Repeat/Until Loop

Until

UNTIL

Designates the end of the Repeat/Until Loop, and defined condition to end loop.

UNTIL= ### (ends loop or repeats it again based on comparison of equal)
UNTIL< ### (ends loop or repeats it again based on comparison of less than)
UNTIL> ### (ends loop or repeats it again based on comparison of greater than)
UNTILR ###:### (ends loop or repeats it again based on comparison of range values. Low value and high value separated by a colon)
UNTILM ### (ends loop when comparison value ANDed with Bit-Mask equal value)

The UNTILR command is inclusive, meaning the value is valid if not only between the two range values but also including the range values themselves.

Return to Repeat/Until Loop

File format

The ASCII program files must be in ASCII text and delineated with a comma.

> Please note that while it is possible to use Unicode character languages in the comments, it is not recommended as it has not been fully tested, and may cause unexpected results in the code execution.

- Each command should be on a separate line.
- The app expects either a CR/LF for end of line, or just a CR.
  - CR means Carriage Return, which is ASCII character 13.
  - LF means Linefeed, which is ASCII character 10.
  - Most text editors will append either a CR/LF or a CR at the end of each line automatically.
- Each program can contain up to 999,999 lines.
- Files can be created in a text editor of your choice, or created in a spreadsheet software and saved as a CSV (Comma Delimited) file.
  - It is recommend that program files be created using Notepad or any simple ASCII-based text editor. Exporting from a spreadsheet software like Excel (even as a .csv file) may work, but may include some non-visible embedded characters. If creating the file in a spreadsheet software, please review the CSV file in an ASCII-based text editor before running it in the Nippon Pulse Terminal program.

Return to ASCII programs

Program contents

The ASCII program can be used to run a series of commands only, or run a series of commands and compare the resulting reply to an expected reply.

If running a series of commands (without comparing) there will be only one column:

```
Command
```

which would look like this:

```
PX=0
PY=0
PZ=0
PU=0
```

If running a series of commands and comparing the reply to an expected reply, there are two columns:

```
Command, Expected Reply
```

which would look like this:

```
PX=0,OK
PY=0,OK
PZ=0,OK
PU=0,OK
```

The program file can also contain lines for comments. Such lines simply start with the single quote (') character followed by a space like this:

```
' wrong reply to demonstrate how app flags it
PX=0,DUMMY
PY=0,DUMMY
PZ=0,DUMMY
PU=0,DUMMY
```

Please note that while it is possible to use Unicode character languages in the comments, it is not recommended as it has not been fully tested, and may cause unexpected results in the code execution.

Saving location naming

The ASCII program files can be accessed in one of two ways:

- Quick load and auto program access
  - These ASCII programs are stored in the [.../AppPrograms] sub folder.
    - If you use a text editor or spreadsheet to write the ASCII programs and want to have them available for quick load or auto program access, the resulting files should have their file extensions changed from .csv or .txt to .prg
    - The ASCII programs must be saved with the following naming convention: (## is 01-10)
      - test##.prg
  - The sub-folder "AppPrograms" accepts up to 10 special program filenames to only be as follows:

test01.prg
test02.prg
test03.prg
test04.prg
test05.prg

test06.prg
test07.prg
test08.prg
test09.prg
test10.prg

- Open File dialog
  - Can access ASCII programs that are stored in any location.
  - The open file dialog will allow you to select a file with the following extensions:

.prg
.csv
.txt

Run ASCII program

The ASCII program files can be accessed in one of two ways:

Quick load and Auto program access

First, in the sub-folder "AppPrograms" there can be up to 10 special program files, to be named only as follows:

test01.prg
test02.prg
test03.prg
test04.prg
test05.prg
test06.prg
test07.prg
test08.prg
test09.prg
test10.prg

You can "quick load" them using the CTRL key and any function key from F1 to F10.

test01.prg	CTRL / F1
test02.prg	CTRL / F2
test03.prg	CTRL / F3
test04.prg	CTRL / F4
test05.prg	CTRL / F5
test06.prg	CTRL / F6
test07.prg	CTRL / F7
test08.prg	CTRL / F8
test09.prg	CTRL / F9
test10.prg	CTRL / F10

The program that matches the key press will automatically be loaded (it if exists, otherwise app will beep) and then run.

Custom named programs

In the sub-folder "AppPrograms" there can be unlimited custom-named programs that can run with the Auto Control [RunProg] command. This will define a custom program name, which would be defined as:

test99mycustomname.prg

You can also use the following syntax to define a "custom" program by adding a custom program name macro after the command:

[RUNPROG?]mycustomname

The custom program filename macro must be alpha-numeric characters only.

Open file dialog

Any program file with any name can also be accessed using the key command:

- CTRL-O (hold CTRL down and press O)

This will display the Open File dialog and you can select any file you want.
The open file dialog will allow you to select a file with the following extensions:

.prg
.csv
.txt

When an ASCII file is opened, the Main Screen turns Pink, the program being loaded appears in the Terminal Display area, and a blue progress bar is displayed to show the progress of the parsing of the ACSII program being loaded. See example below.

Example

In the example code, we use the following color coding to make the code easier to understand.

```
' Comments
ASCII_Command, Expected_Reply
```

- Unless otherwise noted all example code is written for the Commander series of controllers.
    - Please see the command reference for differences in the ASCII commands for the Performax series of controllers.
- The formatting of the ASCII program is the same for all controller series.
- For more information on any ASCII commands shown in the examples, please see the command reference.

Example #1 Example to demonstrate Pass/Fail flags, as well as the difference between delay commands.

Example #2 Example to demonstrate a simple product tester. Sets up the device (the input/output polarity, general purpose I/O configuration, and values for I/O). Enables the motors, moves them then homes the motors. If the motors trigger home input at zero, then unit passes.

Example #3 Example to demonstrate a simple product burn in tester. Sets up the device (the input/output polarity, general purpose I/O configuration, and values for I/O, as well as global motion control settings). This program then moves each of the three motors through their ranges of motion. The end point is the same as the start, allowing this script to be cycled indefinitely with an auto-control script.

Example 1

Example to demonstrate Pass/Fail flags, as well as the difference between delay commands.

```
' wrong reply to demonstrate how app flags it
PX=0,DUMMY
PY=0,DUMMY
PZ=0,DUMMY
PU=0,DUMMY

' correct commands from here on
PX=0,OK
PY=0,OK
PZ=0,OK
PU=0,OK
PP,0:0:0:0
HSPD=65000000,OK
ACC=10,OK
ABORT,OK
X1000Y1000Z1000U1000,OK

' DELAYS command is short delay of 500 ms
DELAYS,
PP,1000:1000:1000:1000
ARCXYP0:0:45000,OK
DELAYS,
PP,-1000:1000:1000:1000
X1000Y1000Z1000U1000,OK

' DELAYM command is medium delay of 1500 ms
DELAYM,
PP,1000:1000:1000:1000
ARCTP10000:0:180000:6000,OK
DELAYM,
PP,19055:0:6000:25614
X1000Y1000Z1000U1000,OK
```

```
' DELAYL command is long delay of 3000 ms
DELAYL,
PP,1000:1000:1000:1000
ARCZUP0:0:315000,OK
DELAYL,
PP,1000:1000:-1000:-1000
X1000Y1000Z1000U1000,OK
DELAY,
PP,1000:1000:1000:1000
ARCTN-18000:0:470000:-56832,OK

' DELAY command default is now 2000 ms
DELAY,
PP,-11493:17879:-56832:20402
```

Example 2

Example to demonstrate a simple product tester. Sets up the device (the input/output polarity, general purpose I/O configuration, and values for I/O). Enables the motors, moves them then homes the motors. If the motors trigger home input at zero, then unit passes.

```
' Product Tester
DELAYS        'Small startup delay to allow unit to stabilize

' Set test configuration
POLX=8,OK
POLY=8,OK
POLZ=8,OK
POLU=8,OK
IOCFG=4227858432,OK
IO=2147483647,2147483647
IOP=0,OK

' Enable motors
EO=7,OK
DELAYS

' Move motors away from Home and test that home input is off
X1000Y1000Z1000,OK
DELAYS
MSTX,0
MSTY,0
MSTZ,0

' Home motors
HX+0,OK
HY+0,OK
HZ+0,OK
DELAYS

' Move motors to 0 point from home then confirm the home input is on
X0Y0Z0,OK
DELAYS
MSTX,64
MSTY,64
MSTZ,64

' Test finished
```

Example 3

Example to demonstrate a simple product burn-in tester. Sets up the device (the input/output polarity, general purpose I/O configuration, and values for I/O, as well as global motion control settings). This program then moves each of the three motors through their ranges of motion. The end point is the same as the start, allowing this script to be cycled indefinitely with an auto-control script.

```
' Test Setup
EO=0,OK
ABS,OK
POLX=168,OK
POLY=168,OK
POLZ=168,OK
POLU=168,OK
IOCFG=4294967295,OK
IO=4294967295,4294967295
ACC=10,OK
DEC=10,OK
LSPD=100,OK
EO=7,OK
CLRX,OK
CLRY,OK
CLRZ,OK
X0,OK
Y0,OK
Z0,OK

' Move gripper to Test location 1
HSPD=32000,OK
DELAYS
Z32000,OK
BTMASK=7,
REPEAT,
    MSTZ,
    RSAVE1,
UNTILM0,

'Load test
DELAY=10000

' Gripper test
Z83200,OK
REPEAT,
    MSTZ,
    RSAVE1,
UNTILM0,
DELAYL
MSTU,0

' Buffer test
HSPD=32000,OK
DELAYS
X122048,OK
REPEAT,
    MSTX,
    RSAVE1,
UNTILM0,
DELAY=4000

' Low speed buffer crush test
HSPD=448,OK
```

```
DELAYS
X128576,OK
REPEAT,
    MSTZ,
    RSAVE1,
UNTILM0,
DELAY=21000

' Home Buffer motor
HSPD=32000,OK
DELAYS
HX-0,OK
DELAYM
IO20=0,OK
DELAYS
IO20=1,OK
STOPX,OK
X-16000,OK
REPEAT,
    MSTX,
    RSAVE1,
UNTILM0,
DELAY=16000

' Swing axis
HSPD=32000,OK
DELAYS
Y117792,OK
REPEAT,
    MSTY,
    RSAVE1,
UNTILM0,
DELAY=4000

' Low speed swing test
HSPD=448,OK
DELAYS
Y122240,OK
REPEAT,
    MSTY,
    RSAVE1,
UNTILM0,
DELAY=50000

' Reset to next test
HSPD=32000,OK
DELAYS
Y-16000Z-16000,OK
REPEAT,
    MSTZ,
    RSAVE1,
UNTILM0,
DELAYL
```

Auto Control

The Auto Control feature is a function that defines how the Nippon Pulse Terminal Program acts when started. There are two ways to execute this script.

1) The script defined by "autocontrol.txt" will automatically run every time you run the Nippon Pulse Terminal Program. The Auto Control feature is activated when the file "autocontrol.txt" is present in the [.../AppPrograms\AutoControl] sub-folder.

2) The script will run when called by a command line parameter. The Auto Control feature is activated when the file "autocontrol_#.txt" is present in the [.../AppPrograms\AutoControl] sub-folder, and the parameter /A# is passed by the command line (where # is a number between 1 and 999).
Examples:

- terminal.exe /A1 will run the script autocontrol_1.txt
- terminal.exe /A999 will run the script autocontrol_999.txt

This feature is very powerful, with a script format that is very simple and a robust set of commands that allows the following key functions to be done automatically:

Setting and controlling the Nippon Pulse Terminal Program startup defaults.

Command	Description
Language	Sets the startup displayed language
Device	Select the device to connect to
ShowMode	Select the Terminal display mode
Show	Display text in Terminal display
Clear	Clear the Terminal display
Refresh	Refresh the device list
ResizeApp	Resize the Terminal application

Control of Auto Control script

Command	Description
LoopStart	Indicates the beginning of a loop
LoopEnd	Indicates the end of a loop
Wait	Pause Auto Control Script
Exit	Stops Auto Control script and closes Terminal Program
ESCExit	Stops Auto Control script and closes Terminal Program when ESC pressed
ESCStop	Stops Auto Control script when ESC pressed
Erase	Erase the Auto Control script file

Runs ASCII programs

Command	Description
RunProg	Runs the selected ASCII program

Stores results and create Pass/Fail logs based on ASCII programs

Command	Description
StoreResults	Stores Terminal display to buffer
ResetResults	Clears the buffer
StoreResultsCLR	Stores Terminal display to buffer, then clears the Terminal display

SaveResults	Saves buffer to file (Overwrite allowed)
SaveResultsAuto	Saves buffer to file, auto numbering to prevent overwrite. Includes Pass/Fail results
SaveResultsAutoESC	Saves buffer to file, auto numbering to prevent overwrite when ESC pressed. Includes Pass/Fail results
DefineCSV	Defines fields/values saved to CSV Pass/Fail log
UpdateCSV	Updates CSV data buffer
TrackCSVResults	Turn On/Off CSV Pass/Fail tracking
ResetCSVResults	Reset CSV Pass/Fail tracking results
PassFailMsg	Popup window showing Pass/Fail status
ShowFailCT	Shows current count of Fails
FailClear	Clear all Fail counts to 0
LoopOnFail	This command will iterate a loop in an AutoControl script if the current Fail count is >= 1

Accept user inputs, Store and display Variables, Constants, Date, Time, and ASCII reply

Command	Description
UserInput	Shows dialog-box for user input to be displayed in Terminal display (Not stored)
VarInput	Shows dialog-box for variable input stored for later use.
UserCheck	Shows dialog-box with two buttons, PASS and FAIL and a prompt for the user
ShowVar	Displays variable in Terminal display
Const	Defines a Constant
ShowC	Displays a Constant
ShowDate	Displays the current date: month, day, year
ShowMDY	Displays the current date M/D/Y
ShowTime	Displays the current time H:M AM/PM
ShowMTime	Displays the current time in military time format H:M
CMDReply	Stores reply from requested ASCII commands

Main Screen

Script

Script commands are always in square brackets ([ ]) like this:

[LANGUAGE1]

- The square brackets are required with no spaces allowed between the brackets, only text.
- The commands are not case sensitive, so either of the following would be acceptable:

[LANGUAGE1]

or

[Language1]

- The text after the command is ignored unless the command expects a parameter.
- You can add comments after a command by using the single quote character like this:

[Language1] ' select Language # 1

- Scripts will allow the use of the TAB character, but the use of spaces is preferred.

Important notes about using the Auto Control script feature:

- You can switch between different devices, languages and even program scripts in a single Auto Control script.
- You could run a program script for one device, save the results, then run a program script for another device, save the results and so on.
- You can store the results from multiple program scripts in a single file as well, simply by appending the results each time and saving it all into one file.
- While the amount of text in the Terminal Display control is limited by the operating system, the results buffer will not have the same limitations.
    - Clear the Terminal Display control using the [Clear] command after each program script is run (remember to save it to results buffer using the [StoreResults] command) and then run the next program script.
    - The results buffer can store huge amounts of data.

Clear

```
[Clear]
```

- This command tells the Terminal app to clear the Terminal Display control.

Example: `[Clear]`

CMDReply

```
[CMDReply#] Command1|Command2|Command3...
```

- This command tells the Terminal app to execute a series of commands and store the results of each command in the Command Reply buffers.
- The # character must be a valid number from 1 to 9.
    - There are 9 Command Reply buffers.
- If the command string sent is more than one command (separated by the | character instead of a comma) then the # character will represent the first Command reply buffer to use for the first command, and the other commands will incrementally increase the Buffer number each time a new command is executed.
    - If you defined the Reply Buffer number to be 4 and you send three commands, then Reply Buffers 4, 5 and 6 will be used sequentially.
- Commands sent this way will not affect the Pass/Fail status.
- The primary purpose for this script command is to send commands and get their reply so the data can be stored in a CSV file.

Example: `[CMDReply1]ID|BSTAT`

CMDVerify

There are two versions of this command. 1) Always display the Verify dialog. 2) Only display the Verify dialog in case of a fail.

It is similar to the `[CMDReply#]` command, but doesn't save the reply data in variables, and it displays this new dialog to verify the device is working. You can choose what commands to send to the device and send multiple commands at one time.

Version #1

[CMDVerify] Command1|Command2|Command3...

- This command tells the Terminal app to execute a series of commands to help verify that the device is working correctly.
- You can execute multiple commands by separating them with the | character.
- Once executed, the Terminal app will display a dialog to indicate that the commands have been executed.
- The dialog will give you a choice of continuing on (press OK button) or to Try Again.

Example: [CMDVerify] ID|VER

It will always display a dialog that looks like this:

Version #2

[CMDVerify] Command1|Command2|Command3{REPLY1|REPLY2|REPLY3}

- This command tells the Terminal app to execute a series of commands to help verify that the device is working correctly.

- You can execute multiple commands by separating them with the | character.

- Once executed, the Terminal app will display a dialog to indicate that the commands have been executed, if the reply values do not match the expected reply values defined or if a serial timeout occurs. The dialog will give you a choice of continuing on (press OK button) or to Try Again. The expected reply values must be encased between curly brackets { } and separated by the | character. There must be the same number of reply values as there are command values.

The second version adds expect-reply values, and it uses the reply values to determine if the verify passed or failed. If the reply values do not match the expected reply values or a serial timeout occurs, the verify dialog will display, otherwise it will not be displayed (it passed).

Const

[Const#] SomeText

- This command defines a Constant, which later can be read back.
- The # character must be a number from 1 to 9.
- After the command, the constants actual text is defined.

Example: [Const2] This is the constant number 2 text

DebugOff

[DEBUGOFF]

This command turns off the Debug mode.

DebugOn

[DEBUGON]

- This command turns on script debugging.
- It will display each command as excited and the index (row) of the command in the script (comments are not counted as rows).
- This is useful for finding problems in the "logic" of a script.

DefineCSV

```
[DefineCSV] CSVFormText
```

- This command defines the formatting for the CSV tracking files.
- Each line of the CSV file is defined by passing a text string (CSVFormatText), where each column is defined with the name of the column, and where the data comes in parentheses () after it.
- The Column name can be any alphanumeric characters.
    - Special characters should be avoided if possible (comma, quotes, etc.).
- The data location part in parentheses () must be one of the following:

(V1) - Variable #1
(V2) - Variable #2
(V3) - Variable #3
(V4) - Variable #4
(V5) - Variable #5

(V6) - Variable #6
(V7) - Variable #7
(V8) - Variable #8
(V9) - Variable #9
(PF) - Pass or Fail
(FC) - Fail Count
(MT) - Time in Military format (Hours:Minutes:Seconds) (Example: 14:03:01)
(TM) - Time (Example: 1:00 PM)
(MDY) - Month / Day / Year (Example: 10/30/2020)
(DT) - Date (Month Day Year) (Example: October 30 2020)
(C1) - Constant #1
(C2) - Constant #2
(C3) - Constant #3
(C4) - Constant #4
(C5) - Constant #5
(C6) - Constant #6
(C7) - Constant #7
(C8) - Constant #8
(C9) - Constant #9
(R1) - Reply Buffer #1
(R2) - Reply Buffer #2
(R3) - Reply Buffer #3
(R4) - Reply Buffer #4
(R5) - Reply Buffer #5
(R6) - Reply Buffer #6
(R7) - Reply Buffer #7
(R8) - Reply Buffer #8
(R9) - Reply Buffer #9

Example: `[DefineCSV]Serial Number(V2)Date(V3)Pass/Fail(PF)Failure Count(FC)`

Device

[Device#]

- This command tells the Terminal app to select the device that is defined by the # character as an index.
  - If # is 3, then the third device in the list is selected.
- The # character must be a valid number from 1 to 9.

See Device selection for more details.

Example: [Device2] ' selects 2nd device in device list

Erase

[Erase]

- This command tells the Terminal app to erase the Auto Control script file immediately.
- This does not end the script, since the script is currently in memory.
  - This simply erases the script file, so the next time the Terminal app is run no script will exist, so it will start up normally.

Example: [Erase] ' make sure script doesn't run a second time when app run again

ESCExit

[ESCExit]

- This commands checks to see if the ESC was pressed (including User Input and Variable Input dialog boxes) and if so, stops the AutoScript and also terminates the Terminal app.

ESCStop

[ESCStop]

- This command checks to see if the ESC was pressed (including User Input and Variable Input dialog boxes) and if so, stops the AutoScript without terminating the Terminal app.

Exit

[Exit]

- This command tells the Terminal app to Exit the Auto Script and also to terminate the application immediately.

Example: [Exit]

FailClear

[FailClear]

- This command clears the Fail Counter flag.
  - It resets it to zero.
- When this flag is zero then it is assumed the current test is in a Pass state.
  - If the value is not zero, then the current text is in a failed state.
- Every time a program sends a command, it compares the expected reply value to the actual reply returned.
  - If they do not match, then the Fail counter is increased by 1.

Language

[Language#]

- This command selects the current language for the Terminal app.
- The # character must be a valid number from 1 to 9. Only the first 9 languages listed can be used.

See Language selection for more details.

Example: [Language3]

LoopEnd

[LoopEnd]

- This command indicates the end of a loop in an Auto Control script.
- The terminal program will only repeat the loop if the [LoopStart] command was executed at some point before this command.
  - If it wasn't, then the script will continue to the next command.
  - If the [LoopStart] command had previously been executed, the script will loop back to that point in the script.

- To prevent the possibility of a forever loop, the Terminal program responds to the ESC key to terminate any repeating of the loop.
- The ESC key does not terminate the AutoScript, it just does not loop back when it executes the next [LoopEnd] command.
    - Pressing the ESC key in the User Input or Variable Input dialog also will prevent the next [LoopEnd] command from repeating the loop.

V2.49+

- [LoopEnd] will now check to see if the ESC key has been pressed. If neither [ESCStop] or [ESCExit] have been executed yet then [LoopEnd] will act like [ESCStop].

LoopOnFail

[LoopOnFail]

- This command will iterate a loop in an AutoControl script if the current Fail count is >= 1.
- The terminal program will only repeat the loop if the [LoopStart] command was executed at some point before this command. If it wasn't, then the script will continue to the next command.
- If the [LoopStart] command had previously been executed, the script will loop back to that point in the script.
- When the Loop does iterate because of a Fail state, the Fail count will be reset back to zero.
- The Fail Dialog will automatically be displayed.

V2.49+

[LoopOnFail] can now have an instruction after it like this:

[LoopOnFail]Update

- This will tell the Terminal app to also execute the same action as a call to [UpdateCSV] would do. This allows you to update the CSV file data buffer with the current information for the Failed test.

LoopStart

[LoopStart]

- This command indicates the beginning of a loop in an Auto Control script.
- The Terminal program will simply store this position in the script.

PassFailMsg

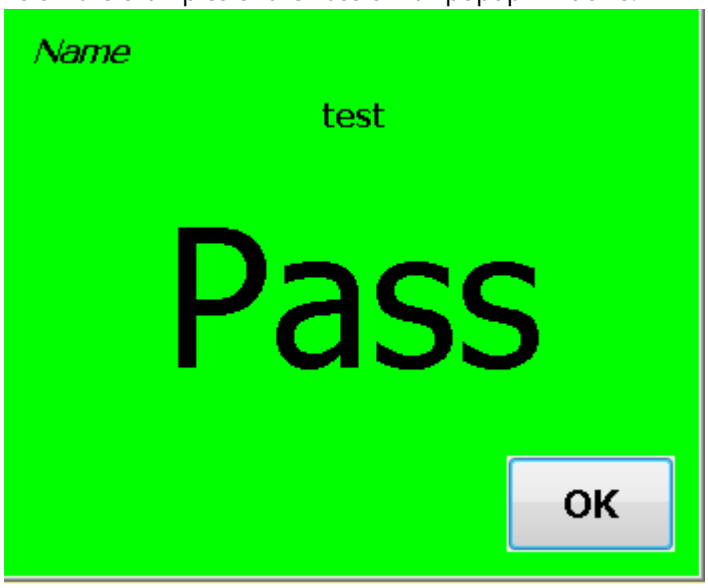
[PassFailMsg]

- This command displays a popup window that shows the current Pass or Fail state.

[PassFailMsg#]

- This command displays a popup window that shows the current Pass or Fail state. It also can display the contents of one Variable.
- The Variable number is defined by the # character, which must be a number from 1 to 9.

Below are examples of the Pass or Fail popup windows:



Refresh

[REFRESH]

This command will Refresh the device list. It will also automatically select the last device defined using the [Device#] command.

ResizeApp

[ResizeApp] Mode

This command will resize the Terminal application. The parameter Mode can be one of the following:

LEFT
CENTER
RIGHT
MAX
LEFTMAX

CENTERMAX
RIGHTMAX

The Mode value determines whether the app will move to the left, center or right and also whether to maximize the app (either only height if Left, Center or Right is used or maximize fully if only Max used).

ResetCSVResults

[ResetCSVResults]

- This command clears the string buffers for CSV results tracking.
- There are two CSV buffers.
    1. One for programs that Pass.
    2. One for programs that Fail.

ResetResults

[ResetResults]

- This command clears the buffer that contains text previously stored using the [StoreResults] command.

Example: [ResetResults]

RunProg

[RunProg#]

- This command tells the Terminal app to run one of the program scripts.
- The # character must be a valid number from 1 to 9, or it can be a question mark (?) to open a named file or force an Open File dialog to be displayed.
    - The program scripts 1 to 9 are the same as if you pressed the CTRL-F1 to CTRL-F9 keys.
    - When the # character is a question mark (?), there are two options.
        - No text after the command forces an Open File dialog to be displayed.
        - You can also use the following syntax to define a "custom" program by adding a custom program name macro after the command
            - [RUNPROG?]mycustomname
            - The custom program filename macro must be alpha-numeric characters only.

See Run ASCII program for more details.

Example: [RunProg3] 'run program script number 3
[RunProg?] 'force an Open File dialog to be displayed so user can select which program script to run
[RunProg?]mycustomname 'run program script test99mycustomname.prg

SaveResults

[SaveResults#]

- This command tells the Terminal app to save the current results buffer text to a disk file.
- The # character must be a valid number from 1 to 9.
    - This number determines which filename to use for the saved disk file.

  - If the file already exists, then the old file is deleted and a new one created, storing the current results buffer.
- The filenames that the Terminal app will store are as follows (based in # character value):

testresults1.txt
testresults2.txt
testresults3.txt
testresults4.txt
testresults5.txt
testresults6.txt
testresults7.txt
testresults8.txt
testresults9.txt

Example: [SaveResults3] ' save results buffer to file # 3

SaveResultsAuto

[SaveResultsAuto] MyFileName

- This command tells the Terminal app to save the current results buffer text to a disk file.
- The text after the command will be the first part of the filename used to save the results buffer data.
  - Do not define a file extension!
- Variables can be used to define the file name:
  - [SaveResultsAuto] MyFile{V1}_{V2}
- The Terminal program will append the following to the filename.
  - First it will auto-generate a unique sequential number for the file.
  - Then it will append the .txt file extension.
  - If the file already exists, it will increase the sequential number by adding 1.
- The files are never deleted, but simply sequentially increased to create the next filename.
- If you generated five files using the name MyFileName then actual files would be named:

MyFileName00001.txt
MyFileName00002.txt
MyFileName00003.txt
MyFileName00004.txt
MyFileName00005.txt

- If CSV tracking is turned ON, the Terminal app will also save the current data in the CSV tracking buffers.
- The filename will be similar except there will be two files, one appended with _Pass and the other _Fail.
  - The file extension will be .csv rather than .txt.
- When file MyFileName00001.txt is generated the CSV filenames would be:

MyFileName00001_Pass.csv
MyFileName00001_Fail.csv

SaveResultsAutoESC

[SaveResultsAutoESC] MyFileName

- This command tells the Terminal app to save the current results buffer text to a disk file when it senses the ESC key was pressed during a Loop when any ESC key command is executed.

    - Commands such as [ESCExit] and [ESCStop] will auto-save the file at the time they are executed.
- The text after the command will be the first part of the filename used to save the results buffer data.
    - Do not define a file extension!
- Variables can be used to define the file name:
    - [SaveResultsAutoESC] MyFile{V1}_{V2}
- The Terminal program will append the following to the filename.
    - First it will auto-generate a unique sequential number for the file.
    - Then it will append the .txt file extension.
    - If the file already exists it will increase the number sequentially.
- The files are never deleted, but simply sequentially increased to create the next filename.
- If you generated five files using the name MyFileName then actual files would be named:

MyFileName00001.txt
MyFileName00002.txt
MyFileName00003.txt
MyFileName00004.txt
MyFileName00005.txt

- If CSV tracking is turned ON, the Terminal app will also save the current data in the CSV tracking buffers.
- The filename will be similar except there will be two files, one appended with _Pass and the other _Fail.
    - The file extension will be .csv rather than .txt.
- When file MyFileName00001.txt is generated the CSV filenames would be:

MyFileName00001_Pass.csv
MyFileName00001_Fail.csv

SetVar

[SetVar#]

- [SetVar#] will set (define) the initial value of a variable. The # character is a value from 1 to 9.

ShellApp

[SHELLAPP] Path_to_filename_to_run

- This command Runs an external application.
- The parameter must be a valid path to an external application to run (ie. EXE, BAT or any other valid app).
- The app is run synchronously by default.
- You can have it run asynchronously by prefixing the path with the characters:
    - {A}

The path is checked to see if the \ character exists in it. If not, then it is assumed it is only the apps filename and the path for AutoControl files is prefixed to it. Otherwise a full valid path is required.

Show

[Show] some text

- This command tells the Terminal app to display text to the Terminal Display control.

- Whatever text is after the command will be displayed.
- If a comment (single quote) character is found in the text, all text after the single quote is ignored.

Example: [Show]Some text to display ' all text after quote is ignored

This is a different Show then the one used for ASCII programs.

ShowC

[ShowC#]

- This command displays a Constant, which was previously defined using the [Const#] command in the Terminal window.
- The # character must be a number from 1 to 9.

Example: [ShowC2]

ShowDate

[ShowDate]

- This command displays the current date (Example: October 30 2020) in the terminal display.

Example: [ShowDate]

ShowFailCT

[ShowFailCT]

- This command displays in the Terminal window the current Failure count, if the Failure count is greater than zero.

ShowMDY

[ShowMDY]

- This command displays the current date in the Month/Day/Year format in the terminal window.

Example: [ShowMDY]

ShowMode

[ShowMode#]

- This command toggles the Terminal Display controls display mode.
- The # character must be a valid number from 1 to 5, which defines the Show Mode.
- The Show mode values are as follows:

Mode 1 - display command on first line and return value on next line
Mode 2 - display command and return value on same line
Mode 3 - display command and return value on same line (shortened version)
Mode 4 - return error lines only for program scripts
Mode 5 - return error lines only for program scripts, plus comments

See Terminal Display mode for examples.

Example: [ShowMode3]

ShowMTime

[ShowMTime]

- This command displays the current time in military time format (Example: 24:01:02) in the terminal window.

Example: [ShowMTime]

ShowTime

[ShowTime]

- This command displays the current time (Example: 1:00 PM) in the terminal window.

Example: [ShowTime]

ShowVar

[ShowVar#]

- This command tells the Terminal app to display the text stored in a Variable to the Terminal Display control.
- The # character must be a number from 1 to 9, which indicates the variable to retrieve text from.
- The variables match the same ones defined in the [VarInput#] command.

Example: [ShowVar1]

StoreResults

[StoreResults]

- This command tells the Terminal app to store whatever is currently displayed in the Terminal display control in a buffer, which can be later saved to a disk file.
- If there is already text in this buffer, the current text is appended after that text.
- The previous text is not erased. If you do not call the [Clear] command sometimes after this command you will get spurious (extra) data in your results file.

Example: [StoreResults]

StoreResultsCLR

[StoreResultsCLR]

- This command tells the Terminal app to store whatever is currently displayed in the Terminal display control in a buffer, which can be later saved to a disk file.
- If there is already text in this buffer, the current text is appended after that text.
- This command automatically clears the Terminal Display text window.

TrackCSVResults

[TrackCSVResults] Mode

- This command turns the CSV tracking either On or Off.
- To turn it ON, define the Mode Parameter as one of the following:

```
[TrackCSVResults] ON
[TrackCSVResults] 1
[TrackCSVResults]
```

- To turn it OFF, define the Mode parameter as one of the following:

```
[TrackCSVResults] OFF
[TrackCSVResults] 0
```

UpdateCSV

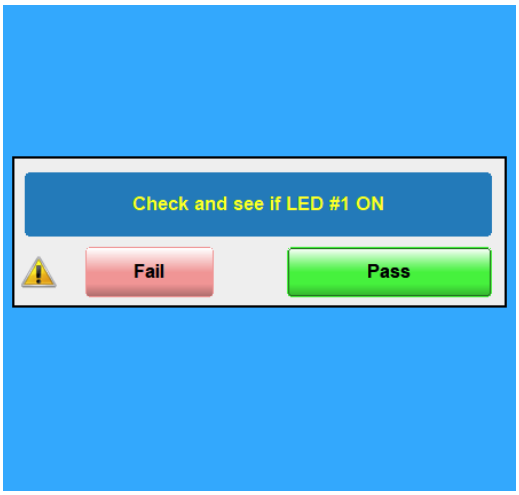
[UpdateCSV]

- This command updates the CSV data buffers with the current data.

UserCheck

[USERCHECK] Prompt Text

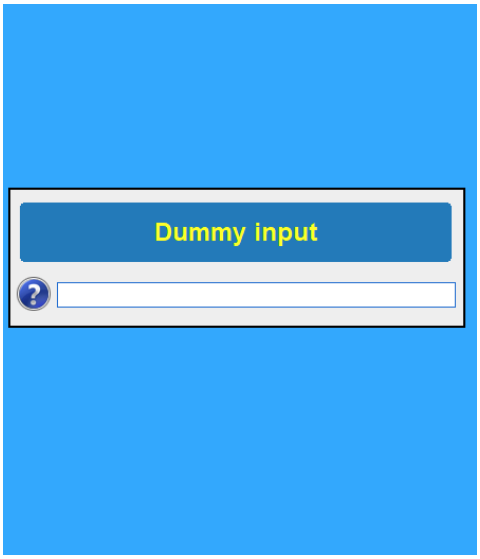
- This command displays a Dialog with two buttons, PASS and FAIL and a prompt for the user.
- The user will visually inspect the device in response to the prompt and then select either a Pass or a Fail state.
- Up to three lines of text can be used for the prompt. You simply use the | character between lines like this:
    - [USERCHECK] First Line of Text|Second Line of Text|Third Line of text



UserInput

[UserInput] Some Text

- This command displays the User Input dialog-box.
- The user then can enter text, which will be displayed in the Terminal Display Box.
- The user must press either the ENTER key to display the entered text or the ESC key to skip displaying the text.
    - The dialog-box will stay visible until the user presses either ENTER or ESC.
- The text after the command will be displayed in the dialog-box as a prompt.
- The text "Enter:" will be added as a prefix to the text string.
    - So, if you input the phrase "Serial Number" then the dialog-box will display the text "Enter: Serial Number".



VarInput

[VarInput#] Some Text

- This command displays the User Input dialog-box.
- The user then can enter text, which will be stored in a memory variable for later use.
- There are nine (9) memory variables allowed.
  - The # character must be a value from 1 to 9.
- The user must press either the ENTER key to store the entered text or the ESC key to skip storing the text (memory variable will be set to a NULL string).
  - The dialog-box will stay visible until the user presses either ENTER or ESC.
- The text after the command will be displayed in the dialog-box as a prompt.
- The text "Enter:" will be added as a prefix to the text string.
  - So, if you input the phrase "Serial Number" then the dialog-box will display the text "Enter: Serial Number".

Wait

[Wait]

- This command tells the Terminal app to wait 2 seconds.

Example: [Wait]

Examples

In the example code, we use the following color coding to make the code easier to understand.

```
' Comments
[AutoControl_Command] Variable_or_userstring
```

- Unless otherwise noted, all example code is written for the Commander series of controllers.
  - Please reference the Command Reference for differences in the ASCII commands for the Performax series of controllers.
- The formatting of the AutoControl program is the same for all controller series.
- For more information on any ASCII commands shown in the examples, please refer to the Command

Reference.

Example #1 Example to demonstrate running ASCII program and changing display languages.

Example #2 Example to demonstrate an AutoControl script with two user-prompted input variables, and an indefinite loop that runs the ASCII program until the ESC key is pressed.

Example #3 Example to demonstrate an AutoControl script that is used for product testing. It has two user prompted input variables, creates a CSV file to log the results, an on screen popup that displays pass/fail, and an indefinite loop that runs the ASCII program until the ESC key is pressed.

Example 1

Example to demonstrate running ASCII program and changing display languages.

```
' this is a test run of the auto-control feature
[Language1]              ' set language to English
[Device1]                ' select first Device
[ResetResults]           ' clear buffer
[Clear]                   clear display
[Show]----------------------------------------
[Show]This is a test run of Auto-control feature
[Show]----------------------------------------
[ShowMode2]              ' display command and return value on same line
[RunProg4]               ' runs ASCII program test04.prg
[StoreResults]           ' copies display contents to buffer
[Clear]                  ' clear display
[Show]----------------------------------------
[Show]Run test program again with different mode
[Show]----------------------------------------
[ShowMode3]              ' display command and return value on same line (shortened version)
[RunProg4]               ' runs ASCII program test04.prg
[StoreResults]           ' copies display contents  to buffer
[SaveResults4]           ' saves buffer contents to testresults4.txt

' rerun tests but in another language
[Language3]              ' Change language
[ResetResults]           ' clear buffer
[Clear]                  ' clear display
[Show]----------------------------------------
[Show]This is a test run of Auto-control feature
[Show]----------------------------------------
[ShowMode2]              ' display command and return value on same line
[RunProg4]               ' runs ASCII program test04.prg
[StoreResults]           ' copies display contents to buffer
[Clear]                  ' clear display
[Show]----------------------------------------
[Show]Run test program again with different mode
[Show]----------------------------------------
[ShowMode3]              ' display command and return value on same line (shortened version)
[RunProg4]               ' runs ASCII program test04.prg
```

```
[StoreResults]          ' copies display contents to buffer
[SaveResults5]          ' saves buffer contents to testresults5.txt

[Language1]             ' set language back to English
[Exit]                  ' terminate app
```

Example 2

Example to demonstrate an AutoControl script with two user-prompted input variables, and an indefinite loop that runs the ASCII program until the ESC key is pressed.

```
[Language1]                     ' set language to English
[Device2]                       ' select second Device
[Clear]                         ' clear display
[VarInput1]Tester Name          ' user input dialog-box requesting "Enter: Tester Name"
[SaveResultsAutoESC]            ' save results buffer text to the next available disk file when the
ESC key is pressed
[ResetResults]                  ' clear buffer
[LoopStart]                     ' Start of Loop
      [VarInput2]ID Number      ' user input dialog-box requesting "Enter: ID Number"
      [ESCExit]                 ' check for ESC key press, if ESC pressed, stops the AutoScript
and also terminates the Terminal app
      [Show]----------------------------------------
      [ShowVar1]                ' display variable #1 (Tester Name)
      [ShowVar2]                ' display variable #2 (ID Number)
      [Show]----------------------------------------
      [ShowMode4]               ' display only lines that fail for ASCII program script
      [RunProg4]                ' runs ASCII program test04.prg
      [ESCExit]                 ' check for ESC key press, if ESC pressed, stops the AutoScript
and also terminates the Terminal app
      [Show]----------------------------------------
      [Show]            End test program
      [Show]----------------------------------------
      [StoreResults]            ' copies display contents to buffer
      [Clear]                   ' clear display
[LoopEnd]                       ' End of Loop (returns to LoopStart until ESC key is pressed)
```

Example 3

Example to demonstrate an AutoControl script used for product testing. It has two user-prompted input variables, creates a CSV file to log the results, an on screen popup that displays pass/fail, and an indefinite loop that runs the ASCII program until the ESC key is pressed.

```
' this is a test run of the auto-control feature
[Language1]                     ' set language to English
[Device1]                       ' select first Device
[Clear]                         ' clear display
[ResetCSVResults]               ' clears the string buffers for CSV results tracking
[TrackCSVResults]ON             ' turns the CSV tracking On
[DefineCSV]ID Number(V2)Pass/Fail(PF)Fail Count(FC)Tester Name(V1)Mil.Time(MT)
Time(TM)Mon/Day/Yr(MDY)Date(DT)       ' Define data to save to the CSV file
[VarInput1]Tester Name          ' user input dialog-box requesting "Enter: Tester Name"
[SaveResultsAutoESC]            ' save results buffer text to the next available disk file when the
ESC key is pressed
[ResetResults]                  ' clear display
[LoopStart]                     ' Start of Loop
      [FailClear]               ' clears the Fail Counter flag
      [VarInput2]ID Number      ' user input dialog-box requesting "Enter: ID Number"
      [ESCExit]                 ' check for ESC key press, if ESC pressed, stops the AutoScript
and also terminates the Terminal app
      [Show]----------------------------------------
```

```
        [ShowVar1]              ' display variable #1 (Tester Name)
        [ShowVar2]              ' display variable #2 (ID Number)
        [UserInput]
        [Show]----------------------------------------
        [ShowMode4]             ' display only lines that fail for ASCII program script
        [RunProg4]              ' runs ASCII program test04.prg
        [ESCExit]               ' check for ESC key press, if ESC pressed, stops the AutoScript
and also terminates the Terminal app
        [Show]----------------------------------------
        [Show]            End test program
        [Show]----------------------------------------
        [ShowFailCT]            ' displays current Failure count
        [StoreResults]          ' copies display contents to buffer
        [UpdateCSV]             ' add the current data to CSV data buffers
        [PassFailMsg2]          ' displays a popup window that shows the current Pass or Fail
state
        [Clear]                 ' clear display
[LoopEnd]                       ' End of Loop (returns to LoopStart until ESC key is pressed)
```

USB

USB communication between the Nippon Pulse terminal program and the Commander and Performax controller series is done using Windows compatible DLL API function calls.

The DLL for the Performax series is PerformaxCom.DLL Version 4.0.1

The DLL for the Commander series is CMDHidApi.DLL. Version 1.3.123 or newer is required.

- Confirm that each connected Device has a unique Device ID. If multiple devices share a Device ID, then they all will be displayed, but only one will be accessible.

When started, the Terminal program will:

1. Check the version of DLL used to connect to the Commander products. The version you are using is displayed in a box next to the Language selection box.

If the DLL is correct it will be listed in blue.

If the DLL is out of date it will be red, with the message "Update DLL!"

2. Scan for all available connected Commander and Performax controllers connected by USB. These will be listed in the drop-down box next to the label “Devices Found" You can easily select the connected controller to which you want to send commands and receive responses.

Refresh USB connected devices

Pressing the  (refresh) button will refresh the USB-connected device list.

Serial

Serial communication between the Nippon Pulse terminal program and the Commander and Performax series is possible, but the communication ports must be manually configured.

The configuration file is located in the [.../AppConfig] sub folder.

The configuration file "COMM_Ports.CFG" can be edited with any simple ASCII-based text editor. For each Serial communication port, a separate line needs to be added.

Device ID {COM port, baud rate}

Below are some examples:

CMD-4CR-01{COM4:115200}
CMD-4CR-02{COM4:115200}
CMD-4CR-01{COM39:9600}
PMX-2ED-SA-01{COM4:9600}

There is also an example of the configuration file supplied called "COMM_Ports.cfg_example" supplied in the [.../AppConfig] folder. If you delete the "_example" from the name it will operate.

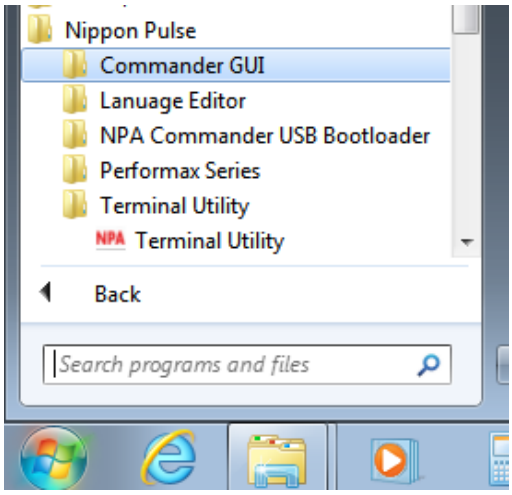
All Serial communication ports listed in the "COMM_Ports.CFG" file will be listed in the drop-down box next to the label "Devices Found," regardless of whether or not they are connected.

Folder / File layout

Below is a tree showing the files included with installed versions of the Nippon Pulse terminal program and their locations.

For portable version click here.

Shortcut in Start menu



\Users\username\Documents\Nippon Pulse Terminal

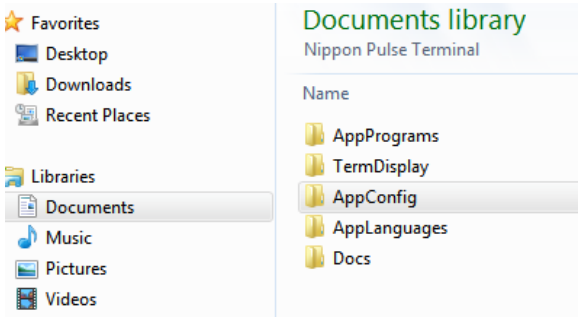
└AppPrograms Location for ASCII Programs

test01.prg

|

test10.prg

└AppPrograms	Location for Auto Control Program
Autocontrol.txt	Auto Control Program
└AutoResults	Location for Auto Results
└AppLanguages	Location for language templates
└AppConfig	Location for serial connection configuration files
COMM_Ports.CFG	Configuration file for comports
└TermDisplay	Default location for saving test from Terminal Display area
└Docs	
Software Utility - Terminal	The help document in many different formats



Executable

\Program Files (x86)\Nippon Pulse\Terminal utility

NPterminal.EXE	Main Terminal program executable
langeditDDT.EXE	Language Editor
SiUSBXp.dll	Performax series USB Driver
PerformaxCom.dll	Performax series DLL API
ezgui50.dll	Terminal application DLL
CMDHidApi.dll	Commander series DLL API (32bit)

Folder / File layout

Below is a tree showing the files included with the portable version of the Nippon Pulse terminal program and their locations.

RunNPTerminal.exe	Portable Terminal program executable

└Terminal	Folder holding all files for Terminal program
terminal.EXE	Terminal program executable
langeditDDT.EXE	Language Editor
SiUSBXp.dll	Performax series USB Driver
PerformaxCom.dll	Performax series DLL API
ezgui50.dll	Terminal application DLL
CMDHidApi.dll	Commander series DLL API
TermDisplay	Default location for saving test from Terminal Dis area
└AppTemplates	
terminal.tpl	Terminal application Template
└AppPrograms	Location for ASCII Programs
test01.prg | test10.prg	
└AppPrograms	Location for Auto Control Program
Autocontrol.txt	Auto Control Program
└AutoResults	Location for Auto Results
└AppLanguages	Location for language templates
└AppConfig	Location for serial connection configuration files
COMM_Ports.CFG	Configuration file for comports

Contact information

Nippon Pulse America, Inc.

4 Corporate Drive Street, Radford, VA 24141 USA
Phone: 1-540-633-1677
E-mail: info@nipponpulse.com
Web: https://www.nipponpulse.com