MQTT for digitalSTROM

this is me, 6 years later: i am still using Chriss Gross' gateway and it has been working great for me. Turns out the little delay that i get sometimes when a few lights have to be turned on via mqtt while the rest is directly via digitalstrom doesn't really matter all that much. So there is really no need to write a more responsive gateway as far as I am concerned. There are new gateways out there, which basically use digitalstrom only a as bus and allow other software such as home assistant and similar ones to take complete control over scenes etc, but there have been numerous times in the last 10 years where i was quite happy, that even though my network was messed up or my server had crashed, my digitalstrom lights still could be turned on and only the addons where disfunct, so i am happy to keep it the way it is also in the future. i am now migrating the gateway to a new server and with that, it will be running inside a docker container, since chriss hasn't maintained it since 9 years or so. So you can probably skip the rest of this article and jump straight over to digitalstrom_mqtt_gateway_in_a_docker_container if you're looking for a quick and dirty solution that seems to work remarkably well :)

old article

I use digitalSTROM at home and at some customers as the main bus system for all things related to light. Here are the main points, why i chose to go with digtialSTROM as opposed to other solutions:

- DS is relatively open. the digitalSTROM Server (DSS) software is open source and comes with various openly documented API's to extend your setup. One is the DSS API which is good for polling status information about the bus system and the other is the VDC-API which allows to create virtual digtialSTROM devices to control stuff outside of your DS network
- the bus itself does not require any additional cables. DS is perfect to install in an already existing house, where you had conventional electrical cabling so far. no extra bus lines are needed as the whole bus works across a relatively slow but very stable powerline network.
- thanks to the DSS's web-interface, the whole bus system can be programmed very easily and it takes almost no technical knowledge to do so. this means, you are independant of electricians once the system is installed and you can easily re-program scenes, move devices to other rooms etc.
- despite the fact, that DS has these api's, uses a built-in web-server and connects to all kinds of devices via TCP/IP it is still very robust. That's due to the fact, that the DSS is not actually required for the pure digitalSTROM components to work together. This means, that even if your DSS crashes (which in my case almost never happened, and if it did it was usually me screwing around in the lower levels of the DSS, where you aren't supposed to change stuff ;)) or if your network fails, your wifi acts up or anything computer related happens, your light switches will still turn your light on and off! you lose all the extra features but at least you and more importantly your wife are not sitting in the dark while you debug what happened ;)
- while DS isn't as cheap as some wifi-based solutions like sonoff and firends, it is relatively cheap compared to other professional systems like KNX, even though it provides a very similar stability and can be nicely integrated.
- DS is owned by Hager, a large manufacturer of electrical hardware in europe. So it can be expected that DS will be around for some time as it already has made it into the market.

current MQTT solution and its downside

one downside of DS is the lack of MQTT support. Fortunately, Chriss Gross wrote an MQTT bridge which uses the DSS API to publish the status of your DS system on MQTT and it also subscribes to a set of topics so you can interact with your DS.

So far so good, but the problem with this MQTT bridge is, that it needs to poll your DSS every few seconds. If you set the interval too low (depends on the size of your installation and hardware-revision of your DSS, in my case i can't go below 1s) it slows down your DSS and has a negative impact on the user experience of the DSS apps and other control methods that use the API or the Web-Interface as well. Your light switches will still work normally though.

what I want to add

While the existing MQTT bridge is fine for changing room states, calling scenes etc, we need something that can enable us to react faster to events that happen on the DS bus. My Idea is, to create an MQTT bridge that connects via the VDC-API to emulate virtual digitalSTROM devices. I could then place a virtual dimmer into a room and set a value of 100% for Scene 1, 75% for Scene 2, 50% for Scene 3, 25% for Scene 4 and of course 0% for off. Now i can publish the value canges for this dimmer via MQTT. The nice thing about this solution is, that the DSS will PUSH status changes to the API and hence to MQTT and we don't need to PULL it as we do with the DSS-API variant.

luckily there is a nice socket-based API for virtual devices available from plan44 a great contributor to the DSS environment. My plan is ot use node.js to implement an MQTT bridge using this api via a vdcd.

The downsides of this approach are:

- we need to put a virtual dimmer in every room and program its values. luckily that only needs to happen once
- the setup is slightly more difficult as we need to also setup the vdcd and possibly vdsd daemons.
- at least with the socket-based API i wasn't able so far to call scene 1 distinctively via the api. the problem here is, that the api only allows for the virtual device to send button pushes or sensor values, but it does not allow to explicitly call a scene. now we can call scene 2-4 with a relative hight certainty by sending 2-4 successive button-pushes. but the problem with scene 1 is, that a single push on the button will either call scene 1 or it will turn off the lights, depending on the current status of the room. I have yet to find a solution to that problem.
- so far the whole solution isn't very stable. when i re-scan the devices on my digitalSTROM bus via the web-interface of the DSS, it will loose the connection to my virtual device. I need to re-start the node.js script to re-connect to the vdcd which provides the API, but as soon as i disconnect the API-Client (my node.js script) the vdcd segfaults. this can of course be solved by simply running it via something like systemd, which whill then simply restart it, but the main issue here is, that my node.js script is not informed, that the connection has been lost by the DSS or that it wants to re-scan all devices. So unless i notice the malfunction personally and fix it manually, my node.js script will not stop and re-initiate the connection.

The upside, that makes it all worth doing:

- NO LAG! which leads to a high WAF (Wife Approval Factor) of the installation :)

Current Stats of this project

I have written a more or less working prototype. a virtual device needs to be speicifed according to the api documentation in json format and my script then takes this definition, creates the device and publishes its status to MQTT. it also subscribes to some MQTT topics in order to send button pushes to the DSS. This allows to also use this connector to integrate some physical hardware like a sonoff swithch without any noticeable lag into your DS setup.

I am very pleased with the performance of this connector. I tested a sonoff integration with this script and compared it with a) the MQTT bridge that uses the DSS api and b) a scene responder running on the DSS scene responder app that will turn my sonoff on and off via http reuqests.

- the Scene-Responder was the slowest. it took several seconds to react to calling a scene. my sonoff attached light bulb came on between 3-5 seconds after the light was turned on
- on second place was the MQTT API, polling my DSS in about a 1second interval. Sometimes though the DSS failed to react quick enought, so that the interval seems sporadically increased a little to maybe 2 seconds from time to thme. Needless to say, this caused the light bulb to switch on with somewhere between almost no delay and up to 2 seconds delay
- the winner was the MQTT connector using the VDC-API. with this connector the light bulb actually was on before the digitalSTROM controlled lights in my room where on themselves! this is due to the fact that the DS controlled lights are switched using a dimmer component and they are 220V LED lights. The dimmer needs to turn on slowly to trigger the LED's, while my Sonoff is a simple relay with a regular light bulb, so no delay here.
- when calling a scene via MQTT, so when using the opposite direction, there was no noticeable difference between using my MQTT implementation and Chriss' DSS-API bridge. So i guess i won't invest too much time in trying to find a way to call scene1 via my API as i can simply do that over the DSS-API based connector.

This was a nice test to see how fast the VDC api and also the whole MQTT side was. The on-command for my sonoff had to first be issued by the DSS and its virtual device connector. it was then pushed over a tcp socket to the VDSD, from there through another tcp socket to the VDCD, form there through tcp to my node.js script. from there to an MQTT broaker then to node-red and finally over a wifi-connection to the sonoff device. All in all three pyhsical devices where involved: my DSS (oldest and slowest revision with up-to-date firmware), my server (with 8-core Intel E5-2630L v3 cpu), and finally the sonoff.

the script

here is the source-code of my current prototype implementation. I am still trying to understand exactly how the API works and how I can possibly react to re-scans in the future. also i would like to implement the blink-feature, so that my sonoffs can be identified by blinking the attached light, like this is the case for the native DS devices. so still some work to be done. also i would love to be able to call scenes directly via the

dss-vd.js

```
//you may use this code under GPL v3 at your own risk :)
var net = require('net');
const mqtt = require('mqtt')
var sleep = require('sleep');

//base_topic can contain more than one level of mqtt topic string, but
it must not have a slash at the begining or the end.

var JSONconfig=`
{
    "vdcd_host" : "127.0.0.1",
    "vdcd_port" : 8999,
    "base_topic" : "myvdc",
    "initmsg" :
    [
        {
            "message":"init",
            "tag":"sonoff",
            "protocol":"json",
            "group":1,
            "uniqueid":"mctest1",
            "name":"mctest1",
            "output":"light",
            "buttons":
            [
                {
                    "id":"button1",
                    "buttontype":0,
                    "group":1,
                    "element":1
                }
            ]
        }
    ]
}`



config = JSON.parse(JSONconfig);
console.log(config);
var base_topic=config.base_topic;
var vdcd_host=config.vdcd_host;
var vdcd_port=config.vdcd_port;
var tags = [];
if ( Array.isArray(config.initmsg)){
    for ( i in config.initmsg ){
        tags.push(config.initmsg[i].tag);
    }
} else {
    tags.push(config.initmsg.tag);
}
```

```
console.log(tags);

var client = new net.Socket();
client.connect(vdcd_port,vdcd_host,function() {
    console.log("connected to: "+vdcd_host+":"+vdcd_port);
    client.write(JSON.stringify(config.initmsg));
});

client.on('data', function(data) {
    console.log("raw data: "+data);
    //split lines:
    lines=data.toString().split("\n");
    for ( var i in lines ){
        console.log ("parsing message "+i);
        line=lines[i];
        if(line.trim() != ""){
            d=JSON.parse(line);
            switch(d.message){
                case "status":
                    console.log("STATUS: "+line);
                    break;
                case "channel":
                    console.log("CHANNEL: "+line);
                    console.log("value: "+d.value);
mclient.publish(base_topic+"/status/"+d.tag+"/value",d.value.toString()
);
                    break
                default :
                    console.log('DATA: ' + line);
            }
        }
    }
});

client.on('close', function() {
    console.log('Connection closed');
    process.exit();
});

// MQTT:

const mclient = mqtt.connect('mqtt://mqtt.psuter.ch')

mclient.on('connect', () => {
    for ( i in tags ) {
        mclient.subscribe(base_topic+'/set/'+tags[i]+'/#');
    }
});

mclient.on('message',(topic,message) => {
```

```
    //
<base_topic>/set/<tag>/<type:button|input|sensor>/<buttonId>/<action>
    regex="^"+base_topic+"/set/([^/]+)/([^/]+)/([^/]+)/([^/]+)$";
    console.log("regex: "+regex);
    ptopic=topic.match(regex);
    if ( ptopic != null ){
        tag=ptopic[1];
        type=ptopic[2];
        id=ptopic[3];
        action=ptopic[4];
        console.log("parsed topic: " + ptopic);
        switch(action){
            case "click": //will send one or more button clicks (push
and release)
                console.log(type+" click status changed to "+message);
                for ( i=1;i <= message ; i++){
                    console.log("sending "+type+" click " +i+ " to
dss");
                    JSONaction=`
{
"message":"${type}",
"id":"${id}",
"value":100,
"tag":"${tag}",
}`
                    client.write(JSONaction);
                    console.log("WRITING: "+JSONaction);
                    if( message > 1 ) {
                        console.log("sleep");
                        sleep.msleep(150);
                    }
                }
                break;
            case "on":
                //message=1 pushes the button / switches the input to
on
                //message=0 releases the button /switches the input to
off
                //message=n pushes the button for n milliseconds /
switches input on for n milliseconds
                console.log(type+" push set to "+message);
                JSONaction=`
{
"message":"${type}",
"id":"${id}",
"value":${message},
"tag":"${tag}",
}`
                client.write(JSONaction);
                break;
        }
```

```
    }
});
```

From:
http://wiki.psuter.ch/ - **pswiki**

Permanent link:
http://wiki.psuter.ch/doku.php?id=mqtt_for_digitalstrom

Last update: **06.07.2024 09:51**