Projektbeschreibung Switcher 2

Beschreibung des Raspberry Pi -Projekts

„Licht Schalten“

„Switch Lights When Not At Home“

-> Haus-Automation <-

Version 2.9b

Peter K. Boxler, Januar 2019

Inhaltsverzeichnis

1. Abstract

This documentation describes a Raspberry Pi gadget that is used to switch lights or other appliances on/off when not at home. Up to 4 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 3 XML files. Switcher support 3 seasons and switches unattended to another season if needed. XML files are parsed at reboot. A external switch 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. It can act as an IoT Gateway.

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

A webinterface allows remote control of switcher.
Switcher2 Python Code and this project description is available on GitHub.

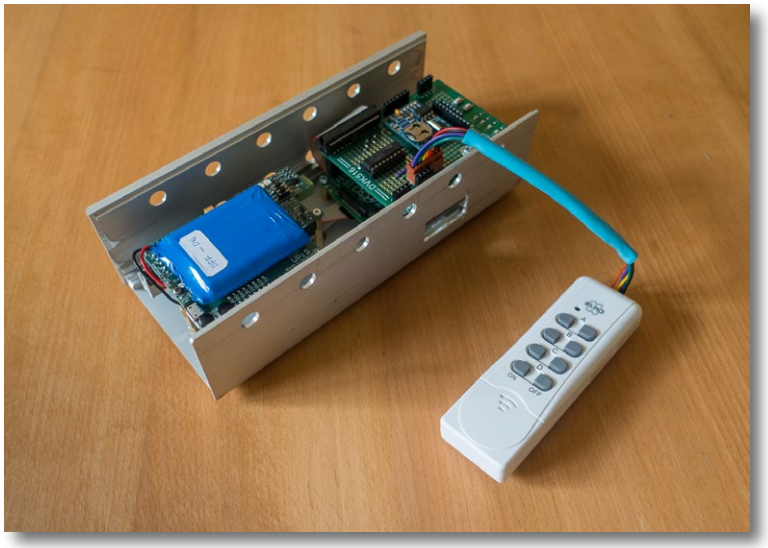
2. Anforderungen

2.1 Ein Rückblick: Switcher Version 1

Switcher Verson 1 wurde in 2014 entwickelt - basierend auf folgenden Anforderungen:

- Bei Ferienabwesenheit sollen 4 Funksteckdosen (Typ ELRO) gemäss einem vorgegebenen Schema ein- und ausgeschaltet werden. Die Funksteckdosen werden in verschiedenen Räumen platziert und die damit individuell geschalteten Lampen sollen Anwesenheit vortäuschen.
- 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 werden können.

Der Aufbau dieser ersten Lösung sieht so aus - ohne Gehäusedeckel. Diese Bilder zeigen die Variante mit Schalten via Funksteckdosen. Der mitgelieferte Handsender wird durch ein Interface vom Code im Raspberry Pi angesteuert.



Switcher Hardware Aufbau (erste Version 2014)

Die 4 schaltbaren Funksteckdosen

2.2 Switcher Version 2

Die erste Version des Switchers erfüllt seit 2014 die gestellten Aufgaben tadellos, durch den produktiven Einsatz zeigten sich aber im Laufe der Zeit Anforderungen und Wünsche und so wurde Mitte 2018 eine Version 2 des Switchers entwickelt: der Switcher 2.

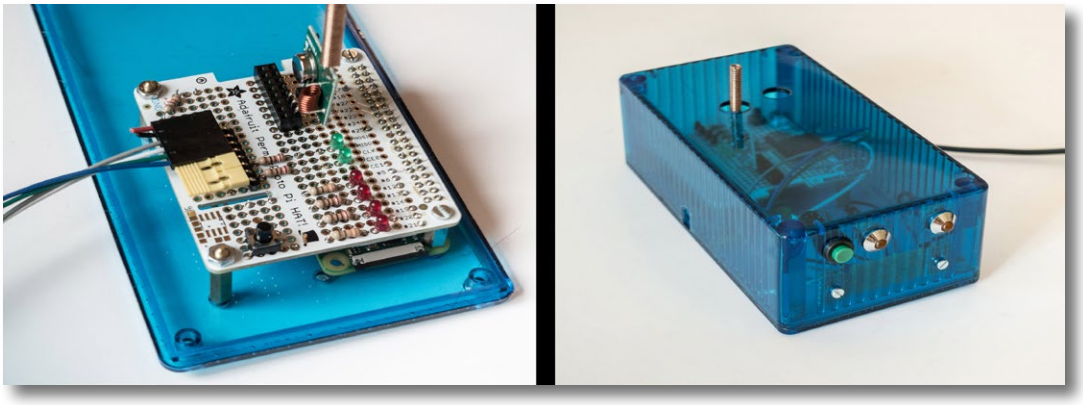
Die Liste der zusätzlichen Anforderungen sieht so aus:

- Switcher2 soll in Richtung **Home-Automation** (Internet of Things) weiter entwickelt werden. Es sollen neben den herkömmlichen 433 Mhz Funksteckdosen auch moderne Wi-Fi Steckdosen (Smart Switch) verwendet werden können (mit MQTT Protokoll).
- Schaltzeiten sollen ja nach Jahreszeit unterschiedlich sein - wie wir wissen, ist es im Sommer lange hell und es bringt nichts, Anwesenheit um 19.30 vorzutäuschen - da ist es eh noch hell draussen. Details der Lösung siehe weiter unten. Automatische Umschaltung für Sommer- und Winterzeit und ebenfalls einer Zwischensaison
- Switcher2 soll unter Python3.x laufen. Durch die in Python3 eingeführte striktere Trennung von Text und Binärdaten sind Teile des alten Codes nicht mehr korrekt lauffähig unter Python 2.x (aus dem Internet: Neu in Python3.
  Zitat: Arguably the most significant new feature of Python 3 is a much cleaner separation between text and binary data. Text is always Unicode and is represented by the str type, and binary data is represented by the bytes type .
- Das bestehende Webinterface soll vereinfacht werden, es soll kein PHP-Programmierung mehr nötig sein. Auch soll das Webinterface responsive sein - also auch auf einem iPhone gut bedienbar sein.
- Die programmierte Implementierung des Schaltens der Steckdosen soll erweiterbar sein - im originalen Switcher ist die Lösung mit Funksteckdosen hardcoded im Switcher-Code. Durch Auslagerung der Schaltlogik in ein separates Python-Modul soll es einfach möglich sein, in Zukunft andere Lösungen zu implementieren. Dies minimiert das Risiko, den Main-Code zu gefährden.
- Der bisherige Code des Switchers basierte auf globalen Variablen, die allen Modulen zugänglich waren. Dies soll im Sinne einer besseren Abgrenzung (divide et impera) durch Anwendung von Objektorientierung massiv verbessert werden. Dies dient der Wartungsfreundlichkeit.
- Die Interprocess-Kommunikation (für die Statusabfrage) soll vereinfacht werden. Der ZeroMQ-Code kann gekürzt werden.
- Die Statusabfrage des Switchers (mittels des Klienten swclient.py) soll um weitere interne Daten

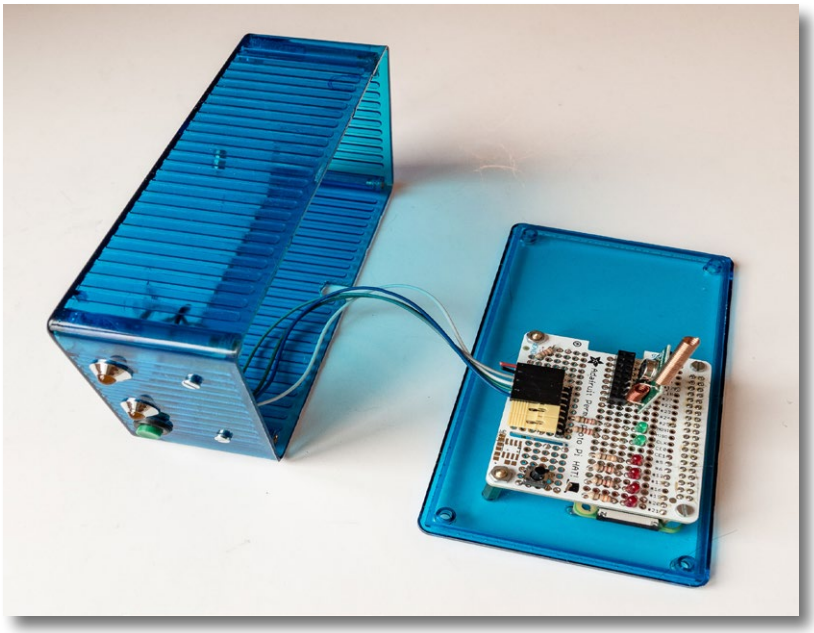
erweitert werden.

- Nach wie vor brauchen total nur 4 Dosen schaltbar sein.
- Es sollen Daten von weiteren MQTT-basierte Klienten angezeigt werden können (Temperatur/ Feuchtigkeit).
- Allgemeine Kosmetik und Vereinfachung im Code und Verbessern der Doku im Code.
- Korrektur der entdeckten Fehler und Mängel im Code.

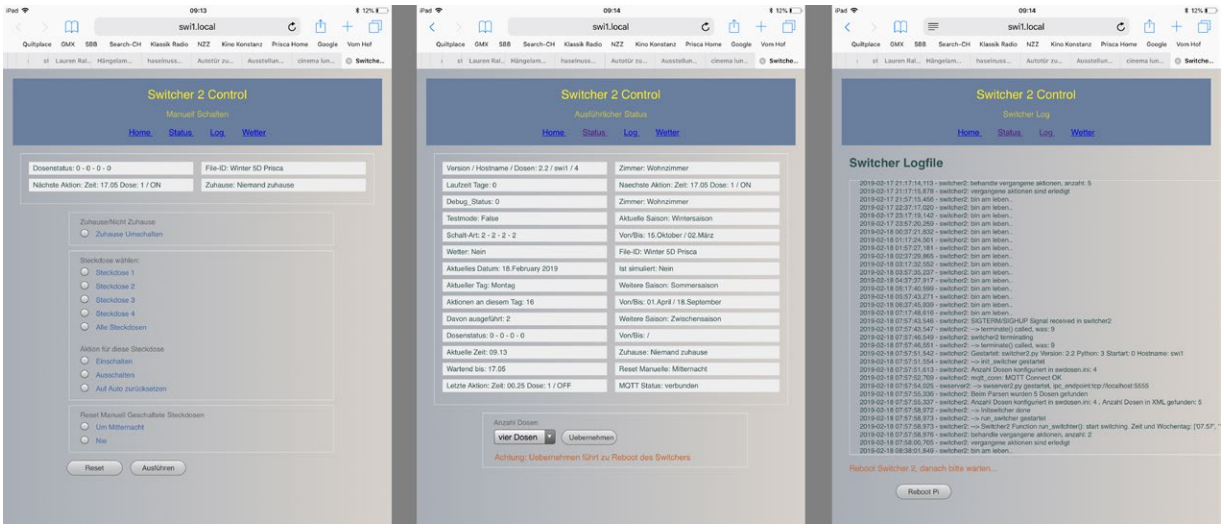
Switcher 2 Hardware:



Switcher 2 Aufbau mit Pi Zero W und 433 MHZ Sendemodul (rote Led für Test)



Switcher 2 Aufbau auf Pi Zero W mit Gehäuse



Switcher2 on the iPad: Basic Screen, Status Screen, Log Screen

3. Implementierung Switcher 2

Der Code für die Ansteuerung des Handsenders ist in der ersten Version fest codiert in Hauptprogramm des Switchers. Die Entflechtung des Schaltcodes war eine Hauptanforderung für den Switcher 2: zukünftige Erweiterungen des eigentlichen Schaltens müssen unabhängig vom Hauptprogramm möglich sein.

Dies wird durch Objektorientierung erreicht: es gibt in diesem Zusammenhang folgende Klassen:

- Klasse Dose: von dieser Klasse gibt es pro physikalisch vorhandene, schaltbare Dose eine Instanz. Jede Instanz der Dosenklasse instantiiert eine Aktor-Klasse - je nach definierter **Schaltart** für diese Dose (im Config-File).
- Klasse Aktor: diese Klasse (**und nur diese)** beinhaltet den Code für das effektive Schalten der Steckdosen - es gibt unterschiedliche Aktor-Klassen: beispielsweise eine für das Schalten mit 433 Mhz Sender und eine andere für Publishing zu einem MQTT Broker (IoT). Bei zukünftigen Weiterentwicklungen ist bloss ein neuer Typ der Aktor-Klasse zu erstellen.

3.1 Typen von Aktor-Klassen

Zur Zeit (September 2018) sind folgende Aktor-Klassen definiert - im Config-File wird für jede Dose einen sog. Schaltart definiert. Die Dose instantiiert dann eine der (momentan) 5 Aktor-Klassen:

- **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 beipielsweise 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:

Jede der definierten Dosen kann 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).

3.2 Die Wetter Klasse

Klasse 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. Verwendung von Wetter setzt Verwendung von MQTT voraus.

4. Schaltlogik

Der Switcher wurde entwickelt, um eine Anzahl von Steckdosen ein- oder auszuschalten, gemäss einer in Steuerfiles definierten Abfolge von Schaltaktionen. Ausserdem können einzelne Steckdosen durch den Benutzer manuell ein- oder ausgeschaltet werden (Webinterface).

4.2.1 Anzahl der Dosen

In den XML-Steuerfiles (siehen hinten) sind 5 Dosen definiert. Die Anzahl der effektiv geschalteten Dosen ist jedoch im Switcher definiert und die Anzahl kann im WebGui 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

4.2.2 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 in den Steuerfiles definierten (programmierten) Aktionen. **Manuelle Schaltaktionen** sind von Benutzer ausgelöst (via Client oder Webinterface). Siehe nächster Abschnitt für Beschreibung der manuellen Beeinflussung des Schaltens.

Die Dosen sind im Switcher-Programm Instanzen der Dosen-Klasse Der Schaltzustand jeder Dose ist in der Dose-Klasse (in der 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 progammierten 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. Manuelle Beeinflussung des Schaltens

Der Normalbetrieb des Switchers besteht in der Abarbeitung der in den Steuerfiles definierten Schaltaktionen für die verschiedenen Dosen (im folgenden genannt automatische oder programmierte 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.
- Via Webinterface (oder Commandline Client in der Testumgebung) können alle oder auch einzelne Dosen **manuell** ein- oder ausgeschaltet werden.
- Smart Switches können auch lokal manuell geschaltet werden

5.1 Nicht Zuhause Taste

Der Switcher ist so designed, dass er das ganze Jahr über aktiv sein kann. Während des nomalen 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.
- **Zuhause**: Switcher schaltet nach Umschalten auf 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 Zustand Zuhause nicht beeinflusst.

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

5.2 Einzelne Dosen schalten

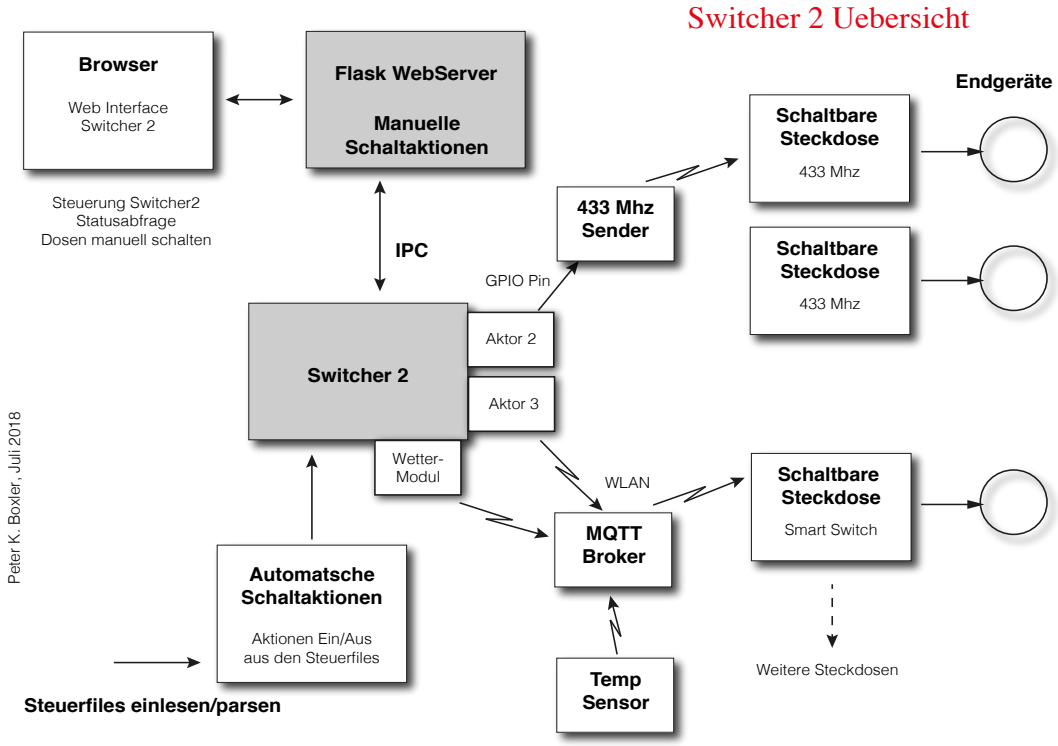
Mit Hilfe des Commandline Clients oder des Webinterfaces kann der Schalt-Zustand einzelner Dosen beeinflusst werden. Es gibt folgende Möglichkeiten:

- 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 ‚Zuhause' gilt ebenfalls: manuell geschaltet hat Vorrang. Beispielsweise: wurde eine Dose manuell eingeschaltet, so wird sie beim Umschalten auf ‚Zuhause' **nicht** ausgeschaltet.

Mit den genannten Anforderungen ergibt sich diese Funktions-Skizze des Switchers 2.



Funktionsweise Switcher 2

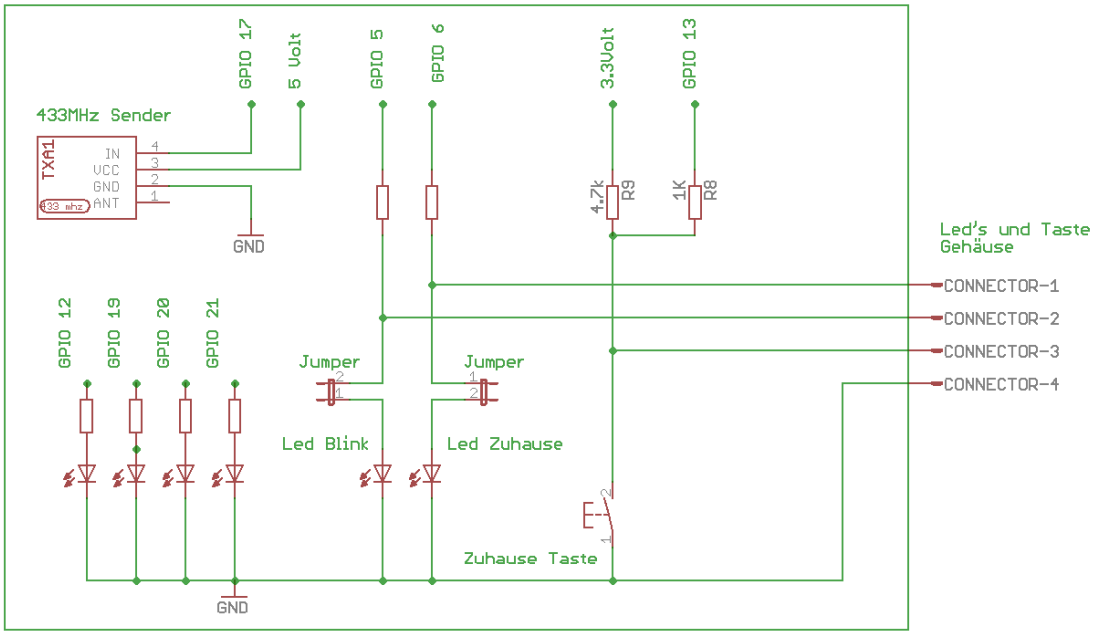
6. Hardware Switcher 2

Um den Stromverbrauch des Switcher2 gering zu halten, wird ein Raspberry Pi Zero mit WiFi verwendet. Der Pi Zero kommt mit grad mal ca. 130 mA Strom aus bei 5 Volt.

Für den Switcher 2 wird ein kleines Add-On Board verwendet (Adafruit), welches 6 Led's, die Home/Not Home-Drucktaste und das 433 Mhz Sendemodule trägt. Die 4 roten Led werden im Test verwendet

(Schaltart 1). Im produktiven Betreib werden diese Led nicht geschaltet. Die beiden grünen Leds und der Drucktaster sind auch am Gehäuse vorhanden. Siehe Schema weiter hinten.

Das 433 MHz Sendemodul ist gesteckt, es ist auch möglich, ein 433 MHz Empfangsmodul einzusetzen. Dies wird zur Ermittlung der Sendepulse benötigt. Dazu gibt es im Netz eine geniale Anleitung, siehe Links.



Schaltschema der Switcher 2 Platine

6.1 Schaltbare Steckdosen für Switcher 2

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 Gerätchen 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 2 (im Module Aktor_3.py) muss das passende MQTT-Protokoll (also Topic und Payload) richtig codiert werden. Siehe Kapitel **MQTT mit Switcher 2** 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 2, Ordner esp_code).

Note:

Wie bereits weiter oben dargelegt, ist es möglich, den Switcher 2 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.

Sonoff Wi-Fi Schalter für Smart Home

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

Shelly 1 Wi-Fi Smart Switch

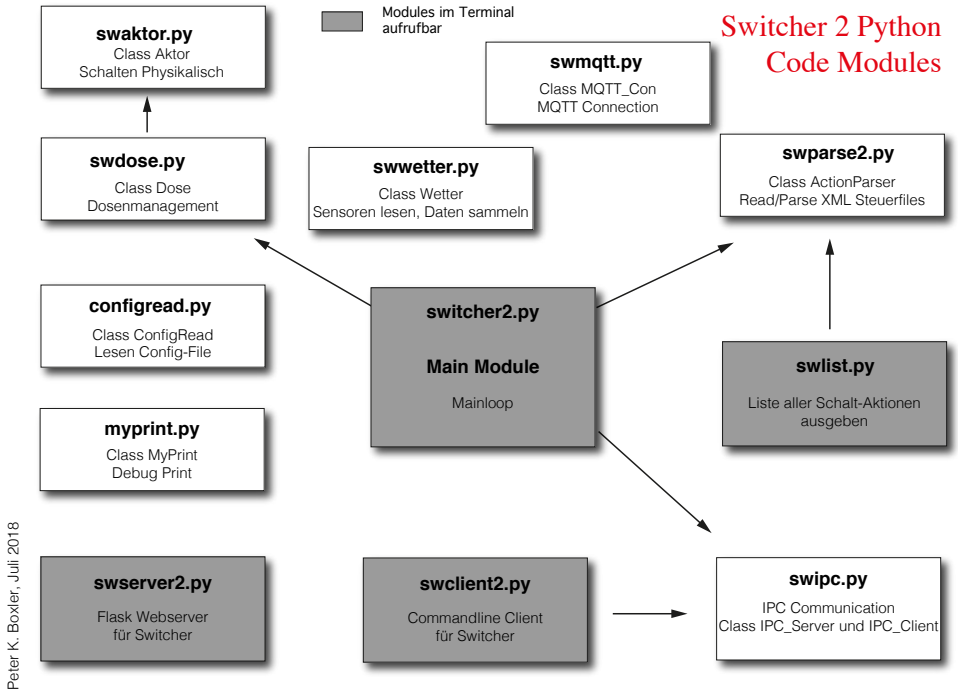
7. Software

7.1 Allgemeines

Die verwendete Script-Sprache ist Python. Das Switcher-Programmsystem besteht aus ca. 2500 Zeilen Python Code. Das Hauptprogramm ist das Modul **switcher2.py**. Der gesamte Code des Switchers-Programmsystems ist in verschiedenen Python Modulen codiert. Durch Einsatz von Objektorientierung wurde weitgehendste Unabhängigkeit erreicht - es gibt keine globalen Variablen (wie in Switcher Version 1) mehr. Dies erleichtert die Wartungsarbeiten und isoliert die div. Funktionen. Allerdings ist der Code nicht sehr pythonmässig: da ich mit Assembler, PL/1, Pascal, und C gross geworden bin, ist der Code an vielen Stellen nicht sehr pythonmässig.

Switcher2 Python Code und diese Dokumentation steht auf GitHub zur Verfügung.

Folgendes Schema gibt einen Überblick:



Die Python Module des Switchers

Die Tabelle im Kapitel ‚Software Komponenten' listet alle Module und deren Funktion. Die grau hinterlegten Module werden auf der Commandline aufgerufen..

7.1.1 Commandline Parameter

Alle aufrufbaren Module kennen mindestens diese 3 Commandline-Parameter für Debug-Output auf stdout oder Logfile - mehr dazu siehe weiter hinten im Kapitel Debug und Logging:

- - d kleiner debug
- - D grosser debug, mehr output
- - A noch mehr debug output

7.2 Das Switcher-Programm

Das Hauptprogramm ist das Modul **switcher2.py**. Dieses erledigt etwa folgende Aufgaben:

Zu Beginn wird im Hauptprogramm eine Instanz der ActionParser-Klasse instantiiert (Module **swparse2.py).** Dort werden die 3 Steuerfiles (siehe unten) eingelesen, geparst und alle notwendigen Schaltaktionen (für alle Dosen und alle 7 Tage und alle Saisons) werden in einer internen Liste abgelegt.
Der **Pseudocode** des Main-Loops des Switchers findet sich im Anhang - verschachtelte Loops sind in Python IMHO schwer lesbar - die Blockstruktur wird ja durch blosses Einrücken definiert - welch ein Schwachsinn. Kein Editor kann damit Blöcke deutlich machen. Der Main-Loop macht etwa folgendes

- abarbeiten der spezifizierten Schaltaktionen für alle Tage der Woche und endlose Wiederholung des Wochenprogramms.
- prüfen, ob der Client (Commandline oder Webserver) via IPC eine Anfrage oder einen Command sendet sendet mittels der Klasse IPC_Server im Modul **swipc.py**.
- prüfen, ob ein Wechsel der Saison vorgenommen werden muss (Sommer/Winter/Zwischen).
- **keep looping.... forever**

7.2.1 Start des Switchers

Das Switcher-Programm kann auf zwei Arten gestartet werden:

- Einfacher Aufruf auf Commandline: python3 switcher2.py [- Commandline Parameter]
- Aufruf als Linux-Daemon, siehe Kapitel weiter hinten

Beim Start des Programms können folgenden Commandline-Parameter spezifiziert werden:

```
-d            kleiner debug, Statusmeldungen werden ausgegeben (stdout)
-D            grosser debug, weitere Statusmeldungen
-A            ganz grosser debug, weitere Statusmeldungen
-s            Simulation (Schnelldurchlauf durch 3 Wochen
-t wochentag  bei dem die Simulation gestartet wird
-h            help, usage wird ausgegeben
-a            es werden nur die im File gefundenen Aktionen ausgegeben
```

7.2.2 Beenden des Switchers

Ein Python Script (wie jeder andere Linux Process) kann auf verschiedene Arten beendet werden. Für den Switcher2 gilt: wird der Switcher2 beendet, so soll sich er sich möglichst graceful beenden, konkret sollen alle Dosen abgeschaltet werden, die Sockets und GPIO Pins sollen aufgeräumt werden.

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 Errocode 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 kann in Python durch `try: / KeyboardInterrupt:` behandelt werden.
- Das Eintreffen eines SIGTERM Signals wird durch einen Signalhandler behandelt. Der Switcher kann also durch `kill pid` beendet werden. Dieses wird auch bei Herunterfahren des Raspberry Pi im OS ausgelöst.

In diesbezüglichen Tests zeigt sich, dass beim Reboot des Pi tatsächlich alle in diesem Moment eingeschalteten Dosen abgeschaltet werden. Dasselbe geschieht beim Stop des Switcher2 Daemons oder bei

Ctrl-C im Terminal Fenster.

7.2.3 MQTT mit Switcher 2

Bei den neuen OS für den Pi (Beispiel Raspian Stretch, Version Juni 2018) ist auch der MQTT Broker **mosquitto** bereits dabei. Bei Verwendung von Schaltart 3 (Smart Switches) muss der Broker vorgängig gestartet werden. Der Broker benützt Port 1883.

```
sudo systemctl enable mosquitto.service
sudo service mosquitto restart
```

Prüfen ob ok damit:

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

Resultat sollte sein:

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

7.2.4 Schaltart 3

Bei 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 Scripts s**wmqtt_pub.py** (Publisher) und **swmqtt_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 ‚switcher2'. Mit Cmd-Line Parm -t kann dies geändert werden.

Wenn dies funktioniert, kann auch der Switcher 2 mit Schaltart 3 gestartet werden.

Note:

Switcher 2 wurde mit mosquitto auf einen **Pi Zero W** erfolgreich getestet - der Winzling schafft das.

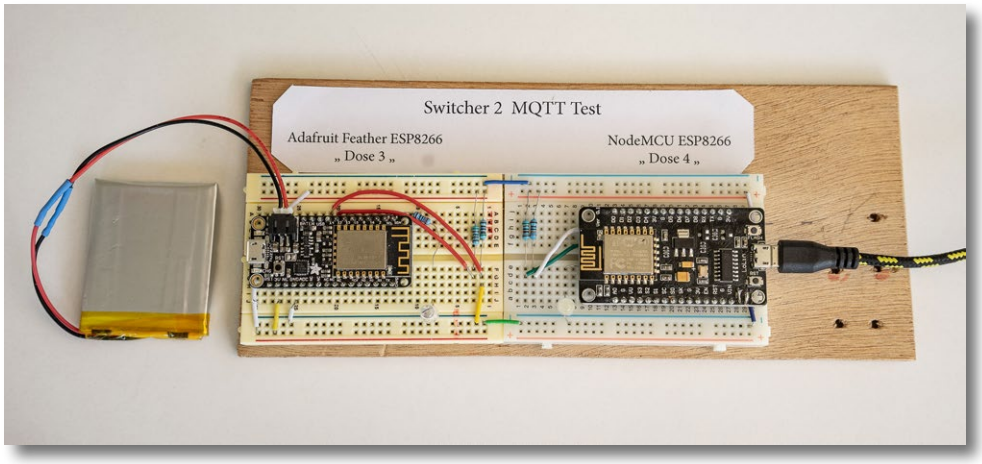
7.2.5 Testaufbau MQTT

Für einen ersten Test mit MQTT (proof of concept) wurde folgender Testaufbau benutzt, siehe Bild. Da zur Zeit noch keine Smart Switches beschafft wurden, soll 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).

Die beiden simulierten Dosen 3 und 4 werden durch den Switcher 2 via MQTT problemlos geschaltet -

sowohl automatische, wie auch manuelle Schaltaktionen.

Mittels des Testscripts swmqtt_pub.py kann die Schaltaktion auch einfach von einer Console ausgelöst werden. Dieses Testscript publiziert dieselben Payloads wie der Switcher 2. (Dosennumer und Ein/Aus). Der Broker läuft auf demselben Pi wie der Switcher selbst.



Testaufbau mit 2 ESP8266 als Dose 3 und 4 konfiguriert

7.2.6 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 Switcher2 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 bloss 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 **swmqtt_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 swmqtt_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...
bin immer noch da...
bin immer noch da...
bin immer noch da...
bin immer noch da...
...
```

Alles weitere ist eigene Handarbeit...

GitHub Switcher 2

7.2.7 Mosquitto Stuff

Der Mosquitto Browser auf dem Pi benutzt einen Config File der in folgendem Ordner liegt:

`/etc/mosquitto/mosquitto.conf`

Dieser File sollte nicht verändert werden - man kann aber (zusätzlich) einen eigenen Config File in folgendem Ordner anlegen:

`etc/mosquitto/conf.d`

Das Install Script **setup_swi.sh** (siehe Dokument Switcher2 install) schreibt den Konfig File. Siehe separates Dokument für Installation des Switchers auf einem Pi.

Siehe dazu bei Adafruit:

Configuring MQTT on the Raspberry P

```
# Config file for mosquitto
#
# See mosquitto.conf(5) for more information.
# from Adafruit 13. 4. 2018 by Peter
user mosquitto
max_queued_messages 200
message_size_limit 0
allow_zero_length_clientid true
allow_duplicate_messages false
log_timestamp true
listener 1883
listener 9001 127.0.0.1
protocol websockets
autosave_interval 900
autosave_on_changes false
persistence false
persistence_file mosquitto.db
allow_anonymous false
password_file /etc/mosquitto/passw.txt
```

Nach Speichern des zusätzlichen Config Files ist Mosquitto neu zu starten mit:

`sudo systemctl restart mosquitto`

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.

7.2.8 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. Siehe Hinweise im Kapitel Links.

Diese Smart Switches werden durch eine Meldung via MQTT Broker ein/ausgeschaltet. Zudem können sie auch manuell geschaltet werden - 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 Switcher2 gilt folgende Tabelle (N steht für Nummer der Dose):

Switcher 2 sendet (publish) Topic	**Payload**
cmnd/doseN/POWER	ON oder OFF
Smart Switch sendet (publish) Topic	**Payload**
stat/doseN/POWER	ON oder OFF

Switcher 2 muss also einen Subscribe mit dem Topic stat/# absetzen, um die Statzsmeldungen der Smart Switches zu empfangen (um damit den Status setzen zu können.

Konfiguration

Die Switches mit Tasmota Firmware sind vor Gebrauch mit den Switcher 2 zu konfigurieren: zuerst WLAN Access (fürs Einbinden ins eigene lokale Netz) und danach (mindestens) die MQTT Parameter.

Die Switches mit Tasmota Software werden durch 4 kurze Tastendrücke (auf der kleinen schwarzen Taste) mit einem Wireless Access Point gestartet und durch Verbinden mit diesem Access Point (PC/Mac, Tablet) können die WiFi Credentials fürs eigene WLAN eingegeben werden. Danach erscheinen die Switches im lokalen Netz und es kann das Web-Interface zur Konfiguration aufgerufen werden. Siehe Links im Anhang.

Das sieht etwa so aus:

Die verschiedenen Webseiten der Tasmota Konfig.

Wichtig für den Zugriff von Switcher 2 sind die Werte in den Feldern topic und full topic - natürlich muss auch der MQTT Broker richtig konfiguriert sein:

- topic muss sein **doseN**, wobei N die Dosennummer 1-5 ist
- full topic muss sein: **%prefix%/%topic%/**

Note:

Nur mit **diesen** Werten kann ein Smart Switch korrekt die vom Switcher 2 publizierte Meldung empfangen und den Verbraucher schalten.

Note 2:

Auch hier können die beiden Scripts **swmqtt_sub.py** und **swmqtt_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.

7.2.9 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 Switcher2 als eigener **Thread**, damit die Wartezeit beim Blinken zeitlich den Main-Loop nicht beeinträchtigt. Der Thread ist ais Daemon-Thread aufgesetzt, damit er beim Beenden des Switchers sicher auch beendet wird.

7.2.10 Implementierung Wetterdaten

Im Konfig-File des Switchers kann **spezifiziert** werden, ob Wetterdaten empfangen und angezeigt werden sollen. Verwendung von Wetter setzt Verwendung von MQTT voraus.

Die Temperatur/Feuchtigkeites-Sensor-Module sind mittels ESP8266 (Module E-12E) und dem Sensor BME280 gebaut. Ziel war dabei möglichst kleiner Stromverbrauch - Batteriebetrieb. Der Code vewendet Watchdogs und 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 Spannungteilers, der im Deep-Sleep Modus inaktiviert wird.

Zum Switcher 2 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 die Topic und Payload, die verwendet werden:

Sketch Indoor sendet (publish) Topic	**Payload**
switcher2/wetter/data	indoor/battstatus/sensorstatus/elapsed_time_ms/TEMP/HUM
Sketch Outdoor sendet (publish) Topic	**Payload**
switcher2/wetter/data	outdoor/battstatus/sensorstatus/elapsed_time_ms/TEMP/HUM

7.3 Switcher Client

Das Switcher-Client Programm **swclient2.py** kann Requests an den Switcher senden. Siehe folgendes Kapitel Interprocess Kommunikation (IPC). Der Client wird bei Bedarf auf der Commandline aufgerufen und erwartet vom Benutzer Eingabe eines Requests. Der Client wird die Anfragen an den laufenden Switcher-Process senden - dieser beantwortet die Anfragen in seiner Funktion als Server. Die wichtigste und wohl am häufigsten verwendete Anfrage ist jene nach dem Status des Switchers, er meint etwa folgendes: sag mir, was du jetzt gerade tust, was du als letztes getan hast und was du als nächstes tun wirst.

Der Status-Anfrage bringt folgende Antwort vom Switcher (auf stdout):

```
pi@raspberrypi ~/switcher $ sudo python swclient2.py
Please give a command: stat
Sending: stat
-------------------------------------
Version           :  2.1
Läuft_seit        :  Thursday 16.08.2018 11.39
Debug_Status      :  0
Testmode          :  Ja
Schalt-Art        :  2 - 2 - 3 - 3 - 1
Simulation        :  Nein
Aktuelles Datum   :  16.August.2018
Aktueller Tag     :  Donnerstag
Aktionen/Tag      :  68
Aktionen done     :  24
Dosenstatus       :  0 - 0 - 1m - 1 - 0
Aktuelle Zeit     :  11.51
Wartend bis       :  12.10
```

```
Letzte Aktion     :  Zeit: 11.40 Dose: 4 / ON
Zimmer            :  Badezimmer
Nächste Aktion    :  Zeit: 12.10 Dose: 3 / ON
Zimmer            :  Arbeitszimmer
Aktuelle Saison   :  Sommersaison
Von/Bis           :  01.März / 30.September
File-ID           :  Sommer Test 4D Peter
Ist simuliert     :  Nein
Weitere Saison    :  Wintersaison
Von/Bis           :  01.November / 31.März
Weitere Saison    :  Zwischensaison
Von/Bis           :    /
Zuhause           :  Niemand zuhause
Reset Manuelle    :  Mitternacht
-------------------------------------
```

In diesem Beispiel ist zu sehen, dass die Dosen 4 (im Badezimmer) eingeschaltet ist (durch automatische Aktion) und dass der Switcher auf Zeit 12.10 Uhr wartet, um dann die Dose 3 (im Arbeitszimmer) einzuschalten. Letzte Aktion war um 11.40 Uhr, als Dose 4 eingeschaltet wurde. Die Dose 3 wurde **manuell** eingeschaltet. Der Home/NotHome Switch ist in Stellung Niemand Zuhause. Die manuell geschalteten Dosen werden um Mitternacht wieder auf Normalbetrieb gesetzt.

Die Zeile **Dosenstatus** (siehe oben, Statusanzeige) zeigt den effektiven Schaltstatus aller Dosen (0=Aus /1=Ein) sowie zusätzlich den Schaltmodus der Dosen.

Der **Schaltmodus** wird mit einen Kleinbuchstaben nach dem Status angezeigt:

- leer die Dose ist automatisch, dh. nach Aktionsprogramm geschaltet worden.
- m die Dose ist manuell geschaltet worden, sie wird nicht mehr automatisch geschaltet
- h die Dose ist in diesem Status infolge Zuhause

7.3.1 Interprocess Kommunikation

Da der Switcher-Prozess als Linux Daemon im Hintergrund läuft und damit keine Verbindung zu einer Konsole hat, ist das Switcher-Programm als Server ausgeführt, welches neben dem Schalten der Dosen auch Requests von einem Clienten beantworten kann. Diese Interprocess-Kommunikation wird über Sockets abgewickelt - es wird dazu das geniale Framework **ZeroMQ** verwendet: ‚an intelligent transport layer for distributed applications'.

Aus dem Guide:

> *ØMQ (also known as ZeroMQ, 0MQ, or zmq) looks like an embeddable networking library but acts like a concurrency framework. It gives you sockets that carry atomic messages across various transports like in-process, inter-process, TCP, and multicast. You can connect sockets N-to-N with patterns like fan-out, pub-sub, task distribution, and request-reply. It's fast enough to be the fabric for clustered products. Its asynchronous I/O model gives you scalable multicore applications, built as asynchronous message-processing tasks. It has a score of language APIs and runs on most operating systems. ØMQ is from iMatix and is LGPLv3 open source.*

Die IPC ist in 2 Klassen implementiert im Module swipc.py. Die Klasse IPC_Server wird im Switcher verwendet - das Switcher-Programm fragt im Main-Loop damit nonblocking den Socket ab und kann so Anfragen vom Client-Programm **swclient2.py** oder vom Webserver **swserver2.py** beantworten.

Die Klasse IPC_Client wird im Client Module swclient2.py verwendet.

7.3.2 Mögliche Anfragen

Bei der Eingabeaufforderung des Clients können folgende Anfragen eingegeben werden:

stat	Status Abfrage gross (ausführlich), der Switcher meldet den internen Status als Liste, diese wird auf stdout ausgegeben.
stad	Status Abfrage klein, zeige weniger Info.
stop	Stop Befehl für den Switcher - dieser wird ornungsgemäss heruntergefahren, wobei alle Dosen ausgeschaltet werden. Der Switcher beantwortet diesen Request mit ackn
dXein	Schalte Dose X (X=1 bis 4) sofort manuell ein - gilt bis nächste Switcher Aktion AUS auf dieser Dose Der Switcher beantwortet diesen Request mit ackn
dXaus	Schalte Dose X (X=1 bis 4) sofort manuell aus - gilt bis nächste Switcher Aktion EIN auf dieser Dose Der Switcher beantwortet diesen Request mit ackn
dXnor	Schalte Dose X (X=1 bis 4) sofort wieder entsprechend den normalen Switcher Aktionen ein. Manueller Modus beenden. Der Switcher beantwortet diesen Request mit ackn
aein	Schalte alle Dosen ein
aaus	Schalte alle Dosen aus
anor	Schalte alle Dose sofort wieder entsprechend den normalen Switcher Aktionen - manueller Modus beenden.
mmit	Reset manuell geschaltete Dosen um Mitternacht
mnie	Reset manuell geschaltete Dosen Nie
term	Beendet den Client swclient2.py - der Switcher läuft weiter.
spez	Für Test und Debug: der Switcher beantwortet diesen Request mit ackn aber mit 500 ms delay. Damit kann der Client getestet werden.
spezNNNN	NNNN= Anzahl Millisekunden, die der Switcher wartet, bis Antwort ackn gesendet wird. Damit kann der Client getestet werden.

7.3.3 Client Commandline Parameter

Es ist auch möglich, dass der Client genau eine einzige Anfrage an den Switchers absetzt (und die Antwort auf stdout ausgibt) - dies ist der Fall, wenn die Anfrage mittels Commendline-Parameter angegeben wird.

Neben den 3 bereits erwähnten Debug-Parametern kennt der Client folgende Commandline Parameter:

- -s Nur eine Status-Anfrage machen, Antwortt auf stdout ausgeben, dann beenden
- - dein N, Dose N manuell einschalten, gilt bis Mitternacht
- - daus N Dose manuell ausschalten, gilt bis Mitternacht
- - dnor N Dose N wieder normal behandeln

7.4 XML Steuerfile

7.4.1 Allgemeines zu den Steuerfiles

7.4.2 Implementierung für Switcher 2

Die 3 Saisons-Anforderung wird gelöst durch die Verwendung von neu drei Steuerfiles - anstelle des bisherigen einzelnen Steuerfiles. Es gibt also einen Steuerfile für Sommer, einen für Winter und einen für die Zwischensaison. Sommer- und Winterfile enthalten je ein Start- und ein Ende-Datum. Beispiel: der Sommerfile enthält Start-Datum des Sommers (Tag.Monat) sowie das Ende-Datum des Sommers. Ebenso der Winterfile. Es gilt: alles was ausserhalb Sommer oder Winter liegt, bedeutet Zwischensaison. Der Zwischensaison-File enthält also keine Start-/End-Daten. Der Switcher ermittelt im Mainloop, welche Saison aktuell gilt und ob sie geändert hat seit der letzten Prüfung. Die Saison kann zudem beliebig via Switcher-Client (oder Webinterface) simuliert werden.

Note:

Alle Schaltzeiten für alle 3 Saisons sind in der internen Liste **list_tage()** gespeichert. Diese wird- wie bisher- durch den Parser bestückt. Eine Saison-Umschaltung erfordert also bloss die Veränderung eines Indexs bei Zugriff auf die Schaltzeiten.

Diese Steuerfiles werden nach Programm-Start eingelesen und geparst und es entsteht eine Aktionen-Tabelle, welche alle Schaltvorgänge (EIN/AUS) für jeden Tage der Woche und für jede Dose enthält in jeder der 3 Saisons. 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. Zudem sind die Start- und Ende-Daten der entsprechenden Saison im File definiert. Die Steuer-Files werden 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.

Die Filenamen der Steuerfiles sind werden folgendermassen ermittelt (in swparse2.py):

- **prefix aus Config-File + ‚-' + ‚saison' + ‚.xml'**

Dies bedeutet: Falls im Configfile als prefix ‚swhaus1' steht, so heissen die 3 Steuerfiles:

- swhaus1-sommer.xml
- swhaus1-winter.xml
- swhaus1-zwischen.xml

Dies ist der Normalfall - Steuerfiles für Test-Umgebung siehe weiter hinten.

Beispiel eines solchen Files findet sich im Anhang.

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.

7.4.3 Steuerfiles zum Testen

Neben den 3 regulären Steuerfiles (welche in der Nacht schalten), gibt es zudem 3 Test-Steuerfiles (einen pro Saison, sie sind identisch), welche tagsüber häufige Schaltaktionen definiert haben (für jeden Tag der Woche identisch) - fürs Testen ist dies sehr hilfreich. Als Software-Entwickler möchte man ja nicht die ganze Nacht aufbleiben, um das korrekte Arbeiten des Switchers zu beobachten...

Im **Config-File** kann dafür ganz einfach ein anderer prefix definiert werden, beispielsweise ‚haus1-test'.

Mit dem Client swclient2.py oder dem Web-Interface kann jederzeit überprüft werden, welcher Steuerfile und welche jahreszeitliche Saison aktiv ist.

Dies bedeutet: Falls im Configfile als prefix ‚**swhaus1-test**' steht, so heissen die 3 Steuerfiles:

- swhaus1-test-sommer.xml
- swhaus1-test-winter.xml
- swhaus1-test-zwischen.xml

7.4.4 Vorhandene Steuerfiles

Original sind im Switcher 2 folgende Steuerfiles vorhanden:

Name	**Zweck**
swhaus1-sommer.xml swhaus1-winter.xml swhaus1-zwischen.xml	Satz Steuerfiles zum Anpassen für eigenen Gebrauch, Schaltaktionen am Abend
swhaus1-test-sommer.xml swhaus1-test-winter.xml swhaus1-test-zwischen.xml	Satz Steuerfiles zum Entwickeln/Testen, die meisten Schaltaktionen finden tagsüber statt
swhaus1-test2-sommer.xml swhaus1-test2-winter.xml swhaus1-test2-zwischen.xml	Satz Steuerfiles zum Entwickeln/Testen, nur wenige Schaltaktionen und alle sind um 11.30 vorbei. Dient zum Testen der Funktionsweise des Switchers zwischen letzter Schaltaktion des Tages und Mitternacht
swhaus2-sommer.xml swhaus2-winter.xml swhaus2-zwischen.xml	Wie erster Satz, für Haus 2
xmlfehler1.xml , xmlfehler2.xml	XML Files mit Fehlern, können zum Testen des Prüfscripts verwendet werden.

7.4.5 Prüfung der Steuerfiles

Die Steuerfiles können mit einem speziellen Testscript **swtestxml .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 swtestxml.py -f xml/swhaus2-zwischen.xml
```

7.5 Config-File

Einige wenige Parameter für das Switcher Programmsystem werden einem Config-File entnommen.

Im File müssen folgende Abschnitte vorhanden sein: `[switcher], [dose] und [aktor_1], [aktor_2], [aktor_3], [aktor_4] und [aktor_5]`

Der Abschnitt [switcher] wird vom Switcher2.py benutzt, der Abschnitt [dose] von der Klasse Dose und die Abschnitte [aktor_n] von den Aktor-Klassen.

Der Config-File sieht so aus (Beispiel Test-Umgebung):

```
[switcher]
# Dieser Abschnitt wird vom Switcher2 und seinen Klassen gelesen
#
# testmode Ja/Nein
testmode = Ja
```

```
# Teil des Filenamens für die XML Steuerfiles, wird ergänzt
saison_1 = sommer
saison_2 = winter
saison_3 = zwischen
# Prefix für die Filenamen der XML Steuerfiles
xmlfile_prefix = swhaus1
#   switcher mit mqtt connection: 1=Ja , 0 = Nein
#  ist noetig für Dosen mit Schaltart = 3 oder für Wetter
setup_mqtt = 1
# soll switcher wetter daten behandeln
wetter = 1
# Date für IPC
ipc_endpoint_c = tcp://localhost:5555
ipc_endpoint_s = tcp://*:5555
# GPIO PIn für zuhausse 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
 # GPIO Pin für Blink LED,  0: kein PIN verwendet
gpio_blink = 5
# OLED Display vorhanden, vorläufig not used
oled = 0

# 5 binary Schalter
schalter = 00001
# manuell_reset: wann sollen manuell geschaltete Dosen zurückgesetzt werden
# 1 : Mitternacht, 0: Nie
manuell_reset=1
# not used
reserve = istreserv


[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 = switcher2
mqtt_pw = itscool
mqtt_qos = 0
mqtt_retain = 0

#
[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_schaltprio = 2
dose_2_schaltprio = 1
dose_3_schaltprio = 1
dose_4_schaltprio = 0
dose_5_schaltprio = 1
dose_6_schaltprio = 1
# schaltart definiert die Art des Schaltens, es wird die entspr. Aktor-Klasse 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.
#   wenn eine dose 1 schaltart 1 hat, müssen die anderen Dosen auch schaltart 1 haben
#   script sorgt dafür (mit Warnung), dass dies eingehalten wird.
#   Amsonsten 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: Testumgebung, 2: Sonoff-SmartSwit-
ches)
#   X: Susbcribe möglich (not used)
#
dose_1_schaltart = 2
```

```
dose_2_schaltart = 2
dose_3_schaltart = 2
dose_4_schaltart = 2
dose_5_schaltart = 1
dose_6_schaltart = 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 keine Parameter


[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/
#
```

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

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

Folgende Werte sind zur Zeit definiert - wird eventuell später erweitert:

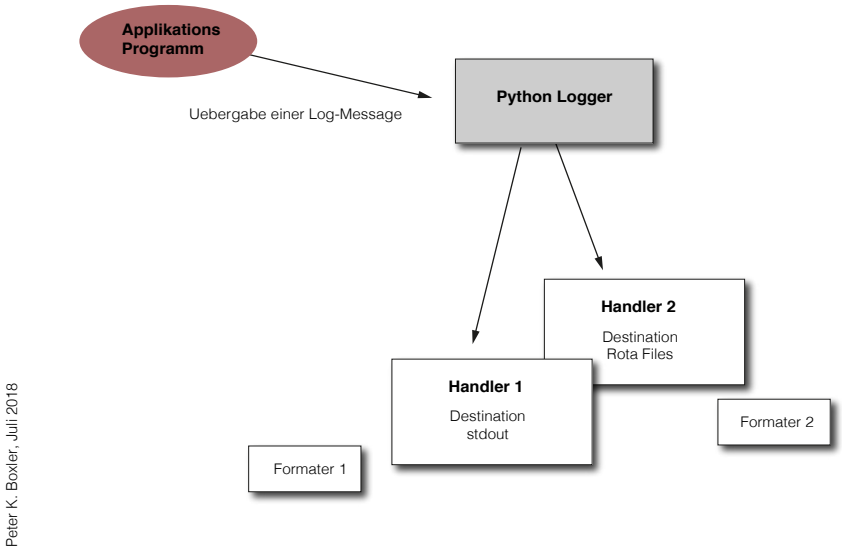
- testmode: Ja oder Nein. Der Switcher verwendet dies, um bei allen Dosen Schaltart 1 (Testumgebung) einzustellen. Zudem werden alle 4 Led's kurz eingeschaltet beim Start des Switchers.
- ipc_endpoint: Socket ID für die Interprcess-Kommunikation - wird vom Switcher2 und von Commandline-Client verwendet.
- xmlfile_prefix: Prefix für die 3 Steuerfiles der Saisons. Wird im Modul swparse2.py ergänzt mit dem Namen der Saison, beispielsweise **steu-sommer.xml**.
- gpio_home: GPIO Pins des Home/Not at Home Schalters
- gpio_home_led: GPIO Pins der Home/Not a Home Led
- gpio_blink: GPIO Pin der Blink Led
- oled: Ja oder Nein, ob OLED Display angeschlossen ist.
- schalter: 5 einzelne Schalter die 0 oder 1 sein können, for future use.
- manuell_reset: ob manuelle Schaltvorgänge forever oder bloss bis Mitternacht gelten sollen.
- reserve: vorläufig nicht verwendet.

7.6 Debug Output und Logging

Mit den beschriebenen Commandline Parameter -d, -D, und -A kann der Detailgrad des Debug-Outputs spezifizert werden. Die in den Programmen (besser gesagt Scripts) vorkommenden MyPrint() Aufrufe werden dem Python Logging System übergeben. Es gibt **keine** direkte Ausgabe auf stdout vom Script (also keine Print-Statements). Jeder MyPrint() Aufruf im Programm trägt eine Level-Angabe - je nach Wichtigkeit genau dieses MyPrint() Aufrufs (durch Entwickler definiert).

Das Logging System ist im Switcher so konfiguriert, dass jede MyPrint-Zeile umgeleitet wird auf die zwei folgende Destinationen

- auf stdout (also Console)
- in den Logfile **switcher2.log**. Diese Ausgaben in den Logfile werden durch das Logging System mit Datum/Zeit ergänzt.



Schema des Python Loggers im Switcher

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 Parms 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 Command-line-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 diesen Commandline Parm**	**im Log erscheinen folgende Level**
DEBUG_LEVEL0	Wichtig: immer sehen	wird immer ausgegeben	ERROR
DEBUG_LEVEL1	wenig sehen, grobe Sicht auf den Ablauf	Ausgabe bei -d und -D und -A	ERROR und WARNING
DEBUG_LEVEL2	mehr sehen, etwas mehr Details im Ablauf	Ausgabe bei -D und -A	INFO und WARNING und ERROR
DEBUG_LEVEL3	alle Details sehen, gibt viel Output !	Ausgabe nur bei -A	INFO und WARNING und 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 **switcher2.log**, der letzte heisst **switcher2.log 1** und der vorletzte heisst **switcher2.log 2**

Auch nach monatelagem 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 konfigriert mit 350 kB pro File. Es wird also nie zu einem Filesystem-Ueberlauf kommen.

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

7.7 Schaltzeiten anzeigen

Das Script **swlist.py** wird benutzt, um die in den XML-Files codierten Schaltzeiten pro Tag , pro Dose und pro Saison auszugeben - für Tests und Doku. Es werden 9 Listen ausgegeben (je 3 pro Saison) - siehe Beispiel im Anhang.

- Schaltzeiten für alle Tage der Woche für alle Dosen.
- Schaltzeiten für jede Dose für alle Tage der Woche
- Grafische Darstellung (Timeline) für alle Tage und Dosen (jedoch nur von 17 bis 24 Uhr)

Die verfügbaren Commandline Parameter für swlist.py sind:

```
usage: swlist.py [-h] [-d] [-D] [-A] [-s] [-w] [-z] [-g]

optional arguments:
  -h, --help  show this help message and exit
  -d          kleiner debug
  -D          grosser debug
  -A          ganz grosser debug
  -s          Aktionsliste Sommer
  -w          Aktionsliste Winter
  -z          Aktionsliste Zwischen
  -g          Liste der Saisons
```

Dieses Script benutzt dieselbe Parsing-Funktion wie das Switcher-Script switcher2.py. Diese Parsing-Funktion ist im Script swparse.py zu finden. Dort ist die Klasse ActionParser definiert.

Note: Dieselben Aktions-Listen werden auch vom Switcher Script switcher2.py ausgegeben, falls dort der Commandine Parameter -a gesetzt ist.

Wichtig:

Unbedingt das Prüfprogramm für die Steuerfiles verwenden, wenn ein File modifiziert wurde. Siehe weiter oben.

7.8 Testprogramme

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

- swdostest.py: zum Testen der Klasse Dosen und Aktor, schaltet in einem Loop 4 Dosen ein und aus.
- printest_1.py und printest_2.py: zum Testen der Klasse MyPrint
- testconfig.py: zum Testen der Klasse ConfigRead
- swmqtt_pub.py: für MQTT Tests
- swmqtt_sub.py: für MQTT Tests
- testtest.py: für Test der Testumgebung (alle LED korrekt)
- swtestxml.py: für Prüfung der Steuerfiles

8. Dämonisierung

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.

8.1 Switcher als Dämon

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

Der Switcher2 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. Die früher benutzte Mechanik mit Eintrag im /etc/rc.local File kann dies nicht bieten - dort wird gestartet und fertig. Darum wird dies nicht mehr häufig benutzt.

Starten

```
sudo /etc/init.d/switcher2.sh start
sudo /etc/init.d/swserver2.sh start
```

Stoppen

```
sudo /etc/init.d/switcher2.sh stop
sudo /etc/init.d/swserver2.sh stop
```

und Statusabfrage mittels

```
sudo /etc/init.d/switcher2.sh status
sudo /etc/init.d/swserver2.sh status
```

Für den Switcher wird der **Init.d-Mechanismus** verwendet, um die Scripts bei Beginn zu starten und beimHerunterfahren zu stoppen.

Es braucht dazu sog. Init-Scripts, diese werden im Verzeichnis /etc/init.d/ abgelegt und sie haben Methoden für start, stop, restart oder auch status definiert. Vorteil von Init.d ist, das ein Programm sowohl beim Boot des Pi gestartet wird, wie auch beim Runterfahren gestoppt wird. Nur damit ist eine graceful termination im Switcher-Programm möglich. Im laufenden Betrieb ist mit „status“ jeder Zeit der aktuelle Zustand abfragbar.

Die beiden Scripts (für Switcher2 und für Webserver) sind im Directory shell_scripts zu finden.

Weitere Details siehe das Dokument **Switcher2 Installation**.

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.

8.2 Switcher ohne Daemon

Der switcher2.py kann auch für sich allein aufgerufen werden kann - mit den genannten Commandline Parameter. Dies wird wohl meist beim Testen verwendet - das Programm ist an die Konsole gebunden. Wird diese beendet, so stirbt auch das Programm.

Der Switcher wird dann so gestartet (aus dem Ordner switcher2)

```
python3 switcher2.py
```

9. Switcher 2 Webserver

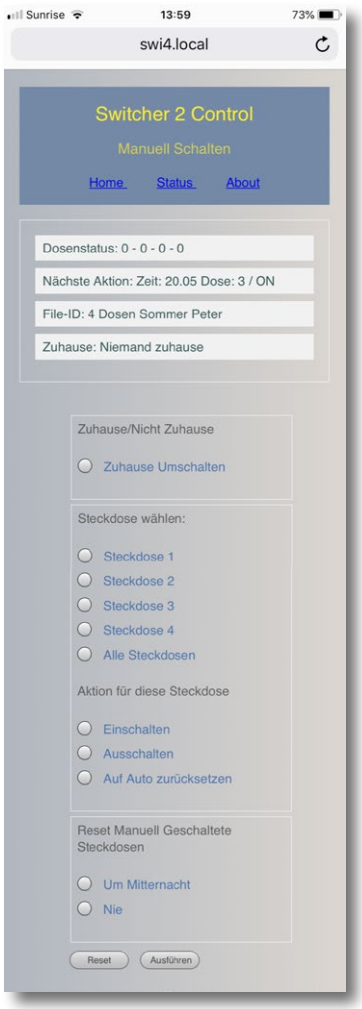
Das Programm **swserver2.py** implementiert einen Python-Webserver mit dem Flask Framework. Der Webserver macht Interprocess Kommunikation mit dem laufenden Switcher. Das Modul **swserver2.py** implementiert einen Webserver und es ist deshalb kein anderer Webserver (Apache, lighttpd, oä.) zu installieren auf dem Pi des Switchers.

Das Webserver-Modul ist aber gleichzeitig auch ein Client gegenüber dem Switcher - nach Eintreffen eines HTML Get Request von einem Browser wird via IPC ein Status-Request an den Switcher abgesetzt. Dessen Antwort wird in der an den Browser ausgegebenen HTML Seite passend eingefügt. Das Flask Framework stellt dazu Funktionalität zur Verfügung. Für die IPC wird die IPC-Server Klasse verwendet (Modul **swipc1.py)**. (Der Commandline-Client **swclient2.py** verwendet ebenfalls dieses Modul.)

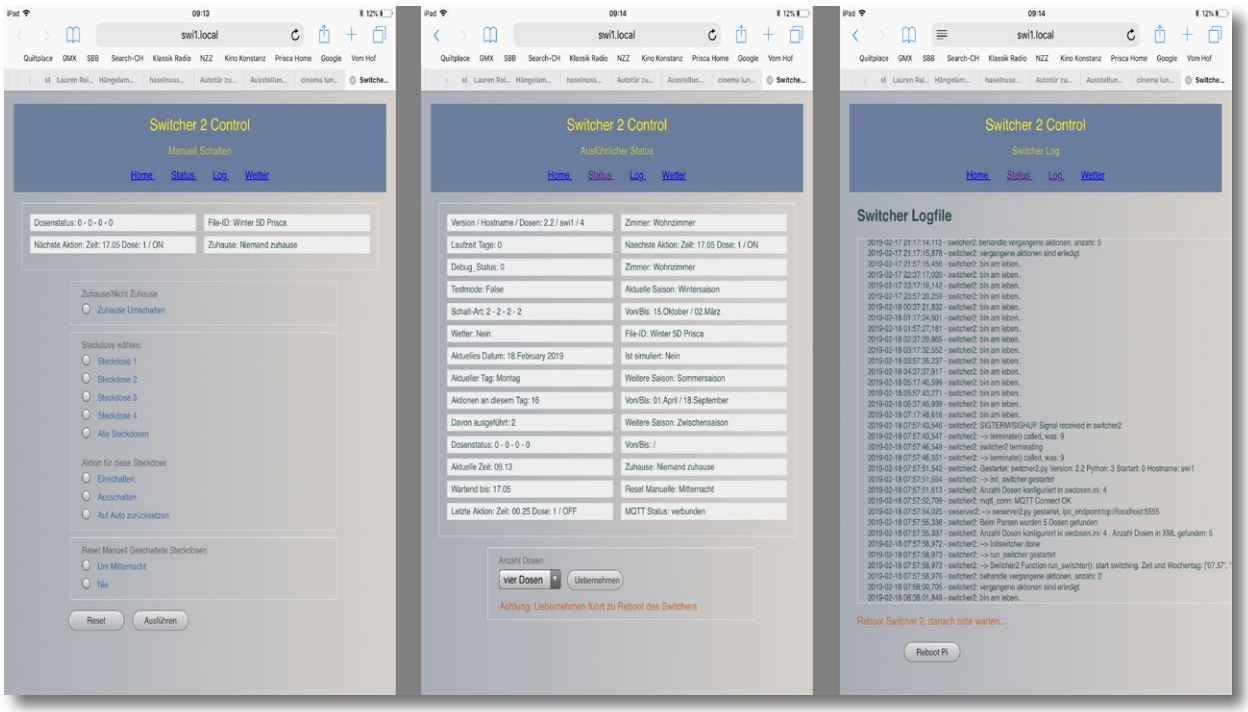
Der Switcher2 Webserver wird ebenfalls beim Boot des Pi durch den init-d Service gestartet.

Mit Hilfe des Webinterfaces können folgende Aufgaben erledigt werden:

- Manuelles Schalten der Dosen
- Statusabfrage des Switchers
- Anschauen des Logs des Switchers (File switcher2.log)
- Reboot des Switchers
- Konfiguration der Anzahl installierten Dosen



Status via Webinterface auf dem iPhone (scrolled)



Webinterface auf dem iPad (Manuell Schalten, Status, Switcher Log)

10. Software Komponenten

Das Switcher-Projekt besteht aus folgenden Codeteilen:

Script Name	Aufrufbar	Funktion
switcher2.py	JA	Hauptprogramm, ruft die verschiedenen Init-Funktionen auf und läuft danach im Main-Loop. Nur unter **Python 3.x** lauffähig, wird beendet bei Verwendung unter Python 2.x
swclient2.py	JA	Commandine Switcher-Client, sendet Requests an den Switcher, verwendet die ipc-Funktionen im Modul swipc1.py
swserver2.py		Flask Webserver für Switcher 2. Nur unter **Python 3.x** lauffähig, wird beendet bei Verwendung unter Python 2.x
swipc.py		Definiert die beiden Klassen für IPC. Klasse **IPC_Client** für den clientseitigen Teil der Interprocess Kommunikation auf Basis zeroMQ. Wird verwendet im Commandline Client und im Webserver. Damit wird mit dem Switcher kommuniziert. Klasse **IPC_Server** für den serverseitigen Teil der Interprocess Kommunikation auf Basis zeroMQ, hier wird geprüft, ob via IPC ein Request vom Client eingetroffen ist. Rer Request wird bearbeitet
swparse2.py		Hier ist die Klasse **ActionParser** definiert. Diese Klasse übernimmt alle Funktionen im Zusammenhang mit den Steuerfiles und den darin vorhandenen Daten. Hier werden die 3 XML-Steuerfiles eingelesen und geparst. Alle Schaltaktionen werden in einer Liste abgelegt - diese Listen werden vom Caller übergeben. Die Methoden dieser Klasse -> siehe Kopf in swparse2.py.
myprint.py		Hier ist die Klasse **MyPrint** definiert- Alle anderen Klassen erben von MyPrint. Diese Klasse übernimmt debug Output. Hie ist auch die Klasse MyLog definiert. MyPrint erbt von MyLog.
swmqtt.py		Implementierung der Klasse MQTT_Conn. Behandlung der Connection zum MQTT Broker
swwetter.py		Implementierung der Klasse Wetter. Behandlung der Meldungen der Temperatur/Feuchtigkeits-Sensoren Outdoor und Indoor.
defglobal2.py		Include mit Definition von bloss noch einer globalen Variablen
defstatus1.py		Include mit den Variablen für kleine Statusausgabe
defstatus2.py		Include mit den Variablen für grosse Statusausgabe

steu-sommer.xml steu-winter.xml steu-zwischen.xml		XML Steuer-Files mit den Schaltsequenzen, einer pro Saison. Es gibt auch Test-Versionen dieser Files mit Schaltzeiten tagsüber für Tests.
swdose.py		Hier ist die Klasse **Dose** implementiert. Dosenmanagement, Dosenstatus führen, Instantierung der Klasse Aktor (physikalisches Schalten der Dosen)
swaktor_1.py		Definiert die Klasse **Aktor_1**. In ihr sind die Details der spezifischen Implementierung des Schaltens der Dosen codiert. **Aktor_1** wird für Test und Debug mit dem speziellen Testaufbau verwendet. Schaltart=1 im Config File.
swaktor_2.py		Definiert die Klasse **Aktor_2**. In ihr sind die Details der spezifischen Implementierung des Schaltens der Dosen codiert. **Aktor_2** wird für originale Implementierung mit Funk-Steckdosen verwendet (Sender ist dabei eine kleiner 433Mhz Sender an einen GPIO-Port). Schaltart=2 im Config File.
swaktor_3.py		Definiert die Klasse **Aktor_3**. In ihr sind die Details der spezifischen Implementierung des Schaltens der Dosen codiert. **Aktor_3** publiziert MQTT Meldungen an einen MQTT Broker. Schaltart=3 im Config File. Broker-Config ebenfalls im Config File.
swaktor_4.py		Definiert die Klasse **Aktor_4**. In ihr sind die Details der spezifischen Implementierung des Schaltens der Dosen codiert. **Aktor_4** wird für originale Implementierung der ersten Version des Switcher mit Handsender via 6 GPIO-Pins angesteuert). Schaltart=2 im Config File.
swaktor_5.py		Definiert die Klasse **Aktor_5**. In ihr sind die Details der spezifischen Implementierung des Schaltens der Dosen codiert. Spezialversion für Send Modul, welche die obskure Library **wiringpi** benötigt.
swlist.py	JA	dies ist ein Stand-alone Programm, welches mittels Klasse **ActionParser** die XML Steuerfile liest, um eine Liste aller definierten Schalt-Aktionen auszugeben (zur Dokumentation).
configread.py		Defintion der Klasse **ConfigRead.** Hier wird der Config-File swconfig.ini eingelesen und dessen Werte an den Caller übergeben.
swconfig.ini		Config-File des Switchers. Test-Programm 1 für die Klasse MyPrint.
Hilfsfunktionen		**Test Programme**
printest_1.py		Test-Programm 1 für die Klasse MyPrint.
printest_2.py		Test-Programm 2 für die Klasse MyPrint.
swdostest.py		Test-Programm für die Klasse Dose.Ebenfalls wird die Klasse ConfigRead getestet. Schaltet 4 Dosen in einem Loop ein/aus. Nur unter **Python 3.x** lauffähig, wird beendet bei Verwendung unter Python 2.x

testconfig.py		Für den Test der Classe ConfigRead
swmqtt_pub.py		MQTT publisher zum Testen der MQTT-Funktionalität. Topic ist ‚switcher2'
swmqtt_sub.py		MQTT Subscriber zum Testen der MQTT-Funktionalität. Topic ist ‚switcher2. Sind Dosen mit Schaltart 3 vorhanden, so können mit diesem Script die vom Switcher2 publizierten Meldungen an die WiFi-Dosen angeschaut werden.
swtestxml.py		Prüfung der XML Steuerfiles.

11. Installation Switcher 2 auf dem Pi

Das separate Dokument Switcher 2 Installieren beschreibt die Installation des Switchers im Detail.

Info im Web für Mosquitto

https://iotbytes.wordpress.com/mosquitto-mqtt-broker-on-raspberry-pi/

http://www.steves-internet-guide.com/mosquitto-logging/

https://learn.adafruit.com/diy-esp8266-home-security-with-lua-and-mqtt/configuring-mqtt-on-the-raspberry-pi

12. Schlussüberlegungen, Kritik

Nun, da der Switcher 2 wunderbar läuft, ist es an der Zeit, im Nachgang einige Gedanken zur Implementierung der neuen Anforderungen formulieren:

- Realisierung der Saison-Anforderung: dies hätte einfacher gemacht werden können: schon zum Zeitpunkt des Parsens eines einzigen Steuerfiles (zB. Sommer) hätten die Schaltzeiten für Zwischensaison und Wintersaison durch Subtraktion ermittellt werden können - basierend auf den Sommer-Schaltzeiten:
  - Zwischensaison: subrahiere 1 Stunde von jeder vorkommenden Schaltzeit,
  - Wintersaison: subtrahiere 3 Stunden von jeder vorkommenden Schaltzeit.
- Noch etwas einfacher wäre, in den Steuerfiles auf absolute Zeitangaben zu verzichten und nur relative Schaltsequenzen pro Tag und Dose zu definieren. Beispiel: Dose 1 am Montag zuerst 90 Min ein, dann 30 Min aus und danach nochmals 50 Min ein. Mit einem Lichtsensor (Helligkeitsmessung) könnte dann diese Schaltsequenz ausgelöst, also gestartet werden. Damit ist das Problem der saisonalen Lichtunterschiede einfach gelöst. Für Testumgebung müsste dieser Startzeitpunkt manuell ausglöst werden können.
- Interprozess-Kommunikation mit zeroMQ Framework: die Kommunikation zwischen Switcher 2 und Client einerseits und dem Webserver andererseits, hätte voll auf Basis MQTT Broker realisiert werden können - der Broker wird ja für Schaltart 3 sowieso verwendet. Vielleicht in Version 3...
- Der Switcher hätte auch noch vermehrt in OO codiert werden können: es hätte eine Klasse ‚Haus' definiert werden können, welche ihrerseits die notwendigen Dosen instantiiert hätte. Die Hausklasse hätte dann von einem einfachen Loop alle 10 Sekunden angestossen werden können: es ist jetzt 12 Uhr 20 und 25 Sekunden - gibt es bei einer der Dosen etwas zu tun, dann tue es. Pro Haus hätte es dann einen Set von Steuerfiles gegeben. Die Hierarchie wäre dann so gewesen: Haus-Dosen-Aktoren.

13. Weiterausbau, Zukunft

Genauso wie jedes andere Software Projekt ist das Entwicklungs- und Testende für Switcher 2 willkürlich und arbiträr definiert. Irgenwann sagt der Entwickler: jetzt ist genug. Software Engineering lebt mit der Aussage: Es gibt keine Software ohne Fehler, höchstens Software, deren bestehende Fehler noch nicht ans Licht gekommen sind.

So ist es auch beim Switcher 2.

Im Hinblick auf Weiterausbau könnten folgende Punkte genannt werden:

- Die Anzahl der vorkommenden Dosen ist nicht wirklich gut implementiert. An einer Stelle im Swit-

cher 2 wird die Anzahl bestimmt gemäss der maximal vorkommenden Anzahl <dose-nr> Einträge in den Steuerfiles. Man könnte aber auch der Ansicht sein, die Anzahl der Dosen sollte im Config File stehen - weil ev. nicht alle vorkommenden Dosen auch automatisch geschaltet werden müssen. Auch das Webinterface ist fix für 4 Dosen ausgelegt. Dies ist alles verbesserungswürdig.

- Wie oben erwähnt, könnte die jetzt implementierte Saison Sache vereinfacht werden. Es gäbe dann nur noch einen einzigen Steuerfile,
- und und...

14. Links

Meine Raspberry Projekte im Web
projects.descan.com

Erste Idee für die Ansteuerung des Handsenders 2014 in einem Artikel von Hermann Kurz gefunden. Er verwendete den ELRO Handsender, da dieser bereits den Sende-Protokoll-Chip PT2262 sowie einen 433 MHz Sender enthält.
http://www.heise.de/hardware-hacks/projekte/Funksteckdose-im-Internet-1814688.html

Interprocess-Kommunikation mittels ZeroQM Framework
http://zeromq.org

Real Time Clock Modul mit Batterie von Adafruit
http://www.adafruit.com/product/264

433 Mhz Sende/Empfangen und Ermitteln der Funksequenzen. Siehe dazu die geniale Anleitung und Software des Knight of Pi.
Link zum Artikel

Zum Senden/Empfangen wird die Klasse rpi-rf von Micha LaQua verwendet - danke für die Vorarbeit. Dieser Code wurde angepasst, damit die eigene MyPrint Klasse verwendet werden.
rpi-rf on GitHub

Send Module 433 Mhz für Raspberry
siehe hier

Protokoll Beschreibung MQTT Sonoff (Tasmota)
MQTT Info für Sonoff Produkte
Ueber MQTT im Smart Home

Zum Thema Inport in Python sind diese Texte lesenswert
The Definive Guide to Python Import
Traps for the unwary in Pythons Import System

Sonoff Smart Switch und dessen Konfiguration
Bezugsquelle Smart Switch mit Tasmota Firmware
Wichtiges zu Tasmota
Setting up the Tasmota Smart Switch
Konfiguration Tasmota (Wiki)
Konfiguration Tasmota 2

Beste Firma für Ideen und Anleitungen
https://www.adafruit.com

In der CH zu empfehlen, führt Adafruit Produkte - leider recht überteuert.
http://www.play-zone.ch

15. Anhang

15.1 Steuer File

Ein Steuer-File in XML Format beinhaltet alle Schaltsequenzen für alle Tage der Woche pro Dose und fro Saison. Es gibt 3 solcher Files, einen für jede Saison: Sommer, Winter, Zwischensaison.

Hier ein Ausschnitt des Files für den Sommer:

```
<aktionen saison = „Sommersaison“>
<file_id>Sommer 4 Dosen</file_id>
<from_date>01.April</from_date>
<to_date>18.September</to_date>

<!--
Dose Nummer 1  Wohnzimmer*****************************
-->

<dose name = „Wohnzimmer“>
<dose_nr> 1 </dose_nr>

<tag nummer = „0“>
 <sequence>
       <ON>00.10</ON>
       <OFF>00.30</OFF>
  </sequence>
  <sequence>
       <ON>20.20</ON>
       <OFF>23.55</OFF>
  </sequence>
  </tag>

<tag nummer = „1“>
  <sequence>
       <ON>00.15</ON>
       <OFF>00.25</OFF>
  </sequence>
 <sequence>
       <ON>21.05</ON>
       <OFF>23.50</OFF>
  </sequence>
  </tag>

<tag nummer = „2“>
   <sequence>
       <ON>20.25</ON>
       <OFF>23.42</OFF>
  </sequence>
  </tag>


(...)
```

15.2 Main Loop Switcher

```
-------------------------------------------
Main Loop Switcher2  (Pseudocode mit {}
-------------------------------------------
while True {                                          MAIN-LOOP
    start_tag festellen                               aktueller Wochentag ermitteln (0 bis 6, Sonntag bis Samstag)

    for tag in range (start_tag , 7) {                  LOOP-Tage über Tage von start_tag bis Tag 6
        hole anzahl aktionen für diesen wochentag     Anzahl Aktionen für den aktuellen Tag ermitteln

        for j in range (aktionen_pro_tag) {           LOOP-1 über alle Aktionen des Tages
            hole nächste_aktion
            if nächste_aktion in vergangenheit         Erledige alle Aktionen die vor der aktuellen Zeit liegen
```

```
                mache aktion, schalte dose
                start_action = j
            else
                break
        }
    }

    for i in range (start_action, aktionen_pro_tag)  {      LOOP-2  Erledige alle restlichen Aktionen des Tages
          warte bis nächste Aktion fällig
          schalte Dose
    }
                                                            Alles abgearbeitet, Nix mehr zu tun für den aktuellen Tag

    while True {                                            LOOP-WAIT
        warte auf Tageswechsel                              Warte im Loop auf den neuen Tag (Mitternacht)
        if neuer Tag {
            Saison ermitteln                                hat Saison gewechselt ?
            wochentag = neuer Tag                           dies ist der neue Tage
            start_tag = 0                                   wir starten wieder bei Tag 0 (Sonntag)
            break
        }

        check IPC
    }

}                                                           Ende des MAIN-LOOP
```

15.3 Code Änderungen für Switcher2

Ausgangspunkt war der Code des originalen Switchers aus 2014. Für Switcher2 wurden folgende Erweiterungen/Änderungen vorgenommen (nicht vollständige und ungeordnete Liste).

- Function/Klasse myPrint() wurde für alle Module verbessert, ist nun in Modul myprint.py - wird für Debug-Output verwendet: Logging zur Console und in einen LogFile. Debug-Output ist abhängig von eingestelltem Debug_Level und ebenfalls vom individuellen Parameter im myPrint-Aufruf.
- Alle oder fast alle print statements in den Modulen sind ersetzt durch myPrint()
- Commandline Argumente werden neu mit argparse geparst
- swclient2.py verwendet eine allgemeine Klasse für IPC für den clientseitigen Teil der Interprocess Kommunikation mit dem Switcher. Dasselbe Modul wird auch vom Switcher Webserver swserver2.py verwendet. Die Socket-Adressse kommt aus dem Config-File.
- Es gibt neu einen Config-File swconfig.ini, der einige Parameter extern einstellbar macht.
- Saison eingeführt: der Switcher kennt nun 3 Saisons: Sommer, Winter und Zwischensaison. Ausgabe der Saisons mit swlist.py. Zudem gibt es spezielle Steuerfiles fürs Testen in denen die Schaltaktionen tagsüber sind.
- Die Schaltlogik wurde komplett redesigned : der Switcher-Client kann auch einzelne Dosen ein- oder ausschalten. Sehr erweitere Commands für den Client, siehe Tabelle im Text. Ebenfalls erweitere Commendline Parameter des Clients selbst.
- Die Statusausgabe des Switchers ist deutlich erweitert worden.
- Es wurde ein Signal Handler eingebaut, der SIGTERM abfängt und behandelt.
- Statusabfrage: die Daten werden in einer List of List gesammelt (Modul swipc2.py). Zum Senden wird diese List in ein JSON Objekt umgewandelt (also in einen String). Der Client empfängt den String und wandelt das JSON Objekt wieder in die urprünglich List of List zurück. Diese Liste wird dann ausgegeben.
- Für fast alle aufrufbaren Module gibt es die Commandline Parm -d und -D für kleinen und grossen Debug-Output.

- Lesen der Steuerfiles wurde ins Modul swparse2.py verlagert (Parsen war schon früher in diesem Modul).
- Einlesen und Parsen des Config-Files wurde in der Funktion config_read() implementiert - dies ist im Modul swfunc2.py
- Im Main Modul switcher2.py gibt es neu die Funktion get_saison(), welche ihrerseits die Funktion check_saison() aufruft, diese ist in swparse2.py.
- Alle SubModule (action, ipc und saison) haben neu eine Init-Funktion, die alle im Init des Switchers aufgerufen werden. Damit kann die Umgebung dieser Modul initialisiert werden.
- Globale Variablen Definition defglobal2.py wurden aufgeräumt und nicht benötigte Variablen wurden entfernt. Init der Listen wurde verbessert.
- Das Schalten der Dosen wurde ausgelagert in die Module swaktor1.py bis swaktor_5.py.
- Switcher hat bisher keine gute Statusmeldung geliefert in der Zeit ‚alle Aktionen des Tages vorbei' und Mitternacht. In dieser Zeitspanne hatte er auch keine Clientanfrage beantwortet. Dies ist verbessert.
- Kommentare teilweise ergänzt zur Erhöhung der Verständlichkeit.
- Dosenschaltung mit 4 simplen Gpio Pins ist realisiert und wurde für Test verwendet mit 4 Led. Läuft einwandfrei. Ist Modul swactor_1.py.
- Erweiterung der Requests, die der Client dem Switcher senden kann, Liste findet sich in der Doku.
- .... und und...

15.4 Raspberry Pi Projekt

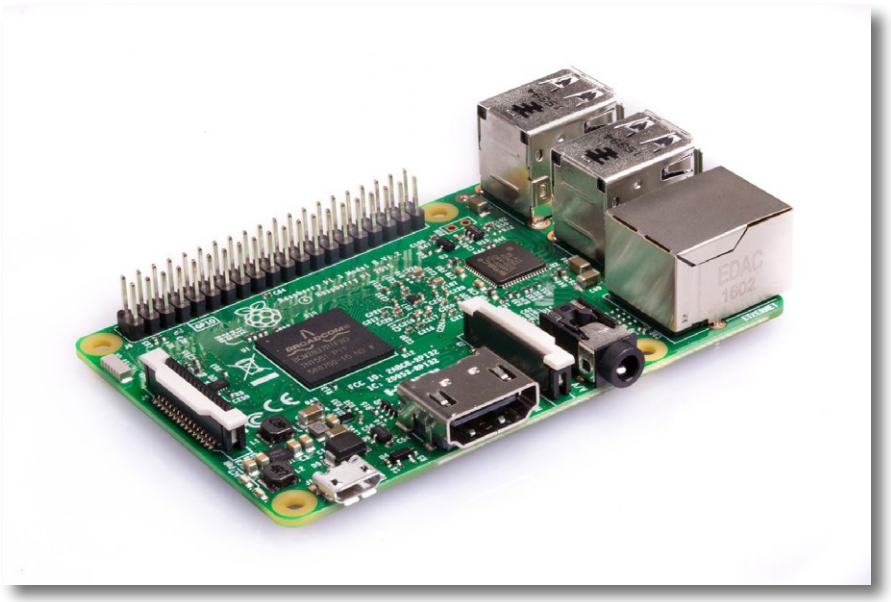
Der Raspberry Pi (genannt Pi) ist ein kreditkarten-kleiner und preisgünstiger (45 CHF) Linux Computer der sich seit seiner Einführung im Jahre 2012 grosser Beliebtheit erfreut und der bereits mehrere Millionen Mal verkauft wurde. Im Internet finden sich unzählige Projektbeschreibungen; alles Mögliche und Unmögliche wird mit diesem Micro-Computer gebaut. Der Initiant Eben Upton ist Engländer und in folgendem Video auf YouTube erklärt die Geschichte des Pi. Der vielseitige und günstige Einplatinen-Computer sollte ursprünglich britische Schüler zum Programmieren anregen. Zur großen Überraschung des Erfinders Eben Upton verkauften sich eine Million Raspberry Pi - und sie finden zunehmend auch kommerzielle Verwendung

Eben Upton on YouTube

Der Pi hat 4 USB-Anschlüsse, einen Lan-Anschluss, ein HDMI-Anschluss für Monitor und einen Audio-Video-Ausgang. Ebenfalls auf dem Board ist ein Anschluss für die Pi-Camera - eine HD-fähige Kleinst-Camera für Fotos und Video. Was den Pi auszeichnet, ist die 40-polige Steckerleiste. Viele dieser Anschlüsse sind General Purpose Input/Output Pins, welche in Progammen angesteuert/gelesen werden können. Es gibt in der Zwischenzeit vielfältige Expansion Boards von unzähligen Anbietern. Der Pi konsumiert bloss 1.5 Watt Leistung: 5V und 300mA Strom.

Dieses Maschinchen hat mich von Anfang an fasziniert, ich hatte mir in 2012 eines angeschafft. Nach einigem Probieren legte ich es auf die Seite - fand einfach kein wirkliches Projekt. Dies hat sich im Frühjahr 2014 geändert und es kam die Idee auf für eine Lampensteuerung bei Ferienabwesenheit.

Raspberry Pi bei Wikipedia



Der Raspberry PI 3 Model B

15.5 Liste aller Schaltzeiten ausgeben

Hier der Output des Scripts swlist_action2.py

```
XML file found: /home/pi/switcher/control.xml
Control-File-ID : 4 Dosen Sommer bei Peter


Liste 1:  Aktionen pro Wochentag ---------------

Wochentag: Sonntag - Anzahl Aktionen: 14
Zeit: 00.10 Dose: 1  Switch On
Zeit: 00.30 Dose: 1  Switch Off
Zeit: 19.55 Dose: 3  Switch On
Zeit: 20.35 Dose: 2  Switch On
Zeit: 20.45 Dose: 1  Switch On
Zeit: 21.12 Dose: 3  Switch Off
Zeit: 21.15 Dose: 2  Switch Off
Zeit: 22.00 Dose: 1  Switch Off
Zeit: 22.10 Dose: 2  Switch On
Zeit: 22.45 Dose: 1  Switch On
Zeit: 22.45 Dose: 2  Switch Off
Zeit: 22.52 Dose: 4  Switch On
Zeit: 23.04 Dose: 4  Switch Off
Zeit: 23.15 Dose: 1  Switch Off
Wochentag: Montag - Anzahl Aktionen: 18
Zeit: 00.15 Dose: 1  Switch On
Zeit: 00.25 Dose: 1  Switch Off
Zeit: 20.15 Dose: 2  Switch On
Zeit: 20.17 Dose: 3  Switch On
Zeit: 21.05 Dose: 1  Switch On
Zeit: 21.14 Dose: 2  Switch Off
Zeit: 21.30 Dose: 3  Switch Off
Zeit: 22.17 Dose: 4  Switch On
Zeit: 22.20 Dose: 1  Switch Off
Zeit: 22.29 Dose: 4  Switch Off
Zeit: 23.05 Dose: 1  Switch On
Zeit: 23.18 Dose: 3  Switch On
Zeit: 23.18 Dose: 4  Switch On
Zeit: 23.20 Dose: 2  Switch On
Zeit: 23.30 Dose: 4  Switch Off
Zeit: 23.40 Dose: 2  Switch Off
Zeit: 23.42 Dose: 1  Switch Off
Zeit: 23.47 Dose: 3  Switch Off
Wochentag: Dienstag - Anzahl Aktionen: 16
```

```
Zeit: 19.50 Dose: 2  Switch On
Zeit: 20.40 Dose: 3  Switch On
Zeit: 20.55 Dose: 1  Switch On
Zeit: 21.15 Dose: 2  Switch Off
Zeit: 21.45 Dose: 1  Switch Off
Zeit: 21.45 Dose: 3  Switch Off
Zeit: 21.55 Dose: 4  Switch On
Zeit: 22.15 Dose: 4  Switch Off
Zeit: 22.20 Dose: 1  Switch On
Zeit: 23.22 Dose: 1  Switch Off
Zeit: 23.36 Dose: 3  Switch On
Zeit: 23.36 Dose: 4  Switch On
Zeit: 23.45 Dose: 2  Switch On
Zeit: 23.46 Dose: 4  Switch Off
Zeit: 23.56 Dose: 2  Switch Off
Zeit: 23.56 Dose: 3  Switch Off
Wochentag: Mittwoch - Anzahl Aktionen: 24
Zeit: 00.02 Dose: 1  Switch On
Zeit: 00.05 Dose: 2  Switch On
--
......


Liste 3: Schaltaktionen Graphisch ------------

Sonntag
17       18        19        20        21        22        23        24        01        02
-----------------------------------000000000000--------0000----------000----------------
Dose 1
---------------------------------000000----------00000-----------------------------------
Dose 2
-----------------------------000000000000-------------------------------------------------
Dose 3
-------------------------------------------------------00-------------------------------
Dose 4


Montag
17       18        19        20        21        22        23        24        01        02
---------------------------------------000000000000--------000000------0-----------------
Dose 1
-------------------------------0000000000---------------------000-----------------------
Dose 2
-------------------------------000000000000-------------------0000----------------------
Dose 3
-----------------------------------------------------00---------00-----------------------
Dose 4


Dienstag
17       18        19        20        21        22        23        24        01        02
---------------------------------------00000000------0000000000---------------------------
Dose 1
-----------------------------00000000000000------------------------0----------------------
Dose 2
-----------------------------------00000000000-------------------000----------------------
Dose 3
-------------------------------------------------000--------------0----------------------
Dose 4


Mittwoch
17       18        19        20        21        22        23        24        01        02
-----------------------------------000000000000---------0000000----000-----------------
Dose 1
-----------------------------------000000000------000---------------000-----------------
Dose 2
-----------------------------------000000000000---------------000---000----------------
Dose 3
-------------------------------------------------------00---00-----------------------0----
Dose 4


Donnerstag
```

```
17       18        19        20        21        22        23        24        01        02
-----------------------------------------00000000000---0000-------000-----------------
Dose 1
-----------------------------------0000000-------------000-------------------------------
Dose 2
-----------------------------0000000000000000000------000000-----------------------------
Dose 3
-----------------------------------------------------00-----------0---------------------
Dose 4


Freitag
17       18        19        20        21        22        23        24        01        02
-------------------------------------000000----000000000000---------0-------------------
Dose 1
---------------------------------000000000--------000---------------0000----------------
Dose 2
-----------------------------------000000-------------------000------0000----------------
Dose 3
-------------------------------------------0--------------0--------0-------------------
Dose 4


Samstag
17       18        19        20        21        22        23        24        01        02
---------------------------------00000000000000000-----------0000-----000-----------------
Dose 1
---------------------------------0000000000000-----------------------------------------
Dose 2
---------------------------------000000000-------------------------------------------
Dose 3
-------------------------------------------------------------0---------------------------
Dose 4


End Aktionen Graphisch ----------------------

Program terminated....
```

This page intentionally left blank (last page)