Maison Jung IOT RaspberryPi User Guide

Two programs run on the home RaspberryPi:

- The PHP web server: serving the website through files stored in `/var/www/`
- The Python server called “maison-jung”: used to take actions on Wemos based microcontrollers on external stimuli (telegram, adafruit, website, ...).
  The python package project code base for the Maison Jung IOT server can be found at: https://github.com/gruvw/maison-jung .

The Python server requires multiple files to be inside a working directory in order to run properly:

- `config.yml`: where are stored API keys (for the telegram bot and Adafruit IO), all the Wemos IPs, etc.
- `schedules.yml`: defines the scheduled tasks.
- `options.yml`: used to build the interactive inline keyboard menu on telegram.
- `database.json`: TinyDB database.

All those files can be found in the working directory of the Python server: on the RaspberryPi at `/var/www/prog/prod/server`.

Arduino Programs

Additionally, all the Arduino programs used can be found on the RaspberryPi at `/var/prog/prod/Arduino/...`.

They are not run by the RaspberryPi, but they are stored there for ease of access and use (centralized single source of truth).

YAML primer

This section tries to explain the basis of YAML, the language used to configure the Python server (`.yml` files).

YAML (stands for: YAML Ain't Markup Language) is a human-readable data serialization standard that is commonly used for configuration files and data exchange. It is designed to be easy to read and write.

Main features of YAML:

- **Format**: Key-value store, meaning every entry is denoted by its key.
- **Hierarchical**: Uses indentation (2 spaces) to denote structure.
- **Readable**: Aimed to be easy for humans to read and write.
- **Flexible**: Supports complex data structures as entries like lists and dictionaries.

Terminology:

- Document/Object: list of entries.
- Entry: a key and a value.
- Key: some text/label to identify a particular value.
- Value: data stored behind a key. It can be some text (either implicitly delimited or explicitly by `"text"` or `'text'`), boolean (`true` or `false`), number, list, object.

Basic file syntax by commented example:

```yaml
# comment, will be ignored

# document start marker
---

# define an entry with a key and a value
key: value

# nesting: a key can hold an object/document
# it references multiple key-value entries
parent:
  child1: value
  child2: value
  enabled: true # boolean value

# the value can be a list
list:
  - item1
  - item2

# list of objects
persons:
  # first object
  - name: John Doe # string value
    age: 30 # integer value
  # second object
  - name: "Jane Smith" # explicitly delimited
    age: 25
  # inline object
  - {name: Tom, age: 21}
```

Scheduler

This section describes how to edit the `schedules.yml` file properly.

The configuration file for the scheduler is split in **sections**:

- `every` : will be executed every day.
- `weekdays` : will be executed from Monday to Friday (both included).
- `weekend` : will be executed Saturdays and Sundays.
- `monday` : will be executed only on Mondays.
- `tuesday` : will be executed only on Tuesdays.
- ...
- `sunday` : will be executed only on Sundays.

Each section contains an object for every **target activity** (lampes, stores, arrosage):

```
section:
  lampes:
  stores:
  arrosage:
```

Each target activity contains a list (possibly empty) of **scheduler instructions** (on a single line):

```
- {time: '19:25', data: 'XAXXAXXXX', enabled: true, id:
my_id}
```

A scheduler instruction is an object that contains the following keys:

- `time` : the 24-hour format time of execution of the entry. Additionally the time can be specified relatively to the sunrise/sunset time of the day: examples `"sunrise/+30"` (30 minutes after sunrise), `"sunset/-20"` (20 minutes before sunset).
- `data` : the feed data that will be set for the activity when the time/date is met.
- `enabled` : boolean flag, `true` means this entry will be executed, `false` means it is currently disabled (and will be ignored). It might come in handy when keeping disabled schedules so one doesn't have to write all the entries when they go in vacation for example (you just enable the ones that you want directly from the config).

- `id` (optional): a text value that specifies that two scheduler instructions are related to each other (often meaning a duration). For example to turn on a light from 8:00 to 9:00 one will need two scheduler instructions with the same id (one to turn on at 8:00 and one to turn off at 9:00). Any single id should not be used more than twice in the whole scheduler config file. You can omit the id field for one-shot instructions (like turn off all lamps).

Note: when the `schedules.yml` file is uploaded, or modified and saved on the RaspberryPi, the Python server will automatically detect and apply all modifications of the schedule with the updated instructions.

Feeds data

Specifications of the feeds data part of the scheduler.

- **Lampes**: 9 characters string, one character for each lamp in the correct order (first character for first lamp, last character for last lamp). Each character can be either “X” (do nothing on this lamp), “A” (turn this lamp on), “Z” (turn this lamp off).
- **Stores**: 6 characters string, grouped by 2 (so 3 groups). The first group of two characters is controlling the blinds of the second floor in the house. The second group of two characters is for the first floor of the house. The last group is for the bureau (not yet connected). Each group is composed of two characters. The first character of a group is a number that specifies which blind to control on the related remote (0 means all, 1 only the first blind, ...). The second character is the action to apply on the selected blind: “A” for open, “Z” for close, “C” for clac-clac, “S” for stop. One can put “XX” for a group if they don’t want to control the blinds of this group.
- **Arrosage**: 3 characters string. The first two characters represent the number of the valve to control. The last character can be “A” for open, “Z” for close.

Leave every empty target in the scheduler config. Only add or remove scheduler instructions. An empty scheduler file looks like the following:

```
---
every:
  lampes:
  stores:
  arrosage:
weekdays:
  lampes:
  stores:
  arrosage:
weekend:
  lampes:
  stores:
  arrosage:
monday:
  lampes:
  stores:
  arrosage:
tuesday:
  lampes:
  stores:
  arrosage:
...
sunday:
  lampes:
  stores:
  arrosage:
```

Examples

Every Monday, turn on lamp 4 from 10:00 to 13:00.

```
monday:
  lampes:
    - {time: '10:00', data: 'XXXAXXXXX', enabled: true, id:
matin01}
     - {time: '13:00', data: 'XXXZXXXXX', enabled: true,
id: matin01}
```

Every day, turn off every lamp one hour after sunset.

```
every:
  lampes:
   - {time: 'sunset/+60', data: 'ZZZZZZZZZ', enabled: true}
```

Every day of the weekend, open valve 3 from 15:00 to 15:30.

```
weekend:
  arrosage:
    - {time: '15:00', data: '03A', enabled: true, id: abc}
    - {time: '15:30', data: '03Z', enabled: true, id: abc}
```

Every day of the week, open the blinds of the second floor 30 minutes after sunrise and close them one hour after sunset.

```
weekdays:
  stores:
    - {time: 'sunrise/+30', data: '0AXXXX', enabled: true,
id: dffrqpyoo}
     - {time: 'sunset/+60', data: '0ZXXXX', enabled: true,
id: dffrqpyoo}
```

On Sundays, close every blind at 23:00.

```
sunday:
  stores:
    - {time: '23:00', data: '0Z0Z0Z', enabled: true}
```