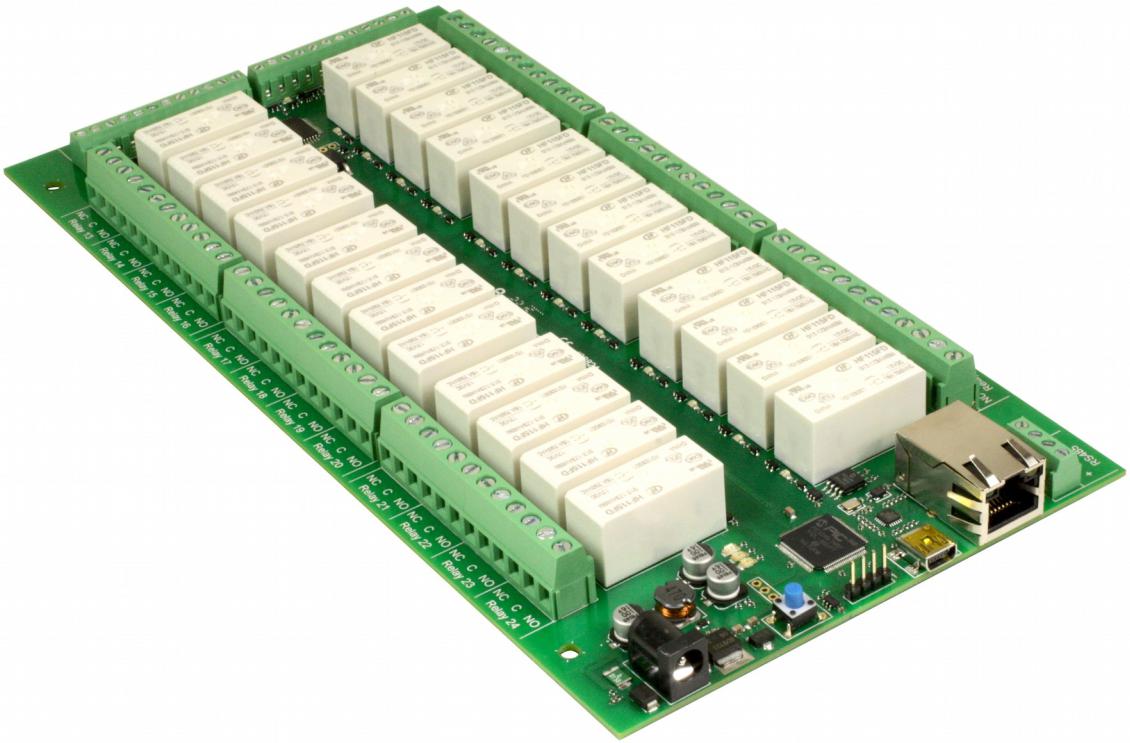
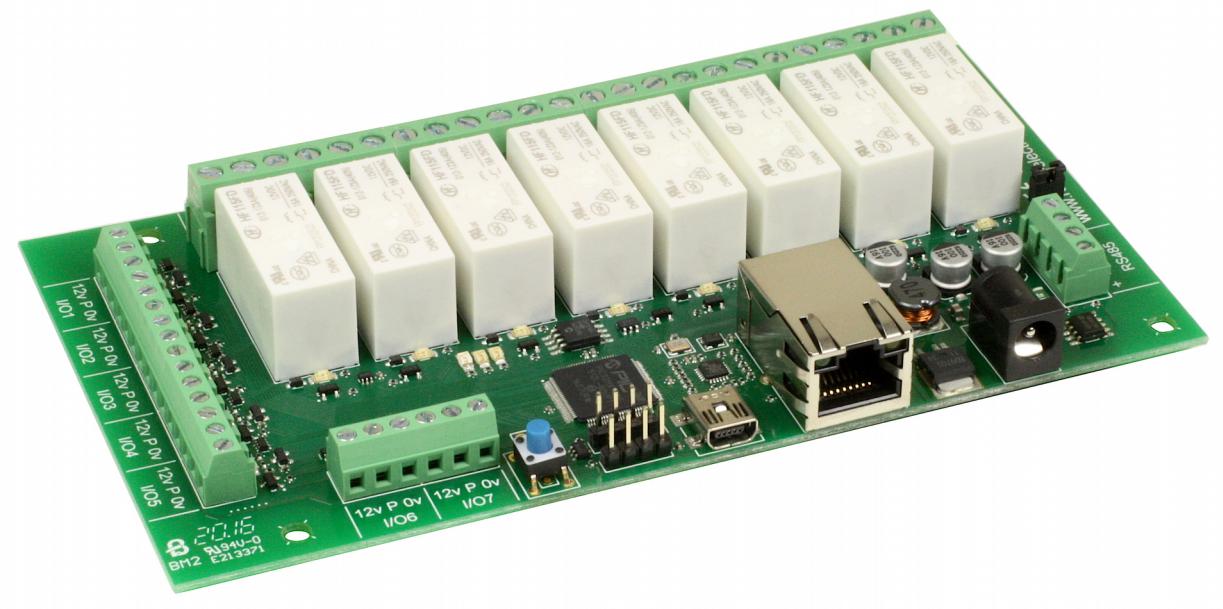
**dScript RoomControl**

User Manual Version 1.0

2020-07-20

# Document history

|  |  |
| --- | --- |
| **Version** | **Change reference** |
| 1.0 | Martin Kraemer – private / personal usage only |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Table of Contents**

[Document history 2](#_Toc46142273)

[1. Document description 5](#_Toc46142274)

[2. Getting started 6](#_Toc46142275)

[3. Configuring the module / application 7](#_Toc46142276)

[a) Enabling the configuration pages 7](#_Toc46142277)

[b) Status page 7](#_Toc46142278)

[c) Network page 7](#_Toc46142279)

[d) The TCP/IP 7](#_Toc46142280)

[e) Webpage Security 7](#_Toc46142281)

[f) Entities Page 8](#_Toc46142282)

[g) Input/Output 9](#_Toc46142283)

[h) Shutters 10](#_Toc46142284)

[i) Debug 10](#_Toc46142285)

[j) Application Page 10](#_Toc46142286)

[4. Hardware wiring 11](#_Toc46142287)

[a) Connecting a Touch Sensor / Switch 11](#_Toc46142288)

[b) Connecting a Motion / Presence Sensor 11](#_Toc46142289)

[c) Connecting a Light 11](#_Toc46142290)

[d) Connecting a Socket 11](#_Toc46142291)

[e) Connecting a Shutter 12](#_Toc46142292)

[5. Calibrating the shutters 13](#_Toc46142293)

[6. Short list – multi click commands 14](#_Toc46142294)

[7. Remote administration scripts 15](#_Toc46142295)

[8. TCP/IP command sets 16](#_Toc46142296)

[a) Binary command set 16](#_Toc46142297)

[b) AES binary command set 20](#_Toc46142298)

[9. The dScriptServer 21](#_Toc46142299)

[10. Home Assistant integration 22](#_Toc46142300)

[11. Custom programming 23](#_Toc46142301)

[12. The legal stuff 24](#_Toc46142302)

[a) Disclaimer 24](#_Toc46142303)

[b) General Public License 24](#_Toc46142304)

# Document description

This documentation covers different robot-electronic modules and the custom application “dScriptRoomControl” which was developed to provide an easy possibility to use dSXXXX modules for home automation purposes. While the application was tested only with dS3484, DS378 and dS2484 other dScript modules should work as well.

This document focuses on the “dScriptRoomControl” application specific information. For general information on how to use your dScript module or programming in dScript language please check out the <https://www.robot-electronics.co.uk/dscript.html> webpage.

**room-control application features:**

* Controlled graphically by secure webpage or optionally one of
* Binary or Binary AES command sets over TCP/IP.
* Also available when optionally programming in dScript is a TTL level serial port and an RS485 serial port.

**dS2484 module:**

* Ethernet connected module, 10/100Mb auto negotiated.
* Relays – 24 x 16Amp 250Vac C/O.
* I/O – 8 x flexible I/O's NPN output, Volt free input or 12-bit analogue input.
* Power – 12VDC 1.5Amp supply required. 2.1mm center positive.
* Connections – Screw Terminals for N/O N/C and Common contacts
* PCB size – 249mm x 123mm

**dS378 module:**

* Ethernet connected module, 10/100Mb auto negotiated.
* Relays – 8 x 16Amp 250Vac C/O.
* I/O – 7 x flexible I/O's NPN output, Volt free input or 10-bit analogue input.
* Power – 12VDC 1Amp supply required. 2.1mm center positive.
* Connections – Screw Terminals for N/O N/C and Common contacts
* PCB size – 156mm x 84mm

**dS3484 module:**

* Ethernet connected module, 10/100Mb auto negotiated.
* Relays – 4 x 16Amp 250Vac C/O.
* I/O – 8 x digital I/O's NPN output, Volt free input
* 4 x 10-bit analogue input
* Power – 12VDC 1Amp supply required. 2.1mm center positive.
* Connections – Screw Terminals for N/O N/C and Common contacts
* PCB size – 156mm x84mm

# Getting started

Start by plugging in the Ethernet cable to connect the module to your network, and the 12v jack plug from your adapter.

As soon as you switch on you might notice the red LED blinking 6 times (needing 3 seconds of time). If this happens it indicates the current application version was loaded the first time and all configuration values are set to their defaults (init() function is executed). If the red LED does not flash this is no error - it just means no new application version was loaded and the last state of configuration values is still active.

After this the blue LED will flash X times. This indicates the major version of the application.

This is followed by the green LED flashing x times, which indicates the minor version of the application loaded.

Please be patient while the red LED is still running. This indicates that the module is still initializing. After the red LED was turned off, the green LED is still running. This is just an indication that the module is still up and you can continue with normal processing.

# Configuring the module / application

## Enabling the configuration pages

There are a set of configuration pages to get the module operating as you want it. These pages are all \_configx.htm, (that's a leading underscore character).

ie.

\_config.htm

\_config2.htm

Anything that starts with \_config is considered a special name for configuration pages and can only be seen if you have the the USB cable plugged in and connected to your PC.

Why only if the USB cable is plugged in?

Its an additional security measure. After you have configured and deployed the module, you really don't want these configuration pages available for others to change. With the USB cable disconnected the \_config pages are not available. If you try to access them, you just get served a “not authorised” page instead.

For now, you do want to look over the config pages. If you have a Win10, Linux or MAC PC, you can go right ahead and plug in the USB cable. These machines will install their own USB drivers. We won't be sending anything to the board, its just the presence of the USB connection that enables the config pages to be served. If you have a Win7 or Win8 PC you will need to install the drivers. Please check the official module board documentation about this task:

<https://www.robot-electronics.co.uk/dscript.html>.

With the USB cable connected, browse to: 192.168.0.9/\_config.htm

(substituting your IP address)

## Status page

Please check the official module board documentation about this page.

<https://www.robot-electronics.co.uk/dscript.html>

## Network page

Please check the official module board documentation about this page.

<https://www.robot-electronics.co.uk/dscript.html>

## The TCP/IP

Please check the official module board documentation about this page.

<https://www.robot-electronics.co.uk/dscript.html>

Additionally, to the default settings this page contains an input box for the “dScriptServer”.

The “dScriptServer” is a server provided via python modul. This server can receive consolidated information about multiple dScirpt modules and control them to allow integration e.g. into home automation systems (read more about this within “The dScriptServer” chaper of this document).

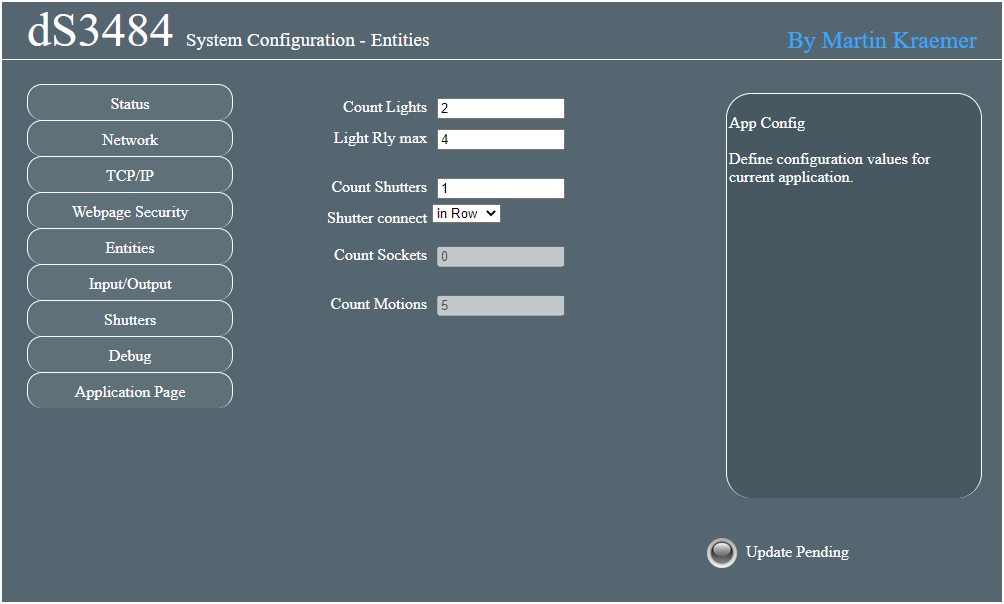
Be careful with inserting non existing / offline or an invalid (e.g. “ “) ip address into this value as this will cause delay when performing IO actions (such as turning a light on/off via a button press) due to the timeout delay of non-reachable dScriptServer.

## Webpage Security

Please check the official module board documentation about this page.

<https://www.robot-electronics.co.uk/dscript.html>

## Entities Page



This page provides settings on the entities managed by this dScriptBoard.

You can define the number of lights managed via the DSBoard using “Count Lights” input.

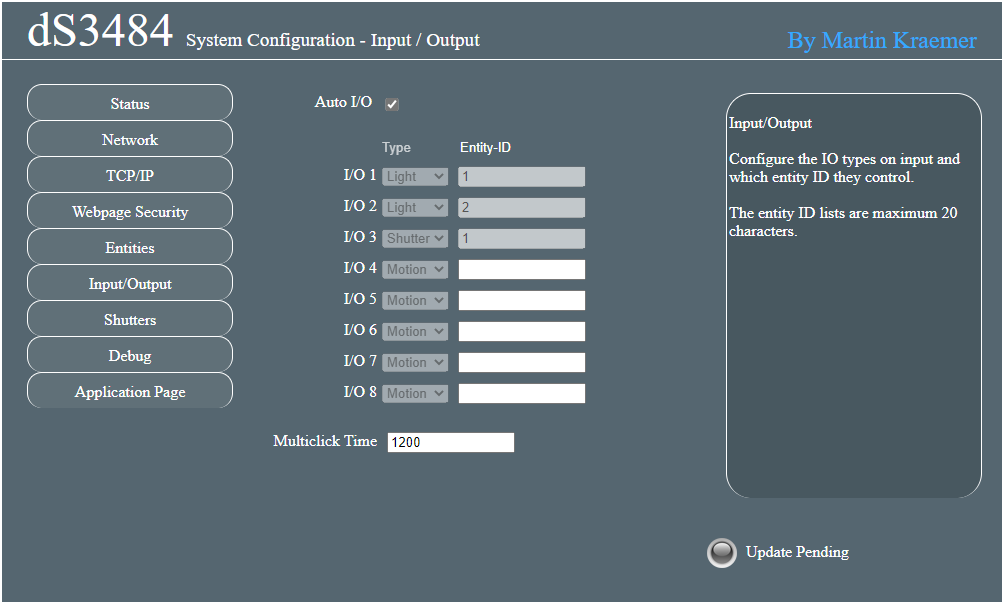
The “Light Rly max” defines the relay ID of the first light. Lights are counted backwards. From highest relay to lowest. This number must be >= lights + shutters\*2 to ensure enough relays are reserved. This is the highest relay ID used for a light but at the same time the first light ID. The “Light Rly max” value updates automatically in case you change the number of lights or shutters to a value which requires more relays.

The number of shutters managed via the DSBoard are defined via “Count Shutters” input.

As you can connect the shutter relays either “in Row” or “in Parallel” it is possible to switch the connection type for **all shutters** here. For details please check the chapter about “[Connecting a Shutter](#_Connecting_a_Shutter)”.

The “Count Sockets” and “Count Motions” are read only values which are calculated automatically based on the given parameters previously or in “Input/Output” tab. They are listed here only for the convenience of a fast overview.

## Input/Output



Configures the IO types of each input and which entity IDs they control. The entity ID lists can be a maximum 20 characters long.

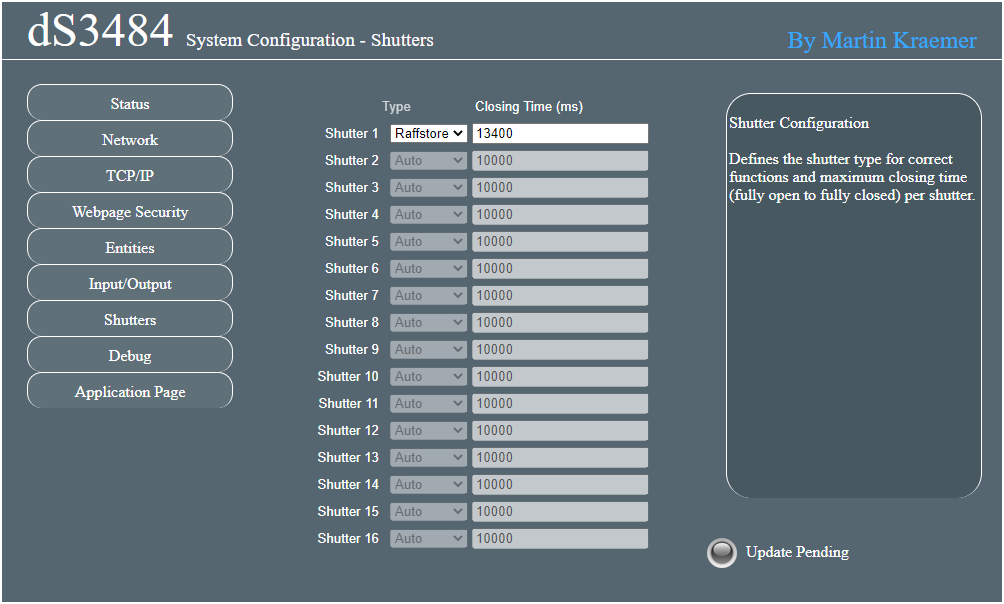
An IO can control a list of lights, shutters, or sockets. These are defined with their entity ID. A single ID controls only one entity. Having a comma separated list can control multiple entities at the same time.

If an IO is defined as “Motion” input, it always controls light entities. A Motion IO is usually connected to a motion or presence sensor. As long as the IO is set to high, the corresponding light entities are set to active, too.

The Auto I/O option provides you with a standardized suggestion for I/O usage based on the defined number of Lights & Shutters. If you activate this mode it is still required to define lights which should be affected by possibly connected Motion IOs.

“Multiclick Time” defines the time (in milliseconds) to provide for multi click/IO-input commands. Be careful with high numbers here - this will increase the time until a command is executed, while too low numbers will cause you not being able to perform high multi click commands. For details on these commands check chapter of “[Short list – multi click commands](#_Short_list_–)”.

## Shutters



This page provides settings for every shutter. Shutters which are not connected / defined within Entities tab are disabled. Per shutter you can specify a corresponding shutter type that you have connected. Furthermore, the closing time per shutter (the time a shutter needs from fully open to fully closed state) can be defined. Alternatively, to defining this value manually here, you can adjust it via the process of “[Calibrating the shutters](#_Calibrating_the_shutters)” within corresponding chapter in this document.

## Debug

The debug page shows all variables and their values currently used by the application.

This page is used by shell/bash scripts which are part of the RoomControl application to administrate dScript RoomControl boards remotely.

If you want to disable the possibility of remote administration while the device is not connected via USB, you should rename the debugapi.htm to \_debugapi.htm and adjust corresponding references within the bash administration scripts and global \_menubar.htm file which allows navigation within \_config pages.

## Application Page

The application page menu entry brings you back into the main application page. Within this you can press virtual inputs (which are not defined as motion IO) and see the status of relays & shutters.

# Hardware wiring

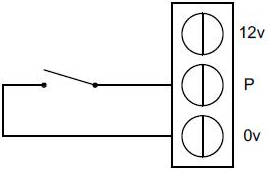
Previously to going through this chapter please read through “[The legal stuff](#_The_legal_stuff)” chapter. You are responsible for your safety and safety of others in any endeavor in which you engage. Do only continue with this chapter if you **know exactly what you are doing** and are at least a professional electrician. Electricity (independently of 5V, 12V, 120V or 230V) bring highly danger and can cause risk to your life and the life of others.

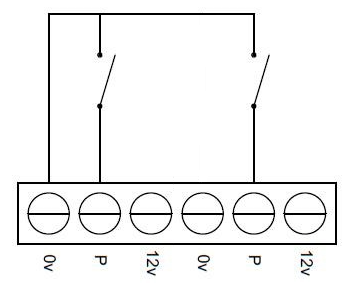
**Whenever performing any changes on your wiring, ensure you have disabled any power supply directly or indirectly connected with your circuit!**

## Connecting a Touch Sensor / Switch

Touch sensors are connected via I/O ports – **not via Relays.** Based on this you have to use 12V powered touch sensors / switches. dScript RoomControl enables digital ports as passive pull-up via source code.

Wire the touch sensor between a pin (P) and ground (0v). When the switch closes the input will become active (left picture below). Wiring multiple touch sensors, you can reuse the ground (0v) connector for multiple pins (see right picture below)





## Connecting a Motion / Presence Sensor

To-Be-Done

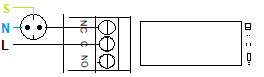
## Connecting a Light

Lights are connected to a single relay. If your light provides a protective conductor phase, please first ensure you have connected the protective conductor (S) to the light. Then Connect the neutral conductor (N) and the light blulb to “NO” (Normal Open) and the Line input (L) (110-240V) to “C” (Circuit).



## Connecting a Socket

Sockets are connected to a single relay. First please ensure you have connected the protective conductor (S) to the socket. Sockets are provided with power by default and can be turned-off if you enable the relay. To achieve this, connect the neutral conductor (N) to the socket and the socket to “NC” (Normal Closed) and the Line input (L) (110-240V) to “C” (Circuit).



## Connecting a Shutter

Shutters can be connected “in Row” or “in Parallel”. While “in Parallel” is the easier to understand solution, it is not as fail prove as “in Row” connection is. Due to software issue or invalid usage it might be possible with “in Parallel” connection that your shutter engine receives power on both engine inputs and based on this is damaged. Using “in Row” connection type it is not possible to provide power on both engine phases. Due to this **“in Row” is the to be preferred connection type.**

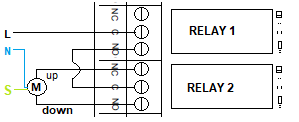
Please ensure you correctly configure the connection type for shutters according to your physical wiring as described within chapter “[Entities Page](#_Entities_Page)” of this document.

### Connecting in Row

Shutters are connected to two relays. Using the “in Row” connection the first relay defines if the shutter is moving or not. The second relay defines which direction the shutter moves. Based on this the states are as follows:

* Both relays are disabled = no movement
* Relay 1 enabled + Relay 2 disabled = Shutter moving up / opening
* Relay 1 enabled + Relay 2 enabled = Shutter moving down / closing
* Relay 1 disabled + Relay 2 enabled = no movement

If your shutter engine provides a protective conductor phase, please first ensure you have connected the protective conductor (S) to the shutter engine. Then connect the neutral conductor (N) to the shutter engine and the “down” phase of the shutter engine to “NO” (Normal Open) of relay 2. Connect the “up” phase of the shutter engine to “NC” (Normal Closed) of relay 2. Now you can connect the “NO” (Normal Open) output of relay 1 to “C” (Circuit) of relay 2. Complete the wiring by connecting the Line input (L) (110-240V) to “C” (Circuit) of relay 1.

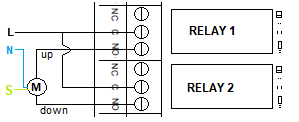


### Connecting in Parallel

Shutters are connected to two relays. Using the “in Parallel” connection the first relay defines movement up and second movement down. Based on this the states are as follows:

* Both relays are disabled = no movement
* Relay 1 enabled + Relay 2 disabled = Shutter moving up / opening
* Relay 1 enabled + Relay 2 enabled = Your shutter might get damaged!
* Relay 1 disabled + Relay 2 enabled = Shutter moving down / closing

If your shutter engine provides a protective conductor phase, please first ensure you have connected the protective conductor (S) to the shutter engine. Then connect the neutral conductor (N) to the shutter engine and the “down” phase of the shutter engine to “NO” (Normal Open) of relay 2. Connect the “up” phase of the shutter engine to “NO” (Normal Open) of relay 1. Complete the wiring by connecting the Line input (L) (110-240V) to “C” (Circuit) of both relays.



# Calibrating the shutters

Previously to calibrating your shutters ensure you have connected them correctly. Furthermore, you must ensure that the “Shutter connect” type is set correctly according to your physical cabling.

**Bring your board into calibration mode**

To start the calibration process, start up your dScript board / module as usually. Perform a five click multi click command on any shutter IO. You will see the blue LED on the device to turn on which indicates that the board is now in shutter calibration mode. Please note that in this mode just shutter IOs are active. It is not possible to send remote commands via dScriptServer interface neither to perform manual actions on any other entity than shutters.

**Change your shutter to “fully open” state**

Now press and hold the touch sensor of the IO controlling the shutter you would like to calibrate until the shutter is fully open. Then release the touch sensor. Ensure you do not keep the touch sensor pressed / the IO longer active than necessary to prevent damage to your shutter engine.

**Reset shutter closing time to zero**

After you have set your shutter into fully open position perform a three click multi click action on the touch sensor of the IO controlling the shutter you would like to calibrate. The board will reset the closing time for the shutter to zero and confirm the successful reset with the red LED blinking three times.

**Change your shutter to “fully closed” state**

Now press, release – then press and hold the touch sensor of the IO controlling the shutter you would like to calibrate until the shutter is fully closed. By reusing the “press and hold” or “press, release – then press and hold” combination, you can adjust the shutter until it fully fits your definition of the “shutter closed” state.

**Save the calculated closing time**

After you set the shutter into “fully closed” state using process described above perform a four click multi click action on the touch sensor of the IO controlling the shutter you would like to calibrate. This saves the currently defined closing time.

**Attention:**

Do not move the shutter further after you have saved the closing time until you get back into normal operation mode. Moving the shutter after you saved the new closing time will cause wrong reference values and due to this damage to the engine of your shutter or other parts of the shutter.

Tipp:

While this is not mandatory for the calibration process, you can see the current (unsaved) closing time of each shutter within the App\_Message box of the application page after you moved it into open / closed direction.

Tipp:

If you want to calibrate multiple shutters on your dScript board / module you do not need to exit the calibration mode every time. You can just start over with changing the next shutter to “fully open” state and then repeat the process as described above.

Tipp:

If you completely messed up the calibration of one shutter, simply exit the calibration mode without saving the calculated closing time. This will keep your previously defined closing time for the shutter.

**Bring the board back into normal operation mode**

After you have completed the calibration of all shutters as you want, you can exit back to normal operation mode by performing a five click multi click on any shutter IO. This will reenable control for lights and sockets such as normal operation for shutters.

# Short list – multi click commands

The multi click actions performed depend on the IO entity type you defined to be connected to the touch sensor. See details about possible options within the “[Input/Output](#_Input/Output)” chapter of this document.

**Light entity IO:**

|  |  |
| --- | --- |
| # of clicks | Action performed |
| 1 | Toggle lights On/Off connected to this IO.  Depending on its current state the light is toggled as the following: If it is set on turn it off – if it is set off turn in on. |
| 2 | Turn on all lights managed by this dScript board / module. |
| 3 | Turn off all lights managed by this dScript board / module. |

**Socket entity IO:**

|  |  |
| --- | --- |
| # of clicks | Action performed |
| 1 | Toggle sockets On/Off connected to this IO.  Depending on its current state the socket is toggled as the following: If it is set on turn it off – if it is set off turn in on. |
| 2 | Turn on all sockets managed by this dScript board / module. |
| 3 | Turn off all sockets managed by this dScript board / module. |

**Shutter entity IO (normal operation mode):**

|  |  |
| --- | --- |
| # of clicks | Action performed |
| 1 | Open the shutter connected to this IO if it is currently stopped.  Stop the shutter connected to this IO if it is currently moving. |
| 1 + hold | Open the (first) raffstore shutter connected to this IO into horizontal angle. As soon as the touch sensor is released the movement stops. |
| 2 | Close the shutter connected to this IO if it is currently stopped.  Stop the shutter connected to this IO if it is currently moving. |
| 2 + hold | Close the (first) raffstore shutter connected to this IO into horizontal angle. As soon as the touch sensor is released the movement stops. |
| 3 | Open all shutters managed by this dScript board / module. |
| 4 | Close all shutters managed by this dScript board / module. |
| 5 | Switch the board into shutter calibration mode. |

**Shutter entity IO (shutter calibration mode):**

|  |  |
| --- | --- |
| # of clicks | Action performed |
| 1 + hold | Open the (first) shutter connected to this IO. As soon as the touch sensor is released the movement stops. Opening time is subtracted from current closing time for calculating new shutter closing time. |
| 2 + hold | Close the (first) shutter connected to this IO. As soon as the touch sensor is released the movement stops. Closing time is added from current closing time for calculating new shutter closing time. |
| 3 | Reset the closing time of the (first) shutter connected to this IO to 0. |
| 4 | Save the currently calculated closing time of the (first) shutter connected to this IO. |
| 5 | Switch the board into normal operation mode. |

# Remote administration scripts

With the dScript RoomControl application you receive some shell scripts. These scripts allow to change settings remotely and/or in bulk.

The remote administration scripts are all based on the *debuapi.htm*. This application page is used to find the individual variable name to address mapping which is then used to execute dscript.cgi action on the board.

Please note that due to incompatibility of curl with browser “local storage” and its usage for the authentication of password it is not possible to use the remote administration scripts if you are using a password.

By default, the debugapi.htm is available independently of the board being connected via USB or not. If you want to higher your security, change the name of *debugapi.htm* to *\_debugapi.htm*. This will ensure the page is only accessible if the board is USB connected. Do not forget to change *debugapi.htm* to *\_debugapi.htm* within the shell scripts, too.

For the execution of the scripts a bash compatible operating system is required. You can either use any Linux based operating system or alternatively Linux on Windows.

Please check out the help (-h | --help) of each individual script for detailed options.

# TCP/IP command sets

There are two TCP/IP command sets on two selectable check boxes, of which one or none may be selected on the TCP/IP config tab. These are Binary and Binary with AES256 encryption (ASCII, Modbus and MQTT might come in future).

## Binary command set

0x30 Get Status

0x31 Set relay

0x32 Set output

0x33 Get Relays

0x34 Get Inputs

0x35 Get Analogue

0x36 Get Counters

0x40 Set Light

0x41 Set Shutter

**0x30 (byte 1 = decimal 48)**

**Get Status (1 byte command, returning 8 bytes)**

This command returns 8 bytes of status data

ModuleID This will be 34 (0x22) for the dS2824

SystemFirmwareMajor 2 for example

SystemFirmwareMinor 18 for example

ApplicationFirmwareMajor 1 for example Application

FirmwareMinor 2 for example

Volts (Power supply volts x 10.) Example 125 is 12.5v

InternalTemperature(highbyte) x10

InternalTemperature(lowbyte) combined to 16 bits, 267 = 26.7 °C

In the above example the returned bytes would be:

0x22 0x02 0x12 0x01 0x02 0x7D 0x01 0x0B

The bytes 7 (0x01) & 8 (0x0B) combined are 0x10B which is 267 decimal, meaning 26.7 °C

**0x31 0x02 0x01 0x00 0x00 0x00 0x00 (byte 1 = decimal 49)**

**Set Relay (7 byte command, returning 1 byte)**

This command turns a relay on or off or pulses it for a time period and returns an ACK/NACK byte. ACK=0, NACK=non-zero (actually the unknown relay number).

0x31 the actual command, the rest are parameters.

0x02 Relay number Valid numbers are 1-24(0x01-0x18)

0x01 Turn relay on (0x00 for off). This is ignored when following pulse time is >100.

0x00 }highbyte Pulse time (this parameter is not supported and always set to 0)

0x00 }midhigh These 4 bytes combined are a 32-bit pulse time for the relay

0x00 }mid low when less than 100 (as it is here 0x00000000) its ignored

0x00 }low byte When >100 this pulses relay on for that number of ms

To pulse relay 5 on for one second the command is:

0x31 0x05 0x00 0x00 0x00 0x03 0xE8

0x000003E8 (or just 0x3E8) is 1000 decimal, which is 1000mS or 1 second. The relay will turn on and then go off 1 second later.

When sending a relay pulse time, the relay on/off byte is ignored. The relay is always on for the duration of the pulse. Nevertheless the pulse time parameter is no longer support by this application and will be ignored if set to different than 0.

**0x32 0x04 0x01 (byte 1 = decimal 50)**

**Set Output (3 byte command, returning 1 byte)**

This command turns an output on or off and returns an ACK/NACK byte. ACK=0, NACK=non- zero (actually the unknown I/O port number).

All I/O's which need to be inputs should have the output turned off. When turned on the NPN transistor can sink up to 100mA.

0x32 The command, the rest are parameters

0x04 the I/O number,4 in this case.

0x01 on (0x01) or off(0x00)

**0x33 0x01 (byte 1 = decimal 51)**

**Get Relay (2 byte command – returning 5 bytes)**

This command is used to get the states of the relays. The second byte is the relay number, relay 1 in this case.

The first returned byte is the state of the requested relay, 0x00 (off) or 0x01 (on).

The next four bytes pack the states of all 32 relays (virtual and actual relays). Bit 7 of byte 2 is relay 32 through to bit 0 of byte 5 which is relay 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte 2 | | | | | | | |  | Byte 3 | | | | | | | |  | Byte 4 | | | | | | | |  | Byte 5 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

If the bit is set the relay is on, off otherwise.

**0x34 0x01 (byte 1 = decimal 52)**

**Get Input (2 byte command – returning 2 bytes)**

This command is used to get the states of the inputs. The second byte is the input number, input 1 in this case.

The first returned byte is the state of the requested input, 0x00 (inactive) or 0x01 (active) the second byte packs the states of all 8 inputs. Bit 7 is input 8 through to bit 0 which isinput

If the bit is high the input isactive.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Byte 2 | | | | | | | |
| Bit No. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

**0x35 (byte 1 = decimal 53)**

**Get Analogue inputs (1 byte command returning 16 bytes)**

This returns all eight possible analogue inputs. 16 bytes are returned, 2 for each analogue input. Inputs bytes that do not exist for corresponding module are filled with 0.

Byte1 byte2 for example:

0x02 0x3E combined to 0x023E, or 574 decimal for input1.

Byte3 byte4

0x01 0xFB combined to 0x01FB, or 507 decimal for input 2.

Bytes 5 – 14 follow in a similar way:

Byte 15 byte 16

0x03 0x2C combined to 0x032C, or 812 decimal for input 8. If the input is configured as a digital port, then the input will be 0 or 1.

**0x36 0x01 (byte 1 = decimal 54)**

**Get Counters (2 byte command – returning 8 bytes)**

This command is used only for backwards compatibility. It was used to get counters.

The second byte is the counter number, counter 1 in this case.

The first 4 bytes returned is the current counter value. This 32-bit (4 bytes) value is returned high byte first. The second group of 4 bytes returned is the capture register for this counter, also a 32-bit (4 byte) value returned high byte first.

Now the command will return always 0 since no counters in that way exist anymore.

**0x40 0x02 0x01 (byte 1 = decimal 64)**

**Set Light (3 byte command, returning 1 byte)**

This command turns a light on, off or toggles it and returns an ACK/NACK byte.

ACK=0, NACK=non- zero (actually the unknown light number).

0x40 the command, the rest are parameters

0x02 the light number - 2 in this case.

0x01 on (0x01) or off (0x00), any other value will toggle

**0x41 0x03 0x32 (byte 1 = decimal 65)**

**Set Shutter (3 byte command, returning 1 byte)**

This command opens/closes or sets a shutter to a specific state and returns an ACK/NACK byte.

ACK=0, NACK=non- zero (actually the unknown light number).

0x40 the command, the rest are parameters

0x03 the shutter number - 3 in this case.

0x32 closed/open state of the shutter (50% in this case) - fully open (0x64), fully closed

(0x00), any other value (between 0 and 100) setting directly the closing state.

Additionally, a value of 0xFF (255) means stop the shutter at current state.

**0x42 0x01 0x01 (byte 1 = decimal 66)**

**Set Socket (3 byte command, returning 1 byte)**

This command turns a light on, off or toggles it and returns an ACK/NACK byte.

ACK=0, NACK=non- zero (actually the unknown light number).

0x40 The command, the rest are parameters

0x01 the socket number - 1 in this case.

0x01 on (0x01) or off (0x00), any other value will toggle

**0x50 (byte 1 = decimal 80)**

**Get Config (1 byte command, returning 10 bytes)**

This command returns 8 bytes of configuration data

Count of pysical relays 8 for example

Count of lights 2 for example

Count of shutters 1 for example

Count of power sockets 5 for example

MAC Address - Byte1 D8 for example (MAC: D8:80:39:BC:E6:4A)

MAC Address - Byte2 80 for example

MAC Address - Byte3 39 for example

MAC Address - Byte4 BC for example

MAC Address - Byte5 E6 for example

MAC Address - Byte6 4A for example

In the above example the returned bytes would be:

0x08 0x02 0x01 0x05 0xd8 0x80 0x39 0xbc 0xe6 0x4a

**0x51 0x03 (byte 1 = decimal 81)**

**Get Light (2 byte command – returning 1 byte)**

This command is used to get the state of a light. The second byte is the light id, Light 3 in this case.

The returning byte can be either 0 (= off) or 1 (=on).

**0x52 0x01 (byte 1 = decimal 82)**

**Get Shutter (2 byte command – returning 2 bytes)**

This command is used to get the states of a shutter. The second byte is the shutter id, Shutter 1 in this case.

The first returned byte is the state of the requested shutter in percentage, values from 0x00 (0=Fully closed) to 0x64 (100=Fully open) are possible. The second byte returns the current movement of the shutter 0x00 (0=stopped), 0x01 (1=opening) and 0x02 (2=closing).

**0x53 0x07 (byte 1 = decimal 83)**

**Get Socket (2 byte command – returning 1 byte)**

This command is used to get the state of a socket. The second byte is the socket id, socket 7 in this case.

The returning byte can be either 0 (= off) or 1 (=on).   
(We assume that sockets are connected via NC instead of NO to relays. Based on this Sockets are inverted with their state. That means if the relay is on, the socket is off. If the relay is off, the socket is on)

## AES binary command set

The AES Binary commands are the same as the Binary commands described above. The only difference is that they are AES encrypted and always 16 bytes in length. The first bytes are the same as described in the Binary command set. The last 4 bytes is the Nonce (a random number) and the bytes in the middle are undefined. The module will decrypt the command, generate the response and encrypt it before returning it to you.

Your program that controls the module will need to encrypt the commands and then decrypt the response. We use AES256 CBC encryption, hence the requirement for a 256-bit (or 32 byte) key. To complete the security, we use a random IV generated by a cryptographically secure random number generator (ISAAC).

To control the module, you will need to send the commands with AES encryption. To help you with this we have examples in C#, Java and Python. The C# and Java applications are complete and may be used or modified as you wish.

To prevent re-play (or Playback) attacks the command packet includes a Nonce. This takes the form of a 32-bit (4 byte) random number in positions 12, 13, 14 & 15 of the 16 byte data packet. For example, when you send a Get Status command (0x30) you will get a 16 byte block returned. The first 8 bytes (0-7) will be as defined for the binary commands. Bytes 8-11 are unused. Bytes 12-15 contain the Nounce.

Commands with generate and send you a Nounce are:

0x30 – Get Status

0x31 – Set Relay

0x32 – Set Output

Commands which require a Nounce to be sent by you are:

0x31 – Set Relay

0x32 – Set Output

A Nounce is only ever used once, you must always use the most recently issued Nounce.

The following example shows how the Nounce provided by the module is used in the next Set Relay or Set Ouput command.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Get Status | 0x30 | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx |
| Response | 0x22 | 0x02 | 0x12 | 0x01 | 0x02 | 0x7D | 0x01 | 0x0B | xx | xx | xx | xx | 0x89 | 0xAB | 0xCD | 0xEF |
| Set Relay | 0x31 | 0x02 | 0x01 | 0x00 | 0x00 | 0x00 | 0x00 | xx | xx | xx | xx | xx | 0x89 | 0xAB | 0xCD | 0xEF |
| Response | 0x00 | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | 0x1A | 0x2B | 0x3C | 0x4D |
| Set Relay | 0x31 | 0x03 | 0x01 | 0x00 | 0x00 | 0x00 | 0x00 | xx | xx | xx | xx | xx | 0x1A | 0x2B | 0x3C | 0x4D |
| Response | 0x00 | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | 0xF1 | 0xE2 | 0xD3 | 0xC4 |
| Set Output | 0x32 | 0x04 | 0x01 | xx | xx | xx | xx | xx | xx | xx | xx | xx | 0xF1 | 0xE2 | 0xD3 | 0xC4 |
| Response | 0x00 | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | xx | 0x5C | 0x47 | 0x9B | 0xED |

If the Nounce you send with the command does not match the last one sent to you, then the Relay (or Output) will not be changed.

No other commands either require or provide a Nounce.

# The dScriptServer

The dScriptServer is part of the dScriptModule python module which is available from the same author on github: <https://github.com/mk-maddin/dScriptModule-PyPi>

The dScriptServer allows to communicate via binary command sets with the boards.  
Every board sends a notification to the defined dScriptServer whenever an action happens (lights on/off; socket on/off; shutter open/close/stop; motion sensor active/inactive etc.). The dScriptServer then throws a corresponding event which allows the further processing of such.

For details please check the documentation of the dScriptModule.

# Home Assistant integration

Using the python module described within the chapter of “The dScriptServer” it is possible to establish a connection with home assistant. A corresponding custom integration from the same author is available on github: <https://github.com/mk-maddin/dScriptModule-HA>

Please check the documentation of the custom integration to get details about using the home assistant integration.

# Custom programming

Feel free to change the dScript RoomControl application to your own needs.

Please check the official dScript documentation about details on writing your own firmware.

<https://www.robot-electronics.co.uk/dscript.html>

After you have done some custom programming please rerun the “create-debugapi.sh” shell helper script to recreate debugapi.htm and rewrite update variables there.

# The legal stuff

## Disclaimer

You must accept that You and You Alone are responsible for your safety and safety of others in any endeavor in which you engage. While the material provided within this documentation, the dScript RoomControl application itself and any other related object or service provided in hopes that you build your own system, you are responsible for verifying its accuracy and applicability to your project. You are Responsible for knowing your limitations of knowledge and experience. If you do any work with “main power” such as 120 or 240 VAC power wiring, you should consult a licensed electrician.

Some illustrative photos do not depict safety precautions or equipment, in order to show the project steps more clearly. This project is not intended for use by children. Use of this documentation, the dScript RoomControl application itself and any other related object or service provided is at your own risk. I disclaim all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with applicable laws.

You expressly agree to hold the author of this application and/or document, such as Devantech. Ltd. harmless for any property damage, personal injury and/or death, or any other loss or damage that may result from the use of this documentation, the dScript RoomControl application itself and any other related object or service provided.

## General Public License

This file is part of dScript RoomControl.

dScript RoomControl is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

dScript RoomControl is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with dScript RoomControl. If not, see <https://www.gnu.org/licenses/>.