JSON over HTTP (POST)

NETIO M2M API protocols docs

Protocol version: **JSON Version 2.4**

Document published: **20.07.2021**

Short summary

JSON over HTTP(s) protocol is a file-based M2M API protocol, where the NETIO device is a HTTP(s) server and the client downloads or uploads one text file document in the json format to control the NETIO power outputs (230V power sockets or IEC-320 power outlets 110/230V).

- In default there is not HTTPs enabled. Only HTTP.
  Just some NETIO devices (PowerPDU 4C) supports HTTPs secured version.
- For some NETIO devices the protocol also includes energy metering values.
- For NETIO PowerDIN 4PZ the protocol also support DI inputs states and counters.
- The JSON protocol must be enabled first in the WEB configuration of the respective device.
  For details, see the "NETIO WEB configuration" chapter.
- This protocol is HTTP(s) based. There can be used different port than 80 for device web configuration. Check the product manual
- Username and password to access the file is hidden in the HTML header.
  There can be different username & password for the read and write access.
  Default configuration is read/write for the user=netio password=netio.
- With write (**netio.json** file upload by http post) the device send you back the current (updated) json answer content in the same structure as the netio.json file.

Supported devices

NETIO 4x Linux based devices: networked power sockets with LAN / WiFi connectivity.

- NETIO PowerPDU 4C
- NETIO 4, 4All (obsolete products)

NETIO 4x firmware – 3.0.1 and later

Standard NETIO devices: networked power sockets with LAN / WiFi connectivity.

- NETIO **PowerCable REST 101x** *(Energy metering)*
- NETIO **PowerBOX 3Px**
- NETIO **PowerBOX 4Kx** *(Energy metering)*
- NETIO **PowerDIN 4KZ** *(Energy metering)*
- NETIO **PowerPDU 4PS**
- NETIO **PowerPDU 8QS** *(Energy metering)*

Note: Firmware – 3.1.3 and later,

Note: This document provides basic info about the M2M API protocol.
Other device functions are described in the product manual.

Quick start with json & NETIO

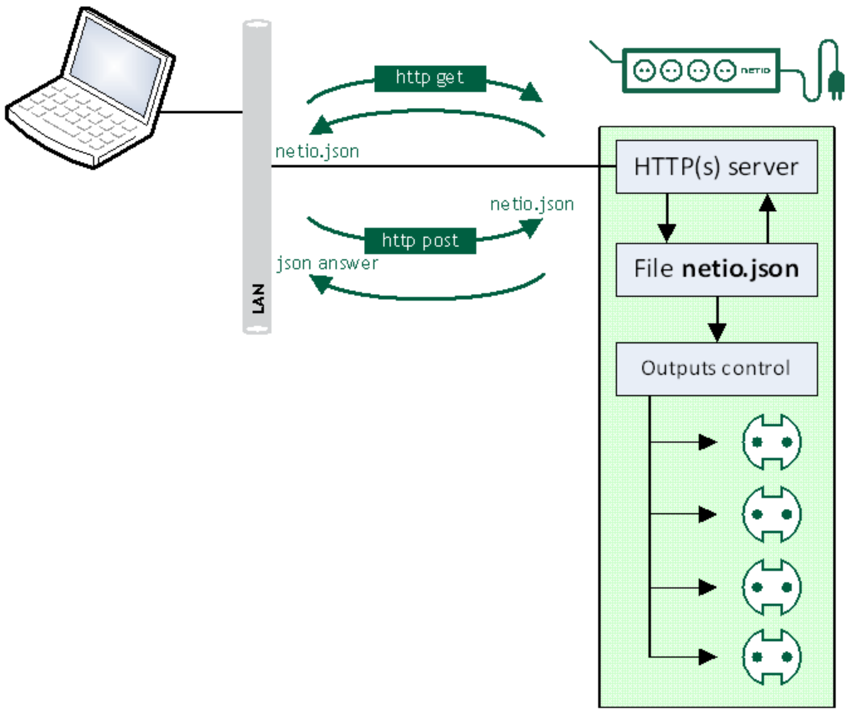
- **READ function - status**
  Read a **netio.json** file from your NETIO by HTTP(s) **GET**: http(s)://<netioIP>/netio.json
  Example: http://192.168.1.1/netio.json

- **WRITE function - control**
  Upload the following json file by HTTP(s) **POST** to: http(s)://<netioIP>/netio.json
  Example: http://192.168.1.1/netio.json

netio.json file (command to switch Power output 1 to ON):

```
{
   "Outputs":[
      {
         "ID":1,
         "Action":1
      }
   ]
}
```

If the netio.json file & command is accepted, then NETIO device returns Status Code "200 OK" and status json file.



Security issues
Do not use default usernames and passwords! Keep your Ethernet and WiFi networks secured.

HTTP(s) options

Most of NETIO devices **do not support** HTTPS protocol

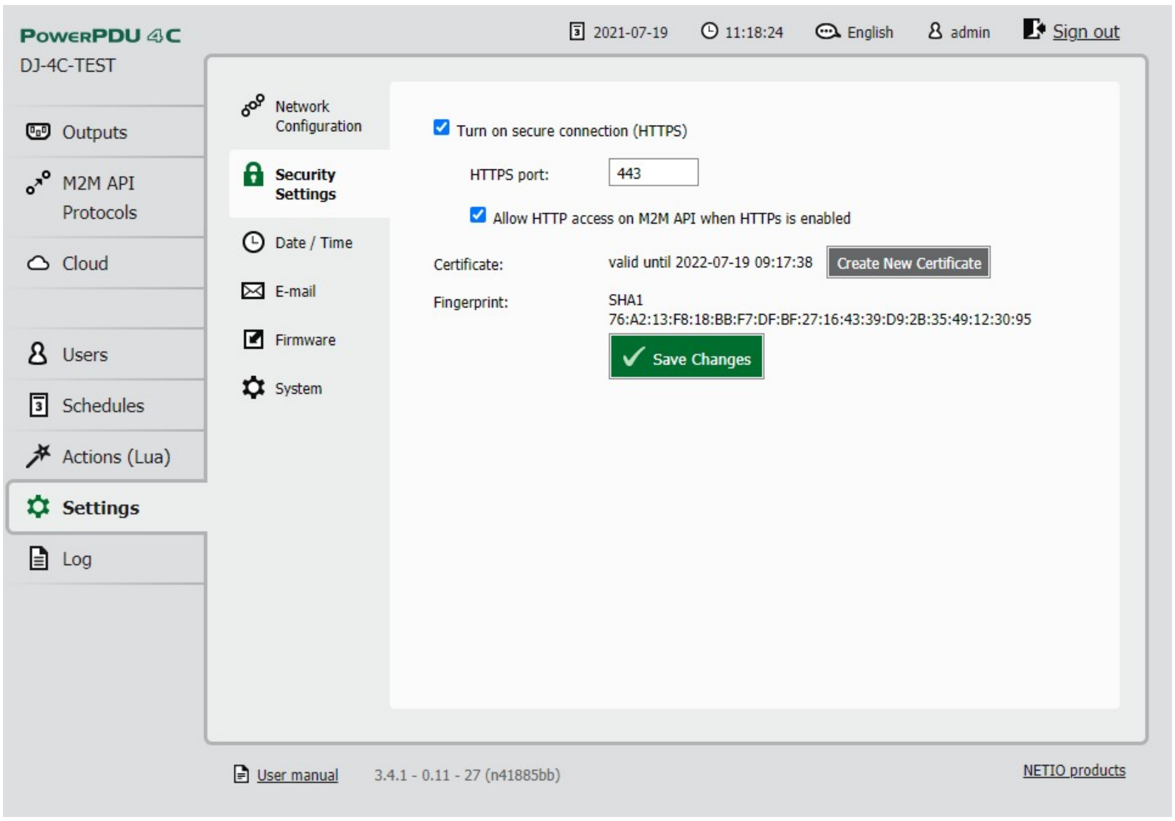
- NETIO **PowerCable REST 101x**
- NETIO **PowerBOX 3Px**
- NETIO **PowerBOX 4Kx**
- NETIO **PowerDIN 4KZ**
- NETIO **PowerPDU 4PS**
- NETIO **PowerPDU 8QS**

NETIO PowerPDU 4C

There are 2 different HTTP(s) ports:

1) The web administration of the device - HTTP(s).
   Web administration is in the Settings/System (HTTP) or System/Security Settings (HTTPs).

2) Separated HTTP(s) port for the M2M API protocols (XML, JSON, URL API).
   Web administration is in each M2M API protocol settings.



☑ Turn on secure connection (HTTPS)

HTTPS port: 443

☑ Allow HTTP access on M2M API when HTTPs is enabled

Certificate: valid until 2022-07-19 09:17:38

Fingerprint: SHA1 76:A2:13:F8:18:BB:F7:DF:BF:27:16:43:39:D9:2B:35:49:12:30:95

3.4.1 - 0.11 - 27 (n41885bb)

- All HTTP(s) protocols (XML / JSON / URL-API) share one HTTP(s) port.
- For PowerPDU 4C have to be both ports (for web + URL API) **http** or **https**.
- Custom HTTPS certificate is not supported.

General NETIO output functions

Output status – “read” function

- **0** – Power **OFF**
- **1** – Power **ON**

Output actions – “write” function

- **0 –** Turn **OFF**
- **1 –** Turn **ON**
- **2 –** Short OFF delay (restart)
- **3 –** Short ON delay
- **4 –** Toggle (invert the state)
- **5 –** No change
- **6 –** Ignored (return value from reading the tag)
  Actual output value is in ”State” tag (0 / 1).

Short ON / OFF delay

This command switches a power output On / Off for a defined time. It is useful for example to power-cycle a server with a defined switch-off time, or to switch on a pump for a defined time.

The short ON / OFF delay interval can be defined in the device web administration. It is specified in ms (milliseconds) and rounded up to hundreds of miliseconds (0,1s).
This interval can be also defined using some M2M API protocol commands. In that case, it is valid only for a single protocol session (the following short ON / Short OFF command). When the connection is closed or restarted, the interval is reset to the device default value (defined in the web administration for each output).

NETIO PowerPDU 4C and NETIO 4/4All: The “short” delay is protected - the power output will remain in the defined state regardless of any other M2M requests received. During this time, the output state can only be changed by pressing the button on the NETIO device and this action cancel M2M short ON/OFF command for the particular output. Other requests to control the particular output are simply ignored and an ERROR logged with reason rejected in a device Log.

Power-Up outputs state

All outputs are OFF during the first 5 to 30 seconds after power-up (depending on device model). After this time, all outputs are set to the selected state based on its individual settings:

- **LAST state**
  After a power outage, the NETIO device sets each power output to the last stored state of this one output.
- **ON**
  The output is turned ON.
- **OFF**
  The output stays OFF.

Note: The function **Scheduler** is checked during Power-Up initialization. When enabled, it can affect one or more power output stated based on current time and date.

NETIO PowerPDU 4C and NETIO 4/4All - Custom based **Lua scripts** can affect output stated too.

General NETIO input read features

Input status

- **0** – "open" / **ON**
- **1** – "closed" / **OFF**

S0 counters

- Number of S0 impulses / "ON" pulses

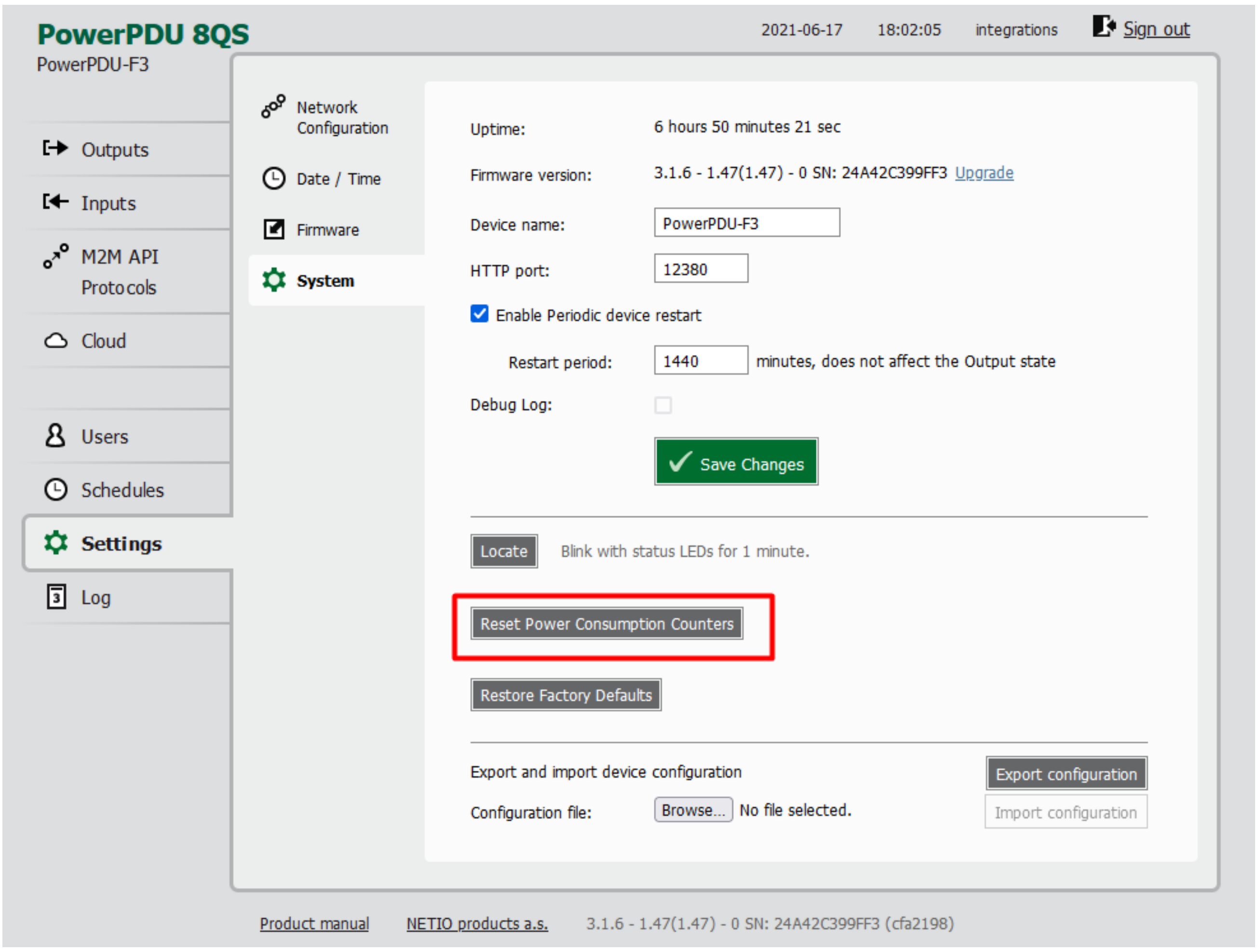
General NETIO Counters

- Power consumption counters can be reset to 0 manually
- Power consumption NR counters are Not Resettable
- S0 pulse counters on DI (Digital Inputs) can be reset to 0 manually

- All counters are not affected by power outage.

How to reset counters to 0

Click to button in the device settings. It will reset the counters.



PowerPDU 8QS
PowerPDU-F3

2021-06-17 18:02:05 integrations Sign out

Uptime: 6 hours 50 minutes 21 sec

Firmware version: 3.1.6 - 1.47(1.47) - 0 SN: 24A42C399FF3 Upgrade

Device name: PowerPDU-F3

HTTP port: 12380

☑ Enable Periodic device restart

Restart period: 1440 minutes, does not affect the Output state

Debug Log: ☐

Blink with status LEDs for 1 minute.

Export and import device configuration

Configuration file: No file selected.

Product manual NETIO products a.s. 3.1.6 - 1.47(1.47) - 0 SN: 24A42C399FF3 (cfa2198)

Energy metering variables

Energy metering is available for:

- NETIO PowerPDU 4C
- PowerCable REST
- PowerBox 4Kx
- PowerDIN 4PZ
- PowerPDU 8QS

Parameters for the **whole NETIO device**:

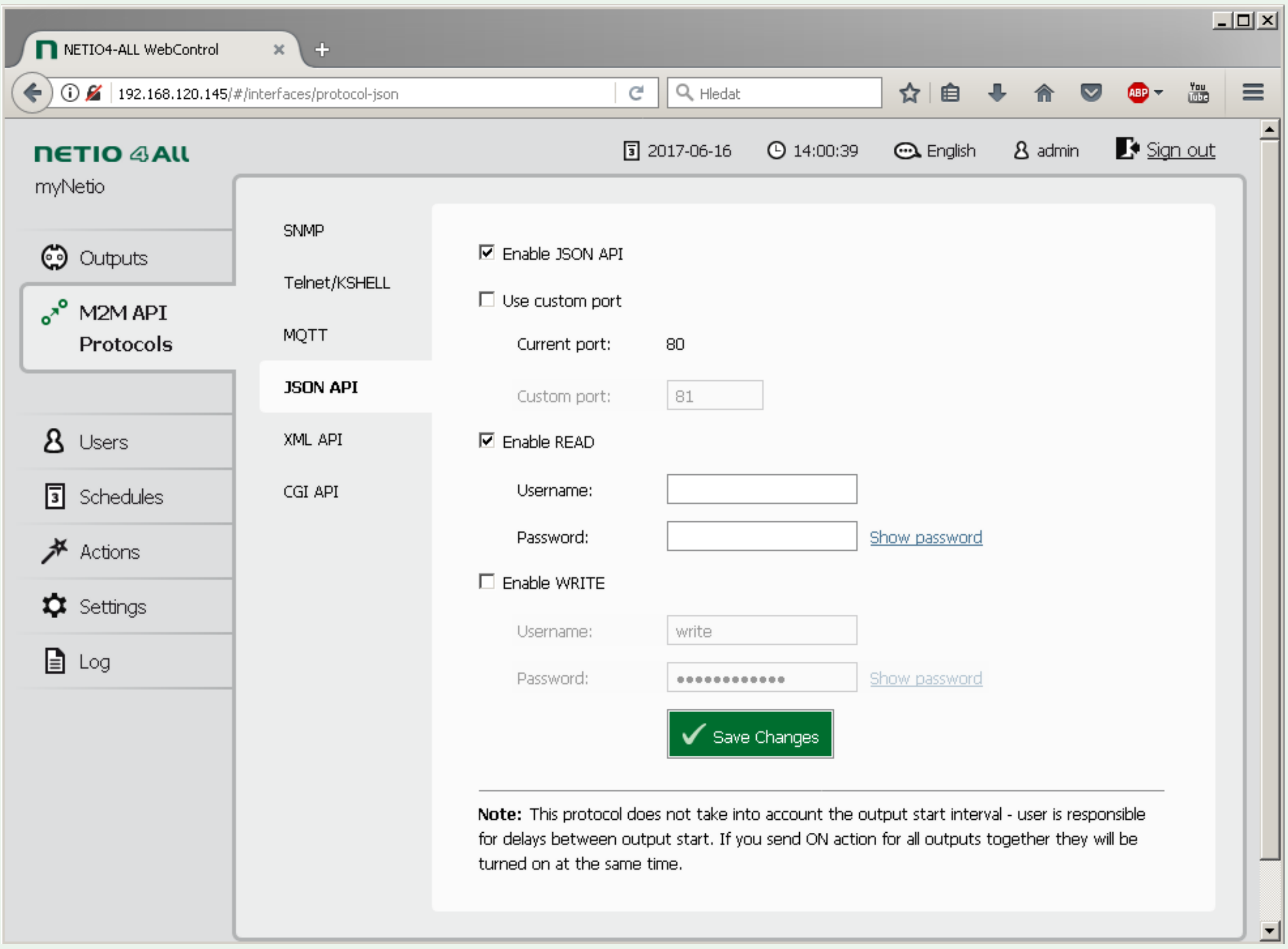
Variable	Unit	Description
Voltage	V	Instantaneous voltage
Frequency	Hz	Instantaneous frequency
Total Current	mA	Instantaneous total current through all power outputs
Total Power Factor	-	Instantaneous True Power Factor – weighted average from all meters
Total Phase	°	Instantaneous True Phase – weighted average from all meters
Total Load	W	Total load (power) of all power outputs (device’s own internal consumption is not included)
Total Energy	Wh	Total Counter of consumed Energy (resettable)
Total Energy NR	Wh	Not Resettable counter of total consumed Energy
Total Reverse Energy	Wh	Counter of Reversed (produced) Energy (resettable)
Total Reverse Energy NR	Wh	Not Resettable counter of total Reversed Energy
Energy Start	-	Date and time of the last reset of all energy counters
Overall True Power Factor		*Historical compatibility only, do not use it.*
Overall Phase		*Historical compatibility only, do not use it.*

Parameters for **each power output**:

Variable	Unit	Description
Current	mA	Electric current for the output
Power Factor	-	TPF True Power Factor for the output
Phase	°	Phase for the specific power output
Energy	Wh	Counter of Energy consumed per output (resettable)
Energy NR	Wh	Not Resettable counter of output consumed Energy
Reverse Energy	Wh	Counter of Energy produced per output (resettable)
Reverse Energy NR	Wh	Not Resettable counter of Reversed (produced) Energy
Load	W	Instantaneous load (power) for the specific power output.

NETIO WEB configuration

M2M API protocols can be enabled and configured only over the web administration – select “M2M API Protocols” in the left-hand side menu and then select the “JSON API” tab.



☑ Enable JSON API

☐ Use custom port

Current port: 80

Custom port: 81

☑ Enable READ

Username:

Password:

☐ Enable WRITE

Username: write

Password: ••••••••••••

Note: This protocol does not take into account the output start interval - user is responsible for delays between output start. If you send ON action for all outputs together they will be turned on at the same time.

Picture 1 –M2M API Protocols / JSON API settings GUI

- **Enable JSON API** – Enable/disable the M2M API protocol
- **Use custom port** – Check to enable custom port setting
  - **Current port** – Currently used port
  - **Custom port** – Custom port set for XML / JSON / CGI API protocols
- **Enable READ** – Enable READ functionality
  - Username / Password for READ
- **Enable WRITE** – Enable WRITE functionality
  - Username – Username for WRITE
  - Password – Password for WRITE (default: MAC address without colons, lowercase)

Notes

- **The device webserver is restarted after saving the JSON API settings.**
- Empty Username and Password means no authentication.
- Credentials are sent in the HTTP header, “Basic authentication” is used.
  The username and password can be also provided in the URL -
  http(s)://username:password@<netioIP>/netio.json

NETIO JSON protocol structure

JSON standard: RFC4627
JSON Template: 3 Space Tab

JSON API – READ (status)

HTTP(s) GET request or HTTP(s) POST request (no file or empty file)

GET Request: **http(s)://<netioIP>/netio.json**

READ response (status json file):

```
{
   "Agent": {
      "Model": "PowerDIN 4PZ",
      "Version": "2.5.4",
      "JSONVer": "2.3",
      "DeviceName": "myNetio_10",
      "VendorID": 0,
      "OemID": 0,
      "MAC": "24:A4:2C:33:25:E1",
      "SerialNumber":"24A42C3325E1",
      "Uptime": 110637,
      "Time": "2017-11-03T13:53:38+00:00",
      "NumOutputs": 4
      "NumInputs": 2
   },
   "GlobalMeasure": {
      "Voltage": 235.8,
      "Frequency": 49.9,
      "TotalCurrent": 20,
      "OverallPowerFactor": 0.22,
      "Phase": -42.90,
      "TotalLoad": 1,
      "TotalEnergy": 965,
      "EnergyStart": "2017-06-23T16:47:53+01:00"
   },
   "Outputs":[
      {
         "ID": 1,
         "Name": "output_1",
         "State": 0,
         "Action": 6,
         "Delay": 5000,
         "Current": 0,
         "PowerFactor": 0,
         "Phase":0.00,
         "Load": 0,
         "Energy": 192,
         "ReverseEnergy":0
      },
      {
         "ID": 2,
         "Name": "output_2",
         "State": 0,
         "Action": 6,
         "Delay": 5000,
         "Current": 0,
         "PowerFactor": 0,
         "Phase":0.00,
         "Load": 0,
         "Energy": 80,
         "ReverseEnergy":0
      },
      {
         "ID": 3,
         "Name": "output_3",
         "State": 0,
         "Action": 6,
         "Delay": 5000,
         "Current": 0,
         "PowerFactor": 0,
         "Phase":0.00,
         "Load": 0,
         "Energy": 196,
         "ReverseEnergy":0
      },
      {
         "ID": 4,
         "Name": "output_4",
         "State": 1,
         "Action": 6,
         "Delay": 5000,
         "Current": 20,
         "PowerFactor": 0.22,
         "Phase": -42.90,
         "Load": 1,
         "Energy": 495,
         "ReverseEnergy":0
      }
   ],
"Inputs":[
    {
        "ID":1,
        "Name":"Input 1",
        "State":0,
        "S0Counter":7
    },
    {
        "ID":2,
        "Name":"Input 2",
        "State":0,
        "S0Counter":0
    }
]}
}
```

Notes:

1) *Old firmware versions contains the MAC tag only. When both tags available, use the SerialNumber, it's identical with printed label on the device.*

2) *Items/values related to metering (Voltage, Frequency, Current, PowerFactor, Load and Energy, etc.) are available only for the NETIO PowerPDU 4C, PowerCable, PowerDIN 4PZ and NETIO 4All models.*

3) *Returned status **netio.json** file always contains "Action" with value "6" for all outputs. This value means "ignore" and works as a placeholder. Output state 0 / 1 is in the **State** value.*

Values description

Global values:

`"Model": "NETIO 4All"`	*Model identification*
`"Version": "3.4.0"`	*Firmware version*
`"JSONVer": "2.3"`	*Protocol version*
`"DeviceName": "myNetio_10"`	*Device name (user defined on web)*
`"VendorID": 0`	*Manufacturer internal use*
`"OemID": 0`	*Manufacturer internal use*
`"MAC": "24:A4:2C:33:25:E1"`	*MAC address of active interface.* *For LAN/Wifi devices only.*
`"SerialNumber": "24A42C3325E1"`	*Serial Number of device – preferred identifier (identical with label on delivery box).*
`"Uptime": 110637`	***[s]** The Uptime value*
`"Time": "2017-11-03T13:53:38+00:00"`	*Date and time of the NETIO device*
`"NumOutputs": 4`	*Number of outputs*
`"NumInputs": 2`	*Number of inputs*
`"Voltage": 235.8`	***[V] I**nstantaneous voltage*
`"Frequency": 49.9`	***[Hz]** Instantaneous frequency*
`"TotalCurrent": 20`	***[mA]** Instantaneous total current through all power outputs*
`"Phase": -41.45`	***[°]** Instantaneous Phase weighted average from all meters*
`"TotalLoad": 1`	***[W]** Total Power of all power outputs*
`"TotalEnergy": 965`	***[Wh]** Counter of total consumed Energy (4B)*
`"TotalEnergyNR": 2530`	***[Wh]** Not resettable counter of total consumed Energy (4B)*
`"TotalReverseEnergy": 1`	***[Wh]** Counter of total produced Energy (4B)*
`"TotalReverseEnergyNR": 8`	***[Wh]** Not resettable counter of total produced Energy (4B)*
`"EnergyStart": "2017-06-23T16:47:53+01:00"`	*Date and time of the last reset of all energy counters*

Values for specific output (example values below are for output 4):

`"ID": 4`	*Output number*
`"Name": "output_4"`	*Output name (user defined on web)*
`"State": 1`	*Output state*
`"Action": 6`	*Output action (6 = Ignored value, use State tag)*
`"Delay": 5000`	***[ms]** Output delay for short On/Off*
`"Current": 20`	***[mA]** Instantaneous current of the output*
`"PowerFactor": 0.22`	*[-] Instantaneous True Power Factor*
`"Phase": -30.81`	***[°]** Instantaneous Phase of the output*
`"Load": 1`	***[W]** Total Power of the output*
`"Energy": 965`	***[Wh]** Counter of output consumed Energy (4B)*
`"EnergyNR": 2530`	***[Wh]** Not resettable counter of consumed Energy (4B)*
`"ReverseEnergy": 1`	***[Wh]** Counter of output produced Energy (4B)*
`"ReverseEnergyNR": 8`	***[Wh]** Not resettable counter of output produced Energy (4B)*

Values for specific input (example values below are for input 2):

`"ID": 2`	*Input number*
`"Name":"Input 2"`	*Input name*
`"State":0`	*Input state (0 = OFF/"open", 1= ON/"closed")*
`"S0Counter":25`	*S0 Counter value (4B)*

JSON API – WRITE (control)

HTTP(s) POST request

ID - number of output

Outputs can be controlled by two options:

1. **Action**: 0 – off, 1 – on, 2 – short off, 3 – short on, 4 – toggle, 5 – no change, (6 – ignore)
2. **State**: 0 – off, 1 – on   (Action = 6 required)

Note: ***Action*** *with other value than 6 has higher priority than the* ***State*** *tag.*
State value is not reflected in case Action = 1 to 5.
If you wish to use ***State*** *tag to control an output,* ***Action = 6 is required****.*

A json file can be submitted as complete structure (e.g. previously received status json with modified control functions) or partial structure as shown below.

If the json & command is accepted, then NETIO returns Status Code “200 OK” and status json file.

Send command: **http(s)://<netioIP>/netio.json**

Switch Power output 1 to ON by Action tag:

```
{
   "Outputs":[
      {
         "ID":1,
         "Action":1
      }
   ]
}
```

or (State tag value will not be reflected)

```
{
   "Outputs":[
      {
         "ID":1,
         "State":0,
         "Action":1
      }
   ]
}
```

or by State tag (Action tag must have value 6)

```
{
   "Outputs":[
      {
         "ID":1,
         "State":1,
         "Action":6
      }
   ]
}
```

Switch Power output 2 to ON for 15 seconds, then switch it OFF.

```
{
   "Outputs":[
      {
         "ID":2,
         "Action":3,
         "Delay":15000
      }
   ]
}
```

Command to control more outputs:
Switch Power output 1 to ON, Toggle Output 2 and Switch Output 4 to ON for 15 seconds:

```
{
   "Outputs":[
      {
         "ID":1,
         "Action":1
      },
      {
         "ID":2,
         "Action":4
      },
      {
         "ID":4,
         "Action":3,
         "Delay":15000
      }
   ]
}
```

Status codes

Status codes	Description
200 OK	User authorized and command received
400 Bad Request	Control command syntax error
401 Unauthorized	Invalid Username or Password
403 Forbidden	Read only
500 Internal Server Error	Internal Server Error or Internal Server not fully started yet (e.g. after setting change or restart)

Response syntax for "OK" state:
Status json file as described above in chapter "NETIO JSON protocol structure" / READ

Response syntax for "Error" state:

```
{
   "result": {
      "error": {
         "code": 200,
         "message": "OK"
      }
   }
}
```

NETIO PowerBOX 3Px – listing of the netio.json file

Note: In the NETIO PowerBOX 3Px model, there are no metering values available.

```
{

"Agent":{"Model":"3PF","DeviceName":"PowerBOX-
F2","MAC":"24:A4:2C:38:DF:F2","SerialNumber":"24A42C38DFF2","JSONVer":"2.3","Time"
:"1970-01-
01T23:05:55+01:00","Uptime":36355,"Version":"2.5.4","OemID":0,"VendorID":0,"NumOut
puts":3},

"Outputs":[

{"ID":1,"Name":"Power output 1","State":0,"Action":6,"Delay":2020},

{"ID":2,"Name":"Power output 2","State":1,"Action":6,"Delay":2020},

{"ID":3,"Name":"Power output 3","State":1,"Action":6,"Delay":2020}

]}
```

NETIO PowerDIN 4PZ – listing of the netio.json file

Note: In the NETIO PowerDIN 4PZ model are just 2 from 4 outputs metered.

```
{
"Agent":{"Model":"4PZ","DeviceName":"powerdin-
4pz","MAC":"24:A4:2C:39:67:17","SerialNumber":"24A42C396717","JSONVer":"2.3","Time
":"2021-03-
24T01:15:32+01:00","Uptime":6020,"Version":"3.0.1","OemID":400,"VendorID":0,"NumOu
tputs":4,"NumInputs":2},

"GlobalMeasure":{"Voltage":241,"TotalCurrent":0,"TotalLoad":0,"TotalEnergy":0,"Ove
rallPowerFactor":0.00,"Frequency":50.07,"Phase":0.00,"EnergyStart":"2020-12-
13T19:34:31+01:00"},

"Outputs":[
{"ID":1,"Name":"Power output
1","State":0,"Action":6,"Delay":2020,"Current":0,"PowerFactor":1.00,"Phase":0.00,"
Energy":0,"ReverseEnergy":0,"Load":0},
{"ID":2,"Name":"Power output
2","State":1,"Action":6,"Delay":2020,"Current":0,"PowerFactor":1.00,"Phase":0.00,"
Energy":0,"ReverseEnergy":0,"Load":0},
{"ID":3,"Name":"Free Contact 3","State":0,"Action":6,"Delay":2020},
{"ID":4,"Name":"Free Contact 4","State":1,"Action":6,"Delay":2020}
],

"Inputs":[
{"ID":1,"Name":"Intput 1","State":0,"S0Counter":290},
{"ID":2,"Name":"Intput 2","State":0,"S0Counter":1}
]}
```

NETIO PowerPDU 8QS – listing of the netio.json file

Note: In the NETIO PowerPDU 8QS model are metered just 2 channels (Output 1 & global) from total 8 outputs.

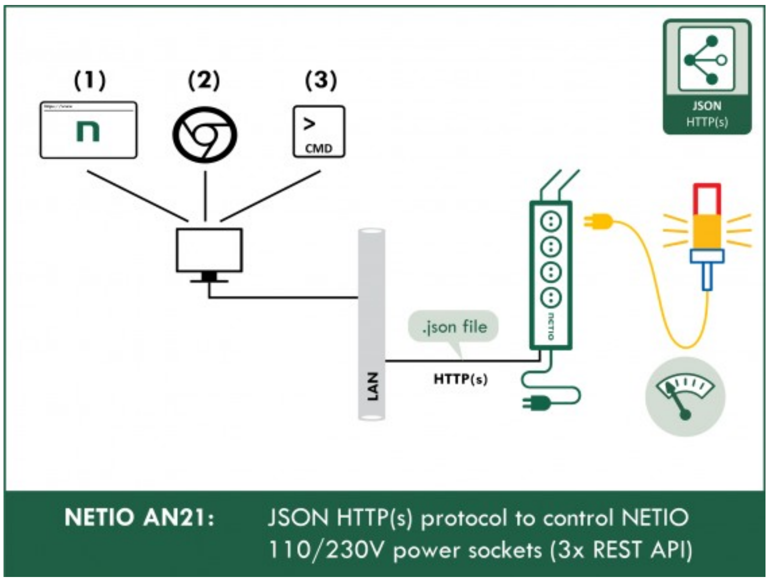
```
{
"Agent":{"Model":"8QS","DeviceName":"PowerPDU-
F3","MAC":"24:A4:2C:39:9F:F3","SerialNumber":"24A42C399FF3
","JSONVer":"2.4","Time":"2021-06-
08T13:44:49+01:00","Uptime":76239,"Version":"3.1.6","OemID
":600,"VendorID":0,"NumOutputs":8,"NumInputs":1},
"GlobalMeasure":{"Voltage":240.97,"TotalCurrent":0,"Overal
lPowerFactor":1.00,"TotalPowerFactor":1.00,"OverallPhase":
0,"TotalPhase":0,"Frequency":50.09,"TotalEnergy":7,"TotalR
everseEnergy":1,"TotalEnergyNR":7,"TotalReverseEnergyNR":1
,"TotalLoad":0,"EnergyStart":"1970-01-01T00:00:00+01:00"},
"Outputs":[
{"ID":1,"Name":"Power output
1","State":0,"Action":6,"Delay":5000,"Current":0,"PowerFac
tor":1.00,"Phase":0.00,"Energy":7,"ReverseEnergy":0,"Energ
yNR":7,"ReverseEnergyNR":0,"Load":0},
{"ID":2,"Name":"Power output
2","State":1,"Action":6,"Delay":5000},
{"ID":3,"Name":"Power output
3","State":0,"Action":6,"Delay":5000},
{"ID":4,"Name":"Power output
4","State":0,"Action":6,"Delay":5000},
{"ID":5,"Name":"Power output
5","State":1,"Action":6,"Delay":5000},
{"ID":6,"Name":"Power output
6","State":1,"Action":6,"Delay":5000},
{"ID":7,"Name":"Power output
7","State":1,"Action":6,"Delay":5000},
{"ID":8,"Name":"Power output
8","State":1,"Action":6,"Delay":5000}
],
"Inputs":[
{"ID":1,"Name":"Input 1","State":1,"S0Counter":0}
]}
```

Examples

AN21: JSON HTTP(S) protocol to control NETIO 110/230V power sockets (3x REST API)

The AN21 Application Note shows how to access measurements and control electrical sockets on a NETIO 4x device from third-party applications using the JSON protocol. AN21 demonstrates several different ways to control NETIO power sockets by transferring a netio.json file over http.

The first method uses the “Device HTTP(s) File Upload” tool in the device’s web interface. The second method transfers the JSON file using a Chrome browser extension. The third method uses CURL (command-line tool) to transfer files over http.


(1)
(2)
(3)
CMD
JSON HTTP(s)
.json file
HTTP(s)
LAN
NETIO AN21: JSON HTTP(s) protocol to control NETIO 110/230V power sockets (3x REST API)

>> Read the AN21 on www.netio-products.com

Document history

Document Revision	Publication Date	Description
1.0	14.11.2017	Initial release - JSON Version 2.0, for FW 3.0.1 (Netio4x devices)
1.1	7.12.2017	Documentation optimization - Action 6
1.2	19.12.2017	Keywords added
1.3	31.8.2018	Values description updated
1.4	6.9.2018	Infographic added
1.5	16.10.2018	Minor edits
1.6	23.11.2018	Detailed description about the Action tag implemented
1.7	29.1.2020	Added SerialNumber & MAC description for PowerCable/PowerBox/PowerPDU/PowerDIN devices JSON version changed to 2.1
1.8	29.1.2020	Version -> 2.2 (due to inconsistencies with N4, where 2.1 already was)
1.9	19.3.2020	Detail description of difference between MAC and SerialNumber tag was added.
2.0	11.9.2020	JSON Version -> 2.3: Added Inputs, Phase, ReverseEnergy; new supported devices
2.1	24.3.2021	New compatible devices listed
2.2	17.6.2021	JSON Version -> 2.4: Added Reversed energy & DI tags into the JSON structure
2.3	20.7.2021	Minor edits