Martin Weissenböck

Home Assistant: YAML, Automationen und Skripte

Was erwartet Sie in diesem Beitrag?

Was geschieht bei einem *Clubabend*, beim Forum *Smart Home | IoT*? Dieser Beitrag behandelt den Clubabend vom 28. Mai 2025 als Nachlese. Die Grundlagen wurden bereits in den PCNEWS 184 vorgestellt und beim Clubtreffen noch einmal im Detail geübt. Alles ist Schritt-für-Schritt in den Download-Ordnern zu finden, auch eine weitere Einführung zu Home Assistant-Themen. Ferner sind alle Schritte mit Screenshots unterlegt.

Die Themen waren…

- … ein Überblick über YAML
- … das Erstellen von Automationen und die verwendeten Editoren
- … Einträge in der Datei `configuration.yaml`
- … der Einsatz von ChatGPT
- … Skripte und Bausteine

In der Langversion steht anstelle von ☝ viel mehr Text.

YAML

YAML ist die Beschreibungssprache, in der im Home Assistant (HA) Konfigurationen und Anweisungen formuliert werden (können), aber auch veröffentlicht werden. Wer mit HA arbeitet, sollte die Grundlagen von YAML kennen. Der Vergleich mit Konstrukten in JavaScript oder JSON kann manchen Lesern helfen.

YAML ist also keine Programmiersprache, sondern eine für Menschen gut lesbare Form zur seriellen Darstellung von *Daten*, verwendbar in beliebigen Programmiersprachen.

YAML ist ein rekursives Akronym und steht für „YAML Ain't Markup Language“:
"YAML ist keine Auszeichnungssprache", "YAML ist keine Markup-Sprache", ursprünglich „Yet Another Markup Language“.

Die Grundlagen von YAML

Jede YAML-Darstellung besteht ausschließlich nur aus:

- Einzelwerten (in JavaScript: Literale)
- Listen (*list*, in JavaScript, Python: Listen, etwas ungenau: Array)
- Assoziativen Listen (*map*, in JavaScript: Objekte, in C/C++: struct)

Listen und Maps können selbst wieder alle Datenarten enthalten, also auch weitere Listen oder Maps.

Einzelwerte

- Zeichenketten
- Zahlen
- `false`, `true`, `No`, `Yes`

Datenstrukturen

- Listen
- Maps (assoziative Listen)
- Zusammensetzungen wie Listen von Maps, Maps von Maps usw.

Listen

In anderen Programmiersprachen werden Daten durch Klammern (oder Schlüsselworte) organisiert. In YAML werden Einrückungen verwendet – sie sind streng zu beachten! Zwei Leerstellen sind üblich, Tabulatoren statt Leerstellen werden meist als Fehler gekennzeichnet.

Beispiele aus der Wikipedia

> *Ich habe die englischen Ausdrücke übernommen.*

In YAML ist jede Zeile ein eigener Listeneintrag, der Zeilenwechsel ist daher wesentlich. Das "–"-Zeichen kann weggelassen werden,

```
- Berüchtigt (Notorious)
- Casablanca
- Ich kämpfe um dich (Spellbound)
- Solange ein Herz schlägt (Mildred Pierce)
```

oder (gleichwertig)

```
Berüchtigt (Notorious)
Casablanca
Ich kämpfe um dich (Spellbound)
Solange ein Herz schlägt (Mildred Pierce)
```

Listen können auch in *einer* Zeile in eckigen Klammern geschrieben werden. Zeichenketten müssen nicht in Hochkommata eingeschlossen werden.

```
[Haferflocken, Bananen, Nüsse]
```

Wer JSON kennt, kann zum besseren Verständnis ein Umwandlungsprogramm verwenden.

JSON (zum Vergleich):

Ein Zeilenwechsel ist in YAML ein (wichtiges) Sprachelement, in JSON dient er nur der besseren Lesbarkeit.

```
[ "Berüchtigt (Notorious)",
  "Casablanca",
  "Ich kämpfe um dich (Spellbound)",
  "Solange ein Herz schlägt (Mildred Pierce)"]
```

Auch einzeilig möglich:

```
["Haferflocken", "Bananen", "Nüsse"]
```

Map (Assoziative Liste):

> *Da in den Home Assistant–Unterlagen meistens von maps und nicht von assoziativen Listen gesprochen wird, verwende ich die englische Bezeichnung.*

Beispiele aus der Wikipedia

- Jeder Eintrag besteht aus einem Schlüssel (*key*) und einem Wert (*value*).
- Dazwischen steht ein Doppelpunkt. Der Doppelpunkt ist das charakteristische Merkmal einer Map.
- Jeder Schlüssel darf pro Map nur einmal vorkommen.
- Werte können wieder alle besprochenen Datentypen sein, also auch Listen und Maps. Maps von Maps oder Listen kommen häufig vor.
- `---` leitet einen neuen Abschnitt ein.
- Mit `#` beginnt ein Kommentar und endet in derselben Zeile.

```
--- # Block
name: John Smith
age: 33
```

`name` und `age` sind Schlüssel (*keys*), `John Smith` und `33` sind Werte (values).

Gleichwertige Schreibweise, bei großen Maps aber unübersichtlich:

```
--- # Inline
{name: John Smith, age: 33}
```

JSON (zum Vergleich):

```
{ "name": "John Smith", "age": 33 }
```

oder

```
{ "name": "John Smith",
  "age": 33
}
```

Der Zeilenwechsel ist in JSON nicht vorgeschrieben. Wenn aber zusammengehörige Klammern genau untereinander stehen, werden Datenstrukturen und Programme leichter lesbar.

JavaScript (zum Vergleich):

Die Keys müssen nicht zwischen Apostrophen stehen.

```
{ name: "John Smith", age: 33 }
```

Block-Ausdrücke

Ein senkrechter Strich bewirkt, dass Zeilenumbrüche beibehalten werden:

```
--- |
  There was a young lady of Wright
  Who travelled much faster than light.
      She departed one day
      In a relative way
  And returned in the previous night.
```

Listen von Maps

> **Tipp**
>
> Dieser Abschnitt ist besonders wichtig – bitte aufmerksam lesen.

Beispiele aus der Wikipedia

- Das "–" leitet ein neues Listenelement ein.
- Untereinander stehende, gleich eingerückte Listenelemente bilden gemeinsam eine assoziative Liste.

```
- {name: John Smith, age: 33}
- name: Mary Smith
  age: 27
```

Entspricht in JSON:

```
[ {"name": "John Smith", "age": 33},
  {"name": "Mary Smith", "age": 27}
]
```

oder

```
[ { "name": "John Smith",
     "age": 33
  },
  { "name": "Mary Smith",
    "age": 27
  }
]
```

Map von Listen

Beispiel aus der Wikipedia

```
men: [John Smith, Bill Jones]
women:
  - Mary Smith
  - Susan Williams
```

Entspricht in JSON:

```
{ "men":   [ "John Smith",
             "Bill Jones"
           ],
  "women": [ "Mary Smith",
             "Susan Williams"
           ]
}
```

Oder:

```
{ "men":   [ "John Smith", "Bill Jones" ],
  "women": [ "Mary Smith", "Susan Williams" ]
}
```

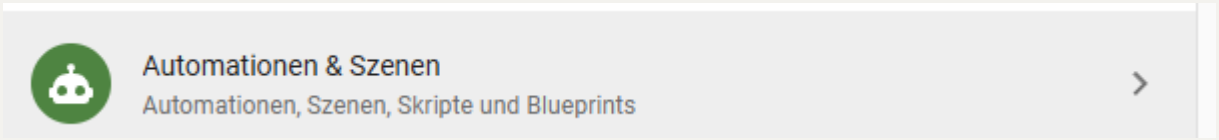
Mehr zu YAML

Weiterentwicklungen von YAML erlauben, den YAML-Code selbst durch Anweisungen zu verändern. Dadurch können beispielsweise YAML-Codestücke mehrfach verwendet werden, vergleichbar mit Makros einer Programmiersprache.

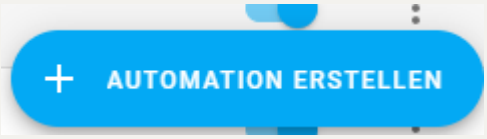
Siehe Advanced YAML syntax cheatsheet.

Automationen

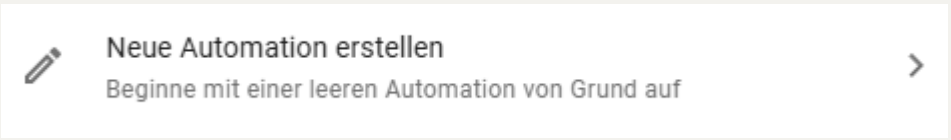
Sie sind *das* zentrale Steuerelement in Home Assistant. Zum Erstellen einer Automation in der Menüleiste links `Einstellungen` und dann `Automationen & Szenen` wählen.



Eine neue Automation wird mit dem Button rechts unten erstellt:



Weiter bei



Automationen können im *visuellen Editor* oder mit *YAML* bearbeitet werden.

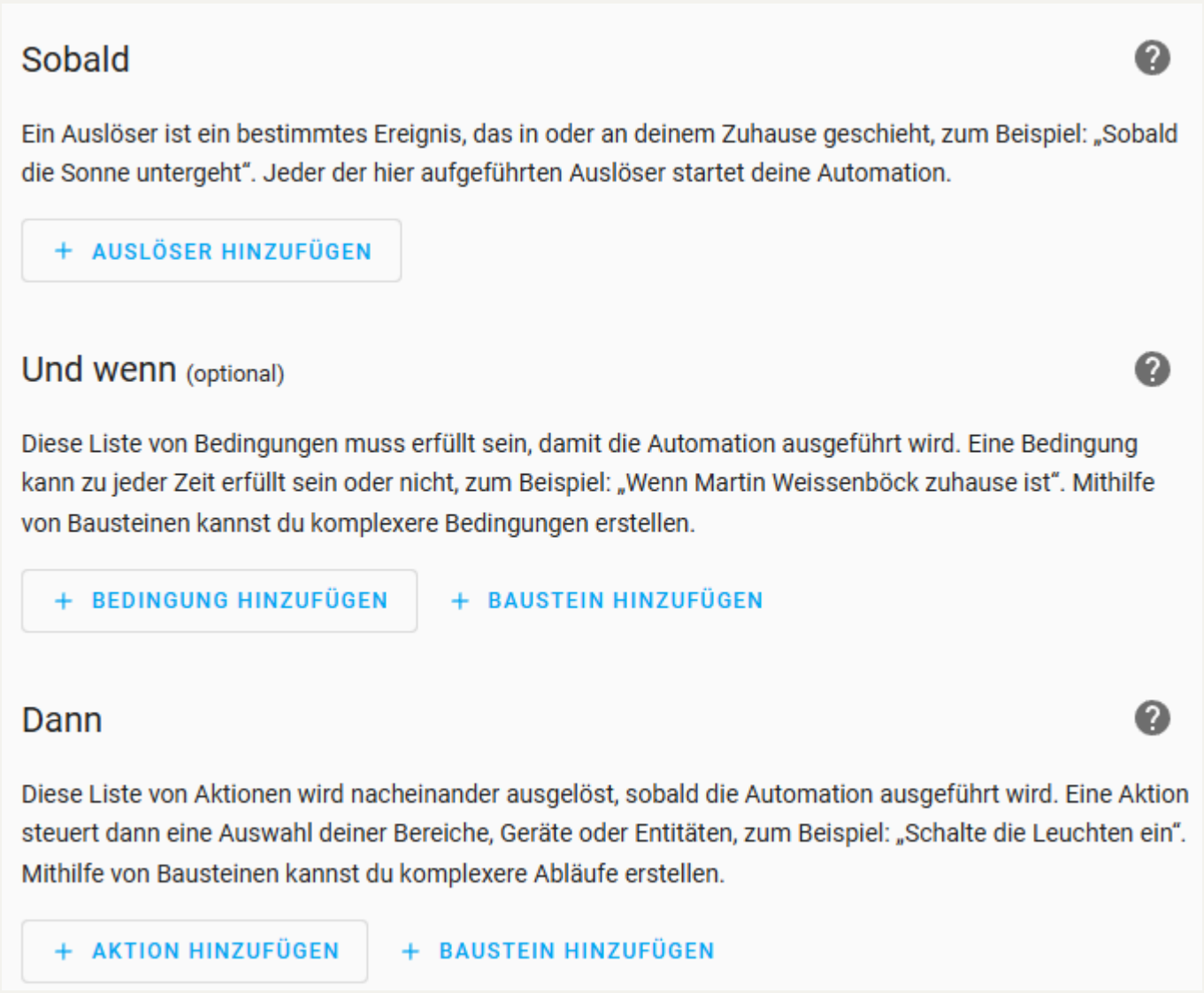
Visueller Editor

Beim Entwickeln von Programmen sind oft drei Abschnitt zu erkennen:
Eingabe der Daten — Verarbeitung — Ausgabe.

Eine Automation besteht im einfachen Fall ebenso aus drei Blöcken:

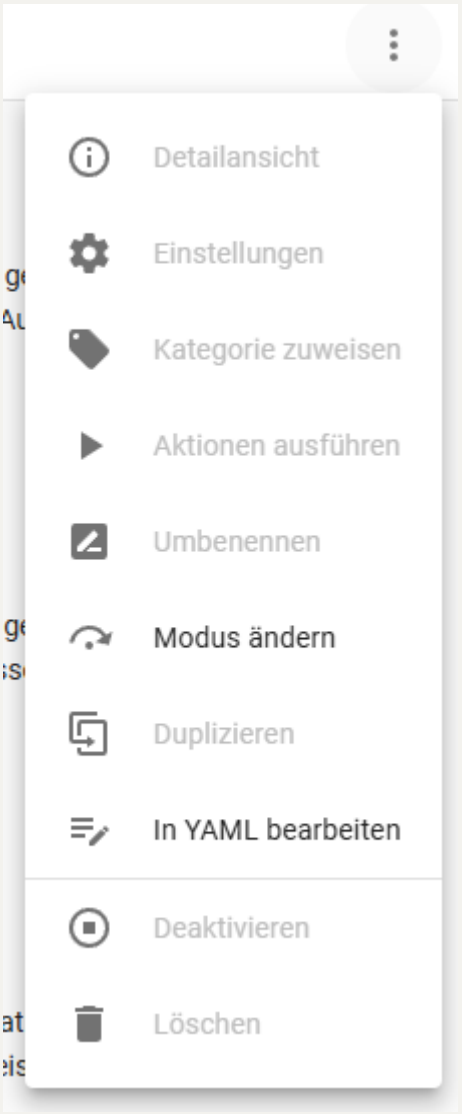
- Die Eingabe der Daten entspricht im HA dem *Auslöser* der Automation.
  Stichworte: `trigger` oder `Sobald`
- In der Verarbeitung werden Bedingungen geprüft.
  Stichworte: `conditions` oder `Und wenn`
- Die Ausgabe wird mit `actions` oder `Dann` eingeleitet

Im Visuellen Editor sieht das so aus:

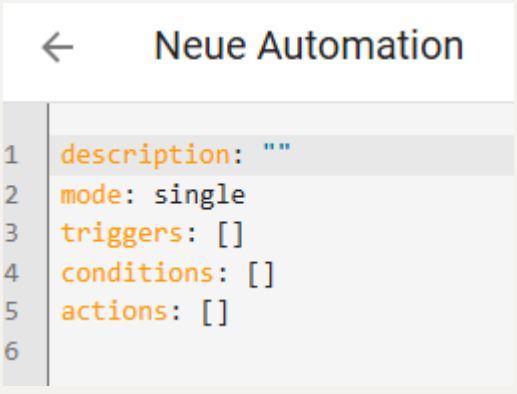


YAML-Editor

Zwischen dem visuellen Editor und dem YAML–Editor kann jederzeit umgeschaltet werden. Dazu wird rechts oben das Drei-Punkt-Menü ⋮ aufgerufen.



In den Zeilen 3 bis 5 sind wieder die drei Blöcke zu finden. In Zeile 1 kann eine Beschreibung eingetragen werden. `single` in Zeile 2 bedeutet, dass das neuerliche Aufrufen einer gerade laufenden Automation ignoriert wird.



Wie sind diese fünf Zeilen syntaktisch aufgebaut? Sie bilden eine Liste von Maps. Die jeweiligen Keys stehen vor den Doppelpunkten und sind in oranger Schrift hervorgehoben.

Vergleiche

KLASSISCHES PROGRAMM	YAML	TEXTEDITOR
Eingabe	`triggers: []`	Sobald
Verarbeitung, Bedingungen	`conditions: []`	Und wenn
Ausgabe	`actions: []`	Dann

Zeile 1 und 2 haben als Wert je eine Zeichenkette, Zeilen 3 bis 5 je eine leere Liste.

- Zeile 1: Description = Beschreibung
- Zeile 2: Die Automation wird (zu *einem* Zeitpunkt) nur *einfach* ausgeführt.

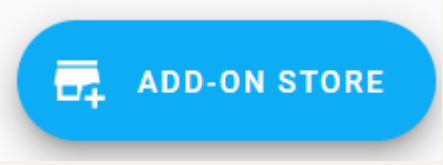
```
1 description: "Erster YAML-Versuch"`
2 mode: "single"
```

File editor

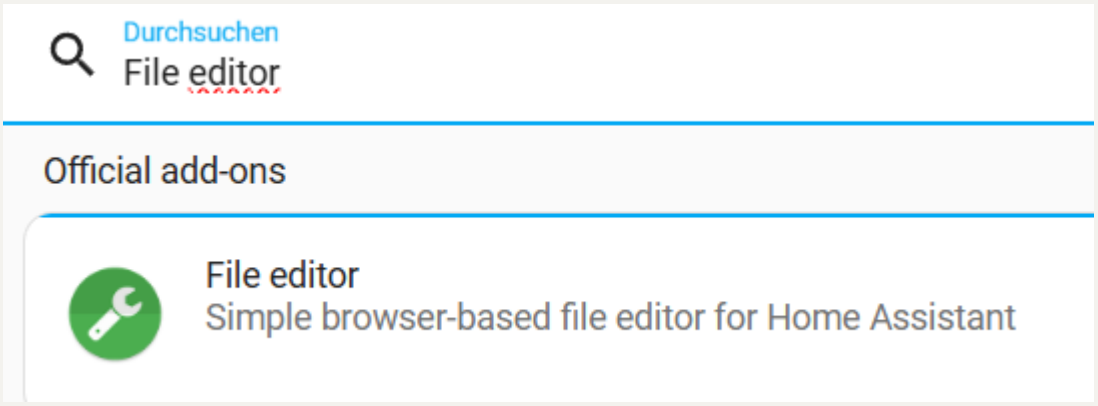
> **Anmerkung**
>
> Dieser Abschnitt kann auch vorerst übersprungen werden.

HA hat einen eigenen *File editor*, also ein Programm, mit dem einzelne Dateien ausgesucht und angesehen oder verändert werden können. Der *File editor* ist ein Zusatzprogramm (Add on). und wird so installiert:

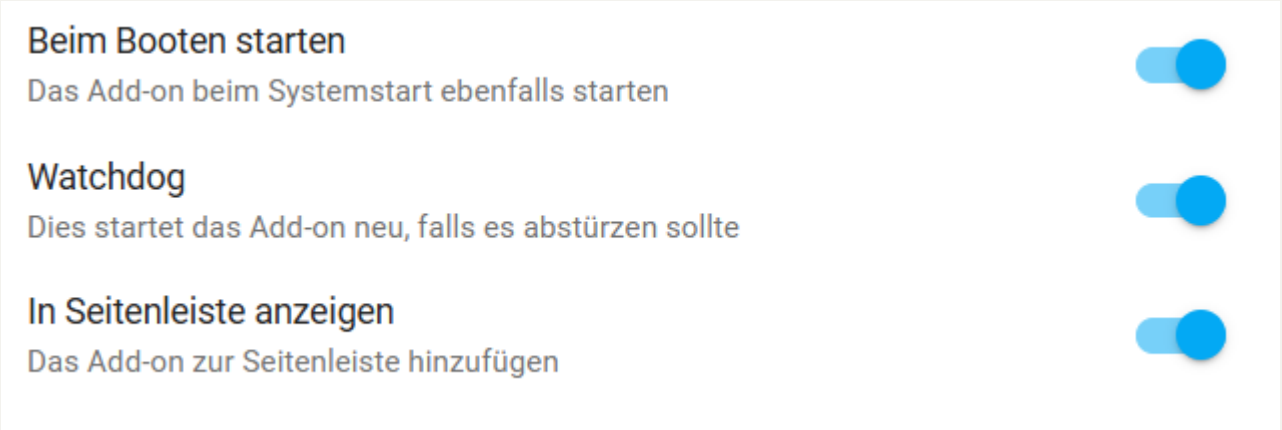
Über `Einstellungen` und `Add-ons` (rechts unten) zum



gehen,



suchen und zum Installieren anklicken. Alle angebotenen Optionen sind nützlich:



`STARTEN` nicht vergessen.

HA arbeitet sehr viel mit Konfigurationsdateien, in denen Geräte, Abläufe, Systemeinstellungen und mehr beschrieben und gespeichert werden. Die Beschreibungssprache ist YAML. *Die Mutter aller Konfigurationen* ist die Datei `configuration.yaml`. Wenn wir *eine einzige* neue Automation erstellen (was in der Praxis kaum vorkommen), könnten wir diese dort unter `automation` eintragen:

```
1 automation:
2   triggers: []
3   conditions: []
4   actions: []
```

Diese Automation (beginnend mit Zeile 1) ist eine Map mit dem Schlüssel `automation`, deren Wert wieder eine Map mit drei Einträgen (Zeilen 2 bis 4) ist. Die Werte der drei Einträge sind selbst leere Listen.

Wir hätten auch diesen Eintrag auch als Map mit einer Liste von Maps schreiben können:

```yaml
automation:
  - triggers: []
    conditions: []
    actions: []
```

- Hier beginnt in Zeile 1 wieder eine Map mit dem Schlüssel `automation`.
- Der zugehörige Wert ist eine Liste mit *einem Element*.
- Dieses Element ist eine Map mit drei Einträgen.
- Die Werte der drei Einträge sind leere Listen.

> **Tipp**
>
> Solche YAML-Konstrukte sind im HA immer wieder zu finden. Es ist sehr wichtig, zu verstehen, was da gemeint ist!

Die zweite Schreibweise erlaubt es, eine HA-Steuerung mit mehr als einer Automation zu schreiben — was ja wohl die Regel ist! Mehrere Automationen werden in die Map `automation:` als *Liste von Maps* eingetragen und üblicherweise durch id-Einträge von einander unterschieden:

```yaml
automation:
  - id: 1
    triggers: []
    conditions: []
    actions: []
  - id: 2
    triggers: []
    conditions: []
    actions: []
```

Die ids stehen hier als Beispiele. Noch einmal zum Verständnis:

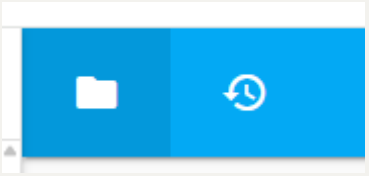
- Zeile 1 bis 9 ist eine Map mit dem Schlüssel `automation` und dem Wert in den Zeilen 2 bis 9
- Die Zeilen 2 bis 9 sind eine Liste mit zwei Einträgen, nämlich Zeilen 2 bis 5 und Zeilen 6 bis 9.
- Die Zeilen 2 bis 5 enthalten eine Map mit vier Einträgen, ebenso die Zeilen 6 bis 9.

> **Wichtig**

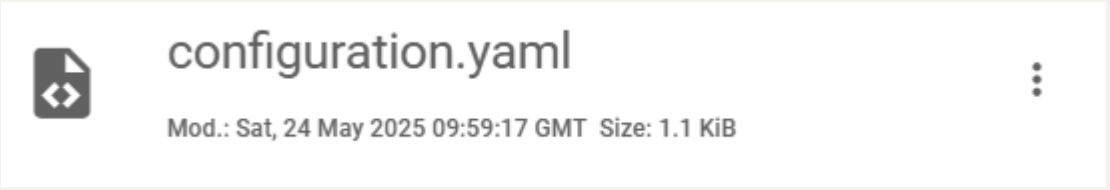
> Zu Erinnerung: YAML verwendet keine Klammern zur Darstellung der Datenstrukturen. Daher sind die Einrückungen sehr wichtig. Üblich sind Einrückungen um genau zwei Stellen.

Allerdings würde bei vielen Automationen die `configuration.yaml`–Datei rasch unübersichtlich werden. Daher verwendet HA die Möglichkeit, eine YAML-Datei auf mehrere Dateien aufzuteilen und diese Teile mit `!include` in die gemeinsame Datei (hier: `configuration.yaml`) zusammenzuführen.

Die Datei `configuration.yaml` zeigt, wie das gelöst wird. Über `File editor` in der linken Seitenleiste und dann über das Verzeichnis-Symbol links oben



kommen wir zu einer Liste der Verzeichnisse und Dateien. Dort ist die zentrale Konfigurationsdatei `configuration.yaml` zu finden.



Anklicken zeigt im Editor eine große Datei. Das sind die ersten 10 Zeilen, die auch nicht verändert werden dürfen:

```yaml
# Loads default set of integrations. Do not remove.
default_config:

# Load frontend themes from the themes folder
frontend:
  themes: !include_dir_merge_named themes

automation: !include automations.yaml
script: !include scripts.yaml
scene: !include scenes.yaml
```

Alle Automationen werden hier in eine eigene Datei, `automations.yaml`, ausgelagert:

```yaml
automation: !include automations.yaml
```

Das vorherige Beispiel sieht nun in `automations.yaml` so aus:

```yaml
- id: 1
  triggers: []
  conditions: []
  actions: []
- id: 2
  triggers: []
  conditions: []
  actions: []
```

Was hat sich geändert?

- Der Schlüssel `automation:` fällt weg, da die Datei `automations.yaml` ja selbst nach `automation:` mit `!include` in die Datei `configurations.yaml` eingefügt wird.
- Alle Einträge in `automations.yaml` sind Listen von Maps und beginnen in Spalte 1.

> **Wichtig**
>
> Wenn auf einer Webseite vom Home Assistant und oft auch in anderen Beiträgen eine Automation als Beispiel gezeigt wird, beginnt sie mit
>
> ```yaml
> automation:
>   - id: 1
>     ...
> ```
>
> Sobald aber die Beispiel-Automation in die Datei `automations.yaml` eingefügt wird, verschwindet `automation:` und es wird
>
> ```yaml
> - id: 1
>   ...
> ```
>
> daraus. Die Zeilen werden auch um zwei Stellen nach links *ausgerückt*.
>
> Dasselbe gilt auch bei den anderen include-Dateien, hier für `themes:`, `script:`, `scene:`.

Der visuelle Editor und YAML

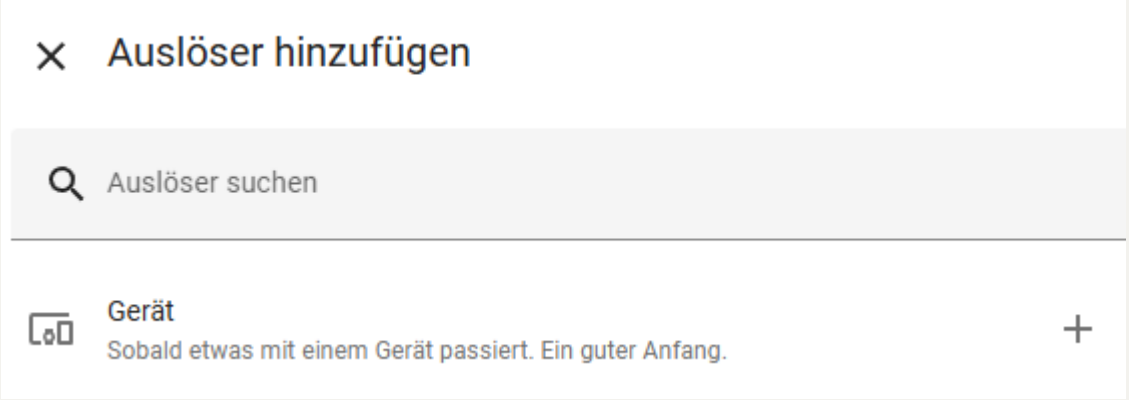
Wir erzeugen eine einfache Automation und schauen, wie dieselbe in YAML dargestellt wird.

Die Aufgabe:

- T1 ist ein Taster, L1 ist eine Lampe.
- T1 und L1 sind bereits angelernt.
- Wenn der Taster T1 einmal gedrückt wird, wird die Lampe L1 umgeschaltet.

Erstellen der Automation:

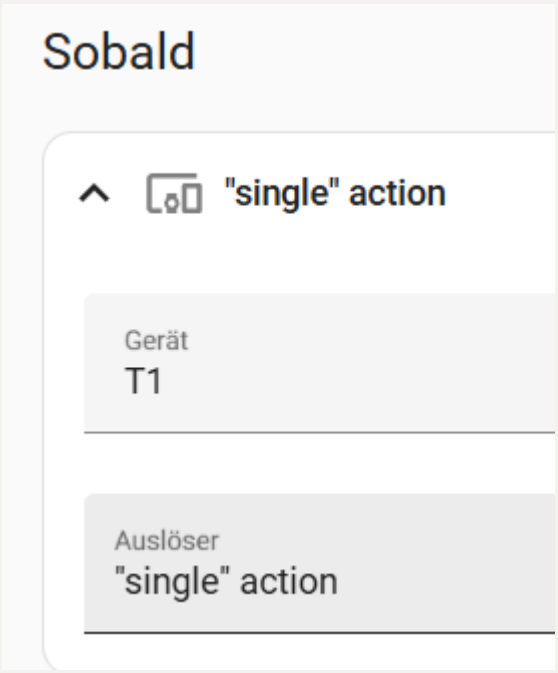
`Einstellungen` → `Automationen & Szenen` → `+ AUTOMATION ERSTELLEN`
→ `Neue Automation erstellen` → `+AUSLÖSER HINZUFÜGEN`



Gerät und Auslöser auswählen:

- Obwohl *Gerät* erst an zweiter Stelle steht, *zuerst* `Gerät` auswählen: In unserem Fall `T1`.
- Dann erst `Auslöser suchen`. Wir wählen `"single" action`. Gemeint ist, dass die Taste nur einmal gedrückt wird.

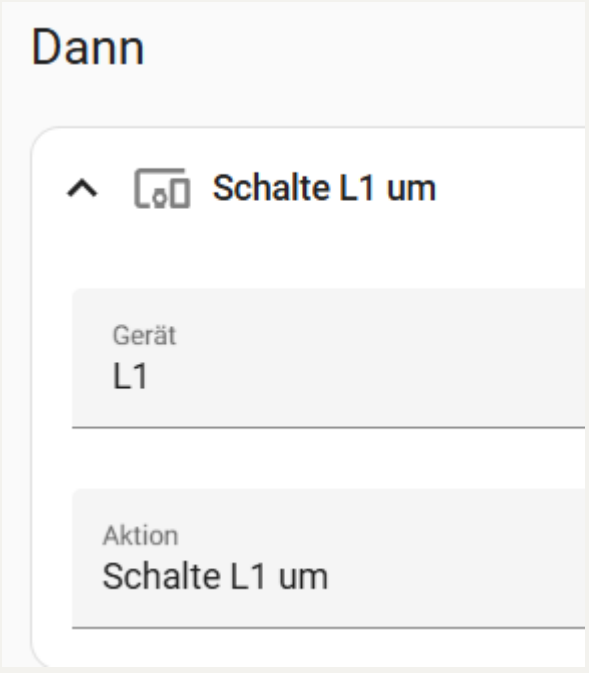
```
1
```



Bei *Und wenn* wird keine Bedingung eingetragen, weiter bei *Dann*:

`+ AKTION HINZUFÜGEN` und alles so einstellen:

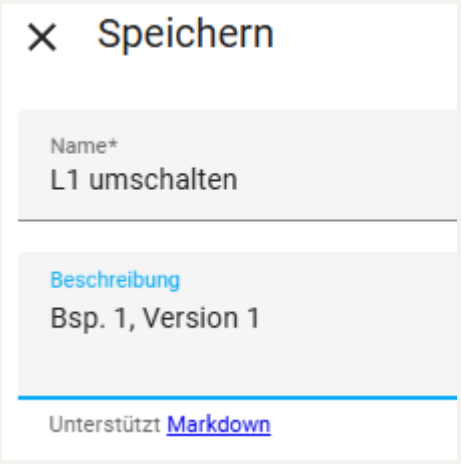
- Wieder wird zuerst das `Gerät` gewählt: `L1`
- Dann erst über `Aktion` auf `Schalte L1 um` einstellen.



Nicht vergessen: alles speichern (rechts unten, blau).

`SPEICHERN`

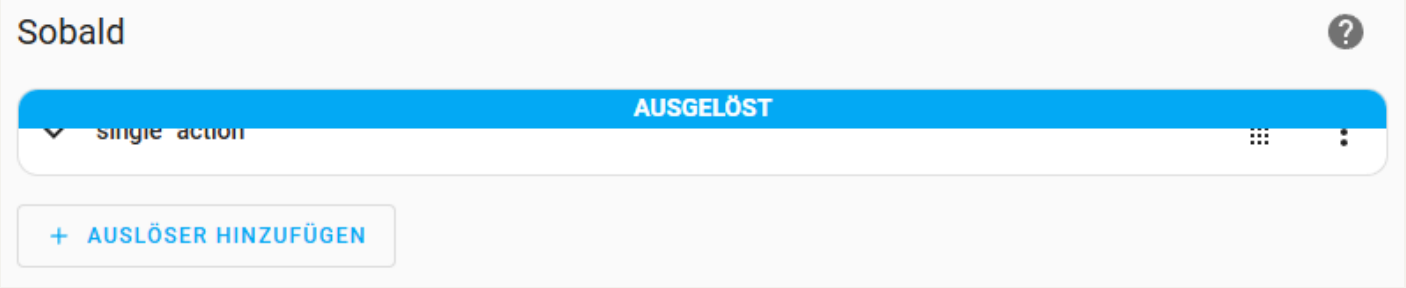
Die neue Automation bekommt beim Speichern einen Namen und eine Beschreibung:



> **Anmerkung**
>
> Mit Markdown kann die Beschreibung grafisch verbessert werden, zum Beispiel durch Fett- oder Kursivschrift.

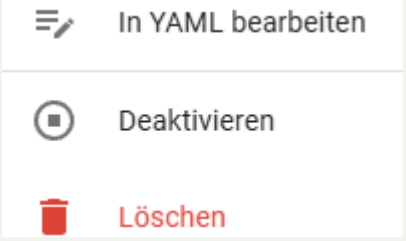
Noch einmal `SPEICHERN` und dann ausprobieren: jedes Mal, wenn der Taster T1 gedrückt wird, schaltet die Lampe L1 um.

Außerdem erscheint im Visuellen Editor das Wort `AUSGELÖST`:



Die YAML-Version

Über das 3-Punkte-Menü ⋮ der neuen Automation (rechts neben `L1 umschalten`) den Eintrag `In YAML bearbeiten` wählen:



Automation zum Umschalten einer Lampe in YAML:

```yaml
alias: L1 umschalten
description: Bsp. 1, Version 1
triggers:
  - domain: mqtt
    device_id: 2a73abd2b10d069a70ddeb7b4e000a23
    type: action
    subtype: single
    trigger: device
conditions: []
actions:
  - type: toggle
    device_id: a1213e273962d1700c88b6c989213151
    entity_id: 73a8e2da66a719a3560fac4186c87958
    domain: light
mode: single
```

Erklärung

- Zeile 1: Der gewähle Name der Automation erscheint unter Alias. Die Automation hat intern eine (lange) Nummer bekommen, aber für den menschlichen Leser Namen besser. Ein Alias kann immer eingesetzt werden, wenn ein kürzerer oder treffenderer Name gewünscht wird.
- Zeile 2: Die Beschreibung landet unter – Überraschung! – description
- Dann kommen die drei Teile
    - triggers: Plural – es können ja auch mehrere sein, Zeile 3 bis 8
    - conditions: Zeile 9, eine leere Liste
    - actions: was zu tun ist, Zeile 10 bis 14.
- mode: das Verhalten beim mehrfachen Start der Automation.

triggers: Die Zeilen 3 bis 8 können so gelesen werden: Über die Komponente mqtt wird das Gerät mit der internen Nummer 2a73abd2b10d069a70ddeb7b4e000a23 abgefragt. Wird auf diesem Gerät die Aktion "Taste einmal drücken" ausgeführt, löst das die Automation aus.

actions: Die Zeilen 11 bis 15 bedeuten: das Gerät mit der internen Nummer a1213e273962d1700c88b6c989213151 ist eine Lampe (light) und hat eine Schaltfunktion mit der internen Nummer 73a8e2da66a719a3560fac4186c87958, mit der das Gerät umgeschaltet wird.

Wo sind die Bezeichnungen T1 und L1 hingekommen? Die YAML-Darstellung ist sehr nahe an den internen Steuerbefehlen und die arbeiten nun einmal mit Nummern.

ChatGPT

Es gibt viele Varianten, wie ein Problem gelöst werden kann. Kann ChatGPT einen Vorschlag machen?

> *Ich habe eine Lichtsteuerung (siehe YAML) mit Home Assistant erstellt und möchte sie so umstellen, dass beim Drücken des Tasters T1 das Licht in 2 Sekunden die volle Helligkeit erreicht und beim zweiten Tastendruck die Lampe L1 nach 6 Sekunden dunkel wird. Ohne Verwendung einer Hilfsvariablen.*

Die Antwort von ChatGPT

Danke! Du hast eine einfache **Toggle-Automation** erstellt, die beim Drücken des Tasters `T1` das Licht `L1` umschaltet. Jetzt möchtest du sie erweitern:

🧭 Ziel

Beim **Tastendruck**:

- Wenn **L1 aus** → Licht **in 2 Sekunden hochdimmen**.
- Wenn **L1 an** → Licht **in 6 Sekunden runterdimmen** und **dann ausschalten**.

Ohne Hilfsvariable wie `input_boolean`.

✅ Angepasste Version deiner Automation

Hier ist dein YAML, erweitert um die gewünschte Logik:

```yaml
alias: L1 umschalten mit sanfter Dimmung
description: Schaltet Licht L1 mit sanftem An- und Ausschalten
trigger:
  - platform: device
    device_id: 2a73abd2b10d069a70ddeb7b4e000a23
    domain: mqtt
    type: action
    subtype: single
```

```yaml
condition: []
action:
  - choose:
      - conditions:
          - condition: state
            entity_id: light.licht_l1
            state: 'off'
        sequence:
          - service: light.turn_on
            data:
              entity_id: light.licht_l1
              brightness: 255
              transition: 2
      - conditions:
          - condition: state
            entity_id: light.licht_l1
            state: 'on'
        sequence:
          - service: light.turn_on
            data:
              entity_id: light.licht_l1
              brightness: 1
              transition: 6
          - delay: "00:00:06"
          - service: light.turn_off
            data:
              entity_id: light.licht_l1
mode: single
```

🛠️ Was du ggf. anpassen musst:

DEIN WERT	MÖGLICHER KLARTEXT	ANPASSEN IN AUTOMATION
`light.licht_l1`	Die Entity-ID deiner Lampe `L1`	✅ **Wichtig**
`device_id` des Triggers	Passt zu deinem Taster `T1`	✅ OK
`subtype: single`	Passt zu einfachem Tastendruck	✅ OK

Falls du dir unsicher bist, wie deine **Entity-ID** heißt:
→ Home Assistant UI → *Einstellungen* > *Geräte & Dienste* > *Entitäten* → suche nach `L1`.

Wenn du möchtest, kann ich auch die **automations.yaml** direkt umschreiben, wenn du mir die Originaldatei oder deinen Editor-Kontext schickst.

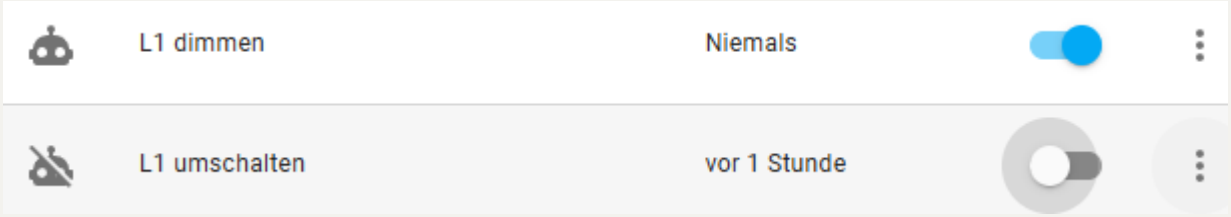
Den Vorschlag umsetzen

- Eine neue Automation erstellen:
- Auf den YAML-Editor umschalten
- Mit dem von ChatGPT gelieferten Code alles überschreiben

Ich habe dann noch die Zeile 1 und 2 geändert:

```
1 alias: L1 dimmen
2 description: Bsp. 2 mit Dimmen
```

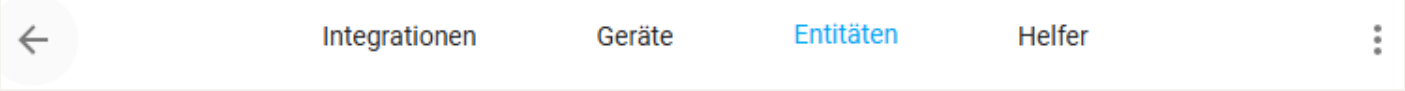
Die erste Lösung (*Bsp. 1 - L1 umschalten*) ausschalten:



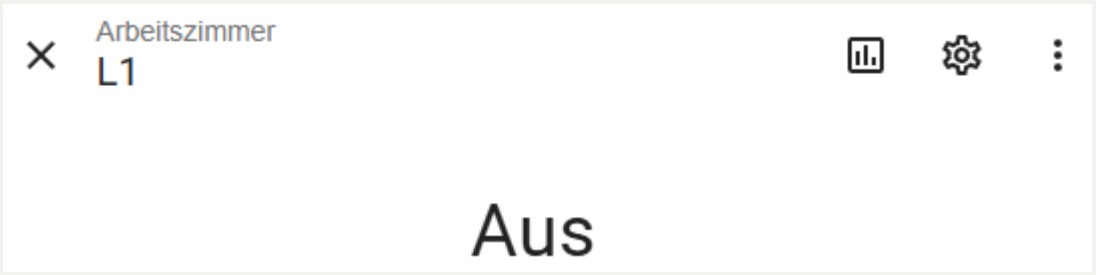
Ausprobieren – und es funktioniert *nicht*.

ChatGPT hat darauf hingewiesen, dass die Entitäten der Lampe zu überprüfen sind. Entitäten sind hexadezimale eindeutige Nummern für jede Funktion jedes einzelnen Geräts (und auch von Programmen).

Den nächsten Hinweis hat ChatGPT auch gegeben. Unter `Einstellungen` → `Geräte & Dienste` sind *Integrationen*, *Geräte*, *Entitäten* und *Helfer* zu finden.



Und dann unter L1



und unter dem Zahnradsymbol die gesuchte Entität:

Jeder Eintrag `light.licht_l1` ist durch `light.l1` zu ersetzen. Jetzt klappt es!

Das ist die endgültige Automation:

```yaml
alias: L1 dimmen
description: Bsp. 2 mit Dimmen
triggers:
  - device_id: 2a73abd2b10d069a70ddeb7b4e000a23
    domain: mqtt
    type: action
    subtype: single
    trigger: device
conditions: []
actions:
  - choose:
      - conditions:
          - condition: state
            entity_id: light.l1
            state: "off"
        sequence:
          - data:
```

```
              entity_id: light.l1
              brightness: 255
              transition: 2
            action: light.turn_on
      - conditions:
          - condition: state
            entity_id: light.l1
            state: "on"
        sequence:
          - data:
              entity_id: light.l1
              brightness: 1
              transition: 6
            action: light.turn_on
          - delay: "00:00:06"
          - data:
              entity_id: light.l1
            action: light.turn_off
mode: single

```

Ergebnis

ChatGPT liefert dazu ein YAML-Programm, das alle gewünschten Funktionen erfüllt. Diese Anfrage an ChatGPT war recht erfolgreich, obwohl noch kleine Änderungen notwendig waren! Allerdings gibt es noch andere Aufgaben, in denen ChatGPT total versagt, zum Beispiel beim Erstellen eines ziemlich einfachen Schaltplans. Aber ChatGPT ist ja noch nicht einmal 3 Jahre alt – das kommt schon noch!

Skripte

Ein Skript kann mit einer Funktion – einem Unterprogramm – in einer höheren Programmiersprache verglichen werden.

Wir haben vorhin eine Automation erstellt, die eine bestimmte Lampe dimmt. Wäre doch nett, diese Idee auf verschiedene Lampen anwenden zu können, ohne jedes Mal eine komplette Automation schreiben zu müssen. Das kann mit einem Skript erreicht werden.

Probieren wir es gemeinsam aus.

> **Anmerkung**

> *Skript* oder *script*?
>
> Im Deutschen (und auch in den Home Assistant-Menüs) *Skript*, im Englischen (und daher auch im Programmcode) *script*.

Das letzte Beispiel weiter verwenden

Zuerst holen wir den YAML-Code des letzten Beispiels in die Zwischenablage:

`Einstellungen` → `Automationen & Szenen` → `L1 dimmen`

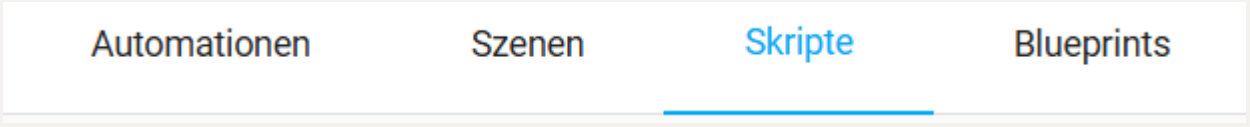
In der Überschriftszeile `Lampe L1` ganz rechts das Drei-Punkt-Menü `⋮` aufrufen. Dann `In YAML bearbeiten` .

Das komplette Programm erscheint im YAML-Editor. Links unten steht `IN DIE ZWISCHENABLAGE KOPIEREN`. Anklicken, fertig!

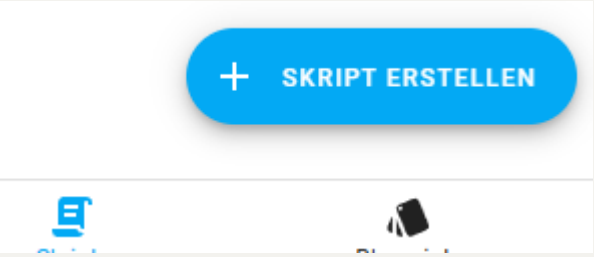
> **Anmerkung**
>
> Das Programm hat etwa 37 Zeilen im Editor. Falls nur kurze Programmstücke zu sehen sind, war es das falsche Drei-Punkt-Menü.

Erstellen des ersten Skripts

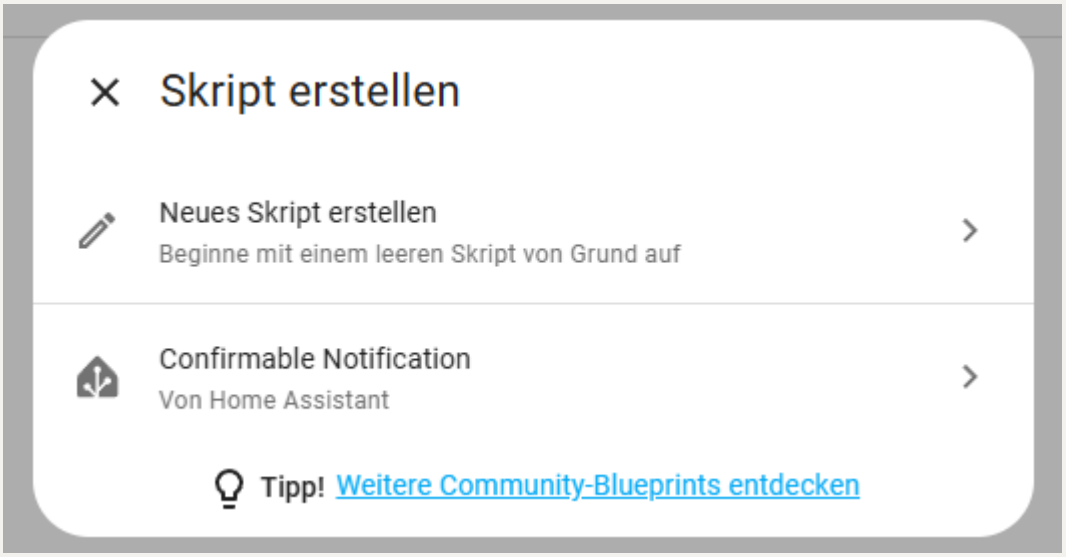
Zum Erstellen beginnen wir wieder mit dem Menüpunkt `Einstellungen` in der linken Seitenleiste. Dann kommt `Automation & Szenen` und dann in der Kopfzeile auf `Skripte` umschalten:



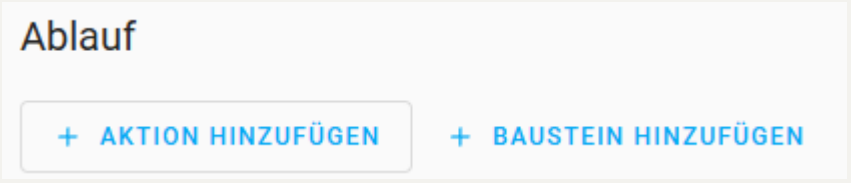
Dann rechts unten `+ SKRIPT ERSTELLEN` wählen.



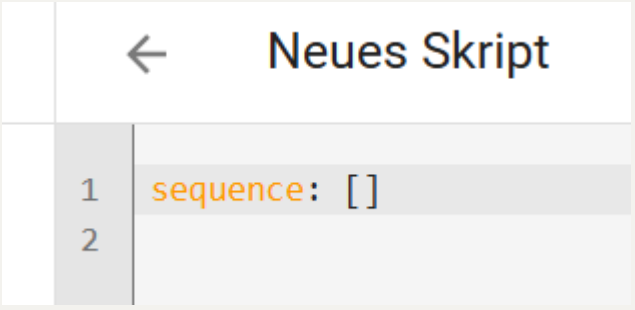
Bei der Auswahl



auf `Neues Skript erstellen` klicken. Es erscheint:



Wir starten wieder über das Drei-Punkt-Menü und `In YAML bearbeiten`. Das neue Skript ist noch recht kurz und erwartet eine `sequence`.



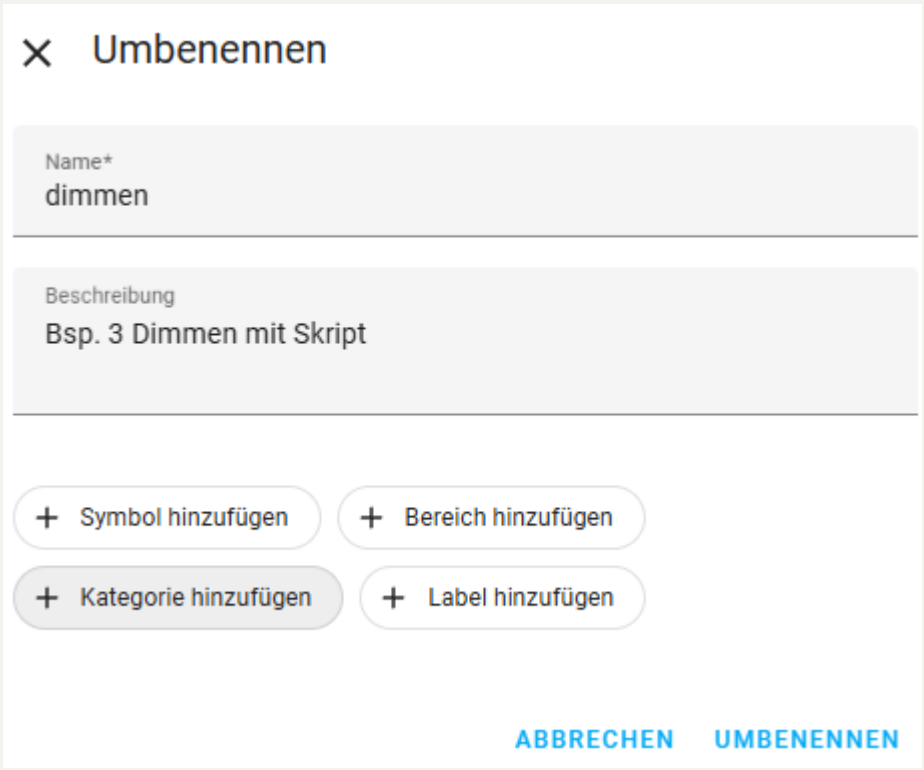
Wir *überschreiben* alles ab Zeile 1 mit dem Code aus der Zwischenablage. Die `triggers`-Liste und die `conditions`-Liste werden entfernt.
`alias` und `description` kommen in die Zeilen 1 und 2 und werden geändert.

Nun sieht das Skript so aus:

```yaml
alias: dimmen
description: Bsp. 3 Dimmen mit Skript
sequence:
  - choose:
      - conditions:
          - condition: state
            entity_id: light.l1
            state: "off"
        sequence:
          - data:
              entity_id: light.l1
              brightness: 255
              transition: 2
            action: light.turn_on
      - conditions:
          - condition: state
            entity_id: light.l1
            state: "on"
        sequence:
          - data:
              entity_id: light.l1
              brightness: 1
              transition: 6
            action: light.turn_on
          - delay: "00:00:06"
          - data:
              entity_id: light.l1
            action: light.turn_off
mode: single

```

`SKRIPT SPEICHERN` . Name und Beschreibung stimmen schon:

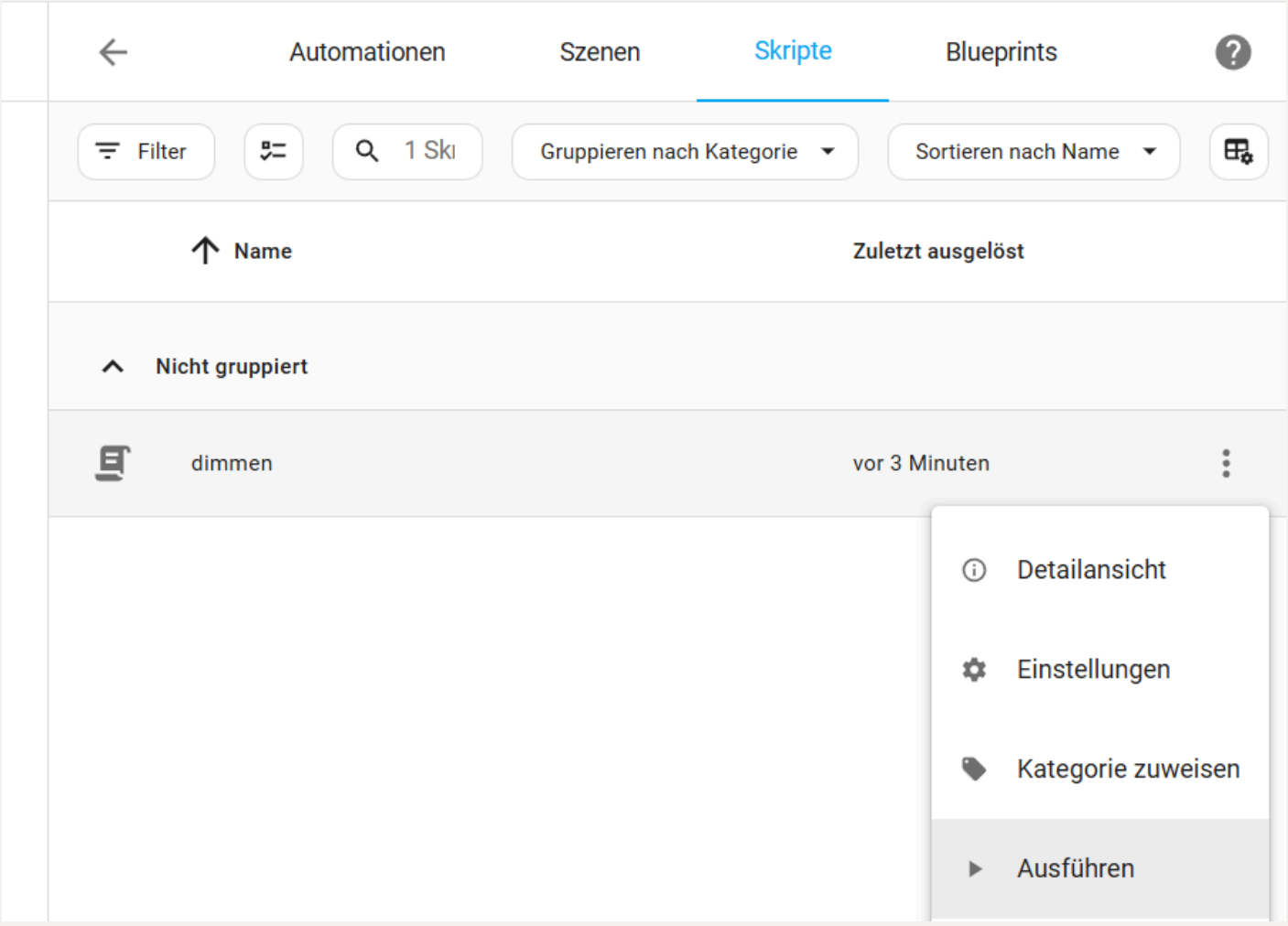


!

Trotzdem auf `UMBENENNEN` klicken.

> **Wichtig**
>
> Der Alias-Name darf nur Kleinbuchstaben, Ziffern und das"`_`"–Zeichen enthalten.

Um das Skript gleich zu erproben, wählen wir im rechten Seitenmenu aus dem Drei-Punkt-Menü `Skript ausführen`. Probieren wir es ein paar Mal hintereinander: die Lampe wird langsam hell und noch langsamer dunkel.



> **Anmerkung**
>
> Ich habe zum Ausprobieren eine IKEA-Lampe verwendet. Die Helligkeit der Lampe wird geringer, geht aber dann nicht *kontinuierlich* gegen Null, sondern bleibt bei einer geringen Helligkeit stehen und schaltet dann unmittelbar ganz ab.

Das war ja recht erfolgreich, aber wie verwenden wir ein Skript in einer Automation? Dazu erzeugen wir eine Kopie von unserer ersten Automation und bauen dort das Skript ein:

`Einstellungen` → `Automationen & Szenen`

`Lampe L1 umschalten` suchen und im Drei-Punkt-Menü auf `Duplizieren` klicken,.

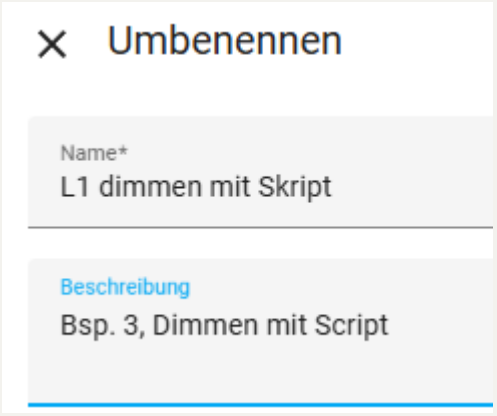
Rechts neben `Neue Automation` über das Drei-Punkt-Menü den YAML-Editor öffnen und die Automation bearbeiten.

Das YAML-Programm wird angepasst:

- `triggers` bleibt unverändert.
- `conditions` wird gelöscht.

- `actions` wird geändert auf `action: script.dimmen`.

`SPEICHERN`. Neuen Namen vergeben: `L1 dimmen mit Skript`



Nicht vergessen: die anderen Automationen abschalten!

Ein Skript für mehrere Lampen

Gelingt es nun, die Bezeichnung der Lampe zu ändern, dass *ein* Skript *mehrere* Lampen steuern kann? Ja – sonst wäre ein Skript kaum nützlich.

In höheren Programmiersprachen heißen diese Elemente *Parameter*, hier *Variablen*.

> **Anmerkung**
>
> Im Wort Parameter wird das zweite "a" betont, nicht das erste "e".

Allerdings sind größere Umbauten notwendig. Das Skript wird dupliziert und geändert:

```
1 alias: dimmen2
2 description: Dimmen von mehreren Lampen
3
```

```yaml
fields:
  lampe:
    description: Die zu steuernde Lampe
    example: light.l1
sequence:
  - choose:
      - conditions:
          - condition: template
            value_template: "{{ is_state(lampe, 'off') }}"
        sequence:
          - target:
              entity_id: "{{ lampe }}"
            data:
              brightness: 255
              transition: 2
            action: light.turn_on
      - conditions:
          - condition: template
            value_template: "{{ is_state(lampe, 'on') }}"
        sequence:
          - target:
              entity_id: "{{ lampe }}"
            data:
              brightness: 1
              transition: 6
            action: light.turn_on
          - delay: "00:00:06"
          - target:
              entity_id: "{{ lampe }}"
            action: light.turn_off
            data: {}
mode: single

```

Zu den Änderungen im Einzelnen:

Zeilen 1 und 2: Der Name des neuen Skripts und eine neue Beschreibung

Zeilen 4: Beginn der Vereinbarungen der Variablen (in höheren Programmiersprachen eine Liste der formalen Parameter)

Zeile 5: `lampe` – der Name des ersten (und hier auch einzigen) Parameters.

Zeilen 6 und 7: Eine optionale Beschreibung von `lampe` und ein möglicher Wert für `lampe`. Diese Erklärung wird beim Schreiben der Automation, die das Skript verwendet, eingeblendet.

Hier verwenden wir die lesbare Darstellung der *entity_id*: keine lange Folge von Hexadezimalziffern, sondern eine Entität aus der Gruppe der Lampen (*light*), und zwar die Lampe *l1*. Entitäten verwenden nur Kleinbuchstaben, Ziffern und das Unterstreichungszeichen.

Zeilen 9 bis 19: Große Änderungen!

- Zuerst wird in den Zeilen 10 bis 12 festgestellt, ob die Lampe abgedreht ist. Nur dann soll die `sequence` in den Zeilen 13 bis 19 ausgeführt werden.

> **Wichtig**
>
> **Ein neues Konzept: Templates**
>
> Zahlen, Namen, Texte oder diverse Ids waren bisher immer konstante Größen und damit zum Zeitpunkt der Umwandlung in die interne Darstellung (zum *Übersetzungszeitpunkt*) bekannt. Wenn aber Werte verwendet werden sollen, die erst zu dem Zeitpunkt bekannt sind, an dem die Automation läuft (zur *Laufzeit*), brauchen wir eine neue Darstellungsform.
>
> In YAML gibt es keine Variablen und auch nicht Funktionsaufrufe. Ein Ersatz dafür sind im Home Assistant *Templates*.
>
> Dazu ist der Name des *Fields* in doppelte geschwungene Klammern einzuschließen: zum Beispiel `{{ lampe }}`. Da aber in YAML (nur) Zeichenketten verwendet werden, kommt das Ganze auch noch in Doppelapostrophe: `"{{ lampe }}"`. Siehe Zeile 15.
>
> Schaut etwas umständlich aus, aber diese Schreibweise ist keine Erfindung von HA. Solche Konstrukte kommen beispielsweise in dynamischen Webseiten vor.
>
> In Templates können auch Funktionen aufgerufen werden. Beispiel in Zeile 12: `"{{ is_state(lampe, 'off') }}"`. Hier wird die Funktion `is_state` aufgerufen. Die Funktion fragt den Zustand (*state*) der `lampe` ab. Wenn *state* gleich `'off'` ist, liefert *is_state* den Wert `true`, sonst aber `false`.

- In der `condition` (Zeile 11) wird der Zustand der Lampe über ein `template` bestimmt. Die Zeile 12 zeigt, wie der Wert des Templates (`value_template`) ermittelt wird.
- Noch einmal im Detail: Zuerst wird festgestellt, *dass* die `condition` über ein `template` ausgewertet wird:
    - `condition: template`
    - Dann kommt die Formel, *wie* der Wert ausgerechnet wird:
    - `value_template: "{{ is_state(lampe, 'off') }}"`.
    - `is_state(lampe, 'off')` bedeutet: ist der Zustand der Lampe gleich `off`, also ausgeschaltet, ist die Bedingung wahr (`true`).

  Es ist eine *Bedingung*: die *Sequenz* wird nur abgearbeitet, wenn die Bedingung *wahr* ist, also wenn die Lampe *ausgeschaltet* ist.

  Zeile 15 zeigt das schon besprochene Template `"{{ lampe }}"`.

Die restlichen Zeilen von 13 bis 19 sind im Vergleich dazu harmlos.

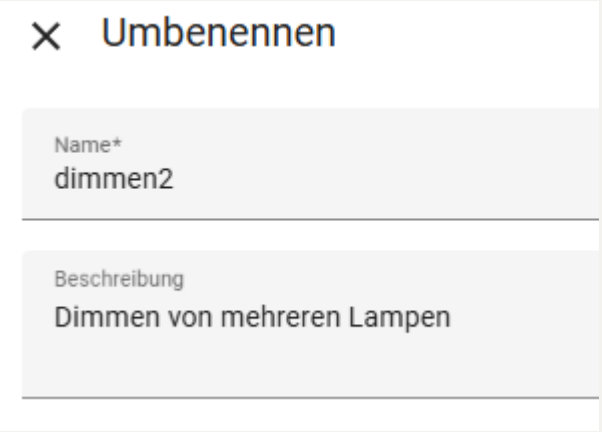
- Eine Befehlsfolge (sequence) wird nun abgearbeitet (Beginn ist Zeile 13).
- Zeilen 14 und 15: Worauf wird die Befehlsfolge angewendet? Das Ziel (target) wird bestimmt. Da es erst zur Laufzeit bekannt ist, wird ein Template verwendet.
- Zeilen 17 und 18: Welche Zusatzinformationen bekommt das Ziel? Das Umschalten von dunkel auf hell soll 2 Sekunden dauern, `transition: 2`
- Zeile 19: Was soll geschehen? Das ist der eigentliche Befehl zum Aufdrehen der Lampe.

Die Zeilen 20 bis 29 beschreiben den Vorgang, wenn die Lampe bereits aufgedreht ist und abgedreht werden soll und entsprechen den Zeilen 10 bis 19. Warum steht hier light.turn_on? Als Ziel ist die Helligkeit 1 (`brightness: 1`) angegeben.

Damit die Lampe aber auf jeden Fall abgedreht ist, wird in den Zeilen 31 bis 34 nach 6 Sekunden die Lampe komplett abgeschaltet.

> **Anmerkung**
>
> Um das Dimmen besser verfolgen zu können, kann die Zeit in Zeile 18 erhöht werden, zum Beispiel von 2 auf 10.

`SKRIPT SPEICHERN`, wenn erforderlich umbenennen:



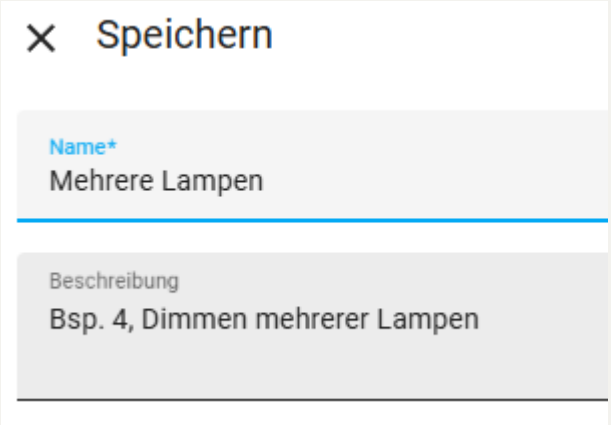
Die zugehörige Automation

Wir duplizieren `L1 Dimmen mit Skript` und ändern die actions–Zeilen, der Rest bleibt unverändert.

```
1 actions:
2   - data:
3       lampe: light.l1
4     action: script.dimmen2
5   - data:
6       lampe: light.l2
7     action: script.dimmen2
```

Na gut, die Zeilennummern (die automatisch erzeugt werden) beginnen wieder mit 1, aber es geht wirklich um die actions–Zeilen:

- Zeile 4: So sieht der Aufruf eines Skripts aus.
- Zeilen 2 und 3: so wird der aktuelle Parameter (auch Argument genannt) an das Skript übergeben.
- Zeile 1: Ja, das ist die Aktion, die abläuft, wenn die Automation gestartet wird.
- Zeile 5 bis 7: hier kommt eine zweite Lampe dazu.



- Ausschaltender anderen Automationen nicht vergessen.
- Ausprobieren!

Allerdings wird hier zuerst die Lampe L1 gedimmt und dann erst die Lampe L2. Das kann so gewünscht sein. Wenn nicht, ist eine weitere Änderung notwendig.

Paralleles Dimmen

ChatGPT hat mir ein neues Skript vorgeschlagen, das beliebig vielen Lampen parallel steuert:

```yaml
script:
  dim_lights_smooth:
    alias: Sanftes Licht An/Aus
    description: Dimmt übergebene Lampen sanft hoch oder runter, je nach aktuellem Zustand
    fields:
      lights:
        description: Liste der Licht-Entity-IDs
        example: ["light.l1", "light.l2", "light.l3"]
        required: true
    mode: parallel
    sequence:
      - choose:
          - conditions:
              - condition: state
                entity_id: "{{ lights[0] }}"
                state: 'off'
            sequence:
              - service: light.turn_on
                data:
                  entity_id: "{{ lights }}"
```

```
21                  brightness: 255
22                  transition: 2
23          - conditions:
24              - condition: state
25                entity_id: "{{ lights[0] }}"
26                state: 'on'
27            sequence:
28              - service: light.turn_on
29                data:
30                  entity_id: "{{ lights }}"
31                  brightness: 1
32                  transition: 6
33              - delay: "00:00:06"
34              - service: light.turn_off
35                data:
36                  entity_id: "{{ lights }}"
37
```

Wir erzeugen ein neues Skript, schalten auf die YAMP-Bearbeitung um und fügen den ChatGPT-Vorschlag ein.

Es ist doch schon bekannt, wie das geht? Bitte selbstständig probieren.

Probleme? Hier noch einmal der Ablauf:

`Einstellungen` → `Automationen & Szenen` → Reiter `Skripte` → `+ SKRIPT ERSTELLEN` → `Neues Skript erstellen` → `⋮` → `In YAML bearbeiten`

> **Anmerkung**
>
> Mit dem neuen Code den Eintrag in Zeile 1 *überschreiben*, nicht vorher löschen.

Die Zeilen 1 und 2 lassen wir weg, weil die Datei ja schon als Skript angelegt worden ist.

Wir könnten auch `lights` durch `lampen` ersetzen, aber lassen wir es einmal so stehen.

ChatGPT ist nicht unfehlbar. Es waren noch ein paar Änderungen notwendig, die sich aber im Dialog geklärt haben. Das ist das endgültige Skript:

```
1 alias: dimmen3
```

```yaml
description: Dimmt übergebene Lampen sanft hoch oder runter, je nach aktuellem Zustand
fields:
  lights:
    description: Liste der Licht-Entity-IDs
    example: "[light.l1, light.l2, light.l3]"
    required: true
mode: parallel
sequence:
  - choose:
      - conditions:
          - condition: template
            value_template: "{{ states(lights[0]) == 'off' }}"
        sequence:
          - data:
              entity_id: "{{ lights }}"
              brightness: 255
              transition: 10
            action: light.turn_on
      - conditions:
          - condition: template
            value_template: "{{ states(lights[0]) == 'on' }}"
        sequence:
          - data:
              entity_id: "{{ lights }}"
              brightness: 1
              transition: 6
            action: light.turn_on
          - delay: "00:00:06"
          - data:
              entity_id: "{{ lights }}"
            action: light.turn_off

```

Wie wird das neue Skript verwendet? Wir duplizieren die Automation `Mehrere Lampen nacheinander dimmen`:

`Einstellungen` → `Automation & Szenen` → `Mehrere Lampen nacheinander dimmen` →Drei-Punkte-Menü `⋮` → `Duplizieren` → `⋮` → `In YAML bearbeiten`

Wir ändern die Zeile 2 auf
`description: Bsp. 5, mehrerer Lampen nacheinander dimmen` description

Die `actions` (Zeilen 10 bis 16) werden zu:

```
1 actions:
2   - action: script.dimmen3
3     data:
4       lights:
5         - light.l1
6         - light.l2
```

Bitte die Zeilennummern wieder ignorieren.

Als `Bsp 5, paralleles Dimmen mehrerer Lampen` speichern.

SPEICHERN

Zusammenfassung

Das war eine sehr detaillierte Erklärung, wie ein Skript entsteht und wie es verwendet wird.

Z2H und ZHA

ZigBee-Netzwerke sind in der Home Automation sehr verbreitet. Mit *Zigbee2MQTT* (Z2M) oder *Zigbee Home Automation* (ZHA) kann ZigBee im Home Assistant verwendet werden.

Z2M ist ein externes Programm, das in mehreren Schritten installiert werden muss. ZHA ist ein Add-On zu Home Assistant und wird über einen Link installiert.

> **Anmerkung**
>
> ZigBee oder Zigbee? In den Spezifikationen wird ZigBee verwendet, im Namen von Programmen Zigbee.

Bei ZHA wird unter *Sobald* statt `single press` der Auslöser `Erste Taste kurz drücken` gewählt.

In ZHA sieht die YAML-Konfiguration des vorherigen Beispiels so aus:

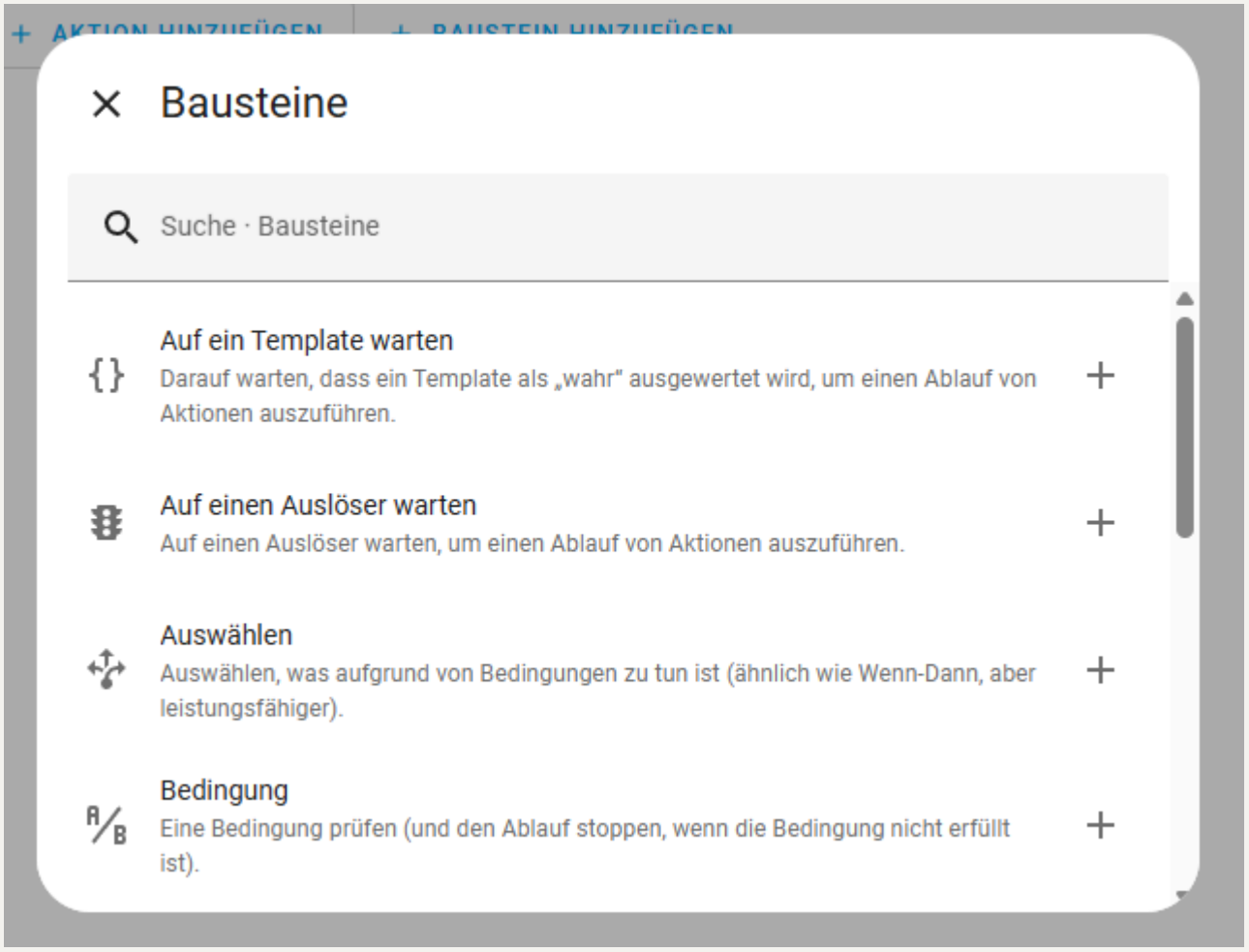
```yaml
alias: Lampe L1 umschalten
description: Erstes Beispiel zur Lampensteuerung
triggers:
  - device_id: 297b39892ff9a6bcd27cfdb5f46d69f5
    domain: zha
    type: remote_button_short_press
    subtype: button_1
    trigger: device
conditions: []
actions:
  - type: toggle
    device_id: 7c96f1e7727aa6c4be2d76c0269feb6b
    entity_id: 70ab137e9e0cda64b00f8cb8231f094c
    domain: light
mode: single
```

Bausteine

HA bietet nach `Einstellungen` → `Automation & Szenen` → `Skripte` → `+ SKRIPT ERSTELLEN` → `Neues Skript erstellen` auch an, Bausteine hinzuzufügen. Aber was sind Bausteine? Probieren wir es aus:

`+ BAUSTEIN HINZUFÜGEN`

Home Assistant bietet viele Bausteine an. Durch scrollen werden noch mehr sichtbar:



Einige dieser Bausteine sind schon von den Automationen bekannt:

- `Auf einen Auslöser warten` entspricht dem Sobald-Schritt in Automationen.
- `Bedingung` funktioniert wie bei den Automationen: wird sie nicht erfüllt, endet der Ablauf.

Weitere Bausteine:

`Eine Zeit warten (Verzögerung)` erklärt sich wohl selbst. Zeiten können in Stunden, Minuten, Sekunden und Millisekunden angegeben werden.

Ein interessantes Thema! Aber das passt besser in einen kommenden Clubabend oder Beitrag.

Das alles wurde beim Clubabend am 25. Mai 2025 besprochen und geübt – wer will, kann gerne bei den nächsten Terminen auch dazu kommen. Dieser Text geht aber bei ein paar Punkten noch mehr ins Detail.

Zu den Downloads und zur Langversion

Alle Zusatzinformationen (anklickbare Links, Downloads us.) sind in der Langversion zu finden.

Link und QR-Code: https://pcnews.at/markdown/index.html

Erstellt mit QR Code Generator – kostenfrei, ohne Login, ohne Konto: https://goqr.me/. Short URL erstellt mit https://t.ly/.

Copyright