Adapter (Aktoren / Initiatoren)

Eine Sammlung von Anbindungsmöglichkeiten verschiedener Aktoren und Initiatoren an die Cap'n Proto API per Scripts.

Tasmota

- man kann bei den Clients für TLS einfach das LetEncrypt Root Cert hinlegen, dann können die Serverseitigen Certs verifiziert werden.
- https://www.tasmota.info/hardware/
- Nous A1T kann man mit Tasmota nur vorgeflashed kaufen!
- Die Gosund EP2 (Steckdose von Usmart) ist ein Nachfolger der Gosund SP111
  - Durch seine kompakte Bauweise passt dieser Stecker in fast jede Ecke und auch lässt er sich recht einfach OTA-Flashen.
  - Diese Steckdose misst außerdem zuverlässig den Strom, vorallem nach Kalibrierung.
  - Durch seinem beliebten Vorgänger, einer maximalen Belastung von bis zu 2500 Watt und die Fähigkeit Strom zu messen ist dieses Allround-Talent das „neue“ optimale Gerät um Tasmota aufzuspielen.
  - Mit einem Pi konnte ich Gosund stecken leicht Flashen, ohne sie zu öffnen (offizieller Weg)
  - https://wiki.gorjup.de/doku.php?id=public:ota - Tuya Convert geht nicht mehr!
- hilfreiche Links
  - https://www.tasmota.info/anleitungen
    - Messfunktion kalbrieren
    - Geräte retten
    - Tasmota Update
- Hinweis: Nous ist die bessere Dose, weil sie 3680 Watt kann, EP2 von Gosund nur 2500 Watt

Tasmota Regel zur Nachlaufsteuerung von Geräten (60 Sekunden)

Beigesteuert von Joris Bijkerk:

```
backlog switchmode1 1; rule1 1


on Event#Switch1#State=1 do
   backlog power1 on; ruletimer1 0
endon
```

```
on Event#Switch1#State=0 do
   ruletimer1 60
endon

on rules#timer=1 do
   power1 0
endon
```

Verschiedene Shelly-Aktoren in bffh.dhall einbetten

> Beispiel-Config von Nils Roßmann (FabLab Karlsruhe e.V.)

Nils hat einen Generischen Aktor erstellt, der noch größtenteils kompatibel zu dem Shelly Aktor ist. Wenn man kein Topic angibt, dann erzeugt es ein Shelly Gen 1 kompatibles Topic. Es wurde erfolgreich getestet mit

- Shelly Plug S
- Shelly plus 1PM (Gen2 API)
- Gosund Plug mit Tasmota Firmware
- Wahrscheinlich auch kompatibel mit Shelly mit 2 Relays

```
actors = {
   plug004 = { module = "MqttSwitch", params = {topic = "cmnd/plug004/POWER", onMsg = "ON", offMsg = "OFF" } },
   shellyplus1pm-XXXXXXXXXXXX = { module = "MqttSwitch", params = {topic = "shellyplus1pm-XXXXXXXXXXXX/rpc", onMsg = "{\"id\": 1, \"src\": \"bffh\", \"method\": \"Switch.Set\", \"params\": {\"id\": 0, \"on\": true}", offMsg="{\"id\": 1, \"src\": \"bffh\", \"method\": \"Switch.Set\", \"params\": {\"id\": 0, \"on\": false}" } },
   shellyplug-s-XXXXXXXXXXXX = { module = "MqttSwitch", params = {=} }
},

actor_connections = [
  {machine = "MachineA1", actor = "plug004"},
  {machine = "MachineA2", actor = "shellyplug-s-XXXXXXXXXXXX"},
  {machine = "MachineA3", actor = "shellyplus1pm-XXXXXXXXXXXX"},
] : List { machine : Text, actor : Text },
```

Aktor: Tasmota

FabAccess bietet über Aktoren-Schnittstellen die passenden Möglichkeiten, um per Python Script eine entsprechende Verbindung zu Tasmota-basierten Geräten aufzubauen.:

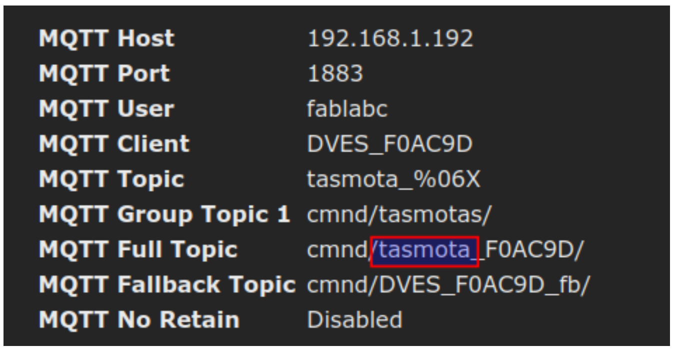
https://gitlab.com/fabinfra/fabaccess/actors/tasmota

Zum Koppeln der Schaltsteckdose mit FabAccess wird einerseits die Wifi-Verbindung zwischen Steckdose und Netzwerk benötigt, andererseits auch eine Datenverbindung per MQTT-Protokoll.

Installation

```
cd /opt/fabinfra/

git clone https://gitlab.com/fabinfra/fabaccess/actors/tasmota.git adapters/tasmota

chmod +x adapters/tasmota/main.py

chown -R bffh:bffh /opt/fabinfra/adapters/tasmota


python3 -m venv env

. env/bin/activate #activate venv

pip install -r requirements.txt
```

Achtung: in main.py sind einige Angabe statisch. Das Topic "tasmota_" wird vorrangestellt, sodass in die bffh Konfiguration nur noch die ID eingetragen werden muss. Hier im Beispiel "F0AC9D"



MQTT Host	192.168.1.192
MQTT Port	1883
MQTT User	fablabc
MQTT Client	DVES_F0AC9D
MQTT Topic	tasmota_%06X
MQTT Group Topic 1	cmnd/tasmotas/
MQTT Full Topic	cmnd/tasmota_F0AC9D/
MQTT Fallback Topic	cmnd/DVES_F0AC9D_fb/
MQTT No Retain	Disabled

Testen

Das Script kann manuell (unabhängig von bffh) getestet werden, um auszuschließen, dass es Probleme mit dem Server an sich gibt:

```
# Grunsätzlicher Syntax:
/opt/fabinfra/adapters/tasmota/env/bin/python3 /opt/fabinfra/adapters/tasmota/main.py

usage: main.py [-h] --host HOST [--port [PORT]] [--user USER] [--password PASSWORD] --tasmota TASMOTA
name {free,inuse,tocheck,blocked,disabled,reserved,raw} ...
main.py: error: the following arguments are required: --host, --tasmota, name, state

#--user USER = Nutzer des MQTT Servers
#--password PASSWORD = Passwort des MQTT Servers
#TASMOTA name = Device Name, aber ohne führendes "tasmota_"
#userid = FabAccess Nutzer (users.toml)

#Gerät "tasmota_1" als "admin" user nutzen (aktivieren)
/opt/fabinfra/adapters/tasmota/env/bin/python3 /opt/fabinfra/adapters/tasmota/main.py --host localhost --user
fabinfra101 --password fablocal --tasmota 1 state inuse Admin

#Gerät "tasmota_1" wieder freigeben (ausschalten)
/opt/fabinfra/adapters/tasmota/env/bin/python3 /opt/fabinfra/adapters/tasmota/main.py --host localhost --user
fabinfra101 --password fablocal --tasmota 1 state free
```

In FabAccess einbinden

bffh.dhall Snippet

```
        YOUR_ACTOR_ID =
        {
           module = "Process",
            params =
            {
                cmd = "/usr/bin/python3",
                args = "/opt/fabinfra/adapters/tasmota/main.py --host 127.0.0.1 --user MQTT_USER --password
MQTT_PASSWORD --tasmota YOUR_ACTOR_ID",
            }
        },
```

FabAccess Config Generator Snippet

```
vim /opt/fabinfra/fabaccess-config-generator/actors.ini
```

```
[tasmota]
module = Process
param_cmd = "/opt/fabinfra/adapters/tasmota/env/bin/python3"
param_args = "/opt/fabinfra/adapters/tasmota/main.py --host 127.0.0.1 --user MQTT_USER --password
MQTT_PASSWORD --tasmota $actor_id"
```

Aktor: Audiodateien (*.mp3) abspielen

Das folgende Aktorscript spielt beispielhaft die 8-Bit Chiptune Melodie von Mac Gyver ab. Folgende Setupschritte werden dafür benötigt. Das Setup basiert auf einem Raspberry OS (Debian 12 Bookworm) auf einem Raspberry Pi 3 B+.

Dieser Sound Actor könnte zum Beispiel verwendet werden, um folgende Dinge zu verrichten:

- Alarmtöne und standardisierte Meldungen in der Werkstatt abspielen / abbrechen
- per TTS (Text to Speech) parametrierte Texte in Sprache umwandeln und abspielen
- einzelne Jingles oder Playlists wiedergeben

Konzept und Installation

Grundsätzlich gibt es zwei Dateien: `play_mp3.py` und `main.py`. Dieses Aktorscript basiert auf dem Python Process Template.

`play_mp3.py` macht nichts, außer eine mp3 abzuspielen. Dazu verwenden wir die am Raspberry Pi bereitgestellte 3,5mm Audioklinke, an der wir einen gewöhnlichen Lautsprecher anstecken und konfigurieren die Soundausgabe entsprechend darauf.

`main.py` startet `play_mp3.py`, wenn in der Client App auf `USE` geklickt wird und beendet den Prozess per kill, wenn auf `GIVEBACK` geklickt wird.

```
cd /opt/fabinfra/adapters/
git clone https://gitlab.com/fabinfra/fabaccess/actors/python_process_template.git mp3play
cd /opt/fabinfra/adapters/mp3play/
python3 -m venv env
. env/bin/activate #activate venv

pip install pygame psutil
```

Wir müssen die Alsa-Audiokonfiguration anpassen und das Standardwiedergabegerät setzen

```
vim /etc/asound.conf
```

```
pcm.!default {
type asym
playback.pcm {
type plug
slave.pcm "hw:1,0"
}
}
```

> Falls die Wiedergabe zu leise ist, kann diese mit `alsamixer` justiert werden.

Berechtigungen anpassen

```
sudo usermod -aG audio bffh
```

Script files

```
cd /opt/fabinfra/adapters/mp3play/play_mp3.py
```

```
#!/opt/fabinfra/adapters/mp3play/env/bin/python3

import pygame

def play():
    print("In Use")
    file = '/opt/fabinfra/adapters/mp3play/8bit-macgyver.mp3'
    pygame.init()
    pygame.mixer.init()
    pygame.mixer.music.load(file)
    pygame.mixer.music.set_volume(1.0)
    pygame.mixer.music.play()

    while pygame.mixer.music.get_busy() == True:
        pass

if __name__ == "__main__":
    play()
```

```
cd /opt/fabinfra/adapters/mp3play/main.py
```

```
#!/opt/fabinfra/adapters/mp3play/env/bin/python3

import argparse
import psutil
import subprocess

def on_free(args, actor_name):
    PROCNAME = "play_mp3.py"
    for proc in psutil.process_iter():
        if PROCNAME in " ".join(proc.cmdline()):
            proc.kill()

def on_use(args, actor_name, user_id):
    cmd = "/opt/fabinfra/adapters/mp3play/env/bin/python3 /opt/fabinfra/adapters/mp3play/play_mp3.py"
    try:
        proc = subprocess.Popen(cmd, shell=True, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, start_new_session=True)
    except OSError as e:
        raise OSError("{0}\nCommand failed: errno={1} {2}".format(' '.join(cmd), e.errno, e.strerror))

def on_tocheck(args, actor_name, user_id):
    print("To Check")

def on_blocked(args, actor_name, user_id):
    print("Blocked")

def on_disabled(args, actor_name):
    print("Disabled")

def on_reserve(args, actor_name, user_id):
    print("Reversed")

def on_raw(args, actor_name, data):
    print("Raw")



def main(args):
    new_state = args.state
    if new_state == "free":
        on_free(args, args.name)
    elif new_state == "inuse":
        on_use(args, args.name, args.userid)
    elif new_state == "tocheck":
        on_tocheck(args, args.name, args.userid)
    elif new_state == "blocked":
        on_blocked(args, args.name, args.userid)
    elif new_state == "disabled":
        on_disabled(args, args.name)
    elif new_state == "reserved":
        on_reserve(args, args.name, args.userid)
    elif new_state == "raw":
        on_raw(args, args.name, args.data)
    else:
        print("Process actor called with unknown state %s" % new_state)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("name", help="name of this actor as configured in bffh.dhall")

    subparsers = parser.add_subparsers(required=True, dest="state")

    parser_free = subparsers.add_parser("free")

    parser_inuse = subparsers.add_parser("inuse")
    parser_inuse.add_argument("userid", help="The user that is now using the machine")

    parser_tocheck = subparsers.add_parser("tocheck")
    parser_tocheck.add_argument("userid", help="The user that should go check the machine")

    parser_blocked = subparsers.add_parser("blocked")
    parser_blocked.add_argument("userid", help="The user that marked the machine as blocked")

    parser_disabled = subparsers.add_parser("disabled")

    parser_reserved = subparsers.add_parser("reserved")
    parser_reserved.add_argument("userid", help="The user that reserved the machine")

    parser_raw = subparsers.add_parser("raw")
    parser_raw.add_argument("data", help="Raw data for for this actor")
    args = parser.parse_args()
    main(args)
```

```
chown -R bbfh:bffh /opt/fabinfra/adapters/mp3play/
```

Das Script manuell testen

```
# Die mp3 abspielen
/opt/fabinfra/adapters/mp3play/env/bin/python3 /opt/fabinfra/adapters/mp3play/play_mp3.py

# Die mp3 abspielen, aber per Aktor-Script
/opt/fabinfra/adapters/mp3play/env/bin/python3 /opt/fabinfra/adapters/mp3play/main.py state inuse Admin

# Die mp3 stoppen, falls sie noch läuft
/opt/fabinfra/adapters/mp3play/env/bin/python3 /opt/fabinfra/adapters/mp3play/main.py state free
```

bffh.dhall Snippet

```
        mp3play =
        {
           module = "Process",
            params =
            {
                cmd = "/opt/fabinfra/adapters/mp3play/env/bin/python3",
                args = "/opt/fabinfra/adapters/mp3play/main.py",
            }
        },
```

FabAccess Config Generator Snippet

```
vim /opt/fabinfra/fabaccess-config-generator/actors.ini
```

```
[mp3play]
module = Process
param_cmd = "/opt/fabinfra/adapters/mp3play/env/bin/python3"
param_args = "/opt/fabinfra/adapters/mp3play/main.py state inuse $actor_id"
```

TTS-Beispiel

Folgendes Snippet kann verwendet werden, um Text in mp3 zu verwandeln und diese dann im Anschluss abzuspielen (mit dem VLC Player Kommando `vlc`). Dafür wird gTTS (Google) verwendet:

```
from gtts import gTTS
import os
text = 'Hallo liebe Werkstattnutzer. Wir schließen die Werkstatt um 20 Uhr. Es ist jetzt 19.45 Uhr. Bitte räumt eure letzten Sachen auf. Wir wünschen euch einen guten Abend. Kommt gut nach Hause!'
language = 'de'
obj = gTTS(text=text, lang=language, slow=False, tld=language)
obj.save("Werkstattansage.mp3")
os.system("vlc Werkstattansage.mp3")
```

Aktor: FabLight

https://gitlab.com/fabinfra/fabaccess/actors/fablight

Aktor: FabLock

https://gitlab.com/fabinfra/fabaccess/actors/fablock

Aktor: Fabpel

https://gitlab.com/fabinfra/fabaccess/actors/fabpel

Aktor: FabReader

https://gitlab.com/fabinfra/fabaccess/actors/fabreader

Aktor: Generisches Python-Template für "Process"

Generisches Python-Template

Das nachfolgende Template ist Vorlage für die meisten in Python geschriebenen Aktoren:

https://gitlab.com/fabinfra/fabaccess/actors/python_process_template

Aktor: Logger (CSVlog)

Dieser Aktor ist eine Contribution vom Makerspace Bocholt

```
mkdir -p /opt/fabinfra/adapters/csvlog/
```

```
cd /opt/fabinfra/adapters/csvlog/
python3 -m venv env
. env/bin/activate #activate venv

pip install paho-mqtt
```

```
vim /opt/fabinfra/adapters/csvlog/main.py
```

```
#!/usr/bin/env python3

import sys
import argparse
import paho.mqtt
import paho.mqtt.publish as publish
import paho.mqtt.client as mqtt
from csv import writer
import time

def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)



def on_free(args, actor_name):
    """
    Function called when the state of the connected machine changes to Free
    again
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, 'leer', 'released']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: release")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_use(args, actor_name, user_id):
    """
    Function called when an user takes control of the connected machine

    user_id contains the UID of the user now using the machine
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'booked']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: inUse")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_tocheck(args, actor_name, user_id):
    """
    Function called when an user returns control and the connected machine is
    configured to go to state `ToCheck` instead of `Free` in that case.

    user_id contains the UID of the manager expected to check the machine.
    The user that used the machine beforehand has to be taken from the last
    user field using the API (via e.g. the mobile app)
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'tocheck']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: toCheck")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_blocked(args, actor_name, user_id):
    """
    Function called when an manager marks the connected machine as `Blocked`

    user_id contains the UID of the manager that blocked the machine
    """



    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'blocked']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: blocked")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)



def on_disabled(args, actor_name):
    """
    Function called when the connected machine is marked `Disabled`
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'disabled']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: disabled")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_reserve(args, actor_name, user_id):
    """
    Function called when the connected machine has been reserved by somebody.

    user_id contains the UID of the reserving user.
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'reserved']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: reserved")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)



def main(args):
    """
    Python example actor

    This is an example how to use the `process` actor type to run a Python script.
    """

    if args.verbose is not None:
        if args.verbose == 1:
            print("verbose output enabled")
        elif args.verbose == 2:
            print("loud output enabled!")
elif args.verbose == 3:
            print("LOUD output enabled!!!")
        elif args.verbose > 4:
            print("Okay stop you're being ridiculous.")
            sys.exit(-2)
    else:
        args.verbose = 0
    # You could also check the actor name here and call different functions
    # depending on that variable instead of passing it to the state change
    # methods.

    new_state = args.state
    if new_state == "free":
        on_free(args, args.name)
    elif new_state == "inuse":
        on_use(args, args.name, args.userid)
    elif new_state == "tocheck":
        on_tocheck(args, args.name, args.userid)
    elif new_state == "blocked":
        on_blocked(args, args.name, args.userid)
    elif new_state == "disabled":
        on_disabled(args, args.name)
    elif new_state == "reserved":
        on_reserve(args, args.name, args.userid)
    else:
        print("Process actor called with unknown state %s" % new_state)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # Parameters are passed to the Process actor as follows:
    # 1. the contents of params.args, split by whitespace as separate args
    # 2. the configured id of the actor (e.g. "DoorControl1")
    # 3. the new state as one of [free|inuse|tocheck|blocked|disabled|reserved]

    parser.add_argument("-q", "--quiet", help="be less verbose", action="store_true")
    parser.add_argument("-v", "--verbose", help="be more verbose", action="count")

    parser.add_argument("name",
                        help="name of this actor as configured in bffh.dhall"
                        )

    # We parse the new state using subparsers so that we only require a userid
    # in case it's a state that sets one.
    subparsers = parser.add_subparsers(required=True, dest="state")

    parser_free = subparsers.add_parser("free")
    parser_inuse = subparsers.add_parser("inuse")
    parser_inuse.add_argument("userid", help="The user that is now using the machine")

    parser_tocheck = subparsers.add_parser("tocheck")
    parser_tocheck.add_argument("userid", help="The user that should go check the machine")

    parser_blocked = subparsers.add_parser("blocked")
    parser_blocked.add_argument("userid", help="The user that marked the machine as blocked")

    parser_disabled = subparsers.add_parser("disabled")

    parser_reserved = subparsers.add_parser("reserved")
    parser_reserved.add_argument("userid", help="The user that reserved the machine")

    args = parser.parse_args()
    main(args)
```

```
chown -R bbfh:bffh /opt/fabinfra/adapters/csvlog/
```

Aktor: Server herunterfahren (Shutdown)

Ein simples Aktorscript in Python, um den Server herunterzufahren - zum Beispiel für eine Maintenance.

Konzept und Installation

Grundsätzlich gibt es eine einzelne `main.py`. Dieses Aktorscript basiert auf dem Python Process Template. Das Script führt `sudo /sbin/shutdown -h now` aus und verhindert, dass dieser Status nach dem Neustart erhalten bleibt. Dazu arbeiten wir mit einem Lock File `shutdown.lock`, welches im Verzeichnis des Scripts (`/opt/fabinfra/adapters/shutdown/`) abgelegt oder gelöscht wird - je nach Aktion.

Berechtigungen anpassen

Wir erlauben dem Nutzer `bffh` das Ausführen des Befehls `shutdown` per `sudo`.

```
sudo echo "bffh ALL=NOPASSWD: /sbin/shutdown" > /etc/sudoers.d/bffh
```

Script files

```
mkdir -p /opt/fabinfra/adapters/shutdown/
vim /opt/fabinfra/adapters/shutdown/main.py
```

```
import argparse
import psutil
import subprocess
import os

'''
This actor scripts shuts down the server, if no shutdown.lock file is existent (pressing "USE" in the client". The lock file is needed because otherwise the server will ALWAYS shutdown as long as the state of the actor was not set back. So we trigger only if the lock file was removed. The removal of the lock file is done in the client by "GIVEBACK"
'''

file_path = os.path.join(os.path.dirname(__file__), "shutdown.lock")
def on_free(args, actor_name):
    if os.path.exists(file_path):
        os.remove(file_path)

def on_use(args, actor_name, user_id):
    try:
        with open(file_path, 'x') as file:
            file.write("DO NOT DELETE")
            cmd = "sudo /sbin/shutdown -h now"
            try:
                proc = subprocess.Popen(cmd, shell=True, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            except OSError as e:
                raise OSError("{0}\nCommand failed: errno={1} {2}".format(' '.join(cmd), e.errno, e.strerror))

    except FileExistsError:
        print("The file '{}' already exists".format(file_path))

def on_tocheck(args, actor_name, user_id):
    print("To Check")

def on_blocked(args, actor_name, user_id):
    print("Blocked")

def on_disabled(args, actor_name):
    print("Disabled")

def on_reserve(args, actor_name, user_id):
    print("Reversed")

def on_raw(args, actor_name, data):
    print("Raw")



def main(args):
    new_state = args.state
    if new_state == "free":
        on_free(args, args.name)
    elif new_state == "inuse":
        on_use(args, args.name, args.userid)
    elif new_state == "tocheck":
        on_tocheck(args, args.name, args.userid)
    elif new_state == "blocked":
        on_blocked(args, args.name, args.userid)
    elif new_state == "disabled":
        on_disabled(args, args.name)
    elif new_state == "reserved":
        on_reserve(args, args.name, args.userid)
    elif new_state == "raw":
        on_raw(args, args.name, args.data)
    else:
        print("Process actor called with unknown state %s" % new_state)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("name", help="name of this actor as configured in bffh.dhall")

    subparsers = parser.add_subparsers(required=True, dest="state")

    parser_free = subparsers.add_parser("free")

    parser_inuse = subparsers.add_parser("inuse")
    parser_inuse.add_argument("userid", help="The user that is now using the machine")

    parser_tocheck = subparsers.add_parser("tocheck")
    parser_tocheck.add_argument("userid", help="The user that should go check the machine")

    parser_blocked = subparsers.add_parser("blocked")
    parser_blocked.add_argument("userid", help="The user that marked the machine as blocked")

    parser_disabled = subparsers.add_parser("disabled")

    parser_reserved = subparsers.add_parser("reserved")
    parser_reserved.add_argument("userid", help="The user that reserved the machine")

    parser_raw = subparsers.add_parser("raw")
    parser_raw.add_argument("data", help="Raw data for for this actor")

    args = parser.parse_args()
    main(args)
```

```
chown -R bbfh:bffh /opt/fabinfra/adapters/shutdown/
```

Das Script manuell testen

```
#USE
/usr/bin/python3 /opt/fabinfra/adapters/shutdown/main.py state inuse 1


#GIVEBACK
/usr/bin/python3 /opt/fabinfra/adapters/shutdown/main.py state free
```

bffh.dhall Snippet

```
        shutdown =
        {
           module = "Process",
            params =
            {
                cmd = "/usr/bin/python3",
                args = "/opt/fabinfra/adapters/shutdow/main.py",
            }
        },
```

FabAccess Config Generator Snippet

```
vim /opt/fabinfra/fabaccess-config-generator/actors.ini
```

```
[shutdown]
#that script is so simple we do not need a special venv for it!
module = Process
param_cmd = "/usr/bin/python3"
param_args = "/opt/fabinfra/adapters/shutdown/main.py"
```

Initiator: Shelly Timeout

Maschine automatisch nach Leerlaufzeit freigeben mit Shelly

Das folgende Script setzt ein Shelly nach einer Idle-Zeit (wenn der Stromschwellwert länger als `TIME_THRESHOLD` Minuten (Standard: 15 Minuten) unterhalb Schwellwert `POWER_THRESHOLD` (Standard: 0) ist) zurück in die Ausgangslange (führt einen Reset zum Status `FREE` durch).

Code: https://gitlab.com/fabinfra/fabaccess/shelly-timeout