06.11.2023

06.11.2023

Einleitung

Es ist das Ziel, GPIO-Header eines Raspberry-Pi-Bastelrechners flexibel und einfach steuern zu können. Die Applikation gliedert sich in zwei Ebenen, die physisch voneinander getrennt sind und über das WebSocket-Protokoll kommunizieren. Dabei handelt es sich um eine Server- und um eine oder mehrere Client-Instanzen - in variabler Ausführung. Der Server wird in Python programmiert, ganz im Gegensatz zum Client. Dort kann es sich um eine simple JavaScript- oder eine Umsetzung in einer anderen nativen App-Sprache handeln. Auch die Verwendung von Schnittstellen ist denkbar.

Dieses Projekt resultiert aus den Applikationen jsPI und pico. Bei beiden Projekten wurde auf node.js und Python als Web- und WebSocket-Server sowie Javascript als fixer Client gesetzt. Beide Applikations-Ebenen waren strikt voneinander abhängig und laufzeitabhängig.
Die Verwendung basierte auf den Bibliotheken socket.io für Python und Javascript.

Auf die Verwendung von socket.io für Server- und Client wird nun verzichtet. Dies erhöht den Aufwand für Client-Applikationen, hält aber den Datentraffic während der Kommunikation überschau- und kontrollierbar. Für die Verwendung von Clients ergeben sich aber viel flexiblere Lösungsansätze. Die Kommunikation erfolgt über eine simple, pfadbasierte Abfrage und orientiert sich sehr frei am REST-API-Prinzip - bei dem Resultate immer einen gültigen Endpunkt darstellen. In diesem Anfragen befinden sich neben dem Pfad weitere Anweisungen, wie mit angefragten Daten verfahren werden soll.

Im Gegensatz zur REST-API, wird hier nicht unterschieden, welcher Anfragetyp in Verwendung ist, und ebenso, dass kein HTTP-Status zurückgegeben wird. Die Rückgabe erfolgt immer in Form von Daten oder NULL.

Bei der Umsetzung der Datenspeicherung soll das bisherige JSON abgelöst werden. An dessen Stelle tritt eine sqlite3-Datenbank mit vorbereiteten Datentabellen.. Die Umgebung ist im Lieferumfang von Python3 dabei. Die Abfrage und Speicherung der Daten erfolgt über ein konformes SQL-Interface mit DB-API 2.0.

06.11.2023

1. Aufgaben

1.1 Laden der Standardkonfigurationen.
1.2 Laden und Speichern von Daten.
1.3 Anfragen an den Speicher

Angaben zu der Aufgabenstellung

1.1. Laden der Standardkonfiguration

Je nach Konfiguration kann ein Raspberry-Pi-Header-Schema geladen werden, das zum Beispiel zu einem Raspberry Pi 2, 3, 3+, 4 oder 5 passt. Auf Basis der gewünschten Pin-Belegung können benutzerspezifische Pins konfiguriert werden. In diesem Bereich des Speichers kann nicht geschrieben werden.
Beim aller ersten Laden der Applikation, wird dieses Schema in die Tabelle **pins** geschrieben und kann dort konfiguriert werden.

1.2. Anfragen an den Server

Bereiche des Speicher lassen sich gezielt adressieren, abfragen und ändern. Dies geschieht durch Subroutinen (SQL), die durch Keywords (in der Anfrage) getriggert werden, wenn bestimmte Formulierungen in der Anfrage enthalten sind. Damit wird auch sichergestellt, dass unerlaubte Anfragen nicht möglich sind.

Hinweis: Werden konfigurierte Pins oder Aktoren geändert, wird immer das Feld *gpio* oder *id* abgefragt. Damit wird sichergestellt, dass keine Pins oder Aktoren doppelt angelegt werden. Diese Prüfung ist durch das Datenmodell festgelegt.

2. System vorbereiten

2.1 Raspberry Pi

Für die Umgebung des Raspberry Pi sind einige notwendige Anpassungen zu treffen. Viele Standardbibliotheken altern aus den Images heraus und müssen ggfs. nachinstalliert werden.

2.1.1 Ab Rasbian 10 Buster (mit 5.10 Kernel) wird das Paket **gpiod** nicht von Haus aus mitgeliefert. Dieses Paket wird für den Zugriff auf GPIOs benötigt.
Manuelle Installation: → `sudo apt install gpiod`

2.2 Python

Für die Ausführung des Programms ist Python ab Version 3.9.x notwendig.

2.2.1 Python-Version ermitteln

→ `python -version`

2.2.2 Für die weitere Installation ist der Paketmanager pip notwendig

a. → `sudo apt update && sudo apt install pip`

b. → `sudo -H pip install --upgrade pip`

2.3 Python-Pakete installieren

Zur Ausführung des Programms sind folgende Pakete notwendig:

2.3.1 websockets

→ `pip install websockets`

WebLink: https://pypi.org/project/websockets/

2.3.2 W1Bus-Sensoren einrichten

a. → `sudo raspi-config` → Interface Options → 1-Wire aktivieren

b. Boot-Konfiguration anpassen

i. → `sudo nano /boot/config.txt`

ii. Eintrag suchen: **dtoverlay=w1-gpio** und folgenden Zeichenkette kommasepariert hinten anfügen: **gpiopin=4**

c. Python-Bibliothek installieren:

→ `pip install w1thermsensor`

WebLink: https://pypi.org/project/w1thermsensor/

2.3.3 ADAFruit-DHT-Sensoren einrichten

a. → `pip install adafruit-circuitpython-dht`

WebLink: https://pypi.org/project/adafruit-circuitpython-dht/

2.3.4 Zusammenfassung aller notwendigen Bibliotheks-Installationen in ein Skript

a. .... (Ticket #104)

2.4 Git installieren und Repo beziehen

Bei einer frischen Installation des Raspberrys ist eine aktuelle Git-Version notwendig.

2.4.1 Installation von Git auf einem Raspberry Pi

→ `sudo apt install git`

2.4.2 Ordner im Home-Verzeichnis anlegen

→ `mkdir /home/<USER>/<FOLDER>/`

2.4.3 Arbeitskopie beziehen

a. `git init /home/<USER>/<FOLDER>/`

b. `cd /home/<USER>/<FOLDER>/`

c. `git remote add - origin https://<TOKEN>@github.com/turbohirn/pinlink.git`
d. `git pull origin main`

2.5 Systemstart einrichten

Zum Start des Programms kann der System-Cron verwendet werden.

2.5.1 Crontab einrichten

→ `crontab -e`

2.5.2 Beispiel des Programmstarts nach Neustart des Hostsystems:

→ `@reboot sleep 30 && cd /home/<USER>/<FOLDER>/<REPO>/ && git pull origin main && bash start.sh`

2.6 Projekt

Die Gliederung der Aufgaben und Fehlermeldungen erfolgt in diesem Github-Projekt → https://github.com/users/turbohirn/projects/5

Der Server

1. Allgemeine Funktionsweise

Der Python-Server stellt die Datenbasis, die Kommunikation und den Hardware-Link zur Verfügung.

2. Datenbasis mit sqlite3

Wie bereits erwähnt, wird das sqlite3-Paket der aktuellen Python-Installationen verwendet. Vorteile sind hier:

- Abfragen über das SQL-Interface,
- Datenhaltung im RAM, während der Laufzeit,
- Einfaches Backup des Datenbestandes,
- host-übergreifende SQL-Abfragen,
- Viewer-AddOns in der Entwicklungsumgebung,

Python-Module → classes.dbservice.py

2.1 Daten zur Initialisierung

Die Key-/Value-Vorlagen der Tabellen und SQL-Statements beim Erststart befinden sich im Python-Modul → classes.tools.base. Jede Tabelle verfügt über jeweils ein Timestamp-Feld (dtime).
Die Daten zur Initialisierung der GPIO-Headers befinden sich im Unterobjekt TABLES im Objekt PRECONF, Übersicht der Listenobjekte im globalen Speicher des Servers:

- ListenObject HEADERS:
  Diese Liste wird zum Start des Servers in die Tabelle pins geschrieben, wenn die Datenbank noch nicht existiert und/oder die nicht Tabellen vorhanden sind. Diese Daten dienen zur Darstellung des Headers im Client und als rein les- und aktualisierbare Vorlage für die weitere Pin-Konfiguration. Delete- und Insert-Operationen werden nicht vorgenommen.
  - Das Feld **gpio** verfügt über den Wert NULL, wenn weder GPIO.IN- oder GPIO.OUT-Status vergeben werden kann. Das betrifft VCC-, Ground- und BUS-Pins.
  - Die Feld **mode** erhält im Standard den Wert -1, und signalisiert einen deaktivierten Zustand. Der Wert 0 steht für den Werttyp GPIO.OUT. Werte ab 1 und größer stehen für den Werttyp GPIO.IN. Weitere Ziffern definieren die Sensortypen im Detail.
    - Festgelegte Modustypen:
      - 1 = Taster ON/OFF
      - 2 = W1Bus
      - 3 = ADAFRUIT DHT 11
      - 4 = ADAFRUIT DHT 22
      - 5 = ADAFRUIT AM2302
  - → Datentabelle pins

- Listenobjekt COLORS:
  Diese Liste dient zur Unterstützung der UI als Farbindikator und wird beim Start in die Tabelle colors geschrieben (ohne Vorlage; Daten lesbar). Statements aus der Tabelle pins werden mit colors verknüpft. Diese Tabelle ist ansonsten funktionslos und gibt das Farbschema der GPIO-Header-Darstellung an.
  - → Datentabelle colors
- Tabelle actors (ohne Vorlage; Daten les- und schreibbar)
  In dieser Tabelle werden konfigurierte Aktoren zu Wert- und Zeitsteuerungen gespeichert.
  - → Datentabelle actors

- Tabelle sensors (ohne Vorlage; Daten les- und schreibbar)
  Sensoren wie Temperatur, Luftfeuchte gespeichert, die mit der Tabelle actors verlinkt sind.
  - → Datentabelle sensors
- Tabelle timers (ohne Vorlage; Daten les- und schreibbar)
  Hier können multiple Start- und Endzeiten für konfigurierte Aktoren in Tabelle actors angelegt werden.
  - → Datentabelle timers
- Tabelle weekdays (ohne Vorlage; Daten nur lesbar)
  Verwendet für Angaben der Wochentage in der Tabelle timers, bei den Zeitsteuerungstypen 1 und 2.
  - → Datentabelle weekdays

3. Datentabellen

3.1 Datenfelder pins

Name	Beschreibung	Datentyp
pin	Pin-Bezeichner für Header-Pin	INTEGER
gpio	GPIO-Bezeichner für Header-Pin	INTEGER
mode	Modus des Pins, INACTIVE = -1 OUT = 0 IN > 0, Weitere Ziffern definieren den Sensortyp	INTEGER
state	Schaltzustand des Pins, 0 = an 1 = aus	INTEGER
input	Art des Sensor- oder Eingabetyps, relevant für Sensortypen 0 = inaktiv 1 = An/Aus → OnOff 2 = Thermischer Sensor → W1Bus 3,4 = Thermischer Sensor → DHT11(3) bzw. 22(4) oder AM2302	INTEGER
name	Bezeichner für Pin/GPIO	VARCHAR
label	Zusätzliches Label für weitere Funktionsweise	VARCHAR
description	Beschreibung des Pins	VARCHAR
actor_id	PRIMARY KEY des Aktors aus den Tabellen actors, timers oder sensors,	INTEGER

Name	Beschreibung	Datentyp
	Damit kann ein Aktor multiple Pins steuern (1:n)	
color_id	ID der Tabelle colors	INTEGER
dtime	Zeitstempel	TIMESTAMP

3.2 Datenfelder colors

Name	Beschreibung	Datentyp
id	Schlüssel/Identifier	INTEGER
color	Farbangabe via HEX-Code	VARCHAR
img_id	DOM-Id für SVG-Icon o.ä.	VARCHAR
dtime	Zeitstempel	TIMESTAMP

3.3 Datenfelder actors

Name	Beschreibung	Datentyp
id	Schlüssel/Identifier	INTEGER
name	Bezeichner.	VARCHAR (48)
color_id	Farbangabe via ID aus Tabelle colors	INTEGER
type	Bezeichner-ID, 1: Feste stündliche Anfangs- und Endzeit pro Wochentag, 2: Interval (Stunde/Minute) pro Wochentag und mit begrenzbarer Uhrzeit, 3: Fest definierte Zeiträume, 4: Countdown in Stunde/Minuten/Sekunden ab gespeicherten Zeitpunkt aktiv	INTEGER
active	Aktivierungs-Flag, Vorher (sensordata/active)	NUMERIC
allow_handle	Flag erlaubt das manuelle Schalten des verknüpften GPIOs	NUMERIC
dtime	Zeitstempel	TIMESTAMP

3.4 Datenfelder sensors

Name	Beschreibung	Datentyp
id	Schlüssel/Identifier	INTEGER
actor_id	PRIMARY KEY des Actors.	INTEGER
name	Bezeichner für diese Wertkonfiguration	VARCHAR

Name	Beschreibung	Datentyp
temp_low	Temperatur, Tiefster erlaubter Wert	INTEGER
temp_high	Temperatur, Höchster erlaubter Wert	INTEGER
temp_treshold	Temperatur, Schaltungsschwellwert	INTEGER
humi_low	Luftfeuchte,Tiefster erlaubter Wert	INTEGER
humi_high	Luftfeuchte, Höchster erlaubter Wert	INTEGER
humi_treshold	Luftfeuchte, Schaltungsschwellwert	INTEGER
dtime	Zeitstempel	TIMESTAMP

3.5 Datenfelder timers

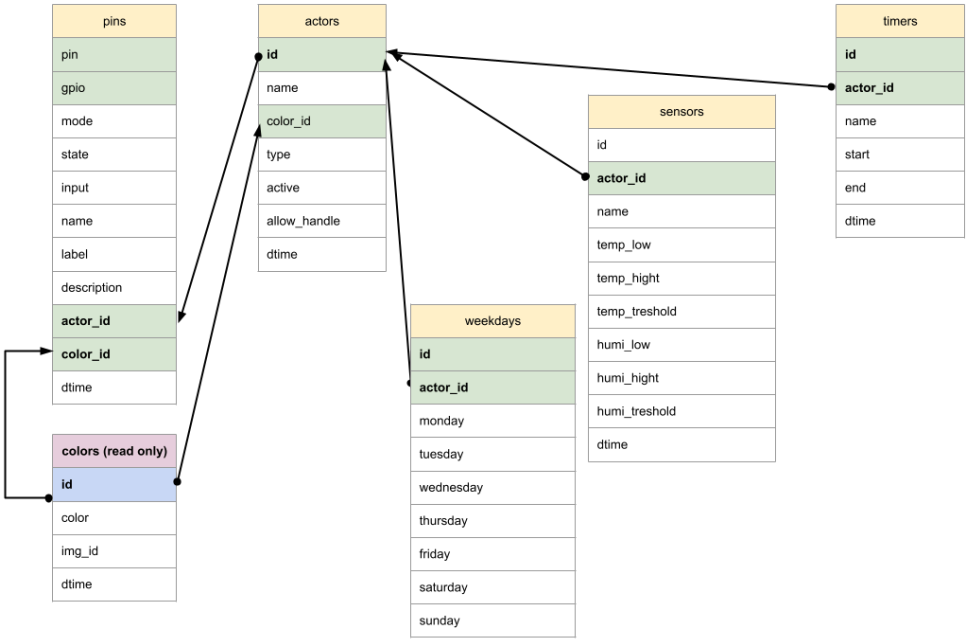
Name	Beschreibung	Datentyp
id	Schlüssel/Identifier,	INTEGER
actor_id	Link zu ID in Tabelle StoreTimerActors	INTEGER
name	Bezeichner für diese Werte Konfiguration	VARCHAR
start	Startzeitpunkt, Type 1 und 3: nur Angabe zu Stunde aus Datetime verwenden,	DATETIME
end	Endzeitpunkt, Type 1 und 3: nur Angabe zu Stunde aus Datetime verwenden, Type 4: Countdown Ende,	DATETIME
dtime	Zeitstempel	TIMESTAMP

3.6 Datenfelder weekdays

Name	Beschreibung	Datentyp
id	Schlüssel/Identifier,	INTEGER
actor_id	Bezeichner-ID für Wochentagszuordnung in StoreTimerActors	INTEGER
monday	Flag ob aktiviert/deaktiviert	INTEGER
tuesday	Flag ob aktiviert/deaktiviert	INTEGER
wednesday	Flag ob aktiviert/deaktiviert	INTEGER
thursday	Flag ob aktiviert/deaktiviert	INTEGER
friday	Flag ob aktiviert/deaktiviert	INTEGER
saturday	Flag ob aktiviert/deaktiviert	INTEGER

Name	Beschreibung	Datentyp
sunday	Flag ob aktiviert/deaktiviert	INTEGER

3.7 Schematisches Schaubild des Datenmodells:



4. Kommunikation via Socket-Server

Dieses Script startet zur Laufzeit eine Reihe von Threads, die während Statusänderungen, Daten austauschen und speichern können. Datenoperationen finden in der bereits erwähnten sqlite3-Umgebung statt und bleiben interner Bestandteil der Python-Programmierung.

1. Initialer Start des Servers:
    a Im Konfigurationsverzeichnis wird die Datei pinlink.ini angelegt.
    b Im RAM des Hosts wird eine Datenbank (sqlite3) initiiert, wenn eine neue Datenbankverbindung aufgebaut wird,
    c Alle notwendigen Tabellen werden in der Datenbank angelegt,
    d Die Stammdaten der Header-Konfiguration werden, in die entsprechenden Tabellen **pins** und **colors**, eingefügt,
2. Neustart des Server:
    a Laden des sqlite3-Backups (pinlink.sqlite) in den RAM,

   b Starten der Program-Threads
3. Beenden des Servers:
   (Kontrolliertes Beenden via Keyboard oder Script)
   a Die Datenbank wird aus dem RAM in das Backup-File pinlink.sqlite geklont und gespeichert,
   b Die Program-Threads zur GPIO-Steuerung und Datenverarbeitung werden gestoppt,
4. Python-Konfiguration

```
[GLOBAL]
db = server/config/pinlink.sqlite
port = 8765
ssl = False
pintime = 0.5
pem = server/ssl/server.pem
demojs = client/http/base.js
excels = gpio, pin, name, label
extabs = pins
```

5. SSL

https://www.ipswitch.com/de/blog/wie-man-mit-openssl-zertifikat-generiert

6. Klassendiagramm

Abfrageschema des GPIO-Headers

1. Allgemeines

Eine gültige GPIO-Header-Query untergliedert sich in Type, Statement-Key, Where-Klauseln und Wertepaare. Bei einer gültigen Abfrage wird folgendes Schema angewandt:

1. Einleitendes Statement (select, insert, update, delete),
2. Angabe des Keys für eine gültige gespeicherte Anfrage,
    a. Der Wert eines gültigen Keys ist immer ein SQL-Statement mit Platzhaltern für String-Werte,
3. Formulierung der Suchbegriffe wie in WHERE-Klauseln von SQL-Statements,
4. Übergabe eines Key-/Value-Paares bei Statement-Typen wie INSERT und UPDATE

Näheres ist in den Beispielen aufgeführt.

2. Beispiele für GPIO-Queries

SELECT		
Beschreibung	**Abfrage**	**AppKey**
Gibt die Tabellen **pins** und **color** vollständig zurück.	/select/check	base.select.check
Gibt alle Einträge der Tabelle **pins** zurück.	/select/pins	base.select.pins
Gibt alle Einträge der Tabelle **pins** zurück, die konfigurierbare GPIOs erlaubt und folgende Bedingung erfüllt: **gpio>0**	/select/gpios	base.select.gpios
Findet Einträge in Tabelle **pins** welche für die Verwendung als GPIO-Header konfiguriert wurden. **Hinweis**: dieses Statement ist für den Programmstart und ohne Parameter.	/select/initialize	base.select.initialize
Findet Einträge in Tabelle **pins** welche für die Verwendung als GPIO-Header im Modus **IN** konfiguriert wurden. Sind Sensoren verbunden, werden die Daten ausgegeben.	/select/gpioin	base.select.gpioin
Findet Einträge in Tabelle **pins**.	/select/pin|pin=2 or mode>-1	base.select.pin
Findet alle Einträge in Tabelle **actors**	/select/actors	base.select.actors
Findet Einträge in Tabelle **sensors**.	/select/sensors	base.select.sensors
Findet Einträge in Tabelle **timers**.	/select/timers	base.select.timers
Findet Einträge in Tabelle **actors** mit Operator.	/select/actor|id=2 or id=3	base.select.actor
Findet Einträge in Tabelle **pins** mit schaltbaren GPIOs.	/select/switch	base.select.switch

INSERT		
Beschreibung	**Abfrage**	**AppKey**
Schreibt Einträge in die Tabelle actors	/insert/actors||name=Aktorname, color_id=#db3000, type=2	base.insert.actors
Schreibt Einträge in die Tabelle sensors	/insert/sensors||name=Sensorname, actor_id=2	base.insert.sensors
Schreibt Einträge in die Tabelle timers	/insert/timers||name=Timer name, actor_id=2	base.insert.timers

UPDATE		
Beschreibung	**Abfrage**	**AppKey**
Setzt Pins in den schaltbaren Zustand. **modus=0**	/update/pins|gpio=17 or gpio=22|mode=0	base.update.pins
Setzt Pins in den lesbaren Zustand. **modus=1** INPUT 1 = OnOff	/update/pins|gpio=10|mode=1, input=1	base.update.actors
Setzt Pins in den lesbaren Zustand. **modus=1** INPUT 2 = W1Bus	/update/pins|gpio=4|mode=1, input=2	base.update.actors
Setzt Pins in den lesbaren Zustand. **modus=1** INPUT 3/4 = Adafruit DHT	/update/pins|gpio=2 or gpio=3|mode=1, input=3	base.update.actors
Ändert Einträge in Tabelle **actors**.	/update/actors|id=2|name=Aktors, color_id=#000000, type=2	base.update.actors
Ändert den Modus (0 oder 1) von	/update/switch|gpio=22|	base.update.switch

UPDATE		
Beschreibung	**Abfrage**	**AppKey**
spezifierten PIN oder GPIO. Das State-Value **muss** angegeben werden. **Achtung**: Dieser Befehl ist nur möglich, wenn input < 1 ist.	state=0	
Ändert den Modus (0 oder 1) von spezifierten PIN oder GPIO. Der Modus wird nicht explizit angegeben, um die Toggle-Value kümmert sich das Statement. **Achtung**: Dieser Befehl ist nur möglich, wenn input < 1 ist.	/update/toggle|gpio=22 or gpio=27 or gpio=17	base.update.toggle

DELETE		
Beschreibung	**Abfrage**	**AppKey**
Löscht Einträge aus Tabelle **actors.**	/delete/actors|id=5	base.delete.actors
Löscht Einträge aus Tabelle **timers**.	/delete/timer|id=5 or id=4	base.delete.timers