

How to Use Lua Scripts and RS-232 Serial Ports to Control The Outlet Sockets on a PowerPDU-4C

This application note describes how to use a Lua script to control the outlet sockets on a PowerPDU-4C smart power management device.

The **PowerPDU4-C** has a built-in RS-232 port through which the power outlets can be set to ON/OFF using a **request-response** protocol and Lua script. The result is a text-based protocol that is the same as a Telnet M2M protocol.

In the device configuration, the serial port (serial console) can be connected to a specified TCP/IP port or redirected to a Lua script (as used in the script described later in this application note). The Lua script reacts to the string received over the serial port, sends a response and switches any of the four specified power outlets ON/OFF.

In order to work with the serial port, several parameters need to be specified including baud rate, block delimiter, handshake. The script itself then uses several basic functions to read the message received over the serial port, parse it, perform the requested action and send back a response. Example of a command to switch on the output no. 1: port 1 1

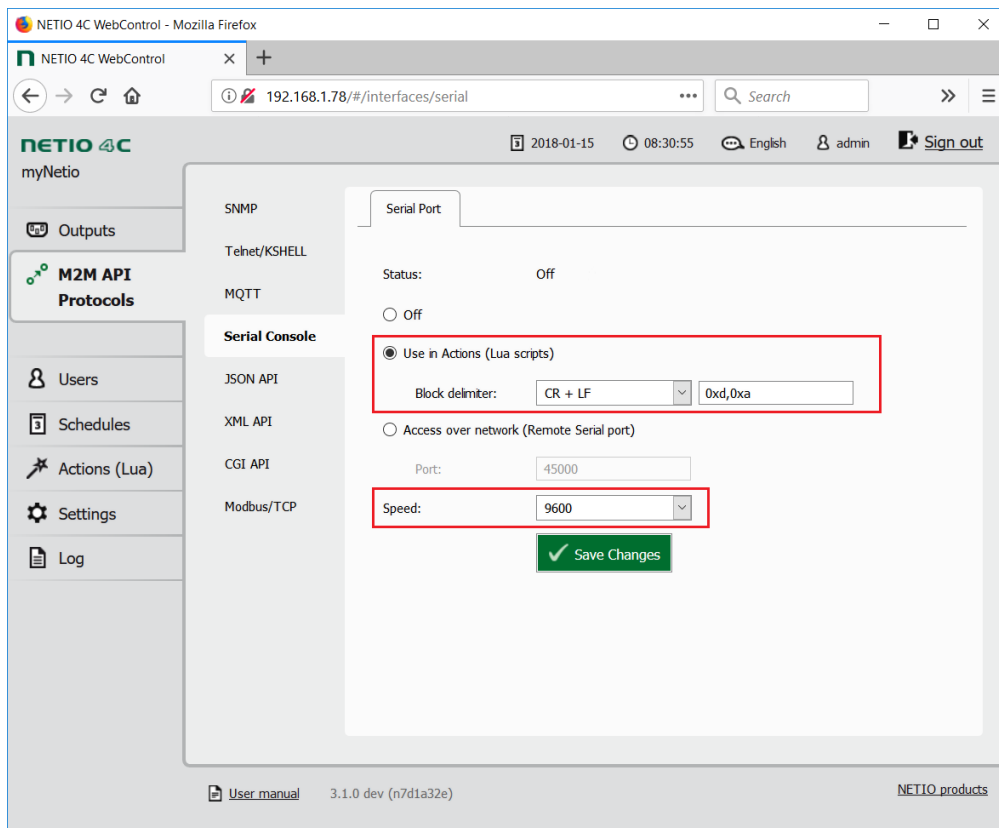
The Lua script includes 2 user-defined messages. The user can define the format of these messages and the contained commands.

1. Serial Port Settings

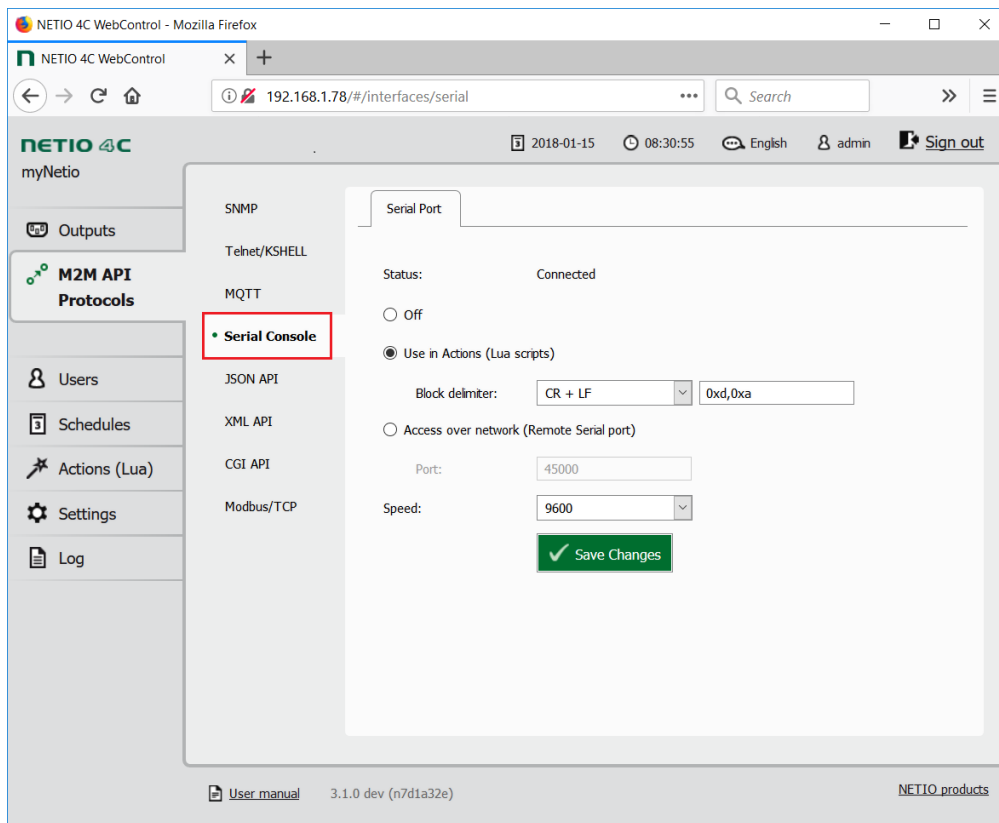
In the web administration, go to M2M API Protocols, click Serial Console and select Use in Actions (Lua scripts).

- The block delimiter is a character that terminates the message and starts the Lua script. It is possible to choose from pre-defined characters (CR+LF, CR, LF, NULL), or to define a custom delimiter.
- For a custom delimiter, the requested character must be entered in hexadecimal form (0x followed by the character code) in the field on the right.
- For the pre-defined characters, the field is filled automatically (the hexadecimal form of CR+LF is "0xd" and "0xa").
- Speed sets the baud rate.
- Click Save Changes to confirm the settings.





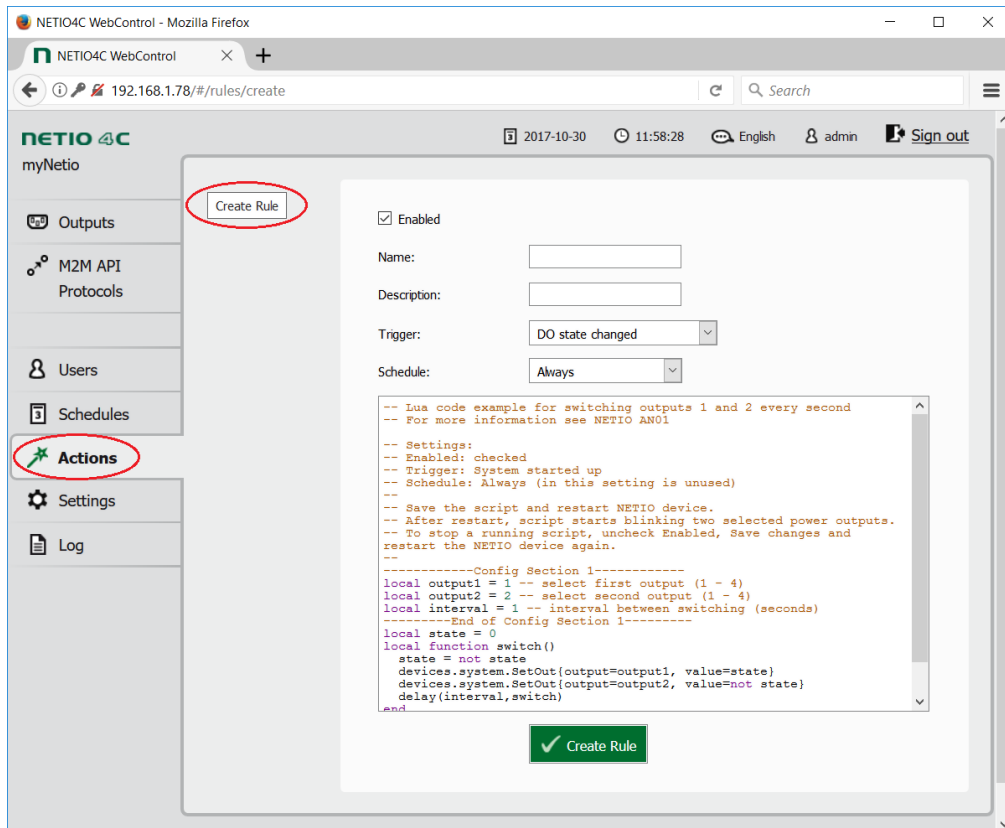
After the serial port is activated, a dark green dot appears next to the tab and the Status changes to “Connected”.



2. Creating The Rule

To create and run the Lua script, do the following:

2.1 In the Actions section of NETIO 4 web administration, click Create Rule to add a rule.



2.2 Fill in the following parameters:

- Enable rule: checked
- Name: Serial console to Lua (user-defined)
- Description: Parse message from Serial console and respond (user-defined)
- Trigger: Serial console has read line
- Schedule: any (does not affect the function in this configuration)

2.3 Copy the following code and paste it into the field for the Lua script:

<-----start of code example----->

```
-----Section 1-----
local shortTimeMs = 1000 -- "short" time for short on/off [ms]
local definableQuestion1 = "^all%-off$" -- prepared customer
string 1
local definableQuestion2 = "^all%-on$" -- prepared customer
string 2
local endingString = "\r\n" -- symbol sent at the end of each
answer (CR+LF in the default setting)
-----End of Section 1-----

-----Section 2-----
-- Serial port - Telnet commands definition
```

```
local portValue = "^port %d$"
local portSet = "^port %d %d$";
local portListValue = "^port list$"
local portListSet = "^port list %d%d%d%d$"
-----End of Section 2-----

local id = event.args.id -- internal variable, do not change
local message = event.args.message -- internal variable, do not change
logf("%s",message)
local function short(output,state)
    devices.system.SetOut{output=output,value=state}
end

local function isBetween(number,lowLimit,highLimit)
    if number >= lowLimit and number <= highLimit then
        return true
    end
    return false
end

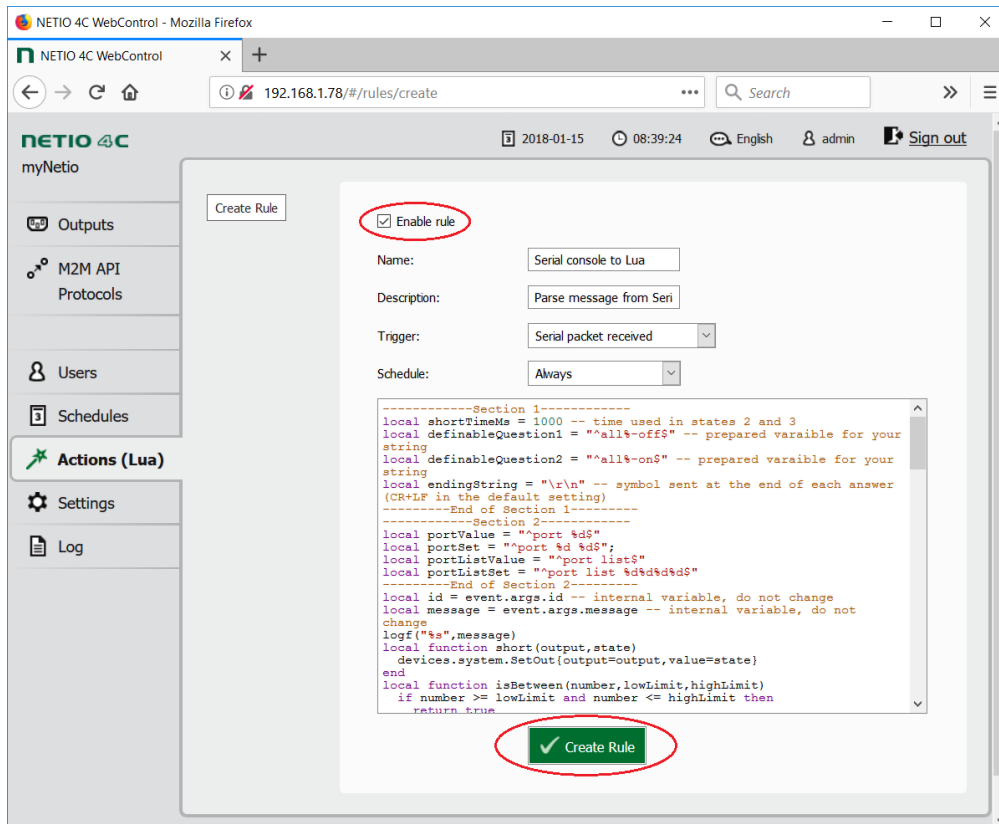
local function setOutput(output,action)
    if action == 0 then -- turn off
        devices.system.SetOut{output = output, value = false}
    elseif action == 1 then -- turn on
        devices.system.SetOut{output = output, value = true}
    elseif action == 2 then -- short off
        devices.system.SetOut{output = output, value = false}
        milliDelay(shortTimeMs,function() short(output,true) end)
    elseif action == 3 then -- short on
        devices.system.SetOut{output = output, value = true}
        milliDelay(shortTimeMs,function() short(output,false) end)
    elseif action == 4 then -- toggle
        if devices.system["output" ..output.. "_state"] == 'on' then
            devices.system.SetOut{output=output,value=false}
        else
            devices.system.SetOut{output=output, value=true}
        end
    elseif action == 5 then
        -- do nothing
    end
end

if string.match(message,portValue) then -- return output state
    local output = tonumber(message:sub(6,6))
    if not isBetween(output,1,4) then
        devices.system.SerialWrite{id=id,message="501 INVALID
PARAMETR" .. endingString}
        return
    end
    local answer = "250 0"
    if devices.system["output" ..output.. "_state"] == 'on' then
        answer = "250 1"
    end
    devices.system.SerialWrite{id=id,message=answer..endingString}
elseif string.match(message,portSet) then -- set output
    local output = tonumber(message:sub(6,6))
    local action = tonumber(message:sub(8,8))
    if not isBetween(output,1,4) or not isBetween(action,0,5) then
```

```
        devices.system.SerialWrite{id=id,message="501 INVALID
PARAMETR" .. endingString}
        return
    end
    setOutput(output,action)
    devices.system.SerialWrite{id=id,message="250 OK" ..
endingString}
elseif string.match(message,portListValue) then -- return output
status
    local answer = "250 "
    for i=1,4 do
        if devices.system["output" ..i.. "_state"] == 'on' then
            answer = answer .. "1"
        else
            answer = answer .. "0"
        end
    end
    devices.system.SerialWrite{id=id,message=answer ..
endingString}
elseif string.match(message,portListSet) then -- set outputs
    local actions = {};
    for i=1,4 do
        actions[i] = tonumber(message:sub(10+i,10+i))
        if not isBetween(actions[i],0,5) then
            devices.system.SerialWrite{id=id,message="501 INVALID
PARAMETR" .. endingString}
            return
        end
    end
    for i=1,4 do
        setOutput(i,actions[i])
    end
    devices.system.SerialWrite{id=id,message="250 OK" ..
endingString}
elseif string.match(message,definableQuestion1) then
    -- command 1 start
    for i=1,4 do
        setOutput(i,0)
    end
    devices.system.SerialWrite{id=id,message="250 OK" ..
endingString}
    -- command 1 end
elseif string.match(message,definableQuestion2) then
    -- command 2 start
    for i=1,4 do
        setOutput(i,1)
    end
    devices.system.SerialWrite{id=id,message="250 OK" ..
endingString}
    -- command 2 end
else
    devices.system.SerialWrite{id=id,message="502 UNKNOWN COMMAND"
.. endingString}
    logf("%s",message)
end
```

<-----end of code example----->

2.4 To finish creating the rule, click Create Rule at the bottom of the screen.



NETIO 4C WebControl - Mozilla Firefox

192.168.1.78/#/rules/create

2018-01-15 08:39:24 English admin Sign out

myNetio

Outputs

M2M API Protocols

Users

Schedules

Actions (Lua)

Settings

Log

Create Rule

☒ Enable rule

Name: Serial console to Lua

Description: Parse message from Seri

Trigger: Serial packet received

Schedule: Always

```

-----Section 1-----
local shortTimeMs = 1000 -- time used in states 2 and 3
local definableQuestion1 = "^all%-off$" -- prepared variable for your
string
local definableQuestion2 = "^all%-on$" -- prepared variable for your
string
local endingString = "\r\n" -- symbol sent at the end of each answer
(CR+LF in the default setting)
-----End of Section 1-----
-----Section 2-----
local portValue = "port %d$"
local portSet = "port %d %d$";
local portListValue = "port list $"
local portListSet = "port list %d%d%d$d$"
-----End of Section 2-----
local id = event.args.id -- internal variable, do not change
local message = event.args.message -- internal variable, do not
change
logf("%s",message)
local function short(output,state)
    devices.system.SetOut(output=output,value=state)
end
local function isBetween(number,lowLimit,highLimit)
    if number >= lowLimit and number <= highLimit then
        return true
    end

```

✓ Create Rule

2.5 Description of The Code

- The message received over the serial port is read to the message variable with this command: `local message = event.args.message`
- The message contains everything sent over the serial line, except the message-terminating character.
- The message format is compared to the predefined patterns, or the user-defined patterns if appropriate (see below) using the `string.match` function. In case of a match, the corresponding section is executed and the action performed.

2.6 Variables

The code contains two variables that can be set:

- **shortTimeMs**
 - Specifies (in milliseconds) for how long is the output switched on or off for actions 2 and 3 respectively (short off / short on).
 - The minimum value is 100 ms.
 - Example – to turn on for 2 seconds: `shortTimeMs = 2000`
- **endingString**
 - Specifies the symbol to terminate all responses that are sent back over the serial line.
 - The default is the CR+LF symbols (in Lua expressed as `\r\n`).

- In a robust protocol, this terminating string will be the same as the string that starts the Lua script.

2.7 List of Basic Commands:

The patterns for these commands are coded in Section2 (*portValue*, *portSet*, *portListValue*, *portListSet* variables).

- **port** <output> <action>
 - Controls one particular socket.
 - <output> is the socket number
 - <action> is one of the possible parameters
 - If <action> is omitted, the command prints the socket state.
- **Parameters:**
 - 0 – turns the socket off
 - 1 – turns the socket on
 - 2 – briefly turns the socket off (if the socket was off, the command turns it on)
 - 3 – briefly turns the socket on (if the socket was on, the command turns it off)
 - 4 – toggles the socket state
 - 5 – leaves the socket state unchanged
 - Example – turn on socket 1: port 1 1
 - Example – turn off socket 3: port 3 0
- **port list** [xxxx]
- Controls all sockets at once.
- Without parameters, lists the socket states.
- List of parameters:
 - 0 – turns the socket off
 - 1 – turns the socket on
 - 2 – briefly turns the socket off (if the socket was off, the command turns it on)
 - 3 – briefly turns the socket on (if the socket was on, the command turns it off)
 - 4 – toggles the socket state
 - 5 – leaves the socket state unchanged
 - Example – turn all sockets on: port list 1111
 - Example – turn on sockets 1 and 3, turn off socket 4 and toggle socket 2: port list 1410
 - Example – turn on sockets 2 and 3 and leave sockets 1 and 4 unchanged: port list 5115
- Note: All “x” in the command need to be replaced with an actual parameter. If the state of a particular socket should not change, use “5” as the parameter. For example, port list 10x1 is not a valid command.

2.8 User-defined RS-232 Commands

The *definableQuestion1* and *definableQuestion2* variables are prepared in the code for creating custom request patterns.

The custom pattern must be enclosed in double quotes and the ^ and \$ characters. This convention ensures that the message received must exactly match the pattern.

- Example – for the *all-on* request: **definableQuestion** = "^all%-on\$"

If the request includes a symbol that serves as a variable, the pattern can represent it with a wildcard (. for any character, %d for a decimal digit).

- Example – the predefined command to check the status of a socket uses "port %d":
portValue = "^port %d\$"
- The actual value of the number cannot be checked in the request. If you want to react to a specific number only, include it literally in the string (**definableQuestion1** = "^getState 1\$"). If the request pattern contains a space, the message must contain it, too.

When creating custom requests, care must be taken so that the patterns do not overlap. For example, "^port %d\$" and "^port 1\$". In this case, a port 1 message would execute the command that is defined earlier in the code.

The following characters are called "magic" in Lua. If they need to be present in the message, they must be escaped with the % sign:

- () . % + - * ? [^ \$
- Example – for the *all-off* request: **definableQuestion** = "^all%-off\$"

The process of composing a response is described below.

Creating a Response to Be Sent After Receiving A Request

For both variables, a code section to perform the requested response is prepared. It is located near the end of the code (see the picture).

```

end
end
for i=1,4 do
    setOutput(i,actions[i])
end
devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
elseif string.match(message,definableQuestion1) then
    -- command 1 start
    for i=1,4 do
        setOutput(i,0)
    end
    devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
    -- command 1 end
elseif string.match(message,definableQuestion2) then
    -- command 2 start
    for i=1,4 do
        setOutput(i,1)
    end
    devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
    -- command 2 end
else
    devices.system.SerialWrite{id=id,message="502 UNKNOWN COMMAND" ..
    endingString}
    logf("%s",message)
end

```

Replace the sample code (between the comments) with your code to perform the desired action.

- To make things easier, the following helper functions are implemented:
 - **setOutput(output,action)**
 - Changes the state of an **output** according to the specified **action**.

- The action parameter can have one of the 5 values (see the description of the port command).
- Example – to toggle output 3: `setOutput(3,4)`
- **isBetween(number,lowLimit,highLimit)**
 - Returns true if the value of the first parameter (number) is between the values of the second and third parameter (lowLimit and highLimit).
 - Examples:
 - `isBetween(1,1,4)` returns true
 - `isBetween(0,1,4)` returns false
 - This function is used to check the input parameters.

`devices.system.SerialWrite{id=id,message="Your message" .. endingString}` can be used to send a message back over the serial line.

- Replace **"Your message"** with a message string (in double quotes) or a variable (without quotes).
- The **.. endingString** adds the terminating string at the end of the message. If you do not wish to terminate the message with that string, delete this part (including the two dots).

Even if you do not want to send any message back, you **must** send at least the terminating string: `devices.system.SerialWrite{id=id,message=endingString}`

2.9 Frequently Asked Questions (FAQs)

2.9.1 Can I have more custom requests (commands)?

Yes. To add another pattern, create a new variable (e.g. `definableMessage3`) with the pattern. Then, it is necessary to add another condition and specify the corresponding action in the code. Insert the following condition just above the last else statement: **elseif `string.match(message,definableMessage3)` then**

Code your action immediately after this condition. If you chose a different variable name, remember to change it in the condition, too. Your code should now look like this:

```
for i=1,4 do
    setOutput(i,actions[i])
end
devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
elseif string.match(message,definableQuestion1) then
    -- command 1 start
    for i=1,4 do
        setOutput(i,0)
    end
    devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
    -- command 1 end
elseif string.match(message,definableQuestion2) then
    -- command 2 start
    for i=1,4 do
        setOutput(i,1)
    end
    devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
    -- command 2 end
elseif string.match(message,definableQuestion3) then
    -- new section for your action
else
    devices.system.SerialWrite{id=id,message="502 UNKNOWN COMMAND" ..
endingString}
    log("%s",message)
end
end
```

2.9.2 I don't want to use the basic commands. Is it possible to disable them?

Simply delete the corresponding section in the code below the setOutput function. The code should now look like this:

```
if devices.system["output" ..output.. "_state"] == 'on' then
    devices.system.SetOut{output=output,value=false}
else
    devices.system.SetOut{output=output, value=true}
end
elseif action == 5 then
    -- do nothing
end
end

if string.match(message,definableQuestion1) then
    -- command 1 start
    for i=1,4 do
        setOutput(i,0)
    end
    devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
    -- command 1 end
elseif string.match(message,definableQuestion2) then
    -- command 2 start
    for i=1,4 do
        setOutput(i,1)
    end
    devices.system.SerialWrite{id=id,message="250 OK" .. endingString}
    -- command 2 end
```

2.9.3 Why does the example use “\r\n” as the terminating string?

This is the default CR+LF string (see Serial port settings). It is necessary that the terminating string is the same as the string that activates the Lua script; in Lua, the CR+LF string is represented as “\r\n”.

2.9.4 Is it possible to get the consumption measurement data?

NETIO 4C cannot measure consumption, so it is not possible to get such data.

2.9.5 Is it possible to change the serial port parameters from the Lua script?

No. This is possible only in the serial port settings.

3.0 Supported FW Versions

3.1.0 and higher

3.1 Document Number

SRE-AN001-edition-1

3.2 Further Information

For more information visit on the PowerPDU-4C

<https://www.serverroomenvironments.co.uk/netio-powerpdu-4c-metered-pdu>