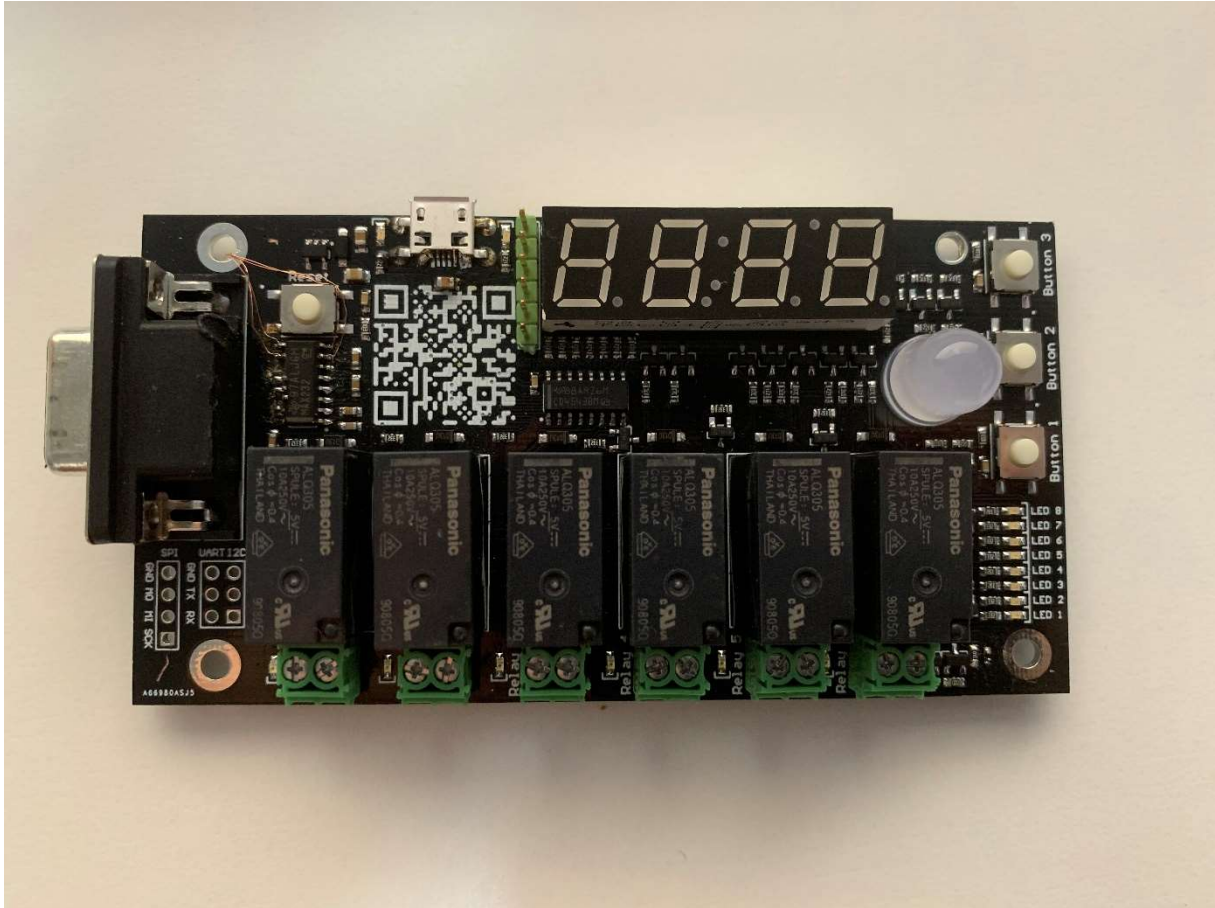# COMMAND REFERENCE v1.0

Extension board for the Edge Device

In this manual for the extension board v1.0 are supported command that can be sent over RS-232 (Serial) to control the parts on the PCB. Since the DSUB connector doesn't supply the PCB with current, the Micro-USB connector on the top left has to be used. It is only for supply and can't be used for communication.

Serial settings:

Every sent command must end with a ";" to be completed, else it is still reading until he receives one.
Every command received by the extension board comes with a ";" too.
Look at for more detailed information.

Baud-Rate: 115'200

Inhalt

RGB

You can set or read the current color of the RGB led. There are only certain colors available.

Colorcode

0
1
2
3
4
5
6
7

SET

Set the RGB to the color with the color-code

```
set.rgb.[color-code]
```

Example:

```
set.rgb.5;
```

GET

Returns the current color of the RGB

```
get.rgb
```

Example:

Received:	rgb.6;
Converted:	Yellow

Relays

There are six relays that are able to switch 230V AC **(but not recommended).**
If the relay is set a red led will shine on the left side of it.
You can connect it through the green terminal below it.
They are numbered from left to right and 1 to 6.

SET
Turn the selected relay on or off

State:

0 -> off
1 -> on

        set.relay.[relay number].[state]

Example:
        set.relay.1.1;

GET
Get the state of the relay

        get.relay.[relay number]
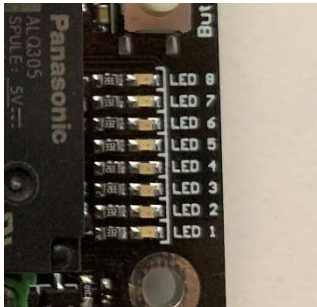
Example:
        Received: relay.3.1;
        Converted: ON

LEDs

There are 8 green LEDs that can be turned on or off. They are numbered from 1 to 8.

SET

Turn the selected LED on or off

State:

0 -> off
1 -> on

    set.led.[led number].[state]

Example:

    set.led.4.1;

GET

Returns the state of the relay

    get.led.[led number]

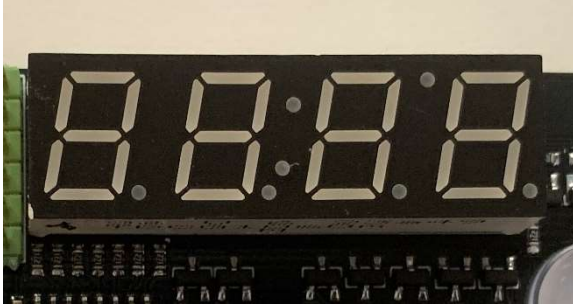Example:

Received:	led.3.1;
Converted:	ON

7 Segment

The segment display can individually show numbers on each position, you can choose where the decimal point is, turn the colon or the upper point on. The display shines in green and can only show numbers.

SET

The position of the single digit:

Pos. 1 -> X000

Pos. 2 -> 0X00

Pos. 3 -> 00X0

Pos. 4 -> 000X

dp pos -> decimal point position. Can be from 1 to 4.

colon -> colon state, 1 = on, 0 = off.

upper -> upper point, 1 = on, 0 = off.

set.segment.[pos. 1][pos. 2][pos. 3][pos. 4].[dp pos].[colon].[upper]
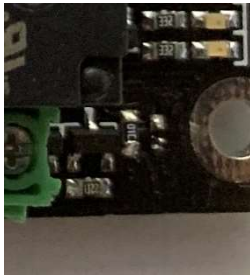
Example:

set.segment.1234.0.1.0

Display shows: 12:34

Temperature

Used to measure the room temperature.

Problem:

The longer the PCBs is turned on the more the sensor heat itself up. It will rise up a few Celsius.

GET

Returns the temperature as one number

```
get.temp
```

Example:

Received:	2918;
Converted:	29.18°C

SHOW

The temperature can be show on the display with this command:

```
set.showTemp.[state]
```

If you turn it off (state=0) the display shows 0000.

Buttons

Used as an input source for the edge device

Sometimes it could trigger a fault state if the input can't be read through the edge device. Only way to fix is, to reset (or repower) the extension board.

GET

There is no command required. When a button is pressed one of the following returns can be received when pressed:

BUTTON1;

BUTTON2;

BUTTON3;

Serial Settings

⤧ Serial Port	/dev/ttyUSB0				🔍

🔧 Settings

Baud Rate	Data Bits	Parity	Stop Bits
▾ 115200	8 ⌄	None ⌄	1 ⌄

⤏ Input

Optionally wait for a start character of [ ] , then

Split input [on the character ⌄] [;]

and deliver [ascii strings ⌄]

�Output

Add character to output messages [;]

⇄ Request

Default response timeout [10000] ms

Functions examples

All function codes can be found on the github.

Read buttons

If the buttons are pressed you get into one of the following if-statements.
There you can set msg.payload that will be sent on the output of the function node.

```
1  if(msg.payload === "BUTTON1;")
2 ▾ {
3      msg.payload = "set.relay.1.1";
4      return msg;
5 ▴ }
6  else if(msg.payload === "BUTTON2;")
7 ▾ {
8      msg.payload = "set.relay.1.0";
9      return msg;
10 ▴ }
11  else if(msg.payload === "BUTTON3;")
12 ▾ {
13      msg.payload = "set.rgb.5";
14      return msg;
15 ▴ }
16  return msg;
```

Control the rgb led with the color picker node

Since they are only a few colors which the rgb led can show, it will automatically set the closest color.

Color picker node settings:

| Format | hsl ⌄ | square ⌄ |

Show hue slider : ☑
Show lightness slider : ☐
Show transparency slider : ☐

If width is 4 or greater:
Always show swatch : ☑
Always show picker : ☑
Always show value field : ☑

➔ If msg arrives on input, pass through to output: ☑

| ✉ Send | one value when released/closed ⌄ |

| Payload | current value as an object ⌄ |

| Topic | optional topic |

| 🏷 Name | |

Function node

```
1  if(msg.payload.l == 1)
2 ▾ {
3      msg.payload = "set.rgb.7";
4      return msg;
5 ▴ }
6  else if(msg.payload.l === 0)
7 ▾ {
8      msg.payload = "set.rgb.0";
9      return msg;
10 ▴ }
11 else if(msg.payload.h < 30)
12 ▾ {
13      msg.payload = "set.rgb.4";
14      return msg;
15 ▴ }
16 else if(msg.payload.h < 90)
17 ▾ {
18      msg.payload = "set.rgb.6";
19      return msg;
20 ▴ }
21 else if(msg.payload.h < 150)
22 ▾ {
23      msg.payload = "set.rgb.2";
24      return msg;
25 ▴ }
26 else if(msg.payload.h < 210)
27 ▾ {
28      msg.payload = "set.rgb.3";
29      return msg;
30 ▴ }
31 else if(msg.payload.h < 270)
32 ▾ {
33      msg.payload = "set.rgb.1";
34      return msg;
35 ▴ }
36 else if(msg.payload.h < 330)
37 ▾ {
38      msg.payload = "set.rgb.5";
39      return msg;
40 ▴ }
```

Set it up like this:

Color picker -> function -> serial out

Use variables

Variables are here to store data if you need to, but there are a few types of them:

Temporary variable ->	Normal variable
Locally in function node ->	Context variable
Reachable from flow ->	Flow variable
Reachable from everywhere ->	Global variable

Normal variable

It is initialized with "var" and followed by the name you want to give the variable.

If you wish to, you can declare it already at the beginning with "=" and a value, like a number a text in "", or another variable writing its name there.

```
1   var VarA = 0;
2   var VarB = "Hallo!";
3   var VarC = VarA;
```

Here the variable names are VarA, VarB and VarC.

These types of variables only exists in the same function node and in every run of this node the variables will be forgotten (lose their saved values);

Context variables

Here you can save your variables locally in you function node without losing them every cycle.

Create/Set Flow variables

To create the variable, you must use the command:

```
context.set('variablename',0);
```

Read Global variables

To read one of the existing variables you simply just add one of the following commands:

```
if(context.get('variablename'))
{
    ...
}
```

```
var x = context.get('variablename');
```

Flow variables.

This type of variables are based on the normal ones, but are reachable in the same flow.
That means: multiple function nodes can read/change the variable.

Create/Set Flow variables

To create the variable, you must use the command:

```
flow.set('variablename',0);
```

Where "0" is the value of your variable (it can be another variable name).

Read Flow variables

To read one of the existing variables you simply just add one of the following commands:

```
4  if(flow.get('variablename'))
5 ▾ {
6       ...
7 ▴ }
8
9  var x = flow.get('variablename');
```

You can either check the variable if it's true or has a certain value.
Or you can set another new/existing variable with the value of the flow variable.

Global variables

Follows the same rules as Flow variables, but they are from every flow and function node reachable.

Create/Set Global Variables

To create the variable, you must use the command:

```
global.set('variablename',0);
```

Read Global variables

To read one of the existing variables you simply just add one of the following commands:

```
if(global.get('variablename'))
{
    ...
}

var x = global.get('variablename');
```