

Switcher 3

Project Description



Version 3.6.4

Juni 2021

Inhaltsverzeichnis

1.	Abstract	4
2.	Anforderungen.....	4
3.	Switcher Hardware	4
3.1	Schaltbare Steckdosen für Switcher 3.....	5
4.	Implementierung Switcher 3	7
5.	Schaltlogik	7
5.1	Anzahl der Dosen	7
5.2	Logik der Dosen	8
5.3	Manuelle Beeinflussung des Schaltens.....	8
5.4	Nicht Zuhause Taste	9
5.5	Einzelne Dosen schalten	9
6.	Software	10
6.1	Allgemeines	10
6.2	Software Komponenten	11
6.3	Aufrufbare Programme	12
6.3.1	switcher3.py	12
6.3.2	swserver3.py	12
6.3.3	swlist3.py.....	12
7.	Web Interface Switcher3	12
7.1	Klassen im Switcher 3 System	14
7.1.1	Beschreibung der Klassen	15
7.1.2	Typen von Aktor-Klassen.....	16
7.1.3	Konfiguration der Dosenart	17
7.2	Testprogramme.....	18
8.	Informationsfluss im Switcher 3 System.....	18
8.1	WeblInterface	18
8.2	Besonderheit: Webseiten Laden	20
9.	Weitere Info	21
9.1	Switcher LED's	21

Inhaltsverzeichnis

9.2	Debugging, Logging	21
9.3	Threading im Switcher3.....	22
9.4	Implementierung Wetterdaten	22
9.5	XML Steuerfile	23
9.5.1	Anpassung der Schaltzeiten.....	23
9.5.2	Prüfung der Steuerfiles.....	24
10.	Anzahl installierte Dosen, Dosenbezeichnungen	24
11.	Betrieb des Switchers.....	25
11.1	Start des Switchers.....	25
11.2	Beenden des Switchers.....	25
11.3	Switcher Start bei Boot des Pi	26
11.4	Hinweise für Testen des Switchers.....	26
12.	MQTT und Switcher 3.....	27
12.2	Fehlerbehandlung bei MQTT Fehlern	27
12.3	Konfiguration für Schaltart 3	27
12.4	Mosquitto Stuff.....	28
12.5	Debugging MQTT	28
13.	Konfiguration der Sonoff/Shelly Smart Switches	29
13.1	Wichtige Bemerkung	30
14.	Anhang	31
14.1	XML Steuerfile	31
14.1.1	Auszug XML Steuerfile.....	31
14.1.2	Ajustierung Schaltzeiten.....	31
14.2	Config-File	32
14.3	Klasse MyPrint.....	36
14.3.1	What is the point exactly.....	36
14.4	Testaufbau MQTT	39
14.4.1	Arduino Sketch für ESP8266.....	40

1. Abstract

This documentation describes a Raspberry Pi gadget that is now in it's third incarnation. It is used to switch lights or other appliances on/off when not at home. Up to 5 power outlets are supported, mutiple on/off sequences for every switch per day are possible. The sequence is repeated every week. Switching times (ON/OFF) for every power outlet are defined in an XML Control-files. The XML file is parsed at reboot. A external pushbutton know as the ,At Home switch' is used to supress switching lights when at home.

Switcher supports 433 Mhz wall outlets as well as Wi-Fi enabled wall outlets (Smart Switches) using the MQTT protocol.

Switcher can also show indoor/outdoor temperatur/humidity using two MQTT-based (ESP8266) temperatur sensors.

A webinterface allows remote control of switcher.

Switcher3 Python Code and this project description is available on [GitHub](#).

2. Anforderungen

Die erste Version des Switchers wurde in 2014 entwickelt, in 2018 wurde eine neue Version eingeführt.
Die Anforderungen waren/sind:

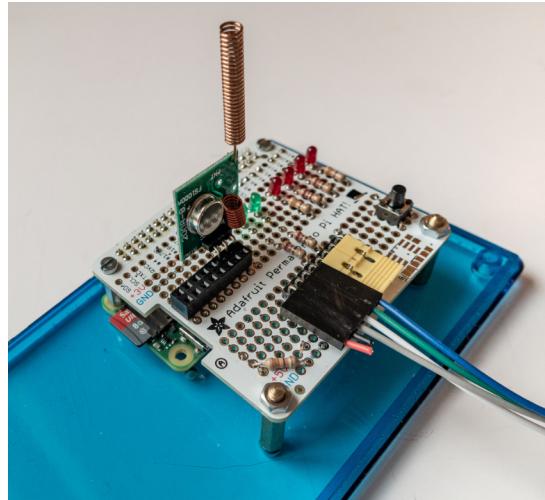
- Bei Ferienabwesenheit sollen maximal 5 Funksteckdosen gemäss einem vorgegebenen Schema ein- und ausgeschaltet werden. Die Funksteckdosen werden in verschiedenen Räumen platziert.
- Die damit individuell geschalteten Lampen sollen Anwesenheit vortäuschen.
- Statt Lampen können selbstverständlich auch andere Verbraucher geschaltet werden: Pflanzenbewässerung, Aquariumpumpen, und und...
- Pro Dose soll jeder Wochentag (von Sonntag bis Samstag) ein eigenes Schaltprogramm haben und die Schaltprogramme werden jede Woche wiederholt.
- Die Schaltaktionen sollen extern des Programms in einem XML-File definiert sein (genannt Controfile).
- Zudem soll via ein Web-Interface der aktuelle Schaltstatus abgefragt und Dosen manuell geschaltet werden können.
- Es sollen unterschiedliche Funksteckdosen verwendet werden können (herkömmlich ELRO), auch Smart Switches (WiFi).

3. Switcher Hardware

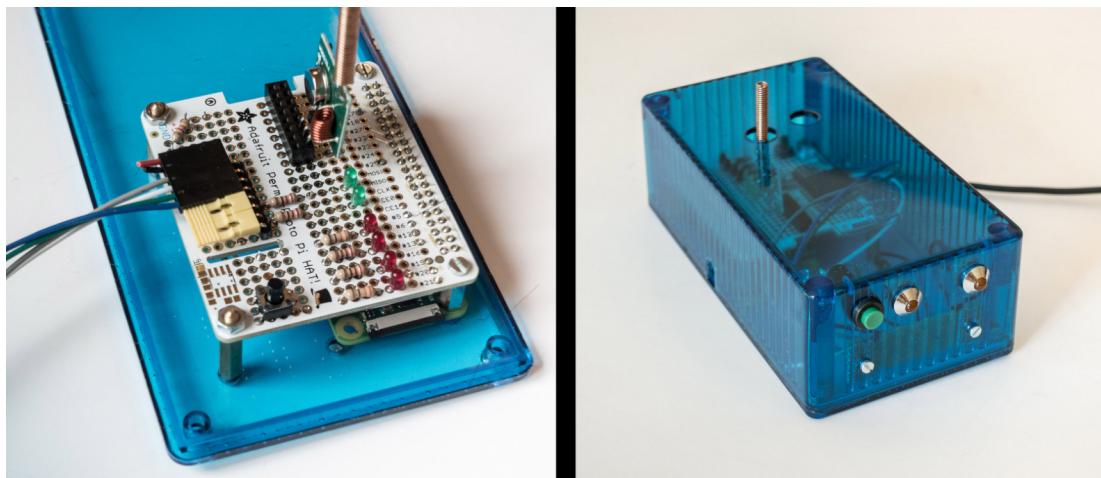
Der Switcher3 läuft auf einem Raspberry 3 Model A+, auf den ein kleines Elektronik-Modul aufgesteckt ist. Es wurde auch mit Erfolg ein Pi Zero verwendet. Das Elektronik-Module enthält diese Komponenten:

- 5 rote LED, die schaltbaren Steckdosen darstellend
- 1 grüne LED, blinkt im Betrieb
- 1 grüne LED, für Anzeige Jemand Zuhause/Niemand Zuhause

- 1 Pushbutton, für Umschaltung des Zuhause-Status
- 1 Sockel, in welchen ein 433 Mhz Sener gesteckt werden kann (für herkömmliche ELRO Funk-Steckdosen).



Switcher 3 Aufbau mit Pi Zero W und 433 MHZ Sendemodul (rote Led für Test)



Switcher 3 Aufbau auf Pi Zero W mit Gehäuse

3.1 Schaltbare Steckdosen für Switcher 3

Es können folgende Arten von schaltbaren Steckdosen verwendet werden:

- Die handelsüblichen 433 Mhz Funksteckdosen, wie sie für die erste Version des Switcher eingesetzt werden (siehe Bilder weiter oben). Wie lange diese Technologie noch im Markt ist, ist jedoch fraglich. ELRO hat dieses Produkt nicht mehr im Sortiment.
- Moderne Wi-Fi enabled Steckdosen resp. Schalteinheiten (Smart Switches). Diese sind schon sehr populär im Zusammenhang mit dem Smart Home (Internet of Things). Beispielsweise werden die

Produkte Sonoff oder Shelly sehr häufig von Hackern/Bastlern verwendet (siehe Bild). Für die Verwendung mit dem Switcher 2 muss die Firmware in diesen Gerätschen durch die bekannte Tasmota Software ersetzt werden (reflashing). Dazu gibt es im Internet und auf YouTube haufenweise Anleitungen. Es gibt auch Firmen in DE, welche reflashte Versionen anbieten - dann muss man dies nicht selbst machen. Siehe Links.

Im Switcher 3 (im Module Aktor_3.py) muss das passende MQTT-Protokoll (also Topic und Payload) richtig codiert werden. Siehe Kapitel MQTT mit Switcher 3 weiter hinten.

- Selbstbau eines Wi-Fi enabled Schalters. Das ist recht einfach, man braucht dazu einen ESP8266, ein Relais, einen kleinen 5 Volt Powersupply und eine Arduino Entwicklungsumgebung. Der Code für den ESP8266 ist recht einfach. Der Beispiel Sketch swesp_dose_1.ino kann als Beispiel und Ausgangslage dienen (siehe Source Code Switcher 3, Ordner esp_code).

Note:

Wie bereits weiter oben dargelegt, ist es möglich, den Switcher 3 gleichzeitig mit verschiedenen Arten von Dosen zu betreiben: Dose 1 mit 433MHz Funktechnologie und Dose 2 bis 4 mit modernen Wi-Fi enabled Steckdosen. Im Config File ist lediglich die richtige Schaltart für jede Dose zu definieren. This is very cool.



Herkömmliche Funksteckdosen



Sonoff Smart Switch mit Verkabelung (durchgehende Erdleitung), Schweizer Norm

Shelly 1



Shelly 1 Wi-Fi Smart Switch

4. Implementierung Switcher 3

5. Schaltlogik

Wie bereits gesagt, wurde der Switcher entwickelt, um eine Anzahl von Steckdosen ein- oder auszuschalten, gemäss einer in einem XML-Steuerfile definierten Abfolge von Schaltaktionen. Außerdem können einzelne Steckdosen durch den Benutzer manuell ein- oder ausgeschaltet werden (Webinterface).

5.1 Anzahl der Dosen

Im XML-Steuerfile (siehe Anhang) sind 5 Dosen definiert. Die Anzahl der effektiv geschalteten Dosen

ist variable und die verwendete Anzahl kann im WebInterface jederzeit verändert werden. Es wird dazu ein eigener Konfig-File verwendet, der vom Switcher updated werden kann. Dieser File wird zu Beginn gelesen. Dateiname: swdosen.ini

5.2 Logik der Dosen

Die Implementierung der Schaltlogik (nicht des physikalischen Schaltens) kann so beschrieben werden: Es gibt **automatische** und **manuelle** Schaltaktionen (Ein/Aus) auf einer Dose.

Automatische Schaltaktionen sind ausgelöst durch die Abarbeitung der im Steuerfile definierten (programmierten) Aktionen. **Manuelle** Schaltaktionen sind von Benutzer ausgelöst (an der Dosen selbst oder auf dem Webinterface). Siehe nächster Abschnitt für Beschreibung der manuellen Beeinflussung des Schaltens.

Die physikalischen Dosen sind im Switcher-Programm abgebildet durch Instanzen der Dosen-Klasse. Der **Schaltzustand** jeder **Dose** ist in der entsprechenden Instanz durch **4 Stati** beschrieben. Der Switcher kann die Dosen nach ihrem Status befragen durch eine Methode der Dosen-Klasse.

- **Interner Status:** zeigt in jedem Moment den Schaltzustand einer Dose gemäss den programmierten Aktionen - er zeigt also, wie der Dosenstatus (ein/aus) wäre, gäbe es keine manuellen Eingriffe. Dieser interne Status muss aber nicht dem wirklichen Schaltzustand der Dose entsprechen, denn dieser kann ja durch manuelle Einflussnahme modifiziert sein.
- **Externer Status:** dies zeigt den effektiven Schaltzustand einer Dose unter Berücksichtigung aller manuellen Eingriffe. Dieser Status wird für die Statusabfrage benutzt - dort möchte man sehen, ob eine Dose momentan tatsächlich Ein oder Aus ist.
- **Schaltmodus** der Dose: dieser ist entweder auto oder manuell. Wird eine Dose manuell geschaltet, so wechselt der Schaltmodus dieser Dose auf manuell. In diesem Modus wird die Dose durch automatische Schaltaktionen nicht mehr verändert. Der Schaltmodus einer Dose kann wieder auf auto zurückgesetzt werden.
- **Schaltpriorität** der Dose: dieser kann 1 oder 2 sein. Wert 1 ist der Normalfall: die Dose wird geschaltet wie im folgenden beschrieben.
Wert 2 bedeutet, dass eine Dose immer gemäss dem definierten Programm geschaltet wird - also unabhängig von Zuhause / Nicht-Zuhause.

5.3 Manuelle Beeinflussung des Schaltens

Der Normalbetrieb des Switchers besteht in der Abarbeitung der im Steuerfile definierten Schaltaktionen für die verschiedenen Dosen (im folgenden genannt **automatische** Schaltaktionen). Das definierte Wochenprogramm wird endlos wiederholt.

Es gibt jedoch auch die Möglichkeit der Einflussnahme auf diesen automatisierten Ablauf. Wir nennen sie **manuelle** Einflussnahmen.

Es gibt:

- eine Drucktaste **Zuhause / Nicht Zuhause**, ein Druck auf diese Taste wechselt den Zustand von Zuhause nach Nicht-Zuhause und umgekehrt (toggle) - dies beeinflusst den automatisierten Ablauf. Diese Umschaltung ist auch am Webinterface möglich. **Note:** In Switcher3 wird nur noch eine Drucktaste unterstützt, Kippschalter wird nicht unterstützt.

- Via Webinterface können alle oder auch einzelne Dosen manuell ein- oder ausgeschaltet werden.
- Smart Switches können auch manuell geschaltet werden - durch Betätigen der Drucktaste am Switch.

5.4 Nicht Zuhause Taste

Der Switcher ist so designed, dass er das ganze Jahr über aktiv sein kann. Während des normalen Daheim-Seins möchte man den Switcher jedoch inaktivieren - es sollen keine Dosen geschaltet werden. Bei abendlichen Abwesenheiten soll er jedoch einfach wieder aktiviert werden. Mit der Drucktaste **Zuhause / Nicht Zuhause** kann der automatische Schaltablauf des Switchers beeinflusst werden. Alle Dosen sind dabei betroffen. Die Zustände sind:

- Nicht Zuhause: Switcher schaltet die Dosen automatisch gemäss den Angaben im Steuer-File (that is what he is designed to do).
- Zuhause: Switcher schaltet nach Umschalten auf Status **Jemand Zuhause** die Dosen mit Schaltpriorität 1 sofort AUS. Programmierte Aktionen (Aus/Ein) werden für diese Dosen NICHT mehr durchgeführt, der aktuelle Dosenstatus wird jedoch intern nachgeführt. Dosen mit Schaltpriorität 2 werden vom Status **Jemand Zuhause** nicht beeinflusst - sie werden IMMER automatisch geschaltet.

Beim Umschalten von **Jemand Zuhause** auf **Niemand Zuhause** werden die Dosen sofort wieder auf den zu diesem Zeitpunkt gültigen Stand gemäss Steuerfile geschaltet. Das vordefinierte Programm wird weiter normal abgearbeitet.

Anders gesagt: Der Switcher (genauer gesagt die Klasse Sequencer) arbeitet das vordefinierte Programm IMMER ab: wenn **jemand zuhause** ist, wird eben nur der interne Status nachgeführt. Damit bilden die internen Stati jederzeit den Stand der bislang abgearbeiteten Schaltaktionen ab.

5.5 Einzelne Dosen schalten

Mit Hilfe des Webinterfaces kann der Schalt-Zustand einzelner Dosen beeinflusst werden. Es gibt folgende Möglichkeiten:

- Eine spezifische Dose manuell ein- der ausschalten. Die Schaltaktion erfolgt sofort und der Zustand ist gültig bis Mitternacht oder forever (im Config-File definierbar). **Allerdings: Reboot des Pi entfernt manuelle Einstellungen.** Die Statusabfrage zeigt diesen manuell geschalteten Zustand einer Dose an.
- Eine spezifische Dose wieder aus dem manuell geschalteten Zustand in den Normalzustand zu setzen - sie wird dann ab sofort wieder gemäss den definierten Schaltaktionen geschaltet.

Für das manuelle Beeinflussen in die programmierten Schaltsequenzen gelten folgende Regeln:

- Manuelles Ein-/Ausschalten einer Dose hat immer Vorrang gegenüber den programmierten Schaltaktionen.
- Beim Umschalten von **Niemand Zuhause** auf **Jemand Zuhause** gilt ebenfalls: manuell geschaltet hat Vorrang. Beispielsweise: wurde eine Dose manuell eingeschaltet, so wird sie beim Umschalten auf **Jemand Zuhause** nicht ausgeschaltet.

6. Software

6.1 Allgemeines

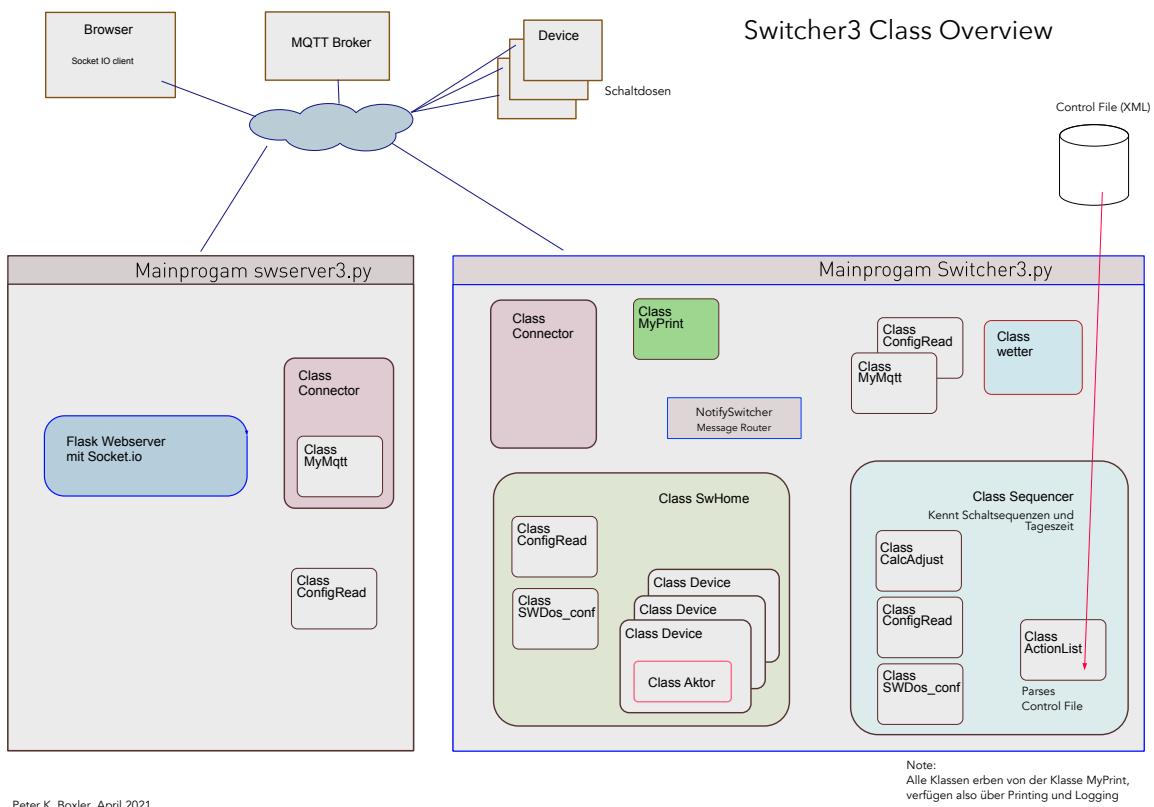
Die verwendete Script-Sprache ist Python Version 3.x. Das Switcher-Programmsystem besteht aus ca. 5800 Zeilen Python Code. Switcher3 hat 2 Hauptprogramme: **switcher3.py** und **swserver3.py**. Während switcher3.py alle Funktionen des Schaltens enthält, implementiert swserver3.py das Frontend mit Hilfe eines Flask Webservers. In diesem Dokument wird vom **Server** gesprochen - damit ist das Programm **swserver3.py** gemeint.

Kommunikation zwischen den beiden Hauptprogrammen (und ihren Komponenten) wird durch **MQTT** Messages erledigt.

Der gesamte Code des Switchers-Programmsystems ist in verschiedenen Python Modulen und Klassen codiert. Durch Einsatz von Objektorientierung und Ekapsulierung wurde weitgehendste Unabhängigkeit erreicht. Dies erleichtert die Wartungsarbeiten und isoliert die diversen Funktionen.

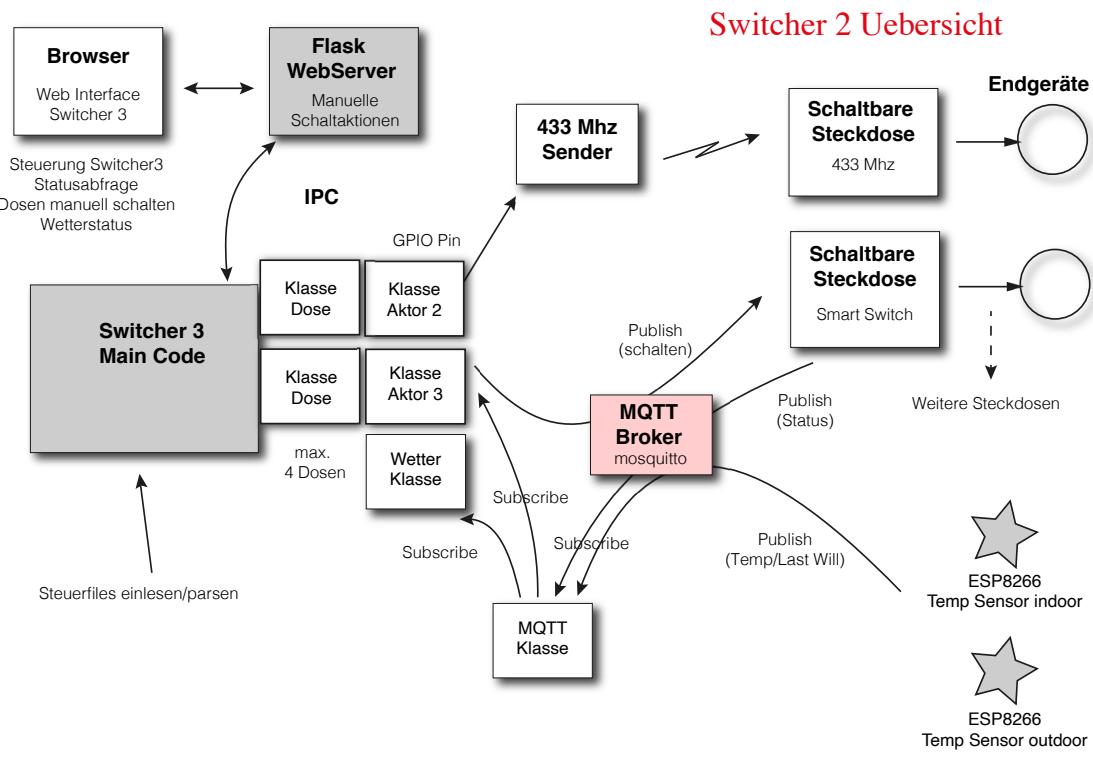
Siehe weiter hinten für ausführliche Diskussion der Klassenhierarchie.

Switcher3 Python Code und diese Dokumentation steht auf [GitHub](#) zur Verfügung.



Klassen Hierarchie Switcher 3

Eine etwas andere Darstellung des Switcher 3 Systems könnte so dargestellt werden:



6.2 Software Komponenten

Die Funktionalität des Switcher 3 Systems ist in 2 Hauptprogramme aufgeteilt. Es gibt verschiedene Gründe dafür:

- Unabhängigkeit der wichtigsten Funktionen: der Webserver soll immer in der Lage sein, den Logfile des Switchers auszugeben - auch wenn der Switcher selbst wegen eines Fehlers abgestürzt ist.
- Divide et impera oder do not put all your eggs in one basket. Die Behandlung des Frontends (Webinterface) soll getrennt sein von der eigentlichen Schaltlogik.
- Weil es Spass macht, MQTT zu benutzen.

Das Switcher 3 System besteht also folgenden Programmen:

Programm	Funktion
switcher3.py	Enthält alle Funktionen/Klassen die das Schalten der Dosen erledigen.
swserver3.py	Implementiert einen Flask Webserver mit Socket.io.
swlist3.py	Hilfsprogramm, gibt eine Liste aller im XML Steuerfile codierten Schaltaktionen aus..

6.3 Aufrufbare Programme

6.3.1 switcher3.py

Dies ist eines der beiden Hauptprogramm, welche für den Betrieb des Switcher 3 gestartet werden müssen. Dieses Programm schaltet die Dosen und versorgt das WebInterface mit den notwendigen Daten, um den jeweils aktuellen Schaltzustand anzuzeigen.

6.3.2 swserver3.py

Dieses zweite Hauptprogramm implementiert einen Flask Webserver, der die html-Seiten zur Verfügung stellt-

Die beiden Programm stehen via MQTT Messages in Verbindung.

6.3.3 swlist3.py

Das Programm **swlist3.py** wird benutzt, um die im XML-Files codierten Schaltzeiten pro Tag und pro Dose gut lesbar auszugeben. Es werden 3 Listen ausgegeben [siehe Beispiel im Anhang](#).

- Schaltzeiten für alle Tage der Woche für alle Dosen.
- Schaltzeiten für jede Dose für alle Tage der Woche:
- Liste der definierten Zimmer

7. Web Interface Switcher3

Die Funktionen von Switcher3 können am WebInterface eingesehen und beeinflusst werden.
Es gibt diese Webseiten:

- Home Page: Anzeige der wichtigsten Infos und der Schaltzustände der definierten Dosen
- Aktionen: Anzeige der Schaltaktionen des aktuellen Tages
- Info: Weitere, detaillierte Statusinformationen des Switcher Systems
- Log: Anzeigen des Switcher Logfiles und Auslösen von Reboot oder ShutDown. Hier kann ebenfalls die Anzahl der definierten Dosen modifiziert werden.

Switcher 3 Beschreibung

Switcher 3 Home

Datum/Zeit: 20.November 2021 : 19:36 Wochentag: Samstag Nächste: 22.12 Uhr, Arbeitszimmer / EIN

Letzte: 17.22 Uhr, Wohnzimmer / EIN

Niemand daheim

Wohnzimmer	Toggle Ein	Toggle auto	Typ1 / Smart
Wintergarten	Toggle Aus	Toggle auto	Typ1 / Funk
Ankleide	Toggle Aus	Toggle auto	Typ1 / Funk
Arbeitszimmer	Toggle Aus	Toggle auto	Typ1 / Funk

Alle Auto Alle ON Alle OFF

Switcher 3 Application (Versions: 3.02 / 3.04)

Home Page, 4 Dosen definiert

Switcher 3 Schalten

Zukünftige Aktionen des Tages (aktuelle Zeit: 19:36)

Zeit: 22.12 (22.12) EIN Dose: 4 Arbeitszimmer
Zeit: 22.20 (22.20) EIN Dose: 2 Wintergarten
Zeit: 23.09 (23.09) AUS Dose: 4 Arbeitszimmer
Zeit: 23.20 (23.20) AUS Dose: 2 Wintergarten
Zeit: 23.43 (23.43) EIN Dose: 3 Ankleide
Zeit: 23.57 (23.57) AUS Dose: 1 Wohnzimmer
Zeit: 23.59 (23.59) AUS Dose: 3 Ankleide

Vergangene Aktionen des Tages

Zeit: 00.02 (00.02) EIN Dose: 1 Wohnzimmer
Zeit: 00.05 (00.05) EIN Dose: 4 Arbeitszimmer
Zeit: 00.17 (00.17) AUS Dose: 4 Arbeitszimmer
Zeit: 00.20 (00.20) EIN Dose: 3 Ankleide
Zeit: 00.25 (00.25) EIN Dose: 2 Wintergarten
Zeit: 00.25 (00.25) AUS Dose: 1 Wohnzimmer
Zeit: 00.35 (00.35) AUS Dose: 3 Ankleide
Zeit: 00.45 (00.45) AUS Dose: 2 Wintergarten
Zeit: 16.57 (21.10) EIN Dose: 2 Wintergarten
Zeit: 17.17 (21.30) AUS Dose: 2 Wintergarten

Switcher 3 Application (Schaltzeiten ajustieren: 46 / 253 / Ja / W)

Anzeige der Schaltaktionen des aktuellen Tages

Switcher 3 Beschreibung

Switcher 3 Info

Home Aktionen Info Wetter Log

Version / Dosen: 3.02 / 4	Start: Samstag, 20 November 2021 : 19:36:02	Laufzeit Tage: 0
Total Schaltaktionen: 00001 / 00003	Woche / Adjust (Min): 46 / 253 / Ja / W	Zuhause: Niemand daheim
File-ID: Prisca 5 Devices	Testmode / Debug: Nein / 1	Heute: Samstag, 20 November 2021 : 19:37:05
Aktueller Tag: Samstag	Aktionen an diesem Tag: 18	Davon ausgeführt: 11
Wartend bis: 22.12 Uhr	Nächste: 22.12 Uhr, Arbeitszimmer / EIN	Letzte: 17.22 Uhr, Wohnzimmer / EIN
Wetter: Nein	Reset Manuelle: Um Mitternacht	Host Info: swi2 / 192.168.1.153

Switcher 3 Application

Weitere Statusinformationen des Switcher System

Switcher 3 Logfile

Home Aktionen Info Wetter Log

```
2021-05-20 11:50:47,927 - swserver3 : swserver3 setup Server done
2021-05-20 11:50:47,929 - swserver3 : swserver3 : start flask WITHOUT debug
2021-05-20 11:50:55,450 - switcher3 : sequencer object created: ActionList()
2021-05-20 11:50:55,594 - switcher3 : sequencer object created: SWDos_conf()
2021-05-20 11:50:55,596 - switcher3 : swc_dosconf Dosen anzahl gefunden: 4
2021-05-20 11:50:55,597 - switcher3 : sequencer Nehme Anzahl Dosen aus swconfig.ini, Anzahl:4
2021-05-20 11:50:58,610 - switcher3 : switcher3 object created: MySequencer, Anzahl Threads: 4, List: [<_MainThread(MainThread, started 1979008096)>, <Thread(Thread-2, started daemon 1969222752)>, <Thread(Thread-3, started daemon 1950344288)>]
2021-05-20 11:50:58,611 - switcher3 : switcher3 object created: SwHome(True)
2021-05-20 11:50:58,612 - switcher3 : switcher3 object created: MQTT_Conn()
2021-05-20 11:50:59,614 - switcher3 : switcher3 Setup returns errorcode:0
2021-05-20 11:51:00,547 - swserver3 : swserver3 process switcher3 läuft
2021-05-20 11:51:00,548 - swserver3 : swserver3 process swserver3 läuft
2021-05-20 11:51:00,612 - switcher3 : sequencer Sequencer_run(): start switching. Zeit und Wochentag: ['11.51', '4']
2021-05-20 11:51:32,364 - swserver3 : swserver3 process switcher3 läuft
2021-05-20 11:51:32,365 - swserver3 : swserver3 process swserver3 läuft
2021-05-20 11:51:34,997 - swserver3 : swc_dosconf Dosen anzahl gefunden: 4
2021-05-20 11:51:39,641 - swserver3 : swc_dosconf Dosen anzahl gefunden: 4
2021-05-20 11:51:39,645 - swserver3 : swc_dosconf --> write dosenconfig 2 called
```

Hinweis: MQTT Status OK

Reboot Pi ShutDown Pi Anzahl Dosen

Switcher 3 Application

Anzeige Logfile

7.1 Klassen im Switcher 3 System

Die wichtigsten Aufgaben sind in Klassen gekapselt. Hier eine kurze Erläuterung der Klassen im Hauptprogramm switcher3.py. Siehe Skizze **Klassen-Hierarchie** weiter oben für eine Uebersicht

Es sind im Switcher3 System folgende Klassen definiert, siehe auch detaillierte Beschreibung weiter unten.

Die Kolonne Testprogramm listet pro Klasse das entsprechende Testprogramm für den Unit-Test..

Klasse	Sourcefile	Enkapsuliert diese Funktion	Testprogramm
SwHome	swc_home.py	Das Haus, welches Dosen hat	
Dose	swc_dose.py	Dose mit ihren Zuständen	swt_dev.py
Aktor_1	swc_aktor_1.py	Schaltaktion für Testaufbau (LED)	swt_dos.py
Aktor_2	swc_aktor_2.py	Schaltaktion für 433 Mhz Dosen	swt_dev.py
Aktor_3	swc_aktor_3.py	Schaltaktion für Smart-Switches	swt_dev.py
Aktor_4	swc_aktor_4.py	Schaltaktion (Legacy Switcher 1)	swt_dev.py
Aktor_5	swc_aktor_5.py	Schaltaktion Spezial 433 Mhz Dosen	swt_dev.py
SwConnector	swc_connector.py	MQTT Verbindung zwischen switcher3.py und swserver3.py	
ConfigRead	swc_config.py	Lesen eines config.ini Files	swt_conf.py
Wetter	swc_wetter.py	Empfangen Wetterdaten von Sensor	
ActionList	swc_actionlist.py	Einlesen, Parsen des XML Steuerfiles, erstellen einer Liste von Aktionen	
CalcAdjust	swc_adjust.py	Zeitanpassung von Aktionen	swt_adj.py
MQTT_Conn	mymqtt.py	Everything MQTT	
MyLog	myprint.py	Implementiert Python Logging	
MyPrint	myprint.py	Print und Logging (The Better Print Statement), erbt von MyLog	swt_pri1.py und swt_pri2.py
SWDos_conf	swc_dosconf.py	Kapselt Behandlung des Dosen-Config Files swdosen.ini	swt_confdos.py
MySequencer	swc_sequencer.py	Generieren von Events basierend auf der Liste der Aktionen	swt_seq.py

7.1.1 Beschreibung der Klassen

Hauptprogramm switcher3.py instanziert folgende Klassen:

- **MyLog:** Diese Klasse implementiert das Python Logging. Da die MyPrint Klasse von MyLog erbt, haben alle weiteren Klassen das Logging mit dabei.

- **MyPrint:** diese Klasse hat im Switcher3 eine spezielle Rolle: ALLE anderen Klassen erben von MyPrint. Die Klasse MyPrint seinerseits erbt von der Klasse MyLog. Siehe Anhang für eine Beschreibung der Funktionalität dieser Klasse.
- **SwConnector:** Alle Messages zwischen den beiden Hauptprogrammen werden durch diese Klasse geschleust. MQTT Message Verkehr ist per Definition asynchron. Im Switcher 3 wird aber auch Synchroner Message Verkehr benutzt - siehe auch Kapitel Synchrone Messages. Die SwConnector Klasse instanziert die Klasse **MQTT_Conn**, welche die Low Level MQTT Funktionalität übernimmt.
- **SwHome:** diese Klasse bildet das Haus ab, welches Dosen enthält. Diese Klasse verwaltet den Status Jemand Zuhause/ Niemand Zuhause. SwHome instantiiert für jede definierte Dose (im Config File) eine Instanz der Klasse Dose.
- **My_Blink:** blinken einer Led (falls konfiguriert im File swconfig.ini (wird nur in switcher3.py verwendet)
- **Dose:** die Klasse Dose verwaltet den Schaltzustand der Dose, gemäss der oben erläuterten Schaltlogik. Dies Klasse Dose ist in gewissem Sinne generisch, sie weiss nicht, wie die Dose physikalisch zu schalten ist. Dies wird von der Aktor Klasse übernommen.
- **Aktor_n:** von dieser Klasse gibt es 5 Ausprägungen. Diese Klasse schaltet eine Dose physikalisch, sie hat aber kein Wissen um den effektiven Schaltzustand der Dose. Jede instanzierte Dosenklasse holt die Info über die physikalische Art der Dose (433 Mhz Funk, Smart Switch) aus dem Config File und instanziert demgemäß die passende Aktor Klasse. Jede Instanz der Klasse Dosen hat also genau eine passende Instanz der Klasse Aktor. Siehe folgendes Kapitel für die Typen der Klasse Aktor.
- **MySequencer:** (von lateinisch: Aufeinanderfolge) diese Klasse generiert Events,
 - Action-Events , diese werden generiert durch das Abarbeiten der definierten Schaltaktionen im XML Steuerfile. Diese Events werden an die entsprechende Instanz der Klasse Dosen weitergeleitet, diese veranlasst die in ihr instantiierte Klasse Aktor die Dose zu schalten.
 - Time of Day Events (Beispiel : Midnight, Time now), diese Events werden ebenfalls an die Klasse Dosen weitergereicht.
 - Update Events: Der Name des Zimmers, in welchen sich eine Dose befindet ist im XML-Steuerfile definiert. Nach dem Parsen dieses Files (durch Klasse ActionListener) wird ein Update Event mit den Namen des Zimmers an jede instatierte Dose gesandt.
- **SWDos_conf:** diese Klasse kapselt die Behandlung (read und write) des Dosen Config Files swdosen.ini.
In diesem File ist die Anzahl der installierten Dosen gespeichert. Siehe Beschreibung im entspr. Kapitel.
- **Wetter:** diese Klasse beinhaltet den Code für das Empfangen der Daten der beiden Temperatur-Sensoren (Indoor und Outdoor, basierend auf ESP8266). Sie macht eine Subscription zum MQTT Broker und bekommt damit die Meldungen, welche die Sensoren mit Publish absetzen. Im Konfig-File des Switchers kann spezifiziert werden, ob Wetterdaten empfangen und angezeigt werden sollen.

7.1.2 Typen von Aktor-Klassen

Zur Zeit sind folgende Aktor-Klassen definiert - im Config-File wird für jede der max. 5 Dose einen sog.

Schaltart definiert. Schaltart meint: von welchem technischen Typ eine vorhandene Dose ist. Die Klasse Dose instantiiert dann die passende Aktor-Klasse (welche das effektive Schalten vornimmt):

- **Aktor_1:** Für Test und Entwicklung, es werden 4 Led's angesteuert via 4 GPIO Ports. Die Led's stehen für die 4 geschalteten Steckdosen. So kann während der Entwicklung die Funktionsweise des Switchers einfach visuell geprüft werden.
- **Aktor_2:** Für Implementierung mit einem kleinen 433 Mhz Sender, dessen Data-In an einen einzelnen GPIO Port angeschlossen ist. Die benötigten Sendepulse wurden ermittelt gemäss einer Anleitung, die im Netz gefunden wurde (siehe Links).
- **Aktor_3:** Für Ansteuerung via einen MQTT Broker. Hier werden die modernen Wi-Fi enabled Switches wie beispielsweise der Marke SONOFF (Smart Switch) gesteuert. Der Aktor-Code publiziert die Ein/Aus Befehle an den MQTT Broker mosquitto, der im Normalfall auf demselben Raspberry Pi läuft. Die IP-Adresse des Brokers ist im Config File definiert.
- **Aktor_4:** Für Ansteuerung des Handsenders wie in der ersten Version des Switchers. Es werden 6 GPIO-Pins verwendet. Dies ist für vorläufige Rückwärts-Kompatibilität des Switchers 2 mit dem Hardware-Aufbau der Version 1 des Switchers.
- **Aktor_5:** Für Implementierung mit einem kleinen 433 Mhz Sender, dessen Data-In an einen einzelnen GPIO Port angeschlossen ist. Hier wird jedoch das 433MHz Sendemodule verwendet (siehe Link). Auf dem Pi muss hierfür die obskure Library wiringpi installiert sein. Diese Version courtesy meinem Freund Habi, dessen Switcher momentan so läuft.

Note:

Wie erwähnt, kann jede der definierten Dosen eine andere Schaltart haben. Es ist also beispielsweise möglich, dass Dose 1 und 2 althergebrachte 433 Mhz Funksteckdosen sind, während Dose 3 und 4 moderne IoT Dosen auf Basis MQTT sind.

Note 2:

Wird im Configfile der Eintrag „testmode“ auf „Ja“ gesetzt, so werden alle Dosen die Schaltart 1 verwenden (Testumgebung, Testsetup mit 5 LED).

7.1.3 Konfiguration der Dosenart

Dieser Abschnitt im Config File definiert den Typ jeder Dose. Also: eine Änderung in einer einzigen Zeile genügt, um eine Dose, resp. deren Schaltart zu definieren.

```
# schaltart definiert die Art des Schaltens, entspr. Aktor-Klasse wird instantiiert
#      1: Für Testaufbau mit 4 Led ,
#      2: Funk mit 433 MHz Sender ,
#      3: IoT MQTT Publish
#      4: Funk mit Handsender (wie original Switcher)
#      5: Funk mit 433 Mhz Sender mit send Module
#
# Achtung:
# Schaltart 1 ist für Testumgebung mit 5 LED.
# Wenn eine dose 1 schaltart 1 hat, müssen die anderen Dosen auch Schaltart 1 haben
# script sorgt dafür (mit Warnung), dass dies eingehalten wird.
# Ansonsten ist schaltart frei wählbar
# Bei Schaltart 3 gibt es zwei weitere Parameter, etwa so: 3,Y,X
#   Y: Art der MQTT Meldung, also Topic und Payload
#       (1: Sonoff-SmartSwitches, 2: future use)
#   X: Smart Switch sendet Rückmeldung, also Susbcrite möglich/nötig: 0 Nein, 1 Ja
```

```
# Also: für 'normale' Sonoff Switches 3,1,1 setzen  
dose_1_schaltart = 2  
dose_2_schaltart = 2  
dose_3_schaltart = 2  
dose_4_schaltart = 3,1,1  
dose_5_schaltart = 3,1,1
```

7.2 Testprogramme

Für den Unit-Test der diversen Module und Klassen stehen Testprogramme zur Verfügung.

Programm	Funktion
swt_dos.py	Unit Test der Dosenklasse und der AktorKlasse, schaltet 4 Dosen ein und aus
swt_conf.py	Unit Test des Lesens eines config.ini Files
swt_test.py	Unit Test der Test Platine mit 5 Led
swt_xml.py	Validierung und Struktur-Test eines XML Steuerfiles
swt_pri1.py	Unit Test der MyPrint Klasse
swt_pri2.py	Unit Test der MyPrint Klasse mit Programmfehler
swt_mqtt_pub.py	Unit Test Publish der MQTT_Conn Klasse
swt_mqtt_sub.py	Unit Test Subscribe der MQTT_Conn Klasse
swt_swi.py	Unit Test des Switchers, simuliert Verhalten des swserver3.py
swt_serv.py	Unit Test des swserver3.py, simuliert Verhalten des switcher3.py
swt_adj.py	Test des Zeit Ajustierung der Schaltzeiten
swt_thread	Beispiel Programm für Threading Class, nach diesem Beispiel ist die MySequencer Classe gebaut.
swt_json.py	Einfacher Test von json.loads() und json.dumps()

Achtung:

Die Testprogramme sind alle im Folder **testscripts** zu finden. Bevor ein solches Programm ausgeführt werden kann, muss es in den Hauptfolder des switcher3 kopiert werden: also in den dem Folder **testscripts** übergeordneten Folder. Also: wechseln in den Folder switcher3, dann (Beispiel):

```
cp testscripts/swt_adj.py .
```

8. Informationsfluss im Switcher 3 System

8.1 WebInterface

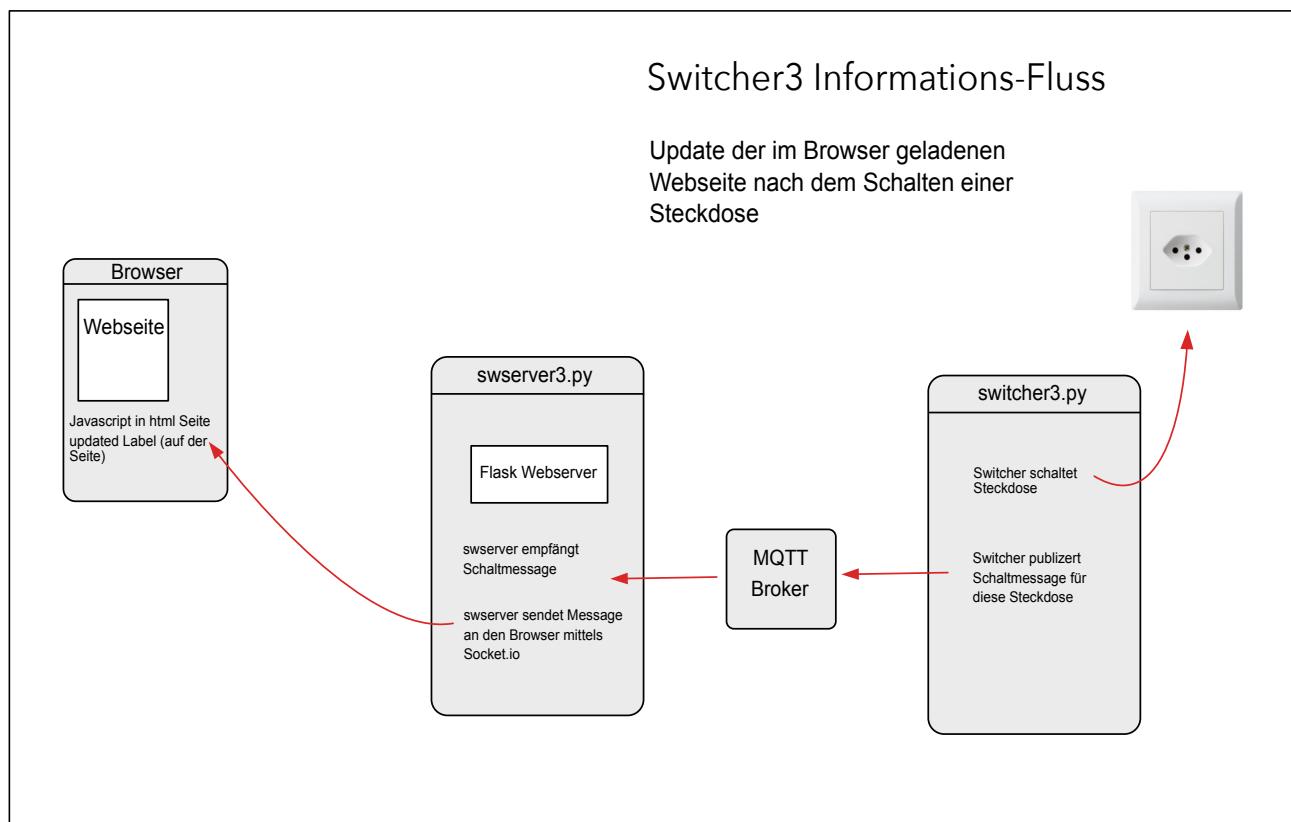
Der Switcher 3 verfügt über ein WebInterface, auf welchem Informationen über den aktuellen

Schaltzustand der Steckdosen angezeigt werden. Dieses WebInterface ermöglicht folgende Funktionen:

- Anzeige und Änderung der Schaltzustände der Steckdosen
- Anzeige der vergangenen/zukünftigen Schaltvorgänge des aktuellen Tages
- Anzeige und Änderung des Home Status (Jemand Zuhause/Niemand Zuhause)
- Anzeige der Wetterdaten
- Anzeige des Switcher 3 Logfiles

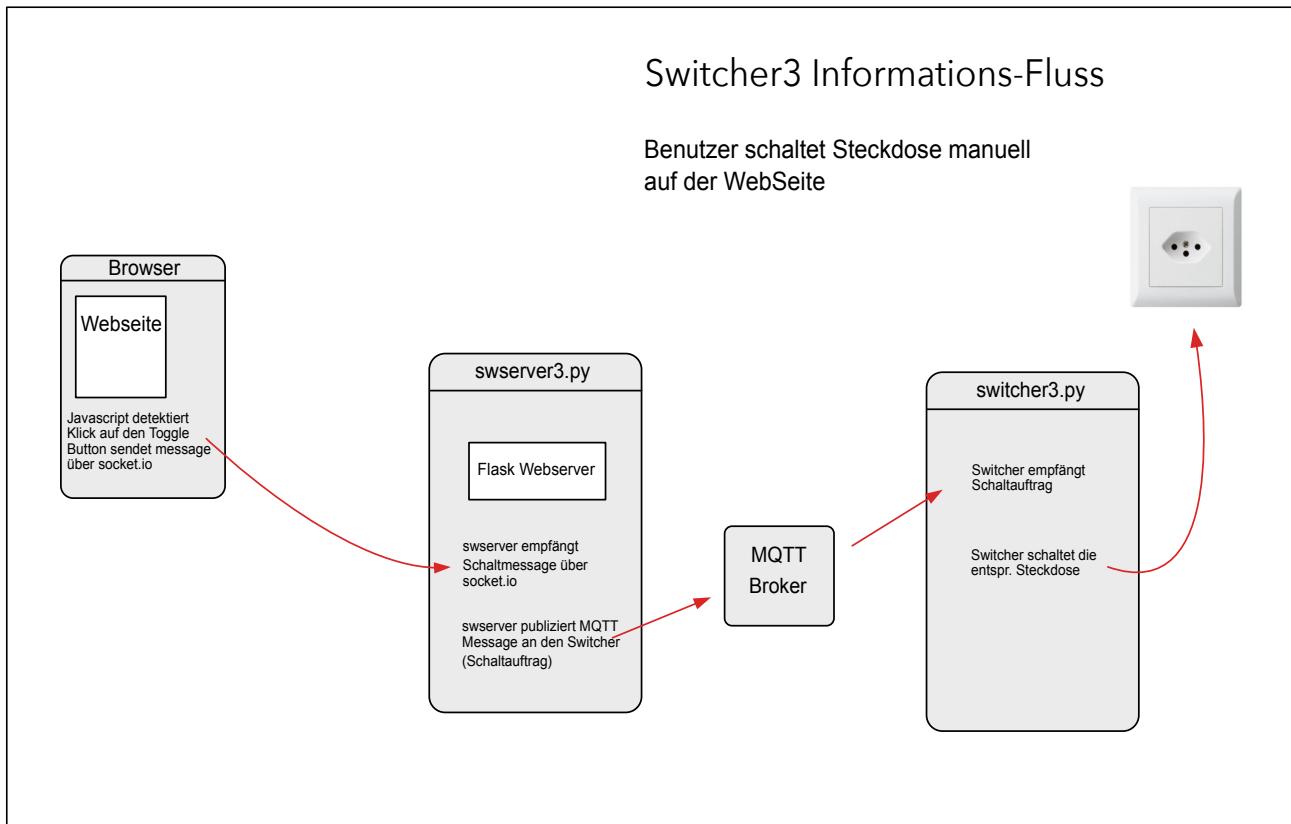
Es ist also möglich, auf dem WebInterface (auf der Webseite im Browser) die Steckdosen manuell zu schalten. Eine erfolgte Schaltung einer Steckdose wird in real-time (mit kurzer Verzögerung) auf der Seite angezeigt - die Webseite ist NICHT neu zu laden. Dies gilt für manuelle Schaltung der Steckdose, als auch für automatische Schaltungen der Dosen. Es ist also möglich, die Webseite einmal zu laden und es sind alle Schaltvorgänge direkt zu sehen. Dies ist eine sehr benutzerfreundliche Lösung.

Der Informationsfluss im Falle einer vom Switcher automatisch geschalteten Dose ist in folgendem Schema ersichtlich.



Der Switcher schaltet Dose und informiert die Webseite über die Änderung.

Falls der Benutzer eine Steckdose auf dem WebInterface manuell schaltet (Klick auf dem Toggle Button), so findet folgender Informationsfluss statt:



Benutzer schaltet Dose manuell auf WebInterface

Switcher 3 und Swserver kommunizieren via MQTT, zwischen dem swserver3.py und dem Browser (resp. der Javascript-Engine des Browsers) wird über Socket.io kommuniziert.

Besonderheit: wird die Homepage des Switchers neu geladen (durch den Benutzer im Browser), so wird eine vordefinierte html-Seite geladen (index.html). Alle Steckdosen sind statisch im Zustand AUS definiert. Sobald der Browser die Seite ganz geladen hat, wird eine Socket.io Message an den swserver3.py gesandt (initial, was etwa heisst: bitte gebe mir alle aktuellen Schaltzustände, damit ich die Webseite anpassen kann).

Dieser Auftrag wird per MQTT Message an den Switcher übermittelt und dieser holt via Funktionsaufruf an die Klasse SwHome alle Infos und gibt sie via MQTT an den swserver zurück. Dieser wiederum sendet socket.io Messages an den Browser und die Javascript Engine updated die Webseite - dann werden die aktuellen Schaltzustände der Dosen erkennbar, ebenso der Status Home.

Diese Lösung scheint komplex, erspart aber jede serverbasierte Modifikation der Webseite - zudem werden Änderungen in (beinahe) real-time angezeigt.

8.2 Besonderheit: Webseiten Laden

Wird im Browser eine der Webseiten des Switchers aufgerufen (im Menu: entweder Home, Aktionen, Info oder Wetter), so wird der Request durch den Server (swserver3.py) in einer Function app.route empfangen und der Request wird an den Switcher mittels MQTT-MESSAGE weitergereicht. Der Switcher stellt die notwendigen Daten zusammen und sendet eine entsprechende MQTT-MESSAGE an den Server zurück. Der Server gibt dann die html-Seite aus (mittels dem Flask-Webserver).

Da aber MQTT-Messages per Definition **asynchron** sind, ist dies nicht so einfach zu implementieren.

MQTT is based on asynchronous messaging that follows the publish-subscribe paradigm: Senders and receivers are decoupled from one another in synchronicity, time, and space and one-to-many relationships are possible

In der MQTT Spezifikation Version 5 wird Request-Response definiert, aber:

It's important to understand that the request-response pattern of MQTT functions and solves problems in a different way than synchronous, one-to-one based protocols like HTTP. An MQTT response usually doesn't "answer" a "question" that the request presents. It is possible to implement a use case for MQTT in a way that is blocking and provides one-to-one messaging that responds with specific information based on parameters of the request. However, in most use cases, the request causes a specific action for the receiver and the response contains the result for this action

Switcher wird aber **nicht** mit Version 5 realisiert (keine neue Paho Library gegenüber Switcher 2). Aus diesem Grund ist Request-Response 'manuell' implementiert in der Klasse **SwConnector**. Bei einem synchronen Message-Austausch enthält die Message (publiziert vom Server) eine **Message-Nummer** (wird vorgängig incrementiert) und ein **Response-Topic** und den **Request** (home,aliste,info,wetter). Der Switcher bekommt diese sog. Sync Meldung, stellt die Daten zur Verfügung und publiziert die Antwort an das in der Meldung gefundene **Response-Topic**. Der Server erhält die Sync Meldung in einem speziellen Sync-Callback, er prüft die Message-Number, sie muss identisch sein. Kommt innerhalb einer festgelegten Zeit keine Meldung zurück, so stellt der Server einen Time-out fest.

Diese **krude** Request-Response Implementierung funktioniert ok, wird aber scheitern, bei mehr als einem Client - also wenn mehrer User gleichzeitig das Switcher WebInterface aufrufen.

9. Weitere Info

9.1 Switcher LED's

Falls gewünscht, können im Switcher 2 LED's angesteuert werden:

- Zuhause LED, leuchtet, wenn jemand zuhause ist
- Blink LED, blinkt alle 2 Sekunden 2 mal ganz kurz (I am alive)

Im Config File ist dabei der verwendete GPIO Pin (Nummer) anzugeben. Ist der Wert im Config File 0, so wird kein GPIO Pin aufgesetzt.

Die Funktion `blink_led()` läuft im Switcher3 als eigener Thread, damit die Wartezeit beim Blinken zeitlich den Main-Loop nicht beeinträchtigt. Der Thread ist als Daemon-Thread aufgesetzt, damit er beim Beenden des Switchers sicher auch beendet wird.

9.2 Debugging, Logging

Wie schon erwähnt, werden in den Switcher Programmen keine 'gewöhnlichen' `print()` Statements verwendet - für Ausgaben auf dem Terminal und in den Logfile wird die Klasse **MyPrint** verwendet. Der sog. DEBUGLEVEL gibt dabei an, wie wichtig die entsprechende Meldung ist. Siehe Anhang für Details der **MyPrint** Klasse. Dies ist der Debug-Output, der auf 2 Arten kontrolliert werden kann:

- Durch Commandline Parameter `-d`, `-D` oder `-A`, **oder**
- Durch einen Eintrag im KonfigFile des Switchers **swconfig.ini**. Siehe Sektion [switcher], value debug (kann sein: `debug = 0|1|2|3`)

Mit anderen Worten: In Entwicklung und Test werden die Switcher Programme auf der Commandline

(im Terminal) aufgerufen, dabei kann der debug Parameter spezifiziert werden. Läuft der Switcher jedoch im Normalbetrieb, so wird er nach dem Boot ohne Commandline Parm gestartet. In diesem Fall kann durch Angabe im ConfigFile auch Debug-Output verlangt werden.

Note: zusätzlich gibt es im Konfigfile einen weiteren Debug Eintrag in der Sektion [dose], value debug_schalt = 1 oder debug_schalt = 0. Wert 1 führt dazu, dass alle Schaltaktionen der Klasse Dose und der Klasse Aktor_n im Log protokolliert werden. Dies ist defaultmäßig aktiviert also Wert = 1.

9.3 Threading im Switcher3

Dies wird nur im Programm switcher3.py verwendet. Die Klasse mySequencer läuft als eigener Thread, ebenso die Klasse My_Blink.

Klasse MySequncer beendet sich bei einem ernsthaften Fehler (durch try: except:). Das Hauptprogramm switcher3.py prüft regelmäßig, ob der Thread MySequencer noch aktiv ist - falls nicht, wird der Switcher beendet - kann nicht weiterfahren ohne den Sequencer.

Alle MQTT Callback Functions (durch Subscribe spezifiziert) laufen je in eigenem Thread.

9.4 Implementierung Wetterdaten

Im Konfig-File des Switchers kann spezifiziert werden, ob Wetterdaten empfangen und angezeigt werden sollen.

Die Temperatur/Feuchtigkeites-Sensor-Module sind mittels ESP8266 (Module E-12E) und dem Sensor BME280 gebaut. Ziel war dabei möglichst kleiner Stromverbrauch - Batteriebetrieb. Die endgültige Schaltung konsumiert im deep-sleep grad mal 12 microAmpere. Der Code verwendet Watchdogs und Deep-Sleep, aus dem er alle 30 Minuten aufwacht, den Sensort liest und die Werte zum MQTT Broker publiziert. Der Sketch swesp_read_sensor_1.ino findet sich im Source Code Switcher 2, Ordner esp_code.

Indoor und Outdoor Sensor werden mit demselben Sketch geladen - Pin14 (High/Low) wird im Sketch abgefragt und dies definiert Indoor oder Outdoor.

Der Sketch misst die Batteriespannung mittels eines Spannungsteilers, der im Deep-Sleep Modus inaktiviert wird.

Zum Switcher 3 gesendet werden folgende Daten:

- Batterie-Status (Beispiel): „3.5 V - perfekt“
- Sensor-Status: Angabe, ob der Sensor ok gelesen werden kann
- Elapsed Time: Anzahl Millisekunden, die der Wachcycle gedauert hat
- Temperatur/Feuchtigkeit: 23.50/27.60

Folgende Tabelle zeigt MQTT Topic und Payload, die verwendet werden

Projekt Switcher 3	Sketch Indoor sendet (publish) Topic	Payload	Seite
	switcher2/wetter/data	indoor/battstatus/sensorstatus/elapsed_time_ms/TEMP/HUM	22
Sketch Outdoor sendet (publish) Topic	Payload		

9.5 XML Steuerfile

Die Schaltzeiten für alle Dosen und die 7 Wochentage sind im XML Steuerfile definiert. Ebenso sind die Bezeichnungen der Dosen hier definiert - also die Zimmer (zB. Wohnzimmer).

Siehe auch **Kapitel Anzahl der installierten Dosen**.

Dieser Steuerfile wird durch die Klasse ActionList eingelesen und geparsst. Es wird eine Aktionen-Tabelle erstellt, welche alle Schaltvorgänge (EIN/AUS) für jeden Tag der Woche und für jede Dose enthält. Es können beliebig viele Schaltvorgänge pro Dose und Tag definiert werden. Das Wochenprogramm wird endlos wiederholt. Die Dosen sind mit einem Text versehen, der das Zimmer anzeigt, in welchem die Dose sich befindet. Dies ist für die Status-Abfrage wichtig (siehe weiter oben).

Jeder File hat im Element <file_id> eine Beschreibung - damit kann der aktuell gültige File identifiziert werden. Der Steuer-File wird mit einem normalen Editor erstellt - oder die vorhandenen Beispiele werden gemäss den eigenen Anforderungen angepasst. Es ist dabei sorgfältig vorzugehen und die Modifikationen sollten mit dem Prüfscript vorgängig geprüft werden, siehe weiter hinten..

Beispiel eines solchen Files findet sich im Anhang.

9.5.1 Anpassung der Schaltzeiten

Die Schaltzeiten im Steuerfile sind für den Sommer (Woche 27) anzugeben. Der bestehende Steuerfile **haus_fr1.xml** berücksichtigt die aktuell gültige Sommerzeit und die geographische Breite der Schweiz. Wird die Sommerzeit abgeschafft, sind die Schaltzeiten im Steuerfile wohl etwas anzupassen.

Im Switcher 3 werden die definierten Schaltzeiten immer um Mitternacht ajustiert (verändert).

Es gibt 2 Arten von Veränderung der Schaltzeiten (gegenüber den im Steuerfile definierten Zeiten):

- Anpassung an die im Laufe des Jahres kürzeren Tage - also zunehmend früherer Dämmerung gegen den Herbst, resp. zunehmend längerer Tage gegen den Sommer.
- Berücksichtigung der Umschaltung Sommer/Winterzeit.

Beide Anpassungen können im Configfile aktiviert/deaktiviert werden.

Die Anpassung der kürzeren Tage wird abhängig von der aktuellen Woche des Jahres **ajustiert** - dh. 52 mal pro Jahr.

Diese Funktion kann im Configfile ein- oder ausgeschaltet werden (Abschnitt sequencer, Zeile: adjust_needed = 1).

Das Datum der Zeitumstellung Sommer/Winterzeit ist ebenfalls im Configfile definiert (Abschnitt sequencer, Zeile: daylight_saving = 1). Bei Beginn der Winterzeit werden die Schaltvorgänge ON um 60 Min. verändert (Beispiel: 21.00 Uhr ON für den Sommer wird zu 20.00 für den Winter)

Note:

Schaltzeiten zwischen 22.00 Uhr und Mitternacht werden nicht ajustiert (dann ist es im Sommer wie im Winter dunkel). Ebenso gilt für Mitternacht bis 03.00 Uhr. Diese Zeiten sind einstellbar im Config File, Abschnitt sequencer.

Der eingebaute Algorythmus ajustiert die Schaltzeiten wie folgt (siehe Klasse **CalcAdjust** im File **swc_adjust.py**. Ajustierung in Minuten:

```
self.adjust_time = int(abs( 60 * ((self.weekyear - 27) * (.17)) ))
```

Siehe Anhang für eine Liste der Anpassung per Woche des Jahres - im Sommer also Null, im Winter maximal.

Es existiert ein Testprogramm, mit welchem die Liste der Ajustierung in Minuten ausgegeben werden kann. Aufruf wie folgt:

```
python3 swt_adj.py -w
```

Wichtig:

Bei der Codierung der Schaltzeiten in den Steuerfiles gilt folgende Regel: Es darf keine Brenndauer (einer Lampe) **über Mitternacht** definiert werden. Mit anderen Worten: jede Lampe muss spätestens um 23.59 ausgeschaltet werden - und kann um 00.01 wieder eingeschaltet werden. Diese Regel vereinfachte den Programmcode wesentlich und sie stellt keine grosse Einschränkung für den Betrieb des Switcher dar.

9.5.2 Prüfung der Steuerfiles

Der Steuerfile kann mit einem speziellen Testscript `swt_xml.py` geprüft werden - es werden dabei verschiedenste Strukturelemente überprüft und ein so geprüfter Steuerfiles wird im Switcher richtig eingelesen. Also unbedingt vorgängig machen, nachdem Steuerfiles editiert wurden.

Aufruf des Prüfscripts (Beispiel):

```
python3 swt_xm.py -f xml/haus1_sommer.xml
```

10. Anzahl installierte Dosen, Dosenbezeichnungen

Dieses Thema braucht etwas Erklärung: Im XML Steuerfile sind (so wie er geliefert wird) 5 Dosen spezifiziert (ist Maximum für den Switcher3). Zudem sind auch 5 Zimmerbezeichnungen eingetragen. Dies heisst aber nicht unbedingt, dass auch 5 Dosen real existieren. Der **Benutzer** kann die bei ihm **vorhandene Anzahl** auf der Log-Seite des WebInterfaces ändern (von 2 bis 5 sind möglich). Bei einer Änderung wird die neu gewählte Anzahl im File **swdosen.ini** festgehalten.

Bei Start des Switchers wird die Anzahl Dosen dort entnommen und das Programm entspr. konfiguriert - darum ist nach einer Änderung ein Reboot nötig - der wird ausgelöst bei Übernehmen.

Achtung: falls im XML File weniger Dosen definiert sind als in oben erwähnten File, wird die Anzahl der Dosen im XML-File berücksichtigt. Dabei wird eine Meldung im Log abgesetzt. Der Config File wird aber nicht verändert - die Logmeldung erscheint also bei jedem Start.

Aus diesem Grund sollte die Anzahl der Dosen im XML Steuerfile stets 5 sein. Und ebenso: im Switcher Config File `swconfig.ini` sollten auch die Angaben für alle 5 Dosen stehen.

Nach Installieren des Switchers (siehe Install Dokument) müssen im XML Steuerfile die Schaltzeiten und auch die Zimmer- oder Verbraucherbezeichnungen mit einem Texteditor angepasst werden. Es empfiehlt sich, den Steuerfile anschliessend mit dem Prüfprogramm zu testen (siehe oben).



Anzahl Dosen einstellen am WebInterface

11. Betrieb des Switchers

11.1 Start des Switchers

Normalerweise werden die beiden Switcher Programme **beim Boot des Pi gestartet** (Aufruf via systemd, siehe Kapitel weiter hinten). Um den Switcher mit verschiedenen Debug-Outputs zu testen, müssen die Programme in einem Terminal gestartet werden, siehe unten.

11.2 Beenden des Switchers

Ein Python Script (wie jeder andere Linux Process) kann auf verschiedene Arten beendet werden. Für den Switcher3 gilt: wird der Switcher3 beendet, so soll sich er sich möglichst graceful beenden, konkret sollen alle Dosen abgeschaltet werden, die Sockets und GPIO Pins sollen aufgeräumt werden.

Will man die Switcherprogramme manuell beenden, so ist einem Terminal folgendes einzugeben:

```
sudo systemctl stop switcher3.service  
sudo systemctl stop swserver3.service
```

Im Code des Switchers wird dies durch folgende Komponenten abgedeckt:

- Im Programm selbst bei Erreichen irgendeiner Kondition durch `sys.exit(errorcode)`. Mit `errorcode` kann einem eventuellen Shell-Script ein Erricode zurückgegeben werden.
- Durch Ctrl-C im Terminal Fenster - dies allerdings nur, wenn der Process im Vordergrund läuft. Dies ist beim Entwickeln/Testen meist der Fall, da das Script meist auf der Commandline aufgerufen wird. Dies wird in Python durch `try: / KeyboardInterrupt:` behandelt.

In diesbezüglichen Tests zeigt sich, dass beim Reboot des Pi tatsächlich alle in diesem Moment eingeschalteten Dosen abgeschaltet werden. Dasselbe geschieht beim Stop des Switcher3 Daemons

oder bei Ctrl-C im Terminal Fenster.

11.3 Switcher Start bei Boot des Pi

Durch den Einsatz des Python-Logging werden alle Ausgaben des Switchers in die Logfiles geschrieben - siehe Kapitel Logging. Deshalb ist ein Ausführen des Switchers im Hintergrund möglich - ein Linux-Dämon.

Ein Dämon ist in UNIX/LINUX ein Process, der im Hintergrund ohne Verbindung zu einer Console läuft.

Der Switcher 3 und auch der Switcher Webserver sollen beim **Boot des Pi automatisch gestartet** werden - zudem sollen sie beim Runterfahren oder Reboot automatisch beendet werden (graceful termination). Sie laufen dann als Daemons und es gibt zusätzlich auch Commandline-Befehle, mit denen sie gestartet und gestoppt werden können. Dies kann beim Testen hilfreich sein - normalerweise werden die Prozesse beim Boot/Reboot gestartet.

Für Start und Stop der Switcher Programm wird systemd benutzt.

[Systemd siehe hier](#)

Die notwendigen Shell-Scripts sind im Dokument **Switcher3_install.pdf** beschrieben.

Nach der Installation können die beiden Prozesse so started/stopped werden:

```
sudo systemctl start swserver3.service  
sudo systemctl stop swserver3.service  
  
sudo systemctl start switcher3.service  
sudo systemctl stop switcher3.service
```

Bemerkung: der Switcher kann auch manuell mit kill -pid beendet werden - beim Herunterfahren übernimmt dies der init Dienst. Im Switcher-Code wird das Kill Signal abgefangen und alles anständig beendet - Alle Dosen aus, Sockets beenden und so weiter.

Mit folgendem Command können die beiden Prozesse im Pi Terminal angezeigt werden:

```
ps -ax | grep python3
```

11.4 Hinweise für Testen des Switchers

Normalerweise werden die beiden Switcher Programme **beim Boot des Pi gestartet** (Aufruf via systemd, siehe Kapitel weiter hinten).

Für Test und Debug werden die Programme jedoch in einem Terminal auf der Commandline gestartet. Man kann entweder nur den Server oder nur den Switcher so testen. In diesem Fall sind die Programme (das eine, das andere oder beide), die seit dem Boot des Pi laufen zu **beenden**. Folgenden Command eingeben:

```
sudo systemctl stop swserver3.service  
sudo systemctl stop switcher3.service
```

Nun können die Programme im Foreground per Commandline gestartet werden:

```
python3 switcher3.py [- Commandline Parameter]  
python3 swserver3.py [- Commandline Parameter]
```

Beim Start der Programme können folgenden Commandline-Parameter spezifiziert werden:

```
-d      kleiner debug, Statusmeldungen werden ausgegeben (stdout)
-D      grosser debug, weitere Statusmeldungen
-A      ganz grosser debug, weitere Statusmeldungen
-h      help, usage wird ausgegeben
```

Für einen Test mit dem Testboard (5 Led's) ist im File swconfig.ini lediglich diese Zeile anzupassen (Nein durch Ja ersetzen), dann werden die Schaltarten aller Dosen beim Start auf 1 gesetzt (Schalten der Led auf dem Testboard).

```
testmode = Ja
```

12. MQTT und Switcher 3

Auf dem Pi wird der MQTT mosquitto verwendet. Siehe Doku **Switcher3_install.pdf** für eine Installations-Anleitung. Der Broker benützt Port 1883. Es gibt folgende Commandi für mosquitto Broker:

```
sudo service mosquitto stop
sudo service mosquitto start
sudo service mosquitto restart
```

Prüfen ob ok läuft mit diesem Command:

```
netstat -tln | grep 1883
```

Resultat sollte sein:

```
tcp      0      0 0.0.0.0:1883      0.0.0.0:*      LISTEN
```

12.1

12.2 Fehlerbehandlung bei MQTT Fehlern

In der ersten Version von Switcher 3 gilt folgendes (wird ev. später verbessert):

- Das Programm switcher3 beendet sich gracefully, wenn **beim Start** keine MQTT Connection hergestellt werden kann.
- Das Programm swserver3 läuft auch, wenn beim Start keine MQTT Connection hergestellt werden kann. Damit ist sichergestellt, dass Aufrufe im Browser immer ein Resultat ergeben. Vor jeder Kontaktaufnahme (via MQTT) mit dem Programm switcher3 wird vom Server geprüft, ob MQTT verfügbar ist. Ist dies nicht der Fall, wird eine **Errorseite** ausgegeben.
- Die Log Seite kann **immer** aufgerufen werden, da zum Anzeigen des Logfiles keine MQTT Verbindung notwenig ist.
- Auf der Logseite ist ein Reboot des Pi möglich.

12.3 Konfiguration für Schaltart 3

Werden Smart Switches verwendet (Schaltart 3), können im Config File (Abschnitt [dose]) zwei weitere

Parameter angegeben werden - dies kann so aussehen (Beschreibung Config File siehe weiter hinten):

`dose_3_schaltart = Wert1 , Wert2 , Wert3`

- **Wert1:** die Schaltart der Dosen (3 für Smart Switches).
- **Wert2:** Definiert das Format von Topic- und Payloads-Strings der MQTT Meldung. Es können damit verschiedene Smart Switch Produkte angesteuert werden. Klasse Aktor_3 muss gemäss den Anforderungen angepasst werden! Zur Zeit sind implementiert in Klasse Aktor_3:
 - 1: Topic und Payload für Testaufbau mit 2 ESP8266, siehe unten) und für Sonoff/Shelly Switches, die Tamosa Firmware enthalten. Konfiguration der Sonoff Switches siehe weiter hinten.
 - 2: not used
- **Wert3:** Definiert, ob der Smart Switch MQTT Meldungen sendet (publiziert), dann kann in der Klasse Aktor_3 ein Subscribe mit Callback abgesetzt werden.

Ein erster Test, ob der Broker ok funktioniert, kann mit den beiden Testprogrammen **swt_mqtt_pub.py** (Publisher) und **swt_mqtt_sub.py** (Subscriber) probiert werden: den Subscriber zuerst starten, danach in zweiter Console den Publisher starten - dieser publiziert genau eine Meldung.

Verwendetes Default Topic ist ‚switcher3‘. Mit Cmd-Line Parm -t kann dies geändert werden.
Wenn dies funktioniert, wird auch der Switcher 3 die Dosen korrekt schalten.

Switcher 3 wurde mit mosquitto auf einen Pi Zero W erfolgreich getestet - der Winzling schafft das.

12.4 Mosquitto Stuff

Weitere Info zu Install und Setup mosquitto sind zu finden im Dokument **Switcher3_install.pdf**.

Den Logfile von mosquitto findet man in folgendem Ordner, bei Problemen lohnt es sich, da mal reinzuschauen mit:

`tail /var/log/mosquitto/mosquitto.log`

[Hier](#) ein Artikel im Netz, welcher das Logging von Mosquitto beschreibt.

12.5 Debugging MQTT

Um den MQTT Message-Austausch zwischen Komponenten (hier switcher3 und swserver3) zu überwachen/inspizieren gibt es diverse hilfreiche Apps, sowohl für Mac/PC, als auch für Mobile Devices.

Sehr geeignet für Inspizieren des Brokers ist die App **MQTT Explorer**, die es für verschiedene OS gibt (Windows, Mac, Linux). Hier Link:

[MQTT Explorer](#)

Es ist auch sinnvoll, auf einem Mobile Device eine MQTT App zu haben. Damit kann der Broker geprüft werden - auch können publish und subscribe abgesetzt werden.

- Ich verwende auf dem iPad eine iOS App genannt **MQTT Inspector**. Damit können Publish und Subscribe gemacht werden und es können beliebige Topics/Payloads gesetzt werden. Damit ist es einfach, den Smart Switch zu testen.

- Für Android (Google PlayStore) ist es die App **MyMQTT**, die dafür gut geeignet ist.

13. Konfiguration der Sonoff/Shelly Smart Switches

Man muss entweder Sonoff Switches mit bereits geladener Tasmota Firmware kaufen oder die Switches selbst flashen. Dies gilt auch für die kleineren Shelly 1 Schalter. Dazu gibt es im Netz und auf YouTube genügend Anleitungen.

Für Switcher3 gibt es ein separates Dokument, welches das Vorgehen detailliert beschreibt. Siehe Dokument **Setup_Sonoff_Devices.pdf**.

Prinzipiell gilt: Diese Smart Switches werden durch eine Meldung via MQTT Broker ein/ausgeschaltet. Zudem können sie auch manuell geschaltet werden (Druck auf Taste) - in diesem Fall senden sie eine Meldung über den neuen Zustand ebenfalls via MQTT Broker. Damit kann der Schaltzustand im Switcher nachgeführt werden. Das verwendete Topic muss in den Smart Switches eingestellt werden. Für Gebrauch mit Switcher3 gilt folgende Tabelle (N steht für Nummer der Dose):

Switcher 3 sendet (publish) Topic	Payload
cmnd/doseN/POWER	ON oder OFF
Smart Switch sendet (publish) Topic	Payload
stat/doseN/POWER	ON oder OFF

Note:

Smart Switches werden bei manuellem Ein/Ausschalten (am Switch selbst, durch Druck auf Taste) also Ein- oder Ausschalten und sie publizieren eine MQTT Message mit Topic **stat/doseN/POWER** und Payload **ON** oder **OFF**. Diese Meldung wird mittels eines Subscribe im actor_3.py empfangen und in der Dosenklasse verarbeitet. Auch das WebInterface wird entsprechend angepasst. Very Cool.

Die Switches mit Tasmota Firmware sind vor Gebrauch mit den Switcher 3 zu konfigurieren: zuerst WLAN Access (fürs Einbinden ins eigene lokale Netz) und danach (mindestens) die MQTT Parameter.

Siehe dazu das Dokument **Setup_Sonoff_Devices.pdf**

Note:

Nur mit diesen Werten kann ein Smart Switch korrekt die vom Switcher 3 publizierte Meldung empfangen und den Verbraucher schalten.

Note 2:

Auch hier können die beiden Programme **swt_mqtt_sub.py** und **swt_mqtt_sub.py** helfen bei der Fehlersuche.

Beispiel: Mit diesem Aufruf kann ein Sonoff- oder Shelly-Switch (der als Dose 4 konfiguriert ist) ein/ausgeschaltet werden:

MQTT Topic: cmnd/dose4/POWER

Payload: ON

Auf dem Pi mittels der genannten Scripts siehr das so aus:

Einschalten

```
python swmqtt_pub.py -t ,cmnd/dose4/POWER' -p ,ON'
```

Output des Scripts:

```
Switcher Test MQTT Publisher,using IP_ADR:192.168.1.125 and Port:1883  
Connected...  
Publish Topic: cmnd/dose4/POWER  
Publish Payload: ON  
Message Published...
```

Ausschalten:

```
python swmqtt_pub.py -t ,cmnd/dose4/POWER' -p ,OFF'
```

--> Alles weiter bitte selbst im Netz suchen - there is tons of stuff out there.

13.1 Wichtige Bemerkung

Ich habe bemerkt, dass sich die Sonoff Smart Switches mit Tasmota Software **manchmal** (selten) nicht neu zum MQTT Broker connecten (reconnect), **falls der Pi zwischenzeitlich rebootet wurde** - also der MQTT Broker kürzer oder länger nicht zu erreichen war. Dann wird nicht mehr geschaltet, die Meldungen erreichen den Smart Switch nicht mehr.

Siehe ausführliche Bemerkungen dazu im Dokument **Setup_Sonoff_Devices.pdf**.

14. Anhang

14.1 XML Steuerfile

14.1.1 Auszug XML Steuerfile

```
<aktionen>
<file_id>Sommer Test 5 Devices</file_id>
<!--
device Nummer 1 Wohnzimmer device 1
-->

<device name = "Wohnzimmer">
<device_nr> 1 </device_nr>

<tag nummer = "0">
<sequence>
<ON>00.15</ON>
<OFF>00.25</OFF>
</sequence>
<sequence>
<ON>04.30</ON>
<OFF>04.55</OFF>
</sequence>
<sequence>
<ON>21.25</ON>
<OFF>22.05</OFF>
</sequence>
<sequence>
<ON>22.40</ON>
<OFF>23.55</OFF>
</sequence>
</tag>

<tag nummer = "1">
<sequence>
<ON>00.15</ON>
<OFF>00.25</OFF>
</sequence>

<sequence>
<ON>22.35</ON>
<OFF>22.58</OFF>
</sequence>
<sequence>
<ON>23.15</ON>
<OFF>23.42</OFF>
</sequence>
</tag>

<tag nummer = "2">
<sequence>
<ON>01.15</ON>
<OFF>01.25</OFF>
</sequence>
....und so weiter...
```

14.1.2 Ajustierung Schaltzeiten

Diese Liste zeigt die Ajustierung der Schaltzeiten in Minuten in Abhängigkeit der Woches des Jahres (Output Testprogramm swt_adj.py) Im Sommer, Woche 27, beträgt die Ajustierungen 0 Minuten.

Beispiel Schaltzeit sei 21.59 Uhr.

Aufruf mit:

```
python3 swt_adj.py -w
```

```
CalcAdjust evstart:1320 evon:15 mostart:180 moon:30 faktor:0.17
week: 1, adjust: 265 new time: 17.34 (from 21.59)
week: 2, adjust: 255 new time: 17.44 (from 21.59)
week: 3, adjust: 244 new time: 17.55 (from 21.59)
week: 4, adjust: 234 new time: 18.05 (from 21.59)
week: 5, adjust: 224 new time: 18.15 (from 21.59)
```

```
week:  6, adjust: 214 new time: 18.25 (from 21.59)
week:  7, adjust: 204 new time: 18.35 (from 21.59)
week:  8, adjust: 193 new time: 18.46 (from 21.59)
week:  9, adjust: 183 new time: 18.56 (from 21.59)
week: 10, adjust: 173 new time: 19.06 (from 21.59)
week: 11, adjust: 163 new time: 19.16 (from 21.59)
week: 12, adjust: 153 new time: 19.26 (from 21.59)
week: 13, adjust: 142 new time: 19.37 (from 21.59)
week: 14, adjust: 132 new time: 19.47 (from 21.59)
week: 15, adjust: 122 new time: 19.57 (from 21.59)
week: 16, adjust: 112 new time: 20.07 (from 21.59)
week: 17, adjust: 102 new time: 20.17 (from 21.59)
week: 18, adjust:  91 new time: 20.28 (from 21.59)
week: 19, adjust:  81 new time: 20.38 (from 21.59)
week: 20, adjust:  71 new time: 20.48 (from 21.59)
week: 21, adjust:  61 new time: 20.58 (from 21.59)
week: 22, adjust:  51 new time: 21.08 (from 21.59)
week: 23, adjust:  40 new time: 21.19 (from 21.59)
week: 24, adjust:  30 new time: 21.29 (from 21.59)
week: 25, adjust:  20 new time: 21.39 (from 21.59)
week: 26, adjust:  10 new time: 21.49 (from 21.59)
week: 27, adjust:   0 new time: 21.59 (from 21.59)
week: 28, adjust:  10 new time: 21.49 (from 21.59)
week: 29, adjust:  20 new time: 21.39 (from 21.59)
week: 30, adjust:  30 new time: 21.29 (from 21.59)
week: 31, adjust:  40 new time: 21.19 (from 21.59)
week: 32, adjust:  51 new time: 21.08 (from 21.59)
week: 33, adjust:  61 new time: 20.58 (from 21.59)
week: 34, adjust:  71 new time: 20.48 (from 21.59)
week: 35, adjust:  81 new time: 20.38 (from 21.59)
week: 36, adjust:  91 new time: 20.28 (from 21.59)
week: 37, adjust: 102 new time: 20.17 (from 21.59)
week: 38, adjust: 112 new time: 20.07 (from 21.59)
week: 39, adjust: 122 new time: 19.57 (from 21.59)
week: 40, adjust: 132 new time: 19.47 (from 21.59)
week: 41, adjust: 142 new time: 19.37 (from 21.59)
week: 42, adjust: 153 new time: 19.26 (from 21.59)
week: 43, adjust: 163 new time: 19.16 (from 21.59)
week: 44, adjust: 173 new time: 19.06 (from 21.59)
week: 45, adjust: 183 new time: 18.56 (from 21.59)
week: 46, adjust: 193 new time: 18.46 (from 21.59)
week: 47, adjust: 204 new time: 18.35 (from 21.59)
week: 48, adjust: 214 new time: 18.25 (from 21.59)
week: 49, adjust: 224 new time: 18.15 (from 21.59)
week: 50, adjust: 234 new time: 18.05 (from 21.59)
week: 51, adjust: 244 new time: 17.55 (from 21.59)
```

14.2 Config-File

Verschiedene Parameter für das Switcher Programmsystem sind in einem Config-File ausgelagert. Die verschiedenen Komponenten/Klassen lesen ihren **eigenen Abschnitt** in diesem File. Dies wird mittels der Klasse ConfigRead vorgenommen. Der Konfigfile ist vom Typ .ini.

Im File müssen folgende **Abschnitte** vorhanden sein: [switcher], [home], [sequencer], [mqqt], [dose] und [aktor_1], [aktor_2], [aktor_3], [aktor_4] und [aktor_5]

Der Abschnitt [switcher] wird vom Switcher3.py benutzt, der Abschnitt [dose] von der Klasse Dose und

die Abschnitte [aktor_n] von den Aktor-Klassen (und so weiter).

Der Config-File sieht so aus (Ausschnitt)

```
#  
# Configfile for Application Switcher3  
#  
[switcher]  
# Dieser Abschnitt wird vom Switcher2 und seinen Klassen gelesen  
# GPIO Pin für Blink LED, 0: kein PIN verwendet  
gpio_blink = 5  
# testmode Ja/Nein  
testmode = Nein  
reserve = notused  
wetter = Nein  
  
[home]  
# dieser Abschnitt wird von der Klasse MyHome gelesen  
#  
# GPIO PIN für zuhause Led, 0: kein PIN verwendet  
gpio_home_led = 6  
# GPIO Pin zuhause Kippschalter (Switcher V1) , 0: kein PIN verwendet  
gpio_home_switch = 0  
# GPIO Pin zuhause Pushbutton, 0: kein PIN verwendet  
gpio_home_button = 13  
  
# 5 binary Schalter  
schalter = 00001  
# manuell_reset: wann sollen manuell geschaltete Dosen zurückgesetzt werden  
# 1 : Mitternacht, 0: Nie  
manuell_reset = 1  
  
[sequencer]  
# XML Steuerfile name  
ctrl_file = swhaus2  
# Info für das ajustieren der Schaltzeiten  
adjust_needed = 1  
evening_start_limit = 22.00  
evening_ontime = 15  
morning_start_limit = 03.00  
morning_ontime = 30  
  
[mqtt]  
# Switcher2 als IoT Gateway als MQTT publisher  
# IP-Adr des MQTT Brokers (empty means: this maschine)  
mqtt_ipadr =  
mqtt_port = 1883  
mqtt_keepalive_intervall = 45  
mqtt_userid = test127  
mqtt_pw = 123-123  
mqtt_qos = 0  
mqtt_retain = 0  
retry_intervall = 4  
retry_number = 3  
userdata = "userdata"  
#  
[dose]  
# dosenmodus  
# modus 0: dose wir nur manuell geschaltet, Schaltaktionen im XML File werden  
ignoriert  
# modus 1: normal zu schaltende dose, also gemäss Angaben im XML File  
# modus 2: dose wird immer geschaltet, egal ob jemand zuhause ist oder nicht  
dose_1_schalt prio = 1  
dose_2_schalt prio = 1  
dose_3_schalt prio = 1  
dose_4_schalt prio = 1  
dose_5_schalt prio = 1  
#  
# schaltart definiert die Art des Schaltens, entspr. Aktor-Klasse wird instantiiert  
# 1: Für Testaufbau mit 4 Led ,
```

Switcher 3 Beschreibung

```
#      2: Funk mit 433 MHz Sender ,
#      3: IoT MQTT Publish
#      4: Funk mit Handsender (wie original Switcher)
#      5: Funk mit 433 Mhz Sender mit send Module
#
# Achtung:
# Schaltart 1 ist für Testumgebung mit 5 LED.
# Wenn eine dose 1 schaltart 1 hat, müssen die anderen Dosen auch Schaltart 1 haben
# script sorgt dafür (mit Warnung), dass dies eingehalten wird.
# Ansonsten ist schaltart frei wählbar
# Bei Schaltart 3 gibt es zwei weitere Parameter, etwa so: 3,Y,X
# Y: Art der MQTT Meldung, also Topic und Payload
#     (1: Sonoff-SmartSwitches, 2: future use)
# X: Smart Switch sendet Rückmeldung, also Susbcrite möglich/nötig: 0 Nein, 1 Ja
#
# Also: für 'normale' Sonoff Switches 3,1,1 setzen

dose_1_schaltart = 2
dose_2_schaltart = 2
dose_3_schaltart = 2
dose_4_schaltart = 3,1,1
dose_5_schaltart = 3,1,1
#
# debug schalten, die dose setzt statusmeldungen beim schalten ab, obwohl genereller
# dbug 0 ist
# hat sich bewährt beim Testen
debug_schalt = 0

[aktor_1]
# Abschnitt wird von der Aktor_1 Klasse gelesen
# hier sind 5 GPIO Pins definiert
# Für Testumgebung mit 5 Led
gpio_1 = 12
gpio_2 = 19
gpio_3 = 20
gpio_4 = 21
gpio_5 = 24

[aktor_2]
# Abschnitt wird von den Aktor_2 Klasse gelesen
# hier ist ein GPIO Pins definiert (für Data zum 433 Mhz Sender)
# Für Version mit 433 Mhz Sender
gpio_1 = 17
# wiederholung beim Senden
repeat = 10
# code Länge beim Senden
codelength = 24
# dies sind die Parameter fürs Senden mit dem 433 MHz Sender
# ist für 4 Dosen definiert
# Werte wurden mit dem Script rpi-rf_receive.py gemessen ab Handsender
send_dat_1_ein = 66897,320,1
send_dat_1_aus = 66900,319,1
send_dat_2_ein = 69983,319,1
send_dat_2_aus = 69972,319,1
send_dat_3_ein = 70737,319,1
send_dat_3_aus = 70740,319,1
send_dat_4_ein = 70929,320,1
send_dat_4_aus = 70932,319,1

[aktor_3]
# Abschnitt wird von den Aktor_3 Klasse gelesen
# Switcher2 als IoT Gateway als MQTT publisher
# IP-Adr des MQTT Brokers (empty means: this maschine)
# zur Zeit nur ein placeholder
placeholder = leer

[aktor_4]
# Abschnitt wird von den Aktor_4 Klasse gelesen
# hier sind 6 GPIO Pins definiert
# Für Funk mit angeflanschtem Handsender
gpio_1 = 4
gpio_2 = 12
gpio_3 = 77
```

```
gpio_4 = 66
gpio_5 = 17
gpio_6 = 99

[aktor_5]
# Abschnitt wird von den Aktor_5 Klasse gelesen
# Für Funk gemäss Habi
# gpio pin für send module (in wiringpi notation: 0 ist Pin 17)
gpio_send = 0
# system code eingestellt an den Dip Switches
system_code = 11101
# Pfad, wo das send module liegt
pfad_1 = /home/pi/switcher2/433_send/
#
```

```
#-----
```

14.3 Klasse MyPrint

14.3.1 What is the point exactly

While writing Python Code one usually inserts print statements such as this:

```
print ("I am here, doing this")
```

to help finding bugs.

This is a bad idea since one has to remove these statements for the production version of the program. Or make them conditional with something like this:

```
if (debug_variable):
    print ("I am here, doing this")
```

The demo program **swt_pri1.py** demonstrates the use of the **class MyPrint**. Depending on the commandline parameter more or less debug output is written to the console **AND** to the logfile.

So use these commandline parms:

- none
- -d: minimal debug output
- -D: more debug output
- -A: even more debug output

The demo program **swt_pri2.py** also demonstrates the use of the class MyPrint.

The program contains a programing error in this statement: nummer[4]=23

An exception is raised and this fact is also recorded in the logfile.

How to use the better print statement::

Use this for output that you always want to see in the log and on the screen

```
mypri.myprint(DEBUG_LEVEL0, "This Pi's IP-Adress: {}".format(ipadr))
```

Use this for output that only appears with commandline parm -d

```
mypri.myprint(DEBUG_LEVEL1, "I am here")
```

Use this for output that only appears with commandline parm -D

```
mypri.myprint(DEBUG_LEVEL2, "program started")
```

Use this for output that only appears with commandline parm -A

```
mypri.myprint(DEBUG_LEVEL3, "program ended")
```

In other words:

Using commandline parm - A gives you everything, this is for detailed debugging

Using commandline parm - D gives you some details but not everything

Using commandline parm -d gives you just a little more than normal

This, of course, depends on the proper use of the myprint statements and their DEBUG_LEVEL.

Note: I always include class definition files in a folder **sub**. You will find the class definition in the file myprint.py in this folder. This file contains actually two classes: MyLog and MyPrint which inherits from MyLog.

Note on logfiles:

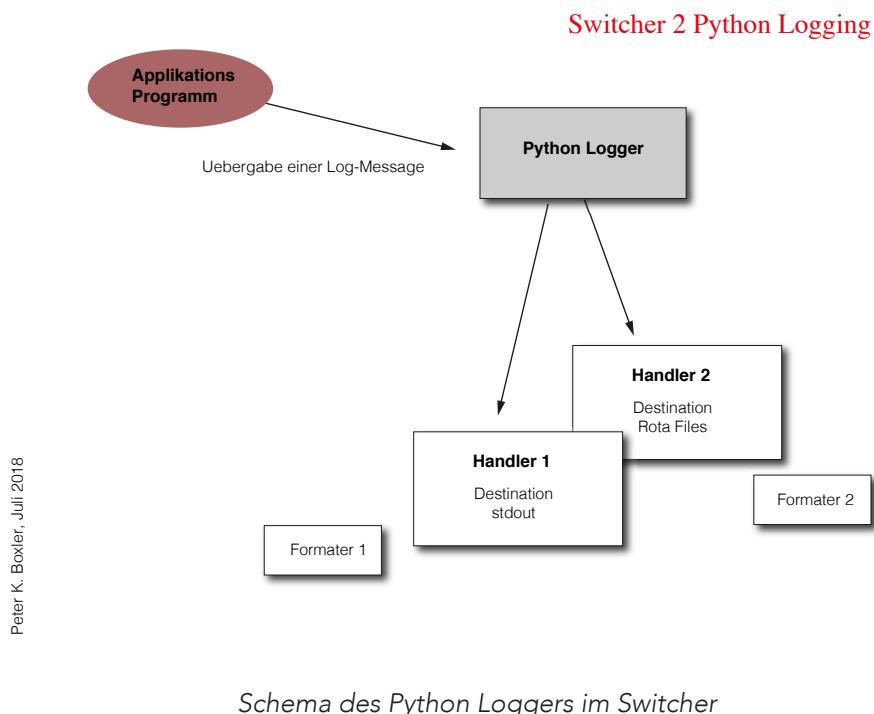
Checkout the docu on logfiles here: The class **MyLog** implements the actual logging.

Die Grösse der Logfile ist in dieser Klasse definiert (siehe File myprint.py) in dieser Zeile:

```
MyLog.filehandler = RotatingFileHandler (filename, maxBytes=300000, backupCount=3)
```

Logging in Python

A maximum of **three** logfiles will be written to the current folder. Since this is a so called **rotating log** the folder will never overflow, each logfile will grow to max 30 kB. See definition in the MyLog class.



Es ist im Switcher Programm ein Logger mit 2 Handlern definiert - je ein Handler für die genannten Destinationen (stdout und Logfile). Jeder Handler hat eigene Regeln für die Formatierung: der Filehandler ergänzt die Meldungen mit Datum/Zeit. Die vom Switcher an den Logger übergebenen Messages mittels MyPrint() Aufrufen werden an die definierten Handler weitergeleitet (das könnten auch mehr als 2 sein).

Wichtig: Print-Statements sind deshalb im im Programm nicht mehr nötig.

Folgende Tabelle zeigt die Zusammenhänge im Switcher-System. Das Python-Logging gliedert die Log-Einträge in verschiedene Level, absteigend nach Wichtigkeit:

CRITICAL	äusserst wichtig
ERROR	sehr wichtig
WARNING	wichtig
INFO:	weniger wichtig
DEBUG:	unwichtig

Wichtig: im Ablauf des Switchers werden alle MyPrint() Aufrufe ausgeführt und die Meldungen dem Logger übergeben. Der Logger wird aber nicht alle Meldungen weiterleiten: es ist dem dem Logger mitzuteilen, ab welchem Level er Einträge berücksichtigen soll. Dies wird mit einem Set-Level Aufruf in der MyPrint Klasse erledigt - aufgrund des Commandline Params beim Aufruf des Switchers. Mit diesem Meccano können wir vollkommen auf Print-Statements fürs Debuggen verzichten.

Die Commandline-Parameter des Switchers sind so zu verstehen:

Commandline Parm	Der Logger macht dies
kein Parameter	will nur die sehr wichtigen Meldungen sehen (Level ERROR und höher), alle anderen Meldungen werden NICHT verarbeitet (weggeworfen).
-d	will die wichtigen und die sehr wichtigen Meldungen sehen, alle anderen Meldungen werden NICHT verarbeitet
-D	will nur die nicht wichtigen, die wichtigen und die sehr wichtigen Meldungen sehen, alle anderen Meldungen werden NICHT verarbeitet.
-A	will alle Meldungen sehen

Aus diesem Grund werden die sehr wichtigen Meldungen (die immer, also auch ohne Commandline-Parm kommen) im Log mit dem Level ERROR angezeigt - das ist leider nicht zu ändern.

Zusammenfassend nochmals diese Tabelle:

MyPrint Aufruf hat Level	Was ist gewünscht	MyPrint Meldung wird ausgegeben bei Parm	Im Log erscheinen folgende Level
DEBUG_LEVEL0	Wichtig, immer sehen	immer ausgegeben	ERROR
DEBUG_LEVEL2	wenig sehen, grobe Sicht auf den Ablauf	Ausgabe bei -d, -D und -A	ERROR und WARNING
DEBUG_LEVEL2	mehr sehen, etwas mehr Details im Ablauf	Ausgabe bei -D und -A	INFO, WARNING und ERROR
DEBUG_LEVEL3	alle Details sehen, gibt viel Output	Ausgabe bei -A	INFO, WARNING, ERROR und DEBUG

Der File-Handler (siehe oben) ist im Switcher so konfiguriert, dass maximal **3 sog. Rotating Files** verwendet werden. Dies bedeutet: wenn die angegebene Grösse des Logfiles überschritten wird, so

wird ein neuer Logfile angelegt und der voll gewordene File bekommt einen neuen Namen. Es gilt dabei: in jedem Moment heisst der aktuelle Logfile immer switcher3.log, der letzte heisst switcher3.log 1 und der vorletzte heisst switcher3.log 2

Auch nach monatelagtem Betrieb des Switchers wird man also immer nur diese 3 Logfiles im Filesystem finden: 2 volle alte Files und der grad aktuelle File. Die maximale Grösse ist im Switcher konfiguriert mit **300 kB pro File**. Es wird also nie zu einem Filesystem-Ueberlauf kommen.

Konklusion: Das Python Logging System macht die Verwendung von File Logs sehr einfach.

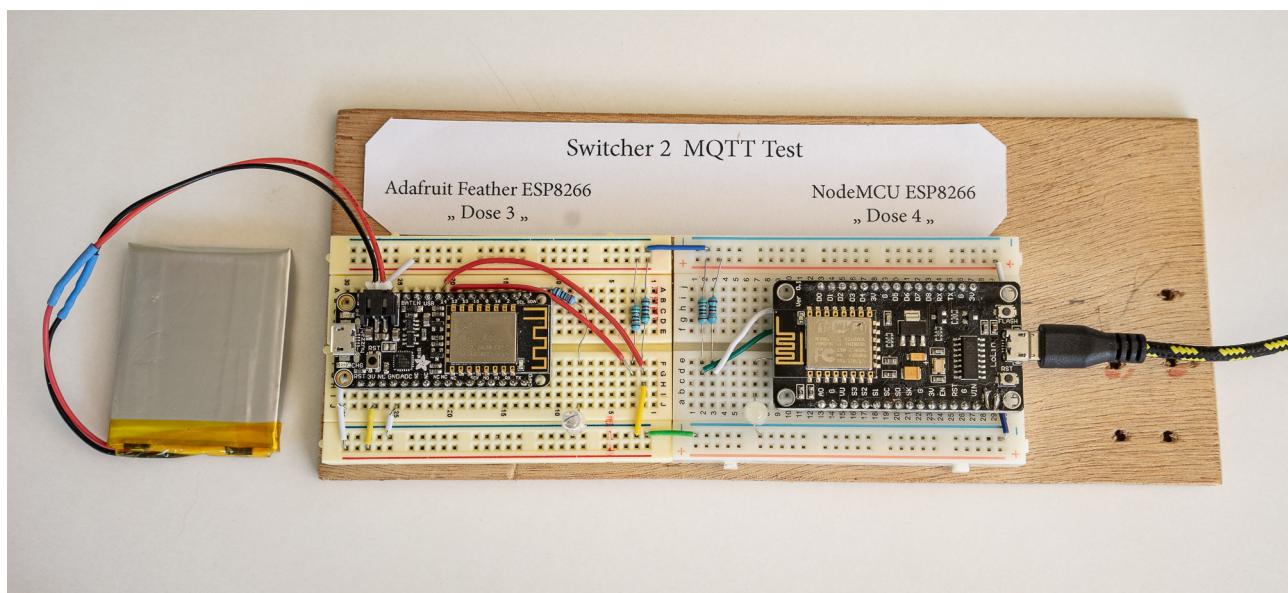
14.4 Testaufbau MQTT

Für einen ersten Test mit MQTT (proof of concept) wurde folgender Testaufbau benutzt, siehe Bild. Dies ist ein einfacher Aufbau mit zwei ESP8266 solche Dosen „simulieren“. Jeder der ESP8266 kann durch Drahtbrücken (oder Dip-Switches) als Dose 1 bis 4 konfiguriert werden. Statt eines 220V Relais wird eine simple LED geschaltet. Im Testaufbau ist links Dose 3 konfiguriert und rechts Dose 4. Entsprechend ist im Config File swconfig.ini im Abschnitt „dose“ die Schaltart dieser beiden Dosen auf 3 gesetzt (die beiden Dosen 1 und 2 werden via 433Mhz Sender angesteuert und haben deshalb Schaltart 2).

Note: hat man Smart Switches (Sonoff) zur Hand, so können diese verwendet werden.

Die beiden simulierten Dosen 3 und 4 werden durch den Switcher 2 via MQTT problemlos geschaltet - sowohl automatische, wie auch manuelle Schaltaktionen. Topic und Payload sind identisch jenen für Smart Switches.

Mittels des Testscripts `swt_mqtt_pub.py` kann die Schaltaktion auch einfach von einer Console ausgelöst werden. Dieses Testscript publiziert dieselben Payloads wie der Switcher 3. (Dosennummer und Ein/Aus). Der Broker läuft auf demselben Pi wie der Switcher selbst.



Testaufbau mit 2 ESP8266 als Dose 3 und 4 konfiguriert

14.4.1 Arduino Sketch für ESP8266

Der Beispiel-Code für ESP8266 wurde mit der Arduino Entwicklungsumgebung erstellt und der identische Code auf die beiden Devices geladen. Der Code swesp_dose_1.ino ist im Ordner esp_code beim Switcher 3 Github zu finden. In jedem Fall ist vor dem Gebrauch folgendes im Code zu ändern:

- Wi-Fi Credentials (SSID und PW) gemäss den lokalen Gegebenheiten anpassen.
- IP-Adr des MQTT Brokers anpassen

Topic und Payload (Wert2, siehe oben) ist in diesem Fall sehr einfach: „ON“ für Dose N einschalten und „OFF“ für Dose X ausschalten.

Die vorgenommenen Schaltaktionen werden jeweils im EEPROM des ESP8266 gespeichert und bei einem Reset oder Stromausfall werden die Dosen beim Setup() gemäss dem letzten Zustand geschaltet.

Der Sketch schreibt viel Output auf den Serial Monitor - ist ja auch blass ein TestSketch, der für Produktion streamlined werden muss - aber er läuft ok.

Der Sketch sendet zudem alle 6 Sekunden eine Meldung an den Broker („bin immer noch da...“) - dies kann für verschiedene Zwecke benutzt werden: Rückmeldung Dosenstatus oder Sensor-Daten melden (falls Smart Switch mit Sensoren ausgestattet ist - Eigenbau ?). Der entsprechende Code im erwähnten Sketch dient lediglich als Beispiel, Topic der publizierten MQTT Meldung ist dabei „cmnd/doseN/Power“, wobei N die Dosennummer ist. Topic und Payload siehe Tabelle weiter hinten.

Diese Meldung kann mit dem Python Script swmqt_sub.py empfangen werden in einem Terminal Fenster auf dem Pi, indem das Script so aufgerufen wird (CmdLine Parameter -t spezifiziert das Topic „switcher2-doseN“):

```
python swmqt_sub.py -t cmnd/dose3/POWER
Output des Scripts sieht so aus:
Switcher Test MQTT Subscribe, using IP_ADR: 192.168.1.125
Using Topic: cmnd/dose3/POWER
Subscribed to MQTT Topic
bin immer noch da...
...
```

Alles weitere ist eigene Handarbeit...

P. K. Boxler, Juni 2021

end of document