PLS600
Programmable DC Power Supplies
Scripting Language Manual

XP Power

Contents

1 OVERVIEW 3
2 SCRIPTING LANGUAGE SYNTAX 4
2.0 Script Naming 4
2.1 Script Format 4
2.2 Whitespace 4
2.3 Strings 4
2.4 Numbers 4
2.5 Labels 4
2.6 Keywords 5
2.7 Variables 5
2.8 Reserved Variables 5
2.9 Statement Syntax 7
3 RUNNING SCRIPTS 10
3.0 Via SCPI commands 10
3.1 Via the front panel 14
4 EXAMPLES 15
4.0 EXAMPLE 1: Sawtooth wave form 15
4.1 EXAMPLE 2: Analog output triangle wave form 16
4.2 EXAMPLE 3: Timer triggered output 17
4.3 EXAMPLE 4: Analog input triggered output 18
4.4 EXAMPLE 5: Arbitrary waveform 19
5 LIMITATIONS 28
6 STATUS AND ERRORS 29

1 OVERVIEW

A full-featured scripting language (loosely based on the *BASIC* programming language) is implemented within the PLS600 power supplies. Scripts can be downloaded and uploaded between the power supply and a computer via SCPI commands using either of the USB or Ethernet (LAN) interfaces.

Up to 10 scripts may be stored verbatim in persistent memory within the power supply and recalled. Scripts may be run/halted from either the front panel or via SCPI commands.

When run, scripts are compiled by the power supply's processor. Errors in the script will cause it to not be executed. If there are no errors, the compiled script is executed.

Certain operating parameters (such as the voltage and current setpoints) are accessible directly by a script allowing complex waveform outputs to be synthesized.

2 SCRIPTING LANGUAGE SYNTAX

2.0 Script Naming

Script names may be up to 32 characters in length. However, on the front panel only the first 14 characters are displayed.

2.1 Script Format

Scripts are written as human-readable text. Scripts are composed of an ordered series of lines. The first line of the script is the starting point of the script when it is run.

Each line may contain one statement. A statement is usually a keyword followed by zero or more parameters (which can include other keywords).

Only one operation is allowed per line. Compounding operations is not permitted.

2.2 Whitespace

Whitespace characters (tabs and spaces) may be inserted between each parameter of a statement.

A line can consist of only whitespace. Such a line is ignored by the compiler. Lines that start with the REM keyword are also ignored by the compiler.

2.3 Strings

Strings are comprised of upper-case (A-Z), lower case (a-z), numbers (0-9) and underscore characters. The first character of a string cannot be a number.

2.4 Numbers

Numbers are comprised of numeric characters (0-9), decimals and minus signs. Only one decimal may appear in a number. Only one minus sign may appear in a number and, if present, it must be the first character of a number.

2.5 Labels

Labels are used to identify points of execution in a script. A label must be put on its own line in a script, it cannot have a statement appended to it in the same line. A label is a string followed by a colon character (with no interposing whitespace)

2.6 Keywords

The following keywords are used in the scripting language.

END
FOR
GOSUB
GOTO
IF
LET
NEXT
RETURN
WAIT
TO
STEP
THEN

The lower-case versions of these keywords are also accepted, but mixing of cases is not.

2.7 Variables

A variable is a string that corresponds to a floating-point value in memory. Variables are used in scripts to hold values and make computations.

A variable may not be named with the same string as a keyword.

2.8 Reserved Variables

The following variables are reserved. These variables may be entered into the script as upper-case or lower-case strings (but not a combination of upper and lower case). These reserved variables map to corresponding system properties. Use of these variables is how control over the power supply by the script is realized. The use of these variables is detailed below:

VOLTAGE_SETPOINT – This variable holds the voltage setpoint of the power supply (in Volts). If the output is enabled when this variable is written, the setpoint will immediately influence the output.

CURRENT_SETPOINT – This variable holds the current setpoint of the power supply (in Amperes). If the output is enabled when this variable is written, the setpoint will immediately influence the output.

POWER_SETPOINT – This variable holds the power setpoint of the power supply (in Watts). If the output is enabled when this variable is written, the setpoint will immediately influence the output.

OVER_VOLTAGE_LIMIT – This variable holds the over-voltage limit of the power supply (in Volts). When this variable is written, the limit will immediately be registered for the purpose of protecting the device connected to the output of the power supply.

OVER_CURRENT_LIMIT – This variable holds the over-current limit of the power supply (in Amperes). When this variable is written, the limit will immediately be registered for the purpose of protecting the device connected to the output of the power supply.

OVER_POWER_LIMIT – This variable holds the over-power limit of the power supply (in Watts). When this variable is written, the limit will immediately be registered for the purpose of protecting the device connected to the output of the power supply.

OUTPUT_MODE – This variable can be used to turn the output of the power supply on and off. Writing a 0.0 to the variable will turn the output off. Writing a 1.0 to this variable will turn the output on.

VOLTAGE_MEASURED – This variable is continuously updated by the power supply as the script runs. It reflects the voltage (in Volts) being output by the power supply.

CURRENT_MEASURED – This variable is continuously updated by the power supply as the script runs. It reflects the current (in Amperes) being output by the power supply.

POWER_MEASURED – This variable is continuously updated by the power supply as the script runs. It reflects the power (in Watts) being output by the power supply.

TIMEBASE – the TIMEBASE variable holds a running count of the number of milliseconds that have elapsed since the script was started.

ANALOG_INPUT_VOLTAGE – This variable is automatically updated by the power supply as the script runs. When in script mode the voltage analog input will be scaled from 0.0V to 10.0V. This variable reflects the voltage being read on the analog input.

ANALOG_INPUT_CURRENT – This variable is automatically updated by the power supply as the script runs. When in script mode the current analog input will be scaled from 0.0V to 10.0V. This variable reflects the voltage being read on the analog input

ANALOG_OUTPUT – Writing a voltage value (from 0.0 to 10.0 Volts) to this variable will cause the same value to be output from the analog output port of the power supply.

NOTE: for those users that are using analog I/O with a 5V range, they will simply need to restrict their script to operate in that range (i.e. treat input voltages above 5V as 5V, never set the analog output above 5V)

NOTE: many of these variables have operating limits that are applied when they are written. For example, attempting to write a value of 500.0 Volts to the voltage setpoint variable will be ignored since none of the PLS600 models support a voltage that high. These limits are applied on a model-by-model basis and match the limits found using the front panel controls of the PLS600 power supply.

2.9 Statement Syntax

The following rules of syntax apply to statements in the script:

REM<any text allowed>

Remark statement: used to enter comments into the script. This statement does not affect script operation.

END

End statement: Used to halt a script. Note: all scripts have an implicit END immediately after the last line of the script.

FOR <variable1> = <constant2 | variable2> TO < constant3 | variable3> STEP <constant4 | variable4>

Loop statement: The first variable after the FOR keyword identifies the loop variable. The variable is initialized to the value in constant2/variable2. Each time the corresponding NEXT statement is reached, the loop variable is compared to constant3/variable3. If they are equal the loop exits and the script continues to execute at the line following the NEXT statement. Otherwise, the value in constant4/variable4 is added to the loop variable and the script continues to execute at the line following the FOR statement.

NEXT <variable>

The NEXT statement must contain the loop variable to a previously encountered FOR statement. If it does not, the NEXT statement is ignored, otherwise refer to the loop statement to see how the NEXT statement is handled.

GOSUB <label>

Subroutines: The script starts to execute at the line following the label declared in this GOSUB statement. The index of the line immediately following the GOSUB statement is pushed onto the return stack.

RETURN

Return from subroutine: The index of the line after the most recently called GOSUB statement is retrieved and the script starts executing from that line. NOTE: if there is no corresponding GOSUB statement then the RETURN statement acts the same as an END statement.

GOTO <label>

Unconditional branch statement: The script starts executing from the line immediately after the specified label.

IF <variable1/constant1> <conditional> <variable2/constant2> THEN <label>

Conditional branch statement: variable1/constant1 is compared to variable2/constant2. The conditional operator can be one of the following:

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

If the condition specified by the conditional operator is true then the script starts executing from the line immediately after the specified label, otherwise the script continues to execute from the line immediately following the IF statement.

<LET> <variable1> = <variable2/constant2>

Assignment statement (simple). The simple assignment statement places the contents of variable2/constant2 into the memory location for variable1.

<LET> <variable1> = <variable2/constant2> <operator> <variable3/constant3>

Assignment statement with math operation: In this assignment operation variable2/constant2 is operated on by variable3/constant3. The operation can be one of (+ - * /). The result of the operation is then place in variable1.

NOTE: for both the simple assignment and assignment with math operation, the LET keyword is optional. Thus:

LET a = a + 1

and

a = a + 1

are identical from the perspective of the compiler.

WAIT <variable | constant>

The WAIT statement delays processing of the next line of the script for the number of milliseconds held in variable/constant.

3 RUNNING SCRIPTS

3.0 Via SCPI commands

The following SCPI commands have been implemented to allow download, upload, storage, retrieval, running and halting of script. As well the status of the scripting system is accessible via SCPI commands.

SYSTem:SCRIpt:NEW – Takes the new script's name (in quotes) as a parameter and creates a new active script with no lines and with the new given name.

SYSTem:SCRIpt:LINE – Takes one line of a script (in quotes) as a parameter and appends it to the active script.

SYSTem:SCRIpt:LINE? – Takes no parameters and returns the next line from the active script (in quotes).

SYSTem:SCRIpt:LOAD – Takes a number from 0 to 9 as a parameter and loads the script specified by the number from persistent memory into the active script RAM.

SYSTem:SCRIpt:STORe – Takes a number from 0 to 9 as a parameter and stores the active script into persistent memory at the location specified by the number.

SYSTem:SCRIpt:RUN – Takes no parameters and, if in script mode and idle, will cause the active script to be compiled and run (if no script errors occur).

SYSTem:SCRIpt:HALT – Takes no parameters and halts any script that is running.

SYSTem:SCRIpt:STATe? – this command will return "IDLE" if the script system is not performing any tasks, "RUN" if the script system is running a script, and "BUSY" if the script system is loading/storing a script from/to persistent memory.

Below is an example of the command stream used to download the EXAMPLE 1 script and store it into the power supply's persistent memory. The < and > characters are not part of the actual stream, but are used to indicate the direction of data flow. A > character indicates a SCPI command from the computer to the power supply and a< character indicates a SCPI response from the power supply to the computer.

```
>*IDN?
<XP Power, PLS605020, 031218030002, 1.01.0741/1.01.1664

>SYSTEM:PROMPT ON
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:NEW "EXAMPLE 1"
<

>SYST:SCRI:STAT?
<IDLE
```

```
>SYST:SCRI:LINE "rem EXAMPLE 1 - Sawtooth waveform "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE " "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "rem (optional) setup "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "voltage_setpoint = 0 "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "current_setpoint = 40 "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "output_mode = 1 "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE " "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "rem loop label used to create infinite loop "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "loop: "
<

>SYST:SCRI:STAT?
<IDLE
```

```
>SYST:SCRI:LINE " "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "rem use FOR/NEXT loop to create ramp "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "for i=0 to 25 step 0.01 "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "let voltage_setpoint = i "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "wait 1 "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "next i "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE " "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "rem jump back to create next ramp "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:LINE "goto loop "
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:STOR 0
```

<

>SYST:SCRI:STAT?
<IDLE

The following SCPI command stream illustrates loading the script from memory location 0, running it, then halting it.

>SYST:SCRI:LOAD 0
<

>SYSTEM:MODE SCRI
<

>SYST:SCRI:STAT?
<IDLE

>SYST:SCRI:RUN
<

>SYST:SCRI:STAT?
<RUN

>SYST:SCRI:HALT
<

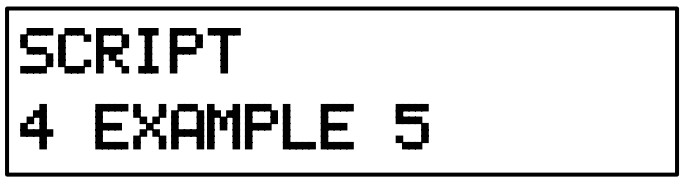
3.1 Via the front panel

Scripts cannot be entered via the front panel, however they can be loaded from non-volatile memory and executed via the front panel.

To run a script, enter the setup menu mode selection menu:

Then select SCRIPT mode:

When the script mode is selected, the user is prompted to select one of the ten scripts stored in memory (presuming the user has downloaded scripts and stored them in memory):



When the setup menu is exited, and the system is in SCRIPT mode, the selected script can be run and halted via the output on/off button.

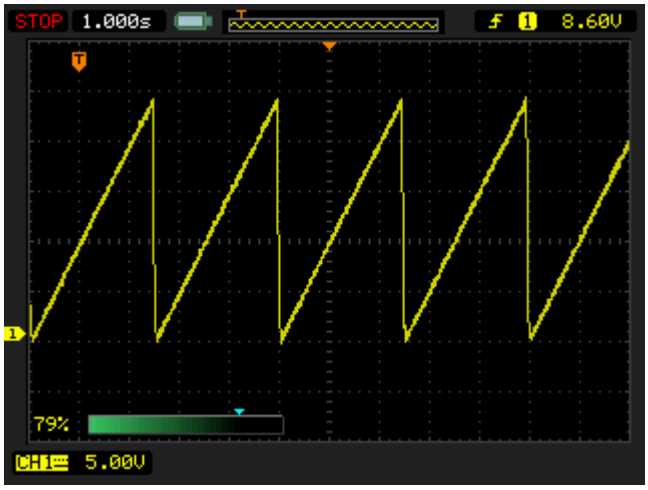
4 EXAMPLES

4.0 EXAMPLE 1: Sawtooth wave

The following script outputs a continuous 0 to 25V sawtooth waveform:

```
rem EXAMPLE 1 - Sawtooth waveform

rem (optional) setup
voltage_setpoint = 0
current_setpoint = 40
output_mode = 1

rem loop label used to create infinite loop
loop:

rem use FOR/NEXT loop to create ramp
for i=0 to 25 step 0.01
let voltage_setpoint = i
wait 1
next i

rem jump back to create next ramp
goto loop
```

Below is a screen shot of the output of the PLS6005040 power supply running the script:

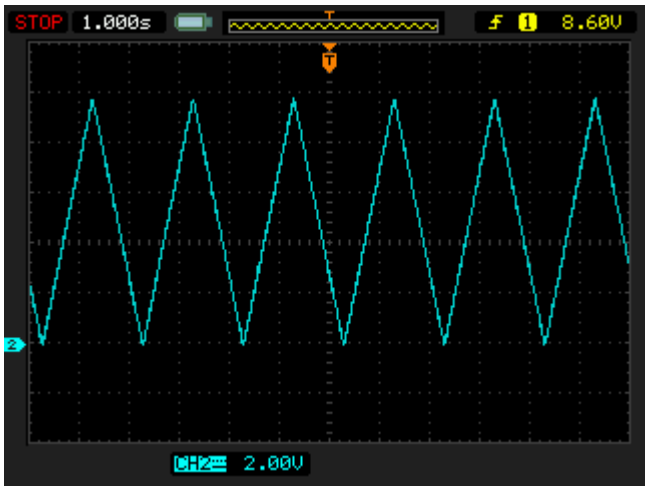


4.1 EXAMPLE 2: Analog output triangle wave

The following script outputs a continuous 0 to 10V triangle waveform from the analog output port of the power supply:

```
rem EXAMPLE 2 - Triangle wave on analog output (0.1V steps)

loop:

rem ramp up
analog_output = 0
for i=0 to 100 step 1
analog_output = analog_output + 0.1
wait 10
next i

rem ramp down
analog_output = 10.0
for i=0 to 100 step 1
analog_output = analog_output - 0.1
wait 10
next i

goto loop
```

Below is a screen shot of the output of the PLS6005040 power supply running the script:



4.2 EXAMPLE 3: Timer triggered output

The following script illustrate that long delays are possible when running scripts:

```
rem EXAMPLE 3 - Timer triggered output

rem (optional) setup
voltage_setpoint = 25
current_setpoint = 20
power_setpoint = 100
output_mode = 0

rem output will turn on 123.456 seconds after start of script
wait 123456
output_mode = 1

rem NOTE: output does NOT turn off when script ends
end
```

NOTE: the 32-bit millisecond clock can implement delays up to 1193 days with a single WAIT statement. Realistically, while reasonably accurate, the 1 millisecond timer cannot be expected to keep accurate time over large numbers of hours and cannot be used as a substitute for a real-time clock.

4.3 EXAMPLE 4: Analog input triggered output

The following script illustrates the use of an analog input port to control the output mode (ON or OFF) of the power supply:

```
rem EXAMPLE 4 - Analog input triggered output

rem (optional) setup
voltage_setpoint = 30
current_setpoint = 10
power_setpoint = 400
output_mode = 0

rem output will turn on when the (voltage) analog input
rem exceeds 2.5V and off when it falls below 1.5V
loop:
if analog_input_voltage > 2.5 then turn_on
if analog_input_voltage < 1.5 then turn_off
wait 1
goto loop

turn_on:
output_mode = 1
wait 1
goto loop

turn_off:
output_mode = 0
wait 1
goto loop
```

4.4 EXAMPLE 5: Arbitrary waveform

The following is an example of an arbitrary waveform generated with a script:

```
rem EXAMPLE 5 - Arbitrary waveform

rem initial conditions
voltage_setpoint = 12
current_setpoint = 40
power_setpoint = 1500
output_mode = 1
wait 500

rem step through peices of waveform
gosub step1
gosub step2
gosub step3
gosub step4
end


rem step 1 - drop to 3V and hold for 250ms then ramp to 6V over 50ms
step1:
voltage_setpoint = 3
wait 250
for i=3 to 6 step 0.06
voltage_setpoint = i
wait 1
next i
return


rem step 2 - output 5 cycles of 4V to 8V sinusoid with frequency of 5Hz
step2:
gosub step2b
gosub step2b
gosub step2b
gosub step2b
gosub step2b
return


rem step 2b - output one cycle
step2b:
voltage_setpoint = 6
wait 1
voltage_setpoint = 6.062821518
wait 1
voltage_setpoint = 6.125581039
wait 1
voltage_setpoint = 6.188216627
wait 1
voltage_setpoint = 6.250666467
wait 1
```

```
voltage_setpoint = 6.31286893
wait 1
voltage_setpoint = 6.374762629
wait 1
voltage_setpoint = 6.436286483
wait 1
voltage_setpoint = 6.497379774
wait 1
voltage_setpoint = 6.557982212
wait 1
voltage_setpoint = 6.618033989
wait 1
voltage_setpoint = 6.67747584
wait 1
voltage_setpoint = 6.736249105
wait 1
voltage_setpoint = 6.794295781
wait 1
voltage_setpoint = 6.851558583
wait 1
voltage_setpoint = 6.907980999
wait 1
voltage_setpoint = 6.963507348
wait 1
voltage_setpoint = 7.018082832
wait 1
voltage_setpoint = 7.07165359
wait 1
voltage_setpoint = 7.124166756
wait 1
voltage_setpoint = 7.175570505
wait 1
voltage_setpoint = 7.225814107
wait 1
voltage_setpoint = 7.274847979
wait 1
voltage_setpoint = 7.322623731
wait 1
voltage_setpoint = 7.369094212
wait 1
voltage_setpoint = 7.414213562
wait 1
voltage_setpoint = 7.457937255
wait 1
voltage_setpoint = 7.500222139
wait 1
voltage_setpoint = 7.541026486
wait 1
voltage_setpoint = 7.580310025
wait 1
voltage_setpoint = 7.618033989
wait 1
voltage_setpoint = 7.654161149
wait 1
voltage_setpoint = 7.688655851
```

```
wait 1
voltage_setpoint = 7.721484054
wait 1
voltage_setpoint = 7.75261336
wait 1
voltage_setpoint = 7.782013048
wait 1
voltage_setpoint = 7.809654105
wait 1
voltage_setpoint = 7.835509251
wait 1
voltage_setpoint = 7.859552972
wait 1
voltage_setpoint = 7.881761538
wait 1
voltage_setpoint = 7.902113033
wait 1
voltage_setpoint = 7.920587371
wait 1
voltage_setpoint = 7.937166322
wait 1
voltage_setpoint = 7.951833524
wait 1
voltage_setpoint = 7.964574501
wait 1
voltage_setpoint = 7.975376681
wait 1
voltage_setpoint = 7.984229403
wait 1
voltage_setpoint = 7.991123929
wait 1
voltage_setpoint = 7.996053457
wait 1
voltage_setpoint = 7.999013121
wait 1
voltage_setpoint = 8
wait 1
voltage_setpoint = 7.999013121
wait 1
voltage_setpoint = 7.996053457
wait 1
voltage_setpoint = 7.991123929
wait 1
voltage_setpoint = 7.984229403
wait 1
voltage_setpoint = 7.975376681
wait 1
voltage_setpoint = 7.964574501
wait 1
voltage_setpoint = 7.951833524
wait 1
voltage_setpoint = 7.937166322
wait 1
voltage_setpoint = 7.920587371
wait 1
```

```
voltage_setpoint = 7.902113033
wait 1
voltage_setpoint = 7.881761538
wait 1
voltage_setpoint = 7.859552972
wait 1
voltage_setpoint = 7.835509251
wait 1
voltage_setpoint = 7.809654105
wait 1
voltage_setpoint = 7.782013048
wait 1
voltage_setpoint = 7.75261336
wait 1
voltage_setpoint = 7.721484054
wait 1
voltage_setpoint = 7.688655851
wait 1
voltage_setpoint = 7.654161149
wait 1
voltage_setpoint = 7.618033989
wait 1
voltage_setpoint = 7.580310025
wait 1
voltage_setpoint = 7.541026486
wait 1
voltage_setpoint = 7.500222139
wait 1
voltage_setpoint = 7.457937255
wait 1
voltage_setpoint = 7.414213562
wait 1
voltage_setpoint = 7.369094212
wait 1
voltage_setpoint = 7.322623731
wait 1
voltage_setpoint = 7.274847979
wait 1
voltage_setpoint = 7.225814107
wait 1
voltage_setpoint = 7.175570505
wait 1
voltage_setpoint = 7.124166756
wait 1
voltage_setpoint = 7.07165359
wait 1
voltage_setpoint = 7.018082832
wait 1
voltage_setpoint = 6.963507348
wait 1
voltage_setpoint = 6.907980999
wait 1
voltage_setpoint = 6.851558583
wait 1
voltage_setpoint = 6.794295781
```

```
wait 1
voltage_setpoint = 6.736249105
wait 1
voltage_setpoint = 6.67747584
wait 1
voltage_setpoint = 6.618033989
wait 1
voltage_setpoint = 6.557982212
wait 1
voltage_setpoint = 6.497379774
wait 1
voltage_setpoint = 6.436286483
wait 1
voltage_setpoint = 6.374762629
wait 1
voltage_setpoint = 6.31286893
wait 1
voltage_setpoint = 6.250666467
wait 1
voltage_setpoint = 6.188216627
wait 1
voltage_setpoint = 6.125581039
wait 1
voltage_setpoint = 6.062821518
wait 1
voltage_setpoint = 6
wait 1
voltage_setpoint = 5.937178482
wait 1
voltage_setpoint = 5.874418961
wait 1
voltage_setpoint = 5.811783373
wait 1
voltage_setpoint = 5.749333533
wait 1
voltage_setpoint = 5.68713107
wait 1
voltage_setpoint = 5.625237371
wait 1
voltage_setpoint = 5.563713517
wait 1
voltage_setpoint = 5.502620226
wait 1
voltage_setpoint = 5.442017788
wait 1
voltage_setpoint = 5.381966011
wait 1
voltage_setpoint = 5.32252416
wait 1
voltage_setpoint = 5.263750895
wait 1
voltage_setpoint = 5.205704219
wait 1
voltage_setpoint = 5.148441417
wait 1
```

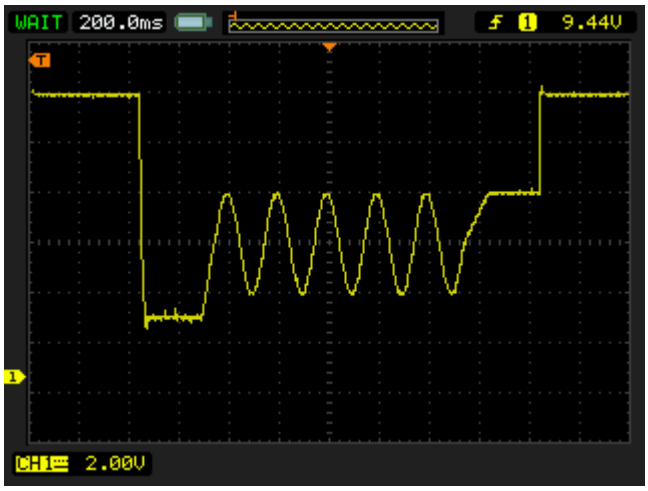
```
voltage_setpoint = 5.092019001
wait 1
voltage_setpoint = 5.036492652
wait 1
voltage_setpoint = 4.981917168
wait 1
voltage_setpoint = 4.92834641
wait 1
voltage_setpoint = 4.875833244
wait 1
voltage_setpoint = 4.824429495
wait 1
voltage_setpoint = 4.774185893
wait 1
voltage_setpoint = 4.725152021
wait 1
voltage_setpoint = 4.677376269
wait 1
voltage_setpoint = 4.630905788
wait 1
voltage_setpoint = 4.585786438
wait 1
voltage_setpoint = 4.542062745
wait 1
voltage_setpoint = 4.499777861
wait 1
voltage_setpoint = 4.458973514
wait 1
voltage_setpoint = 4.419689975
wait 1
voltage_setpoint = 4.381966011
wait 1
voltage_setpoint = 4.345838851
wait 1
voltage_setpoint = 4.311344149
wait 1
voltage_setpoint = 4.278515946
wait 1
voltage_setpoint = 4.24738664
wait 1
voltage_setpoint = 4.217986952
wait 1
voltage_setpoint = 4.190345895
wait 1
voltage_setpoint = 4.164490749
wait 1
voltage_setpoint = 4.140447028
wait 1
voltage_setpoint = 4.118238462
wait 1
voltage_setpoint = 4.097886967
wait 1
voltage_setpoint = 4.079412629
wait 1
voltage_setpoint = 4.062833678
```

```
wait 1
voltage_setpoint = 4.048166476
wait 1
voltage_setpoint = 4.035425499
wait 1
voltage_setpoint = 4.024623319
wait 1
voltage_setpoint = 4.015770597
wait 1
voltage_setpoint = 4.008876071
wait 1
voltage_setpoint = 4.003946543
wait 1
voltage_setpoint = 4.000986879
wait 1
voltage_setpoint = 4
wait 1
voltage_setpoint = 4.000986879
wait 1
voltage_setpoint = 4.003946543
wait 1
voltage_setpoint = 4.008876071
wait 1
voltage_setpoint = 4.015770597
wait 1
voltage_setpoint = 4.024623319
wait 1
voltage_setpoint = 4.035425499
wait 1
voltage_setpoint = 4.048166476
wait 1
voltage_setpoint = 4.062833678
wait 1
voltage_setpoint = 4.079412629
wait 1
voltage_setpoint = 4.097886967
wait 1
voltage_setpoint = 4.118238462
wait 1
voltage_setpoint = 4.140447028
wait 1
voltage_setpoint = 4.164490749
wait 1
voltage_setpoint = 4.190345895
wait 1
voltage_setpoint = 4.217986952
wait 1
voltage_setpoint = 4.24738664
wait 1
voltage_setpoint = 4.278515946
wait 1
voltage_setpoint = 4.311344149
wait 1
voltage_setpoint = 4.345838851
wait 1
```

```
voltage_setpoint = 4.381966011
wait 1
voltage_setpoint = 4.419689975
wait 1
voltage_setpoint = 4.458973514
wait 1
voltage_setpoint = 4.499777861
wait 1
voltage_setpoint = 4.542062745
wait 1
voltage_setpoint = 4.585786438
wait 1
voltage_setpoint = 4.630905788
wait 1
voltage_setpoint = 4.677376269
wait 1
voltage_setpoint = 4.725152021
wait 1
voltage_setpoint = 4.774185893
wait 1
voltage_setpoint = 4.824429495
wait 1
voltage_setpoint = 4.875833244
wait 1
voltage_setpoint = 4.92834641
wait 1
voltage_setpoint = 4.981917168
wait 1
voltage_setpoint = 5.036492652
wait 1
voltage_setpoint = 5.092019001
wait 1
voltage_setpoint = 5.148441417
wait 1
voltage_setpoint = 5.205704219
wait 1
voltage_setpoint = 5.263750895
wait 1
voltage_setpoint = 5.32252416
wait 1
voltage_setpoint = 5.381966011
wait 1
voltage_setpoint = 5.442017788
wait 1
voltage_setpoint = 5.502620226
wait 1
voltage_setpoint = 5.563713517
wait 1
voltage_setpoint = 5.625237371
wait 1
voltage_setpoint = 5.68713107
wait 1
voltage_setpoint = 5.749333533
wait 1
voltage_setpoint = 5.811783373
```

```
wait 1
voltage_setpoint = 5.874418961
wait 1
voltage_setpoint = 5.937178482
wait 1
return


rem step 3 - ramp 6V to 8V over 100ms and hold for 200ms
step3:
for i = 6 to 8 step 0.02
voltage_setpoint = i
wait 1
next i
wait 200
return


rem step 4 - return to initial condition
step4:
voltage_setpoint = 12
return
```

Below is a screen shot of the output of the PLS6005040 power supply running the script:



5 LIMITATIONS

The following limitations apply to scripts that are to be run on the PLS600 power supplies:

a) The total size of the script text (including its name) must be 32768 characters or less. This limit includes the space added for a null-terminator appended to the end of each line of the script as it is downloaded.

b) The total number of elements in the compiled script must be less than 500. Note: each line of a script is compiled into one element except for lines using the keywords FOR, IF, and LET (when using a math operator), which are compiled into two elements. Remark statements and blank lines do not get compiled (i.e. no element is generated).

c) Scripts are executed using a 1 millisecond timer. Up to 10 elements are executed per millisecond. If a WAIT statement is encountered, the execution for that millisecond stops.

d) Script lines must be less than 256 characters in length.

e) Up to 100 variables may be defined (not including reserved variables).

f) Variable names may not exceed 32 characters in length.

g) Up to 100 labels may be declared.

h) Label names may not exceed 32 characters in length.

i) Nested subroutines may be called up to a depth of 10.

j) Latency between the analog inputs and outputs and the corresponding script reserved variables can be up to 3 milliseconds.

k) Variables are implemented as 32-bit floating-point numbers. It is the user's responsibility to be cognizant of the accuracy limitations of floating point variables and to code their scripts accordingly.

6 STATUS AND ERRORS

When a script is downloaded it is not checked for errors until it is compiled. A script is compiled each time an attempt is made to run the script. If any errors are found during compilation, the following message is displayed briefly on the front panel:

It is the user's responsibility to review the script to determine the cause of the error.