Node-RED

Node-RED ist ein Programmierwerkzeug, mit dem sich Hardwaregeräte, APIs und Online-Dienste auf neue und interessante Weise miteinander verbinden lassen.

Es bietet einen browserbasierten Editor, der es einfach macht, Abläufe unter Verwendung der breiten Palette von Knoten zu verdrahten, die mit einem einzigen Mausklick zur Laufzeit bereitgestellt werden können. (Quelle: https://nodered.org/, übersetzt mit DeepL.)

Und die Arbeit mit diesem grafischen Entwicklungswerkzeug macht wirklich Spaß!

Was erwartet Sie in diesem Beitrag?

Sie werden...

- ... die wichtigsten Eigenschaften von Node-RED kennen lernen
- ... Node-RED-Steuerungen erstellen können
- ... die notwendigen Softwarekomponenten kennen lernen und sie installieren können
- ... Bausteine aus der Node-RED-Modulsammlung (sie ist Teil von npm) verwenden können
- ... das Zusammenwirken Node-RED, MQTT, Mosquitto , ZigBee, Zigbee2MQTT, Tasmota und ioBroker verstehen

Google liefert zum Suchbegriff *Node-RED* etwa 492 Millionen Einträge: Überblicksartikel, detaillierte Anleitungen, Tipps und Tricks, Source-Code, Grafiken und vieles andere mehr. Was will dann *dieser* Beitrag? *Neugierig machen, zum Probieren einladen und ein paar Tipps geben, die sonst nur schwer zu finden sind.*

Aus Platzgründen sind einige Abbildungen hier verkleinert dargestellt. Sowohl dieser Text, die Kurzfassung, wie auch die wesentlich ausführlichere Langfassung können über den Link am Ende des Beitrags abgerufen werden, auch mit den Bildern in Originalgröße. Ein ☝ im Text darauf hin, dass in der Langversion an dieser Stelle mehr Text, ein Programmcode oder eine detaillierte Anleitung zu finden ist.

Wofür wird Node-RED verwendet?

Übliche Programmiersprachen arbeiten den Programmcode Zeile für Zeile (*linear*) ab, enthalten dabei Verzweigungen, Schleifen und Funktionen (Unterprogramme). Manche Sprachen können Interrupts behandeln. Andere wieder erlauben asynchrone Abläufe: dabei wartet ein Programm vorerst nicht auf das Ergebnis einer langsameren Abfrage (etwa auf eine Datenbank- oder Internetabfrage) und macht weiter, bis das Ergebnis benötigt wird.

In einem automatisierten Haus, einem Smart Home, treten (meist zu nicht vorhersehbaren Zeiten) *Ereignisse* (events) auf, auf die das Steuerungsprogramm mit angemessenen Aktionen reagieren soll.

> **Note**
>
> Beispiel: Jemand betritt einen Raum: der Türkontakt löst ein Ereignis aus. Das Node-RED-Programm sendet ein Signal aus, mit dem das Licht eingeschaltet wird. Zusatzaufgabe: aber nicht tagsüber! Na gut, Bewegungsmelder sind schon erfunden, aber wir fangen eben mit einfachen Aufgaben an.

Node-RED verwendet Ereignisse intern auch für Vorgänge der Programmverwaltung. Wenn etwa ein Programmzweig hinzugefügt oder entfernt wird, wird ein *Ereignis* ausgelöst.

Node-RED läuft unter JavaScript: ein Grund, sich kurz mit JavaScript zu beschäftigen. Die erstellten Steuerungen werden in die *JavaScript Object Notation* (JSON) übertragen und können damit gespeichert oder weiter gegeben werden, so auch die Beispiele aus diesem Beitrag.

JavaScript und ECMAScript

JavaScript wurde 1995 von Netscape entwickelt, um Webseiten dynamisch zu machen. 1997 wurde daraus die standardisierte Sprache ECMAScript. Nach dem hauptsächlichen Einsatz in Webbrowsern wurde JavaScript zu einer vollwertigen Programmiersprache weiter entwickelt und kann daher auch für eigenständige Programme verwendet werden – Node-RED ist ein Beispiel dafür.

> **Note**
>
> Die Programmiersprachen *Java* und *JavaScript* haben dasselbe Entstehungsjahr, ein paar Ähnlichkeiten in der Syntax und die ersten vier Buchstaben des Namens gemeinsam - sonst nichts! LiveScript (der ursprüngliche Name) auf JavaScript umzubenennen war eine reine Marketingmaßnahme.

Node-RED installieren

Was brauchen wir dafür? *Alle folgenden Programme können kostenlos aus dem Internet geladen werden.*

Node-RED ist in JavaScript geschrieben. Damit ein JavaScript-Programm auch außerhalb eines Browsers läuft, ist eine Laufzeitumgebung notwendig. Dafür wurde node entwickelt. Mehr dazu auch auf der Wikipedia.

Node

Die folgende Beschreibung der Installation bezieht sich auf Windows. Schon auf der Startseite ist eine Schaltfläche für den Download zu finden. Die aktuelle Version ist (im August 2024) 20.17.0.

Die Installation ist denkbar einfach: die Datei node-v20.17.0-x64.msi (oder eine aktuellere Version) herunter laden und ausführen. Wer eine ältere Version installiert hat, sollte diese vorher entfernen. Weitere Betriebssystem werden auf der Download-Webseite angeboten. Eine gute Anleitung ist hier zu finden.

npm wird bei der Installation von Node gleichzeitig installiert.

npm

Eine dazu passende, gut gewartete Programmbibliothek ist npm, ursprünglich eine Abkürzung für *Node Package Manager*. Auch dazu gibt es eine detaillierte Beschreibung auf der Wikipedia. Und natürlich brauchen wir das Node-RED-Programm selbst.

Node-RED

Die aktuelle Version ist 4.0.2 (August 2024). Varianten werden für verschiedene Betriebssysteme angeboten samt einer Detailanleitung für Windows. Der *Quick Start* beschreibt alle Schritte. Das Installieren von Node.js haben wir schon erledigt. Mit der Eingabe von `cmd` im Windowssuchfenster unten wird die Eingabeaufforderung geöffnet. Dort

```
npm install -g --unsafe-perm node-red
```

eingeben und anschließend Node-RED mit der Eingabe

```
node-red
```

starten. Node-RED meldet die aktuelle Version (hier: v4.0.2) und weitere Details.

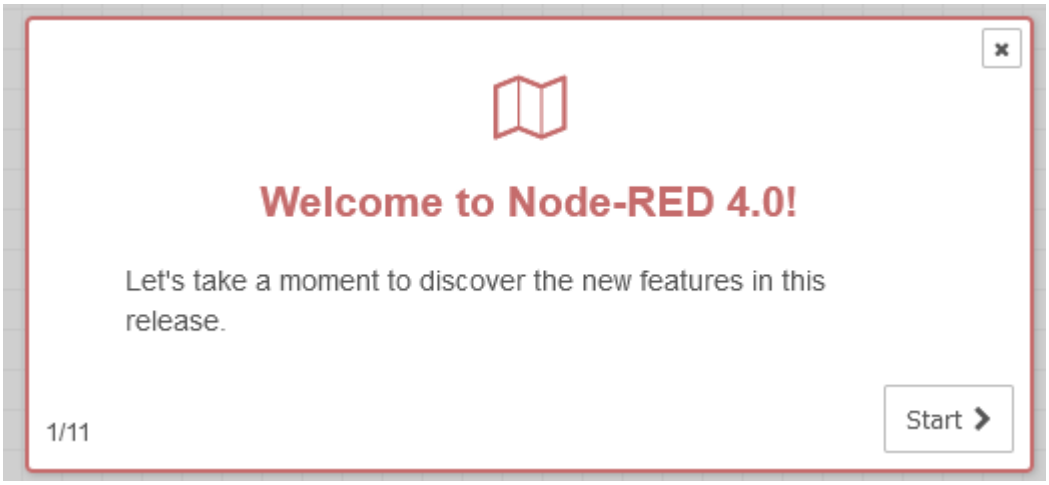
Diese Meldung ist wichtig:

```
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials

file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------

24 Aug 18:37:52 - [info] Server now running at http://127.0.0.1:1880/
24 Aug 18:37:52 - [warn] Encrypted credentials not found
```

Passwörter und andere Daten, die geheim gehalten werden sollen, werden als Credentials separat abgespeichert. Sie sollten unbedingt mit einem Passwort gesichert werden, das nur der Benutzer kennt. Die Datei settings.js ist im Verzeichnis `.node-red` zu finden.

Node-RED kennenlernen

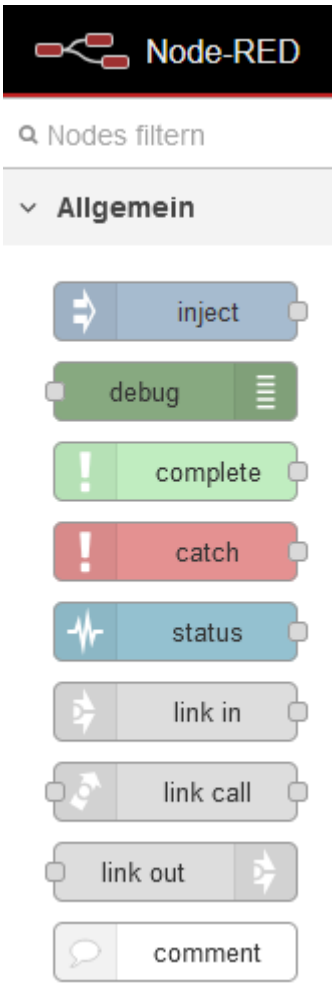
Node-RED läuft nun auf dem eigenen Rechner und wird über http://127.0.0.1:1880 aufgerufen, Der Browser zeigt die Begrüßung:



> **Note**
>
> Bei echten Anwendungen wird Node-RED auf einem kleinen Rechner, etwa auf einem Raspberry oder einem anderen Linux-Rechner, installiert. Der Rechner ist dann dauernd in Betrieb und soll wenig Strom aufnehmen.

Links am Bildschirm sind bereits vorinstallierte Programm-Module, sogenannte *Paletten*, zu finden. Die Node-RED Gemeinschaft hat eine große Anzahl weiterer Paletten für die unterschiedlichsten Anwendungsfälle vorbereitet. Wer will, kann auch eigene Paletten zusammen stellen.

Die Palette **Allgemein** beginnt mit den Nodes `inject` und `debug` . Diese wollen wir gleich einmal verwenden.

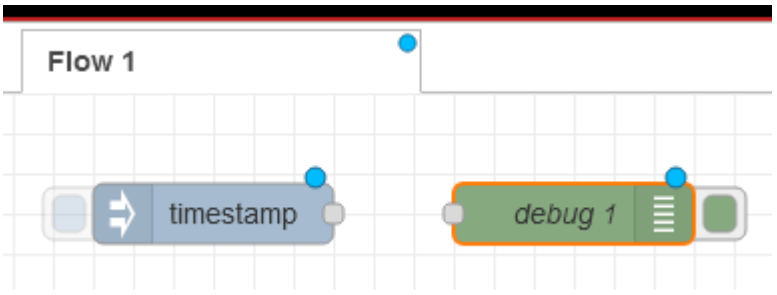


Der erste Flow

Flow bedeutet *Ablauf*, *Fluss*, *Bewegung*...

Vorläufig bleiben wir bei den Standard-Paletten. In jeder Palette gibt es wie erwähnt *Nodes* zur Auswahl. Wir bringen den inject–Node mit Anklicken und Ablegen auf die Arbeitsfläche in der Mitte des Schirms (drag-and-drop). Ebenso einen debug–Node. Der inject–Node injiziert Steuersignale in die Schaltung, der debug–Node gibt Signale (in Textform) aus. Beide sind für das Testen von Steuerungen wichtige Werkzeuge.

Die Arbeitsfläche sieht jetzt so aus:



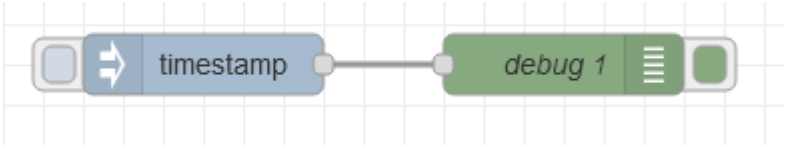
Die Nodes auf dieser Arbeitsfläche bilden einen *Flow*.

> **Note**
>
> Beliebig viele Flows können angelegt werden. Sie erhalten fortlaufende Nummern, können aber mit einem Rechtsklick auf den Flownamen umbenannt werden. Die Flows bringen Ordnung in die SmartHome Steuerung. Als Namen können beispielsweise Räume oder Stockwerke oder auch bestimmte Funktionen gewählt werden.

> **Tip**
>
> Anstatt nur ein paar Flows mir vielen Nodes zu erstellen, ist es besser, viele kleinere Flows mit wenig Nodes einzurichten. Das beschleunigt die Programmierung und das Testen. Außerdem wird die Steuerung übersichtlicher.

Der inject–Node und der debug-Node sind zu verbinden: den kreisförmigen (Ausgangs-)Anschlusspunkt am rechten Rand des inject-Nodes anklicken und mit gedrückter Maustaste eine Linie zum linken (Eingangs-)Anschlusspunkt des debug–Nodes ziehen. Maustaste auslassen — fertig! So wird in Node-RED *verdrahtet*.

Solche einfachen Schritte lassen die Vorteile der grafischen Programmierung erkennen.

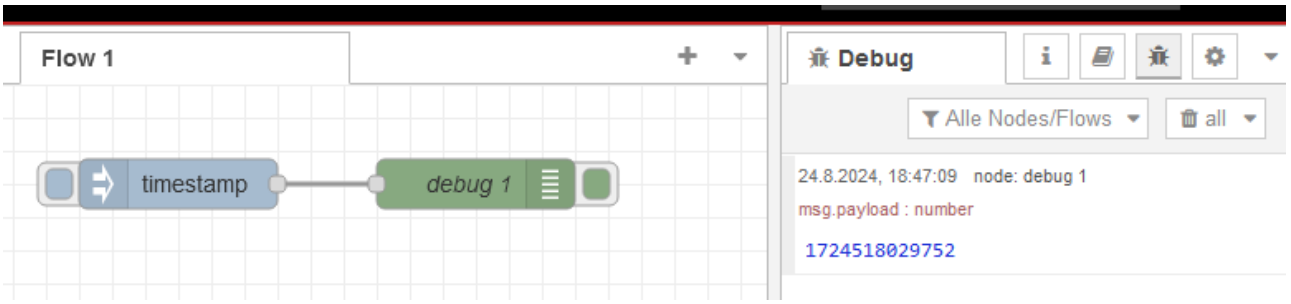


Dieses Bild vermittelt auch den Fluss der MQTT-Daten: durch die Verbindung hat der debug–Node die Daten des inject–Nodes abonniert.

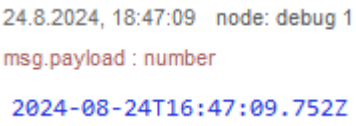
Debugger

Das Debuggerfenster links am Bildschirm zeigt die Ausgaben der debug–Nodes und weitere Informationen, zum Beispiel Fehlermeldungen. Zum Öffnen ist `Strg` `g` und `d` einzugeben oder das Ikon anzuklicken. Es stellt ein *schädliches Insekt* (engl. *bug*) dar.

Mit dem Anklicken der linken grauen Schaltfläche des inject–Nodes sendet dieser ein Signal aus. Voreingestellt ist ein Zeitstempel (*timestamp*) mit dem aktuellen Datum und der Uhrzeit. Wie in Linux üblich, wird die Anzahl der Millisekunden seit Beginn der Epoche am 1. Jänner 1970 00:00 UTC angeben.

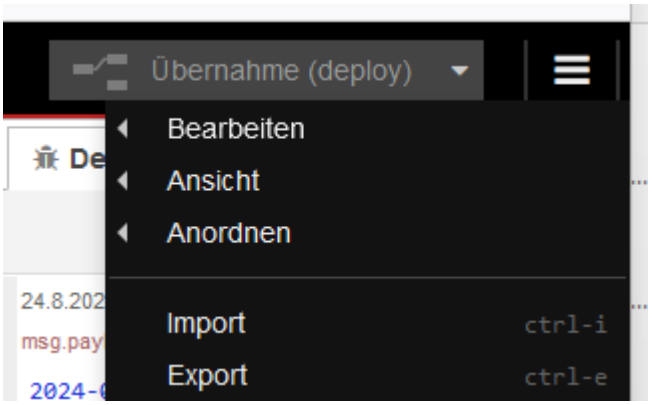


Ein Klick auf die Zahl liefert Datum und Uhrzeit in leichter lesbarer Form.



Großartig — wir haben unseren ersten Flow erstellt und erprobt!

Sichern von Flows



Nun, dieser erste Flow ist ja nicht sehr kompliziert und kann, wenn er irrtümlich gelöscht wird, leicht wieder hergestellt werden. Trotzdem ist es eine gute Idee, gleich einmal mit dem Sichern anzufinden.

Zum Sichern wird ein Flow oder das ganze Programm als JSON-Objekt exportiert:

Das Hamburger-Menü (drei waagrechte Striche rechts oben) öffnen und `Export` auswählen. Im neuen Fenster rechts oben `Formatiert` wählen und dann `Download` anklicken.

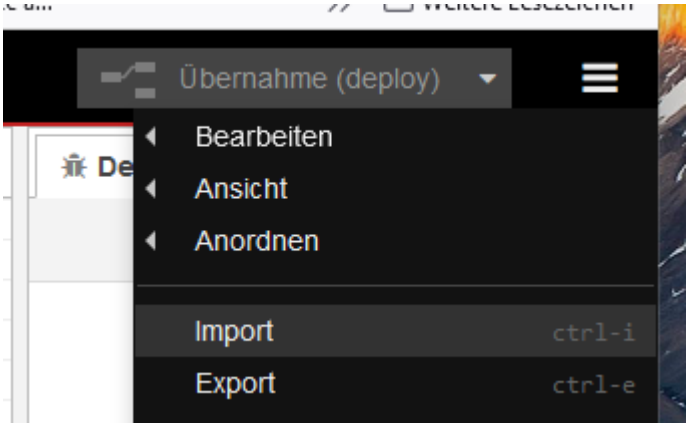
Im Download-Ordner findet sich nun eine Datei mit dem Namen `flows.json` (oder auch `flows(1).json` usw.) Ihr Inhalt (beispielsweise):

```
[
    {
        "id": "4d7eab1cefb4aa8b",
        "type": "tab",
        "label": "Flow 1",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "c46de3abe9b2fc1f",
```

```
        "type": "inject",
        "z": "4d7eab1cefb4aa8b",
        "name": "",
        "props": [
            {
                "p": "payload"
            },
            {
                "p": "topic",
                "vt": "str"
            }
        ],
        "repeat": "",
        "crontab": "",
        "once": false,
        "onceDelay": 0.1,
        "topic": "",
        "payload": "",
        "payloadType": "date",
        "x": 120,
        "y": 60,
        "wires": [
            [
                "45fbb50f8b149142"
            ]
        ]
    },
    {
        "id": "45fbb50f8b149142",
        "type": "debug",
        "z": "4d7eab1cefb4aa8b",
        "name": "debug 1",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "false",
        "statusVal": "",
        "statusType": "auto",
        "x": 300,
        "y": 60,
        "wires": []
    }
]
```

Ganz schön viel Code für so eine einfache Steuerung. Aber das Programmstück ist ja nicht für einen menschlichen Leser gedacht. Wir erkennen einen Liste von drei JSON-Objekte für den *Flow* und für `inject` und `debug`.

Um nun das Programm wieder zurück zu spielen, erzeugen wir einen leeren Flow und wählen im ≡-- Menü `Import` und danach die gespeicherte Datei.



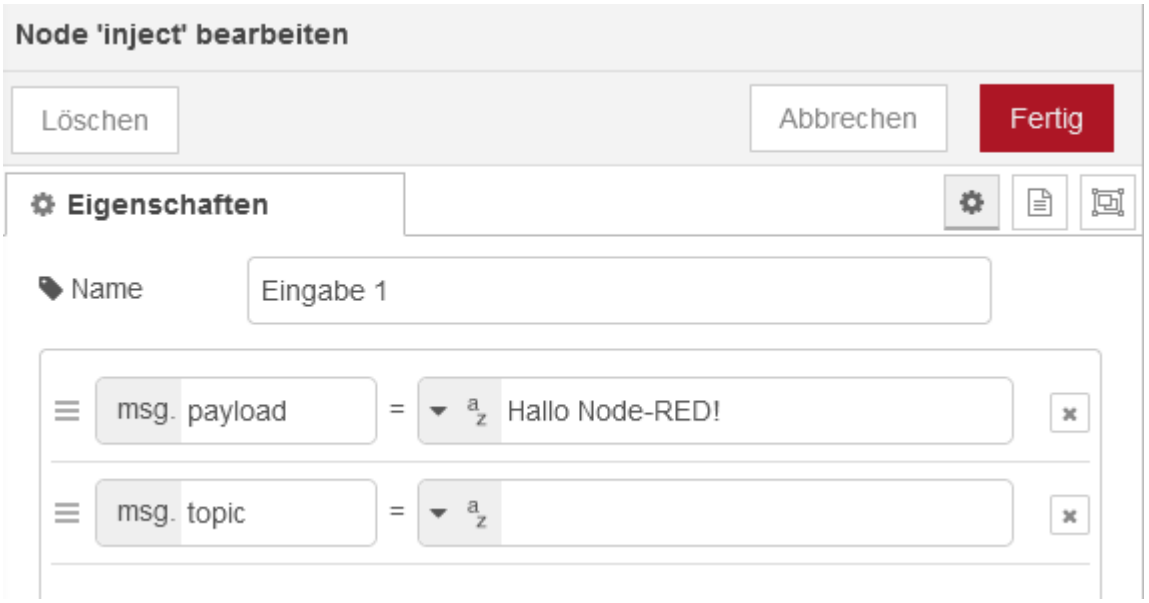
> **Tip**
>
> In vielen Programmieranleitungen und Beschreibungen werden Flows auf diese Art veröffentlicht: eine Datei mit JSON-Inhalt wird importiert und das Programmstück kann sofort verwendet werden.

Ein paar Verbesserungen

Doppelklick auf den inject–Node: wir geben ihm den Namen `Eingabe 1`. Außerdem soll nicht die Uhrzeit, sondern die Zeichenkette "Hallo Node-RED!" gesendet werden.

payload ist der Standardname innerhalb eines JSON-Objekts für die zur Steuerung oder zur Übertragung von Zuständen verwendeten Daten. Um eine Zeichenkette zu übertragen, wird der Datentyp auf Zeichenkette (String) umgestellt; dazu das Symbol `a/z` anklicken und den gewünschten Text dahinter (ohne Anführungszeichen) eingeben.

Der Eintrag topic bleibt unverändert, spielt aber bei bestimmten Nodes eine wichtige Rolle. Mehr dazu im zweiten Teil.

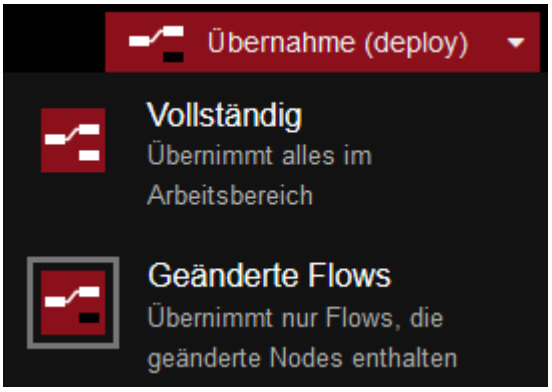


Auf `Fertig` klicken

Nach jeder Änderung, die ja vorerst nur im Browserfenster sichtbar ist, muss die eigentliche Steuerung auf den aktuellen Stand gebracht werden. Solange das nicht der Fall ist, erinnert *ein blauer Kreis* auf dem jeweiligen Symbol und neben dem Flow-Namen daran.

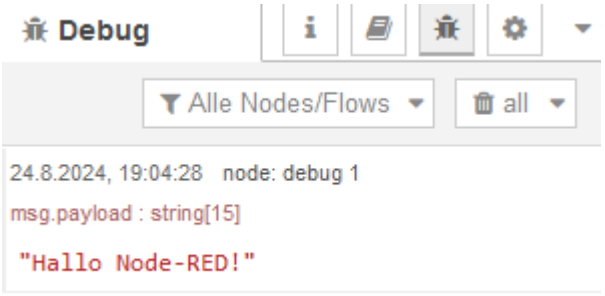
> **Tip**

> Wir können (vor allem bei großen Steuerungen) Zeit sparen, wenn wir mit einem Klick auf den Pfeil neben `deploy` auf `Geänderte Flows` umstellen: damit werden nur die Änderungen und nicht die gesamte Steuerung übertragen.



`Übernahme` und Löschen der debug-Liste mit `Strg` `Alt` `l` (Hinweis: kleines "L")

`Inject` anklicken, der Text wird im Debug-Fenster angezeigt.



> **Note**
>
> Über das Testen:
>
> - Der Inject-Node speist beliebige Signale in das System ein.
> - Der Debug-Node liest Signale aus und zeigt sie im Debugfenster.

JSON

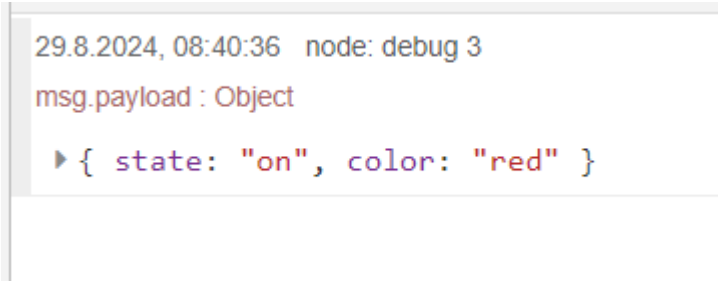
Daten im JSON-Format spielen bei Node-RED eine große Rolle. Wenn beispielsweise eine RGB-Lampe eingeschaltet und gleichzeitig auf rotes Licht umgestellt werden soll, könnte ein Steuerbefehl als JSON-Objekt (vereinfacht) so aussehen:

```
1 { "state":"on", "color":"red"}
```

Wieder muss der Typ bei `payload` umgestellt werden: die gewungenen Klammern `{ }` erinnern an die JSON-Darstellung.

> **Note**
>
> RGB steht für Red-Green-Blue oder Rot-Grün-Blau.

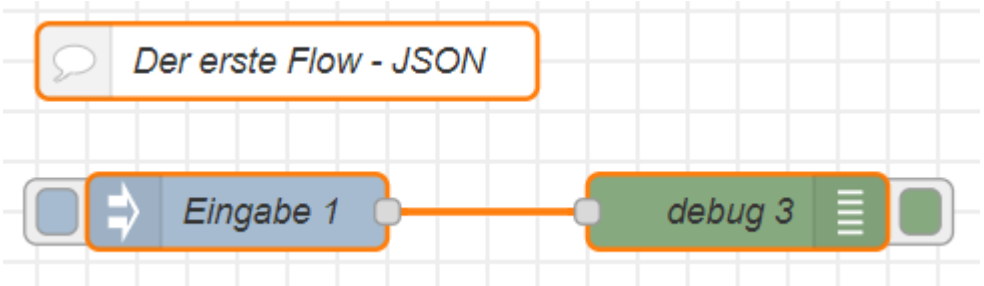
Im Debug-Fenster sieht das Ergebnis (als JSON-Objekt) so aus:



Auf den kleinen Pfeil klicken liefert eine übersichtlichere Darstellung. Diese ist bei größeren Objekten sehr nützlich.

Kommentare

Mit dem Element *comment können* und *sollen* Kommentare in die Grafik eingebaut werden.



Die orange Umrandung zeigt an, dass diese Elemente gerade *ausgewählt* sind. Wie bei einer Textverarbeitung können ausgewählte Elemente kopiert, verschoben oder gelöscht werden.

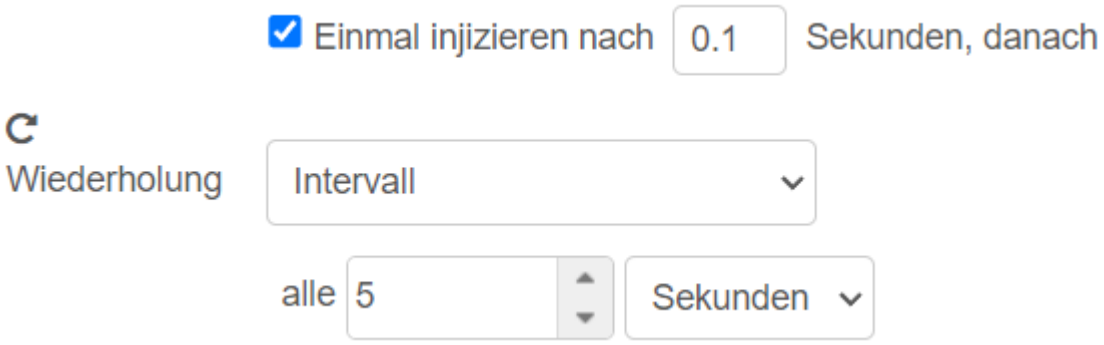
Weitere Varianten

inject–Node

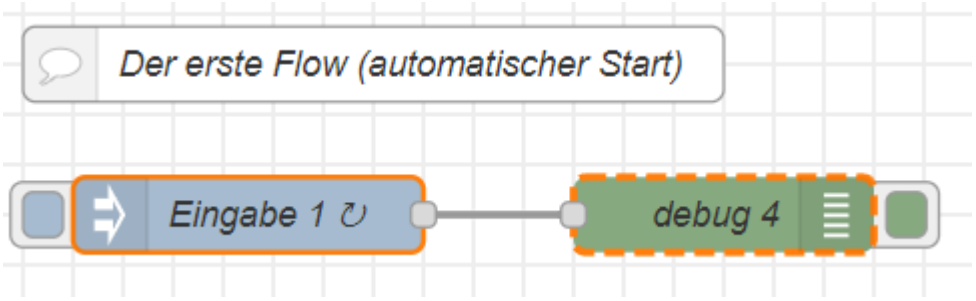
Der Node soll ...

1. automatisch 0,1 Sekunden nach dem Start des Programms aktiv werden, ohne dass die Schaltfläche betätigt werden muss und
2. den (eingestellten) Befehl alle 5 Sekunden wiederholen.

So wird's gemacht:



Im Arbeitsbereich wird die Wiederholung mit einem kreisförmigen Pfeil angezeigt:



> **Tip**
>
> Wenn in dem Node die Wiederholung eingeschaltet ist, startet der Ablauf auch dann, wenn *Einmal injizieren* nicht ausgewählt ist.$}

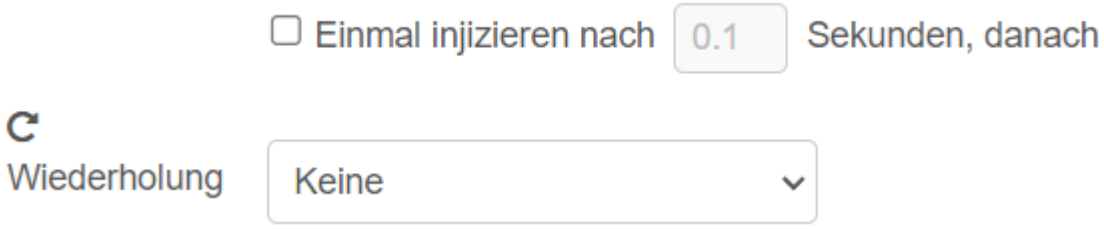
debug–Node

Nicht vergessen: Änderungen Jeweils mit `Fertig` abschließen, die Debug-Liste mit `Strg` `Alt` `l` [kleines L] löschen ünd dann auf `Übernahme` klicken. Der Ablauf startet automatisch!

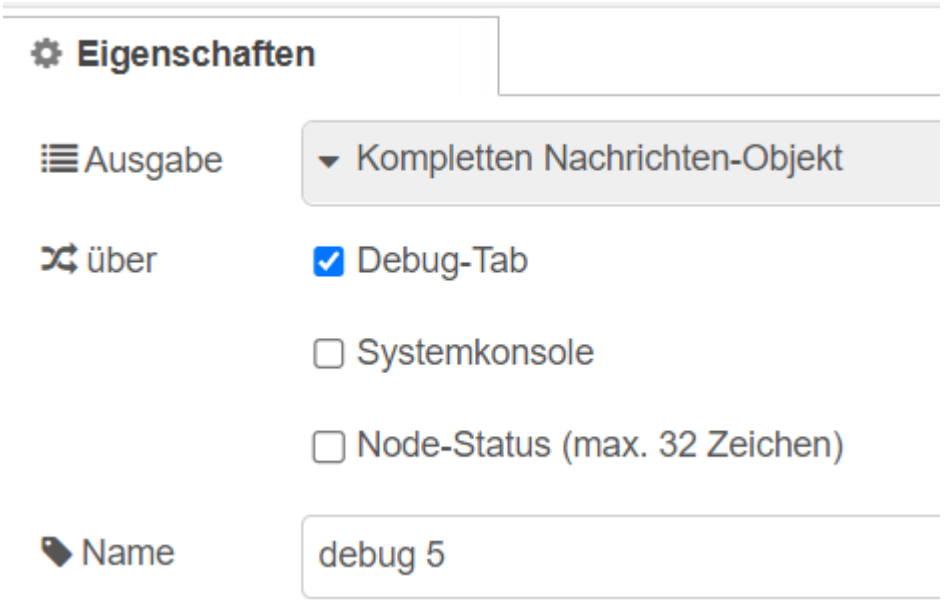
Na ja, vielleicht nicht ganz sinnvoll, eine bereits eingeschaltete Lampe immer wieder einzuschalten. Aber hier geht es ja vor allem um das Kennenlernen von Node-RED.

Ein nachgeschaltetes Stromstoßrelais wäre interessanter.

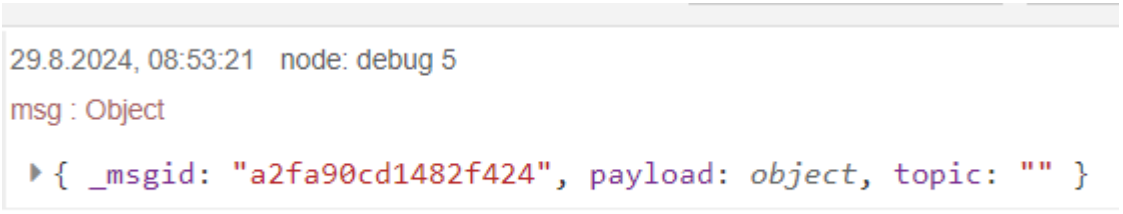
Schalten wir den automatischen Start und die Wiederholung wieder aus:



Eine ausführliche Liste der Debug-Daten wird durch das Umstellen auf `Komplettes Nachrichten-Objekt` erzeugt.



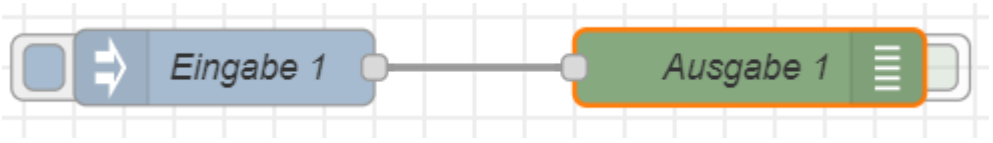
Das Ergebnis:



Mit Klicks auf die Pfeile nach rechts werden die Objekte aufgeklappt. Das Ergebnis ist übersichtlicher:

Bei größeren Projekten werden viele Debug-Nodes verwendet, die dann auch verständliche Namen erhalten sollen. Der so gewählte Name (hier: *Ausgabe 1*) erscheint danach auch in der Debug-Liste.

Wird ein Debug-Node einmal nicht benötigt und soll er trotzdem noch nicht entfernt werden, kann er über die Schaltfläche rechts deaktiviert werden. Die Schaltfläche wird dann hellgrau.



> **Tip**
>
> Das Debug-Protokoll kann nicht beliebig lang sein, ältere Meldungen werden automatisch gelöscht. Das kann dazu führen, dass Ausgaben, die gerade angesehen werden, plötzlich verschwinden, weil in einem anderen Flow gerade sehr viele Debug-Einträge dazu kommen. Debug-Nodes sollten deaktiviert werden, wenn sie gerade nicht benötigt werden.

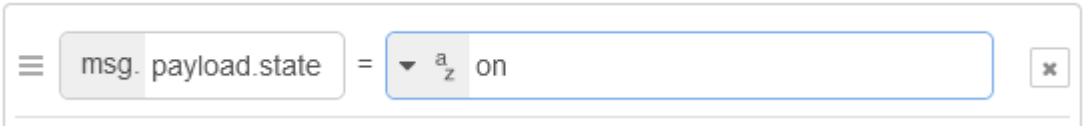
Eine Lampensteuerung

Ein- und Ausschalten

Vorerst wird der Zustand der Lampe (`eingeschaltet` oder `ausgeschaltet`) durch einen Text im Debug–Fenster simuliert. Der spätere Umbau auf eine echte Steuerung ist dann einfach.

Der vorhandene inject–Node wird geändert:

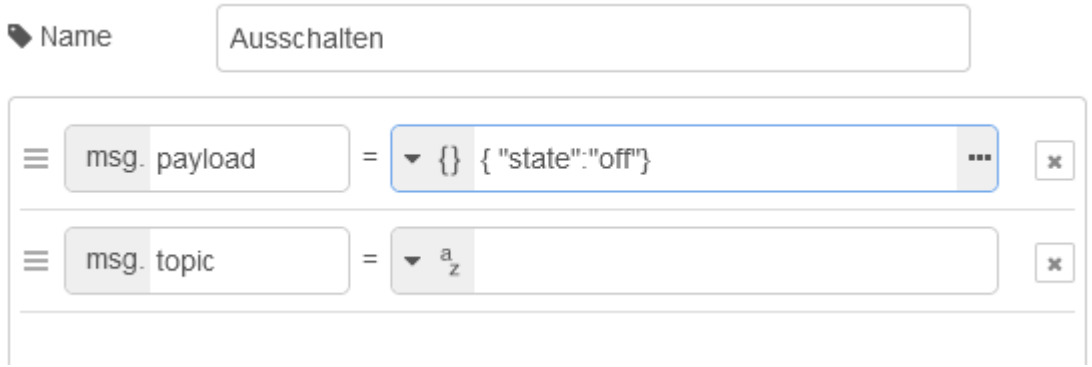
Statt ein JSON-Objekt zu senden, wäre hier auch die Einstellung



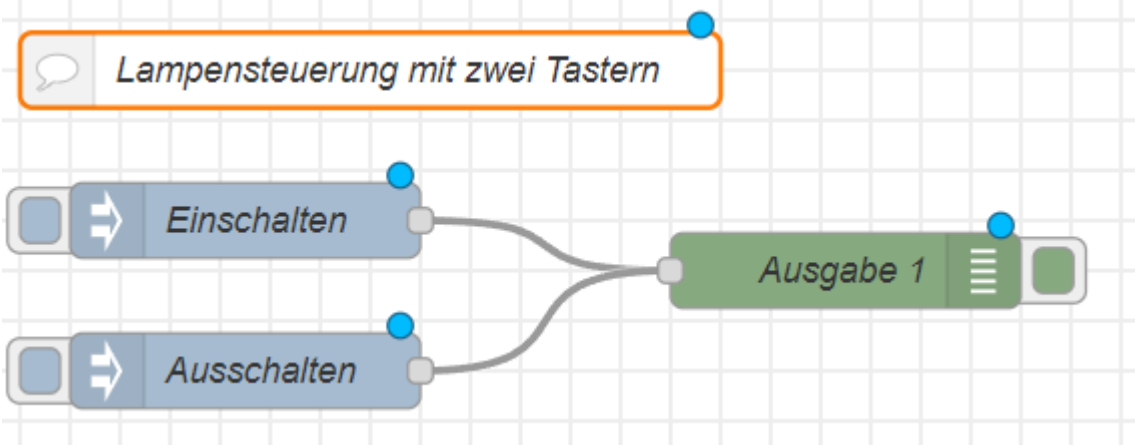
möglich. Das ist zwar einfacher zu schreiben, aber an anderen Stellen nicht zu verwenden. Bleiben wir also bei der Übergabe eines JSON-Objekts.

Wir brauchen noch einen zweiten Node für das Ausschalten. Wie bei einer Textverarbeitung kopieren wir den vorhandenen Node und ändern die Kopie:

- Zum Auswählen den Node anklicken - der Rand wird orange
- Mit `Strg` `C` oder mit einem rechten Mausklick, gefolgt von `c` oder `Kopieren`, in die Zwischenablage kopieren
- Den Cursor ein wenig unterhalb positionieren, die Kopie mit `Strg` `V` einfügen und an die gewünschte Stelle verschieben
- Den neuen Node doppelt anklicken und andern auf:

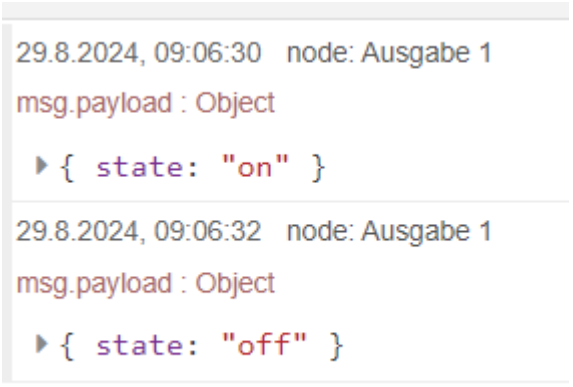


Der neue Node wird mit der *Ausgabe 1* verbunden und der Kommentar aktualisiert. Alles so platzieren, dass es nett aussieht:

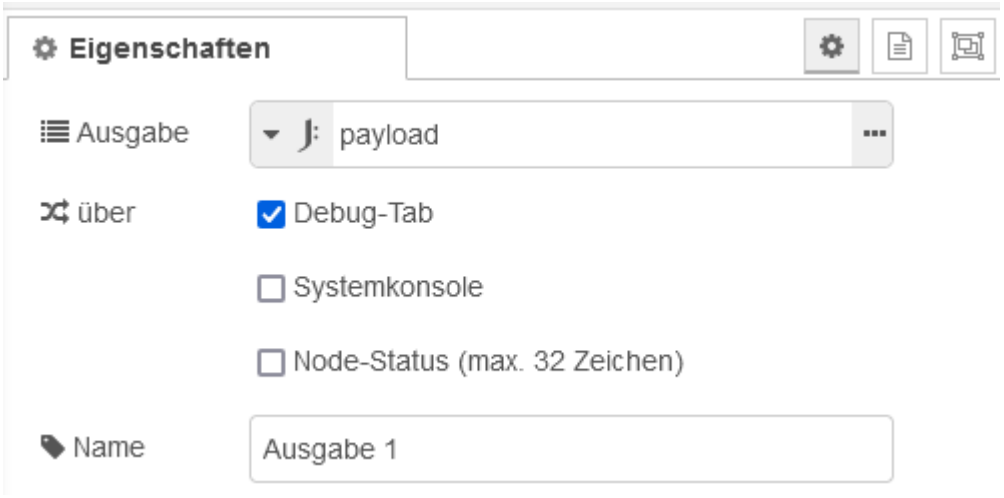


Was fehlt noch? Richtig, die `Übernahme`.

Wenn wir nun den ersten und dann den zweiten inject–Node anklicken, sehen wir das Ergebnis im Debug–Fenster.

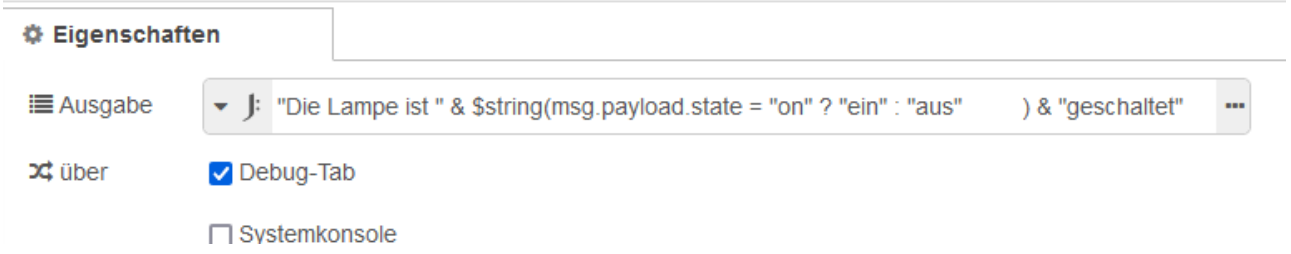


Im debug–Node kann die Ausgabe noch verschönert werden.



J: steht für JSONata, eine Sprache, mit der JSON-Daten ausgewertet und umgewandelt werden können.

Ändern auf:



Die Erklärung, was JSONata alles kann, wäre einen eigenen Beitrag wert.

Hier die Kurzfassung:

- Mit msg.payload.state = "on"wird geprüft, ob state "on" ist. Das `=`-Zeichen ist in JSONata ein Vergleichsoperator, der Vergleich ergibt *false* oder *true*.
- Wenn *true* liefert der Ausdruck `msg.payload.state = "on" ? "ein" : "aus"` die Zeichenkette `ein`, andernfalls `aus`. Genau genommen sollte auch noch `state = "off"` geprüft werden. Möglich wäre
  `msg.payload.state = "on" ? "ein" : (msg.payload.state = "off" ? "aus" : "nicht ")`
  Lassen wir das weg, , das Beispiel soll ja einfach bleiben.

- Für die weitere Verarbeitung wird JSONata mitgeteilt, dass der ganze Ausdruck ein String ist:
  `$string(msg.payload.state = "on" ? "ein" : "aus")`
- Das "&"-Zeichen verkettet alles zu einem lesbaren Satz.
  "Die Lampe ist " & $string(msg.payload.state = "on" ? "ein" : "aus") & "geschaltet"

> **Note**
>
> Es heißt *eingeschalte**t*** und *ausgeschalte**t***, nicht *eingeschalte**n*** und *ausgeschalte**n***.

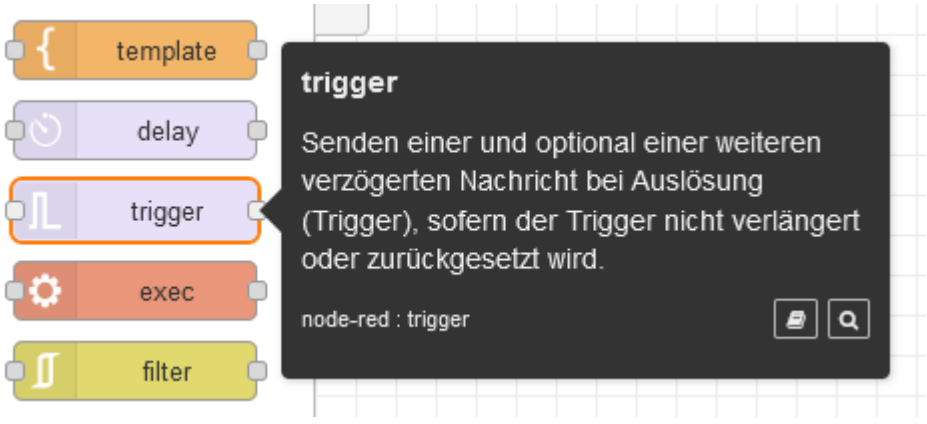
Die neue Ausgabe:

```
29.8.2024, 09:11:10   node: Ausgabe 1
msg : string[27]
"Die Lampe ist eingeschaltet"
29.8.2024, 09:11:12   node: Ausgabe 1
msg : string[27]
"Die Lampe ist ausgeschaltet"
```

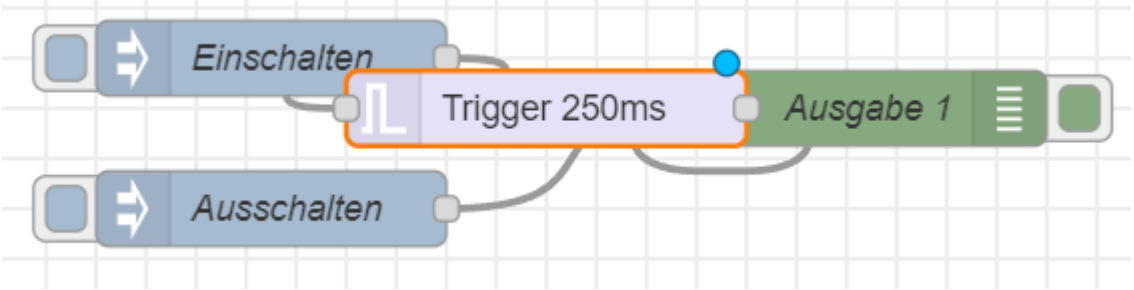
Ein 3-Minuten-Licht

Nach einem Knopfdruck soll das Licht im Stiegenhaus 3 Minuten lang brennen, jedoch setzen wir zum Testen die Zeit auf 5 Sekunden.

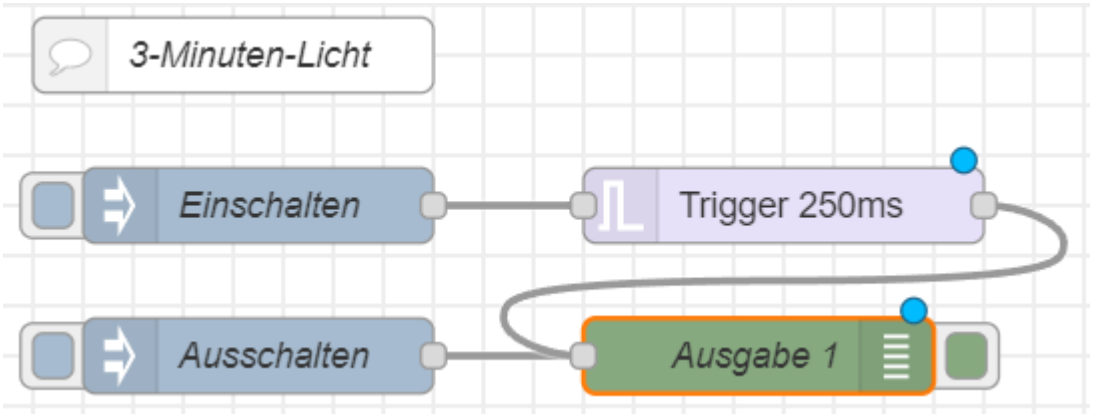
Wir wählen aus der Palette `Funktion` den Node *trigger*.



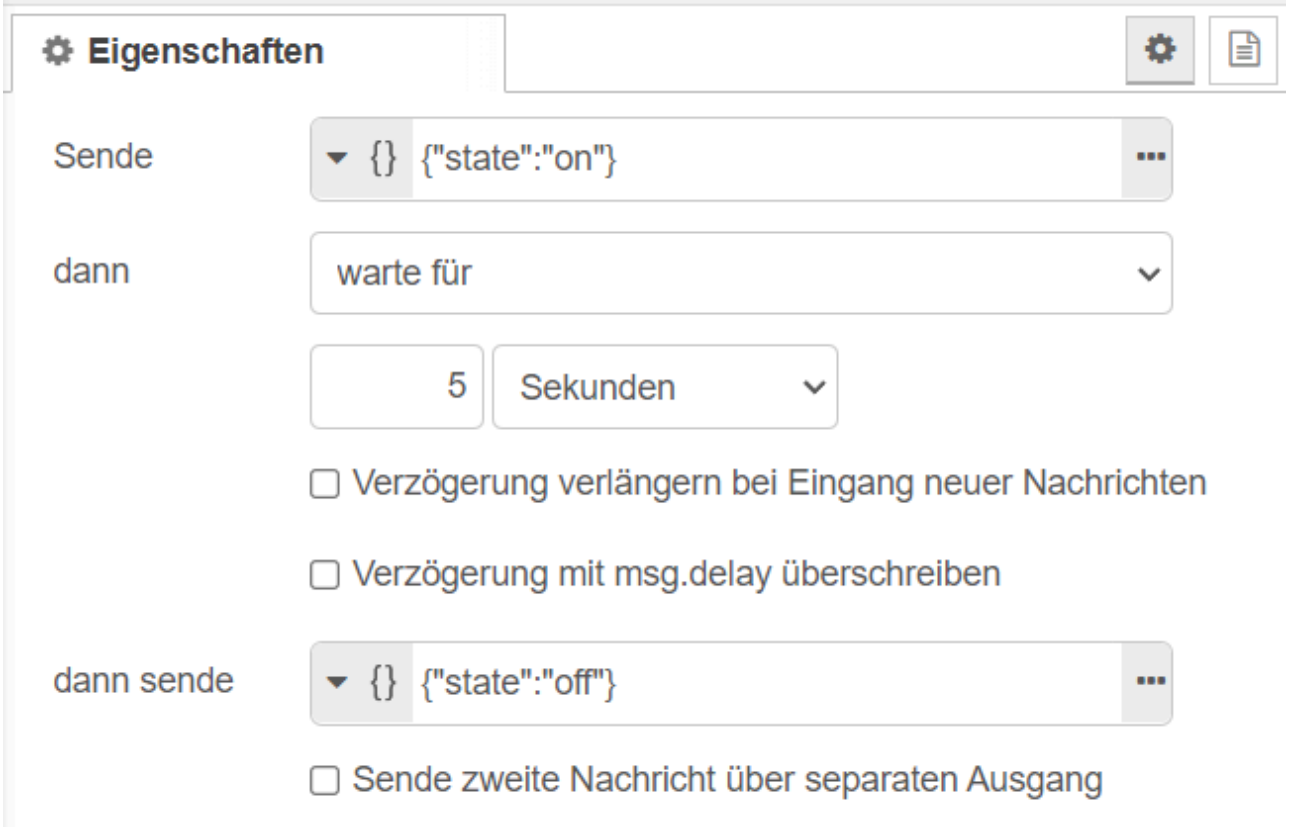
Wir ziehen den trigger–Node auf den Flow und lassen ihn zwischen Einschalten und Ausgabe 1 fallen:



Das schaut jetzt nicht sehr schön aus, daher ordnen wir die Nodes ein wenig:

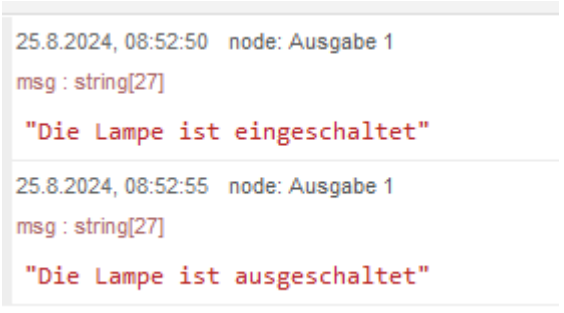


Der Verbindungslinien passen sich selbsttätig an. In dem neuen Node *trigger* ist einiges zu ändern:



> **Tip**
>
> Der Typ der beiden Sende-Einträge ist von *Zeichenkette* auf *JSON* umzustellen.

Dann übernehmen, Debugger-Fenster löschen, ausprobieren:



Die Lampe wird eingeschaltet und nach 5 Sekunden selbsttätig ausgeschaltet.

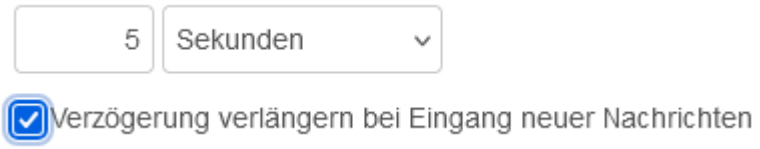
Der Node *Ausschalten* ist noch da - damit kann das Licht vorzeitig ausgeschaltet werden.

> **Note**

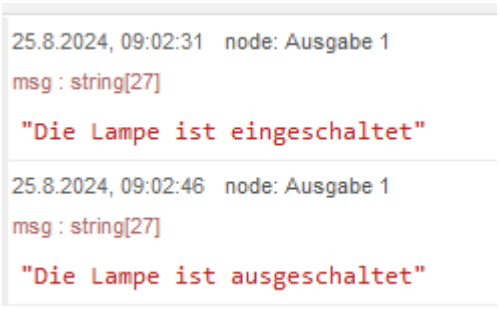
> Der Node *Einschalten* sendet beim Anklicken immer noch {"state":"on"}. Eigentlich ist es gleichgültig, was gesendet wird — es muss nur irgendein Signal beim trigger–Node ankommen. Lassen wir es einfach, wie es ist.

Zeit verlängern

Der trigger–Node zählt hier 5 Sekunden lang. Um die Zeit zu verlängern, solange das Licht noch brennt, ändern wir den Node:



Wenn jetzt während der 5 Sekunden die Taste `Einschalten` immer wieder gedrückt wird, verlängert sich die Einschaltzeit.



Der Zeitstempel zeigt, dass jetzt das Licht 15 Sekunden lang gebrannt hat.

Eine Schaltuhr

Neue Aufgaben: wie können wir

- die Lampe von 20 Uhr bis 24 Uhr oder
- von Sonnenuntergang bis 23 Uhr eingeschaltet lassen?

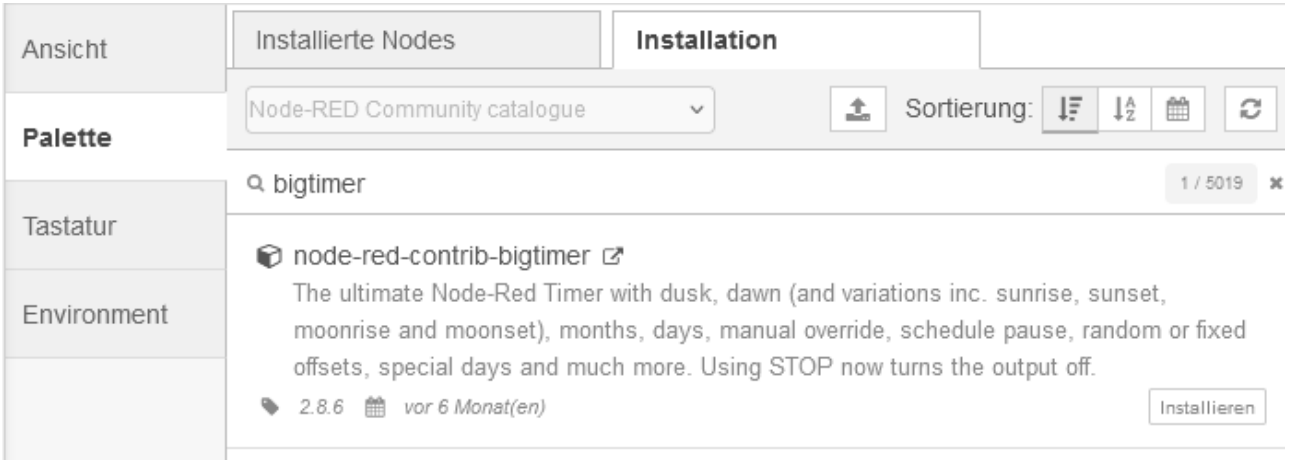
Dazu holen wir einen passenden Node aus der umfangreichen Programmbibliothek *npm*.

Einen Node aus *npm* verwenden

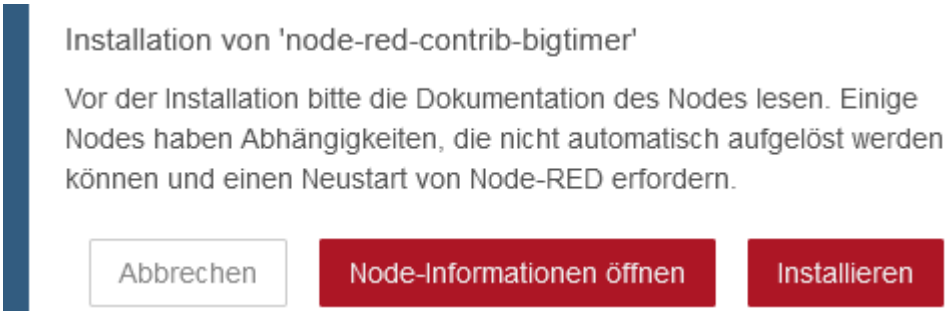
So geht das:

- Auf das Hamburger-Menü ☰ klicken
- `Palette verwalten` und
- `Installation` wählen
- Im Feld *Module durchsuchen* `bigtimer` eingeben

> **Note**
>
> Node-RED zeigt an, dass mehr als 5000 Module verfügbar sind. 😊 Manche sind sehr einfach, manche reichlich komplex und wieder einige fehlerhaft. 👺



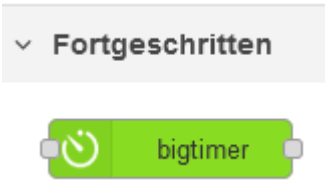
Auf `Installieren` klicken und nochmals mit `Installieren` bestätigen



`Schließen`

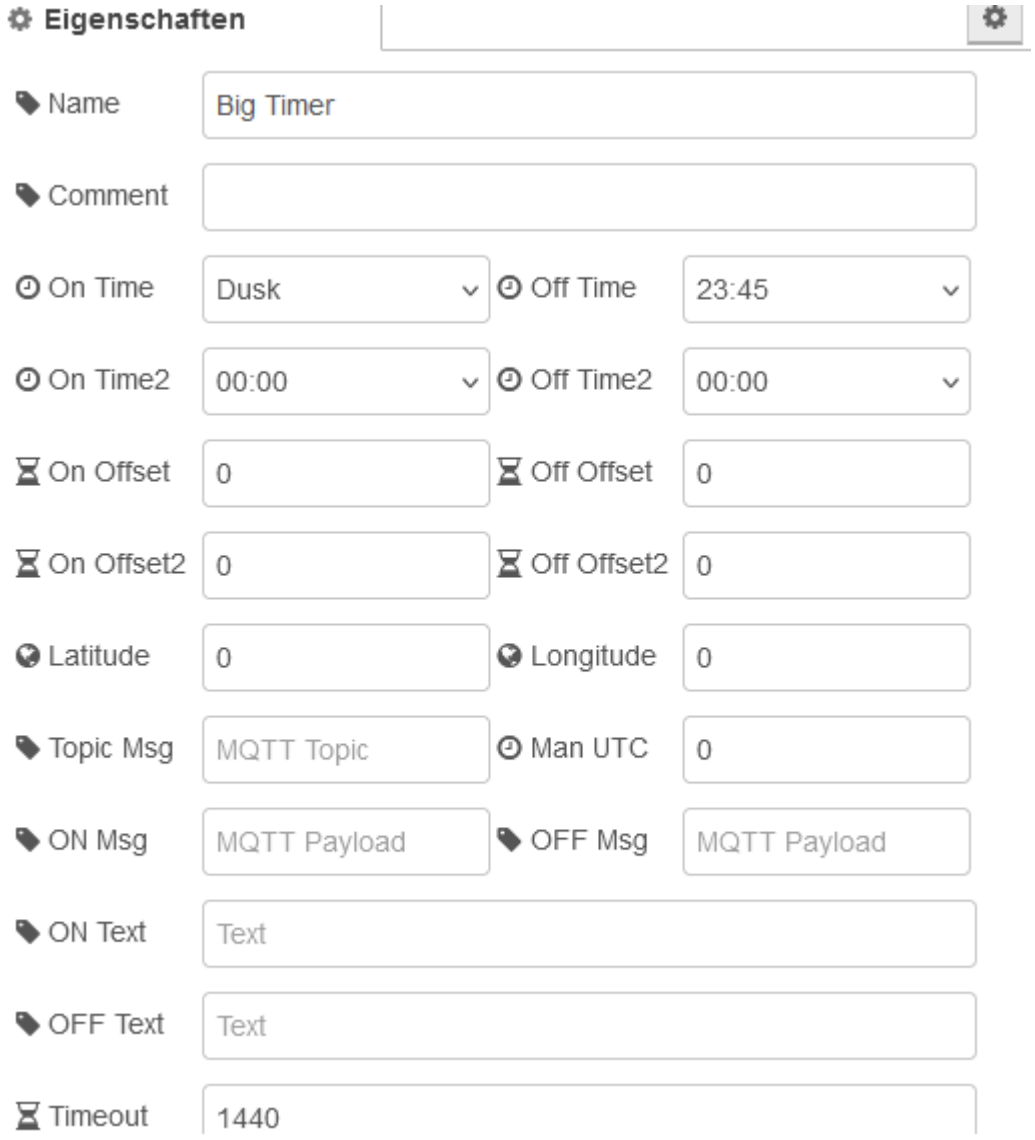
> **Note**
>
> Der Name der meisten Node-RED Paletten beginnt mit `node-red-contrib`-: npm enthält mehr als zwei Millionen Programmpakete und da muss schon eine Systematik bei den Namen eingehalten werden. `contrib` weist darau hin, dass dieses Programmpaket (diese Palette) ein Beitrag von einem Node-RED-Nutzer ist.

Bei den Paletten ganz links am Bildschirm finden wir an letzter Stelle unter der Überschrift *Fortgeschritten* eine neue Palette



Wir ziehen einen Big Timer–Node auf die Arbeitsfläche.

Der Big Timer ist gutes Beispiel dafür, wie vielfältig und leistungsfähig die Open Source Module sind, die für Node-RED angeboten werden. Hier gibt es eine Menge einzustellen, ein konkretes Beispiel ist in der Langfassung zu finden.



Zum Einschalten dienen die On...-Felder, zum Ausschalten die Off...-Felder.

Zwei Timer können unabhängig von einander eingestellt werden: Timer 1 (ohne Suffix), Timer 2 (mit Suffix 2).

Zeitangaben

- Unter `On Time` und `Off Time` werden Zeiten in 15-Minuten-Abständen eingestellt.
- Zusätzlich stehen bei `On Time` und `Off Time` folgende Optionen zur Auswahl:
  - Day End = 24:00 Uhr. Der zusätzliche Eintrag ist notwendig, weil 00:00 für das Fehlen der Zeitangabe verwendet wird.

    > **Note**
    >
    > Besser wäre es wohl gewesen, eine Option `Keine Auswahl` vorzusehen und `00:00` als gültige Zeit zu belassen. Das Beispiel zeigt die Vorteile der Open Source-Software: wer will kann seine eigene Version vom *Big Timer* erstellen und verwenden.

  - Dawn = Beginn der bürgerlichen Morgendämmerung, die Sonne steht 6° unter dem Horizont
  - Dusk = Ende der bürgerlichen Abenddämmerung, die Sonne steht 6° unter dem Horizont
  - Solarnoon = wahrer astronomischer Mittag, die Sonne steht genau im Süden
  - Sunrise = Sonnenaufgang

    - Sunset = Sonnenuntergang
    - Night = Beginn der astronomische Nacht, Ende der astronomischen Abenddämmerung. Die Sonne steht 18° unter dem Horizont
    - Night end = Ende der astronomischen Nacht, Beginn der astronomischen Morgendämmerung: Die Sonne steht 18° unter dem Horizont
    - Moonrise = Mondaufgang
    - Moonset = Monduntergang
- Bei der `Off Time` kann alternativ die Einschaltdauer mit 1, 2, 5, 10, 15, 30, 60, 90 und 120 Minuten eingestellt werden.
- Der Offset-Eintrag dient zur Korrektur um Minuten, die zur eingestellten Zeit addiert werden (negative Zahlen sind auch möglich).

In ON Msg, OFF Msg, ON Text und OFF Text sind Strings (ohne Anführungszeichen) einzutragen.

Damit der Big Timer die astronomischen Zeiten errechnen kann, muss der Ort angegeben werden.

- Beispiel Wien: Latitude = geografische Breite = 48.21, Longitude = geografische Länge = 16.37.

> **Note**
>
> - Geografische Breite nördlich des Äquator: positiv, südlich: negativ
> - Geografische Länge ostwärts von Greenwich: positiv, westwärts: negativ.

Die Einstellungen werden auch auf der Webseite des Big Timers erklärt.

Der Big Timer fragt alle Einstellungen einmal pro Minute ab und ändert dementsprechend die drei Ausgänge:

- Ausgang 1 sendet jede Minute ein `msg`-Objekt mit den Zeichenketten, die unter `ON Msg` bzw. `OFF Msg` eingetragen worden sind, als `payload`. Das `msg`-Objekt enthält noch eine Reihe weiterer Informationen
- Ausgang 2 sendet jede Minute ein `msg`-Objekt mit den Zahlen `0` bzw. `1` als `payload`. Das msg-Objekt enthält noch eine Reihe weiterer Informationen.
- Ausgang 3 liefert *nur bei einer Änderung* ein `msg`-Objekt mit den Zeichenketten von `ON Text` bzw. `OFF Text` als `payload`.

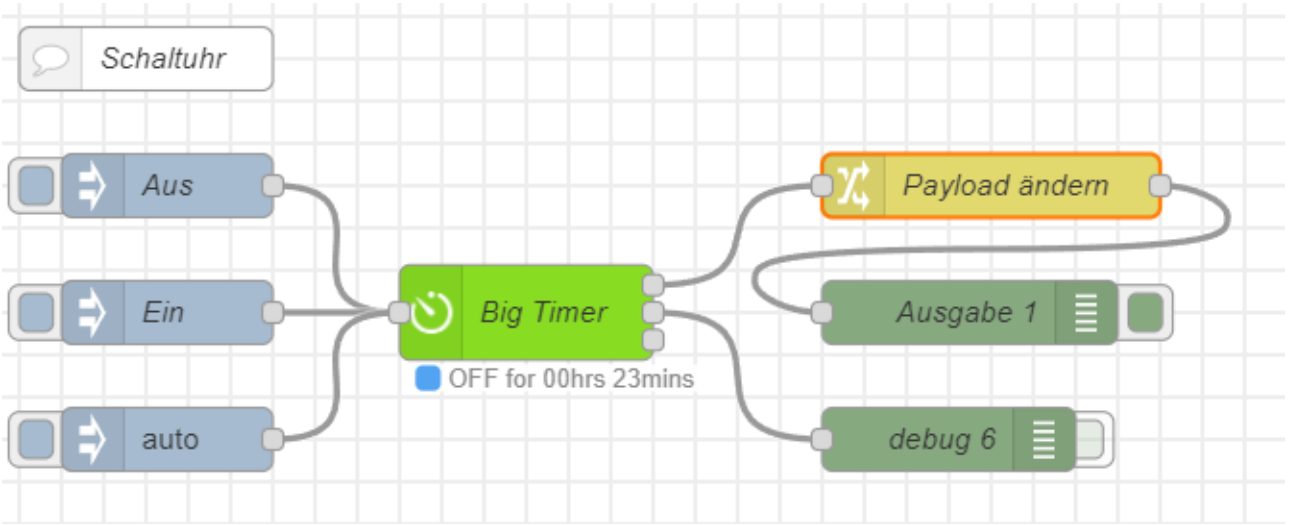
Big Timer ist ein universell einsetzbarer Node. Nur die Dokumentation könnte ausführlicher sein — manche Funktionen muss man einfach ausprobieren.

Ansteuern der Lampe

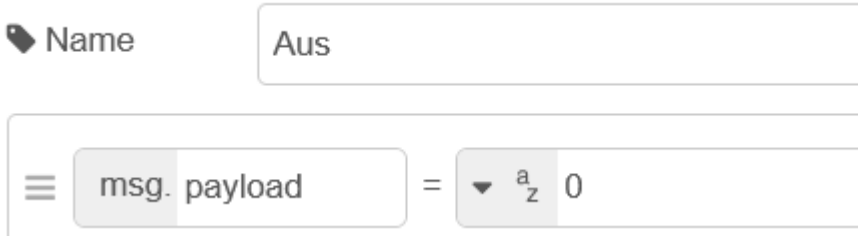
Der Big Timer gibt als `msg.payload` einen String, zum Beispiel `on` oder `off` aus. Wir brauchen aber zum Steuern unserer (im Debugger simulierten) Lampe die Objekte `{"state":"on"}` und `{"state":"off"}`. Wie das im Details gemacht wird, steht in der Langfassung.

Würden wir `{"state":"on"}` in das Feld `ON Msg` eintragen, nützt das auch nichts — es bleibt ein String, wir brauchen aber ein Objekt.

Abhilfe schafft der Node `change` aus der Palette Funktion. Eine zweite Lösung verwendet den json–Node. Wir fügen außerdem noch drei inject-Nodes und einen debug-Node hinzu:



Das Testen des Big Timers ist etwas mühsam, wenn wir darauf warten, dass eine eingestellte Zeit den Ausgang umschaltet. Die beiden Nodes links in der Arbeitsfläche (*Aus* und *Ein*) senden `0` bzw. `1` und *übersteuern* den Big Timer.

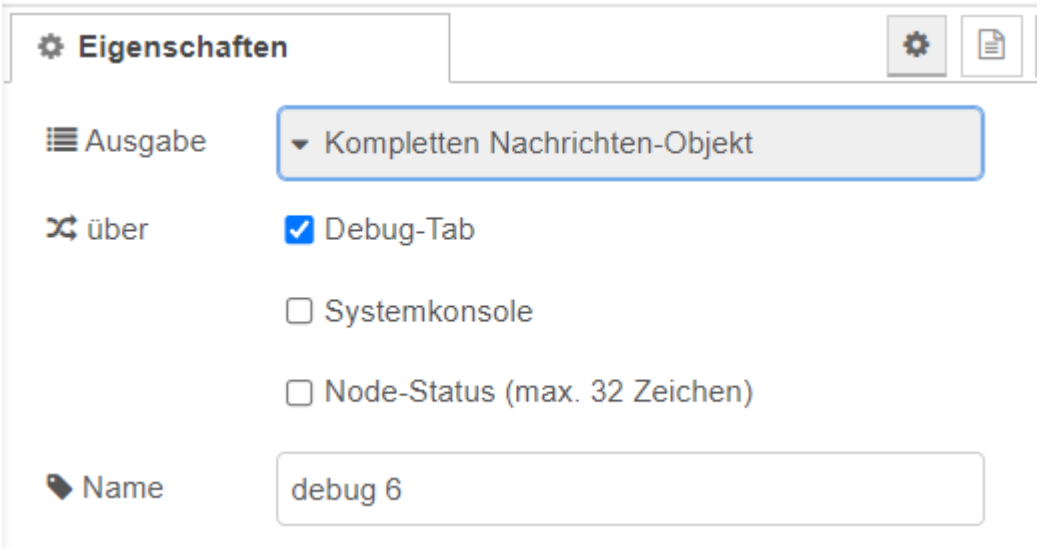


Analog dazu der Node *Ein*.

Der dritte Node *auto* schaltet mit Payload `auto` den automatischen (zeitgesteuerten) Ablauf wieder ein.

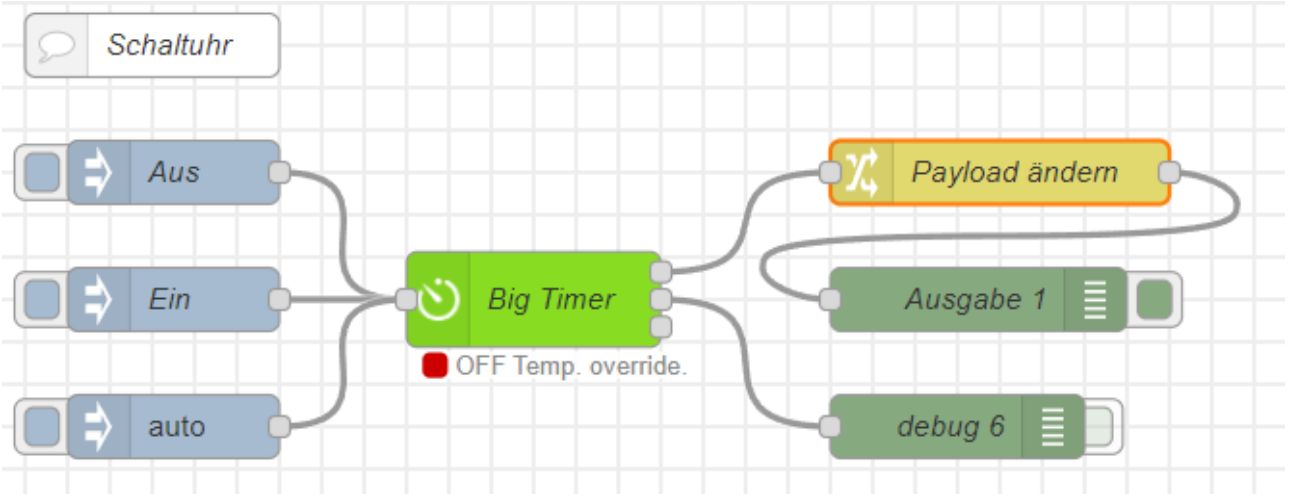
Bei dem change–Node ist auch nicht viel einzustellen. Wir verwenden JSONata, um aus `msg.payload` ein `msg.payload.state` zu machen.

Ausgabe 1 bleibt ungeändert. Der neue Knoten *debug 6* am Ausgang 2 zeigt uns interessante Details über die inneren Zustände des Timers.

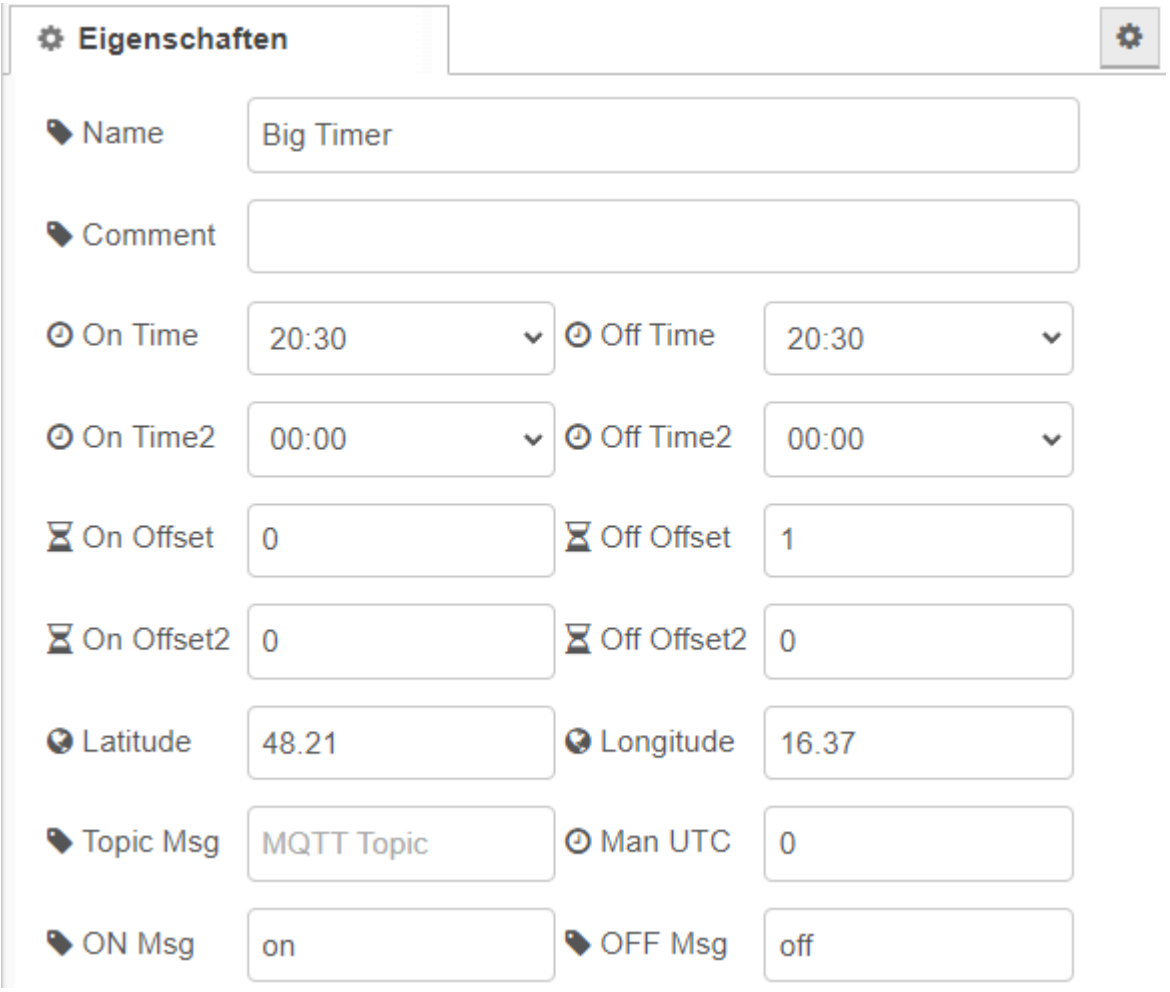


Wird auf *Ein* oder *Aus* geklickt, zeigt der Debugger den Zustand der Lampe:

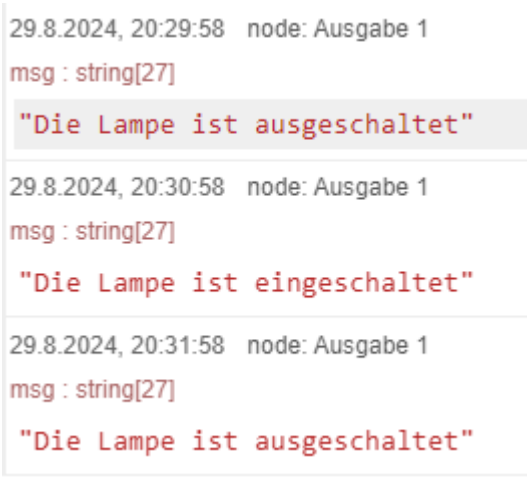
Gleichzeitig zeigt der Big Timer an, dass die Zeitsteuerung durch die Ein- und Aussignale außer Kraft gesetzt (*übersteuert*) worden ist.



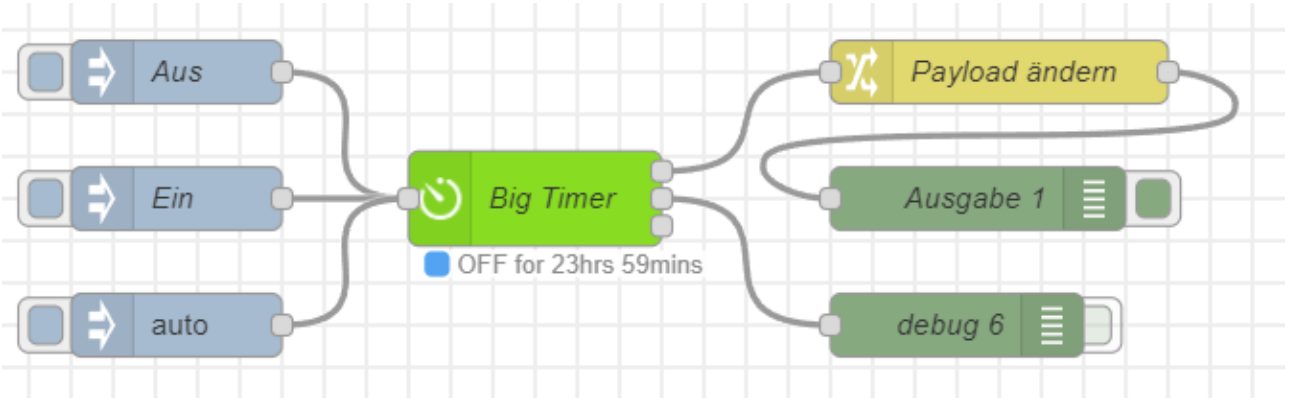
Wir lassen den Timer die Lampe mit der *On Time* um 20:30 einschalten und mit der *Off Time* um 20:31 ausschalten. Die geografischen Koordinaten sind eingetragen Wir könnten daher auch beispielsweise mit dem Sonnenuntergang schalten.



Wie erwartet wird die Lampe um 20:30 eingeschaltet und um 20:31 ausgeschaltet:



Jede Minute wird diese Meldung ausgegeben. Ob der Timerausgang gerade `ON` oder `OFF` ist, ist auch direkt im Arbeitsbereich unterhalb des Timers zu sehen. Die Lampe wird erst wieder am nächsten Tag um 20:30, also in 23 Stunden und 59 Sekunden eingeschaltet.



Das sind die Details, die `debug 6` als Ausgang 2 ausgibt:

```
1  {
2    "payload": 0,
3    "reference": ":on:off:1225",
4    "topic": "status",
5    "state": "OFF Auto",
6    "time": "00hrs 05mins",
7    "name": "Big Timer",
8    "lon": "16.37",
9    "lat": "48.21",
10   "start": 1230,
11   "end": 1231,
12   "start2": 0,
13   "end2": 0,
14   "dusk": 1224,
15   "dawn": 331,
16   "solarNoon": 778,
17   "sunrise": 364,
18   "sunset": 1191,
19   "night": 1310,
20   "nightEnd": 246,
21   "moonrise": 1331,
22   "moonset": 802,
23   "now": 1225,
24   "timer": 0,
25   "duration": 5,
26   "onOverride": -1,
27   "offOverride": -1,
28   "onOffsetOverride": -1,
29   "offOffsetOverride": -1,
30   "stamp": 1724610305964,
31   "extState": "OFF for 00hrs 05mins",
32   "_msgid": "c8e82916bcb72a09"
33
34 }
```

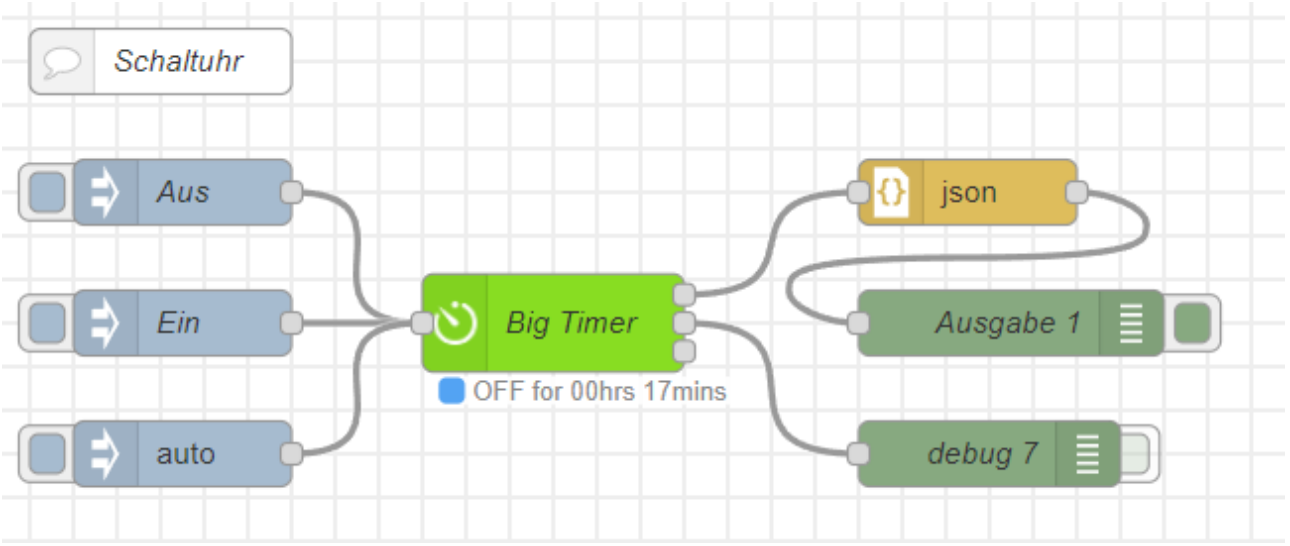
Alle Zahlen von Zeile 10 bis Zeile 23 sind Zeitangaben in Minuten seit Mitternacht.

Eine Alternative zum change-Node

In Node-RED können Aufgaben meist auf verschiedene Art gelöst werden. Der json–Node aus der Palette Parser kann den change–Node ersetzen: er verwandelt einen *String* in ein *JSON-Objekt* oder umgekehrt.

Wir ändern den Eintrag im Big Timer

und auch die Schaltung

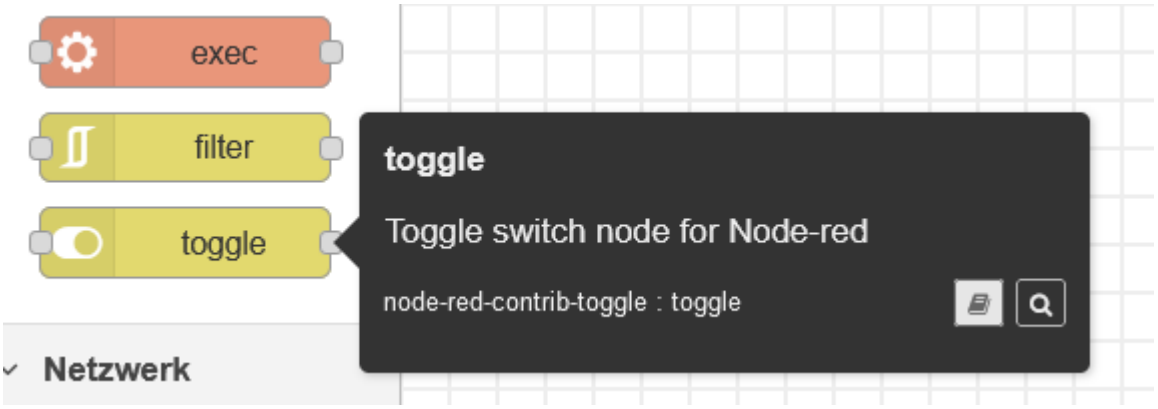


Jetzt sendet die Schaltuhr den *String* `{"state":"on"}` und der json–Node macht daraus ein *JSON–Objekt*.

Stromstoßrelais

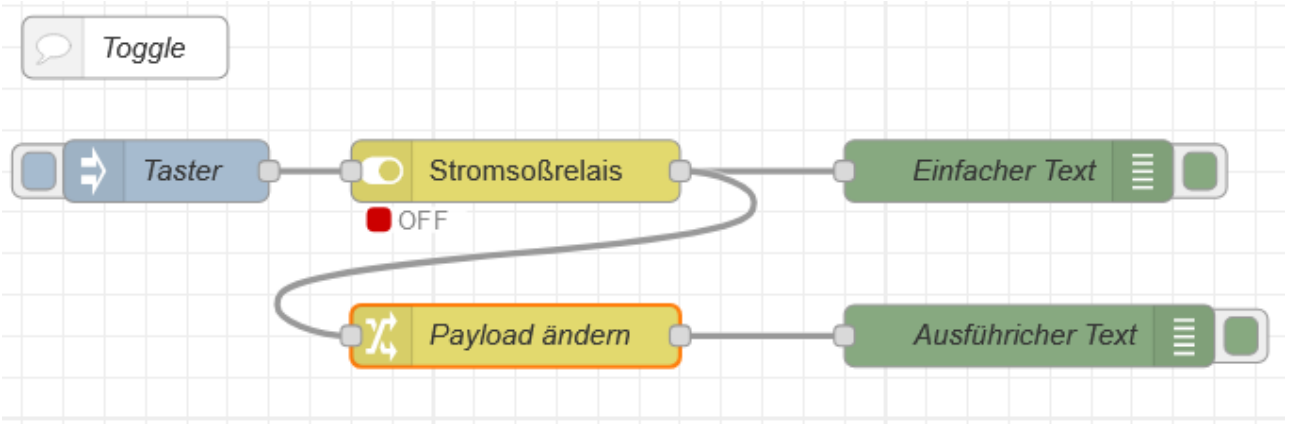
Manche Lichtsteuerungen arbeiten mit Stromstoßrelais, vor allem, wenn das Licht von mehreren Stellen ein- und ausgeschaltet werden soll. Erster Tastendruck - Licht geht an. Zweiter Tastendruck: Licht geht aus.

Natürlich erlaubt Node-RED auch diese Steuerung. Dazu laden wir wie schon vorhin beschrieben die Palette `node-red-contrib-toggle`. *toggle* heißt *umschalten*. Der Node *toggle* erscheint in der Gruppe der Funktion[en].

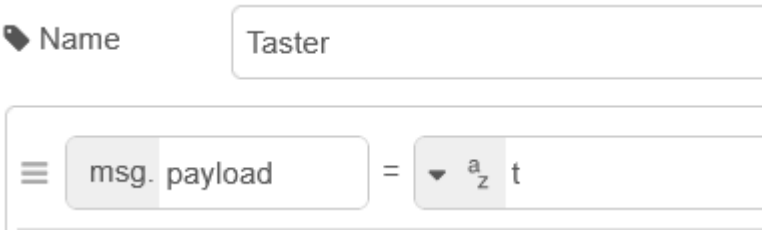


Eine Anleitung zur Funktion von `toggle` erhalten wir mit einem Klick auf `toggle` und einen weiteren Klick auf das erste Ikon. (Was es genau darstellt, weiß ich nicht — vielleicht ein Buch?)

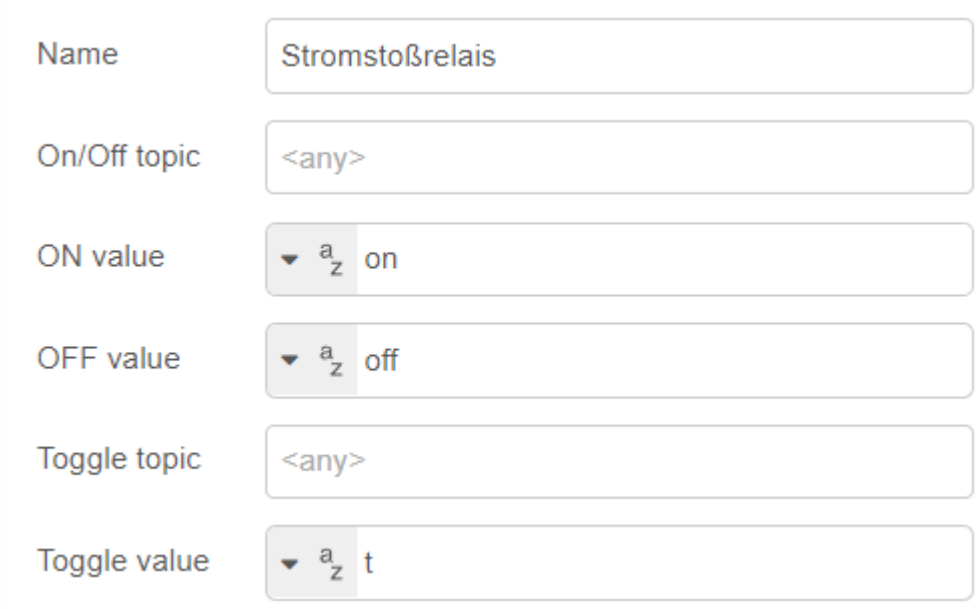
Der folgende Flow enthält gleich zwei Lösungen für die Aufgabe:



Über den Taster wird eine msg.payload mit dem Stringwert *t* eingebracht. Das *t* soll an *toggle* erinnern.



Beim toggle–Node ist mehr einzustellen:

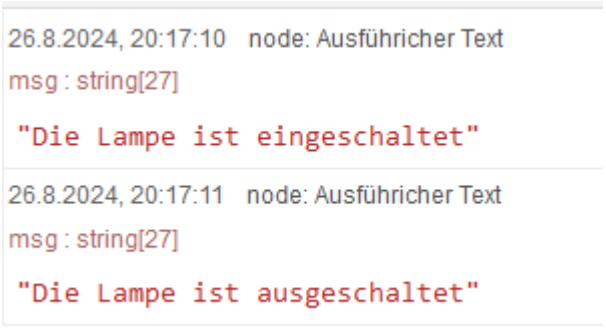


- Der Name wird auf `Stromstoßrelais` geändert
- Topic bleibt unberücksichtigt.
- Wenn der Schalter auf *ON* steht, wird als `msg.payload` der String `on` ausgegeben, beim Zustand *OFF* daher `off`.
- Das passiert nur dann, wenn am Eingang eine `msg.payload` mit dem Stringwert `t` ankommt.

Der erste `debug`–Node mit dem Namen *Einfacher Text* gibt nur den Inhalt von `msg.payload` aus, der zweite debug–Node ist (durch einen Klick auf seine Schaltfläche) deaktiviert:

> **Tip**
>
> Nicht auf das `Übernehmen` vergessen!

Das war aber schon einmal schöner. Hier hilft wie bei dem vorherigen Beispiel ein change–Node und ein debug–Node namens *Ausführlicher Text*. Aktivieren wir den zweiten debug–Node (*Ausführlicher Text*) und deaktivieren wir den ersten (*Einfacher Text*). Die beiden Nodes sind schon im letzten Beispiel vorgekommen.



Jetzt passt die Ausgabe!

> **Note**
>
> Der Zustand des toggle–Node wird unterhalb des Nodes in der Arbeitsfläche mit grünen und roten Kreise und ON und OFF angezeigt.

Zusammenfassung

Die drei Beispiele zeigen, wie herkömmliche Lichtsteuerungen durch Smart Home-Komponenten ersetzt werden können. Jede herkömmliche Lösung hat ihre Vor- und Nachteile. Node-RED kann jedoch noch mehr, nämlich die Vorteile aller drei Varianten in *einer* Steuerung kombinieren. Das ist Thema für einen weiteren PCNEWS-Beitrag.

Arbeiten mit realer Hardware

Es ist zwar recht nett, eine Steuerung zu simulieren. Richtig interessant wird es mit echter Hardware. Dazu müssen die Steuerbefehle über das LAN den Rechner (mit dem Node-RED-Programm) verlassen können und als Funksignale die Hardwarekomponenten erreichen. Dazu dienen MQTT und ZigBee.

MQTT

Die Funktion von MQTT wurde bereits im PCNEWS Heft 181 beschrieben. Für die Experimente ist ein MQTT-Broker notwendig. Häufig wird dafür Mosquitto verwendet. Es gibt davon eine Windows-Version.

Natürlich brauchen wir zusätzlich *echte* Hardware, nicht nur *simulierte*. Node-RED liefert bereits MQTT Signale, die aber meistens noch weiter in die Steuerbefehle der verwendeten Hardware übersetzt werden müssen. Zwei Varianten werden hier vorgestellt: Tasmota und ZigBee. Beide wurden in den PCNEWS bereits erklärt. Geplant ist, später auch Matter dazu zu nehmen.

Tasmota

Tasmota-Komponenten verwenden das WLAN und können ohne zusätzliche Hardware mit dem MQTT-Broker reden. Daher werden sie hier zuerst besprochen. Im PCNEWS Heft 181 sind Details zu Tasmota-Komponenten zu finden.

Aktoren und Sensoren

Ein paar Vorschläge:

- Mit dem ESP32, wie er im PCNEWS Heft 180 beschrieben worden ist, können wir viele Experimente machen.
- Wer gleich *richtige Verbraucher* steuern will, kauft einen Zwischenstecker mit Tasmota-Firmware, zum Beispiel den NOUS A1T, erhältlich unter anderem bei Reichelt oder im Doppelpack bei Amazon.

Damit haben wir Ausgabeelemente (Aktoren). Leider gibt es nur wenig Tasmota-Eingabegeräte, zum Teil auch relativ teuer.

- An den ESP32 lassen sich Taster oder Temperaturfühler anschließen. Er kann sogar Magnetfelder messen. Somit ist der ESP32 auch ein Sensor.

Paletten für Tasmota

In *npm* werden mehrere Paletten für Tasmota angeboten. zum Beispiel *node-red-contrib-tasmota* und *node-red-contrib-tasmotas*. Nach der Installation des MQTT-Brokers und der Tasmota Palette wird der `Ausgabe 1–Node` durch einen passenden Tasmota-Node ersetzt. Eventuell ist noch ein change–Node zur Anpassung des Steuerbefehle dazwischen zu schalten, Fertig! Alle Beispiele funktionieren nun mit realer Hardware

Wie das im Details aussieht, wird im Folgebeitrag beschrieben.

ZigBee

ZigBee baut ein eigenes Funk-Netzwerk auf und braucht daher noch einen ZigBee-Adapter, meist in Form eines USB-Sticks.

Bei ZigBee ist die Auswahl an Sensoren und Aktoren schon wesentlich größer.

> ⚠ **Warning**
>
> *ZigBee* muss in der Spezifikation der Komponente vorkommen. Wenn zusätzlich Tuya dabei steht, ist Vorsicht geboten. Manche Bauteile verwenden zwar das ZigBee-Protokoll, brauchen aber trotzdem den Tuya-Server, Erfahrungsberichte im Internet sind oft eine große Hilfe.

Sensoren

Ein paar Sensoren zur Auswahl:

- Ein Türkontakt eignet sich für Experimente auch gut.

- Ein Bewegungsmelder ist auch ein netter Sensor, nicht allzu teuer und in der Praxis gut zu brauchen. Aber zwischen zwei Schaltvorgängen muss eine Pause von (je nach Einstellung) 30 bis 90 Sekunden liegen. Bei den ersten Experimenten stört das.
- Wer einen Schalter oder Taster verwenden möchte, wie er auch im Haushalt vorkommt, hat eine große Auswahl.

Paletten für ZigBee

Das Programm Zigbee2MQTT kann über die Kommandozeile angesprochen werden und wandelt ZigBee-Kommandos in MQTT um. Für Node-RED gibt es auch mehrere Programmbibliotheken, zum Beispiel *node-red-contrib-zigbee2mqtt*. Alles, was unter Paletten für Tasmota erklärt wurde, gilt auch hier.

Zigbee- und Tasmota-Steuerung

Tasmota

Bei Nodes der Tasmota-Palette wird als Payload zum Einschalten häufig der String `on`, die Zahl `1` oder der Wahrheitswert `true` verwendet, für das Ausschalten `off`, `0` oder `false`. Der change–Node ist daher entsprechend zu ändern.

Beim Verwenden von Big Timer und toggle können wir den change-Node weglassen, weil die beiden Nodes ja schon `on` oder `off` liefern. Die Flows werden dadurch einfacher

Die debug-Nodes sind beim Übergang zu *echten Steuerungen* durch die Tasmota-Nodes zu ersetzen.

ZigBee

Bei Nodes der ZigBee-Palette (Zigbee2MQTT) werden dagegen häufig mit dem Objekt `{"state":"on"}` als Payload eingeschaltet. In den bisherigen Beispielen haben wir diese Form verwendet.

ioBroker und Node-RED

ioBroker ist ein Programmpaket, das verspricht, besonders viele Komponenten unterschiedlicher Hersteller und mit unterschiedlichen Protokollen zu einer kompletten Internet-of-Things-Lösung (IoT) zu verbinden. Besonders bemerkenswert: die schönen Grafiken und Schnittstellen.

Der ioBroker ist modular aufgebaut. Viele Module kommen aus der ständig wachsenden Gemeinschaft der Nutzer. Ablaufsteuerungen des Smart Home können *auch* (aber nicht ausschließlich) mit Node-RED erstellt werden. Dabei wird Node-RED so angepasst, dass es als Modul innerhalb des ioBrokers läuft.

- Vorteil:
  - Eine komplizierte Installation entfällt: einfach den Node-RED-Modul auswählen und die Installation bestätigen.
  - Alle Komponenten, die der ioBroker — auch mit unterschiedlichen Protokollen — kennt, können verwendet werden.
- Nachteile:
  - Die Seite Adapter node-red verspricht interessante Informationen, ist aber hoffnungslos veraltet und enthält viele tote Links.
  - Neue Node-RED-Versionen müssen erst an den ioBroker angepasst werden. Das kann leider lange dauern. Im Broker ist immer noch Node-RED 3.1.11 aktuell, obwohl schon seit Monaten Node-RED 4.0.2 als eigenständige Installation verfügbar ist und auch in diesem Beitrag verwendet wird.

> **Note**
>
> Im ioBroker wird der Node-RED Adapter mit Version 5.2.1 angegeben,Das ist aber nur die ioBroker-interne modifizierte Version. *Verwirrend*!

Zusammenfassung

Der Beitrag zeigt (vor allem in der Langversion) Schritt für Schritt die Installation und die Anwendung von Node-RED für eine einfache Lampensteuerung. Die Lampe wird durch einen Ausgabetext simuliert.

In der Fortsetzung dieses Beitrags

- wird echte Hardware angesteuert
- werden weitere Nodes von Node-RED erklärt
- werden Steuerungen mit JavaScript programmiert
- wird ein Blick auf die Anwendung von Zustandsdiagrammen geworfen
- wird das Smart Home über einen Bot mit Telegram verknüpft

To be continued ... Fortsetzung folgt ...

Zur Langversion

Die Entwicklung aller Beispiele wird in der Langversion sehr detailliert und mit vielen Zusatzinformationen und Bildern erklärt. Ferner können alle vorgestellten Flows auch unter dieser Adresse geladen werden. Lehrreicher ist es trotzdem, die Beispiele von Hand einzugeben.

Link und QR-Code: http://pcnews.at/markdown/n182/index.html oder https://t.ly/ROrK9

Erstellt mit QR Code Generator - kostenfrei, ohne Login, ohne Konto : https://goqr.me/

Short URL erstellt mit https://t.ly/

Copyright


BY NC SA