Plugins (Aktoren / Initiatoren)

Eine Sammlung von Anbindungsmöglichkeiten verschiedener Aktoren und Initiatoren (Erklärung siehe hier) an die FabAccess-API durch BFFH Plugins (bezeichnen wir auch als Adapter).

Mit Hilfe von Plugins realisieren wir z.B. die Verbindung zwischen FabAccess BFFH und physischer FabHardware durch Auswerten von MQTT-Nachrichten.

- Verschiedene Shelly-Aktoren in bffh.dhall einbetten
- Aktor: Audiodateien (*.mp3) abspielen
- Aktor: eq3-eqiva-smartlock
- Aktor: FabLight
- Aktor: FabLock
- Aktor: Fabpel
- Aktor: FabReader
- Aktor: Fail Actor (für Debugging Zwecke)
- Aktor: Generisches Python-Template für "Process"
- Aktor: Machine Logger (CSVlog)
- Aktor: Server herunterfahren (Shutdown)
- Aktor: spacermake (Primary-Secondary mit Nutzungsprotokoll)
- Aktor: Tasmota
- FabFire Tools
- Initiator: Generisches Python-Template für "Process"
- Initiator: Shelly Timeout

Verschiedene Shelly-Aktoren in bffh.dhall einbetten

Nils R. vom FabLab Karlsruhe e.V. hat das folgende Konfigurationsbeispiel für einen **MqttSwitch** Aktor erstellt, das größtenteils kompatibel zum Shelly Aktor ist. Wenn in folgender Definition kein Topic angegeben wird, 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" }
  ]
}
```

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

pip3 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 proc.status() != psutil.STATUS_ZOMBIE:
            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: eq3-eqiva-smartlock

Ein Aktor zum Steuern von eQ3 Eqiva Türschlössern

- Sources: https://gitlab.com/fabinfra/fabaccess/actors/eq3-eqiva-smartlock
- Dokumentation: eqiva Bluetooth Smart Türschlossantrieb

Aktor: FabLight

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

Aktor: FabLock

Der FabLock Actor wird verwendet, um Türschlösser über unseren Client oder die API (z.B. pyfabapi) zu bedienen. Er basiert prinzipiell auf dem Template Aktor: Generisches Python-Template für "Process". Der Aktor unterstützt darüber hinaus jedoch noch die dafür notwendigen Spezialargumente, die über den Spezialzustand `raw` (binäre Daten) übermittelt werden können (siehe hier) und welche in der Dhall-Hauptkonfiguration über das spezielle Trait Prodable definiert werden. Dafür gibt es im Borepin Client die Aktionsknöpfe `UNLOCK` und `IDENTIFY`. Diese senden passende Argumente an den Prozess:

- `"action: unlock"`
- `"action: identify"`

> Die Hard- und Software unterstützt nur nicht-permanente Schlösser.

Zur Kommunikation mit der FabLock-Hardware wird MQTT verwendet.

Jedes physische FabLock-Modul muss dafür eine eindeutige ID haben. Die ID ist 5 Ziffern lang und hat führende Nullen (`--fablock` Parameter), also z.B. `00001`.
Um die einzelnen Schlösser im FabLock-Modul zu unterscheiden, hat jedes eine 5-stellige ID mit führenden Nullen (`--lock` Parameter), also z.B. `00091`.

Da verschiedene Schlösser unterschiedliche Auslösezeiten haben und diese eingehalten werden müssen, um die Funktionalität des Schlosses zu erhalten, können diese nur an der Hardware eingestellt werden. Die LED blinkt alle 500ms Sekunden, wenn sie identifiziert wird. Und leuchtet dauerhaft, wenn das Schloss entriegelt ist. Der Zustand des Riegels wird alle 30 Sekunden gemeldet.

Quellcode für den Aktor: https://gitlab.com/fabinfra/fabaccess/actors/fablock

Installation

```
mkdir -p /opt/fabinfra/adapters/
cd /opt/fabinfra/adapters/
git clone https://gitlab.com/fabinfra/fabaccess/actors/fablock.git
chmod +x fablock/main.py
chown -R bffh:bffh /opt/fabinfra/adapters/fablock/
```

```
cd /opt/fabinfra/adapters/fablock/
python3 -m venv env
. env/bin/activate #activate venv
pip install -r requirements.txt
```

Konfigurationsparameter

- `--host` MQTT Server Addresse
- `--port` MQTT Server Port
- `--user` MQTT User (optional)
- `--password` MQTT Passwort (optional)
- `--fablock` FabLock ID
- `--lock` Lock ID

MQTT Befehle

Das allgemeine Schema lautet: `/fablock/[FabLock-ID]/[Lock-ID]/[event]`. In unserem Python Aktor verwenden wir im Einklang mit der dafür geschriebenen Firmware (FabLock for ESP8266 oder FabLock for Ants Make AM-022) die folgenden Topics:

- `/fablock/00001/00001/trigger` - Schloss betätigen- die passende Aktion für den Borepin-Button `UNLOCK`
- `/fablock/00001/00001/identify` - LED blinken lassen und visuell leicht erkennbar machen - die passende Aktion für den Borepin-Button `IDENTIFY`
- `/fablock/00001/00001/feedback` - Status des Schlossriegels (Deadbolt) erhalten

In FabAccess einbinden

Eine nützliche Beispielkonfiguration für FabLocks findet sich in der tfom2023-Demo.

bffh.dhall Snippets

Ressource

```
        LBoxx_1 =
        {
            name = "FabLock Tools",
            description = "LBoxx with Tools of the FabLock Project",
```

```
            disclose = "tfom23.disclose",
            read = "tfom23.read",
            write = "tfom23.lboxx.write",
            manage = "tfom23.manage",
            category = "LBoxx",
            prodable = True,
        },
```

Aktor

```
        fablock_lboxx_1 = {
        module = "Process",
        params = {
            cmd = "python",
            args = /opt/fabinfra/adapters/fablock/main.py --host 127.0.0.1  --user MQTT_USER
--password MQTT_PASSWOR --fablock 00001 --lock 00001"
            }
       },
```

Zuweisung

```
    actor_connections = [
        { machine = "LBoxx_1", actor = "fablock_lboxx_1" }
        ],
```

Aktor: Fabpel

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

Aktor: FabReader

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

Aktor: Fail Actor (für Debugging Zwecke)

Wer einen einfachen Aktor für Testzwecke benötigt, welcher keine reale Ressource im Zustand modifiziert, der kann sich einen einfachen Aktor dafür bauen, direkt als Shell-Script. Dieser löst weiter nichts aus wie eine beispielhafte Fehlerausgabe mit dem Exit Code `115` auf dem Fehlerkanal `stderr`.

vim fail-actor.sh

```
mkdir -p /opt/fabinfra/adapters/fail-actor/
```

```
vim /opt/fabinfra/adapters/fail-actor/fail-actor.sh
```

```
#!/usr/bin/env bash


echo "This is some error output" > /dev/stderr
exit 115
```

```
chmod +x /opt/fabinfra/adapters/fail-actor/fail-actor.sh
```

Aktor: Generisches Python-Template für "Process"

Generisches Python-Template für Aktoren

Das nachfolgende Template ist Grundlage für die meisten in Python geschriebenen Aktoren-Plugin und dient dazu Zustände von Ressourcen von einem Zustand in anderen Zustand zu verändern. Eine Übersicht über die erlaubten Zustände, die gesetzt werden können, findet sich hier.

Es implementiert einen einfachen Python-Zugang zu den in Rust geschriebenen Methoden des Prozess-Aktors. Das Template eignet sich also **nicht** für Initiatoren!

Im Code ist das passende Gegenstück (Plugin) hier zu finden.

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

> Siehe auch Initiator: Generisches Python-Template für "Process"

Aktor: Machine Logger (CSVlog)

Dieser Aktor ist eine Contribution vom Makerspace Bocholt und hat zwei Aufgaben:

1. Nutzungszeit erfassen: Alle FabAccess-Events in Bezug zu einer Ressource (Maschine) mit einem Zeitstempel in einer CSV-Liste speichern, um diese später auswerten zu können (für Abrechnung, Statistik und Co.). Erfasst wird die theoretisch gebuchte Zeit des Nutzer, als auch die tatsächliche, aktive Zeit der Maschine (wenn sie erkennbar höheren Strom verbraucht) - d.h. ein Zyklus `FREE` → `INUSE` → `FREE`. All diese Daten sind deutlich detaillierter und maßgeschneiderter als der Standard Audit-Log von FabAccess - dieser kann im Zweifelsfall jedoch stets zu Rate gezogen werden - z.B. wenn es unerklärliche Auffälligkeiten im CSV-Logfile gibt.
2. eine MQTT-Nachricht auf der Topic-Basis `fabaccess/log/` mit `Actorname`, `User-ID` und `Status` erzeugen. Der `Aktorname` ist dabei die Ressourcenbezeichnung mit einem vorgesetzten Präfix `Log` z.B. `LogSchweissen2`.

> Beim Erfassen von Nutzungszeiten und Verbrauchspreisen sollte daran gedacht werden, dass Nutzer ggf. hektischer werden können, wenn es ins Geld geht. Deshalb sollte ggf. statt der gebuchten Zeit nur die tatsächliche Nutzungszeit abgerechnet werden. Das ist jedoch keine Konzeptfrage und für jeden Space eine andere Ausgangssituation.

Stromerfassung der Aktoren

Alle stromerfassenden Aktoren sind auf Basis von Tasmota Firmware, bzw. verwenden Tasmota MQTT Nachrichten. Über den Befehl `PowerHigh 20` in der Tasmota Konsole wird eine Leistungsgrenze festgelegt (in diesem Fall 20 Watt) und sagt uns damit, wann die Ressource für uns auch wirklich in Betrieb ist. Wenn die Ressource genutzt (d.h. > 20 Watt) wird, wird eine MQTT-Nachricht auf `tele/<Ressourcenname>/MARGINS` mit dem Payload `{"MARGINS":{"PowerHigh":"ON"}}` erzeugt. Wenn die Ressource < 20 Watt verbraucht, wird eine MQTT-Nachricht auf `tele/<Ressourcenname>/MARGINS` mit dem Payload `{"MARGINS":{"PowerHigh":"OFF"}}` erzeugt. Auch die etwaige installierte Mehrkanal-Stromerfassung in den Zählerschränken erzeugt ihre Nachrichten gemäß Tasmota-Format (können bei Bedarf aber auch "Shelly" sprechen).

machinelog.rs

Das `machinelog.rs` Skript hat mehrere Funktionen:

- Erfassen, wann die Ressource gebucht (`IN USE`) und zurückgegeben (`GIVEBACK`) wurde. Hierzu werden die log-Nachrichten vom jeweiligen CSVlog Aktor über MQTT ausgewertet
- Wenn eine Ressource gebucht wurde, werden die `{"MARGINS":{"PowerHigh":"ON"}}` und `{"MARGINS":{"PowerHigh":"OFF"}}` abgefangen und die Nutzungszeit aufsummiert
- Es wird jede angefangene volle Minute der Nutzungszeit eine Nachricht an den jeweiligen FabCounter oder FabReader zur Ressource geschickt (falls montiert)
- Es wird ermittelt, ob eine Ressource eine Support-Ressource (Kompressor, Absaugung, etc.) braucht und ob diese Support-Ressource bei der Buchung oder bei der Nutzung eingeschaltet werden soll, und ob diese Support-Ressource eine Nachlaufzeit benötigt
- Beim Zurückgeben (`GIVEBACK`) einer Ressource wird die Nutzung in einer CSV-Datei dokumentiert und eventuelle Support-Ressourcen gegebenenfalls hinzugerechnet

Installation

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

```
mkdir -p /opt/fabinfra/adapters/
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 kompletten Host Server herunterzufahren - zum Beispiel für eine Maintenance. Es kann auch verwendet werden, um nur den BFFH Dienst z.B. neu zu starten. Entsprechend der Wunschbefehle müssen nur die Script Files leicht angepasst werden und ggf. einzelne Befehle in die Datei `/etc/sudoers.d/bffh` eingetragen werden. Siehe auch Andere Befehle einfügen.

Konzept und Installation

Grundsätzlich gibt es eine einzelne `main.py`. Dieses Aktorscript basiert auf dem Python Process Actor 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`, indem wir es `sudoers.d` Verzeichnis einfügen:

```
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"
```

Andere Befehle einfügen

Neben einem Host-Shutdown können wir auch andere Befehle eintragen und dafür neue Aktoren anlegen und/oder die Scripts entsprechend anpassen - zum Beispiel das Lauschen auf bestimmte Argumente. So können wir unserem Nutzer `bffh` außerdem erlauben den Dienst selbstständig zu starten und zu stoppen:

```
sudo echo "bffh ALL=NOPASSWD: /usr/bin/systemctl start bffh.service" > /etc/sudoers.d/bffh
sudo echo "bffh ALL=NOPASSWD: /usr/bin/systemctl stop bffh.service" > /etc/sudoers.d/bffh
sudo echo "bffh ALL=NOPASSWD: /usr/bin/systemctl restart bffh.service" > /etc/sudoers.d/bffh
```

Wenn wir uns als bffh Nutzer einloggen, können wir dann ausführen:

```
sudo su - bffh

# als bffh Nutzer
sudo systemctl start bffh.service
sudo systemctl stop bffh.service
sudo systemctl restart bffh.service
```

Aktor: spacermake (Primary-Secondary mit Nutzungsprotokoll)

Dieser Aktor ist eine Zuarbeit vom Makerspace Bocholt. Der Aktor agiert zusammen mit Aktor: Machine Logger (CSVlog) und dient dazu, sekundär eingeschaltene Geräte in einem Log zu erfassen, damit Verbräuche zu tracken und verwendet dabei das DIY-Hardwarekonzept Primary-Secondary Schaltung. Der Aktor schickt außerdem Anzeigeinformationen an ggf. installierte FabReader oder FabCounter-Anzeigen. spacermake greift deshalb unter anderem auf die bestehende FabFire Adapter Konfiguration (`config.toml`) zurück.

spacermake installieren

Wir klonen das Projekt:

```
mkdir -p /opt/fabinfra/adapters/
git clone https://github.com/LastExceed/spacermake.git
cd spacermake/
```

Vor dem Kompilieren müssen wir noch ein paar Pfade für Konfigurations- und Logdateien anpassen. Das Script ist aktuell nicht optimal konfigurierbar und muss deshalb vor dem Nutzen geeignet angepasst und erst dann kompiliert werden!

```
vim /src/main.rs
```

```
# config Pfade anpassen
static ref SLAVES_BY_MASTER: HashMap<String, HashSet<String>> = parse_toml_file("master-slave_relations.toml");
static ref SLAVE_PROPERTIES: HashMap<String, [bool; 3]> = parse_toml_file("slave_properties.toml");
static ref MACHINE_IDS: HashMap<String, String> = parse_toml_file::<toml::Table>("/opt/fabinfra/adapters/fabfire_adapter/config/config.toml")

...

# und den MQTT Server anpassen (wir fügen auch noch eine Zeile für Benutzer und Passwort ein)
```

```
    let mut mqttoptions = MqttOptions::new("spacermake", "localhost", 1883);
    mqttoptions.set_credentials("fabinfra101", "fablocal");
```

```
vim /src/utils/logs.rs
```

```
        .open("machinelog.csv")?

# und weiter unten:

        .open("machinelog_debug.csv")?
```

Wir installieren Rust, falls noch nicht vorhanden ist:

```
# Wir installieren nun das aktuelle Rust per rustup (als normaler Nutzer). Rustup erlaubt das flexible Installieren beliebiger Rust-Versionen
curl https://sh.rustup.rs -sSf | sh

# cargo in .bashrc einfügen und Umgebung neu laden
echo 'source "$HOME/.cargo/env"' >> ~/.bashrc
source ~/.bashrc

# wir prüfen, ob wir die aktuelle Rust Version haben
rustup show

#  oder installieren sie ...
rustup install stable
rustup default stable
```

Dann erzeugen wir die Binary:

```
cd /opt/fabinfra/adapters/spacermake/
cargo build --release
```

spacermake konfigurieren und testen

```
cd /opt/fabinfra/adapters/spacermake/
```

Log Files

```
# leere Log Files anlegen. Sonst startet spacermake nicht
touch machinelog.csv
touch machinelog_debug.csv
```

Primary-Secondary (Master-Slave) konfigurieren

Die folgenden beiden *.toml Dateien müssen konfiguriert werden. Eine Beispielkonfiguration:

```
vim master-slave_relations.toml
```

```
master1 = ["slave1", "slave2"]
master2 = ["slave3", "slave1"]
```

```
vim slave_properties.toml
```

```
slave1 = [false, false, true]
slave2 = [true, true, true]
```

Die in `slave_properties.toml` angegebenen Boolean-Werte definieren folgendes:

- `runsContinuously` - Slave läuft immer (`true`) oder soll von Buchungs bis Rückgabe des Masters laufen (`false`)
- `needsTrailingTime` - Slave soll nach Abschalten des Masters 30 Sekunden nachlaufen (`true`) oder Secondary geht sofort aus (`false`)
- `isTasmota` - gibt an, ob es ein Tasmota (`true`) oder Shelly (`false`) ist

Berechtigungen anpassen

Wir übergeben die Dateien alle dem Nutzer `bffh`:

```
cd /opt/fabinfra/adapters/
chown -R bffh:bffh spacermake/
```

Manuell prüfen

Wir prüfen manuell, ob die Binary startet:

```
/opt/fabinfra/adapters/spacermake/target/release/spacermake
```

spacermake als systemd Service

```
sudo vim /etc/systemd/system/spacermake.service
```

```
[Unit]
Description=FabAccess Primary-Secondary Actor with usage log protocol
Require=network-online.target
After=network-online.target

[Service]
Type=simple
User=bffh
Group=bffh
ExecStart=/opt/fabinfra/adapters/spacermake/target/release/spacermake
Restart=always
WorkingDirectory=/opt/fabinfra/adapters/spacermake

[Install]
WantedBy=multi-user.target
```

Wir aktualisierne den Daemon, aktiveren und starten den Dienst dann:

```
sudo systemctl daemon-reload
sudo systemctl enable spacermake.service --now
```

Die Logs finden wir dann mit:

```
sudo journalctl -f -u spacermake.service
```

Aktor: Tasmota

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

Quellcode für den Aktor: https://gitlab.com/fabinfra/fabaccess/actors/tasmota

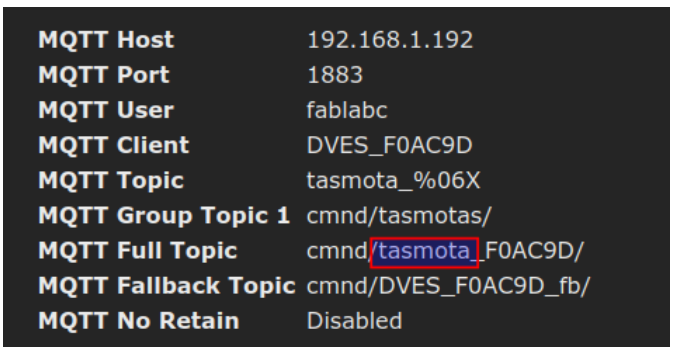
Zum Koppeln der Schaltsteckdose mit FabAccess wird einerseits die Wifi-Verbindung zwischen Steckdose und Netzwerk benötigt, andererseits auch eine Datenverbindung per MQTT-Protokoll.

Installation

```
mkdir -p /opt/fabinfra/adapters/
cd /opt/fabinfra/adapters/
git clone https://gitlab.com/fabinfra/fabaccess/actors/tasmota.git
chmod +x tasmota/main.py
chown -R bffh:bffh /opt/fabinfra/adapters/tasmota/

cd /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"



Konfigurationsparameter

- `--host` MQTT Server Addresse
- `--port` MQTT Server Port
- `--user` MQTT User (optional)
- `--password` MQTT Passwort (optional)
- `--tasmota` Tasmota ID

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"
```

FabFire Tools

Über FabFire

FabFire ist eine selbst entwickelte Spezifikation unter Nutzung von Mifare DESFire EV2 Karten im Zusammenhang mit unserem Ökosystem bestehend aus FabAccess Client, FabAccess Server und eingebundenen FabReadern. Die Eselsbrücke FabFire zu DESFire lässt sich relativ gut einprägen.

FabFire Adapter

Der FabFire Adapter übersetzt MQTT Nachrichten von der FabReader-Hardware in die API.

Hier geht's zum GitLab Repository: https://gitlab.com/fabinfra/fabaccess/fabfire_adapter

Installation

Projekt auschecken (als Benutzer `bffh`)

```
su - bffh
```

```
mkdir -p /opt/fabinfra/adapters/
cd /opt/fabinfra/adapters/
git clone https://gitlab.com/fabinfra/fabaccess/fabfire_adapter.git --recursive
cd fabfire_adapter/
git checkout rebuild #wir verwenden den aktuelleren rebuild Branch
```

Weg 1: Eine einfache, native Installation ohne Overhead ist mit der virtuellen Python3-Umgebung möglich:

```
python3 -m venv env
. env/bin/activate
pip3 install -r requirements.txt
```

Weg 2: Alternative mit dem Dockerfile und Podman:

```
podman build -f Dockerfile -t fabinfra/fabfire_adapter
```

Konfiguration

Im Unterverzeichnis `config/config.toml` werden alle FabReader eingepflegt, die angebunden werden sollen.

Server-Verbindungen

In der Sektion `[mqtt]` geben wir die Verbindung zum MQTT-Server an. Parameter sind:

- `hostname` (Pflicht)
- `port` (Pflicht)
- `username` (optional)
- `password` (optional)

> **Hinweis:** aktuell kann keine MQTTS-Verbindung definiert werden. Siehe https://gitlab.com/fabinfra/fabaccess/fabfire_adapter/-/issues/3

In der Sektion `[bffh]` geben wir die Verbindung zum BFFH Server an. Parameter sind:

- `hostname` (Pflicht)
- `port` (Pflicht)

Reader-Verbindungen

In der Sektion `[reader]` geben wir dann die Details der FabReader ein, diese sind:

- `[readers.<Reader-Name>]` als Untersektion, eingerückt mit Tabulator - je Reader wird eine neue Sektion eröffnet (siehe Beispiel)
- `id` - die FabReader ID
- `machine` - die URN der Ressource, die der FabReader kontrollieren soll

Eine Beispielkonfiguration, wie sie auch auf unserem Raspberry Pi 3 Demo Server vorzufinden ist:

```
[mqtt]
hostname = "127.0.0.1"
port = 1883
username = "fabinfra101"
password = "fablocal"
```

```
[bffhd]
hostname = "127.0.0.1"
port = 59661

[readers]
    [readers.zam-raum1-ecke1-lamp]
    id = "00001"
    machine = "urn:fabaccess:resource:zam-raum1-ecke1-lamp"

    [readers.zam-raum1-ecke2-arrow]
    id = "00002"
    machine = "urn:fabaccess:resource:zam-raum1-ecke2-arrow"
```

Benutzung

Der FabFire Adapter kann manuell wie folgt gestartet werden:

```
/opt/fabinfra/adapters/fabfire_adapter/env/bin/python3 main.py
```

Oder mit Podman:

```
podman run localhost/fabinfra/fabfire_adapter:latest
```

Ein erfolgreicher Log Output sollte so aussehen:

```
INFO:root:Registered handler for reader 00001
INFO:root:Registered handler for reader 00002
INFO:root:Initialization done
fabreader/0001/startOTA
fabreader/0001/cancelOTA
fabreader/0001/requestOTA
fabreader/0001/startOTA
fabreader/0001/cancelOTA
```

Der Adapter muss in Betrieb bleiben, damit die Leser funktionieren. Deshalb installieren wir diesen als `systemd` Service:

```
sudo vim /etc/systemd/system/fabfire_adapter.service
```

```
[Unit]
Description=FabFire Adapter - translate MQTT messages from FabReader to API calls to bffhd
After=network-online.target

[Service]
User=bffh
Restart=on-failure
WorkingDirectory=/opt/fabinfra/adapters/fabfire_adapter/
ExecStart=/opt/fabinfra/adapters/fabfire_adapter/env/bin/python3 main.py

[Install]
WantedBy=multi-user.target
```

Danach aktivieren wir den Dienst und starten ihn. Die Logs prüfen wir über `journalctl`:

```
systemctl daemon-reload
systemctl enable fabfire_adapter.service --now
journalctl -f -u fabfire_adapter.service
```

Fehlerbehebung

OSError: File not found: schema/connection.capnp

```
Feb 14 18:35:43 fabaccess python3[6956]:   File "capnp/lib/capnp.pyx", line 4365, in
capnp.lib.capnp.load
Feb 14 18:35:43 fabaccess python3[6956]:   File "capnp/lib/capnp.pyx", line 3561, in
capnp.lib.capnp.SchemaParser.load
Feb 14 18:35:43 fabaccess python3[6956]: OSError: File not found: schema/connection.capnp
```

Dieser Fehler erscheint, wenn das Git-Archiv nicht rekursiv ausgecheckt wurde oder aber `main.py` nicht aus dem korrekten Arbeitsverzeichnis (`WorkingDirectory`) aus gestartet wird.

FabFire Provisioning Tool

Das FabFire Provisioning Tool dient zur Bereitstellung neuer Karten für das FabAccess-Kartensystem.

> **Unterstützt werden nur DESFire EV2 Karten! Weitere Infos siehe Funktionsprinzip / Grundlagen**

Hier geht's zum GitLab Repository: https://gitlab.com/fabinfra/fabaccess/FabFire-Provisioning-Tool

Installation

> Falls BFFH Server bereits als Paket installiert wurde, dann ist der Installationsschritt überflüssig und kann übersprungen werden, weil `fabfire_provision` bereits im System installiert ist!

> Wir beziehen uns hier auf die fabfire_provision Version **0.1.0**. Die Version kann geprüft werden mit dem Befehl: `fabfire_provision --version`

Zunächst klonen wir das git Repository (als Benutzer, der eine Installation von `rustup` vorweist und damit die ausführbaren Befehle `cargo` und `rustc`). In unserem Demo Setup ist das der Benutzer `fabinfra-root`.

```
mkdir -p /opt/fabinfra/tools/
cd /opt/fabinfra/tools/
git clone https://gitlab.com/fabinfra/fabaccess/FabFire-Provisioning-Tool.git
fabfire_provision
cd fabfire_provision/
```

```
sudo apt install libpcsclite-dev
```

Danach kompilieren wir zunächst die Anwendung, um eine ausführbare Binary `fabfire_provision` im Ausgabeverzeichnis `/opt/fabinfra/tools/fabfire-provision/target/release/` zu erhalten:

```
cargo build --release
```

Wir kopieren diese Binary in das Allgemeinverzeichnis `/usr/bin` und passen den Eigentümer an:

```
sudo cp /opt/fabinfra/tools/fabfire_provision/target/release/fabfire_provision /usr/bin/
sudo chown root:root /usr/bin/fabfire_provision
```

Hinweis: Das Tool kann auch mit der im Projektordner beiliegeden `Cross.toml` und dem cross-rs Tool für andere Architekturen kompiliert werden:

```
sudo apt install podman
cargo install cross
```

```
cross build --target aarch64-unknown-linux-gnu --release
cross build --target=armv7-unknown-linux-gnueabihf --release
```

Benutzung

Eine allgemeine Übersicht der Programmiermöglichkeiten erhalten wir zunächst mit:

```
fabfire_provision --help
```

```
Simple program to greet a person

Usage: fabfire_provision [OPTIONS]

Options:
      --id <APP_ID>
          Application id to use [default: 4604226]
      --picc-masterkey <PICC_MASTERKEY>
          Masterkey for the PICC
      --app-masterkey <APP_MASTERKEY>
          Masterkey for the Application
      --app-authkey <APP_AUTHKEY>
          user authentication key
  -m, --magic <MAGIC>
          Magic string to identify cards [default: FABACCESSDESFIRE1.0]
  -s, --space <SPACE>
          Name of the issuing space
  -i, --instance <INSTANCE>
          BFFHd Instance for the space
  -c, --contact <CONTACT>
          Contact option for lost cards
  -t, --token <TOKEN>
          User token, currently this should be set to the Username (will be generated for you
if not given)
  -f, --format
          Whether to format the card
  -h, --help
          Print help
```

```
  -V, --version
          Print version
```

Das Provisioning Tool wird wie folgt verwendet. Zunächst wird eine Mifare DESFire EV2 Karte auf einen FabReader aufgelegt. Aktuell wird dafür aus der Konfiguration der Erstgelistete verwendet.

> Es gibt noch keinen Parameter, um einen bestimmten Reader in der Werkstatt auszuwählen (Siehe Issue #1). Wir empfehlen deshalb den als aller erstes definierten FabReader als "Anlerngerät" zu verwenden. Die Konfiguration erfolgt in der Konfiguration des FabFire Adapters.

Die Karte sollte während des Schreibvorgangs nicht vom Gerät entfernt werden. Wir führen dann den folgenden Befehl mit Parametern aus, welche sich teilweise mit unserer Hauptkonfiguration decken, um die aufgelegte Karte für den ausgewählten Nutzer (im Beispiel für den Admin Benutzer) entsprechend zu formatieren:

- `--space` - der Name des Spaces
- `--instance` - der Name der FabAccess-Instanz. Ähnlich zu instanceurl, aber ohne Protokollteil. Kann z.B. ein Hostname oder ein FQDN sein.
- `--contact` - die Angabe, wo bzw. wer im Falle des Kartenverlusts zu kontaktieren ist. Idealweise geben wir hier eine URL zu einer Kontakt- oder Impressumseite an
- `--token` - der jeweilige Benutzername, wie er in z.B. in users.toml definiert ist

```
fabfire_provision --space "FabAccess Demo Setup" --instance fabaccess.local --contact
https://fab-access.org/impressum --token "Admin"
```

Die Ausgabe des Befehls spuckt einen Schlüssel aus (cardkey), den wir in die Benutzerdatenbank importieren müssen. Das geht aktuell mit Hilfe der Datei users.toml

Weitere Argumente können mit den entsprechenden Kommandozeilenargumenten übergeben werden, das sind:

- `--app-authkey` - App Authentication Key (siehe AN12696 - Kapitel 2.7)
- `--app-masterkey` - App Master Key (siehe AN12696 - Kapitel 2.7)
- `--id` - Application Identifier (AID). Der Standard im FabFire Provisioning Tool lautet `4604226` und in FabAccess Server bzw. Borepin lautet `0x464142`. Idealerweise vergeben wir für unseren Space eine eigene AID!

- `--magic` - Der Standard Magic Key lautet `FABACCESS\0DESFIRE\01.0\0` und sollte dabei belassen werden
- `--picc-masterkey` - Der Master Key der Keycard (PICC)

Eine Karte formatieren (löschen)

Das Formatieren einer Karte löscht alle Dateien und Schlüssel geht so:

```
fabfire_provision --format
```

Fehlerbehebung

Error: NoService

Der FabReader ist nicht bereit. Er ist nicht angeschlossen oder wurde nicht erkannt.

Failed to connect to card x on reader y

> **TODO**

Failed to transmit APDU command to card: x

> **TODO**

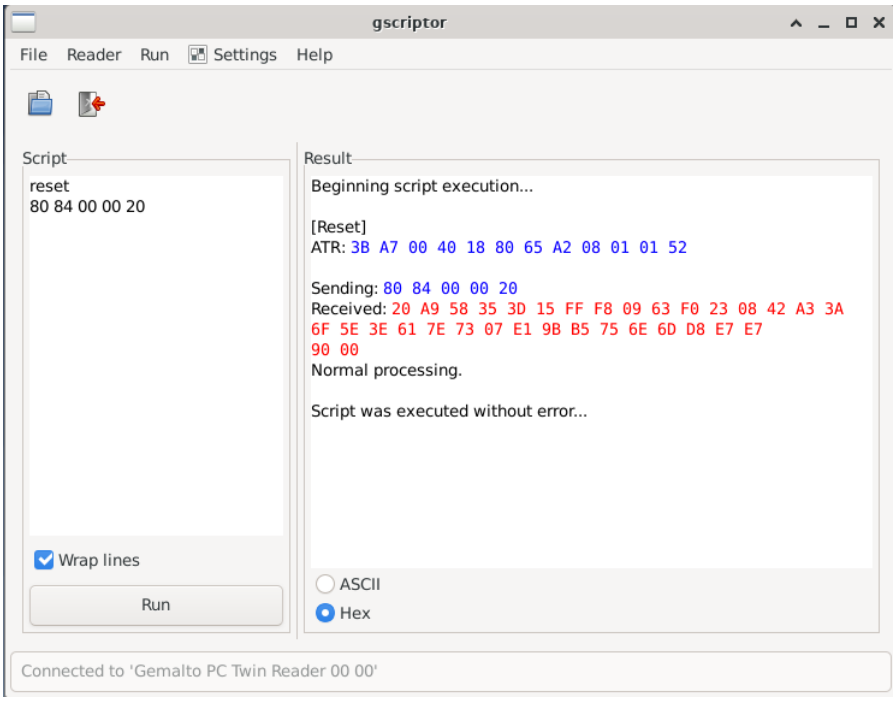
Smartcard Reader Tools unter Linux

Folgende Tools können beim Umgang mit Smartcard Readern allgemein hilfreich sein. Wir installieren dazu `pcsc-tools`. Siehe auch https://pcsc-tools.apdu.fr

```
sudo apt install pcscd pcsc-tools
sudo systemctl status pcscd.service
```

Die installierten Werkzeuge können mit folgenden Befehlen ausgeführt werden:

- `pcsc_scan`
- `ATR_analysis`
- `scriptor`
- `gscriptor`



Grafische Oberfläche mit `gscriptor`

Initiator: Generisches Python-Template für "Process"

Generisches Python-Template für Initiatoren

Ein Initiator erlaubt als das aktive Verändern des Zustand einer Ressource (siehe hier für erlaubte Zustände), an die er gebunden ist. Diese auf dieser Seite z ufindenen Templates schreiben ohne besondere Logik verschiedene Zustände einer Ressource für einen Nutzer.

Sie haben damit nur beschränkt reellen Nutzen, zeigen jedoch, wie ein Initiator (auch) zu bedienen ist.

Initiator Beispiel für BFFH-Initialisierung (einmalig beim Start)

```
mkdir -p /opt/fabinfra/adapters/initiator_process_example/
vim init.once.py
```

```
#!/usr/bin/env python

import os
import sys
import time
import logging
LOG_FILE = os.path.join(os.path.dirname(__file__), 'init.once.log')
logging.basicConfig(
    filename=LOG_FILE,
    format='%(asctime)s %(levelname)-8s %(message)s',
    level=logging.DEBUG,
    datefmt='%Y-%m-%d %H:%M:%S')

try:
    # Statements, die per print() an stdout gegeben werden, werden in bffhd geschrieben

    # Versetze die Ressource (Maschine) in den Zustand "Reserved" durch den Nutzer "Admin"
    state = '{ "state": { "Reserved": { "id": "Admin" } } }'
    print(state)
```

```
    sys.stdout.flush()
    logging.debug(state)

    # wir brauchen wenigstens ein paar Sekunden nach dem letzten flush, damit alles an bffhd
übertragen wird
    # falls der Status nicht verändert wird: in Betracht ziehen die Sleep Time zu erhöhen
    time.sleep(3)

except Exception as e:
    logging.debug(e)
    sys.exit(1)

sys.exit(0)
```

```
chmod +x /opt/fabinfra/adapters/initiator_process_example/init.once.py
```

Initiator Beispiel in Dauerschleife

Dieses Sample geht in Dauerschleife alle verfügbaren Zustände durch und schreibt sie dem Nutzer `Admin` zu.

```
mkdir -p /opt/fabinfra/adapters/initiator_process_example/
vim init.loop.py
```

```
#!/usr/bin/env python

import os
import sys
import time
import logging
LOG_FILE = os.path.join(os.path.dirname(__file__), 'init.loop.log')
logging.basicConfig(
    filename=LOG_FILE,
    format='%(asctime)s %(levelname)-8s %(message)s',
    level=logging.DEBUG,
    datefmt='%Y-%m-%d %H:%M:%S')

# Init sleep
time.sleep(10)
```

```
while True:
    try:
        # Statements, die per print() an stdout gegeben werden, werden in bffhd geschrieben

        t_sleep = 3 #darf nicht 0 sein. Sonst wechseln die Zustände zu schnell durch und verursachen Fehler und ein gefülltes BFFH Log (journalctl)

        # Versetze die Ressource (Maschine) in den Zustand "Free"
        state = '{ "state": "Free" }'
        print(state)
        sys.stdout.flush()
        logging.debug(state)
        time.sleep(t_sleep)

        # Versetze die Ressource (Maschine) in den Zustand "Blocked" durch den Nutzer "Admin"
        state = '{ "state": { "InUse": { "id": "Admin" } } }'
        print(state)
        sys.stdout.flush()
        logging.debug(state)
        time.sleep(t_sleep)

        # Versetze die Ressource (Maschine) in den Zustand "Reserved" durch den Nutzer "Admin"
        state = '{ "state": { "Reserved": { "id": "Admin" } } }'
        print(state)
        sys.stdout.flush()
        logging.debug(state)
        time.sleep(t_sleep)

        # Versetze die Ressource (Maschine) in den Zustand "ToCheck" durch den Nutzer "Admin"
        state = '{ "state": { "ToCheck": { "id": "Admin" } } }'
        print(state)
        sys.stdout.flush()
        logging.debug(state)
        time.sleep(t_sleep)

        # Versetze die Ressource (Maschine) in den Zustand "Blocked" durch den Nutzer "Admin"
        state = '{ "state": { "Blocked": { "id": "Admin" } } }'
        print(state)
        sys.stdout.flush()
        logging.debug(state)
```

```
        time.sleep(t_sleep)

        # Versetze die Ressource (Maschine) in den Zustand "Disabled"
        state = '{ "state": "Disabled" }'
        print(state)
        sys.stdout.flush()
        logging.debug(state)
        time.sleep(t_sleep)

    except Exception as e:
        logging.debug(e)
        sys.exit(1)

sys.exit(0)
```

```
chmod +x /opt/fabinfra/adapters/initiator_process_example/init.loop.py
```

> **Achtung:** Der Initiator wird nicht automatisch neu von BFFH aufgerufen, wenn das Script scheitert (z.B. wegen Schreib- oder Lesefehlern oder sonstigen Exceptions). In diesem Fall muss der BFFH Server neugestartet werden oder mehr Programmlogik in den Initiator eingebaut werden, damit sich das Script selbst neu startet!

bffh.dhall Snippet

Wir binden den Inititator entsprechend in unsere BFFH Konfiguration ein.

```
    , initiators =
      { init_once =
        { module = "Process"
        , params.cmd =
            "/opt/fabinfra/adapters/initiator_process_example/init.once.py"
        }
      , init_loop =
        { module = "Process"
        , params.cmd =
            "/opt/fabinfra/adapters/initiator_process_example/init.loop.py"
        }
      }
    , init_connections =
```

```
[ { machine = "zam-raum1-ecke1-lamp", initiator = "init_once"}
, { machine = "zam-raum1-ecke2-arrow", initiator = "init_loop"}
]
```

> Siehe auch Aktor: Generisches Python-Template für "Process"

Initiator: Shelly Timeout

Ressource 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 hardkodiert: 15 Minuten) unterhalb eines Schwellwert `POWER_THRESHOLD` (Standard: 0) ist) zurück in die Ausgangslange und führt einen Reset zum Status `FREE` durch.

Konfigurierbare Parameter (Reihenfolge ist zu beachten!):

- Host (IP, FQDN)
- Shelly_ID
- Power Treshold (Volt) als Integer (Ganzzahl)

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

Installation

```
mkdir -p /opt/fabinfra/adapters/
cd /opt/fabinfra/adapters/
git clone https://gitlab.com/fabinfra/fabaccess/initiators/shelly-timeout.git
chmod +x shelly-timeout/main.py
chown -R bffh:bffh /opt/fabinfra/adapters/shelly-timeout/

cd /opt/fabinfra/adapters/shelly-timeout/
python3 -m venv env
. env/bin/activate #activate venv
pip install -r requirements.txt
```

In FabAccess einbinden

bffh.dhall Snippet

```
        YOUR_INITIATOR_ID =
        {
           module = "Process",
            params =
            {
                cmd = "/usr/bin/python3",
                args = "/opt/fabinfra/adapters/shelly-timeout/main.py 127.0.0.1 shelly1-
REDACTED_ID 5",
            }
        },
```

Außerdem muss eine `init_connection` zu einer Ressource definiert werden, um den Initiator zu verwenden.