Hardkernel SmartPower3 SCPI manual

Lukáš Říha

July 30, 2023

Contents

1 Introduction

SmartPower3 is a small power supply designed to provide power supply to up to two connected devices and measure and log the connected devices power consumption. The first firmware versions allowed control of the SmartPower3 via devices buttons, rotary encoder and display. As the system became more mature, some users showed interest in remote control of the power supply. New versions added more refined functionality and SCPI (Standard Commands for Programmable Instruments) capability - effectively enabling (at least partial) remote control of the SmartPower3. This manual tries to describe the commands available in SCPI device mode.

For more information regarding SCPI and its usage please see the online specification and description available at:
http://www.ivifoundation.org/docs/scpi-99.pdf
and other online resources.

2 Device modes

The SmartPower3 device can basically be used in two modes - the default original "buttons and screen" mode and SCPI mode. The default (original) mode tries to be compatible in function with the previous firmware version, while SCPI mode allows programming of the device via serial using SCPI commands and (like the default mode) optionally output logging information to WiFi connected UDP target.

The mode is changed via screen and encoder, on the second device screen, near the bottom. The option is (surprisingly) called `Mode` and allows setting two options:

1. Default - provides re-worked two level serial menu that enables setting SmartPower3 functionality.
2. SCPI - reserves serial connection for SCPI control of the device.

3 Commands

1. IEEE Mandated Commands

   These commands are required in any SCPI implementation (SCPI std V1999.0 4.1.1).

   1.1 `*CLS`
   Command clears all registers and error queues.

   1.2 `*ESE`

   1.3 `*ESE?`

   1.4 `*ESR?`

   1.5 `*IDN?`

   1.6 `*OPC`

   1.7 `*OPC?`

   1.8 `*RST`
   Device reset - return device to defined known state. Turns output voltage off and stops logging output.

   1.9 `*SRE`

   1.10 `*SRE?`

   1.11 `*STB?`

   1.12 `*TST?`

   1.13 `*WAI"`

2. Required SCPI Commands

   Commands required by Required SCPI commands (SCPI std V1999.0 4.2.1)

   2.1 `SYSTem:ERRor[:NEXT]?`
   Query removes the last error from error buffer and reports it. Repeatedly calling this query eventually causes the error buffer to become empty.

   2.2 `SYSTem:ERRor:COUNt?`
   Query reports number of errors since device start-up or last device clear (Please see *CLS command).

   2.3 `SYSTem:VERSion?`
   Query that reports SCPI standard version this device should adhere to.

3. Non-Required SCPI Commands

   This section contains commands specific to the device and query or control its main functionality.

   3.1 `SYSTem:CAPability?`
   Some desc.

   3.2 `SYSTem:COMMunicate:NETwork:MAC?`
   This query returns the MAC address of the power supply. MAC address consist of two number groups: the first three bytes are known as the Organizationally Unique Identifier (OUI), which is distributed by the IEEE, and the last three bytes are the device's unique serial number. The six bytes are separated by hyphens. The MAC address is unique to the instrument and cannot be altered by the user.
   Return Param <XX-XX-XX-YY-YY-YY>

   3.3 `SYSTem:COMMunicate:NETwork:ADDRess`
   This command sets the static address of the power supply.

   3.4 `SYSTem:COMMunicate:NETwork:ADDRess?`
   Queries the static IP address of the power supply.

3.5 `SYSTem:COMMunicate:NETwork:GATE`
This command sets the Gateway IP address of the power supply. Gateway IP address is represented with 4 bytes each having a range of 0-255 separated by dots.

3.6 `SYSTem:COMMunicate:NETwork:GATE?`
Queries the Gateway IP address of the power supply.

3.7 `SYSTem:COMMunicate:NETwork:SUBNet <string>`
This command sets the subnet IP Mask of the power supply.

3.8 `SYSTem:COMMunicate:NETwork:SUBNet?`
Queries the value of manually set network subnet mask.

3.9 `SYSTem:COMMunicate:NETwork:DHCP`
This command sets the DHCP operating mode of the Ethernet module. If DHCP is set to 1, the module will allow its IP address to be automatically set by the DHCP server on the network. If DHCP is set to 0, the default IP address is set according to .

3.10 `SYSTem:COMMunicate:NETwork:DHCP?`
This query reports the DHCP operating mode currently set.

3.11 `SYSTem:COMMunicate:SOCKet:ADDRess`
Command sets logging target IP address (UDP logging server address).

3.12 `SYSTem:COMMunicate:SOCKet:ADDRess?`
Queries logging target IP address (UDP logging server address).

3.13 `SYSTem:COMMunicate:SOCKet:PORT`
Command sets logging target port (UDP logging server port).

3.14 `SYSTem:COMMunicate:SOCKet:PORT?`
Queries logging target port (UDP logging server port).

3.15 `SYSTem:COMMunicate:SOCKet:CONNect`
Command connects to the target socket. In case of SmartPower3 and UDP WiFi logging, this basically means connecting to WiFi network, the same way one can connect by moving cursor over WiFi icon when disconnected and pressing encoder button.

3.16 `SYSTem:COMMunicate:SOCKet:DISConnect`
Command disconnects to the target socket. In case of SmartPower3 and UDP WiFi logging, this basically means disconnecting from WiFi network, the same way one can disconnect by moving cursor over WiFi icon when connected and pressing encoder button.

3.17 `SYSTem:COMMunicate:SOCKet:FEED <data_handle>`
Connect or disconnect the logging output to connected socket - WiFi. The data handle parameter can be one of two values:

`LOG` - connect the logging output

`NONE` - disconnect the logging output

3.18 `SYSTem:COMMunicate:SOCKet:FEED?`
Report what data source is connected to the logging output. The result can be one of the following values:

`LOG`

`NONE`

3.19 `SYSTem:COMMunicate:SERial:FEED <data_handle>`
Connect the logging output to connected serial port. The data handle parameter can be one of two values:

`LOG` - connect the logging output

`NONE` - disconnect the logging output

3.20 `SYSTem:COMMunicate:SERial:FEED?`
Report what data source is connected to the logging output. The result can be one of the following values:

`LOG`

NONE

3.21 SYSTem:COMMunicate:RLSTate

Sets the device operation mode. Equivalent to the Mode screen menu item. Acceptable options are:

LOCal - Corresponds to the Default menu selection.

REMote - Corresponds to the SCPI menu selection.

3.22 SYSTem:COMMunicate:RLSTate?

Queries the selected device operation mode. Equivalent to the Mode screen menu item. Possible responses are:

LOCal - Corresponds to the Default menu selection.

REMote - Corresponds to the SCPI menu selection.

3.23 FETCh[:SCALar]:VOLTage[:DC]? [expected_value, [resolution,]] <channel_list>

Command allows reading voltage on channel specified by <channel_list>. Parameter expected_value has no use in this case and is included for reasons of compatibility. Parameter resolution allows specifying the resolution of the result. If ommited, the result is returned in Volts. Parameter <channel_list> allows specifying which channel(s) result should be read. The order of the channels is important - results are returned in that order.

If all parameters are ommited, the command will return value for channel 1, in Volts.

Example 1:

FETCh:VOLTage? 1 V, 0.001V, (@1:3)

will return voltage read on all three channels (including the power supply supply channel), in milliVolts, in order of channel 1, 2, 3.

Example 2:

FETCh:VOLTage? (@3,1)

will return voltage read on channels 3 and 1 (in that order), in units of Volts.

3.24 FETCh[:SCALar]:CURRent[:DC]? [expected_value, [resolution,]] <channel_list>

Command allows reading of current on channel specified by <channel_list>, averaged over 2 samples. If resolution is not specified, the value is returned in Amperes. Parameter <channel_list> allows specifying which channel(s) result should be read. The order of the channels is important - results are returned in that order.

If all parameters are ommited, the command will return value for channel 1, in Amperes.

Example 1:

FETCh:CURRent? 1 A, 0.001A, (@1:3)

will return current read on all three channels (including the power supply supply channel), in milliAmperes, in order of channel 1, 2, 3.

Example 2:

FETCh:CURRent? (@3,1)

will return current read on channels 3 and 1 (in that order), in units of Amperes.

3.25 FETCh[:SCALar]:POWer[:DC]? [expected_value, [resolution,]] <channel_list>

Command allows reading power on channel specified by <channel_list>. Parameter expected_value has no use in this case and is included for reasons of compatibility. Parameter resolution allows specifying the resolution of the result. If ommited, the result is returned in Watts. Parameter <channel_list> allows specifying which channel(s) result should be read. The order of the channels is important - results are returned in that order.

If all parameters are ommited, the command will return value for channel 1, in Watts.

Example 1:

FETCh:POWer? 1 V, 0.001V, (@1:3)

will return power read on all three channels (including the power supply supply channel), in milliWatts, in order of channel 1, 2, 3.

Example 2:

`FETCh:POWer? (@3,1)`

will return power read on channels 3 and 1 (in that order), in units of Watts.

3.26 `[SOURce#]:CURRent <numeric_value>`

Sets output current on source `#`. If `#` is ommited, the default value is 1.
`<numeric_value>` is the value that should be set on the device. Default unit is Amperes. Other possible values include `MIN|MAX`, which set minimum and maximum persmissible value, respectively. Another option, if you decide to use `A` unit is to specify units as fractions, such as `mA` (milliAmperes) or `UA` (microAmperes).

3.27 `[SOURce#]:CURRent?`

Querries the output current set on channel `#`. Note that this differs from the FETCh series of commands in that no measurements are taken and value set by `[SOURce#]:CURRent` is returned.
Returns value in Amperes.

3.28 `[SOURce#]:VOLTage <numeric_value>`

Sets output voltage on source `#`. If `#` is ommited, the default value is 1.
`<numeric_value>` is the value that should be set on the device. Default unit is Volts. Other possible values include `MIN|MAX`, which set minimum and maximum persmissible value, respectively. Another option, if you decide to use `V` unit is to specify units as fractions, such as `mV` (milliVolts) or `UV` (microVolts).

3.29 `[SOURce#]:VOLTage?`

Querries the output voltage set on channel `#`. Note that this differs from the FETCh series of commands in that no measurements are taken and value set by `[SOURce#]:VOLTage` is returned.
Returns value in Amperes.

3.30 `OUTPut#[:STATe] <parameter>`

This command turns the output channel `#` on or off. If `#` is ommited, the command defaults to channel number 1.
Possible parameter value is one of `ON|1|OFF|0`.

3.31 `OUTPut#[:STATe]?`

This query returns the output states of channel `#`. Returns 0 or 1.

4 Python Examples

The power supply in SCPI mode can also be controlled by various VISA implementations. This enables the user to programmatically perform and repeat various test and measurements, that might otherwise be quite complicated or outright impossible with sufficient precision.

The following is a simple example of how the device might be used.

More information on how to use PyVISA can be found at
https://pyvisa.readthedocs.io/en/latest/introduction/index.html.

4.1 Prerequisites

It is expected that you have Python3, PyVISA and backed installed. We are going to use PyVISA-py for the backend.

The exact steps how to install these prerequisites depend on your selection of operating system and its version. For example, most Linux distributions will have Python installed by default, so its installation might be skipped.

Once Python is installed, you could, for example, install the remaining dependencies for the current user by running the following in terminal:

```
pip install -U pyvisa pyvisa-py
```

4.2 Connecting to the Power Supply

The next step is to connect to the SmartPower3 power supply. To do so, please follow these steps:

1. Connect SmartPower3 to your computer using serial cable.
2. On the setting screen, set SmartPower3 to `SCPI` mode.
3. Open your terminal application and start Python interpreter by typing:
   `python`
4. Import PyVISA and connect by typing the following

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager()
>>> rm.list_resources()
('ASRL/dev/ttyUSB0::INSTR')
>>> inst = rm.open_resource('ASRL/dev/ttyUSB0::INSTR')
>>> print(inst.query('*IDN?'))
```

   The `('ASRL/dev/ttyUSB0::INSTR')` string will most likely be different in your case. You might even have more devices listed, depending on your hardware. That is OK, just select the one that corresponds to your serial connection.

   The last command (`print(inst.query("*IDN?"))`) should result in the terminal printing out the power supply identification, which should look similar to

   `Hardkernel Co Ltd,SmartPower3,<mac_address>,Build date: Jul 19 2023 19:32:24`, where `<mac_address>` will be the WiFi module MAC address (a string similar to `94:3C:C6:CC:AA:78`) and the Build date will reflect your current firmware version.

5. In case the last command didn't produce the desired result and you got timeout error response, there are couple things you can check to remedy the situation:

   5.1 Check that the device is in SCPI mode.

   5.2 Check device serial baud rate. It is quite possible that your SmartPower3 is set to a higher baud rate than 9600, which is PyVISA default. To set correct baud rate, you can use the following command:

```
inst.baud_rate = 115200
```

   where `115200` should be changed to the actual value set on your SmartPower3.

   5.3 It is possible that the response was followed by an empty line. In such a case, there is a mismatch between line endings returned by the device and line endings that PyVISA expects. That can be remedied by issuing the following command:

```
inst.read_termination = '\r\n'
```

   For more troubleshooting info please see PyVISA documentation available at https://pyvisa.readthedocs.io/en/latest/introduction/communication.html#.

4.3 Doing Something Usefull

Once you have succeded in issuing the identification command and getting a proper response, we can get to something more useful.

4.3.1 Doing a Voltage Sweep

Let's suppose that you have just finished a construction of a new hardware device that is supposed to be powered by 12 Volts DC. But you also want to check how the device behaves at a lower voltage, gradually bringing the voltage up from 5V to the full 12V. That could be achieved by the following commands:

```
# by now, you have pyvisa imported and are connected to the device
import time
start_voltage = 5
end_voltage = 12
inst.write(f'source1:volt {start_voltage}')  # the starting voltage on channel 1
# uncommenting the next line turns channel 1 on and the sweep is done under power
# it is advisable to first check the functionality of your program
# with the output turned off, in case there is a mistake
# inst.write('output1 on')
while(start_voltage <= end_voltage):
    print(f'start_voltage = {start_voltage}')
    inst.write(f'source1:volt {start_voltage}')
    start_voltage += 1  # increase the value of start_voltage by 1
    time.sleep(2)  # wait 2 seconds
```

The example above is rather rudimentary and does not use the possibilities of the device to its full extent.

If you wanted to set values on a finer scale than whole Volts and not use Python float data type - which is advisable, as it can lead to unpredictable results - you could modify the above script as follows:

```
# by now, you have pyvisa imported and are connected to the device
import time
start_voltage = 3000  # the value represented in milliVolts
end_voltage = 5000  # the value represented in milliVolts
inst.write(f'source1:volt {start_voltage}mV')  # note the added unit of mV
# uncommenting the next line turns channel 1 on and the sweep is done under power
# it is advisable to first check the functionality of your program
# with the output turned off, in case there is a mistake
# inst.write('output1 on')
while(start_voltage <= end_voltage):
    print(f'start_voltage in Volts = {start_voltage/1000}')
    inst.write(f'source1:volt {start_voltage}mV')  # note the added unit of mV
    # increase the value of start_voltage by 20mV, which is the smallest
    # possible step in this voltage range
    start_voltage += 20
    time.sleep(2)  # wait 2 seconds
```

4.3.2 Plotting Power Data From Measurements on Channel 1

The `contrib/SCPI_examples` directory of Smartpower3 git repository contains file `sp3-scpi-datalogger.py` which does exactly what the subtitle advertises.

But before you use it, you should be aware of a couple of important points:

1. Check what voltage your device operates on. The file sets output voltage to 5.1V (required for Raspberry Pi 4), so if your device operates on other voltage, change `device_input_voltage` variable in the `run` function to the required value.
2. The script expects the device to be connected to the first (left) channel.
3. The script expects that you use Serial connection to your SmartPower3 and it lets you choose your serial device.
4. It asks you how long you want to run the logging - plese enter numeric value, seconds.

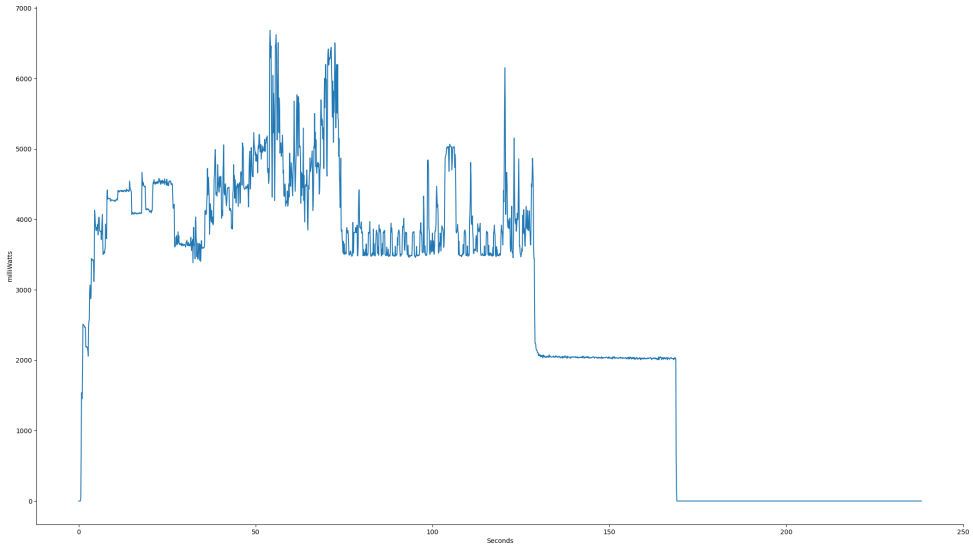

Figure 1: Script output

5. Before you run the script with your device connected to SmartPower3, please run the script without the device to make sure everything is working as expected, the power is set to correct level etc. Doublecheck everything.

6. The script saves logged data to intermediate .csv file in the same directory as the script. You might want to use that file for further analysis.

7. The scrip is pretty basic and definitely can be improved upon - don't hesitate to modify it to suit your needs. But please doublecheck before powering your device.

If everything goes well, the script will output window with power plot similar to Figure 1.

If things don't go as planned, please check the following:

1. Check that the device mode is set to SCPI. If not and you try to use PyVisa, you will most likely get complains about read method not having read termination and `>>> Unknown Command <<<`.

2. Check the device you try to power is connected to the correct channel.