

# 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](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](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 beiliegenden `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

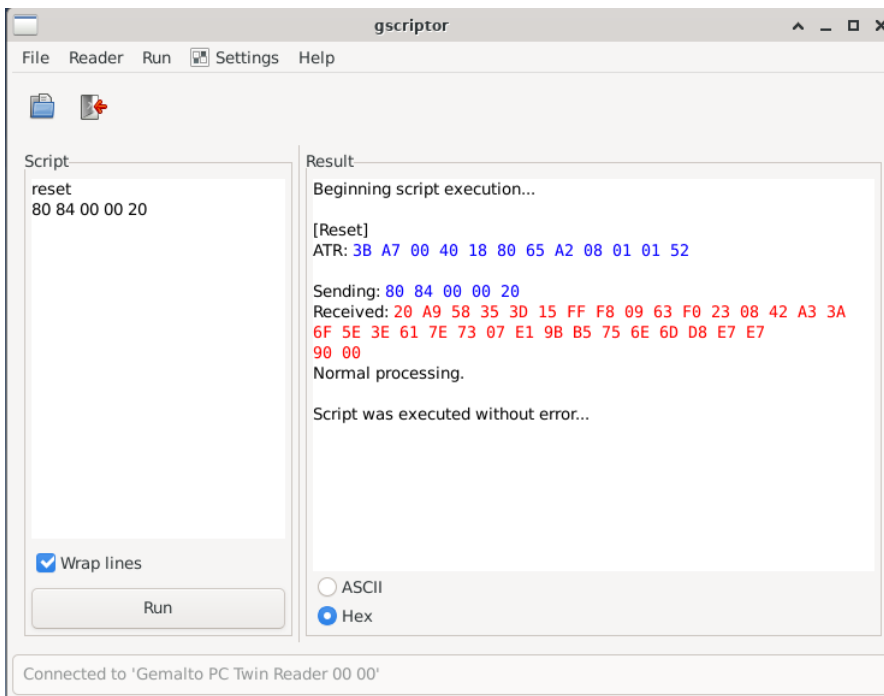
Folgende Tools können beim Umgang mit Smartcard Readern allgemein hilfreich sein. Wir installieren dazu `pcsc-tools`. Siehe auch <https://pcsc-tools.apdu.fr>

```
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`

Version #29

Erstellt: 21 Oktober 2024 12:52:07 von Mario Voigt (Stadtfabrikanten e.V.)

Zuletzt aktualisiert: 11 März 2025 17:27:00 von Mario Voigt (Stadtfabrikanten e.V.)