



myminapp

Handbuch	
Datum:	2024-02-05
Programmversion:	0.9.1
Autor:	berryunit

Inhalt

Inhalt.....	2
Quickstart	3
1. Basics	5
1.1 Applikationsstruktur	5
1.2 Begriffe und Erläuterungen	6
1.3 Details.....	6
2. Applikation.....	7
2.1 Applikationsinstanzen	7
2.2 Applikationsdefinition.....	8
2.3 Applikationsserver.....	10
2.4 Web-Frontend.....	11
2.4.1 Bedienung.....	11
2.4.2 Beispiele	12
2.4.3 Sonstiges.....	19
3. Commands.....	20
4. Devices	21
5. Utilities	23
5.1 Helper	23
5.2 Logger	23
5.3 Monitor.....	23
5.4 Scheduler.....	24
5.5 Storage.....	25
6. Eigene Klassen erstellen.....	26
6.1 Erste Schritte	27
6.2 Weitere Schritte.....	30
7. Backup.....	31
Anhang	32
A. Lizenzen	32
B. Installieren mit pip.....	34
C. Zertifikate erstellen.....	35
D. Daten für statistische Zwecke aufzeichnen	39
E. Beispielszenario	42
F. Programmentwicklung	43
G. Credits	46

Quickstart

'myminapp' ist eine minimale aber vollständige Python-Applikation, die beispielhaft veranschaulicht, wie kleine DIY-Projekte im Heimbereich mit geringem Aufwand programmiert werden können.

Sie bietet die folgenden Features out-of-the-box:

- Komplette, transparente Applikationsstruktur mit Commands, Logging und Datenspeicher
- Einfache Anwendung via Python Shell sowie Einbindung in Scripts und Programme
- Applikationsserver und Web-Frontend für HTTP(S)-Anfragen
- Zeitgesteuerte Automatisierung
- Mehrsprachige Nachrichten
- Diverse Commands, die direkt nützlich sein oder als Muster dienen können
- Einfache Erweiterbarkeit zur Umsetzung individueller Aufgaben

Vorausgesetzt wird ein Rechner, auf dem Python ab Version 3.10 läuft. Für SBCs wie Raspberry Pi ist eine USB-SSD zu empfehlen, da SD-Karten für den Dauerbetrieb nicht geeignet sind.

Um myminapp zu installieren und den Command 'helloworld' auf unterschiedliche Art auszuführen, kann wie folgt vorgegangen werden.

Schritt 1 - myminapp installieren

Die Releasedatei 'myminapp-<Versionsnummer>.zip' von <https://github.com/berryunit/myminapp> herunterladen und in ein Verzeichnis mit Schreib- und Leserechten entpacken.

Im Folgenden wird angenommen, dass die Releasedatei unter Linux vom Benutzer 'u1' nach '/home/u1/myminapp' entpackt wurde. Außerdem wird angenommen, dass Python Version 3 auf dem System mit **python3** aufzurufen ist.

Schritt 2 - Command 'helloworld' via Python-Shell ausführen

1. die Python-Shell im Verzeichnis 'home/u1' (oberhalb von myminapp) starten. Dazu eingeben:

```
python3
```

2. Innerhalb der Python-Shell die folgenden Eingaben machen:

```
from myminapp.app import App
app1 = App(1)
app1.perform_command({'cmd':'helloworld', 'value':'Hello World'})
app1.close()
```

3. Die Python-Shell verlassen mit:

```
exit()
```

Schritt 3 - Anfragen via Applikationsserver ausführen

Den Applikationsserver im Verzeichnis 'home/u1' (oberhalb von myminapp) starten. Dazu eingeben:

```
python3 -m myminapp.appserver
```

3.1 - Command 'helloworld' via CURL ausführen

Ein weiteres Terminal öffnen und die folgende Eingabe machen (ohne Zeilenumbruch):

```
curl --get --data-urlencode '{"cmd": "helloworld", "value": "Hello World"}'  
http://localhost:8081
```

3.2 - Command 'helloworld' via Browser-URL-Zeile ausführen

Einen Browser öffnen und in die URL-Zeile eingeben:

```
http://localhost:8081/{"cmd": "helloworld", "value": "Hello World"}
```

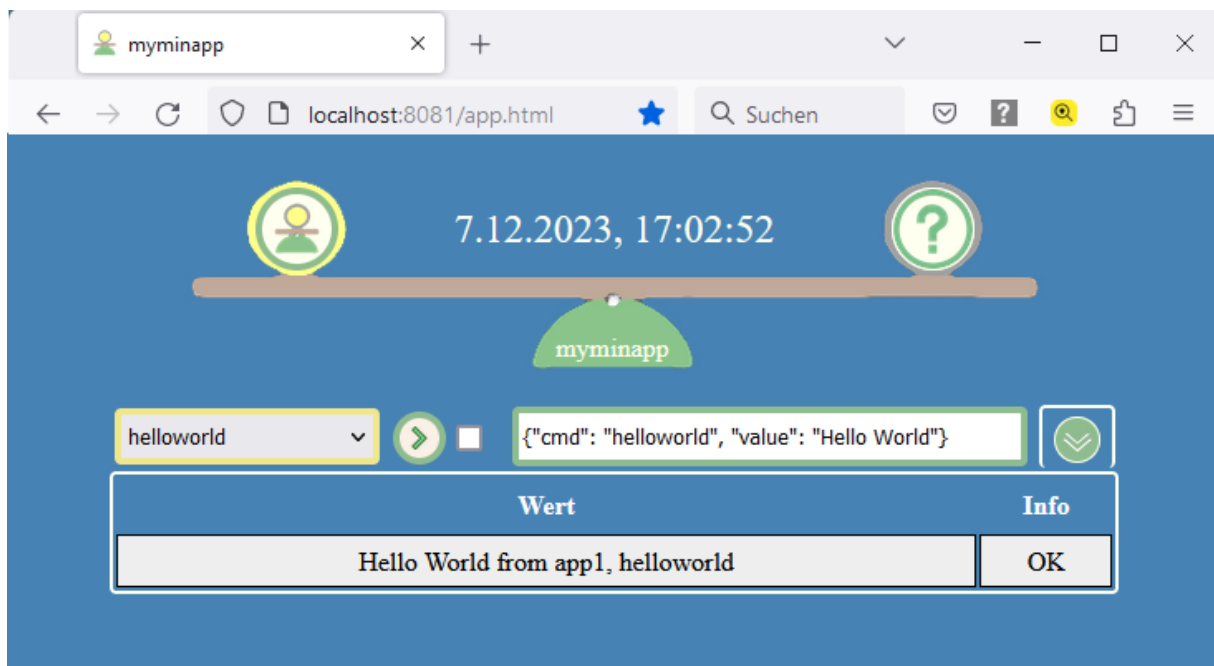
3.3 - Command 'helloworld' via Web-Frontend ausführen

1. Einen Browser öffnen und in die URL-Zeile eingeben:

```
http://localhost:8081/app.html
```

2. Den Command 'helloworld' auswählen und den Befehlsausführungs-Button drücken.

Beispiel-Bild (Mozilla Firefox, myminapp-Spracheinstellung 'de'):



Den Applikationsserver durch folgende Eingabe im Terminal beenden:

```
Strg+C
```

Schritt 4 - Command 'helloworld' via Testklasse ausführen

Im Verzeichnis 'home/u1' (oberhalb von myminapp) eingeben:

```
python3 -m myminapp.test.command.test_helloworld
```

1. Basics

Dieses Kapitel enthält grundlegende Informationen, die bekannt sein sollten.

1.1 Applikationsstruktur

Verzeichnis	Datei	Beschreibung
myminapp/		Applikationsverzeichnis
	app.py	Modul mit der zu instanzierenden Klasse 'App'
	appdef.py	Definition/Konfiguration der Applikation
	appserver.py	Applikationsserver für HTTP(S)-Anfragen
/lib/command/	*.py	Mitgelieferte und eigene Command-Klassen
/lib/device/*	*.py	Mitgelieferte und eigene Device-Klassen
/lib/text/	message.py	Mehrsprachige Nachrichtentexte
/lib/util/	helper.py	Klasse 'Helper' für Hilfsaufgaben
/lib/util/	logger.py	Klasse 'Logger' als Standardlogger-Wrapper
/lib/util/	monitor.py	Klasse 'Monitor' für das Applikationsinstanz-Monitoring
/lib/util/	scheduler.py	Klasse 'Scheduler' zur Automatisierung per Zeitplan
/lib/util/	storage.py	Klasse 'Storage' für den Datenspeicherzugriff
/web/	*.*	Web-Frontend-Dateien
/data/cert/*	*.*	Zertifikatsdateien (optional)
/data/log/	*.log	Logdateien je Applikationsinstanz (app1.log, app2.log, ...)
/data/storage/	*.db	Datendateien je Applikationsinstanz (app1.db, app2.db, ...)
/data/temp/*	*.*	Temporäre, instanzbezogene Dateien
/doc/*	*.*	Dokumentation
/test/command/	test_*.py	Mitgelieferte und eigene Command-Testklassen
/test/util/	test_*.py	Utility-Testklassen

1.2 Begriffe und Erläuterungen

Begriff	Erläuterung
Modul	Python-Modul in Form einer Datei mit Name in Kleinschrift und Endung '.py'. Modulname identisch, aber ohne Endung. Beispiele: 'helloworld.py', 'test_helloworld.py'; 'helloworld', 'test_helloworld'.
Klasse	Python-Klasse, die innerhalb einer Moduldatei definiert wird (hier immer nur eine Klasse pro Moduldatei). Klassenname entsprechend Modulname, aber ohne Unterstrich und mit Großbuchstabe beginnend. Bei mehreren Namenselementen sogenannter CamelCase. Beispiele: 'HelloWorld', 'TestHelloWorld'.
App	Klasse 'App' des Moduls 'app'. Mit einer Instanz dieser Klasse wird die Applikation gestartet (app1, app2, ...). Zur Definition/ Konfiguration dient das Modul 'appdef'. HTTP(S) Anfragen werden über das Modul 'appserver' verarbeitet.
Command	Befehl zur Ausführung einer bestimmten Aufgabe über eine Instanz der Klasse 'App'. Wird durch eine mitgelieferte oder eigene Command-Klasse implementiert, die von der Superklasse 'Cmd' abgeleitet ist.
Device	Zu verstehen im Sinne von 'Gerät', 'Bauteil', 'Element', das von Command-Klassen verwendet werden kann. Wird durch eine mitgelieferte oder eigene Device-Klasse implementiert, die ein Smarthome-Gerät, eine Kamera, einen Diagramm-Generator, einen E-Mail-Versand oder eine sonstige physische oder virtuelle Einheit repräsentiert. Device-Klassen bilden die Ausstattung für Command-Klassen.
Utility	Unterstützende Klasse wie beispielsweise 'Helper'.
Logging	Nachrichtenausgabe über die Klasse 'Logger', die ihrerseits den Python-Standard-Logger verwendet. Das Logging erfolgt unter dem Namen der Applikationsinstanz. Wird die Logausgabe in Dateien spezifiziert, so erfolgt die Ausgabe nach '/data/log/' (app1.log, app2.log, ...).
Storage	Datenspeicher zur optionalen Speicherung von Command-Ausgaben. Der Zugriff erfolgt über die Klasse 'Storage'. Der Datenspeicher ist durch die in Python enthaltene Version von SQLite implementiert. Pro Applikationsinstanz wird beim Erstzugriff automatisch eine SQLite-Datenbank unter '/data/storage/' erzeugt (app1.db, app2.db, ...).
Test	<p>Für alle mitgelieferten Command-Klassen sowie alle Utility-Klassen ist je eine Testklasse vorhanden, die von 'unittest.TestCase' abgeleitet ist. Macht ein Command Gebrauch von Device-Klassen, so werden diese implizit mit getestet, sofern die betreffenden Device-Klassen-Spezifikationen in der Applikationsdefinition nicht auskommentiert sind. Über die Klasse 'TestUtil' werden Command-Klassen dynamisch instanziiert.</p> <p>Die Namen der Testmodule/Testklassen entsprechen den Namen der zu testenden Module/Klassen, wobei 'test_' respektive 'Test' vorangestellt wird. Beispiele: 'test_helloworld', 'TestHelloWorld'.</p> <p>Für jede eigene Command-Klasse sollte eine entsprechende Testklasse nach dem gegebenen Muster umgesetzt werden. Das bedeutet: Ableitung von 'unittest.TestCase', Implementierung der öffentlichen Methode 'test_command' und der privaten Methode '__perform_command'.</p>

1.3 Details

Da myminapp sich als Beispiel-Applikation versteht, ist der Quellcode ein essentieller Bestandteil. Dieses Handbuch informiert daher auch nur grundlegend über die Verwendung von myminapp; Details können und sollen dem Quellcode entnommen werden.

Darüber hinaus sind im Anhang dieser Dokumentation periphere Details zu finden, beispielsweise zu Lizenzen, zum Zertifikat-Import und zu konzeptionellen Aspekten der Entwicklung.



2. Applikation

Im Kapitel 'Quickstart' ist beschrieben, wie myminapp aus der Releasedatei installiert werden kann. Multiple Installationen werden unterstützt: Das myminapp Applikationsverzeichnis kann einfach unter weitere Verzeichnisse kopiert werden ('myminapp_dev/myminapp', 'myminapp_test/myminapp' etc.). Bei Anwendung des Applikationsservers ist lediglich darauf zu achten, unterschiedliche Portnummern für die einzelnen Installationen zu verwenden.

Alternativ kann myminapp auch mit dem Package Installer for Python (pip) installiert werden. Siehe dazu Kapitel 'Installieren mit pip' im Anhang.

2.1 Applikationsinstanzen

Pro Installation wird die Applikation über eine oder mehrere Instanzen der Klasse 'App' ausgeführt, die in der Moduldatei 'app.py' codiert ist. Das Prinzip ist wie folgt:

- Instanziierung mit Übergabe der Instanznummer (1 für die erste Instanz, 2 für die zweite etc.)
- Aufruf der Methode 'perform_command' ein- oder mehrfach, um einen Command auszuführen
- Aufruf der Methode 'close', um Ressourcen zu schließen und die Instanz zu beenden

Zur Veranschaulichung sei das Beispiel aus dem Kapitel 'Quickstart' zitiert:

```
from myminapp.app import App
app1 = App(1)
app1.perform_command({'cmd':'helloworld', 'value':'Hello World!'})
app1.close()
```

Im Wesentlichen führt eine Instanz folgende Aufgaben aus:

Methode `__init__`:

- Den Instanznamen setzen ('app1' für Instanznummer 1, 'app2' für Instanznummer 2 etc.)
- Instanzvariablen initialisieren
- Logger und Monitor für diese Instanz öffnen
- Scheduler starten, die dieser Instanz zugeordnet sind

Methode `'perform_command'`:

- Den auszuführenden Command zuordnen
- Den Command, sofern noch nicht im Cache, instanzieren und dem Cache hinzufügen
- Die Eingabeparameter setzen, den Instanznamen ergänzen und den Command ausführen
- Das Resultat empfangen und verarbeiten
- Warnungen und Fehler verarbeiten
- Logging und Monitoring ausführen
- Optional fehlerfreie Resultate in den Datenspeicher der Instanz ausgeben
- Das Resultat zurückgeben

Methode `'close'`:

- Schließen der Scheduler
- Schließen der im Cache befindlichen Commands
- Schließen des Loggers und des Monitors
- Beenden der Instanz

Weil Logging und Datenspeicherung instanzbezogen erfolgen, sind Instanzen weitgehend unabhängig voneinander anwendbar. So wäre beispielsweise eine Instanz für die Automatisierung nutzbar und eine zweite Instanz für Aufgaben nach Bedarf, ohne dass Logging, Datenspeicher oder temporäre Daten sich überschneiden. Alle Instanzen verwenden in diesem Fall dieselbe Applikationsdefinition.

2.2 Applikationsdefinition

Die Moduldatei 'appdef.py' dient zur Definition/Konfiguration der myminapp-Applikation.

Die Moduldatei umfasst im Wesentlichen folgende Abschnitte:

- Home- und Daten-Verzeichnis, Logging und Sprache
- Felder
- Commands
- Devices
- Command-Presets
- Command-Scheduler-Sets

Im Folgenden werden die einzelnen Abschnitte kurz erläutert. Details sind wiederum der Moduldatei selbst zu entnehmen.

Home- und Datenverzeichnis, Logging und Sprache:

Die Variablen 'HOME' und 'DATA_HOME' werden bei der Instanziierung der Klasse 'App' automatisch dynamisch gesetzt. Sie sollten nicht geändert werden.

Das Logging wird über folgende Variablen spezifiziert: 'LOG_STDOUT', 'LOG_FILE', 'LOG_FILE_MAX_BYTES', 'LOG_FILE_BACKUP_COUNT' und 'LOG_LEVEL'.

Die Variable 'LANG' dient der Spracheinstellung. Aktuell werden 'en' und 'de' unterstützt.

Felder:

Es handelt sich um Datenfelder und Präfixfelder, die über alle Applikationsschichten hinweg gelten und speziell bei Commands und im Datenspeicher Verwendung finden.

Datenfelder werden in der Variablen 'FIELD_DEFSETS' definiert. Die bereits definierten Datenfelder passen zu den mitgelieferten Commands, so dass sie im Normalfall nicht geändert werden sollten. Eigene Felder können dem Muster entsprechend hinzugefügt werden.

Als Beispiel für ein Datenfeld sei hier 'value' zitiert, das für Command 'helloworld' verwendet wird:

```
FIELD_DEFSETS = {
    ...,
    "value": {
        "format": "str", "storageformat": "TEXT",
        "en": {"label": "Value", "desc": "Variable for various purposes"},
        "de": {"label": "Wert", "desc": "Variable für diverse Zwecke"}
    },
    ...,
}
```

In der Variablen 'PREFIX_FIELD_DEFSETS' sind Felder definiert, die das Präfix für jede Entität des Datenspeichers bilden. Diese Definitionen sollen nicht geändert werden.

Commands:

Über die Variable 'COMMAND_DEFSETS' werden die mitgelieferten und eigene Commands spezifiziert. Als Beispiel sei die Spezifikation für Command 'helloworld' zitiert:

```
COMMAND_DEFSETS = {
    ...,
    "helloworld": {
        "module": "myminapp.lib.command.helloworld", "class": "HelloWorld",
        "input": ["value"],
        "output": ["value", "info"],
        "storage": True, "logging": True
    },
    ...
}
```

In der ersten Zeile unter dem Namen des Commands sind der explizite Modulname sowie der Klassenname anzugeben. Diese Daten werden verwendet, um Commands dynamisch zu instanziiieren.

Die Eingabefelder sind bei 'input' als Liste zu spezifizieren, die Ausgabefelder bei 'output'. Das Feld 'info' ist generell für ein einheitliches Ausgabeformat über alle Commands hinweg erforderlich.

Mit 'storage' kann bestimmt werden, ob fehlerfreie Command-Resultate in den Datenspeicher geschrieben werden sollen, und zwar in die Entität mit dem Namen des Commands (hier also 'helloworld'). Mit 'logging' wird festgelegt, ob unabhängig vom Log-Level ein Log-Eintrag für den Command-Aufruf geschrieben werden soll.

Devices:

Die Variablen 'NAME_DEVICE_MAP' und 'DEVICE_CLASS_CONNECTION_SPEC' dienen der Spezifikation von Devices.

In 'NAME_DEVICE_MAP' werden einem Namen Device-spezifische Daten zugeordnet. Der Name muss eindeutig sein, sollte aus Großbuchstaben bestehen und nicht länger als etwa 10 Stellen sein; bei Bedarf können Unterstriche verwendet werden. Als Beispiel sei eine 'TV' genannte Device zitiert, der eine Fritz-Schaltsteckdose zugeordnet ist, die über die Device-Klasse 'Fritz' repräsentiert wird:

```
NAME_DEVICE_MAP = {
    ...,
    "TV": {"devclass": "Fritz", "devcat": "P", "devid": "12345 1234567"},
    ...
}
```

In 'DEVICE_CLASS_CONNECTION_SPEC' werden einer in 'NAME_DEVICE_MAP' spezifizierten Device-Klasse Verbindungsdaten zugeordnet. Das folgende Beispiel bezieht sich auf die Device-Klasse 'Fritz', die sich mit einer Fritzbox verbindet:

```
DEVICE_CLASS_CONNECTION_SPEC = {
    ...,
    "Fritz": {"conntype": "hub", "address": "192.168.178.1", "port": 49000,
              "user": "fritzuser", "password": "..."},
    ...
}
```

Command-Presets:

Mit der Variablen 'COMMAND_PRESETS' wird eine komfortable Command-Ausführung unterstützt. Davon wird beispielsweise im Web-Frontend Gebrauch gemacht, wenn dort ein Command ausgewählt wird, wie im Kapitel 'Quickstart' für den Command 'helloworld' gezeigt. Als Beispiel sei das Preset zitiert, das dazu definiert wurde:

```
COMMAND_PRESETS = {
    ...,
    "helloworld": {"cmd": "helloworld", "value": "Hello World"},
    ...,
}
```

Command-Scheduler-Sets:

Zwecks Automatisierung kann in der Variablen 'COMMAND_SCHDSETS' die zeitgesteuerte Ausführung von Commands spezifiziert werden, indem auf Command-Presets referenziert wird, die zuvor in 'COMMAND_PRESETS' angelegt wurden. Als Beispiel sei pro forma auch hier 'helloworld' bemüht, auch wenn die Automatisierung dieses Befehls kaum Sinn machen dürfte:

```
COMMAND_SCHDSETS = {
    ...,
    "schedule_1": {"cmdpreset": "helloworld", "days": "1,2",
                  "time": "12:59:01", "appnum": 1},
    "schedule_2": {"cmdpreset": "helloworld", "days": "3,4,5,6,7",
                  "intervalunit": 'h', "appnum": 1},
    "schedule_3": {"cmdpreset": "helloworld", "days": "*",
                  "time": "10:00:00-03:00:00", "intervaldiff": 30, "appnum": 1},
    ...,
}
```

Zur Erläuterung dieser Spezifikationen siehe die Informationen zum Scheduler im Kapitel 'Utilities'.

2.3 Applikationsserver

Die myminapp-Applikation kann optional über den Applikationsserver ausgeführt werden, der in der Moduldatei 'appserver.py' codiert ist. Der Applikationsserver verarbeitet GET-Anfragen via HTTP(S). Für Beispiele dazu sei auf das Kapitel 'Quickstart' verwiesen.

Das Modul 'appserver' verfügt über eine Main-Methode mit optionalen Aufrufparametern:

Parameter	Erläuterung
-h, --help	Zeigt einen Hilfetext zum Aufruf des Servers an.
-appnum	Instanznummer, mit der die Applikation instanziiert wird. Default ist 1.
-host	Host, auf dem der Server erreichbar ist. Default ist 127.0.0.1 alias 'localhost'. Soll der Server von außerhalb des Hosts erreichbar sein, ist die explizite IP-Adresse (oder der Hostname) anzugeben, beispielsweise 192.168.178.26.
-port	Port, unter dem der Server erreichbar ist. Default ist 8081. Soll der Server für mehrere Applikationsinstanzen parallel genutzt werden, ist die Portnummer bei jedem Server-Start zu variieren, beispielsweise 8081 für Instanz 1, 8082 für Instanz 2 etc.
-cert	Nur für HTTPS: Zertifikatsdatei, beispielsweise './myminapp/data/cert/cert.pem'.
-key	Nur für HTTPS: Schlüsseldatei, beispielsweise './myminapp/data/cert/key.pem'.

Beispielaufruf mit Angabe von Host und Port:

```
python3 -m myminapp.appserver -host 192.168.178.26 -port 8082
```

Im Wesentlichen führt der Applikationsserver folgende Aufgaben aus:

- Instanzieren der Klasse 'App' unter Berücksichtigung der optionalen Aufrufparameter
- Empfangen und Prüfen von HTTP(S)-GET-Anfragen
- Erkennen von Anfragen vom Web-Frontend am Präfix 'web='
- Konvertieren der Anfragedaten vom Typ 'JSON' zum Python-Typ 'dict'



- Aufruf der Methode 'perform_command' an der Applikationsinstanz
- Verarbeitung der Resultatdaten, mit spezieller Aufbereitung bei Anfragen vom Web-Frontend
- Konvertieren der Resultatdaten vom Python-Typ 'dict' zum Typ 'JSON'
- Senden der Antwort
- Schließen der Applikationsinstanz bei Eingabe von Strg+C im Terminal

2.4 Web-Frontend

Das Web-Frontend setzt einen laufenden Applikationsserver voraus. Läuft der Server lokal und wurde er mit Default-Werten gestartet, ist im Browser anzugeben: `http://localhost:8081/app.html`.

Läuft der Server auf einem anderen Host, ist die explizite IP-Adresse oder der Hostname anzugeben, beispielsweise: `http://192.168.178.26:8081/app.html`. In diesem Fall wird der Browser darauf hinweisen, dass die Verbindung nicht sicher ist. Im Heimnetz kann diese Warnung normalerweise toleriert und die Verbindung trotzdem hergestellt werden.

Soll die Warnung jedoch vermieden werden, ist die Kommunikation über HTTPS zu organisieren, indem entsprechende Zertifikatsdateien genutzt werden. Unter '/data/cert/' stehen dafür selbstsignierte Zertifikatsdateien (für Host-IP 192.168.178.26) sowie ein selbsterstelltes CA-Zertifikat zur Verfügung. Weitere Informationen dazu enthält der Anhang im Kapitel 'Zertifikate erstellen'.

Es sei an dieser Stelle angemerkt, dass die myminapp-Applikation nur für die Anwendung im Heimnetz geeignet ist, da sie nicht über Sicherheits-Features verfügt, die für direkte öffentliche Zugriffe erforderlich sind. Sofern ein Anwender allerdings einen geeigneten Router konfiguriert hat, um auf sein Heimnetz von außen auf sichere Weise zuzugreifen, kann auf diesem Wege prinzipiell natürlich auch ein Zugriff auf die myminapp-Applikation erfolgen.

Hinweis: Die im Folgenden dargestellten Beispielfelder wurden mit deutscher Spracheinstellung erzeugt (LANG = 'de' in 'appdef.py').

2.4.1 Bedienung

Die Anwendung des Web-Frontends ist bewusst einfach gehalten und besonders komfortabel, wenn Command-Presets definiert wurden.

Die wesentlichen Bedienelemente sind in der folgenden Abbildung mit den Ziffern 1 bis 7 markiert und weiter unten erläutert.

The screenshot shows the myminapp web frontend interface. At the top, there is a status bar with a user icon (1), the date and time '9.12.2023, 12:05:09', and a help icon (2). Below this is a green 'mymnapp' logo. The main interface features a dropdown menu (3) set to 'state_2', a green play button (4), and a command input field (5) containing '{"cmd": "state", "devname": "SOLAR, DESKTOP, FRIDGE, TV, WASHER"}'. To the right of the input field is a green checkmark button (6). Below these elements is a table (7) displaying the status of various devices.

Name	Kategorie	Zustand	Leistung W	Energie kWh	°C	Info
SOLAR	P	ON	35.4	687.783	5.0	OK
DESKTOP	P	ON	28.2	68.756	20.5	OK
FRIDGE	P	ON	3.0	84.058	24.0	OK
TV	P	ON	2.4	42.824	18.5	OK
WASHER	P	ON	0.2	9.333	22.0	OK

Zu den Bedienelementen:

- 1 = Button zum Öffnen einer Info-Box
- 2 = Button zum Öffnen des Handbuchs in einem neuen Browser-Tab
- 3 = Auswahl eines Command-Preset. Der Command wird in die Eingabezeile (5) übernommen
- 4 = Button zur Ausführung des Commands, Checkbox zur wiederholten Ausführung alle 15 Sekunden
- 5 = Eingabezeile
- 6 = Download der Datenausgabe
- 7 = Datenausgabe

2.4.2 Beispiele

Im Folgenden wird die Anwendung des Web-Frontends anhand diverser Commands beispielhaft gezeigt. Dabei wird ein Beispiel-Szenario angenommen, das zu den mitgelieferten Commands, Devices und Spezifikationen passt.

Spezifikation des Commands 'state' in 'COMMAND_DEFSETS':

```
"state": {
  "module": "myminapp.lib.command.state", "class": "State",
  "input": ["devname", "devcat"],
  "output": ["devname", "devcat", "state", "power", "energy", "temp", "info"],
  "storage": True, "logging": False
},
```

Beteiligte Presets zur Ausführung des Commands 'state' in 'COMMAND_PRESETS':

```
"state_1": {"cmd": "state", "devname": "EM"},
"state_2": {"cmd": "state", "devname": "SOLAR, DESKTOP, FRIDGE, TV, WASHER"},
```

Mit diesen Presets können Statusdaten des Energiemeters sowie diverser Schaltsteckdosen abgefragt werden.

Beteiligte Zeitplanungseinträge in 'COMMAND_SCHEDSETS':

```
"schedule_1": {"cmdpreset": "state_1", "days": "*", "intervalunit": 'm', "appnum": 1},
"schedule_2": {"cmdpreset": "state_2", "days": "*", "intervalunit": 'm', "appnum": 1},
```

Mit diesen Zeitplanungseinträgen wird der mit 'state_1' und 'state_2' spezifizierte Command zu Beginn und zum Ende eines 5-Minuten-Abschnitts ausgeführt (siehe auch Kapitel 'Scheduler').

Da für den Command 'state' die Option 'storage' True gesetzt wurde, sorgt die Applikationsinstanz dafür, dass die Ausgabedaten des Commands in der instanzbezogenen Datenbank gespeichert werden. Der Name der Entität/Datenbanktabelle entspricht immer dem Namen des Commands, hier also 'state'.

Im Prinzip kann die Applikationsinstanz im 24/7-Betrieb laufen, so dass kontinuierlich Daten für die statistische Auswertung gesammelt werden.

Spezifikation des Commands 'epstats':

```
"epstats": {
  "module": "myminapp.lib.command.epstats", "class": "EPStats",
  "input": ["type", "entity", "units", "from", "to", "devname", "devcat"],
  "output": ["fields", "entries", "charts", "info"],
  "storage": False, "logging": False
},
```



Beteiligte Presets zur Ausführung des Commands 'epstats':

```
"epstats_1_power": {"cmd": "epstats", "type": "power", "entity": "state",  
                    "units": 5, "from": "4h", "to": "**",  
                    "devname": "EM"},
```

```
"epstats_2_power": {"cmd": "epstats", "type": "power", "entity": "state",  
                    "units": 4, "from": "4h", "to": "**",  
                    "devname": "SOLAR, DESKTOP, FRIDGE, TV, WASHER"},
```

Mit diesen Presets können statistische Daten für das Energiemeter sowie diverse Schaltsteckdosen aus der Datenbanktabelle/Entität 'state' ermittelt werden.

Eine kurze Erläuterung zu einigen Parameter-Werten (Details siehe Quellcode in 'epstats.py'):

- **type** - 'power' für durchschnittliche Leistung in Watt, 'energy' für Verbrauch in Kilowattstunden, oder 'powerplus' (nur mit "devcat": "M") für durchschnittliche Einspeiseleistung in Watt, beispielsweise bei Solarenergieüberschuss
- **units** - Anzahl der Zeiteinheiten für die Gruppierung der Datenausgabe: 1=Jahr; 2=Jahr, Monat; 3=Jahr, Monat, Tag; 4=Jahr, Monat, Tag, Stunde; 5=Jahr, Monat, Tag, Stunde, Minute
- **from** - Kompletter oder verkürzter Zeitstempel als Nummer ('20231210150101', '20231210'). Alternativ Anzahl Tage, Stunden oder Minuten ('5d', '4h', '30m') in die Vergangenheit ausgehend von 'to', wenn dort ein Zeitstempel oder Asterisk angegeben ist
- **to** - Kompletter oder verkürzter Zeitstempel wie bei 'from', oder Asterisk (*) für die aktuelle Zeit. Alternativ Anzahl Tage, Stunden oder Minuten ('5d', '4h', '30m') in die Zukunft ausgehend von 'from', wenn dort ein Zeitstempel angegeben ist
- **devname** - Device-Name oder CSV-Namensliste

Der folgende Ausschnitt zeigt die Datenausgabe nach Auswahl von 'epstats_1_power'. Vorausgesetzt ist, dass der Command 'state' zuvor per Zeitplanung automatisiert ausgeführt wurde.



The screenshot shows the myminapp interface. At the top, there's a header bar with a user icon, the date and time '9.12.2023, 11:46:46', and a help icon. Below the header is a green bar with the 'myminapp' logo. The main content area features a dropdown menu set to 'epstats_1_power' and a text input field containing the command: `{"cmd": "epstats", "type": "power", "entity": "state", "units": 5, "from": "4h", "to": "**", "devname": "EM"}`. Below this is a table with 11 columns: Jahr, Monat, Tag, Stunde, 5 Minuten, Name, Kategorie, Anzahl, Min., Max., and Leistung W. The table displays 20 rows of data for the year 2023, month 12, and day 9, showing power consumption in 5-minute intervals.

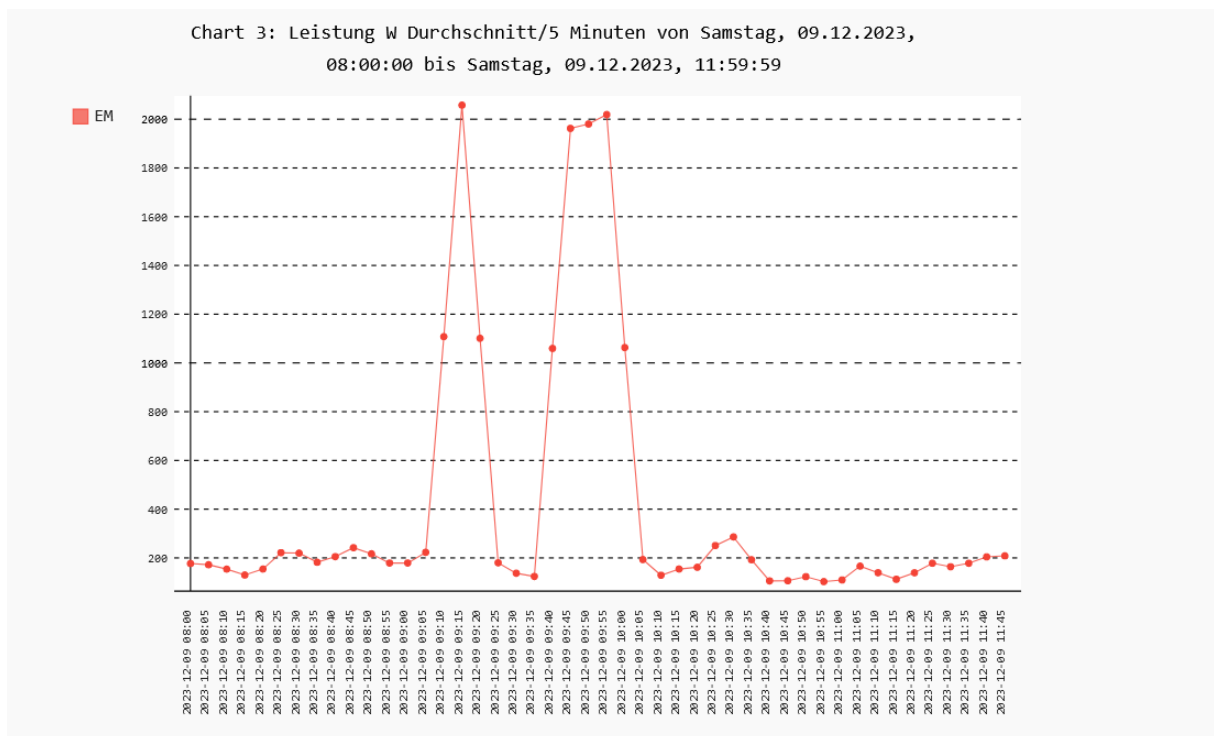
Jahr	Monat	Tag	Stunde	5 Minuten	Name	Kategorie	Anzahl	Min.	Max.	Leistung W
2023	12	9	8	0	EM	M	2	175.9	178.4	177.2
2023	12	9	8	5	EM	M	2	167.6	176.8	172.2
2023	12	9	8	10	EM	M	2	139.8	168.2	154.0
2023	12	9	8	15	EM	M	2	120.1	139.8	130.0
2023	12	9	8	20	EM	M	2	119.8	188.7	154.3
2023	12	9	8	25	EM	M	2	189.1	253.2	221.2
2023	12	9	8	30	EM	M	2	190.7	248.9	219.8
2023	12	9	8	35	EM	M	2	174.0	190.8	182.4
2023	12	9	8	40	EM	M	2	168.2	241.7	205.0
2023	12	9	8	45	EM	M	2	241.8	242.1	242.0
2023	12	9	8	50	EM	M	2	192.5	241.7	217.1
2023	12	9	8	55	EM	M	2	164.6	192.8	178.7
2023	12	9	9	0	EM	M	2	165.4	192.3	178.9
2023	12	9	9	5	EM	M	2	181.1	264.7	222.9
2023	12	9	9	10	EM	M	2	185.1	2030.9	1108.0

Dem Command-Preset entsprechend wurden statistische Daten ausgehend von der aktuellen Zeit, die ganz oben im Bild sichtbar ist, über 4 Stunden zurück in die Vergangenheit ermittelt ("from": "4h", "to": "*"). Hier handelt es sich um die Stunden 11, 10, 9 und 8 des 9. Dezember 2023.

Die Gruppierung der Datenausgabe erfolgte nach den 5 Zeiteinheiten Jahr, Monat, Tag, Stunde und Minute ("units": 5).

Die Daten wurden zuvor kontinuierlich in 5-Minuten-Abschnitten ermittelt, was auch hier in der Datenausgabe Berücksichtigung findet. Der Wert 2 in der Spalte 'Anzahl' bestätigt, dass pro 5 Minuten jeweils 2 Datensätze geschrieben wurden, ein Datensatz zu Beginn und einer zum Ende des jeweiligen Zeitabschnitts.

Eine Besonderheit stellt die Möglichkeit dar, SVG-Chart-Dateien per Button anzuzeigen und damit zu interagieren. Der Command 'epstats' liefert die Daten dazu, und der Applikationsserver bereitet die Datenausgabe entsprechend auf. Die Buttons für Balken-, Kuchen- und Linien-Chart sind in der Kopfzeile bei der Spalte 'Leistung W' sichtbar. Die folgende Abbildung zeigt den Linien-Chart zur vorausgehend dargestellten Datenausgabe in einem weiteren Browser-Tab.



Eine Änderung von "units": 5 zu "units": 4 in der Eingabezeile bewirkt die Gruppierung nach Stunden:

9.12.2023, 11:49:40

myminapp

epstats_1_power

{ "cmd": "epstats", "type": "power", "entity": "state", "units": 4, "from": "4h", "to": "*", "devname": "EM" }

Jahr	Monat	Tag	Stunde	Name	Kategorie	Anzahl	Min.	Max.	Leistung W
2023	12	9	8	EM	M	24	119.8	253.2	187.8
2023	12	9	9	EM	M	24	100.7	2095.6	1010.9
2023	12	9	10	EM	M	24	93.7	1962.0	238.9
2023	12	9	11	EM	M	19	98.3	208.7	157.3

Das nächste Beispiel zeigt statistische Daten, die mit Preset 'epstats_2_power' ermittelt wurden. Auch in diesem Beispiel wurden die Daten für die letzten 4 Stunden ermittelt, und die Gruppierung erfolgt nach Stunden.

Im Unterschied zum vorausgehenden Beispiel, bei dem nur die eine Device 'EM' berücksichtigt wurde, sind hier die Devices 'DESKTOP', 'FRIDGE', 'SOLAR', 'TV' und 'WASHER' berücksichtigt. Auffällig sind die Leistungsdaten der Device 'WASHER' vor allem in Stunde 9. Dies korreliert mit dem über Device 'EM' gemessenen Gesamtverbrauch im vorausgehenden Beispiel. Ergänzend werden Balken-, Kuchen- und Linien-Chart für den betreffenden Zeitraum dargestellt.

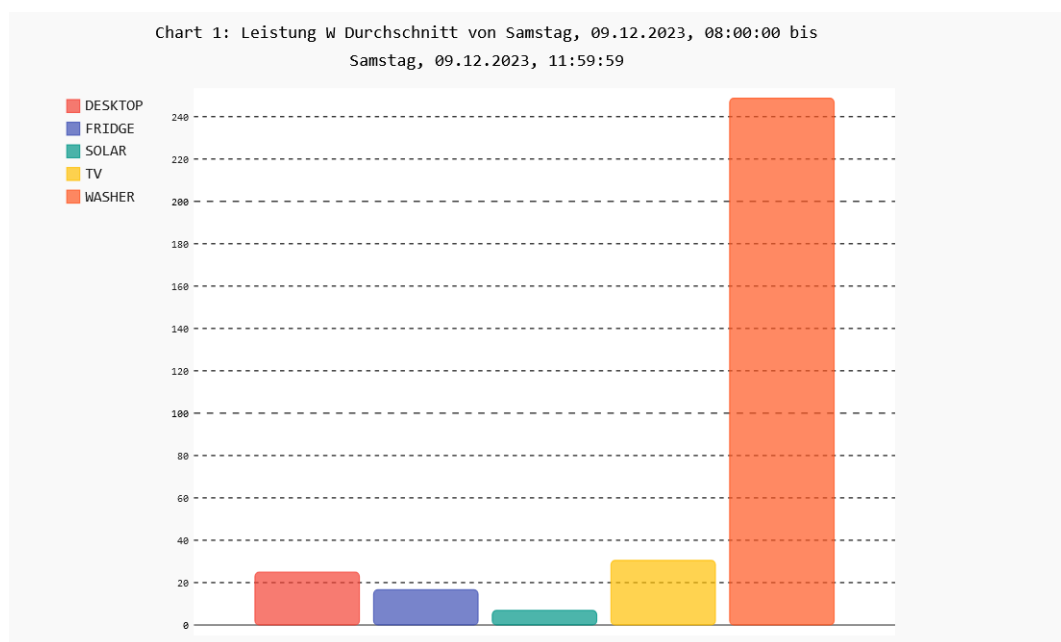
 9.12.2023, 11:56:15 



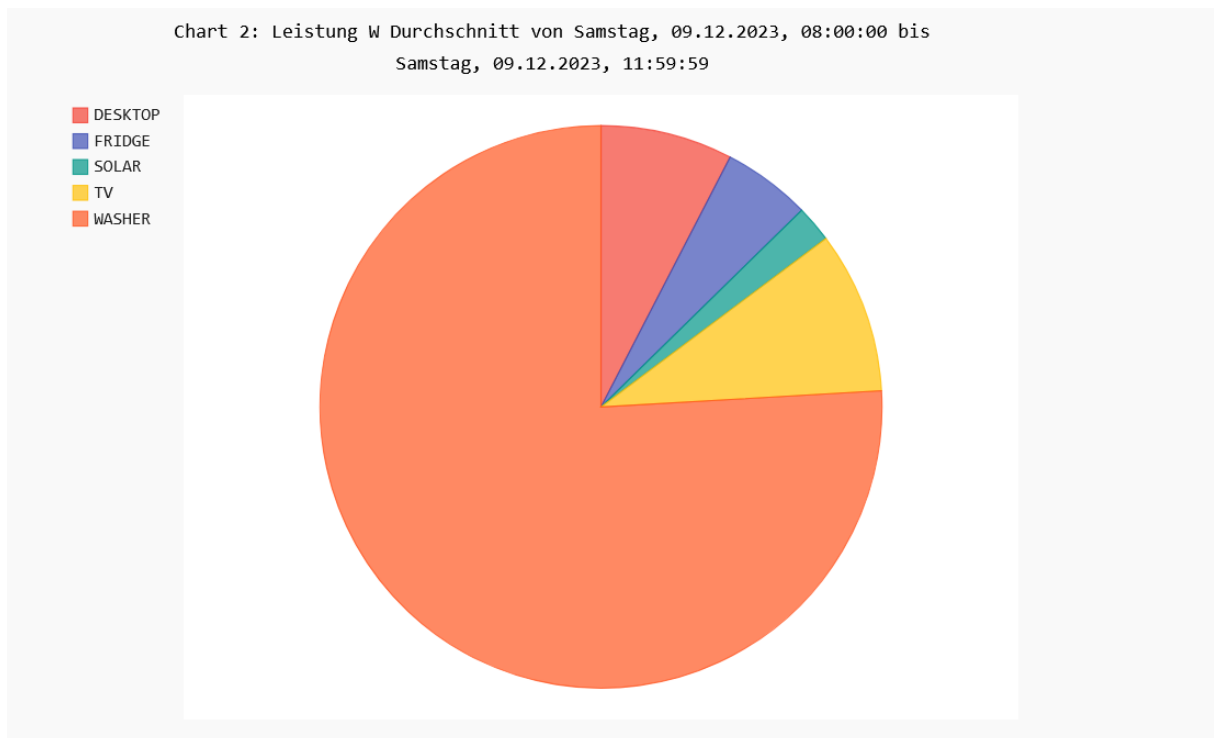
epstats_2_power   { "cmd": "epstats", "type": "power", "entity": "state", "units": 4, "from": "4h", "to": "*", "devname": "SOLAR, DESKTO" } 

Jahr	Monat	Tag	Stunde	Name	Kategorie	Anzahl	Min.	Max.	Leistung W
2023	12	9	8	DESKTOP	P	24	2.1	2.2	2.2
2023	12	9	8	FRIDGE	P	24	2.9	57.4	10.1
2023	12	9	8	SOLAR	P	24	0.1	0.2	0.2
2023	12	9	8	TV	P	24	2.4	71.2	33.6
2023	12	9	8	WASHER	P	24	0.2	0.2	0.2
2023	12	9	9	DESKTOP	P	24	2.1	39.0	32.1
2023	12	9	9	FRIDGE	P	24	2.8	63.1	20.4
2023	12	9	9	SOLAR	P	24	0.1	3.6	1.0
2023	12	9	9	TV	P	24	2.4	70.5	39.0
2023	12	9	9	WASHER	P	24	0.2	1912.0	868.4
2023	12	9	10	DESKTOP	P	24	30.6	50.6	34.8
2023	12	9	10	FRIDGE	P	24	2.9	62.8	14.4
2023	12	9	10	SOLAR	P	24	3.6	11.5	7.6
2023	12	9	10	TV	P	24	2.4	23.7	6.0
2023	12	9	10	WASHER	P	24	0.2	1833.8	115.8

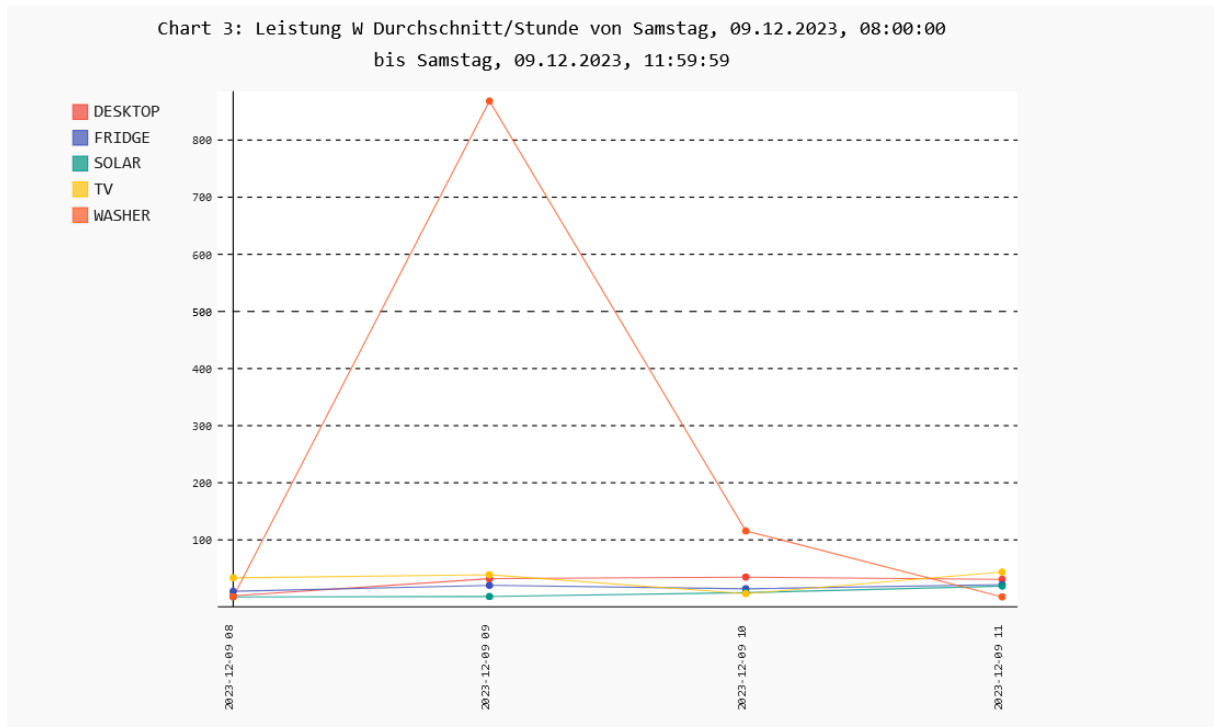
Balken-Chart:



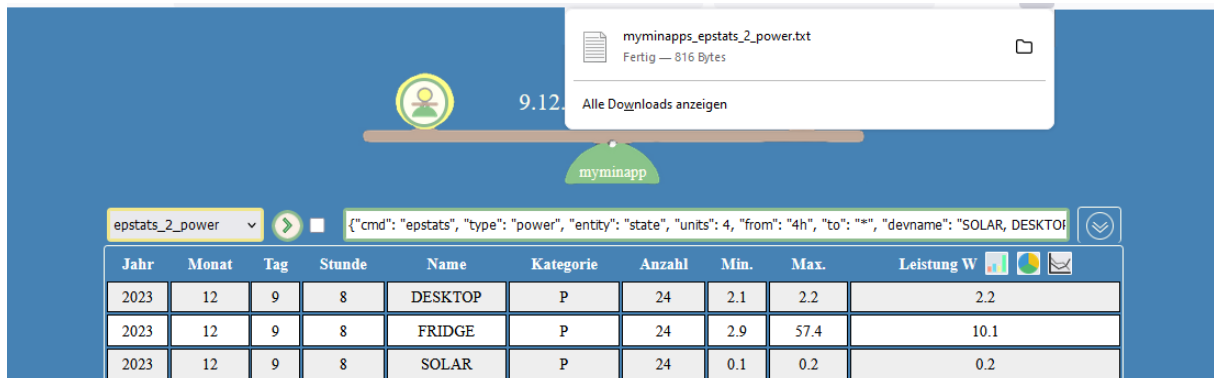
Kuchen-Chart:



Linien-Chart:



Um eine Datenausgabe herunterzuladen, ist der im folgenden Bild rechts direkt über der Tabelle sichtbare Button zu betätigen. Der Download erfolgt als CSV-Datei mit Trennzeichen Semikolon an das Ziel, das durch die Browser-Einstellungen für Downloads festgelegt wurde.



myminapps_epstats_2_power.txt
Fertig — 816 Bytes

Alle Downloads anzeigen

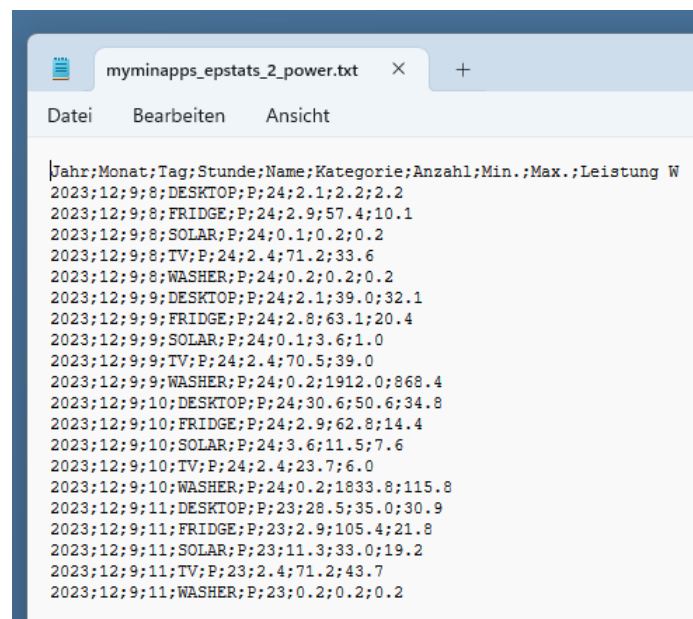
9.12.2023, 12:05:09

myminapp

epstats_2_power

{ "cmd": "epstats", "type": "power", "entity": "state", "units": 4, "from": "4h", "to": "*", "devname": "SOLAR, DESKTOP, FRIDGE, TV, WASHER" }

Jahr	Monat	Tag	Stunde	Name	Kategorie	Anzahl	Min.	Max.	Leistung W
2023	12	9	8	DESKTOP	P	24	2.1	2.2	2.2
2023	12	9	8	FRIDGE	P	24	2.9	57.4	10.1
2023	12	9	8	SOLAR	P	24	0.1	0.2	0.2



myminapps_epstats_2_power.txt

Datei Bearbeiten Ansicht

```
Jahr;Monat;Tag;Stunde;Name;Kategorie;Anzahl;Min.;Max.;Leistung W
2023;12;9;8;DESKTOP;P;24;2.1;2.2;2.2
2023;12;9;8;FRIDGE;P;24;2.9;57.4;10.1
2023;12;9;8;SOLAR;P;24;0.1;0.2;0.2
2023;12;9;8;TV;P;24;2.4;71.2;33.6
2023;12;9;8;WASHER;P;24;0.2;0.2;0.2
2023;12;9;9;DESKTOP;P;24;2.1;39.0;32.1
2023;12;9;9;FRIDGE;P;24;2.8;63.1;20.4
2023;12;9;9;SOLAR;P;24;0.1;3.6;1.0
2023;12;9;9;TV;P;24;2.4;70.5;39.0
2023;12;9;9;WASHER;P;24;0.2;1912.0;868.4
2023;12;9;10;DESKTOP;P;24;30.6;50.6;34.8
2023;12;9;10;FRIDGE;P;24;2.9;62.8;14.4
2023;12;9;10;SOLAR;P;24;3.6;11.5;7.6
2023;12;9;10;TV;P;24;2.4;23.7;6.0
2023;12;9;10;WASHER;P;24;0.2;1833.8;115.8
2023;12;9;11;DESKTOP;P;23;28.5;35.0;30.9
2023;12;9;11;FRIDGE;P;23;2.9;105.4;21.8
2023;12;9;11;SOLAR;P;23;11.3;33.0;19.2
2023;12;9;11;TV;P;23;2.4;71.2;43.7
2023;12;9;11;WASHER;P;23;0.2;0.2;0.2
```

Im Folgenden werden noch einige weitere Beispiele ohne Kommentar dargestellt.

Preset 'state_2': Aktueller Status diverser Devices

"state_2": { "cmd": "state", "devname": "SOLAR, DESKTOP, FRIDGE, TV, WASHER" },



9.12.2023, 12:05:09

myminapp

state_2

{ "cmd": "state", "devname": "SOLAR, DESKTOP, FRIDGE, TV, WASHER" }

Name	Kategorie	Zustand	Leistung W	Energie kWh	°C	Info
SOLAR	P	ON	35.4	687.783	5.0	OK
DESKTOP	P	ON	28.2	68.756	20.5	OK
FRIDGE	P	ON	3.0	84.058	24.0	OK
TV	P	ON	2.4	42.824	18.5	OK
WASHER	P	ON	0.2	9.333	22.0	OK

Preset 'psdischarge_1': Konditionelles Entladen einer Powerstation

```
"psdischarge_1": {"cmd": "psdischarge", "value": "minlevel=42"},
```

Wert	Datei	Objekt	Info
OFF (level=6)	app1-numcam757.bmp		OK

Preset 'pscharge_1': Konditionelles Laden einer Powerstation

```
"pscharge_1": {"cmd": "pscharge", "value": "cmax=50, smin=100"},
```

Wert	Info
cmax=50,ccur=-87,smin=100,scur=38,p1=OFF,p2=OFF,p3=OFF	OK

Preset 'plug_OFF': Ausschalten einer Schaltsteckdose

```
"plug_OFF": {"cmd": "setting", "devname": "PLUG", "value": "OFF"},
```

Name	Kategorie	Wert	Info
PLUG	P	OFF	OK

Freie Selektion auf Basis von Preset 'select_1'

```
"select_1": {"cmd": "selection", "value": "SELECT * FROM state WHERE id = 1;"},
```

An dem ausgewählten Preset wurden in der Befehlszeile Anpassungen vorgenommen:

```
{\"cmd\": \"selection\", \"value\": \"SELECT * FROM state WHERE t1 = 2023 AND t2 = 12 AND t3 = 9 AND t4 = 11;\"}
```

So können über alle in der Datenbank der Applikationsinstanz befindlichen Datenbanktabellen/Entitäten im Prinzip beliebige Selektionen vorbereitet, bei Bedarf angepasst und ausgeführt werden.



9.12.2023, 12:12:32



select_1

{ "cmd": "selection", "value": "SELECT * FROM state WHERE t1 = 2023 AND t2 = 12 AND t3 = 9 AND t4 = 11;" }

ID	Zeitstempel	Jahr	Monat	Tag	Stunde	Minute	Sekunde	Name	Kategorie	Zustand	Leistung W	Energie kWh	°C	Info
2667	20231209110000	2023	12	9	11	0	0	SOLAR	P	ON	11.3	687.761	4.0	OK
2668	20231209110000	2023	12	9	11	0	0	DESKTOP	P	ON	35.0	68.722	20.5	OK
2669	20231209110000	2023	12	9	11	0	0	FRIDGE	P	ON	2.9	84.039	23.5	OK
2670	20231209110000	2023	12	9	11	0	0	TV	P	ON	2.8	42.783	18.5	OK
2671	20231209110000	2023	12	9	11	0	0	WASHER	P	ON	0.2	9.333	23.0	OK
2672	20231209110002	2023	12	9	11	0	2	EM	M	ON	-98.3	1110.517	None	OK
2673	20231209110002	2023	12	9	11	0	2	PS_IN1	P	OFF	0.0	84.69	None	OK
2674	20231209110002	2023	12	9	11	0	2	PS_IN2	P	OFF	0.0	49.94	None	OK
2675	20231209110002	2023	12	9	11	0	2	PS_IN3	P	OFF	0.0	41.59	None	OK
2676	20231209110002	2023	12	9	11	0	2	PS_OUT	P	OFF	0.0	74.91	None	OK
2677	20231209110002	2023	12	9	11	0	2	PLUG	P	OFF	0.0	2.62	None	OK
2678	20231209110452	2023	12	9	11	4	52	SOLAR	P	ON	12.8	687.762	4.0	OK
2679	20231209110452	2023	12	9	11	4	52	DESKTOP	P	ON	32.4	68.725	20.5	OK
2680	20231209110452	2023	12	9	11	4	52	FRIDGE	P	ON	2.9	84.039	23.5	OK
2681	20231209110452	2023	12	9	11	4	52	TV	P	ON	2.7	42.783	18.5	OK

2.4.3 Sonstiges

Das Web-Frontend kann auch auf Smartphones per Browser angewendet werden, bevorzugt im horizontalen Format. Durch die komfortable Auswahl von Command-Presets ist die Bedienung problemlos möglich. Eine spezielle Smartphone-App ist daher nicht vorgesehen.

Darüber hinaus besteht die Möglichkeit, Anfragen als GET-Request zu formulieren, auszuführen und dann als Favoriten im Browser zu speichern. Die Ausgabe erfolgt im JSON-Format, wie das folgende Beispiel zeigt.

192.168.178.26:8081/%7B%22cmd%3A%22%3A%22helloworld%22%2C%22value%22%3A%22Hello%20World%22%7D%7B%22code%22%3A%220%22%2C%22text%22%3A%22OK%22%2C%22trace%22%3A%22%22%2C%22data%22%3A%7B%22value%22%3A%22Hello%20World%20from%20app1%2C%20helloworld%22%2C%22info%22%3A%22OK%22%7D%7D

← → ↻ 🔒 192.168.178.26:8081/{ "cmd": "helloworld", "value": "Hello World" }

JSON Rohdaten Kopfzeilen

Speichern Kopieren Alle einklappen Alle ausklappen 🔍 JSON durchsuchen

```

0:
  code: 0
  text: "OK"
  trace: ""
  data:
    value: "Hello World from app1, helloworld"
    info: "OK"

```

3. Commands

Im Folgenden wird jeder Command unter seinem Modulnamen kurz beschrieben. Details sind in der jeweiligen Moduldatei `/lib/command/<Modulname>.py` und in der Applikationsdefinition zu finden.

Zur Erstellung eigener Command-Klassen siehe Kapitel 'Eigene Klassen erstellen'.

Command	Erläuterung
cmd	Superklasse für alle mitgelieferten und eigenen Commands.
epstats	Erstellung statistischer Daten zu Energie (e nergy) und Leistung (p ower) auf Basis von Daten, die beispielsweise von Command 'state' automatisiert ermittelt wurden. Verwendet die Device-Klasse 'Chart' um Balken-, Kuchen- und Liniendiagramme zu erzeugen.
helloworld	'Hello World'-Implementierung. Der Quellcode kann als einfaches Muster für eigene Command-Klassen dienen.
monitorview	Darstellung der aktuellen Monitoring-Daten einer Applikationsinstanz. Verwendet das Utility 'Monitor'.
pscharge	Automatisches Laden einer Powerstation über ausgediente Laptop-Netzteile. Verwendet die Device-Klassen 'DTSU' und 'Fritz' um aktuelle Leistungsdaten zu erhalten, sowie die Device-Klasse 'Bosch' zum Schalten der Netzteile über Schaltsteckdosen.
psdischarge	Automatisches Entladen einer Powerstation zur nächtlichen Grundlastabdeckung. Verwendet die Device-Klasse 'Numcam757' um den Füllstand einer Powerstation vom Display zu erfassen, sowie die Device-Klasse 'Relay5V' zum Schalten eines kleinen USB-Verbrauchers an der Powerstation, um die Display-Beleuchtung zu aktivieren und den Stromfluss von der Powerstation zum Wechselrichter zu schalten.
selection	Freie Datenselektion aus beliebigen Entitäten des Datenspeichers.
setting	Produktübergreifendes Schalten von Schaltsteckdosen und Relais sowie Setzen von Wechselrichter-Limits. Verwendet die Device-Klassen 'Fritz', 'Bosch', 'Nohub', 'Relay5V' und 'OpenDTU'.
state	Produktübergreifendes Ermitteln von Status-Werten von Schaltsteckdosen, Relais, Energiemetern und Wechselrichtern. Verwendet die Device-Klassen 'Fritz', 'Bosch', 'Nohub', 'Relay5V', 'DTSU' und 'OpenDTU'. Kann automatisiert angewendet werden, um die Basis für statistische Daten zu schaffen (siehe Command 'epstats').
textdoc	Erzeugt eine einfache Python-Docstring-Dokumentation in Form von Textdateien, die unter <code>/data/temp/textdoc/myminapp</code> gespeichert werden.

Die folgenden Commands verwenden keine Device, haben daher keine externen Abhängigkeiten und sind direkt nach der Installation anwendbar:

- helloworld
- monitorview
- textdoc

Der folgende Command ist anwendbar, sobald eine Datenbanktabelle/Entität im Datenspeicher vorhanden ist:

- selection

4. Devices

Hier wird jede Device unter ihrem Modulnamen kurz beschrieben. Details sind in der jeweiligen Moduldatei `'/lib/device/<Unterverzeichnis>/<Modulname>.py'` und in der Applikationsdefinition zu finden. Sofern externe Abhängigkeiten bestehen, wird jeweils darauf hingewiesen.

Zur Erstellung eigener Device-Klassen siehe Kapitel 'Eigene Klassen erstellen'.

Device	Erläuterung
camera.numcam757	<p>Ermittelt den Füllstand einer Powerstation 'Anker 757' mittels einer einfachen kleinen HD-Kamera, die vor dem Display der Powerstation positioniert ist. Gibt bei erfolgreicher Bilderfassung den Füllstand als Zahl zurück (0 bis 100), sonst -1. Das Bild kann optional als Bitmap-Datei gespeichert werden, beispielsweise wie bei Command 'psdischarge' unter <code>'/data/temp/camera/app<n>-numcam757.bmp'</code>.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none">• Externes Paket 'pygame' (pip install pygame)• Einfache Noname-HD-Kamera• Powerstation 'Anker 757'
graphics.chart	<p>Erstellt Balken-, Kuchen- und Liniendiagramme als SVG-Dateien (Scalable Vector Graphics), die unter <code>'/data/temp/chart/'</code> gespeichert werden.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none">• Externes Paket 'pygal' (pip install pygal)
inverter.opendtu	<p>Kommuniziert via HTTP mit OpenDTU Hardware, um Statusdaten eines 'Hoymiles' Wechselrichters zu erhalten oder ein Wechselrichter-Limit zu setzen.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none">• Externes Paket 'requests' (pip install requests)• Hardware wie unter <code>'https://github.com/tbnobody/OpenDTU'</code> beschrieben• Wechselrichter 'Hoymiles', beispielsweise 'HM 300' oder 'HM 600'
meter.dtsu	<p>Kommuniziert via Modbus RTU mit einem Energiemeter, das an zentralem Ort in das Stromnetz des Hauses integriert ist, typischerweise im Verteilerschrank. Gelesen werden die aktuelle Gesamtleistung in Watt (plus und minus) sowie der Gesamtverbrauch in Kilowattstunden.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none">• Externes Modul 'minimalmodbus' (pip install minimalmodbus)• Energiemeter 'Chint DTSU666'
plug.bosch	<p>Kommuniziert via HTTPS mit einem Hub vom Typ 'Bosch Smart Home Controller II', um Statusdaten von Schaltsteckdosen zu lesen und diese zu schalten.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none">• Externes Paket 'requests' (pip install requests)• 'Bosch Smart Home Controller II'• Mindestens eine Schaltsteckdose 'Bosch Smart Plug Compact'



plug.fritz	<p>Kommuniziert via HTTP respektive TR-064 mit einem Hub vom Typ 'FRITZ!BOX', um Statusdaten von Schaltsteckdosen zu lesen und diese zu schalten.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none"> • Externes Paket 'fritzconnection' ('pip install fritzconnection') • 'FRITZ!BOX', die 'FRITZ!DECT' Schaltsteckdosen unterstützt • Mindestens eine Schaltsteckdose vom Typ 'FRITZ!DECT' ('200' oder '210')
plug.nohub	<p>Kommuniziert via HTTP ohne Hub direkt mit 'Shelly' Schaltsteckdosen, um Statusdaten von ihnen zu lesen und sie zu schalten.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none"> • Externes Paket 'requests' (pip install requests) • Mindestens eine Schaltsteckdose 'Shelly Plus Plug S'
relay.relay5v	<p>Kommuniziert via GPIO (General Purpose Input Output) mit einem Relais, um Schaltzustände zu lesen und Schaltvorgänge auszulösen.</p> <p>Abhängigkeiten:</p> <ul style="list-style-type: none"> • Externes Paket 'RPi.GPIO' • 'Raspberry Pi' SBC (Single Board Computer), beispielsweise 'Raspberry Pi 4B' • 5 Volt Relais, beispielsweise 'Elegoo DC 5V'

5. Utilities

Im Folgenden wird jedes mitgelieferte Utility unter seinem Klassennamen kurz beschrieben. Die jeweilige Moduldatei befindet sich unter `/lib/util/<Modulname>.py`.

5.1 Helper

Die Klasse 'Helper' bietet diverse Methoden, die aus allen Applikationsschichten genutzt werden können. Insbesondere sind folgende Features zu nennen:

- Bezug von Nachrichtentexten über Methode 'mtext'
- Bezug des Nachrichten-Levels über Methode 'mlevel'
- Bezug einer Exception-Trace-Back-Zeile über Methode 'tbline'
- Konvertierung von und nach JSON über die Methoden 'dict2json' und 'json2dict'
- Diverse Zeit- und Kalender-bezogene Methoden
- Weitere Methoden

5.2 Logger

Die Klasse 'Logger' dient der Nachrichtenausgabe und verwendet ihrerseits den Python-Standard-Logger (Python-Modul 'logging'). Unterstützt werden die Log-Level 'ERROR', 'WARNING', 'INFO' und 'DEBUG'. Die Spezifikation erfolgt in der Applikationsdefinition. Die Klasse wird ausschließlich von den Klassen 'App' und 'TestLogger' angewendet.

Das Logging erfolgt unter dem Namen der Applikationsinstanz. Wird die Logausgabe in Dateien spezifiziert, so erfolgt die Ausgabe nach `'/data/log/'` (`app1.log`, `app2.log`, ...).

Die Moduldatei `'/lib/text/message.py'` enthält Nachrichtentexte mit einem numerischen Code in den Sprachausprägungen Englisch und Deutsch.

Unterstützt werden die Nachrichten-Level 'ERROR', 'WARNING', 'INFO' und 'DEBUG'. Für die Level 'ERROR', 'WARNING' und 'DEBUG' ist jeweils eine Liste für Nachrichten-Codes deklariert, so dass eine bestimmte Nachricht einfach durch hinzufügen ihres Codes in eine der Listen klassifiziert werden kann. Für Nachrichten mit Codes, die in keiner der drei Listen vorkommen, gilt Level INFO.

Beim Logging werden die Nachrichten-Level über den Nachrichten-Code automatisch zugeordnet.

5.3 Monitor

Über die Klasse 'Monitor' werden Statusdaten der jeweiligen Applikationsinstanz in eine SQLite-Datenbank geschrieben, die sich im Arbeitsspeicher befindet und den Zugriff aus multiplen Threads unterstützt. Das Monitoring ist insbesondere nützlich, wenn Zeitplanungssätze verarbeitet werden.

Siehe dazu auch die Informationen zum Command 'monitorview' im Kapitel 'Commands' weiter oben sowie das folgende Kapitel 'Scheduler'.

5.4 Scheduler

Über die Klasse 'Scheduler' werden die in der Applikationsdefinition spezifizierten Zeitplanungssätze verarbeitet. Die Klasse wird ausschließlich von den Klassen 'App' und 'TestScheduler' angewendet.

Beim Start einer Applikationsinstanz wird für jeden zugeordneten Zeitplanungssatz eine Instanz der Klasse 'Scheduler' sowie ein nebenläufiger Thread erzeugt. Daraus wird alle 3 Sekunden die Methode 'get_action_number' aufgerufen; ist die Aktionsnummer größer 0, wird die geplante Aktion ausgeführt.

Im Folgenden werden die Parameter für die Spezifikation von Zeitplanungssätzen beschrieben.

Parameter	Erläuterung
cmdpreset	Name eines in 'COMMAND_PRESET' spezifizierten Commands, der vom Scheduler aufgerufen wird.
days	Tage, an dem die Ausführung erfolgen soll. Unterstützt werden folgende Angaben: <ul style="list-style-type: none">'*' (Asterisk) - alle Tage'0, 1, ..., 7' - CSV-Liste von Tagen (0 oder 7 für Sonntag, 1 für Montag etc.)
time	Zeitpunkt oder Zeitrahmen für die Ausführung. Unterstützt werden folgende Formate: <ul style="list-style-type: none">Zeitpunkt zwecks einmaligem Aufruf: 'HH:MM:SS', beispielsweise '10:00:00' für 10 Uhr. Ist nicht zulässig in Kombination mit Parameter 'intervaldiff' und 'intervalunit'Zeitrahmen zwecks mehrfachem Aufruf: 'HH:MM:SS-HH:MM:SS', beispielsweise '10:00:00-15:00:00' für 10 bis 15 Uhr. Ist die zweite Uhrzeit kleiner als die erste, gilt der Folgetag. Erfordert die Angabe des Parameters 'intervaldiff' oder 'intervalunit' <p>Wird der Parameter 'time' nicht angegeben, so wird als Zeitrahmen der gesamte Tag angenommen (00:00:00-23:59:59 Uhr), was die Angabe des Parameters 'intervaldiff' oder 'intervalunit' erforderlich macht.</p>
intervaldiff	Ausführungsintervall in Sekunden, beispielsweise 60 für 60 Sekunden. Zulässig sind Angaben von 5 bis 43200 (5 Sekunden bis 12 Stunden). Die erste Ausführung beginnt unmittelbar nach Start der Applikationsinstanz. In Kombination mit einer Zeitpunkt-Angabe beim Parameter 'time' ist dieser Parameter unzulässig.
intervalunit	Ausführungsintervall nach Zeiteinheit. Dieser Parameter ist insbesondere geeignet, um Statusdaten in regelmäßigen Abständen zu ermitteln (beispielsweise mit Command 'state'), um sie später statistisch auszuwerten (beispielsweise mit Command 'epstats'). Unterstützt werden folgende Angaben: <ul style="list-style-type: none">'m' - Ausführung zu Beginn und zum Ende eines 5-Minuten-Abschnitts'h' - Ausführung stündlich zu Beginn und zum Ende einer vollen Stunde'd' - Ausführung täglich zu Beginn und zum Ende eines vollen Tages <p>Die erste Ausführung erfolgt nach Start der Applikationsinstanz zu Beginn der jeweiligen Zeiteinheit. Wurde die Instanz beispielsweise um 10:12:00 gestartet, würde die erste Ausführung bei 'm' gegen 10:15:00 Uhr, bei 'h' gegen 11:00:00 Uhr und bei 'd' gegen 00:00:00 Uhr (des Folgetages) erfolgen.</p> <p>In Kombination mit einer Zeitpunkt-Angabe beim Parameter 'time' ist dieser Parameter unzulässig.</p>



appnum	Instanznummer. Hiermit wird der Zeitplanungseintrag einer Applikationsinstanz zugeordnet, beispielsweise 1 für Applikationsinstanz 1 ('app1').
--------	--

Beispiele für Zeitplanungseinträge in 'COMMAND_SCHEDSETS':

```
"schedule_1": {"cmdpreset": "state_1", "days": "*", "intervalunit": 'm', "appnum": 1},
"schedule_2": {"cmdpreset": "state_2", "days": "*", "intervalunit": 'h', "appnum": 1},
"schedule_3": {"cmdpreset": "state_3", "days": "*", "intervalunit": 'd', "appnum": 1},

"schedule_4": {"cmdpreset": "state_4", "days": "1,3,5", "time": "18:00:00-03:00:00",
               "intervaldiff": 30, "appnum": 1},

"schedule_5": {"cmdpreset": "helloworld", "days": "7", "time": "15:01:01", "appnum": 1}
```

Hinweis: Die kontinuierliche, zeitbezogene Ermittlung von Daten zur späteren statistischen Auswertung mag auf den ersten Blick trivial erscheinen. Bei genauerer Betrachtung ergeben sich jedoch diverse Aspekte, die zu berücksichtigen sind. Daher wird das Thema im Anhang unter 'Daten für statistische Zwecke aufzeichnen' gesondert behandelt.

5.5 Storage

Der Datenspeicher ist durch die in der Python-Standardbibliothek enthaltene Version von SQLite implementiert.

Die Klasse 'Storage' wird primär von der Klasse 'App' (sowie der Klasse 'TestStorage') angewendet. Pro Applikationsinstanz wird beim Erstzugriff automatisch eine Datenbank unter '/data/storage/' erzeugt (app1.db, app2.db etc.).

Um valide Command-Ausgaben automatisch zu speichern, ist bei der Spezifikation eines Commands in 'COMMAND_DEFSETS' beim Parameter 'storage' der Wert True anzugeben. Es wird dann automatisch eine Datenbanktabelle (Entität) mit dem Namen des Commands und den beim Parameter 'output' spezifizierten Datenfeldern erzeugt.

Jede Entität erhält außerdem den Satz von Präfix-Feldern, die in 'PREFIX_FIELD_DEFSETS' spezifiziert sind. Diese Felder umfassen den Primärschlüssel ('id') sowie Felder zur einfachen zeitlichen Zuordnung jedes Eintrags einer Entität ('t0', 't1', ..., 't6').

Die Klasse 'Storage' kann auch von Command- oder Device-Klassen verwendet werden, so wie es beispielsweise bei den Commands 'epstats' und 'selection' der Fall ist.

6. Eigene Klassen erstellen

Die myminapp-Applikation kann mit einer IDE wie Visual Studio Code leicht durch eigene Command- und Device-Klassen erweitert werden. Dazu im Folgenden einige Hinweise.

Zu Command-Klassen:

Alle Command-Klassen sind von der Superklasse 'Cmd' im Modul 'cmd' abzuleiten. Sie verfügt über folgende Methoden:

- **__init__(self, name:str)**: Wird von den abgeleiteten Klassen mit ihrem Klassennamen aufgerufen
- **exec(self, input:dict)**: Wird von den abgeleiteten Klassen überschrieben, um den jeweiligen Command auszuführen
- **close(self)**: Wird von den abgeleiteten Klassen überschrieben, um Ressourcen zu schließen
- **add_command_output(self, code:int, text:str, trace:str, dataset:dict, datasets:list)**: Dient der standardisierten Command-Ausgabe. Wird von den abgeleiteten Klassen einfach oder mehrfach aufgerufen, um das jeweilige Resultat in Form einer Liste (datasets) bereitzustellen

Als Beispiel zur Ableitung einer eigenen Command-Klasse kann der Quellcode der Klassen 'HelloWorld' in 'helloworld.py' und 'TestHelloWorld' in 'test_helloworld.py' dienen. Aus dem sehr einfachen Quellcode und den ergänzenden Kommentaren sollte alles Wesentliche hervorgehen.

Zu Device-Klassen:

Da Device-Klassen jeweils eine physische oder virtuelle Einheit im weitesten Sinne repräsentieren, kann der Code für die Implementierung sehr individuell ausgeprägt sein. Es wurde daher keine Superklasse vorgesehen.

Beim Erstellen ist jedoch generell zu beachten, dass Device-Klassen ausschließlich von Command-Klassen verwendet werden. Die entsprechende Struktur, die durch die bereits vorhandenen Klassen gegeben ist, soll nicht durchbrochen werden. Außerdem ist es zur Erhaltung der Transparenz wünschenswert, bezüglich Initialisieren und Schließen von Device-Klassen einheitlich zu verfahren. Daher sollten immer die folgenden Methoden implementiert werden:

- **__init__(self, ...)**: Impliziter Aufruf bei der Instanziierung, nach Bedarf mit oder ohne Parameter
- **close(self)**: Schließen von Ressourcen bei Bedarf, sonst formell

Als einfaches Musterbeispiel für eine Device-Klasse sei 'DTSU' in 'meter.dtsu.py' empfohlen.

Allgemein:

Auch im Blick auf die Quellcode-Dokumentation ('Docstrings') und die Trennung zwischen öffentlichen und privaten Methoden sollte dem Muster der mitgelieferten Klassen entsprechend einheitlich verfahren werden, um die Transparenz der myminapp-Applikation zu erhalten.

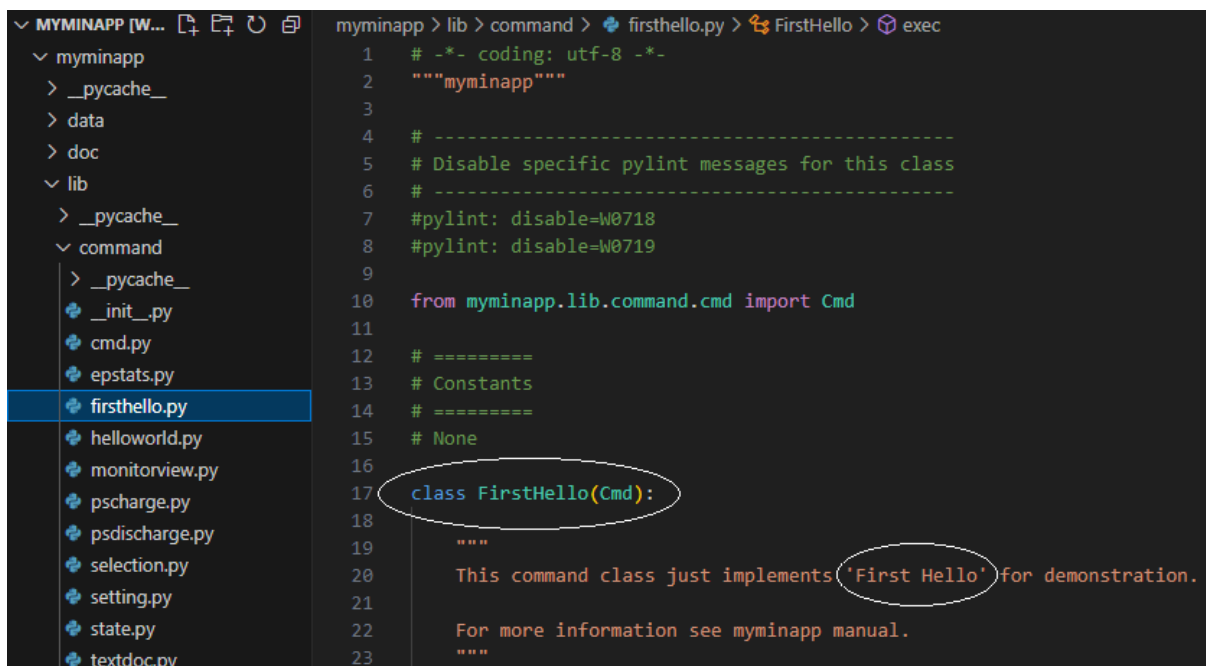
Schließlich sind die erforderlichen Spezifikationen zur neuen eigenen Command- oder Device-Klasse in der Applikationsdefinition zu ergänzen.

6.1 Erste Schritte

Im Folgenden wird Schritt für Schritt beschrieben, wie ein erster eigener Command erstellt werden kann, genannt 'firsthello'.

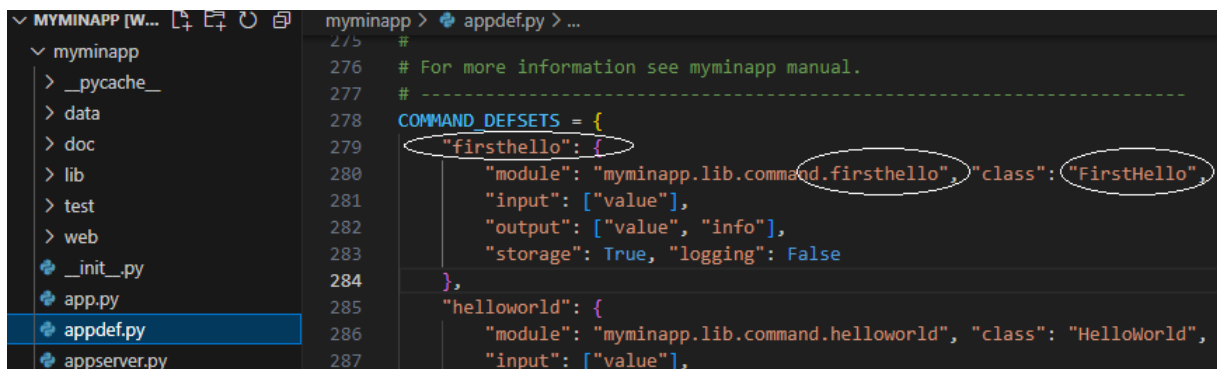
Schrittfolge 1:

- Die IDE der Wahl öffnen (beispielsweise Visual Studio Code) und das Verzeichnis 'myminapp' öffnen. In den Bereich 'lib/command' navigieren und die Datei 'helloworld.py' zu 'firsthello.py' kopieren.
- Im Quellcode der kopierten Datei den Klassennamen von 'HelloWorld' zu 'FirstHello' ändern (auf Groß-Kleinschreibung achten) und den Docstring-Text entsprechend anpassen.



```
1  # -*- coding: utf-8 -*-
2  """myminapp"""
3
4  # -----
5  # Disable specific pylint messages for this class
6  # -----
7  #pylint: disable=W0718
8  #pylint: disable=W0719
9
10 from myminapp.lib.command.cmd import Cmd
11
12 # =====
13 # Constants
14 # =====
15 # None
16
17 class FirstHello(Cmd):
18     """
19     This command class just implements 'First Hello' for demonstration.
20     For more information see myminapp manual.
21     """
```

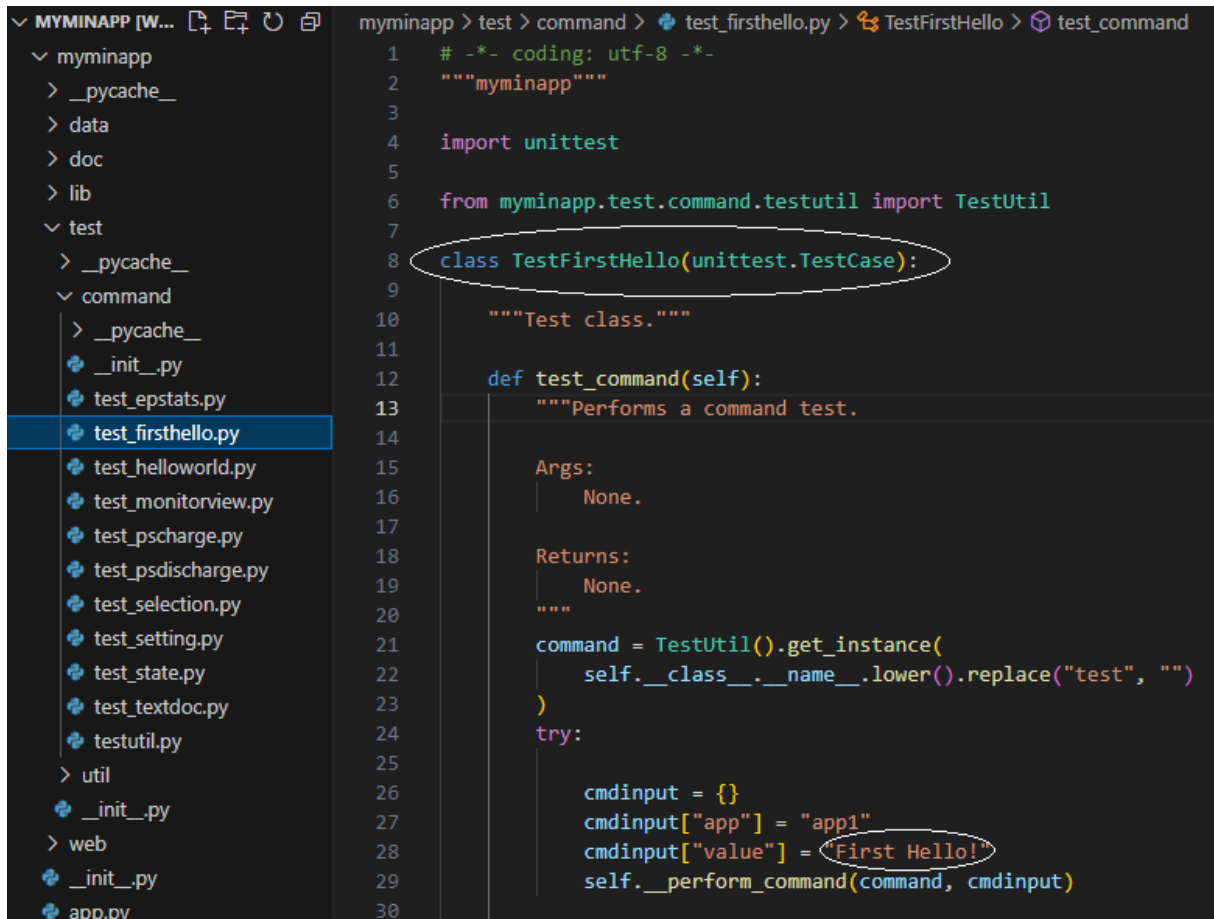
- Zur Datei 'appdef.py' navigieren, im Quellcode unter 'COMMAND_DEFSSETS' die Command-Spezifikation für 'helloworld' kopieren und direkt darüber einfügen.
- Im kopierten Bereich den Befehlsnamen 'helloworld' zu 'firsthello' ändern und ebenso den Modul- und den Klassennamen sinngemäß anpassen. Hier besonders auf Kommasetzung, Groß-Kleinschreibung, Einrückungen etc. achten.



```
275 #
276 # For more information see myminapp manual.
277 # -----
278 COMMAND_DEFSSETS = {
279     "helloworld": {
280         "module": "myminapp.lib.command.helloworld", "class": "HelloWorld",
281         "input": ["value"],
282         "output": ["value", "info"],
283         "storage": True, "logging": False
284     },
285     "firsthello": {
286         "module": "myminapp.lib.command.firsthello", "class": "FirstHello",
287         "input": ["value"],
```

Schrittfolge 2:

- In den Bereich 'test/command' navigieren und die Datei 'test_helloworld.py' zu 'test_firsthello.py' kopieren.
- Im Quellcode der kopierten Datei den Klassennamen von 'TestHelloWorld' zu 'TestFirstHello' ändern (auf Groß-Kleinschreibung achten) und den Eingabe-Parameter 'value' anpassen.



```
1  # -*- coding: utf-8 -*-
2  """myminapp"""
3
4  import unittest
5
6  from myminapp.test.command.testutil import TestUtil
7
8  class TestFirstHello(unittest.TestCase):
9
10     """Test class."""
11
12     def test_command(self):
13         """Performs a command test.
14
15         Args:
16             None.
17
18         Returns:
19             None.
20         """
21         command = TestUtil().get_instance(
22             self.__class__.__name__.lower().replace("test", "")
23         )
24         try:
25
26             cmdinput = {}
27             cmdinput["app"] = "app1"
28             cmdinput["value"] = "First Hello!"
29             self.__perform_command(command, cmdinput)
30
```

Wenn alle Änderungen in der IDE gespeichert sind, ein Terminal oberhalb des Verzeichnisses 'myminapp' öffnen und eingeben:

```
python3 -m myminapp.test.command.test_firsthello
```

Das Ergebnis sollte dann etwa wie folgt aussehen:

```
...
Result firsthello: [{'code': 0, 'text': 'OK', 'trace': '', 'data': {'value': 'First Hello!
from app1, firsthello', 'info': 'OK'}}]
.
-----
Ran 1 test in 0.006s

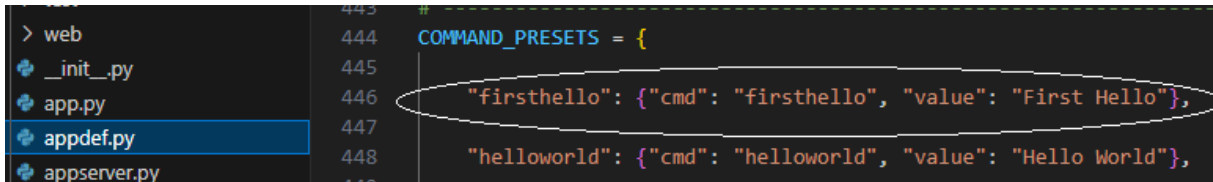
OK
...
```

Damit ist der erste eigene Command bereits erfolgreich implementiert und getestet!

Schrittfolge 3:

Der neue Command 'firsthello' kann nun auch auf den anderen Wegen aufgerufen werden, die im Kapitel 'Quickstart' für 'helloworld' beschrieben sind. Um ihn im Web-Frontend direkt auswählbar zu machen, ist noch ein passender Eintrag in 'appdef.py' unter 'COMMAND_PRESETS' zu ergänzen:

```
"firsthello": {"cmd": "firsthello", "value": "First Hello"},
```

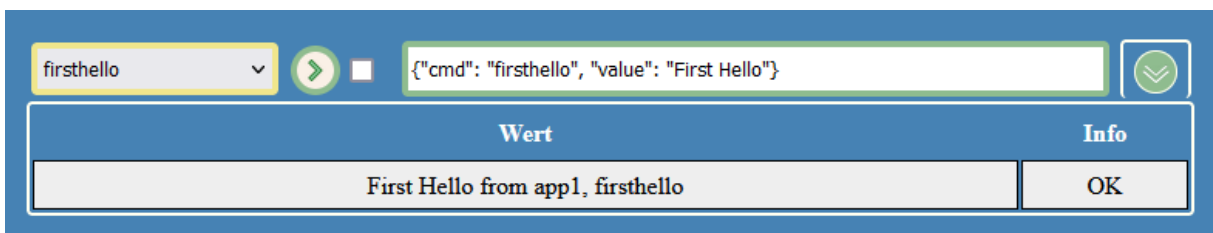


Jetzt den Applikationsserver starten mit `python3 -m myminapp.appserver`

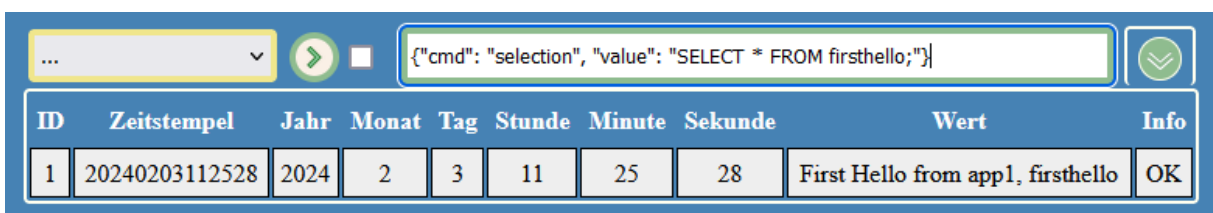
Hinweis: Die folgenden Bilder wurden mit der Spracheinstellung LANG = 'de' in 'appdef.py' erzeugt.

```
2024-02-03 11:18:20,294 | INFO | app1 | 900 | Logger 'app1' erstellt mit Handler(n): ['<StreamHandler <stdout> (INFO)>', '<Rotating
FileHandler /home/ui/dev/myminapp/myminapp/data/log/app1.log (INFO)>']
2024-02-03 11:18:20,296 | INFO | app1 | 100 | Applikationsinstanz 'app1' wurde mit Prozess-ID 22613 gestartet.
2024-02-03 11:18:20,299 | INFO | app1 | 150 | Appserver host: 127.0.0.1, port: 8081, pid: 22613, appnum: 1. Beenden mit Strg+C.
```

Dann im Browser mit `http://localhost:8081/app.html` das Web-Frontend aufrufen und den neuen Command auswählen und ausführen:



Da in der Command-Spezifikation für 'firsthello' die Option "storage" True gesetzt wurde, ist nun auch bereits ein Eintrag in der Datenspeicher-Entität des Commands vorhanden. Um den Eintrag zu selektieren, wird hier beispielhaft der mitgelieferte Befehl 'selection' verwendet und direkt in die Befehlszeile eingegeben:



Den Applikationsserver mit `Strg+C` im Terminalfenster beenden:

```
2024-02-03 11:25:28,720 | INFO | app1 | 110 | Command 'firsthello' dem Cache hinzugefügt.
2024-02-03 11:37:49,503 | INFO | app1 | 110 | Command 'selection' dem Cache hinzugefügt.
^C2024-02-03 11:44:42,556 | INFO | app1 | 151 | Appserver hat Stop-Anfrage erhalten...
2024-02-03 11:44:42,556 | INFO | app1 | 113 | SchlieÙe Commands im Cache...
2024-02-03 11:44:42,557 | INFO | app1 | 114 | Command 'firsthello' geschlossen.
2024-02-03 11:44:42,557 | INFO | app1 | 114 | Command 'selection' geschlossen.
2024-02-03 11:44:42,557 | INFO | app1 | 117 | Commands geschlossen.
2024-02-03 11:44:42,558 | INFO | app1 | 122 | SchlieÙe Monitor...
2024-02-03 11:44:42,558 | INFO | app1 | 124 | Monitor geschlossen.
2024-02-03 11:44:42,558 | INFO | app1 | 901 | Logger 'app1' wird geschlossen.
```

6.2 Weitere Schritte

Automatisierung

Bei Bedarf könnte 'firsthello' auch per Zeitplan aufgerufen werden, indem ein Schedule-Eintrag in 'appdef.py' unter 'COMMAND_SCHEDULESETS' ergänzt wird. Um den Command täglich mit einem Intervall von 120 Sekunden aufzurufen, könnte beispielsweise ergänzt werden:

```
"schedule_hello": {"cmdpreset": "firsthello", "days": "*", "intervaldiff": 120, "appnum": 1},
```

Hinweis: Der Applikationsserver ist nach Änderungen in 'appdef.py' jeweils neu zu starten.

Umsetzung individueller Aufgaben

Um individuelle Aufgaben zu realisieren, können eigene Command-Klassen praktisch beliebig ausgestaltet werden, sofern das dem vorausgehend beschriebenen Muster entsprechend erfolgt.

Bei minimalen Aufgaben reicht dafür vielleicht schon eine private Methode wie '__get_result' in 'FirstHello' aus, oder es können weitere private Methoden programmiert werden.

Funktionen, die entweder nicht in Python enthaltene Pakete oder virtuelle/physische Geräte erfordern (externe Abhängigkeiten), sowie komplexe oder umfangreiche Funktionen sollten generell in einer eigenen Device-Klasse repräsentiert werden, die dann von der Command-Klasse verwendet wird.

Auch für eigene Device-Klassen kann nach dem vorausgehend für Command-Klassen beschriebenen Muster verfahren werden, eine vorhandene Klasse zu kopieren, die erforderlichen Umbenennungen durchzuführen und dann nach Bedarf zu codieren. Außerdem sind für Device-Klassen in 'appdef.py' unter 'NAME_DEVICE_MAP' und 'DEVICE_CLASS_CONNECTION_SPEC' entsprechende Ergänzungen vorzunehmen.

Viel Spaß beim Experimentieren und Umsetzen eigener Ideen!

7. Backup

Wie allgemein bekannt sollten relevante Daten in angemessenen Abständen gesichert werden. Dies kann durch einfaches Kopieren des jeweiligen kompletten myminapp-Verzeichnisses erfolgen.

Je nach Ausprägung und Nutzung der myminapp-Applikation können im laufenden Betrieb möglicherweise Sperren auf Dateien liegen, insbesondere im Bereich '/data/storage'. Es ist dann zu empfehlen, die betreffenden Applikationsinstanzen kurz zu beenden, die Kopie durchzuführen und die Applikationsinstanzen dann wieder zu starten.

Anhang

A. Lizenzen

1. Python

Siehe <https://docs.python.org/3/license.html>

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 3.12.0 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 3.12.0 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All Rights Reserved" are retained in Python 3.12.0 alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 3.12.0 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 3.12.0.
4. PSF is making Python 3.12.0 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 3.12.0 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.12.0 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.12.0, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python 3.12.0, Licensee agrees to be bound by the terms and conditions of this License Agreement.

2. myminapp

myminapp

Copyright (c) 2024 - berryunit

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.



THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3. Ergänzende Bibliotheken/Pakete/Module

Um die Applikation 'myminapp' zu installieren und mit Teilen der mitgelieferten Grundausstattung auszuführen, wird Python 3 ab Version 10 benötigt. Weitere Bibliotheken/Pakete/Module werden dafür nicht benötigt.

Allerdings setzen mitgelieferte Device-Klassen ergänzende Python-Elemente voraus, worauf sowohl im Quellcode als auch in dieser Dokumentation hingewiesen wird. Sofern der Anwender solche Device-Klassen nutzt, obliegt es ihm, die betreffenden Lizenzbestimmungen zu beachten.



B. Installieren mit pip

Als Beispiel-Applikation sollte myminapp aus der Releasedatei in ein Benutzerverzeichnis installiert werden, wie im Kapitel 'Quickstart' beschrieben. So kann am einfachsten auf den Quellcode zugegriffen werden, und parallele Installationen sind einfach zu handhaben.

Alternativ kann myminapp aber auch vom Python-Paket-Index (PyPI) installiert werden. Die Installation sollte mit Benutzerrechten erfolgen. Dazu ist vom Benutzer (hier 'u1') einzugeben:

```
pip install myminapp
```

Damit würde myminapp beispielsweise an das folgende Ziel installiert:

```
/home/u1/.local/lib/PythonVersionsnummer/site-packages/myminapp
```

Zur Anzeige der Installation eingeben:

```
pip show myminapp
```

Zur Deinstallation eingeben:

```
pip uninstall myminapp
```

C. Zertifikate erstellen

Für die Kommunikation via HTTPS werden Zertifikate benötigt. Im Heimnetzwerk können dafür selbst-signierte Zertifikate verwendet werden, einschließlich eines CA-Zertifikats, das im Normalfall von einer 'Certification Authority' (Zertifizierungsstelle) gestellt wird. Dieses kann in Browser importiert werden, damit sie die HTTPS-Verbindung zum Applikationsserver als vertrauenswürdig einstufen und daher keine Warnungen ausgeben oder den Zugriff verhindern.

Die myminapp-Applikation wird bereits mit folgenden selbst erstellten Zertifikatsdateien geliefert:

- /data/cert/ca/mycacert.crt - CA-Zertifikatsdatei
- /data/cert/cert.pem - Zertifikatsdatei für den Applikationsserver (192.168.178.26)
- /data/cert/key.pem - Zugehörige Schlüsseldatei

Im Folgenden wird beispielhaft beschrieben, wie diese Zertifikatsdateien unter Windows mittels 'Open SSL' selbst erstellt werden können.

Schritt 1: OpenSSL herunterladen und installieren

Das Tool 'Win64 OpenSSL v3.2.0' Light von <https://slproweb.com/products/Win32OpenSSL.html> herunterladen und installieren. Für das Beispiel wurde Version v3.2.0 verwendet. Nach der Installation steht der 'Win64 OpenSSL Command Prompt' zur Verfügung. Diesen öffnen und per 'cd' in ein geeignetes Verzeichnis wechseln, beispielsweise nach 'c:/mycerts'. Darin werden für das Beispiel die Verzeichnisse 'ca' und 'ca/private' benötigt.

Die weiteren Schritte sind über den 'Win64 OpenSSL Command Prompt' auszuführen (Befehle ohne Zeilenumbruch eingeben).

Schritt2: CA-Zertifikat erstellen

2.1 Privaten CA-Schlüssel generieren

```
openssl genpkey -algorithm RSA -aes128 -out ./ca/private/mycakey.pem -outform PEM -pkeyopt  
rsa_keygen_bits:2048
```

Resultat: ./ca/private/mycakey.pem

2.2 CA-Zertifikat (auch 'Root-Zertifikat' genannt) erstellen

```
openssl req -x509 -noenc -sha256 -days 3650 -key ./ca/private/mycakey.pem -out  
./ca/mycacert.crt  
...  
Country Name (2 letter code) [AU]:DE  
State or Province Name (full name) [Some-State]:SH  
Locality Name (eg, city) []:  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ME  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []:  
Email Address []:
```

Resultat: ./ca/mycacert.crt

Schritt 3: Zertifikat für Applikationsserver erstellen

3.1 Privaten Schlüssel generieren

```
openssl genpkey -algorithm RSA -out ./key.pem -outform PEM -pkeyopt rsa_keygen_bits:2048
```

Resultat: ./key.pem



3.2 Zertifikats-Signierungs-Request (CSR - Certificate Signing Request) erzeugen

Hiermit werden Daten des privaten Schlüssels mit Daten des 'Zertifikat-Antragstellers' kombiniert. Das Resultat wird Zertifikat-Request genannt und für die Erstellung des eigentlichen Zertifikats benötigt.

```
openssl req -new -key ./key.pem -out ./cert.csr
...
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:SH
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ME
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Resultat: ./cert.csr

3.3 Extensions-Konfigurationsdatei 'cert.ext' mit Editor erstellen

Hiermit werden die eigentlich als 'Common Name' anzugebenden Domains spezifiziert. Offenbar ist die Angabe direkt in Common Name seit langer Zeit schon ein Kandidat für Deprecation, daher sollte generell die Extensionsdatei verwendet werden. Das Muster für die betreffende Sektion ist:

```
[alt_names]
IP.1 = 127.0.0.1
IP.2 = x.x.x.x
...
DNS.1 = localhost
DNS.2 = abc.my.domain
DNS.3 = *.my.domain
...
```

Beispiel-Inhalt (wie für die mitgelieferten Zertifikatsdateien verwendet):

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.178.26
```

3.4 Zertifikat erstellen

Auf Basis des CSR und der CA wird nun ein Zertifikat erstellt und signiert. Außerdem wird unter './ca' eine Datei 'mycacert.srl' erzeugt. Darin wird eine 'serial number' generiert, die dem erstellten Zertifikat eindeutig zugeordnet werden kann. Dies ist erforderlich, damit die CA bei künftig erstellten Zertifikaten immer eine neue eindeutige Nummer generieren kann.

```
openssl x509 -req -in ./cert.csr -CA ./ca/mycacert.crt -CAkey ./ca/private/mycakey.pem -
CAcreateserial -out ./cert.crt -days 3650 -sha256 -extfile ./cert.ext
...
Certificate request self-signature ok
...

Resultat 1: ./ca/mycacert.srl
Resultat 2: ./cert.crt
```



3.5 PEM-Variante aus .crt-Zertifikat erstellen

```
openssl x509 -in ./cert.crt -out ./cert.pem -outform PEM
```

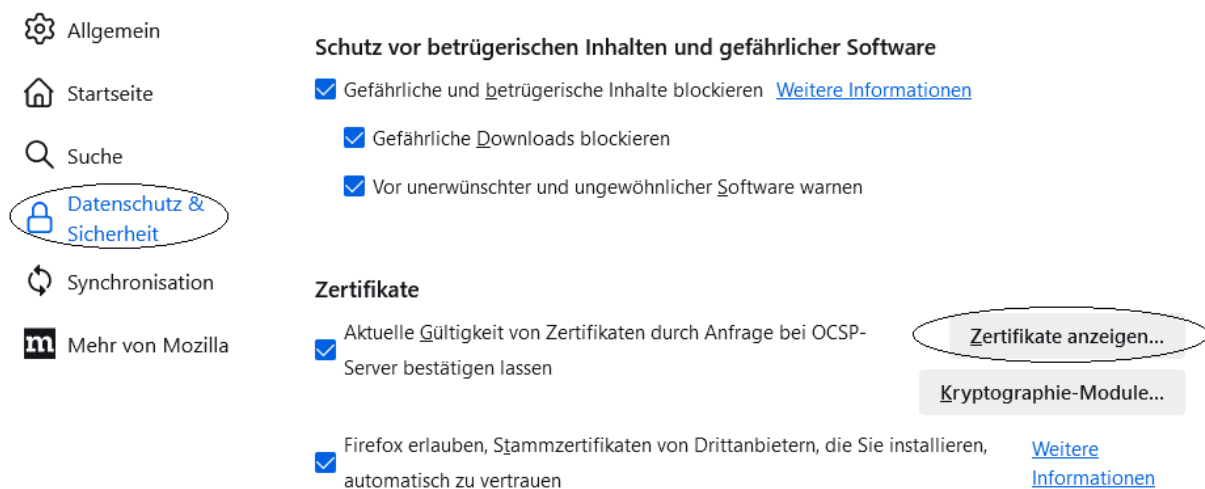
Resultat: ./cert.pem

Damit stehen nun alle benötigten Dateien zur Verfügung. Die im Verzeichnis '/data/cert/' befindlichen mitgelieferten Dateien sind mit den entsprechenden selbst erstellten Dateien zu ersetzen.

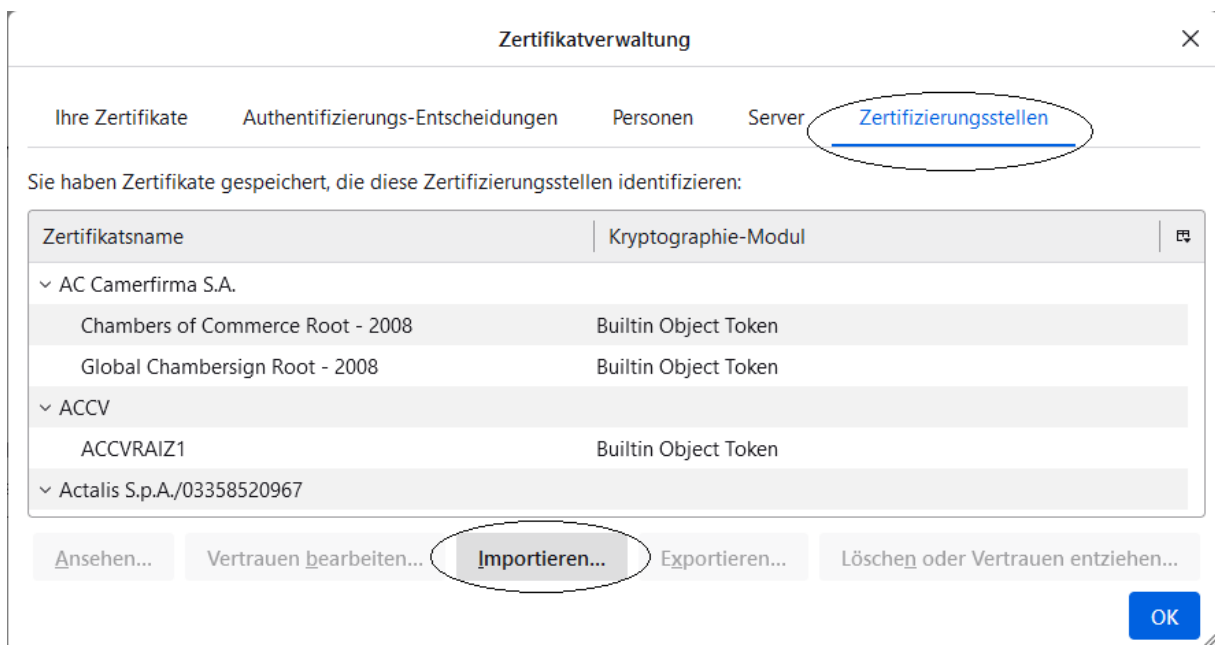
Schritt 4: CA-Zertifikat in Browser importieren

Der Vorgang wird beispielhaft anhand Mozilla Firefox dargestellt. Bei anderen Browsern ist der Vorgang ähnlich.

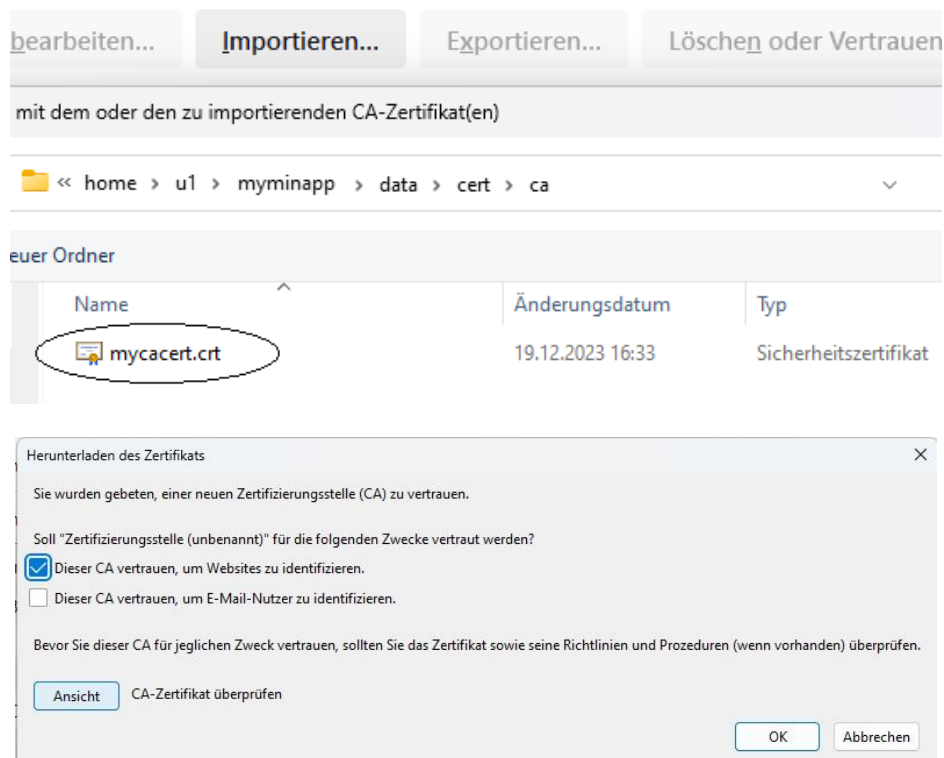
4.1 Über den Menüpunkt 'Einstellungen' in den relevanten Bereich navigieren



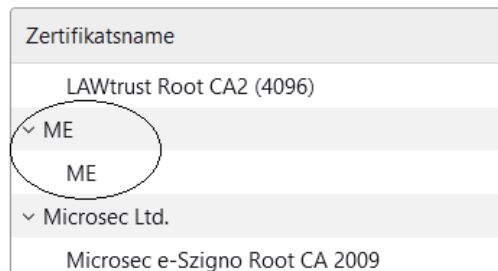
4.2 In der Zertifikatsverwaltung unter 'Zertifizierungsstellen' den Button 'Importieren' betätigen



4.3 Die CA-Zertifikatsdatei 'mycacert.crt' importieren



Das importierte CA-Zertifikat wird unter dem Namen der Organisation, hier 'ME', angezeigt:



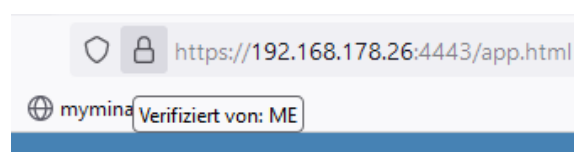
Schritt 5: Applikationsserver mit den passenden Argumenten starten

Die beiden Zertifikatsdateien 'cert.pem' und 'key.pem' sollten in das myminapp-Verzeichnis nach '/data/cert/' kopiert werden und dort die mitgelieferten Dateien ersetzen.

Der Aufruf des Applikationsservers kann dann beispielsweise wie folgt aussehen (ohne Zeilenumbruch):

```
python3 -m myminapp.appserver.py -host 192.168.178.26 -port 4443 -cert  
./myminapp/data/cert/cert.pem -key ./myminapp/data/cert/key.pem
```

Im Browser kann die HTTPS-Verbindung nun ohne Warnungen hergestellt werden:



D. Daten für statistische Zwecke aufzeichnen

Bei der kontinuierlichen, zeitbezogenen Ermittlung von Daten zur späteren statistischen Auswertung sind diverse Aspekte zu berücksichtigen. Im Folgenden wird dies anhand der Aufzeichnung und statistischen Auswertung von Statusdaten erläutert.

Ziel war in diesem Kontext, Verbrauchs- und Leistungsdaten bezogen auf diverse Zeiteinheiten praxistgerecht und zuverlässig darstellen zu können. Dazu werden die Utilities 'Scheduler' und 'Storage' sowie die Commands 'state' und 'epstats' verwendet.

1. Zeiteinheiten

Als Zeiteinheiten wurden Jahr, Monat, Tag, Stunde und Minute festgelegt. Diese Zeiteinheiten können einem Zeitstempel wie '2023-12-09 11:05' und seinen einzelnen Elementen direkt zugeordnet werden. Sekunden oder sogar Millisekunden wären hier nicht sinnvoll, da die Aussagefähigkeit der Statistiken praktisch nicht verbessert würde; außerdem ist die Aktualisierungstaktung physischer Einheiten wie Schaltsteckdosen etc. häufig langsamer.

2. Zugriff

Um den Zugriff auf gespeicherte Daten über Zeiteinheiten einfach und schnell zu machen, wurde für alle Entitäten des Datenspeichers ein Eintrags-Präfix definiert, das jedem Eintrag vorangestellt wird und sowohl den Zeitstempel als auch seine einzelnen Elemente enthält. Pro forma wurde hier auch die Zeiteinheit Sekunde berücksichtigt. Das Präfix besteht aus 8 Feldern vom Typ Integer (Ganzzahl):

id - Eindeutige ID (Primärschlüssel)
t0 - Zeitstempel (Timestamp), hier pro forma inklusive Sekunde
t1 - Jahr
t2 - Monat
t3 - Tag
t4 - Stunde
t5 - Minute
t6 - Sekunde (pro forma)

Die Formulierung einer Abfrage kann nun sehr einfach ein bestimmtes Jahr, einen bestimmten Monat etc. einschließen. Dies ist sowohl zeitraumbezogen möglich, beispielsweise für die ersten 5 Tage des Dezember 2023 ($t1 = 2023$ AND $t2 = 12$ AND $t3 > 0$ AND $t3 < 6$), als auch zeiteinheitenbezogen, beispielsweise tagesübergreifend für Stunde 10 im Monat Dezember 2023 ($t1 = 2023$ AND $t2 = 12$ AND $t4 = 10$). Zu beachten ist, dass Stunden von 0 bis 23 und Minuten von 0 bis 59 gezählt werden.

3. Schreiben

Ein weiterer wesentlicher Aspekt ist der Zeitpunkt des Schreibens von Statusdaten, und zwar unter Berücksichtigung der statistischen Aussagefähigkeit pro Zeiteinheit.

Prinzip A

Würde beispielsweise ein bestimmter Status genau einmal pro Stunde geschrieben, so wäre dies auf den ersten Blick völlig in Ordnung. So würden sich beispielsweise 3 Verbrauchszählerstände, die über Stunde 10 bis 12 jeweils zur 30. Minute geschrieben würden, wie folgt darstellen lassen:

Stunde 10: 250 kWh
Stunde 11: 280 kWh
Stunde 12: 307 kWh



Soll nun aber der Verbrauch pro Zeiteinheit dargestellt werden, so muss dieser durch die statistische Auswertung ermittelt werden, indem die Differenz zum jeweils vorausgehenden Eintrag gebildet wird ($307 - 280 = 27 \text{ kWh}$ und $280 - 250 = 30 \text{ kWh}$). Damit sind zwei Probleme verbunden:

- Die Differenz umfasst generell zwei Zeitpunkte in unterschiedlichen Zeiteinheiten. Dies macht die Zuordnung der Verbrauchswerte bezogen auf eine Zeiteinheit systematisch ungenau.
- Entstehen Lücken in der kontinuierlichen Aufzeichnung der Statusdaten, so ist die Differenz nicht mehr auf zwei direkt aufeinander folgende Zeiteinheiten bezogen. Daraus resultieren irreführende statistische Werte, die beispielsweise in einem Balken- oder Linien-Chart kaum plausibel darstellbar sind.

Prinzip B

Ein deutlich robusteres Prinzip ist, die Statusdaten jeweils zu Beginn und zum Ende einer Zeiteinheit zu schreiben. Die Differenz der beiden Werte ist damit immer auf die jeweilige Zeiteinheit bezogen, und Lücken in der kontinuierlichen Aufzeichnung führen nicht zu irreführenden Werten, sondern sind auch in der statistischen Darstellung als Aufzeichnungslücken erkennbar.

Im Idealfall könnte das vorausgehende Beispiel dann etwa wie folgt aussehen:

Stunde 10: 240 und 268 kWh (Differenz 28 kWh)
Stunde 11: 268 und 290 kWh (Differenz 22 kWh)
Stunde 12: 290 und 320 kWh (Differenz 30 kWh)

'Idealfall' bedeutet hier, dass der Bis-Wert eines Eintrags dem Von-Wert des Folgeeintrags entspricht (im Beispiel 268 und 290 kWh). Dies dürfte auch in den meisten Fällen zutreffen, wenn in den ersten und letzten Sekunden der Zeiteinheit gemessen respektive geschrieben wird.

Es ist aber nicht ausgeschlossen, dass kleinere Abweichungen resultieren, insbesondere wenn gerade beim Übergang zur nächsten Zeiteinheit ein hoher Energiefluss stattfindet. Das könnte dann wie folgt aussehen:

Stunde 10: 240.00 und **268.00** kWh (Differenz 28.00 kWh)
Stunde 11: **268.05** und 290.00 kWh (Differenz **21.05** kWh)
Stunde 12: 290.00 und 320.00 kWh (Differenz 30.00 kWh)

Solche Unschärfen sind also zu erwarten, wirken sich aber auf die statistische Auswertung nur in sehr geringem Maße aus. Bei größeren Zeiteinheiten (Tag, Monat, Jahr) sind sie praktisch irrelevant.

Wichtig ist allerdings, dass die beiden Zeitpunkte des Schreibens durch die Zeitwerte im Eintrags-Präfix konsistent abgebildet werden. Im konkreten Kontext übergibt die Applikationsinstanz jeweils den vom Scheduler für eine Aktion gesetzten Zeitstempel zum Schreiben. Daher werden die Daten auf jeden Fall mit den korrekten Zeitwerten gespeichert, auch wenn die Ausführung einer Aktion das Schreiben am Ende einer Zeiteinheit ausnahmeweise einmal bis in den Beginn der nächsten Zeiteinheit verzögern sollte.

4. Änderungshäufigkeit und -relevanz

Weitere Aspekte sind Häufigkeit und Relevanz der Änderung von Statusdaten. Würde beispielsweise pro Minute oder sogar pro Sekunde gespeichert, obwohl die Daten sich gar nicht oder nur geringfügig geändert haben, so würde dies nicht nur das Speichervolumen unnötig erhöhen, sondern auch bei der statistischen Darstellung zu vielen Einträgen ohne Relevanz führen.



Auf diesem Hintergrund wurde im konkreten Kontext als kleinste sinnvolle Zeiteinheit ein 5-Minuten-Abschnitt festgelegt. Daher spezifiziert im Zeitplanungseintrag die Angabe 'm' beim Parameter 'intervalunit' nicht eine, sondern 5 Minuten: 00:00 bis 04:59, 05:00 bis 09:59 etc. Entsprechend signalisiert der Scheduler eine Aktion zu Beginn und zum Ende eines 5-Minuten-Abschnitts.

Ebenso werden die 5-Minuten-Abschnitte bei der statistischen Auswertung durch eine entsprechend formulierte Abfrage berücksichtigt. Die folgenden Abfragevarianten sollen dies veranschaulichen.

Der Rahmen der Abfragen, die hier über Command 'selection' beispielhaft ausgeführt werden und Daten aus der Datenbanktabelle/Entität 'state' selektieren, sieht wie folgt aus:

```
{"cmd": "selection", "value": "<SELECT ...> FROM state <where clause> <group by clause>;"}
```

Eine Where-Clause schränkt die Selektion auf Stunde 15 des 10. Dezember 2023 sowie die Device 'DESKTOP' ein und lautet für alle Abfragevarianten gleich:

```
WHERE t1=2023 AND t2=12 AND t3=10 AND t4=15 AND devname='DESKTOP'
```

Selektiert werden zunächst: Minute 't5', Sekunde 't6', eine Umrechnung von 't5' auf den betreffenden 5-Minuten-Abschnitt 'm5', sowie die Leistung in Watt 'power':

```
SELECT t5, t6, (t5/5)*5 AS 'm5', ROUND(power, 2) AS 'power'
```

```
t5; t6; m5; power
0; 1; 0; 29.0
4; 53; 0; 29.3
5; 2; 5; 29.3
9; 54; 5; 29.0
10; 1; 10; 29.0
14; 53; 10; 30.2
...
```

Das Resultat zeigt, dass jeweils zu Beginn und zum Ende eines 5-Minuten-Abschnitts geschrieben wurde, hier in Minute 0, Sekunde 1 und Minute 4, Sekunde 53 etc. Beginn und Ende können um einige Sekunden variieren, da der Scheduler alle 3 Sekunden aufgerufen wird und geringfügige Latenzen auftreten können; die Zeiten bleiben aber immer innerhalb des Zeitabschnitts.

Die folgende Selektion reduziert das Resultat auf den 5-Minuten-Abschnitt 'm5' und die Leistung 'power', aber eine Gruppierung findet noch nicht statt:

```
SELECT (t5/5)*5 AS 'm5', ROUND(power, 2) AS 'power'
```

```
m5; power
0; 29.0
0; 29.3
5; 29.3
5; 29.0
10; 29.0
10; 30.2
...
```

Schließlich wird gruppiert nach 'm5' mit dem Durchschnitt von 'power' selektiert. In dieser Form ist das Resultat für die Darstellung der durchschnittlichen Leistung pro 5 Minuten in Statistik-Tabellen und -Charts brauchbar.

```
SELECT (t5/5)*5 AS 'm5', ROUND(AVG(power), 2) AS 'power' ... GROUP BY 1
```

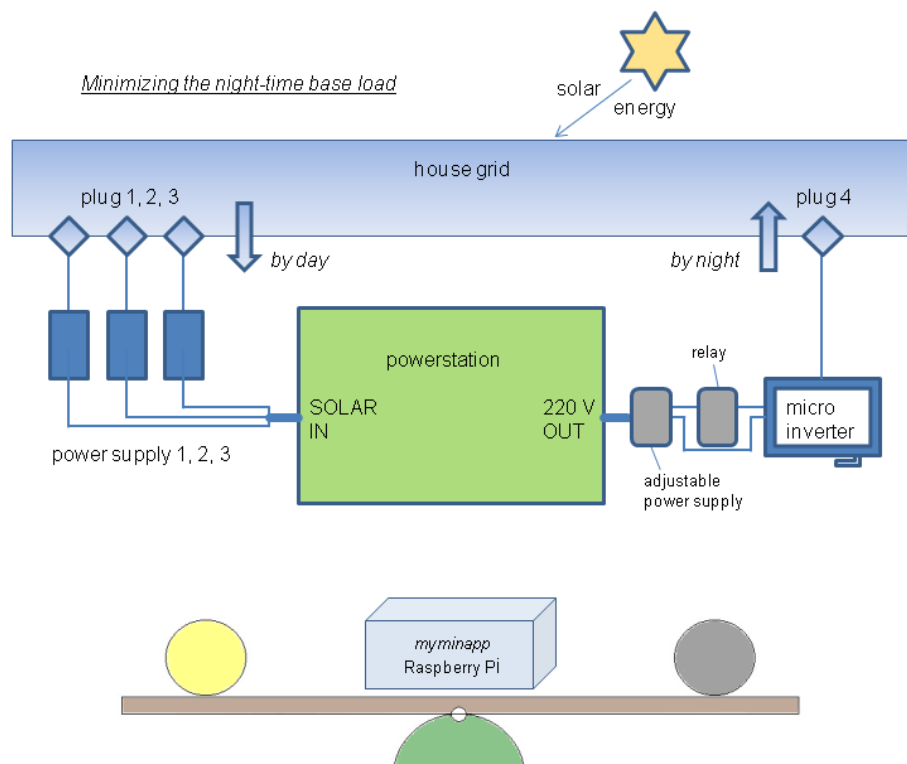
```
m5; power
0; 29.15
5; 29.15
10; 29.6
...
```

E. Beispielszenario

Mit dem folgenden Schaubild wird beispielhaft ein Automatisierungs-Szenario skizziert, das mit der myminapp-Applikation und einigen mitgelieferten Commands und Devices umgesetzt wurde.

Tagsüber wird überschüssige Leistung aus einem Standard-Balkonkraftwerk aus dem Hausnetz über bis zu drei ausgediente Laptop-Netzteile an eine Powerstation abgegeben (siehe Command 'pscharge').

Nachts wird verfügbare Leistung aus der Powerstation in das Hausnetz eingespeist, womit die nächtliche Grundlast minimiert wird (siehe Command 'psdischarge').



F. Programmentwicklung

Dieses Kapitel informiert über konzeptionelle Aspekte der Entwicklung von 'myminapp'.

1. Zielsetzung

Das Projekt 'myminapp' ist als Beispielsammlung in Form einer vollständigen und funktionsfähigen Python-Applikation zu verstehen. Sie soll es ermöglichen, kleinere DIY-Projekte im Heimbereich mit geringem Aufwand zu realisieren.

Die Applikation soll auf einfachste Weise vom Terminal aus anwendbar sein, in andere Programme eingebunden werden können und Anfragen via HTTP(S) im JSON-Format unterstützen.

Aktionen sollen in Form von Befehlen organisiert und ausgeführt werden. Die Befehle sollen als Klassen gekapselt und von einer einzigen Superklasse abgeleitet werden, so dass Anwender ihre eigenen Befehle für individuelle Aufgaben auf einfache Weise ergänzen können. Als Begriff für 'Befehl' wird im Folgenden 'Command' verwendet.

Die weitere Ausstattung der Applikation soll in Form von Klassen erfolgen, die jeweils eine physische oder virtuelle Einheit repräsentieren und von den Commands genutzt werden können. Als Begriff für 'Einheit' wird im Folgenden 'Device' verwendet.

2. Programmiersprache

Aus den folgenden Gründen wurde Python als Programmiersprache gewählt:

- Python ist eine mächtige, weit verbreitete und sehr beliebte Programmiersprache, die leicht erlernt und angewendet werden kann
- Sie ist für einfachste bis anspruchsvollste Aufgaben plattformübergreifend einsetzbar
- Das Reservoir an allgemein verfügbaren Informationen, Tutorials, Beispielen und zusätzlich installierbaren Bibliotheken/Paketen ist praktisch unerschöpflich
- Als Quellcode/Bytecode-Interpretersprache ist Python besonders flexibel, was beispielsweise die dynamische Einbindung von Ressourcen betrifft
- Unter Sicherheitsaspekten ist zu beachten, dass Python-Programme in der Regel offen im Quellcode vorliegen; im hier adressierten Anwendungsbereich ist dies jedoch unkritisch
- Die Lizenz der Python Software Foundation (PSF) ist eine freizügige Open-Source-Lizenz (siehe <https://docs.python.org/3/license.html>)

3. Richtlinien

Die folgenden Richtlinien zielen darauf, eine möglichst einfach strukturierte, stabile und flexible Applikation zu erhalten, die leicht anwendbar sowie gut lesbar und erweiterbar ist.

3.1 Allgemeine Richtlinien für die Umsetzung der Applikation

- Keine Abhängigkeit von einer bestimmten Entwicklungsumgebung oder Plattform
- Keine Abhängigkeit von bestimmten Frameworks
- Keine Abhängigkeit von externen Python-Paketen bei grundlegenden Features
- Möglichst wenige Abhängigkeiten von externen Python-Paketen bei speziellen Features
- Organisation der Applikation in einer einfachen, transparenten Struktur
- Applikationsdefinition/-konfiguration in nur einer Datei mit Python-Datentypen
- Nachrichten in Deutsch und Englisch in nur einer Datei mit Python-Datentypen
- Organisation des Programmcodes in Klassen in jeweils einer exklusiven Datei

- Grundsätzlich objektorientierter Code, der aber auch prozedurale Anteile haben darf
- Modul-, Klassen-, Methoden- und Variablennamen sowie Code-Kommentare in Englisch
- Docstring-Dokumentation der Module, Klassen und Methoden in Englisch
- Einfache Befehls-Basisklasse für die Ableitung eigener Befehle
- Dynamische Instanziierung der Befehlsklassen und dynamischer Aufruf ihrer Methoden
- Datenformat für Befehls-Ein- und Ausgaben intern: Python-Typ 'dictionary'
- Datenformat für Befehls-Ein- und Ausgaben extern: JSON
- Kommunikation mit dem Datenspeicher nach dem Muster eines Interface
- Implementierung des Datenspeichers mit SQLite (in Python enthalten)
- Testklassen nach einheitlichem Muster für jeden Befehl und jedes Utility
- Dokumentation der Applikation in einem Handbuch im PDF-Format in Deutsch und Englisch

3.2 Auf Sicherheitsaspekte zielende Richtlinien

- Die Applikation ist für den Heimbereich und den Einsatz im lokalen Netzwerk gedacht. Es wird daher zugunsten der einfachen Anwendbarkeit auf Authentifizierung, Rollen und Rechteverwaltung verzichtet
- Die Kommunikation via HTTPS soll optional unterstützt werden, um bei Anwendung des Web-Frontends den aktuellen Standards von Browsern zu entsprechen
- Die Manipulation der Applikation über die angebotenen Kommunikationswege respektive die mitgelieferten Befehle soll so gut wie ausgeschlossen sein
- Die Applikationsdefinition/-konfiguration soll nur durch direktes Editieren der betreffenden Datei erfolgen können

3.3 Richtlinien für das Coding

- Der Code soll im Wesentlichen objektorientiert sein und Python-Standards entsprechen
- Die Qualität des Codes sollte gut sein, aber sehr hohe Anforderungen können und sollen hier nicht erfüllt werden
- Der Code soll praxisgerecht im Sinne der Zielsetzung der Applikation sein; beispielsweise sollte nicht zu fein granuliert werden, um die Applikationsstruktur übersichtlich zu halten
- Jede Klasse soll in einer eigenen Datei (einem eigenen Modul) codiert werden, um die Struktur transparent zu halten und eine gute Wartbarkeit zu unterstützen
- Innerhalb der Klassen soll jeweils nach öffentlichen und privaten Methoden unterschieden werden ('def a_public_method()' und 'def __a_private_method()')
- Methoden sollen inklusive Kommentarzeilen typischerweise nicht mehr als 50 und im Ausnahmefall maximal 200 Zeilen umfassen

3.4 Richtlinien für das Exception-Handling

- Um die Applikation der Zielsetzung entsprechend möglichst einfach zu halten, soll auf eigene Exception-Ableitungen verzichtet werden
- Wenn innerhalb einer Methode in jedem Fall eine finale Aktion stattfinden soll, ist der betreffende Code in einen try-final-Block einzuschließen (beispielsweise um eine Datenbankverbindung sicher zu schließen, bevor die Methode verlassen wird)
- Innerhalb einer Methode von Hilfs- und Ausstattungsklassen (Utility, Device) sollen try-except-Blöcke möglichst vermieden werden, es sei denn eine Exception muss unbedingt direkt behandelt werden (beispielsweise für Retries ohne die Methode zu verlassen)
- Bei Plausibilitätsprüfungen und in ähnlichen Fällen kann raise Exception angewendet werden
- Commands sollen eine bei einer Aktion auftretende Exception mit einem Fehlercode, Fehlertext und Trace-Text in das Resultat übertragen. Dies ermöglicht die Ausführung einer Serie von Aktionen innerhalb eines Commands



- Die Behandlung von Exceptions soll möglichst nur auf der höchsten Ebene der Applikation erfolgen
- Die auf höchster Ebene behandelten Exceptions sollen immer mit Fehlertext und Trace-Text geloggt werden
- Der Applikationsserver, der HTTP(S)-GET-Anfragen verarbeitet, soll beim Scheitern der grundlegenden Prüfung einer Anfrage HTTP-Code 400 (File not found), und beim generellen Scheitern eines Commands HTTP-Code 500 (internal server error) mit ergänzendem Fehlertext zurückliefern
- Das Web Frontend soll bei Exceptions innerhalb der Command-Ausführung den Fehlertext im Resultat der Antwort erhalten, nicht den Trace-Text

4. Grundausrüstung

Die Applikation soll in einer Grundausrüstung geliefert werden, die außer den grundlegenden Features auch bereits umgesetzte Commands und Devices enthält. Diese sollen sowohl funktionsfähig sein als auch als Muster dienen können.

Abhängigkeiten über die Python-Standardbibliothek hinaus sollen möglichst auf Devices beschränkt bleiben.

5. Entwicklungsumgebung

Es sollte eine für Python gut geeignete IDE wie Microsoft Visual Studio Code genutzt werden, beispielsweise unter Windows mit dem Windows Subsystem für Linux (WSL). Die Versionskontrolle soll mit GIT erfolgen.

Im Sinne der Zielsetzung der Applikation wird im Übrigen auf jegliches Framework verzichtet.

6. Distribution, Installation, Veröffentlichung

Zur Distribution wird die Applikation in eine Releasedatei 'myminapp-<Versionsnummer>.zip' gepackt.

Zur Installation ist die Releasedatei zu entpacken. Danach ist die Applikation direkt mit Python 3.10 oder höher anwendbar, da die grundlegenden Features ohne externe Abhängigkeiten auskommen. Wichtig ist auch, dass mehrere Installationen parallel autark existieren können, indem die Releasedatei einfach in weitere Verzeichnisse entpackt oder eine vorhandene Installation kopiert wird.

Diese Art der Installation ist der einfachste Weg, um auf den Quellcode zuzugreifen, und Kopien für Ableitungen sind einfach zu handhaben. Alternativ sollte myminapp auch über den Package Installer for Python (pip) installierbar sein.

Die Veröffentlichung der Applikation soll via GitHub (und Python Package Index (PyPI)) erfolgen.

7. Entwicklungsprozess

Ob eine Weiterentwicklung der Applikation durch den Autor oder ein Team stattfindet, wird an dieser Stelle offen gelassen.



G. Credits

Vielen Dank an...

- die vielen Fachleute und engagierten Menschen, die ihr Wissen und ihre Erfahrung zu den Themen Erneuerbare Energien und Smart Home teilen
- die Profi- und Hobby-Softwareentwickler, die über Tutorials, Videos, Blogs und auf anderen Wegen ihr Wissen und ihre Erfahrung teilen, hier speziell zur Programmierung mit Python
- die Python Software Foundation speziell
- die Open Source Community allgemein
- Microsoft und die involvierten Entwickler für Visual Studio Code und GitHub
- DeepL für die Unterstützung bei der Textübersetzung ins Englische

