Get power limit and mode

A command to get the power limit and the power mode is: ***get_user_power_limit***. A response to this command is JSON with following fields:

- *powerMode* - a case sensitive string which describes the device performance mode. Possible values:
  - Low;
  - Normal;
  - High.
- *powerLimit* - an integer which stands for a user power limit (not the value from overclock tab).

Set power limit and mode

A command to set the power limit and the mode is: ***set_user_power_limit***. This command must include a payload. Example: *{"softRestart":true,"powerMode":1,"powerLimit":1000}*.

Fields are:

- *powerMode* - a case insensitive string or integer which describes a performance mode of the device. Possible values:
  - Low;
  - Normal;
  - High.
- *powerLimit* - an integer which stands for a user power limit (not the value from overclock tab).
- *softRestart* - a boolean which describes whether the device must try to set overclock parameters without stopping a mining[1]. Possible values:
  - true - try to set parameters without stopping a mining.
  - false - do not try to set parameters without stopping a mining.

The response to this command is API command status.

Get fan mode and manual mode speed

A command to get the fan mode and the manual mode speed is: ***get_fan_mode***. A response to this command is JSON with following fields:

- *fan_mode* - a string which describes the device fan mode. Possible values:

[1] The device can fail to do so. In such a case mining will be stopped even if the soft_restart is set to true.

    - *auto* - the device will select fan speed based on its current state.
    - *manual* - the device will always use the selected fan speed.
- *manual_fan_speed_percent* - an integer (percents) which describes the speed of fans. This field is taken into account only if *fan_mode* is *manual*. Minimum value is 10 and maximum is 100.

Set fan mode and manual mode speed

A command to set the fan mode and the manual mode speed is: ***set_fan_mode***. This command must include a payload. Example: *{"fan_mode":"auto","manual_fan_speed_percent":"93"}*.

Fields are:

- *fan_mode* - a case sensitive string which describes the device fan mode. Possible values:
    - *auto* - the device will select fan speed based on its current state.
    - *manual* - the device will always use the selected fan speed.
- *manual_fan_speed_percent* - an integer(percents) which describes speed of fans on startup. This field is taken into account only if *fan_mode* is Manual. Minimum value is 10 and maximum is 100.
- The response to this command is API command status.

Get overclock info

A command to get the overclock info is: ***get_overclock_info***. A response to this command is JSON with all overclock info.

The meaning of the most useful fields can be looked up in the overclock info json section of the appendix.

Set overclock info

A command to set the overclock info is: ***set_overclock_info***. This command must include a payload. Example: *{"board_temp_target":74, "freq_target":430, "power_limit":3850, "voltage_target":1420, "power_max":3950, "voltage_limit":1470, "voltage_min":1300, "soft_restart":true}*.

The meaning of the most useful fields can be looked up in the overclock info json section of the appendix.

The response to this command is API command status.

Delete overclock info

A command to delete the overclock info is: ***delete_overclock_info***.
The response to this command is API command status.

Get board slots state

A command to get the board slots states is: ***get_board_slots_state***. A response to this command is JSON with following field:

- *auto_disable* - a boolean value which describes whether to reboot the device on any board error in an attempt to recover it, or disable the board if recovery limit exceeded and 54x ("Chip id read error") persists.
- *failed_to_power_on_hashboard_reboots* - an integer value which describes how many reboots were performed in an attempt to repair boards.
- *failed_to_power_on_hashboard_max_reboots* - an integer value which describes a limit of the recovery reboots.
- *limit_boards_power* - a boolean value which describes whether to use power from disabled or not working boards. Possible value:
    - *true* - do not use power from disabled or not working boards. The device will limit power of each board to the power it would get if the factory number of boards would be working and each board would get an equal share of the power.
    - *false* - use power from disabled or not working boards. The rest of the boards will get more power.
- *enabled* - an array of booleans. Each value describes the state of the corresponding board slot.
- *auto_disabled* - an array of JSON objects. One object per disabled board. Each object describes the reason and time of disabling a board. Array indexes correspond to board indexes. Fields are:
    - *reason* - an integer field which describes error code which is a reason the board is disabled.
    - *time* - an integer value which describes a time when the board was disabled. The format is Unix timestamp.

Set board slots state

A command to set the board slots states is: ***set_board_slots_state***. This command must include a payload. Example: *{"auto_disable":true, "failed_to_power_on_hashboard_reboots":0, "failed_to_power_on_hashboard_max_reboots":5, "limit_boards_power": false, "enabled":[true, true, true, false]}*.

Field is:

- *auto_disable* - a boolean value which describes whether to reboot the device on any board error in an attempt to recover it, or disable the board if recovery limit exceeded and 54x ("Chip id read error") persists.
- *failed_to_power_on_hashboard_reboots* - an integer value which describes how many reboots were performed in an attempt to repair boards.
- *failed_to_power_on_hashboard_max_reboots* - an integer value which describes a limit of the recovery reboots.
- *limit_boards_power* - a boolean value which describes whether to use power from disabled or not working boards. Possible value:
  - *true* - do not use power from disabled or not working boards. The device will limit power of each board to the power it would get if the factory number of boards would be working and each board would get an equal share of the power.
  - *false* - use power from disabled or not working boards. The rest of the boards will get more power.
- *enabled* - an array of booleans. Each value describes the state of the corresponding board slot. The field is optional. If the array is empty it will enable all boards. Thus *"enabled":[]* is the same as *"enabled":[true, true, true, true]*.

Reset recovery reboot counter

A command to reset the recovery reboot counter is: **reset_failed_to_power_on_hashboard_reboots**.

The response to this command is API command status.

Get firmware version

A command to get the overclock info is: ***get_firmware_version***. A response to this command is JSON with following fields:

- *custom_version* - a string which describes the device custom version.
- *firmware_version* - a string which describes the device firmware version.

Set boards cool fan percent

A command to set pwm% of the fans on cool down (Startup Cooling Fan Speed %) is: ***set_boards_cool_fan_percent***. This command must include a payload. Example: *{"boards_cool_fan_percent":"30"}*.

Field is:

- *boards_cool_fan_percent* - a string representing integer value which describes how much PWM to set on cool down. Minimum value is 10 and maximum is 100.

The response to this command is API command status.

Power status

A command to get whether a device is suspended or deep_suspended is: **power_status**. A response to this command is JSON with following fields:

- *suspend* - a string which describes whether the device is suspended. Possible values:
  - true - string which means that device is suspended.
  - false - string which means that device is not suspended.
- *deep_suspend*[2] - a string which describes whether the device is deep_suspended. Possible values:
  - true - string which means that device is deep_suspended.
  - false - string which means that device is not deep_suspended.

Deep power off

A command to suspend the mining is: **deep_power_off**. An effect of this command will hold after reboot.

The response to this command is API command status.

Deep power on

A command to resume the mining after *deep_power_off* command is: **deep_power_on**.

The response to this command is API command status.

Delete upfreq result

A command to delete autotune (upfreq) results is: **delete_upfreq_results**.

The response to this command is API command status.

[2] Deep suspend is a simple suspend but it holds after reboot.

Get cool temp

A command to get a temperature to which the device will cool down is: **get_cool_temp**. A response to this command is JSON with following fields:

- *type* - a string field which describes what cooling temperature will use the device. Possible values:
  - *default* - use the default cooling temperature of the device.
  - *env_temp* - use an environment temperature.
  - *manual* - use the *manual_temp* field value.
- *manual_temp* - an integer field which describes a temperature to which the device will cool down.

Set cool temp

A command to set a temperature to which the device will cool down is: **set_cool_temp**. This command must include a payload. Example: *{"type":"manual", "manual_temp":35}*.

Fields are:

- *type* - a string field which describes what cooling temperature will use the device. Possible values:
  - *default* - use the default cooling temperature of the device.
  - *env_temp* - use an environment temperature.
  - *manual* - use the *manual_temp* field value.
- *manual_temp* - an integer field which describes a temperature to which the device will cool down. Note that this value must be passed even if the *type* is not *manual*.

The response to this command is API command status.

Get environment temperature limit

A command to get an environment temperature limit is: **get_env_temp_limit**. A response to this command is JSON with following fields:

- *enabled* - a boolean value which describes whether the device must resume or suspend its work when *resume_env_temp* or *suspend_env_temp* are respectively reached.
- *resume_env_temp* - a string which describes an environment temperature when the device will start mining, if it was suspended due to too high environment temperature.
- *suspend_env_temp* - a string which describes an environment temperature when the device will stop mining.

Set environment temperature limit

A command to set environment temperature suspend and resume temperatures is: **set_env_temp_limit**. This command must include a payload. Example: *{"enabled": "true", "resume_env_temp": "50", "suspend_env_temp": "65"}*.

Fields are:

- *enabled* - a boolean value which describes whether the device must resume or suspend its work when *resume_env_temp* or *suspend_env_temp* are respectively reached.
- *resume_env_temp* - a string which describes an environment temperature when the device will start mining, if it was suspended due to too high environment temperature.
- *suspend_env_temp* - a string which describes an environment temperature when the device will stop mining.

Note that *resume_env_temp* must not be greater or equal to *suspend_env_temp.* The response to this command is API command status.

Install AMS

A command to install the AMS is: **ams_install**. This command must include a payload. Example: *{"api_key":"517a56aa-a252-4a7a-9978-a76089aa2a5a", "update_interval": 10}*.

Field is:

- *api_key* - a string which is the AMS API key.
- *update_interval* - an integer value which describes an interval (seconds) between send of the device data to the AMS server. The default value is 5. The value is optional, if no interval specified the 5 seconds will be applied.

The response to this command is API command status.

Uninstall AMS

A command to uninstall the AMS is: **ams_uninstall**.
The response to this command is API command status.

Get AMS API key

A command to get the AMS API key is: **get_ams_install_data**. A response to this command is JSON with following fields:

- *api_key* - string which represents the AMS APIkey if any.
- installed - string which describes whether the AMS is installed. Possible values:
    - true - string which means that AMS is installed. The *api_key* field will return the AMS API key.
    - false - string which means that AMS is not installed. The *api_key* field will be an empty string.

Get upfreq save params

A command to get an upfreq save params is: **get_upfreq_save_params**. To get an explanation of the upfreq restore algorithm read a dedicated appendix 2. A response to this command is JSON with following fields:

- *freq_delay_air* - a string which describes a delay between iterations of changing of freq for an air cooling device. The smaller the delay the faster the tuning.
- *freq_delay_liquid* - a string which describes a delay between iterations of changing of the freq for a liquid cooling device. The smaller the delay the faster the tuning.
- *freq_delay_air_default* - a string which describes the default value of *freq_delay_air*.
- *freq_delay_liquid_default* - a string which describes the default value of *freq_delay_liquid*.
- *voltage_offset_air* - a string which describes a voltage change (mV) for an air cooling device. The value can be negative to down a voltage.
- *voltage_offset_liquid* - a string which describes a voltage change (mV) for a liquid cooling device. The value can be negative to down a voltage.
- *voltage_offset_air_default* - a string which describes the default value of *voltage_offset_air*.
- *voltage_offset_liquid_default* - a string which describes the default value of *voltage_offset_liquid*.
- *decrease_voltage_power_limit_tolerance_percent* - a string which describes a value for which the limit can be exceeded by upfreq restore. When the upfreq restore is starting the first thing to do is to raise the frequency. After this the device will wait for stabilisation of temperature. This limit applies at this moment for voltage and power.
- *decrease_voltage_power_limit_tolerance_percent_default* - a string which describes the default value of *decrease_voltage_power_limit_tolerance_percent*.
- *power_limit_tolerance_percent* - a string which describes a value for which the power limit can be exceeded on upfreq restore.

- *power_limit_tolerance_percent_default* - a string which describes the default value of *power_limit_tolerance_percent*.
- *iin_limit_tolerance_percent* - a string which describes a value for which the inn limit can be exceeded on upfreq restore.
- *iin_limit_tolerance_percent_default* - a string which describes the default value of *iin_limit_tolerance_percent*.
- *iout_limit_tolerance_percent* - a string which describes a value for which the iout limit can be exceeded on upfreq restore.
- *iout_limit_tolerance_percent_default* - a string which describes the default value of *iout_limit_tolerance_percent*.

Set upfreq save params

A command to set an upfreq save params is: **set_upfreq_save_params**. To get an explanation of the upfreq restore algorithm read a dedicated appendix 2. This command must include a payload. Example:
{"freq_delay_air":"2.337","freq_delay_liquid":"2.322","voltage_offset_air":"-22","voltage_offset_liquid":"0","decrease_voltage_power_limit_tolerance_percent":"1.30","power_limit_tolerance_percent": "2.6", "iin_limit_tolerance_percent": "2.4","iout_limit_tolerance_percent": "2.65"}.

Fields are:

- *freq_delay_air* - a string which describes a delay between iterations of changing of freq for an air cooling device. The smaller the delay the faster the tuning.
- *freq_delay_liquid* - a string which describes a delay between iterations of changing of frequency for a liquid cooling device. The smaller the delay the faster the tuning.
- *voltage_offset_air* - a string which describes a voltage change (mV) for an air cooling device. The value can be negative to down a voltage.
- *voltage_offset_liquid* - a string which describes a voltage change (mV) for a liquid cooling device. The value can be negative to down a voltage.
- *decrease_voltage_power_limit_tolerance_percent* - a string which describes a value for which the limit can be exceeded on the decrease of the voltage and power on upfreq restore.
- *power_limit_tolerance_percent* - a string which describes a value for which the power limit can be exceeded on upfreq restore.
- *iin_limit_tolerance_percent* - a string which describes a value for which the inn limit can be exceeded on upfreq restore.

- *iout_limit_tolerance_percent* - a string which describes a value for which the iout limit can be exceeded on upfreq restore.

The response to this command is API command status.

Start profiles generation

A command to start profiles generation is: **generate_profiles**.
The response to this command is API command status.

Stop profiles generation

A command to stop profiles generation is: **stop_profiles_generation**.
The response to this command is API command status.

Get profiles generation status

A command to get the profiles generation status is: **get_profiles_generation_status**. A response to this command is JSON with following fields:

- *generating_profiles* - a string which describes whether profile generation is currently underway. Possible values:
  - *"true"* - generation is underway.
  - *"false"* - generation is not underway.
- *has_generated_profiles* - a string which describes whether the device has generated profiles. Possible values:
  - *"true"* - the device has generated profiles.
  - *"false"* - the device has not generated profiles.

Delete generated profiles

A command to stop profiles generation is: **delete_generated_profiles**.
The response to this command is API command status.

Get Profile Switcher

A command to get the profiles switcher settings is: **get_profile_switcher**. A response to this command is JSON with following fields:

- *enabled* - a string which describes whether profile generation is currently underway. Possible values:
  - *"true"* - generation is underway.
  - *"false"* - generation is not underway.
- *lower_temp* - a string which describes the temperature celsius upon reaching which the device will select a lower profile.
- *raise_temp* - a string which describes the temperature celsius upon reaching which the device will select a higher profile.
- *max_profile_id* - a string which describes an ID of profile which is the highest profile that could be set by the switcher system.
- *ignore_pwm* - an integer field which describes whether the switcher will ignore current PWM when increasing profile. Possible values:
  - *true* - ignore PWM when increasing profile.
  - *false* - do not increase profile if PWM is >= 90%.
- *profiles* - a JSON array which contains JSON objects. Each object contains data about profiles. It has following fields:
  - *id* - an integer field which contains profile id. The one which could be set to *max_profile_id*.
  - *name* - a string field which contains the profile name.

Set Profile Switcher

A command to set the profiles switcher settings is: **set_profile_switcher**. This command must include a payload. Example: {"enabled":"true","lower_temp":"100","raise_temp":"90","ignore_pwm":"false","max_profile_id":"1000000"}.

Fields are:

- *enabled* - a string which describes whether profile generation is currently underway. Possible values:
  - *"true"* - generation is underway.
  - *"false"* - generation is not underway.
- *lower_temp* - a string which describes the temperature celsius upon reaching which the device will select a lower profile.
- *raise_temp* - a string which describes the temperature celsius upon reaching which the device will select a higher profile.
- *max_profile_id* - a string which describes an ID of profile which is the highest profile that could be set by the switcher system.

- *ignore_pwm* - an integer field which describes whether the switcher will ignore current PWM when increasing profile. Possible values:
  - *true* - ignore PWM when increasing profile.
  - *false* - do not increase profile if PWM is >= 90%.

Note that *lower_temp* must be greater than *raise_temp.* The profile switcher can work only with generated profiles. The response to this command is API command status.

Get stats

A command to get an api stats info is: **stats**. A response to this command is a JSON array which contains JSON objects. Each object contains data on the corresponding chain.

Check upfreq results

A command to see whether upfreq results exist is: **has_upfreq_results**. A response to this command is JSON with following fields:

- *has_upfreq_results* - a string which describes whether upfreq results exist.

Enable/disable power fan

A command to enable or disable the power fan is: **set_psu_fan**. This command must include a payload. Example: *set_psu_fan {"enabled": "true"}*.

Fields are:

- *enabled* - string which describes whether the psu fan must be enabled or disabled. Possible values:
  - *"true"* - enable psu fan.
  - *"false"* - disable psu fan.

The response to this command is API command status.

Get liquid cooling

A command to get whether the device is in the liquid cooling mode is: **get_liquid_cooling**. A response to this command is JSON with following fields:

- *liquid_cooling* - string which describes whether the device is in liquid cooling mode or not. Possible values:
  - *"true"* - the device is in liquid cooling mode.
  - *"false"* - the device is not in liquid cooling mode.
- *is_fan_machine* - string which describes whether the device is a fan cooling device. Possible values:
  - *"true"* - the device is a fan cooling device.
  - *"false"* - the device is not a fan cooling device.
- *cool_mode* - string which describes the cooling mode of the device. Possible values:
  - *"air"* - the mode for devices which are cooled by fans.
  - *"liquid"* - the mode for devices which are cooled by immersion, but is not factory made for this.
  - *"hydro"* - the mode for devices which are cooled by a liquid cooling system.
  - *"immersion"* - the mode for devices which are factory made to be cooled by immersion.

Set liquid cooling

A command to set the cooling mode is: **set_liquid_cooling**. This command must include a payload. Example: *set_liquid_cooling {"liquid_cooling": "true"}*.

Fields are:

- *liquid_cooling* - string which describes whether the device is in liquid cooling mode or not. Possible values:
  - *"true"* - the device is in liquid cooling mode.
  - *"false"* - the device is not in liquid cooling mode.

The response to this command is API command status.

Get lower profile if autotune failed policy

A command to see whether the device will lower profile if autotune fails is: **get_lower_profile_if_autotune_failed**. A response to this command is JSON with following fields:

- *enabled* - string which describes whether the device will lower profile if autotune fails. Possible values:
  - *"true"* - the device will lower profile if autotune fails.
  - *"false"* - the device will not lower profile if autotune fails.

Set lower profile if autotune failed policy

A command to set whether the device will lower profile if autotune fails is: **set_lower_profile_if_autotune_failed**. This command must include a payload. Example: *set_lower_profile_if_autotune_failed {"enabled": "true"}*.

Fields are:

- *enabled* - string which describes whether the device will lower profile if autotune fails. Possible values:
  - *"true"* - the device will lower profile if autotune fails.
  - *"false"* - the device will not lower profile if autotune fails.

The response to this command is API command status.

Get additional PSU status

A command to see whether the device has an additional PSU enabled is: **get_additional_psu**. A response to this command is JSON with following fields:

- *enabled* - string which describes whether the device has an additional PSU enabled. Possible values:
  - *"true"* - the device has an additional PSU enabled.
  - *"false"* - the device does not have an additional PSU enabled.

Set additional PSU status

A command to set whether the device has an additional PSU enabled is: **set_additional_psu**. This command must include a payload. Example: *set_additional_psu {"enabled": "true"}*.

Fields are:

- *enabled* - string which describes whether the device has an additional PSU enabled. Possible values:
  - *"true"* - the device has an additional PSU enabled.
  - *"false"* - the device does not have an additional PSU enabled.

The response to this command is API command status.

Upgrade PSU firmware

A command to upgrade PSU firmware is: **upgrade_psu_firmware**. Example: *upgrade_psu_firmware*.

The response to this command is API command status.

Get list of allowed pools

A command to get a list of pools which are allowed to work with is: **get_allowed_pools**. A response to this command is JSON with following fields:

- *pools* - a JSON array which contains up to 10 strings. Each string contains a URL of the pool which is allowed to work with or empty.

Set list of allowed pools

A command to set a list of pools which are allowed to work with is: **set_allowed_pools**. This command must include a payload. Example: *set_allowed_pools {"pools":["stratum+tcp://example.com:3333", "stratum+tcp://example.com:443"]}*.

Fields are:

- *pools* - a JSON array which can contain up to 10 strings. Each string must be a single url of the pool which is allowed to work with. Leave empty to remove a pool at the index.

The response to this command is API command status.

Get stratum off

A command to get whether the stratum off is enabled is: **get_stratum_off**. A response to this command is JSON with following fields:

- *enabled* - string which describes whether the stratum off is enabled. Possible values:
  - *"true"* - the stratum off is enabled.
  - *"false"* - the stratum off is disabled.
- *stratum_off_server* - string which describes the stratum off server. That is IP plus port. Example: 192.168.1.1:9999.
- *stratum_off_user* - a string which describes the stratum off user name.
- *stratum_off_pass* - a string which describes the stratum off password.

Set stratum off

A command to set the stratum off is: **set_stratum_off**. This command must include a payload. Example: *set_stratum_off {"enabled":true,"stratum_off_server":"192.168.1.1:9999","stratum_off_user":"stratum off.user","stratum_off_pass":"123"}*.

Fields are:

- *enabled* - string which describes whether the stratum off is enabled. Possible values:
  - *"true"* - the stratum off is enabled.
  - *"false"* - the stratum off is disabled.
- *stratum_off_server* - string which describes the stratum off server. That is IP plus port. Example: 192.168.1.1:9999.
- *stratum_off_user* - a string which describes the stratum off user name.
- *stratum_off_pass* - a string which describes the stratum off password.

The response to this command is API command status.

Get proxy info

A command to get the proxy info is: **get_proxy_info**. A response to this command is JSON with following fields:

- *enabled* - string which describes whether the traffic is routed through proxy. Possible values:
  - *"true"* - the traffic is routed through proxy.
  - *"false"* - the traffic is not routed through proxy.
- *host* - string which describes a proxy address.
- *user* - string which describes a proxy username.
- *password* - string which describes a proxy password.

Set proxy info

A command to set the proxy info is: **set_proxy_info**. This command must include a payload. Example: *set_proxy_info {"enabled":"true","host":"192.168.1.1:9999", "user": "proxy_user", "password": "proxy_password"}*.

Fields are:

- *enabled* - string which describes whether the traffic is routed through proxy. Possible values:
  - *"true"* - the traffic is routed through proxy.
  - *"false"* - the traffic is not routed through proxy.
- *host* - string which describes a proxy address. It must be in format ip:host. For example: 192.168.1.1:9999.

- *user* - string which describes a proxy username.
- *password* - string which describes a proxy password.

The response to this command is API command status.

Get compute info

A command to get the compute info is: **get_compute_info**. A response to this command is JSON with following fields:

- *wmt_port* - string which describes a port for Whatsminer Tool connection. Default value: 8889.

Set compute info

A command to set the compute info is: **set_compute_info**. This command must include a payload. Example: *set_compute_info {"wmt_port":"8888"}*.

Fields are:

- *wmt_port* - string which describes a port for Whatsminer Tool connection. Default value: 8889.

This command will take effect only after reboot. However, the *wmt_port* value will change immediately. The response to this command is API command status.

Get generate profiles parameters (get_generate_profiles_params)

A command to get the parameters of the profiles generation is: **get_generate_profiles_params**. A response to this command is JSON with following fields:

- *power_limit_air* - string which describes a limit for power in air cooling mode.
- *power_limit_liquid* - string which describes a limit for power in liquid cooling mode.
- *power_limit_hydro* - string which describes a limit for power for hydro devices.
- *power_limit_immersion* - string which describes a limit for power for immersion devices.
- *freq_step* - string which describes an amount for which a frequency is adjusted per step.

- *power_limit_air_default* - string which describes a default limit for power in air cooling mode.
- *power_limit_liquid_default* - string which describes a default limit for power in liquid cooling mode.
- *power_limit_hydro_default* - string which describes a default limit for power for hydro devices.
- *power_limit_immersion_default* - string which describes a default limit for power for immersion devices.
- *freq_step_default* - string which describes a default amount for which a frequency is adjusted per step.

Set generate profiles parameters (set_generate_profiles_params)

A command to set the parameters of the profiles generation is: **set_generate_profiles_params**. This command must include a payload. Example: *set_generate_profiles_params {"power_limit_air":"3950","power_limit_liquid":"3950","power_limit_hydro":"10000","power_limit_immersion":"9000","freq_step":"2.500000"}*.

Fields are:

- *power_limit_air* - string which describes a limit for power in air cooling mode.
- *power_limit_liquid* - string which describes a limit for power in liquid cooling mode.
- *power_limit_hydro* - string which describes a limit for power for hydro devices.
- *power_limit_immersion* - string which describes a limit for power for immersion devices.
- *freq_step* - string which describes an amount for which a frequency is adjusted per step.

The response to this command is API command status.

Get advanced fan mode (get_advanced_fan_mode)

A command to get the advanced settings of the fans is: **get_advanced_fan_mode**. A response to this command is JSON with following fields:

- *use_chip_temp* - a string which describes which temp the cooling system will use for its work. Possible values:
    - “true” - use chip temp when adjusting fans speed.
    - “false” - use board temp when adjusting fans speed.
- *use_custom_fan_algo* - a string which describes whether to use the custom fan algorithms. That is an algorithm which utilises a PID controller mechanism. Possible values:
    - “*true*” - use a custom fan algorithm.
    - “*false*” - use standard fan algorithm.
- *use_custom_fan_algo_suspend* - a string which describes whether to use the custom fan algorithms while suspend mode. That is an algorithm which utilises a PID controller mechanism. Possible values:
    - “*true*” - use a custom fan algorithm.
    - “*false*” - use standard fan algorithm.
- *chip_temp_protect_default* - a string which describes a maximum chip temperature. Usually the default value is 120°C.
- *chip_temp_target_default* - a string which describes a target chip temp. By default this value is 5°C less than *chip_temp_protect_default*.
- *adjust_chip_temp* - a string which describes an offset value which is added to the max chips temp. The device will try to keep max chip temp close to the *chip_temp_target_default* + *adjust_chip_temp*. This value is respected only if *use_chip_temp* is *true*. This offset must be less or equal to 0.
- *adjust_board_temp* - a string which describes an offset value which is added to the max board temp.The device will try to keep max board temp close to the Board Target Temp + *adjust_board_temp*. This value is respected only if *use_chip_temp* is *false*. This offset must be less or equal to 0.
- *use_target_temp_ramp* - a string which describes if use temp ramp on exit from suspend mode. Possible values:
    - “true” - use.
    - “false” - don’t use.
- *ramp_adjust_target_temp* - a string which describes an offset value which is added to the current target temp on exit from suspend mode, working only if *use_target_temp_ramp* is “true”. This value must be in the range from -30.0 to 30.0.
- *ramp_target_temp_speed* - a string which describes how fast the target temp rises after suspend mode, working only if *use_target_temp_ramp* is “true”. This value is measured in degrees per second and must be in the range from 0.01 to 100.0.

Those settings do not survive reboot.

Set advanced fan mode (set_advanced_fan_mode)

A command to set the advanced settings of the fans is: **set_advanced_fan_mode**. This command must include a payload. Example: *{"use_chip_temp":"false","use_custom_fan_algo_suspend":"false","use_custom_fan_algo":"false","adjust_chip_temp":"0.000000","adjust_board_temp":"-0.000000","use_target_temp_ramp":"false","ramp_adjust_target_temp":"2","ramp_target_temp_speed":"0.15"}*.

Fields are:

- *use_chip_temp* - a string which describes which temp the cooling system will use for its work. Possible values:
  - “*true*” - use chip temp when adjusting fans speed.
  - “*false*” - use board temp when adjusting fans speed.
- *use_custom_fan_algo_suspend* - a string which describes whether to use the custom fan algorithms while suspend mode. That is an algorithm which utilises a PID controller mechanism. Possible values:
  - “*true*” - use a custom fan algorithm.
  - “*false*” - use standard fan algorithm.
- *use_custom_fan_algo* - a string which describes whether to use the custom fan algorithms. That is an algorithm which utilises a PID controller mechanism. Possible values:
  - “*true*” - use a custom fan algorithm.
  - “*false*” - use standard fan algorithm.
- *adjust_chip_temp* - a string which describes an offset value which is added to the max chips temp. The device will try to keep max chip temp close to the *chip_temp_target_default* + *adjust_chip_temp*. This value is respected only if *use_chip_temp* is *true*. This offset must be less or equal to 0.
- *adjust_board_temp* - a string which describes an offset value which is added to the max board temp.The device will try to keep max board temp close to the Board Target Temp + *adjust_board_temp*. This value is respected only if *use_chip_temp* is *false*. This offset must be less or equal to 0.
- *use_target_temp_ramp* - a string which describes if use temp ramp on exit from suspend mode. Possible values:
  - “true” - use.
  - “false” - don’t use.
- *ramp_adjust_target_temp* - a string which describes an offset value which is added to the current target temp on exit from suspend mode, working only if *use_target_temp_ramp* is “true”. This value must be in the range from -30.0 to 30.0.

- *ramp_target_temp_speed* - a string which describes how fast the target temp rises after suspend mode, working only if *use_target_temp_ramp* is “true”. This value is measured in degrees per second and must be in the range from 0.01 to 100.0.

The response to this command is API command status. Those settings do not survive reboot.

Summary (summary)

A command to get the summary is **summary**. That is the factory command. However it has additional fields. Those fields are:

- *Power Realtime* - an integer field which describes a current power.
- *Miner Memory Usage* - an integer which describes an amount of memory used by the firmware. Unit: MB.
- *Miner PID* - an integer which describes the current PID of the firmware.
- *PSU Serial No* - a string field which describes a PSU serial number.
- *PSU Name* - a string field which describes a PSU model.
- *PSU Vin0* - a floating point field which describes an input voltage for 1st phase.
- *PSU Vin1* - a floating point field which describes an input voltage for 2nd phase. It is zero when supply is one-phase.
- *PSU Vin2* - a floating point field which describes an input voltage for 3rd phase. It is zero when supply is one-phase.
- *PSU Vout* - a floating point field which describes an output voltage.
- *PSU Iout* - a floating point field which describes an output current.
- *PSU Iin0* - a floating point field which describes an input current for the 1st phase.
- *PSU Iin1* - a floating point field which describes an input current for the 2nd phase. It is zero when supply is one-phase.
- *PSU Iin2* - a floating point field which describes an input current for the 3rd phase. It is zero when supply is one-phase.
- *Chip Temp Protect* - an integer value which describes a limit temperature for chips.
- *Chip Temp Target* - an integer value which describes a target temperature for chips.
- *PSU Temp0* - a floating point field which describes a PSU temperature from the 1st sensor.
- *PSU Temp1* - a floating point field which describes a PSU temperature from the 2nd sensor.
- *PSU Temp2* - a floating point field which describes a PSU temperature from the 3rd sensor.

- *PSU Fan Speed* - an integer field which describes a speed of the PSU fan.
- *Status* - a string field which describes the current status of the device. Possible states:
    - *Suspended: High Env. Temp* - the device is suspended due to too high environment temperature.
    - *Suspended* - the device is suspended.
    - *Generating Profiles* - the device is generating profiles.
    - *Restoring* - the device sets a profile from saved results.
    - *Tuning* - the device is tuning a profile.
      Miner Type - a string field which describes a model of the device.
- *Factory GHS* - an integer field which describes a hashrate of the device with the factory settings. Unit: GH/s.

Reset MAC address (reset_mac)

A command to set the MAC address to a new value is: **reset_mac**. This command does not require a payload. In such a case the MAC address will be generated automatically. However, an optional argument *mac* is accepted. Example: *{"mac":"C8:11:05:00:53:3A"}*.

Fields are:

- *mac* - a string which describes a MAC address to set.

The response to this command is API command status.

Changelog

2.0.2

Add commands *get_debug_options* and *set_debug_options*.

2.0.3

Fix a bug with a cutted down response of devs command on devices with a huge number of chips.

Appendix 1

Overclock info JSON

This JSON contains all overclock info. Among others, this JSON will include following fields:

1. *board_temp_target* - an integer which describes boards target temperature of the device.
2. *freq_target* - an integer which describes a target frequency of the device.
3. *power_limit* - an integer which describes a power limit of the device. This value must be less than powerMax.
4. *voltage_target* - an integer which describes a target voltage of the device. This value must be less than voltageLimit and greater than voltageMin.
5. *power_max* - an integer which describes a max power of the device. This value must be greater than powerLimit.
6. *voltage_limit* - an integer which describes a voltage limit (maximum voltage) of the device. This value must be greater than VoltageMin and voltageTarget.
7. *voltage_min* - an integer which describes a minimum voltage of the device. This value must be less than voltageLimit and voltageTarget
8. *soft_restart* - a boolean which describes whether the device must try to set overclock parameters without stopping a mining[3]. This field can be passed to set commands, but it is never returned by get commands. Possible values:
    a. true - try to set parameters without stopping a mining.
    b. false - do not try to set parameters without stopping a mining.

    The preconditions for the *sorf_restart* to work is:
    1. The current status of a device must be Mining.
    2. There must be an upfreq result for the profile which is set by the command.
    3. The feature works only for the enterprise version of the firmware.

Note that when this data is returned by the get_overclock_info command not all of those fields are on the same level. Some fields would be available under the *fields* field. Those fields are: *board_temp_target*, *freq_target*, *power_limit*, *voltage_target*, *power_max*, *voltage_limit*, *voltage_min*, *soft_restart*. Also, those fields will be JSON objects. Each object will contain following fields:

min - a minimum value for the field.
current - a current value of the field.
default - a default value for the field.
max - a maximum value for the field.

[3] The device can fail to do so. In such a case mining will be stopped even if the soft restart is set to true.

API command status

That are responses to commands which by their nature do not explicitly require any response. The response simply shows whether a particular command succeeded or not. Not exhaustive list of examples:

- Successful completion of a command:
  - *{"STATUS":"S","When":1723807869,"Code":131,"Msg"[4]:"API command OK","Description":""}*
- Unsuccessful completion of command:
  - *{"STATUS":"E","When":1723808069,"Code":132,"Msg":"API command ERROR","Description":""}*

Note that both above mentioned responses mean that the specified command does exist. When a command is unknown to the device it responds with the following message:

- *{"STATUS":"E","When":1723810620,"Code":14,"Msg":"invalid cmd","Description":""}*

[4] The "Msg" field can contain data which is a response of the command.

Appendix 2

Upfreq restore algorithm

Upfreq restore procedure consists of 4 steps.

On the first step the device sets a voltage. The voltage to be set is selected by the device based on current environment temperature. Then the *voltage_offset_air* or *voltage_offset_liquid* is added to the voltage value. That is the final value which will be set.

On the second step the device raises frequency. It is done by a number of small frequency changes to make the process more safe. The delay between each step could be set via *freq_delay_air* and *freq_delay_liquid* fields of the **set_upfreq_save_params** command.

On the third step the device waits for a temperature to become stable.

On the fourth step the device starts a fine tuning of the voltage. It selects a voltage which provides the closest hashrate to the specified.