PI Firmware Commands/Queries

demo software commands

May 5, 2020

TABLE OF CONTENTS

1.0 PI FIRMWARE 3

1.1 Source Repository, Compiling, and Executable 3

1.2 Metrics Captured 3

1.3 Capture Rate(s) 4

2.0 SOFTWARE SPECIFICATION 4

2.1 Timestamp, Date-Time Naming Convention 4

2.2 BlackBox Naming Convention 4

2.3 Log File 5

2.3.1 Log File Location:..5

2.3.2 Log File Structure..5

2.3.3 Log File Path..6

2.3.4 Log File Upload..7

3.0 SOCKET INTERFACE SPECIFICATION 8

3.1 Socket Instantiation 8

3.2 Socket Command Structure 8

3.3 Query List 8

3.4 Command List 11

3.5 Client Connection 15

4.0 BLACKBOX SOFTWARE & SYSTEMD 16

1.0 PI FIRMWARE

1.1 Source Repository, Compiling, and Executable

The source code has all been written in C, for speed of execution and compatibility with the GPIO library used on the pi (bcm2835). The github repo where the source code can be accessed is:

https://github.com/johnpolakow/PI_Blackbox

The directory for code executing on the pi is "**PI_Firmware**". To get a sample client application, which already works with the pi firmware, look in the "**Client_Code"** folder.

For compiling the firmware on the pi, here is the command (make sure bcm2835 library has been installed if this is a fresh OS):

```
gcc -g -o exec Execution_Engine.c -l pthread -l bcm2835 -l m
```

If debugging capabilities are desired use this command:

```
gcc -g -fsanitize=address -static-libasan -o exec Execution_Engine.c -l pthread -l
bcm2835 -l m
```

The client code is easy to compile:

```
gcc -o client client.c
```

IF EXECUTING MANUALLY, MUST EXECUTE PROGRAM WITH SUDO

Note: the binary executable for the program must be executed on the raspberry pi with the "sudo" command. The pi requires programs which access the GPIO pins to have elevated access. If sudo is not used there will be a segmentation fault.

```
sudo ./exec
```

Note2: By default the pi has been configured to automatically start execution of the program at boot, MANUAL EXECUTION IS NOT NECESSARY. Under normal circumstances it is not necessary to execute the program manually. If you desire to for testing or debug purposes, make sure to preface the command with sudo.

1.2 Metrics Captured

The metrics directly measured by the Pi Blackbox are:

- Power delivered to the Cooler Driver Board (Volts, mA, Watts)
- Dewar Temp Diode Voltage, and associated LUT temperature (Volts, Kelvins)
- Power delivered to heat load resistor in the Dewar (V, mA, mW)
- Heat Load State (On/Off)
- Relay State for Cooler Driver (On/Off)
- Temperature of x3 attached K-type thermocouples (Celsius)

1.3 Capture Rate(s)

Each metric once per second. Every second a complete iteration of the various metrics are read and saved to storage. The pi can be queried over the socket interface as often as necessary to retrieve the metrics. If querying more than once per second, examine the timestamp in the metrics readout to ascertain whether a duplicate read has been sent. The pi responds with whatever the latest data is.

The Pi firmware captures each metric sequentially, then writes the data points to file. Each datalog point of the above metrics is paired with:

- Timestamp (MM/DD/YYYY hh:mm:ss)
- Elapsed Running Time when sample taken

2.0 SOFTWARE SPECIFICATION

2.1 Timestamp, Date-Time Naming Convention

A timestamp shall be represented in the format:
[MM-dd-YYYY]_[hh.mm.ss]

MM	2 digit month
dd	2 digit day of month
YYYY	4 digit year
hh	2 digit hour
mm	2 digit minute
ss	2 digit second

* 2 digit refers to the convention of including a leading zero if the value is less than ten.
** To avoid AM/PM confusion, the two digit hour shall be in the 24 hour based format. For example the hour 6:01:35 PM is represented as 18:01:35

Example:
Timestamp for Jan. 27th 2020 @ 3:18:34 PM is- **01/27/2020_15.18.34**

2.2 Log File

2.2.1 Log File Structure

// filename: XXXXXX

```
## PI_BB LOG FILE ##
firmware version XXXXX, PCB version
PI_BB: [blackbox#]
CCC:            CCC_ID
Compressor:     Compressor_ID
```

Dewar:	**Dewar_ID**
Displacer:	**Displacer_ID**
Test Execution:	**[Manual] or [profile filename]**
Test_Type:	**[default/Performance/Lifetime/BurnIn]**
Log Start:	**[Timestamp]**

[COLUMN LEGEND]

DATA	**[TIMESTAMP]**
COMMAND	**[TIMESTAMP]**
DATA	**[TIMESTAMP]**
STOP	**[TIMESTAMP]**

To avoid confusion and misinterpretation, the units of measure are included in both the Legend and immediately following the value of each metric. For instance a voltage is recorded as:

12.013V

2.2.2 Log File Path

If a USB drive is inserted, the Linux kernel has been configured to mount it to:

/home/pi/USB_drive/

The flash drive is the default storage device if present. The Log file path is then:

```
/home/pi/USB_drive/Logs/filename.log
```

If a USB flash drive is not detected, the micro SD card will be used to store the data, and the path will be:

```
/home/pi/Logs/filename.log
```

The log file is automatically created and stored when the “>start_test” command is issued.

Note: When configuring the software to auto-launch at boot, it broke the save to USB functionality, so all Logs are being stored on the pi at:

```
/home/pi/Logs/filename.log
```

USB save functionality will be restored in the next software release.

2.2.3 Log File Upload

Currently the log file upload to hub is **not implemented**. The logs can be accessed manually however.

3.0 SOCKET INTERFACE SPECIFICATION

3.1 Socket Instantiation

The socket hosted on the pi opens and listens on port **2020**, waiting for any client to connect. The client can connect to the pi over Ethernet or Wifi, it is interface agnostic. The Pi listens on port 2020 for any incoming connection. The pi waits for clients to connect to it, the pi does not initiate any connections. The socket is configured as TCP/IP protocol, with the following socket parameters:

```
.ai_family = AF_INET;
.ai_socktype = SOCK_STREAM;
.ai_flags = AI_PASSIVE;
```

A sample client program has been written and is included to illustrate how to use the interface. The pi will always send a response to any client instruction.

3.2 Socket Command Structure

All “commands” or instructions will originate from the hub computer. There are two types of instructions sent to a PI_BB:

- Get Instruction– will refer to as a ‘Query’
- Set Instruction – referred to as a ‘Command’

Query Instructions must begin with a ‘**?**’.

Command Instructions must begin with a ‘**>**’.

If an instruction is sent without the proper prefix of ‘**?**’ or ‘**>**’, or there is a typo in the instruction, the parser responds with an error message.

The commands/queries are not case sensitive, even though the commands are shown as all capital letters in this document.

3.3 Query List

Not all queries are implemented yet. The non functional queries are greyed out.

?status Return the following parameters:

hostname

eth0 IP address

wlan0 address

flash drive disk usage

SD card disk usage

DUT ID for:

compressor

DDCA

displacer

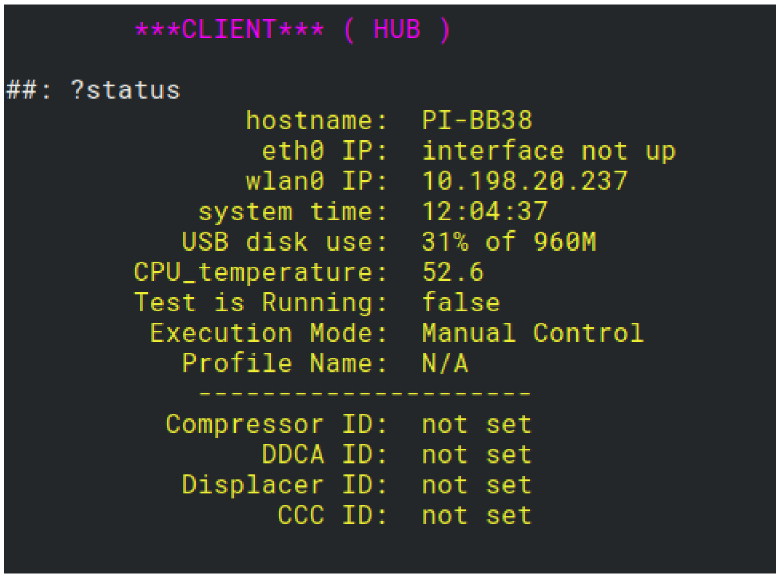
CCC

Pi CPU temperature

Test status (whether executing a test or not)

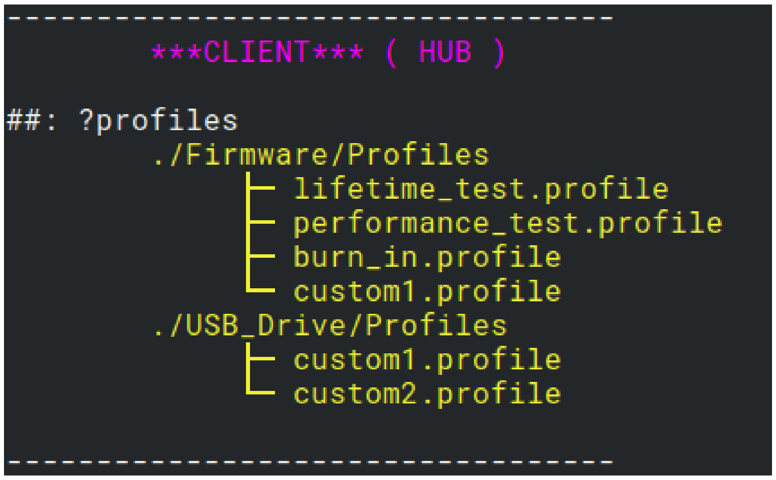
Execution set to manual or profile

Current time & date on PI



```
        ***CLIENT*** ( HUB )

##: ?status
               hostname:  PI-BB38
                eth0 IP:  interface not up
               wlan0 IP:  10.198.20.237
            system time:  12:04:37
           USB disk use:  31% of 960M
        CPU_temperature:  52.6
        Test is Running:  false
         Execution Mode:  Manual Control
           Profile Name:  N/A
            ---------------------
          Compressor ID:  not set
                DDCA ID:  not set
           Displacer ID:  not set
                 CCC ID:  not set
```

?profiles list the profiles currently stored on SD card



```
-----------------------------------
        ***CLIENT*** ( HUB )

##: ?profiles
        ./Firmware/Profiles
             ├─ lifetime_test.profile
             ├─ performance_test.profile
             ├─ burn_in.profile
             └─ custom1.profile
        ./USB_Drive/Profiles
             ├─ custom1.profile
             └─ custom2.profile

-----------------------------------
```

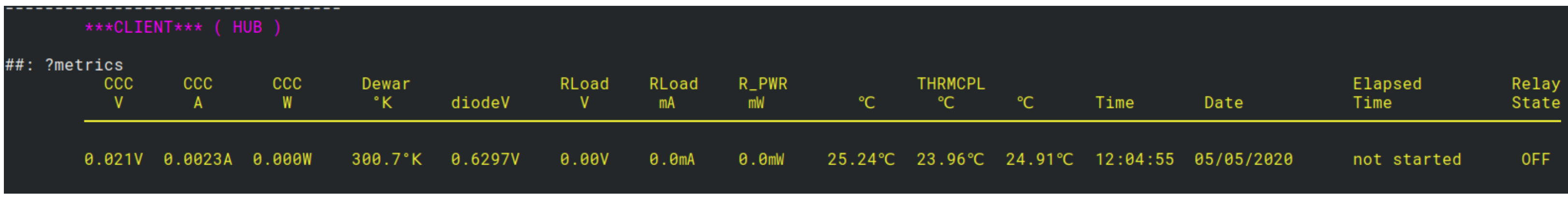
?logs list the logs currently stored on SD card

?profile list the current selected profile, if any

?show [filename] print the contents of the file. Could be a profile, log, config file, etc.

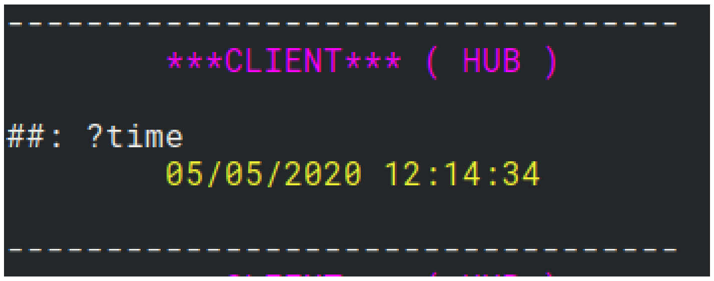
PI_Blackbox Software Behavioral Document

?metrics Returns a snapshot of the current measurements.



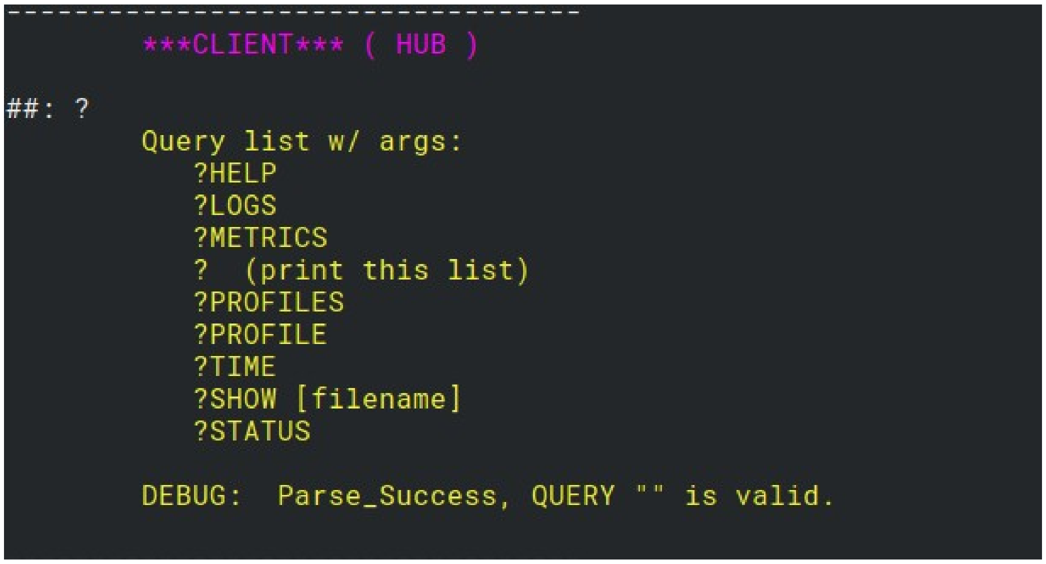
```
----------------------------------
        ***CLIENT*** ( HUB )

##: ?metrics
          CCC      CCC       CCC       Dewar                 RLoad     RLoad     R_PWR               THRMCPL                                                 Elapsed          Relay
           V        A         W         °K       diodeV        V         mA        mW         ℃         ℃         ℃        Time        Date               Time             State
        ____________________________________________________________________________________________________________________________________________________________________________

        0.021V  0.0023A  0.000W    300.7°K    0.6297V     0.00V     0.0mA     0.0mW     25.24℃   23.96℃   24.91℃   12:04:55   05/05/2020         not started      OFF
```

?time Current date-time PI thinks it it right now



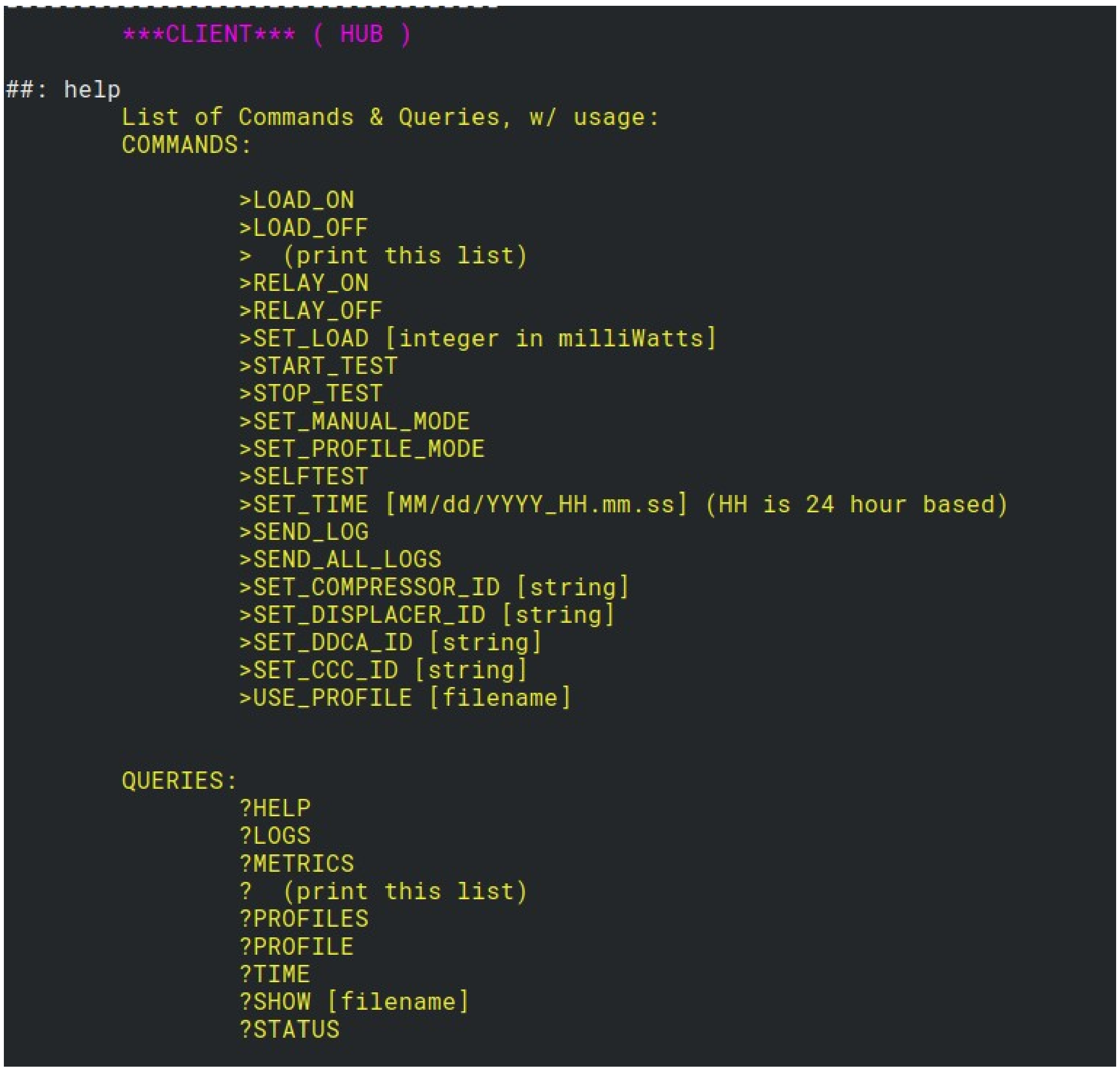
```
------------------------------------
        ***CLIENT*** ( HUB )

##: ?time
        05/05/2020 12:14:34

------------------------------------
```

? If no argument is supplied with the query instruction, the list of query instructions is printed



```
----------------------------------
        ***CLIENT*** ( HUB )

##: ?
        Query list w/ args:
           ?HELP
           ?LOGS
           ?METRICS
           ?  (print this list)
           ?PROFILES
           ?PROFILE
           ?TIME
           ?SHOW [filename]
           ?STATUS

        DEBUG:  Parse_Success, QUERY "" is valid.
```

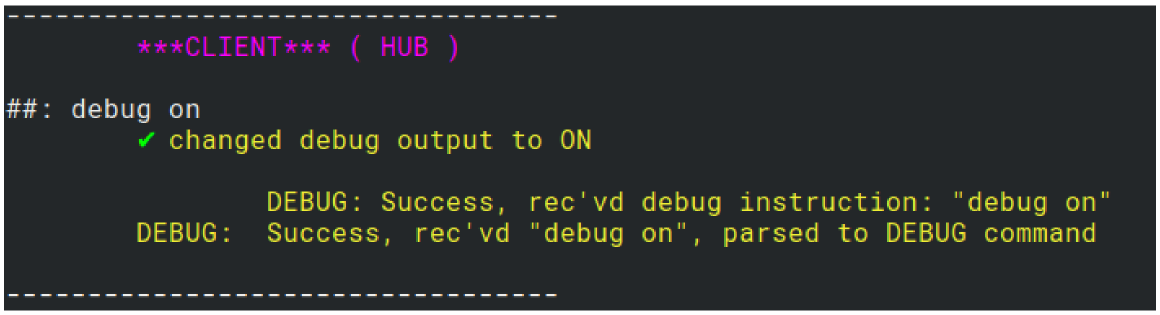
help Prints query instructions and command instructions with usage info. Note this command does not require a '?' or '>'.



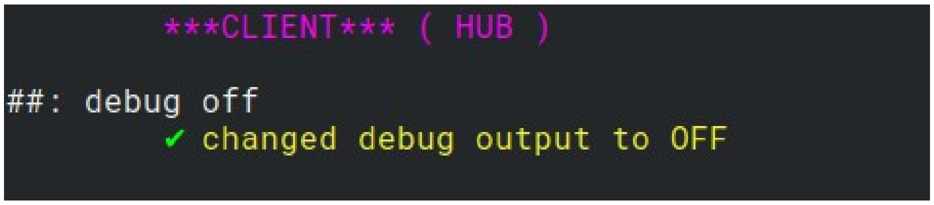
```
        ***CLIENT*** ( HUB )

##: help
        List of Commands & Queries, w/ usage:
        COMMANDS:

                >LOAD_ON
                >LOAD_OFF
                >  (print this list)
                >RELAY_ON
                >RELAY_OFF
                >SET_LOAD [integer in milliWatts]
                >START_TEST
                >STOP_TEST
                >SET_MANUAL_MODE
                >SET_PROFILE_MODE
                >SELFTEST
                >SET_TIME [MM/dd/YYYY_HH.mm.ss] (HH is 24 hour based)
                >SEND_LOG
                >SEND_ALL_LOGS
                >SET_COMPRESSOR_ID [string]
                >SET_DISPLACER_ID [string]
                >SET_DDCA_ID [string]
                >SET_CCC_ID [string]
                >USE_PROFILE [filename]


        QUERIES:
                ?HELP
                ?LOGS
                ?METRICS
                ?  (print this list)
                ?PROFILES
                ?PROFILE
                ?TIME
                ?SHOW [filename]
                ?STATUS
```

3.4 Command List

DEBUG [ON/OFF] Turn additional debug information on or off. This may help when writing software to interface with the pi. Note the ">" character is not used to prepend the command with.

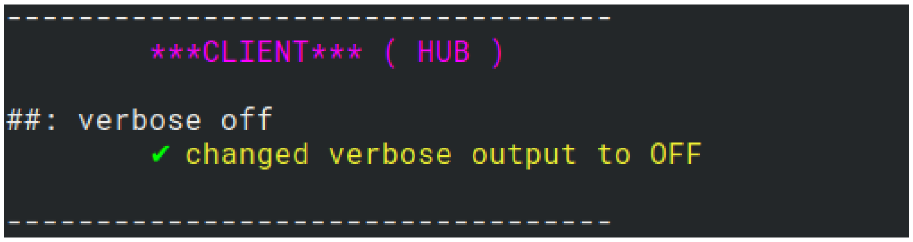


```
-----------------------------------
        ***CLIENT*** ( HUB )

##: debug on
        ✔ changed debug output to ON

                DEBUG: Success, rec'vd debug instruction: "debug on"
        DEBUG:  Success, rec'vd "debug on", parsed to DEBUG command

-----------------------------------
```



```
        ***CLIENT*** ( HUB )

##: debug off
        ✔ changed debug output to OFF
```

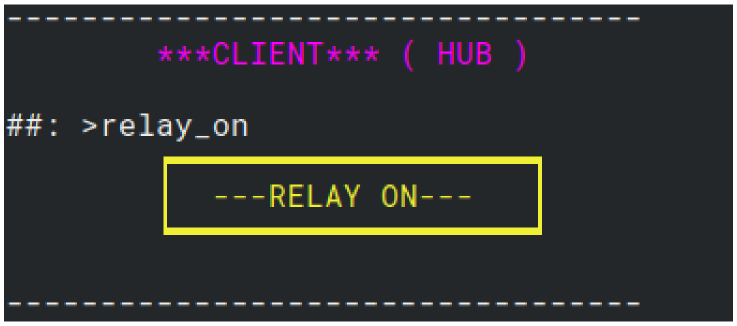
VERBOSE [ON/OFF] Turn verbose responses on or off. Verbose responses include the column headers for the metrics command. Note the ">" character is not used to prepend the command with.



```
----------------------------------
        ***CLIENT*** ( HUB )

##: verbose off
        ✔ changed verbose output to OFF

----------------------------------
```

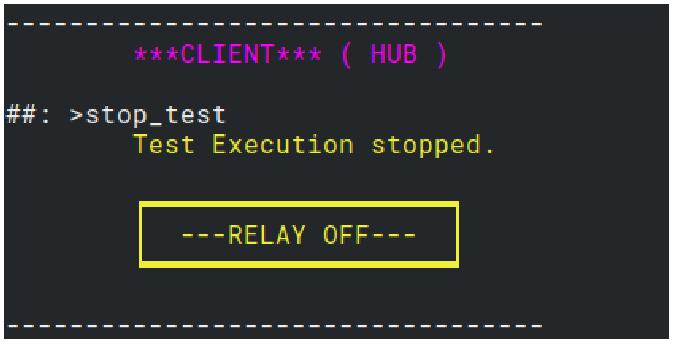
>START_TEST Starts test execution, within the confines of selected test parameters. At present, because profile functionality is not complete yet, the >start_test command only causes the log file to begin recording.

>RELAY_ON Turn on the relay.



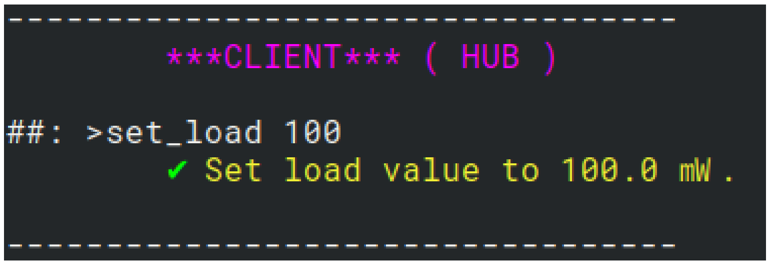
```
----------------------------------
        ***CLIENT*** ( HUB )

##: >relay_on
          ---RELAY ON---

----------------------------------
```

>RELAY_OFF Turn off the relay.



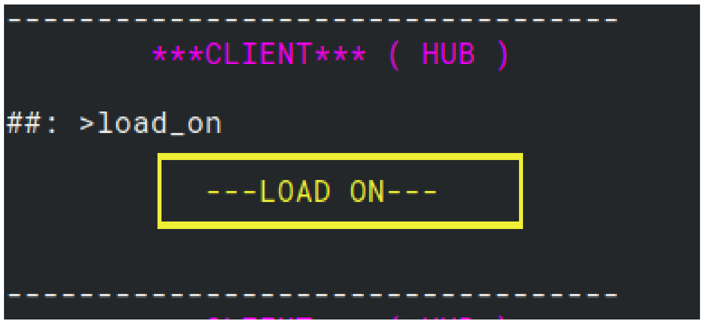
```
----------------------------------
        ***CLIENT*** ( HUB )

##: >stop_test
        Test Execution stopped.

          ---RELAY OFF---

----------------------------------
```

>SET_LOAD [value] Set the load resistor power dissipation in **milliWatts**. The load is not turned on with this command, just set to the value of the supplied argument. When the LOAD_ON command is issued, then power will be applied at the load dissipation value supplied in this command.



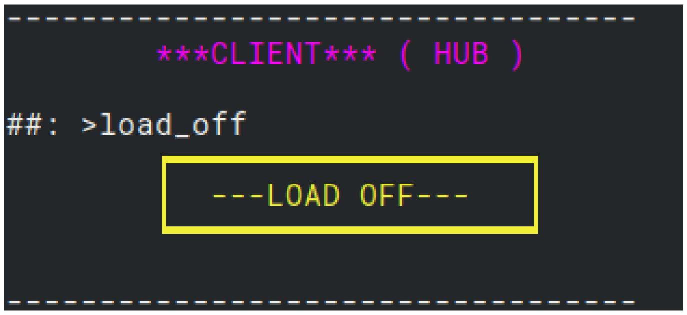
```
----------------------------------
        ***CLIENT*** ( HUB )

##: >set_load 100
        ✓ Set load value to 100.0 mW.

----------------------------------
```

>LOAD_ON Set the load resistor current to the value prescribed by the SET_LOAD command. Default load current is 0 if the load has not been set yet.



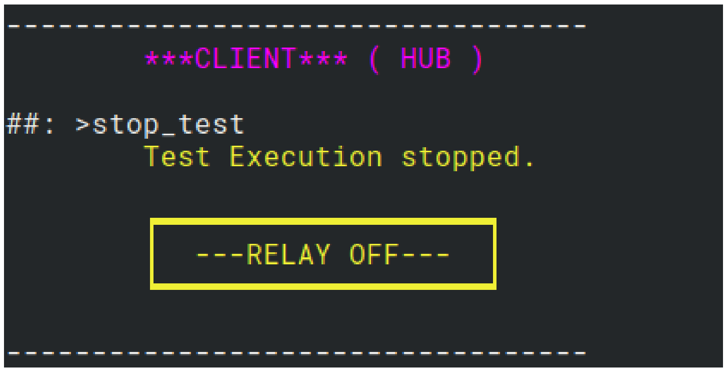
```
----------------------------------
        ***CLIENT*** ( HUB )

##: >load_on
         ---LOAD ON---

----------------------------------
```

>LOAD_OFF Turn load resistor current off. This command is only valid if a test is currently executing AND test execution is set to manual.



```
----------------------------------
        ***CLIENT*** ( HUB )

##: >load_off
         ---LOAD OFF---

----------------------------------
```

>STOP_TEST Turn off cooler relay. Turn off load resistor current. Stop data-logging.



```
----------------------------------
        ***CLIENT*** ( HUB )

##: >stop_test
        Test Execution stopped.

         ---RELAY OFF---

----------------------------------
```

>SET_MANUAL_MODE Set test execution to manual. Default value is manual mode.

>SET_PROFILE_MODE Set test execution to the current profile setting. This command works, however running a profile is not implemented yet.

>USE_PROFILE [profile] The file given as an argument is now selected as the profile to be executed. Verify the profile exists. The profile file is parsed to validate it is acceptable. Begin test execution by issuing START_TEST.

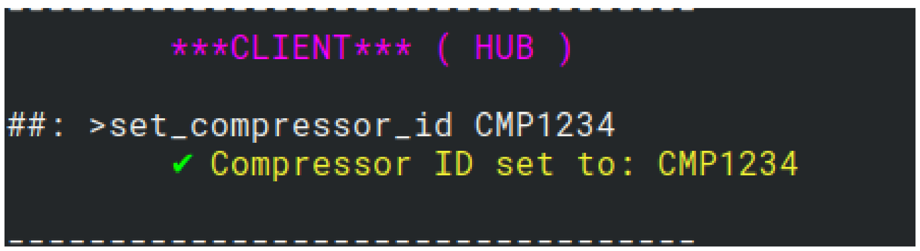
>SELFTEST Run the built-in self test operations and report back results. Not implemented yet.

>SET_TIME [MM-dd-YYYY]_[hh.mm.ss] Set Pi to argument timestamp.

>SEND_LOG upload log file(s) from the last test to the hub. This may span more than one file if the test lasted more than one day. Not implemented yet.

>SEND_ALL_LOGS upload all log files stored on the flash drive to the hub PC. If no flash drive is present, an error message is displayed regarding missing flash drive. Not implemented.

>SET_COMPRESSOR_ID



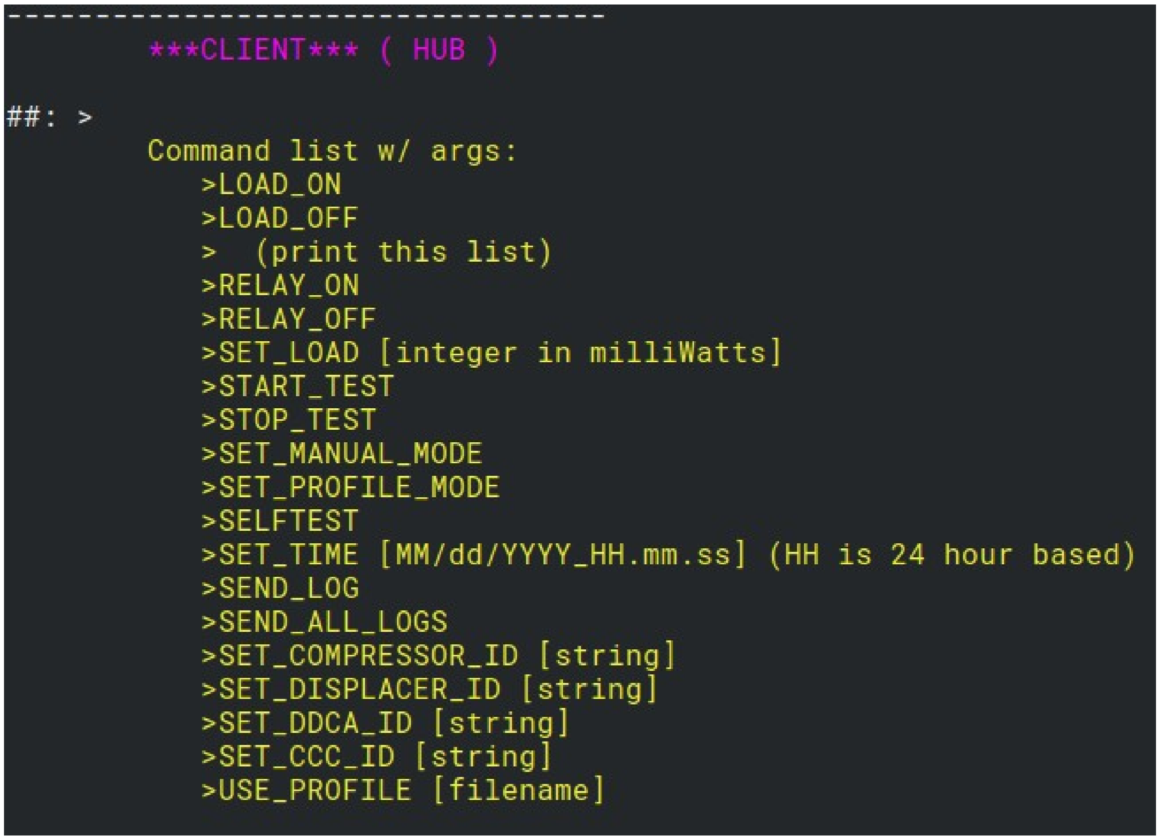
```
----------------------------------
        ***CLIENT*** ( HUB )

##: >set_compressor_id CMP1234
        ✔ Compressor ID set to: CMP1234

----------------------------------
```

>SET_DISPLACER_ID

>SET_DEWAR_ID

>SET_CCC_ID

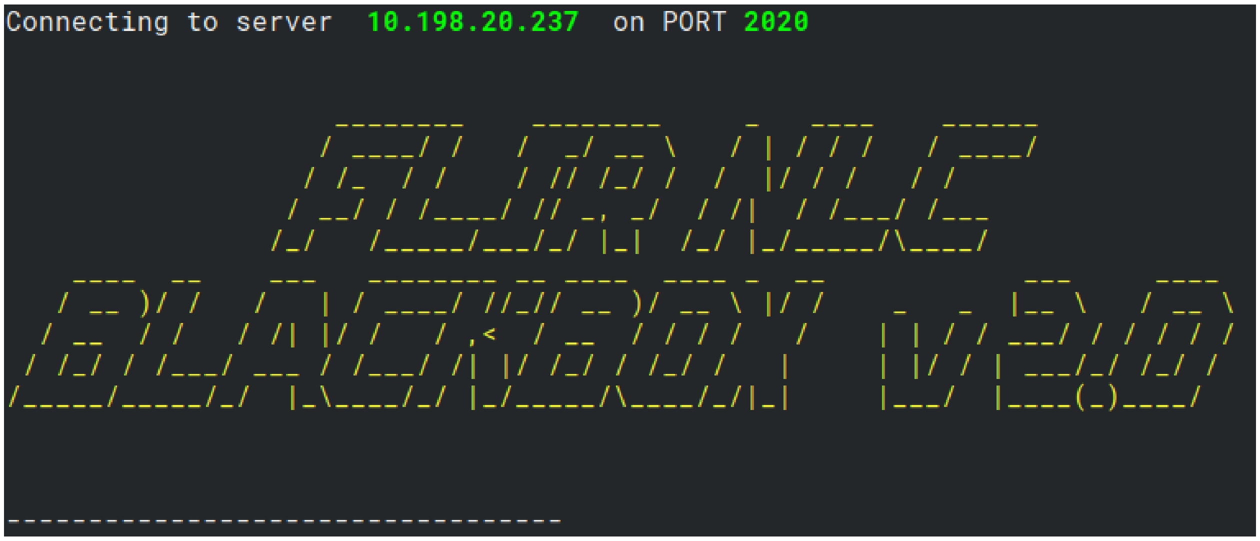
>	If no argument is supplied with the Command instruction, the list of all Commands is printed.



```
----------------------------------
        ***CLIENT*** ( HUB )

##: >
        Command list w/ args:
           >LOAD_ON
           >LOAD_OFF
           >  (print this list)
           >RELAY_ON
           >RELAY_OFF
           >SET_LOAD [integer in milliWatts]
           >START_TEST
           >STOP_TEST
           >SET_MANUAL_MODE
           >SET_PROFILE_MODE
           >SELFTEST
           >SET_TIME [MM/dd/YYYY_HH.mm.ss] (HH is 24 hour based)
           >SEND_LOG
           >SEND_ALL_LOGS
           >SET_COMPRESSOR_ID [string]
           >SET_DISPLACER_ID [string]
           >SET_DDCA_ID [string]
           >SET_CCC_ID [string]
           >USE_PROFILE [filename]
```

3.5 Client Connection

When a client connects, the following welcome message is sent by the pi:



```
Connecting to server  10.198.20.237  on PORT 2020


                     ________    ________     _    ____     ______
                    / ____/ /    / _/ __ \   / | / / /    / ____/
                   / /_  / /     / // /_/ /  /  |/ / /    / /
                  / __/ / /___ _/ // _, _/  / /|  / /___/ /___
                 /_/   /_____/___/_/ |_|  /_/ |_/_____/\____/
    ____  __    ___   ________ __ ____  ____  _  __         ___    ____
   / __ )/ /   /   | / ____/ //_// __ )/ __ \ |/ /    _   _|__ \  / __ \
  / __  / /   / /| |/ /   / ,<  / __  / / / /   /    | | / /__/ / / / / /
 / /_/ / /___/ ___ / /___/ /| |/ /_/ / /_/ /   |     | |/ / __/_/ /_/ /
/_____/_____/_/  |_\____/_/ |_/_____/\____/_/|_|     |___/____(_)____/



----------------------------------
```

4.0 BLACKBOX SOFTWARE & SYSTEMD

To configure the firmware to launch at boot, it was turned into a service launched by the **systemd** kernel subsystem. A file called “Blackberry.service” is located at:

/lib/systemd/system/Blackberry.service

Systemd auto-launches the executable at boot, and if the executable fails or quits for some reason, systemd will continuously attempt to relaunch the executable. This is good and verifies as long as the pi is on, we can connect to it.

One downside of the auto-launch feature is the executable doesn't launch in a terminal instance, so you can’t monitor commands and events as they happen on the pi. If you want to monitor the output on the pi you can stop the service, and launch the standalone executable. Here’s how to do it:

To stop the service:

sudo systemctl stop Blackberry.service

To restart the service:

sudo systemctl start Blackberry.service

To see the status:

sudo systemctl status Blackberry.service

To disable the service from auto-launching at boot:

sudo systemctl disable Blackberry.service

To enable the service from auto-launching at boot:

sudo systemctl enable Blackberry.service

If for some reason the service is not working, you can view the error reporting in the following locations:

dmesg

journalctl

/var/log/boot.log

/var/log/daemon.log

/var/log/messages

/var/log/syslog

/var/log/user.log