State Monitoring Protocol

Til Blechschmidt

til@blechschmidt.de

February 27, 2017

Abstract

Since there is no universal, open and secure implementation of a protocol to control any arbitrary device from another this document aims to provide an outline of the SM-Protocol that aims to provide such a protocol.

Contents

List of source codes

Part I
Structure

1 Network layer

1.1 Host Extensions for IP Multicasting

At the heart of this protocol is RFC 1112 or the Host Extensions for IP Multicasting. It enables communication between multiple devices without any additional overhead by utilizing features that are baked into the TCP/IP stack instead of every device keeping track of every connected devices. This is of great importance since most devices utilizing this protocol do not have the processing power to handle multiple TCP connections.

1.2 JavaScript Object Notation

On top of the network layer, this protocol utilizes the JavaScript Object Notation since it offers high flexibility and readability. This enables easy debugging due to it's great human-friendly structure as well as easy extendability so that the protocol does not restrict devices. An example of such a message looks like this:

```
{
        "action": "write",
        "channel": [0, 0, 1],
        "payload": 255
}
```

Listing 1.1: Simplest write request

Note though that the action property can take any arbitrary value but these ones are reserved and should be used when possible to ensure interoperability between devices.

read	Requesting the current value from a device
write	Sending a value to a device to write
state	Notify other devices about a change in state

For more details have a look at Part II of the document.

2 Devices

Due to the previously in section 1.1 mentioned network layer all devices that make use of this protocol are required to have support for the TCP/IP stack and the multicast specifications. In addition to that the devices need to support basic JSON parsing and composition. If the device is only using actions that require distribution or reception of JSON data the opposing requirement can be omitted.

2.1 List of devices known to work

- ESP8266
- Raspberry Pi
- Raspberry Pi Zero
- Generic x86 device

Part II

Protocol

3 Concept

As visible in Listing 1.1 it is preferred to make use of the basic structure consisting of the action, channel and payload fields in order to maximize interoperability between devices. However this is not enforced and if it is required that devices make use of the network layer using their own data structure for more complex tasks they may do so. To give an example of such a situation take a look at Section 5.

3.1 Types

Since JSON is not enforcing types any key can take a value of an arbitrary type. Because of that the previously mentioned keys can have any value. Action and channel are preferred to have the types of a String and an array of three numbers respectively to reduce issues with different implementations. The type of the payload is not fixed though so that it can match the device type easily. A dimmable lamp for example might take an integer as well as a boolean (3.1) whereas a gas boiler might take a more complex object (3.2).

```
{
        "action": "write",
        "channel": [2, 5, 1],
        "payload": 249 OR true
}
```

Listing 3.1: Request to dimmable lamp

```
{
        "action": "write",
        "channel": [1, 1, 19],
        "payload": {
                "waterTemp": 59,
                "ecoMode": false
        }
}
```

Listing 3.2: Request to a gas boiler

4 Differencing between human caused actions and automated ones

Since there are many different types of interactions between devices in the network it is of great importance to keep track which ones originate from human activity and which don't. This only applies to write requests though since state changes are always an automated result, caused by a write request that may or may not have a human individual as its source. In all previous examples you didn't see such a notation. That's because it defaults to a non-automated write request that was caused by a human. So if you omit the "automation" field in your write request it is blindly assumed, that there was a human that caused this by pressing a light switch as an example. A practical example of such a notation, flagging your request as automated, can be found at Listing 4.1.

```
{
        "action": "write",
        "channel": [1, 0, 1],
        "automated": true,
        "payload": 89
}
```

Listing 4.1: Automated request

5 Encryption

Hey there! What's up! Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.