Zustände (Traits)

Traits bieten die Möglichkeit, den Zustand von Ressourcen zu ändern. Ressourcen können mehrere Traits besitzen und diese kombiniert nutzen. Mit Traits erhalten Nutzer Zugriff auf die Ressource, nachdem sie einen Claim erhalten haben. Dabei können Traits verwendet werden, um Ressourcen aus bestehenden Traits zusammenzusetzen oder spezifische Traits zu implementieren. Um eine optimale Anzeige der Traits für Nutzer in Clients zu ermöglichen, kann einer Ressource ein "Hint" hinzugefügt werden. Dieser ermöglicht es einem Client, eine verbesserte Ansicht der Ressource für Nutzer zu generieren.

Traits und OIDs

Traits werden anhand einer OID (Object Identifier) bereitgestellt. Die OID-Struktur von FabAccess folgt dem Schema:

```
1.3.6.1.4.1.61783.612.1.2
              |    |  | |
  RLKM UG PEN ┘    |  | |
                   |  | |
 FabAccess subtree ┘  | |
                      | |
               Traits ┘ |
                        |
               Doorable ┘
```

Für das Projekt wurde eine gesonderte Private Enterprise Numer (PEN) bei IANA auf "FabInfra" registiert. Das feste PEN-Präfix lautet `1.3.6.1.4.1`. Die PEN-Nummer für RLKM lautet `61783` (Siehe https://www.iana.org/assignments/enterprise-numbers/?q=61783.). Das Projekt FabAccess hat eine fest vergebene Unternummer `612`. Danach folgt die Unterklasse `1` (Traits) und dann die möglichen Zustandsnummern (Enumeration) `0` ... `5` (z.B. Doorable, Locatable, ...).

> In älteren Versionen der API wurde die PEN 48398 genutzt ("Paranoidlabs").

Die OID-Bezeichner werden in der Zustandsdatenbank abgespeichert. Siehe auch Datenbeispiel eines Dumps.

Übersicht über die vorhandenen Traits und OIDs

In FabAccess gibt es bereits vordefinierte Traits für grundlegende Funktionen, mit denen viele Zustände von Ressourcen abgebildet werden können.

FabAccess-API

> Aktuell ist diese API in Verwendung. Siehe
> https://gitlab.com/fabinfra/fabaccess/fabaccess-api

In diesem API-Modell gibt es nur die folgenden 7 Zustände (Traits) von 0 bis 6:

- 0: `Free` - eIne Ressource frei verfügbar für Nutzer mit Zugriff machen
- 1: `InUse:<UserId>` - eine Ressource durch einen Nutzer "In Benutzung" setzen
- 2: `ToCheck:<UserId>` - eine Ressource auf "muss überprüft werden" setzen (z.B. ob Reinigung nach Benutzung notwendig ist) - **Achtung**: Dieser Zustand kann nur über die API und nicht über Client genutzt werden (weil nicht implementiert. Siehe auch eine Ressource bedienen)
- 3: `Blocked:<UserId>` - eine Ressource blockieren (z.B. weil die Maschine defekt ist)
- 4: `Disabled` - eine Ressource deaktivieren (z.B., weil die Ressource zum aktuellen Zeitpunkt zu laut wäre und deshalb die Benutzung verboten ist)
- 5: `Reserved:<UserId>` - eine Ressource für die spätere Nutzung vorreservieren - **Achtung**: dieser Zustand kann aus dem Borepin Client nicht gesetzt werden (Siehe auch eine Ressource bedienen), jedoch im Process Aktor und im Python Process Actor Template schon
- 6: `totakeover` - eine Maschine von einem anderem Nutzer übernehmen - **Achtung** : dieser Zustand ist weder in BFFH, noch im Process Aktor, noch im Python Process Actor Template implementiert. Dieser Zustand existiert aktuell nur in der API-Beschreibung.

Neben den definierten Zuständen gibt es auch die Möglichkeit, direkte Rohdaten zu liefern:

- `Raw:<data>` - Ressourcen können bei Statuswechsel auch direkt mit Binärdaten versorgt werden, um zum Beispiel spezielle Operationen wie das Übersenden einer STL-Datei an einen 3D-Drucker zu erlauben

> Die API verwendet in einen OID-Wert (value) `1.3.6.1.4.1.48398.612.2.4` vom OID-Typ (type) `1.3.6.1.4.1.48398.612.2.14` für den Zustand (state) im Allgemeinen.

Prodable

Kommt vom englischen Wort "prodded" und bedeutet soviel wie "anstubsen". Ein Prodable ist also etwas Anstubsbares. Das lässt sich im Sinne einer Ereignisschleife verstehen, in der eine Sache die Möglichkeit hat, etwas zu tun. Wird im BFFH Server verwendet - es findet Nutzen beim FabLock. Dort hat es den Zweck vor dem Öffnen eines Schließfachs in einem Spind mit mehreren Fächern vorher kurz eine LED aufleuchten zu lassen, die signalisiert, wo genau sich das Fach befindet.

Prodable protokollieren keine Ressourcennutzung durch einen Nutzer.

Der `Prodable` Trait existiert namentlich so in der neuen C# API nicht mehr - er wurde in `Locatable` umbenannt. Weiterhin gibt es spezialisierte Traits `Doorable` für Eingangstüren und `Lockers` für Schließfächer.

FabAccess-API-cs

> Diese API Version ist aktuell noch nicht in Verwendung. Siehe https://gitlab.com/fabinfra/fabaccess/fabaccess-api-cs

Claimable (OID 1.3.6.1.4.1.61783.612.1.0)

Der Trait "Claimable" stellt einen Sonderfall dar, da er dazu dient, dass sich ein Nutzer über den Claim-Zustand einer Ressource informieren kann.

Nutzer können auf diesem Trait keine Aktionen ausführen, sondern lediglich den Zustand abfragen.

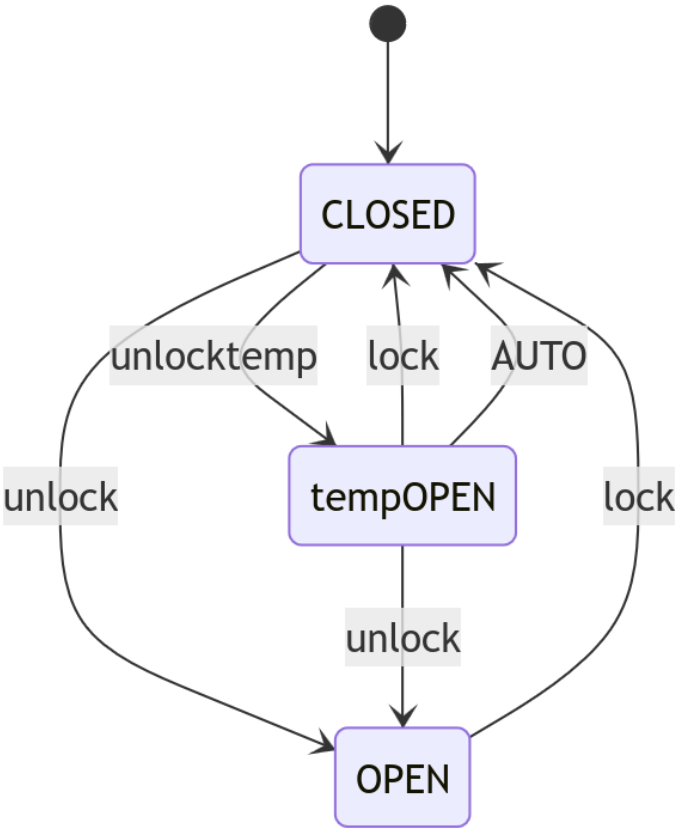
Powerable (OID 1.3.6.1.4.1.61783.612.1.1)

"Powerable" ist der grundlegendste Trait, den FabAccess unterstützt. Er dient dazu abzubilden, ob eine Ressource eingeschaltet bzw. mit Strom versorgt ist.

Doorable (OID 1.3.6.1.4.1.61783.612.1.2)

Der Trait "Doorable" ermöglicht die Abbildung von Türen oder anderen Schließsystemen. Dabei besteht die Möglichkeit, kurzzeitige Öffnungen zu realisieren. Die genaue Zeitdauer, für die die Ressource geöffnet wird, wird dabei vom Server bestimmt.

States



als Mermaid

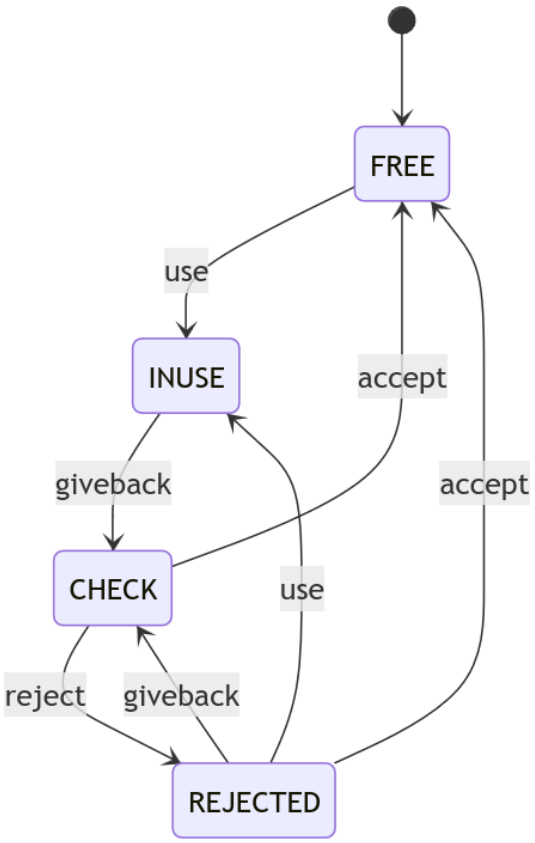
```
stateDiagram
    [*] --> CLOSED
    CLOSED --> OPEN: unlock
    OPEN --> CLOSED: lock
```

```
    CLOSED --> tempOPEN: unlocktemp
    tempOPEN --> OPEN: unlock
    tempOPEN --> CLOSED: lock
    tempOPEN --> CLOSED: AUTO
```

Checkable (OID 1.3.6.1.4.1.61783.612.1.3)

Der komplexere Trait "Checkable" ermöglicht die Abbildung von Ressourcen, die nach der Benutzung überprüft werden müssen. Bei einer Überprüfung durch einen berechtigten Nutzer kann die Ressource entweder für alle wieder freigegeben oder zurückgewiesen werden. Falls die Ressource zurückgewiesen wurde, kann der ursprüngliche Nutzer die Ressource weiterhin verwenden oder erneut zur Überprüfung einreichen, nachdem die Fehler behoben wurden.

States



als Mermaid

```
stateDiagram
    [*] --> FREE
```

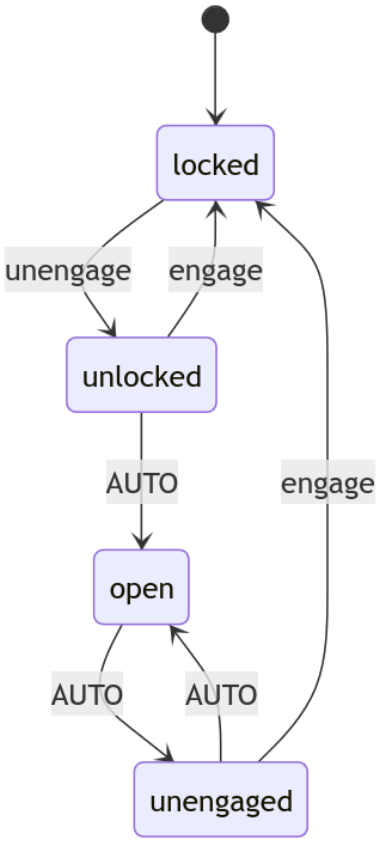
```
FREE --> INUSE: use
INUSE --> CHECK: giveback
CHECK --> FREE: accept
CHECK --> REJECTED: reject
REJECTED --> INUSE: use
REJECTED --> CHECK: giveback
REJECTED --> FREE: accept
```

Lockers (OID 1.3.6.1.4.1.61783.612.1.4)

"Lockers" ist einer der komplexeren Traits, mit dem Schlösser von Ressourcen genauer abgebildet werden können. Der Nutzer kann die Ressource nicht in jedem Zustand zurückgeben; erst wenn alles korrekt zurückgegeben wurde, kann die Ressource auch wieder gesperrt werden.

Die Zustandsänderungen zwischen den vom Nutzer verwendbaren Übergängen können von einem Initiator herbeigeführt werden.

States



als Mermaid

```
stateDiagram
    [*] --> locked
    locked --> unlocked: unengage
    unlocked --> locked: engage
    unlocked --> open: AUTO
    open --> unengaged: AUTO
    unengaged --> locked: engage
    unengaged --> open: AUTO
```

Locatable (OID 1.3.6.1.4.1.61783.612.1.5)

Der Trait "Locatable" ermöglicht die Identifizierung von Ressourcen, wie beispielsweise Schließfächer oder 3D-Drucker in einer Druckerfarm. Dabei kann entweder eine kurzfristige Identifikation abgegeben werden oder die Identifizierung dauerhaft gesetzt werden.

> **Hinweis:** Die Mermaid-Diagramme sind mit https://mermaid.live gerendert und als PNG exportiert und hier importiert worden, da BookStack keinen integrierten Mermaid Renderer besitzt.

Version #29
Erstellt: 2024-10-21 14:46:19 CEST von Mario Voigt (Stadtfabrikanten e.V.)
Zuletzt aktualisiert: 2025-04-19 22:00:54 CEST von Mario Voigt (Stadtfabrikanten e.V.)