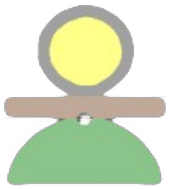


mywattsmon

Handbuch	
Datum:	2024-10-06
Programmversion:	0.9.5
Autor:	berryunit

Inhalt

Inhalt 2
1. Quickstart 3
1.1 Installieren 3
1.2 Anwenden 3
1.3 Weitere Informationen 3
2. Konfigurieren 4
2.1 Überblick 5
2.2 Allgemeine Angaben 6
2.3 window 6
2.3.1 colors 7
2.3.2 grid 7
2.4 database 8
2.5 devices 9
2.6 Zu beachten 9
3. Geräteklassen 9
3.1 Standards 9
3.2 Eigene Geräteklassen 11
4. FAQ 12

1. Quickstart

‚mywattsmon' ist eine minimale Python-Applikation zur Überwachung von elektrischer Leistung und Energie im Smarthome.

- Unterstützung von Geräten wie Energiezähler, Schaltsteckdosen etc.
- 24/7 Monitorprozess mit Datenspeicherung nach Zeitplan
- Optionales Monitorfenster
- Geringer Ressourcenbedarf
- Leicht konfigurierbar via JSON-Datei
- Erweiterbar durch eigene Geräte-Klassen

Vorausgesetzt wird ein Rechner, auf dem Python ab Version 3.11 läuft. Für SBCs wie Raspberry Pi ist eine Festplatte (beispielsweise eine USB-SSD) zu empfehlen, da SD-Karten für den Dauerbetrieb im Allgemeinen nicht geeignet sind.

1.1 Installieren

Die Applikation sollte in ein Benutzerverzeichnis installiert werden, da sie Daten speichert und individuell erweitert werden kann.

```
python -m pip install mywattsmon -U -t <Zielverzeichnis>
```

Alternativ kann die Release-Datei vom Repository heruntergeladen und entpackt werden.

1.2 Anwenden

Im Folgenden wird angenommen, dass die Applikation auf einem Linux-Computer in das Home-Verzeichnis des Benutzers installiert wurde (beispielsweise in `/home/u1/mywattsmon`) und dass die Aufrufe vom Home-Verzeichnis aus erfolgen (`/home/u1`).

Den Monitorprozess starten (beenden mit `Ctrl+C`):

```
python -m mywattsmon.app.monitor
```

Das Monitorfenster starten (beenden mit `Ctrl+C`, im Fenster per Exit-Button oder Escape-Taste):

```
python -m mywattsmon.app.window
```

Hinweis: Beim ersten Start der Applikation wird das Datenverzeichnis `mywattsmon-data` parallel zum Applikationsverzeichnis erstellt. Darin ist unter anderem die Konfigurationsdatei `config.json` mit einer Konfiguration der Geräteklasse `Mock` enthalten. Da diese Klasse Zufallszahlen liefert, ist die Applikation direkt nach der Installation ohne weitere Konfiguration ausführbar.

1.3 Weitere Informationen

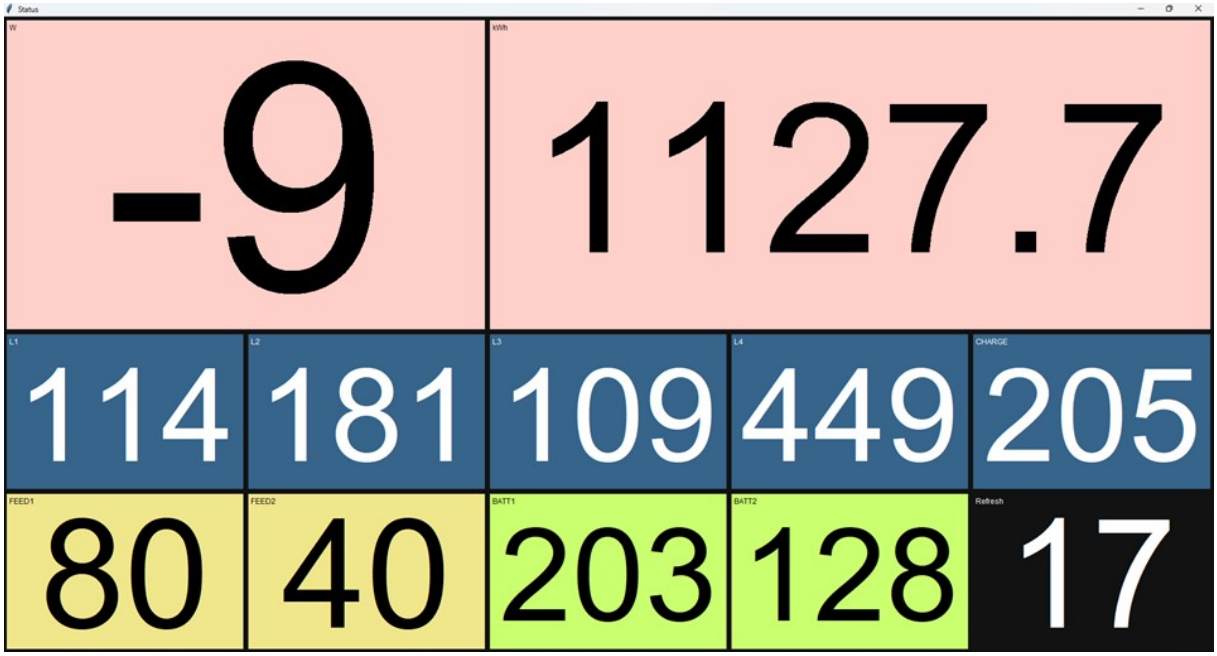
- Dokumentation: `/mywattsmon/doc/*`
- Repository: `https://github.com/berryunit/mywattsmon`
- Lizenz: MIT

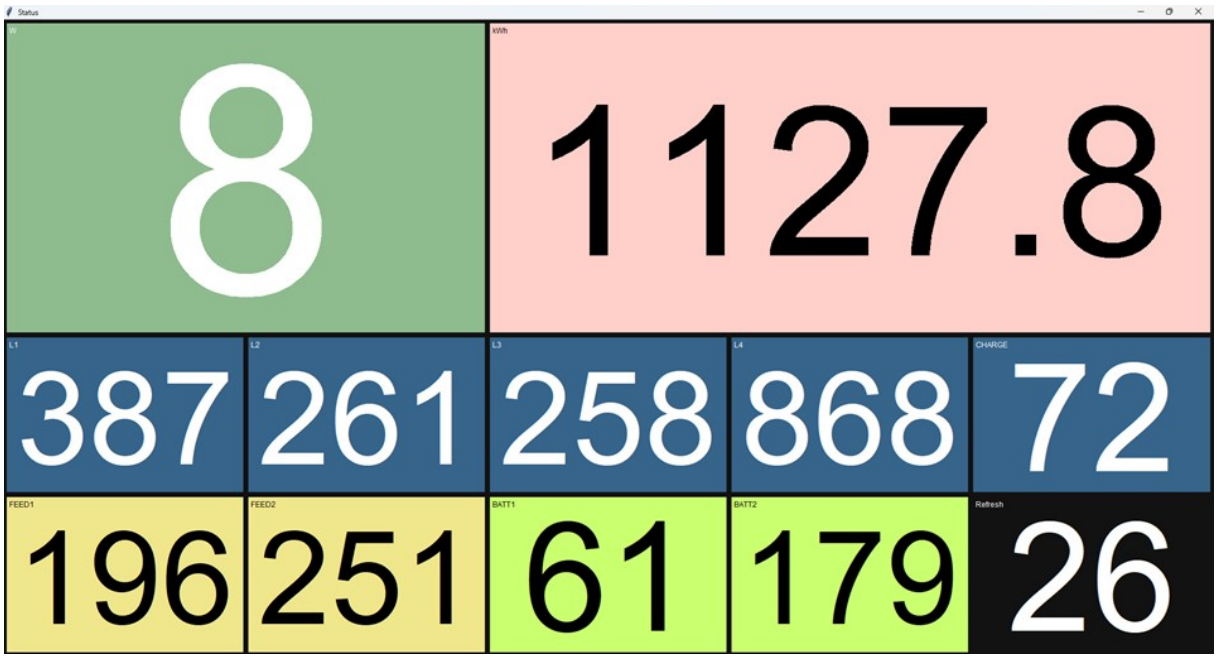
2. Konfigurieren

Die folgenden Beispielbilder zeigen das Monitorfenster mit Zufallszahlen der vorkonfigurierten Geräte-Klasse `Mock`.

In der ersten Zeile werden die aktuelle Gesamtleistung und die kumulierte Gesamtenergie dargestellt. Damit wird ein zentraler Energiezähler simuliert, der diese Werte in Watt respektive Kilowattstunden liefert. Die farbliche Darstellung des Leistungswerts ist davon abhängig, ob dieser negativ oder positiv ist (Verbrauch oder Einspeisung in das öffentliche Stromnetz).

Im Feld unten rechts läuft ein Countdown bis zur nächsten Aktualisierung der Werte. Die anderen Felder simulieren die Leistung in Watt für Geräte wie beispielsweise Schaltsteckdosen.





2.1 Überblick

Zunächst wird der Inhalt der Datei `/mywattsmon-data/config.json` anhand eines Beispiels im Überblick dargestellt. Anschließend werden die einzelnen Konfigurationsbereiche detailliert beschrieben.

```
{
    "title":"mywattsmon configuration",
    "loglevel":"info",
    "logtofile":"false",
    "port":42001,
    "window":{
        "title":"Status",
        "size":"max",
        "interval":{"default":30,"20:00-22:00":60,"22:00-05:00":300},
        "nightmode":{"timeframe":"22:00-05:00","colors":"ref0"},
        "colors":{
            "window":{"bg":"gray7"},
            "values":{
                "ref0":{"bg":"black","fg":"gray7"},
                <...>,
                "ref6":{"bg":"gray7","fg":"white"}
            }
        },
        "grid":{
            "W":{
                "units":"M1","var":"power","colors":"ref1",
                "rownum":0,"colnum":0,"rowspan":2,"colspan":2
            },
            <...>,
            "Refresh":{
                "units":"<refresh>","var":"<countdown>","colors":"ref6",
                "rownum":3,"colnum":4,"rowspan":1,"colspan":1
            }
        }
    },
    "database":{
        "times":["00:00","03:00","06:00","09:00","12:00","15:00",
                "18:00","21:00","23:59"],
        "tablename":"kwh",
        "columns":{
            "em":{"units":"M1","var":"energy"},
            <...>,
            "batt2":{"units":"M14","var":"energy"}
        }
    },
    "devices":{
        "Mock":{
            "module":"mywattsmon.app.device.mock",
            "units":{
                "M1":{"uid":"m1"},
                <...>,
                "M14":{"uid":"m14"}
            }
        }
    }
}
```

2.2 Allgemeine Angaben

Schlüssel	Wert	Beschreibung
`title`	`<Titeltext>`	Informativer Text für die Konfiguration. Beispiel: `"mywattsmon configuration"`
`loglevel`	`test, info, warning, error`	Stufe, ab der Informationen an das Log-Ziel ausgegeben werden. Beispiel: `"info"`
`logtofile`	`true, false`	Ausgabe an eine tägliche Log-Datei unter `/mywattsmon-data/log/` (`true`) oder an die Standardausgabe (`false`). Beispiel: `"false"`
`port`	`<Portnummer>`	Lokaler TCP Port , an dem der Monitorprozess auf lokale Anfragen (des Monitorfensters) wartet. Beispiel: `42001`

2.3 window

Schlüssel	Wert	Beschreibung
`title`	`<Titeltext>`	Informativer Text für den Fensterrahmen. Beispiel: `"mywattsmon configuration"`
`size`	`max, full, <w*h+x+y>`	Fenstergröße: Größtmögliches Fenster mit Rahmen (`max`), Fullscreen (`full`), oder explizite Breite, Höhe, X- und Y-Position. Beispiele (alternativ): `"max"` `"full"` `"640*400+10+10"`
`interval`	`<Liste mit Schlüssel-Wert-Paaren>`	Angabe zeitabhängiger Aktualisierungs-Intervalle. Schlüssel: `<Zeitrahmen im Format HH:MM-HH:MM>`. Wert: `<Anzahl Sekunden>`. Liegt die aktuelle Zeit außerhalb aller angegebenen Zeitrahmen, wird der Schlüssel `default` verwendet. Beispiel: `{` `"default":30,` `"20:00-22:00":60,` `"22:00-05:00":300` `}`
`nightmode`	`<Liste mit Schlüssel-Wert-Paaren>`	Angabe eines Zeitrahmens und eines Farbsets für eine dunkle Darstellung. Schlüssel: `timeframe`. Wert: `<Zeitrahmen im Format HH:MM-HH:MM>`. Schlüssel: `colors`. Wert: `<Farbreferenz-`

		schlüssel>. Beispiel: { "timeframe":"22:00-05:00", "colors":"ref0" }
colors	<siehe Abschnitt 2.3.1 colors>	Angabe der Hintergrundfarbe und diverser Farbsets zur farblichen Darstellung der Werte. Anzugeben sind von Python unterstützte Farben.
grid	<siehe Abschnitt 2.3.2 grid>	Spezifikation des Gitters zur Darstellung der Werte im Fenster.

2.3.1 colors

Schlüssel	Wert	Beschreibung
window	<Schlüssel-Wert-Paar>	Hintergrundfarbe des Fensters. Schlüssel: bg. Wert: <Farbe>. Beispiel: {"bg":"gray7"}
values	<Liste mit Schlüssel-Wert-Paaren>	Angabe von Referenzschlüsseln mit Zuordnung einer Hintergrund- und einer Vordergrundfarbe (Schlüssel bg respektive fg). Werden die Schlüssel mit Plus- oder Minuszeichen ergänzt, so wird die Farbe abhängig davon gesetzt, ob der aktuelle Wert positiv oder negativ ist . Beispiel: { "ref0":{"bg":"black","fg":"gray7"}, "ref1":{ "bg+":"darkseagreen","fg+":"white", "bg-":"#FFCFC9","fg-":"black" } }

2.3.2 grid

Schlüssel	Wert	Beschreibung
<Label>	<Liste mit Schlüssel-Wert-Paaren>	Gitterelement-Spezifikation. Für jedes Element ist ein Label anzugeben und wie folgt zu spezifizieren: **Schlüssel** **Wert** units <Geräteeinheitennamen> var <Variable (power oder energy)> colors <Farbreferenzschlüssel> rownum <Zeilennummer (0 bis n)> colnum <Spaltennummer (0 bis n)> rowspan <Zeilenanzahl(0 bis n)> colspan <Spaltenanzahl (0 bis n)>

		Beispiel: `{` `  "W":{` `  "units":"M1","var":"power","colors":"ref1",` `  "rownum":0,"colnum":0,"rowspan":2,"colspan":2` `  },` `  "kWh":{` `  "units":"M1","var":"energy","colors":"ref2",` `  "rownum":0,"colnum":2,"rowspan":2,"colspan":3` `  },` `  "L1":{` `  "units":"M2,M3","var":"power","colors":"ref3",` `  "rownum":2,"colnum":0,"rowspan":1,"colspan":1` `  },` `  <...>,` `  "Refresh":{` `  "units":"**<refresh>**","var":"**<countdown>**",` `  "colors":"ref3","rownum":3,"colnum":4,` `  "rowspan":1,"colspan":1` `  }` `}` *Hinweis: Im letzten Beispieleintrag wird mit `<refresh>` und `<countdown>` das Element zur Darstellung des Aktualisierungs-Countdown spezifiziert.*

2.4 database

Schlüssel	Wert	Beschreibung
`times`	`<Liste>`	Liste von Uhrzeiten im Format HH:MM, zu denen die aktuellen Werte in die Datenbank geschrieben werden. Beispiel: `["00:00","06:00","12:00","18:00","23:59"]` *Hinweis: Die Uhrzeiten `00:00` und `23:59` angeben, um die Daten eines Tages abfragen zu können.*
`tablename`	`<Tabellenname>`	Name der Tabelle in Kleinbuchstaben, in die geschrieben werden soll. Beispiel: `"kwh"`
`columns`	`<Liste mit Schlüssel-Wert-Paaren>`	Spalten-Spezifikation. Für jede Spalte ist ein mit SQLite kompatibler Spaltenname in Kleinbuchstaben anzugeben und wie folgt zu spezifizieren: **Schlüssel** — **Wert** `units` — `<Geräteeinheitennamen>` `var` — `<Variable (power oder energy)>` Beispiel: `"em":{"units":"M1","var":"energy},` `"l1":{"units":"M2,M3","var":"energy},` `<...>,` `"batt2":{"units":"M14","var":"energy}`

2.5 devices

Schlüssel	Wert	Beschreibung
`<Klassenname>`	`<Liste mit Schlüssel-Wert-Paaren>`	Geräteklassen-Spezifikation. Für jedes Gerät sind ein einfacher Klassenname und der explizite Modulname anzugeben, der die Klasse enthält. Für jede Geräteeinheit sind der Geräteeinheitenname und dazu die intern erforderlichen Daten anzugeben, beispielsweise eine Geräteeinheiten-ID (`uid`). Beispiel für Geräteklasse `Mock`: `"Mock":{` `  "module":"mywattsmon.app.device.mock",` `  "units":{` `  "M1":{"uid":"m1"},` `  "M2":{"uid":"m2"},` `  <...>,` `  "M14":{"uid":"m14"}` `  }` `}` Weitere Informationen dazu enthält das folgende Kapitel „Geräteklassen“.

2.6 Zu beachten

Werden Änderungen an der Konfiguration vorgenommen, so ist insbesondere zu beachten:

- Die durch das JSON-Format vorgegebenen Regeln sind zu beachten. Dies betrifft insbesondere die Klammerung, die Kommasetzung sowie die korrekte Syntax der Spezifikationen.

- Sollen Änderungen an der Datenbank-Konfiguration vorgenommen werden, so ist die bisher verwendete Datenbank `/mywattsmon-data/db/monitor.db` zuvor durch Kopieren zu sichern und dann zu entfernen. Die Datenbank und die konfigurierte Datenbanktabelle werden dann automatisch neu erstellt.

3. Geräteklassen

Eine Geräteklasse repräsentiert ein physisches Gerät wie beispielsweise eine Fritzbox oder einen Bosch Smart Home Controller. Alle mitgelieferten Geräteklassen folgen Standards, die in diesem Kapitel beschrieben und bei der Erstellung eigener Geräteklassen zu berücksichtigen sind.

3.1 Standards

Das Modul `mywattsmon.app.device.abstract` enthält die Klasse `Abstract`, die ein informelles Python-Interface darstellt. Das Interface gibt drei Methoden vor, die von der jeweiligen Klasse zu überschreiben sind. Nachfolgend ist der Code der `Klasse Abstract` dargestellt, gefolgt von Beispiel-Code zur Ausführung einer Geräteklasse.

```
class Abstract:

    """
    This class is an informal Python interface and acts as a blueprint
    for all device classes. Each device class  implements (overrides)
    its abstract methods.
    """

    def set_config(self, config:dict):
        """Sets the configuration for this device as needed.

        Args:
            config (dict): Specifications from the configuration file.

        Returns:
            None.
        """
        pass

    def close(self):
        """Closes resources as needed.

        Args:
            None.

        Returns:
            None.
        """
        pass

    def get_infoset(self):
        """Gets information from all configured device units.

        The form of the result is standardized. The following data
        must be supplied for each unit:

        data = {}
        data['power'] = 0
        data['energy'] = 0.0
        data['state'] = 'OFF'
        data['code'] = 2
        data['info'] = ''
        data['trace'] = ''

        infoset = {}
        infoset[<unit1>] = <data_from_unit1>
        infoset[<unit2>] = <data_from_unit2>
        infoset[<unit3>] = <data_from_unit3>
        ...

        See also the code of the implemented device classes.

        Args:
            None.

        Returns:
            dict: The data of all units as 'infoset'.
        """
        pass


from mywattsmon.app.device.mock import Mock
instance = Mock()
instance.set_config(…)
while …:
       infoset = instance.get_infoset()
instance.close()
```

Wie genau eine Klasse im Detail zu konfigurieren ist, wird durch die Klasse selbst bestimmt. Im Folgenden ist eine explizite Konfiguration für alle mitgelieferten Geräteklassen dargestellt, die auf physische Geräte zugreifen. Private Attribute wie Kennwörter etc. sind dabei unkenntlich gemacht.

```
"devices":{
    "DTSU":{
        "module":"mywattsmon.app.device.dtsu",
        "units":{
            "EM":{"port":"/dev/ttyUSB2","address":68}
        }
    },
    "VEDirect":{
        "module":"mywattsmon.app.device.vedirect",
        "units":{
            "BATT1":{"port":"/dev/ttyUSB0"},
            "BATT2":{"port":"/dev/ttyUSB1"}
        }
    },
    "Fritz":{
        "module":"mywattsmon.app.device.fritz",
        "connection":{
            "address":"192.168.178.1",
            "user":"fritzuser",
            "password":"…"
        },
        "units":{
            "HEATER":{"uid":"11657 …"},
            "L1":{"uid":"11630 …"},
            "L2":{"uid":"11630 …"},
            "L3":{"uid":"11630 …"},
            "FEED1":{"uid":"11657 …"},
            "FEED2":{"uid":"11657 …"},
            "CHARGE":{"uid":"11657 …"}
        }
    },
    "Bosch":{
        "module":"mywattsmon.app.device.bosch",
        "connection":{
            "address": "192.168.178.27",
            "certfile": "/home/pi/mywattsmon-data/cert/bosch/shccert.pem",
            "keyfile": "/home/pi/mywattsmon-data/cert/bosch/shckey.pem"
        },
        "units":{
            "WASHER":{"uid":"hdm:ZigBee:3425b4fffe…"},
            "COOLER":{"uid":"hdm:ZigBee:70ac08fffe…"},
            "EHOOD":{"uid":"hdm:ZigBee:f4b3b1fffe…"},
            "HEATER1":{"uid":"hdm:ZigBee:f082c0fffe…"},
            "HEATER2":{"uid": "hdm:ZigBee:6c5cb1fffe…"}
        }
    }
}
```

3.2 Eigene Geräteklassen

Eigene Geräteklassen müssen den vorausgehend dokumentierten Standards folgen und können dem Muster der mitgelieferten Klassen entsprechend implementiert und konfiguriert werden. Da der Monitor die Geräteklassen dynamisch lädt, sollten die eigenen Klassen dann direkt anwendbar sein.

Die eigenen Geräteklassen sind im Datenverzeichnis unter `/mywattsmon-data/py` vorzuhalten. Die Dateien werden beim Start der Applikation von dort aus nach `/mywattsmon/app/device/user/` kopiert und sind entsprechend als Modul `mywattsmon.app.device.user.<module name>` zu konfigurieren.

4. FAQ

1. Wie sieht die Verzeichnisstruktur der Applikation aus?

Die folgende Tabelle informiert über die Verzeichnisstruktur.

Hinweis: Jede Python-Datei repräsentiert ein Modul mit jeweils einer Klasse. Beispielsweise repräsentiert die Datei `/mywattsmon/app/monitor.py` das Modul `mywattsmon.app.monitor` mit der Klasse `Monitor`.

Verzeichnis	Inhalt
`mywattsmon`	Applikationsverzeichnis.
`/app`	Entry-Point-Klassen `Monitor` und `Window`.
`/app/device`	Geräteklassen.
`/app/device/user`	Kopierte eigene Geräteklassen, sofern im Datenverzeichnis vorhanden.
`/app/helper`	Hilfsklassen.
`/doc`	Dokumentation im PDF-Format, Readme-Datei, Lizenzdatei, Beispiel-Konfigurationsdateien.
`/test`	Testklasse (abgeleitet von `unittest.TestCase`).
`/test/data`	Testdaten (wird automatisch erstellt).
`mywattsmon-data`	Datenverzeichnis mit der Konfigurationsdatei `config.json`. Wird automatisch parallel zum Applikationsverzeichnis erstellt.
`/db`	Datenbank.
`/log`	Logausgaben.
`/py`	Eigene Geräteklassen.

2. Auf welchen Plattformen läuft die Applikation?

Prinzipiell auf allen Plattformen, auf denen Python 3.11 oder höher läuft. Getestet wurde mit Python 3.11 und 3.12 unter Linux und Windows.

3. Funktioniert der Monitor auch ohne das Monitorfenster?

Ja, das Monitorfenster ist optional und läuft in einem eigenen Prozess. Die Kommunikation mit dem Monitorprozess erfolgt via Socket (lokaler TCP Port).

4. Kann die Applikation mehrfach parallel installiert und ausgeführt werden?

Ja. Es ist allerdings darauf zu achten, die Applikation unter verschiedenen Verzeichnissen zu installieren und aufzurufen, beispielsweise unter `/home/u1/`**`mywm1`**`/mywattsmon` und `/home/u1/`**`mywm2`**`/mywattsmon`. Außerdem müssen unterschiedliche Portnummern konfiguriert werden, beispielsweise `42011` und `42012`.

5. Kann beim Aufruf des Monitors oder des Monitorfensters ein alternatives Datenverzeichnis angegeben werden?

Ja. Dazu ist beim Aufruf die Option `-d` oder `--datapath` zu verwenden. Es ist darauf zu achten, bei beiden Aufrufen dasselbe Datenverzeichnis anzugeben.

6. Können mehrere Datenbanken und/oder Datenbanktabellen konfiguriert werden?

Nein. Es werden immer die Datenbank `/mywattsmon-data/db/monitor.db` und die konfigurierte Datenbanktabelle verwendet. Wird die Datenbankkonfiguration geändert, so muss die Datenbank zuvor durch Kopieren gesichert und dann entfernt werden. Sie wird dann automatisch mit der aktuell konfigurierten Datenbanktabelle neu erstellt.

7. Wie können die Daten aus der Datenbank gelesen werden?

Es handelt sich um eine SQLite-Datenbank, die mit jedem kompatiblen SQL-Client gelesen werden kann. Der in der konfigurierten Datenbanktabelle automatisch erstellte Zeitstempel erlaubt eine zeitliche Eingrenzung beim Selektieren der Datensätze, und mit den numerischen Datenfeldern sind via SQL Berechnungen durchführbar. So können sehr einfach individuelle Statistiken generiert werden. Falls eine solche Auswertung der Daten vorgesehen ist, sollten regelmäßig Kopien von `/mywattsmon-data/db/monitor.db` erstellt werden, um die Daten zu sichern.

Zur Veranschaulichung der Datenauswertung hier einige Abfragen mit dem SQLite CLI-Tool:

```
$ sqlite3 "/home/u1/mywattsmon-data/db/monitor.db"
sqlite> select max(id) from kwh;
280
sqlite> select id, ts, em from kwh where id = 280;
280|2024-10-06 09:00|1794.7
sqlite> select round(max(em)-min(em),1) from kwh where ts like '2024-09-25%';
2.1
sqlite> select round(max(em)-min(em),1) from kwh where ts >= '2024-09-23' and
ts <= '2024-09-29';
11.4
sqlite> select round(max(em)-min(em),1) from kwh where ts like '2024-09%';
34.3
sqlite> .quit
```

8. Können die mitgelieferten produktiven Geräteklassen direkt verwendet werden?

Die produktiven Geräteklassen erfordern jeweils ein physisches Gerät wie beispielsweise Energiezähler oder Schaltsteckdose. Außerdem können diese Klassen Python-Abhängigkeiten haben, die eine zusätzliche Installation erfordern (beispielsweise `requests`, `minimalmodbus` oder `fritzconnection`. Siehe dazu den Quellcode der Klassen unter `/mywattsmon/app/device/`). Wenn die Voraussetzungen gegeben sind, kann die Konfiguration entsprechend angepasst und die Geräteklasse verwendet werden.

9. Was passiert mit den eigenen Daten und eigenen Geräteklassen beim Update der Applikation?

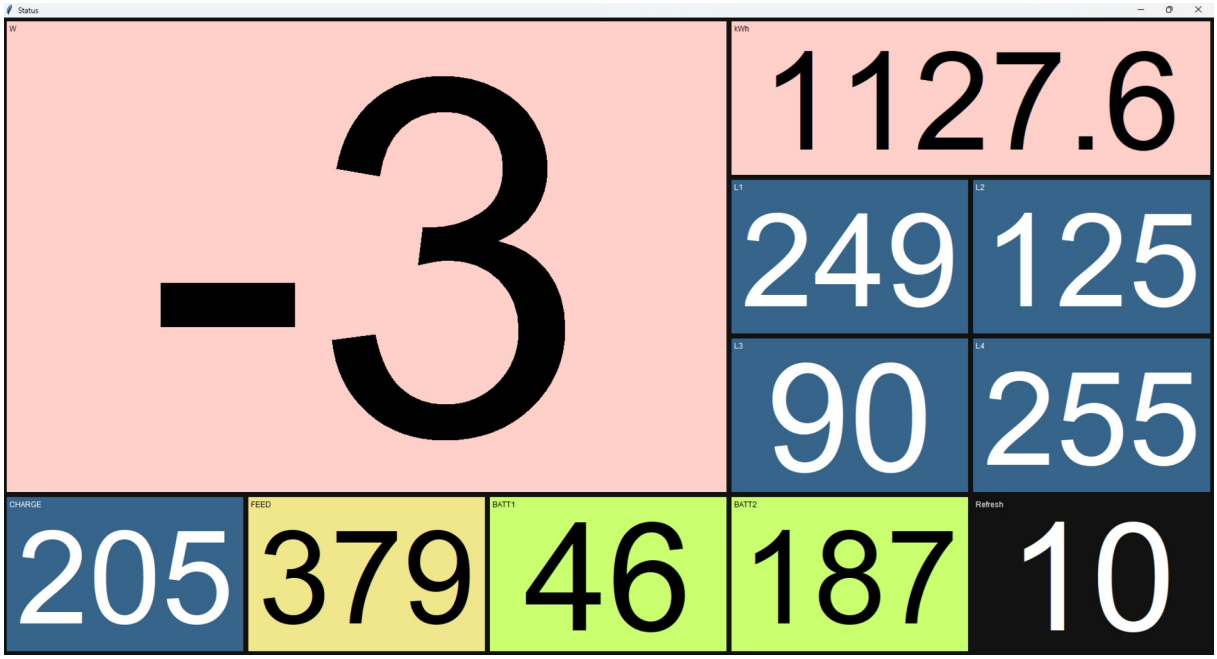
Ein Update mit `python -m pip install mywattsmon -U -t <Zielverzeichnis>` aktualisiert das gesamte Applikationsverzeichnis mit allen Inhalten. Das standardmäßig parallel zum Applikationsverzeichnis installierte Datenverzeichnis `/mywattsmon-data` ist davon nicht betroffen und wird auch beim nächsten Start der Applikation nicht überschrieben.

Die Geräteklassen, die sich im Daten-Unterverzeichnis `/mywattsmon-data/py` befinden, werden beim Applikationsstart nach `/mywattsmon/app/device/user` kopiert und dynamisch geladen. Daher sind sie auch nach einem Update der Applikation wieder anwendbar.

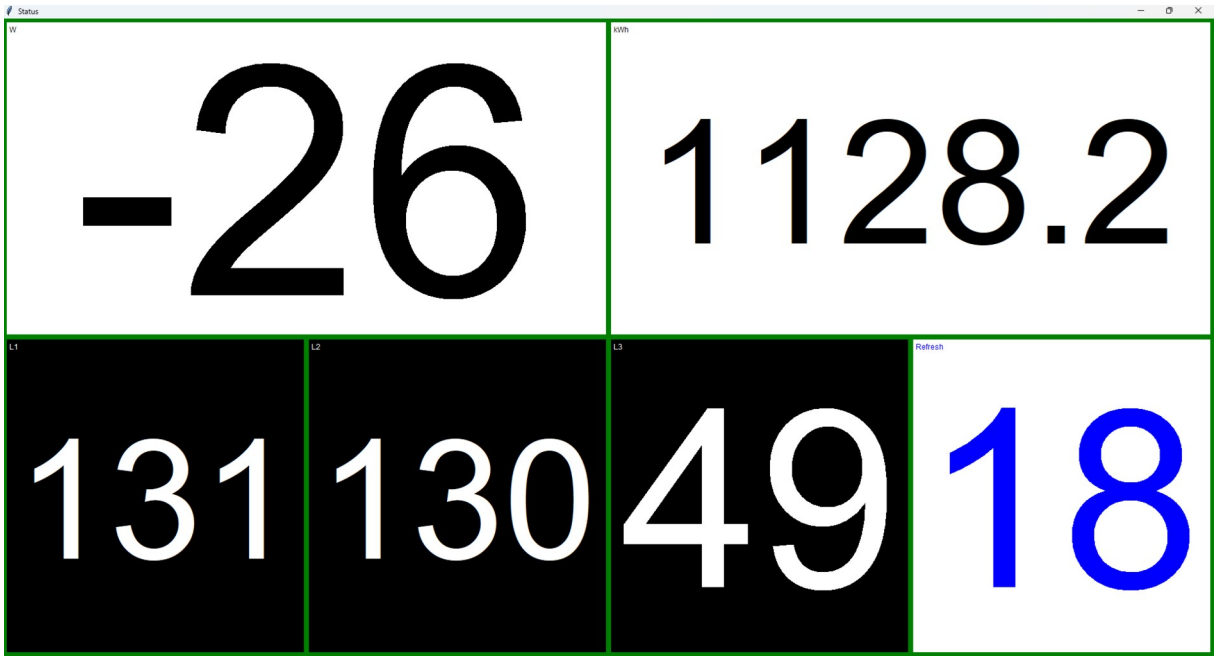
10. Welche Konfigurationsbeispiele werden mitgeliefert?

Eine vollständige Konfigurationsdatei mit allen mitgelieferten produktiven Geräteklassen, sowie Beispieldateien, in denen die Klasse `Mock` sowie eine Beispielklasse `UserMock` konfiguriert sind, stehen unter `/mywattsmon/doc/` zur Verfügung. Private Attribute wie Kennwörter etc. sind darin unkenntlich gemacht.

Fensteransicht mit der Variante `sample_mock_2_config.json`:



Fensteransicht mit der Variante `sample_usermock_config.json`:



Hinweis: Die in dieser Variante konfigurierte Beispiel-Geräteklasse ist in der Datei `usermock.py` codiert, die sich ebenfalls unter `/mywattsmon/doc/` befindet.

11. Was ist der Unterschied zwischen Gerät und Geräteeinheit?

Als Gerät wird in diesem Kontext ein physisches Gerät wie beispielsweise eine Fritzbox, ein Bosch Smart Home Controller oder ein Victron Energy Charge Controller bezeichnet. Eine Geräteklasse repräsentiert ein physisches Gerät. Eine Geräteeinheit ist beispielsweise eine Schaltsteckdose, die per DECT mit der Fritzbox oder per ZigBee mit dem Bosch Controller verbunden ist. Beim Victron Energy Charge Controller sind Geräteeinheit und Gerät praktisch gleichzusetzen.

12. Welcher Wert wird ausgegeben, wenn in der Fenstergitter- oder Datenbankkonfiguration bei `units` mehrere Geräteeinheitennamen angegeben werden?

Die Werte der angegebenen Geräteeinheiten werden addiert (`power` in Watt, `energy` in Kilowattstunden).

13. Was bedeuten Sonderzeichen oder Buchstaben im Fenstergitter?

Wird in einem Gitterelement keine Zahl, sondern ein Sonderzeichen oder ein Buchstabe dargestellt, so haben diese folgende Bedeutung:

Zeichen	Bedeutung
–	Kein Wert (None/null)
~	Die Geräteeinheit ist ausgeschaltet
v	Ungültiger Wert
w	Warnung (siehe Log)
x	Fehler (siehe Log)

14. Kann auf den Monitor auch remote zugegriffen werden?

Das ist nicht vorgesehen. Prinzipiell wäre das zwar möglich, aber die Applikation ist für den lokalen Betrieb ausgelegt.