Home Assistant SmartMonitor

Version alpha 0.1

Décembre 2022

Présentation

SmartMonitor est un logiciel embarqué dans un ESP32 permettant de communiquer avec Home Assistant, il sera vu comme un « device » et permettra d'afficher/actionner différents éléments de Home Assistant via son broker Mqtt.

Le projet est disponible sur Gihhub : https://github.com/PM04290/Home-Assistant-SmartMonitor

Programmation de l'ESP32

L'explication ci-dessous ne tient compte que de l'utilisation de l'IDE Arduino.

Préparation

Vous devez avoir installé le core 1.x pour ESP32 en ajoutant le lien ci-dessous dans les **préférences**:
https://dl.espressif.com/dl/package_esp32_index.json

⌦ Le core 2.x sera utilisé dans les prochains mois afin de pouvoir utiliser les ESP32-C et S.

Pour télécharger le dossier **data** vous devez avoir installé le plug-in ci-dessous:
https://github.com/me-no-dev/arduino-esp32fs-plugin

Il existe de très bons tutoriels si vous avez des problèmes sur ces deux points.

WT32-SC01

Si vous pocédez le module WT32-SC01, vous pouvez télécharger directement le firmware smartmonitor_WT32-SC01,bin sans avoir à tout recompiler ni vous préoccuper des librairies.
Il est disponible chez aliexpress ou chez le fabricant.

ESP32 WROOM ou WROVER

Dépendances des libraires

- Disponibles dans le gestionnaire de librairies:
  - ArduinoJson
  - ESPAsyncWebServer
  - LovyanGFX
- En téléchargement sur mon Github
  - HAintegration (https://github.com/PM04290/HAintegration)

Préparation du module

Avant de compiler le projet il faut configurer celui-ci pour votre couple ESP / TFT.
pour cela éditez le fichier display_setup.h; sélectionnez le define ESP32_TFT:

```
// Choix de la cible
//#define SC01
//#define SC01Plus (bientôt avec core 2.x)
#define ESP32_TFT
```

Puis en bas du fichier, il faut configurer le type de **Panel**, de **Bus** et de **TouchScreen** .

```
class LGFX : public lgfx::LGFX_Device
{
    lgfx::Panel_ILI9341 _panel_instance;
    lgfx::Bus_Parallel8 _bus_instance;
    lgfx::Touch_XPT2046 _touch_instance;
```

Ensuite il faut indiquer le raccordement qui est fait sur l'ESP32, l'exemple ci-dessous est donnée pour un écran TFT équipé du bus de data 8bits.

```
        auto cfg = _bus_instance.config();
        cfg.freq_write = 20000000;
        cfg.pin_wr = 4;
```

```
        cfg.pin_rd = 2;
        cfg.pin_rs = 15;  // Data / Command

        // LCD data interface, 8bit MCU (8080)
        cfg.pin_d0 = 12;
        cfg.pin_d1 = 13;
        cfg.pin_d2 = 26;
        cfg.pin_d3 = 25;
        cfg.pin_d4 = 17;
        cfg.pin_d5 = 16;
        cfg.pin_d6 = 27;
        cfg.pin_d7 = 14;

        _bus_instance.config(cfg);
        _panel_instance.setBus(&_bus_instance);
```

Puis la configuration du Panel

```
        auto cfg = _panel_instance.config();

        cfg.pin_cs           =    33;
        cfg.pin_rst          =    32;
        cfg.pin_busy         =    -1;

        cfg.panel_width      =   240;
        cfg.panel_height     =   320;
        cfg.offset_x         =     0;
        cfg.offset_y         =     0;
        cfg.offset_rotation  =     1;
        cfg.dummy_read_pixel =     8;
        cfg.dummy_read_bits  =     1;
        cfg.readable         =  true;
        cfg.invert           = false;
        cfg.rgb_order        = false;
        cfg.dlen_16bit       = false;
        cfg.bus_shared       =  true;

        _panel_instance.config(cfg);
```

Si le TFT est équipé du TouchScreen, il est possible de le configurer, sinon la définition peut être supprimée (ou commentée).

```
        auto cfg = _touch_instance.config();

        cfg.x_min      = 0;
        cfg.x_max      = 239;
        cfg.y_min      = 0;
        cfg.y_max      = 319;
        cfg.pin_int    = -1;
        cfg.bus_shared = true;
        cfg.offset_rotation = 1;
```

```
        cfg.spi_host = VSPI_HOST;
        cfg.pin_cs   = 33;
        cfg.pin_mosi = GPIO_NUM_23;
        cfg.pin_miso = GPIO_NUM_19;
        cfg.pin_sclk = GPIO_NUM_18;

        cfg.freq = 2700000;

        _touch_instance.config(cfg);
        _panel_instance.setTouch(&_touch_instance);
```

De même si un rétroéclairage est disponible, vous pouvez vous baser sur la définition du SC01 pour le configurer, par exemple:

```
        auto cfg = _light_instance.config();

        cfg.pin_bl = 35;
        cfg.invert = false;
        cfg.freq   = 44100;
        cfg.pwm_channel = 7;

        _light_instance.config(cfg);
        _panel_instance.setLight(&_light_instance);
```

Sans oublier d'ajouter le déclaration au début de Class:

```
    lgfx::Light_PWM     _light_instance;
```

N’hésitez pas à vous reporter au site original de la librairie LovyanGFX (https://github.com/lovyan03/LovyanGFX) afin de trouver des informations.

Cette partie est, pour le moment, la plus compliquée du projet en attendant que la banque de données s’enrichisse d’une liste d’écrans utilisés par d’autres.

Si vous souhaitez utiliser des pin libres de l’ESP32 pour gérer des Sensor ou des Switch, visible dans H.A. comme une entité, il faut indiquer leurs numéros dans le tableau suivant:

```
static uint8_t pinAvailable[] = {5, 22, 23};
```

⌦ A ce jour, seul le type binary_sensor est disponible.

Compilez le projet, téléchargez le dossier **data** et téléversez le projet.

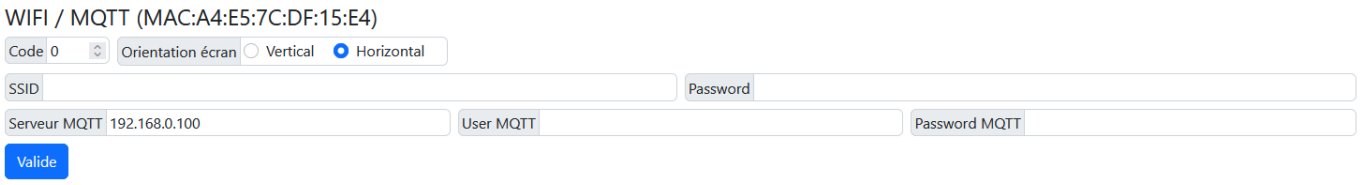
Les **data** contiennent une configuration par défaut qui affiche uniquement l’élévation du soleil car l’intégration est normalement présente sur toutes les installations H.A.
Cette configuration ne contient pas les caractéristiques de votre wifi, il faut donc établir une connexion au serveur web du module.
Quand le module ne trouve pas de réseau wifi, il passe automatiquement en mode **Point d’accès**; vous devez donc vous connecter à son wifi propriétaire : **smartmon0**
Le mot de passe par défaut est : 12345678

Ensuite, utilisez un navigateur internet sur L’URL **http://smartmon0.local** afin d’entrer les premiers

paramètres de configuration (l'adresse IP par défaut est 192.168.4.1)

La page web propose 3 zones:
- Pages / Actionneurs
- Wifi
- Mises à jour (firmware et fichiers)

Commencez par configurer le réseau wifi ainsi que les caractéristiques d'accès au serveur Mqtt de H.A.



> Si vous avez un serveur DHCP qui contrôle les adresses MAC, celle du module est indiquée dans le titre de la zone afin d'ouvrir le bail adapté sur votre serveur DHCP.

⚠ Important
Si vous devez utiliser plusieurs modules, 2 par exemple, indiquez 1 dans Code. Après le redémarrage, le module répondra à l'adresse **smartmon1.local** et ainsi le module suivant ne sera pas être perturbé.

C'est aussi lors de cette configuration que vous pouvez choisir l'orientation de l'afficheur : Vertical / Horizontal.

Après la préparation du module, avec la configuration du wifi et du serveur Mqtt, si vous démarrez vous devriez avoir une icône d'erreur et l'information de l'heure qui n'arrive pas : H.A. n'envoie pas encore de données.

Préparation de H.A.

Avant tout il faut que H.A. publie les états des différentes entités que vous souhaitez gérer avec le module, et il faut commencer par l'heure, afin de valider la bonne connexion du module au serveur.

Mise à jour de l'heure

Créez une automatisation :

```
- id: sm_mqtt_publish_time #id uniquement si vous éditez le fichier .yaml
  alias: Publication heure pour SmartMonitor
  description: ''
  trigger:
  - platform: time_pattern
    minutes: /1
  condition: []
  action:
  - service: mqtt.publish
    data:
      topic: smartmonitor/dateheure
      payload_template: '{{ now().strftime(''%d/%m/%Y %H:%M'') }}'
  mode: single
```

Une fois l'automatisation activée, l'heure est publiée toutes les minutes; si elle ne s'affiche pas sur le module il faut vérifier la configuration Mqtt : Adresse, utilisateur et mot de passe.

A savoir que le module ne fait aucune interprétation du texte qui est envoyé, vous pouvez vous servir de se Topic pour envoyer n'importe quel texte qui sera affiché en haut au centre de l'écran.

Publication des sensors

Pour réaliser cela il faut utiliser le mqtt_statestream de H.A.

Dans le fichier **configuration.yaml** il faut ajouter :

```
mqtt_statestream:
  base_topic: smartmonitor # Ne pas changer
  publish_attributes: true     # Obligatoire
  include:
    domains:
      - sun
```

Commençons simplement par le domaine sun qui va permettre de valider la configuration présente dans le module.

Après le redémarrage de H.A. vous devriez voir l'élévation sur le module.

⚠ le **base_topic** ne doit pas être changé et la ligne **publish_attributes: true** est obligatoire car elle permet au module de récupérer les informations complémentaires des capteurs : unité, type, etc.

Basez vous sur la documentation de Home Assistant afin de régler correctement la diffusion des informations utiles à SmartMonitor. Les balises **include** / **exclude** ainsi que **domains** et **entities** permettent un réglage très fin de la diffusion.

Voici un exemple trivial, non filtré, de la publication des toutes les entités des domaines indiqués:

```
mqtt_statestream:
  base_topic: smartmonitor
  publish_attributes: true
  include:
    domains:
      - weather
      - sensor
      - binary_sensor
      - cover
      - switch
      - light
      - sun
      - alarm_control_panel
```

Sur une grosse configuration, le serveur Mqtt va être chargé de beaucoup de données pas nécessairement utiles au module.

N'hésitez pas à utiliser Mqtt Explorer afin de visualiser la liste des données publiées.

A noter que si vous avez publié trop de données (comme l'exemple ci-dessus) avant de mieux filtrer votre besoin, il est possible avec MQtt explorer de supprimer l'arborescence smartmonitor et de relancer le serveur Mqtt. L'arborescence sera reconstituée avec uniquement les valeurs filtrées.

Configuration de SmartMonitor

Maintenant que plusieurs données sont disponibles sur le serveur Mqtt, il est possible de configurer l'affichage sur le module.

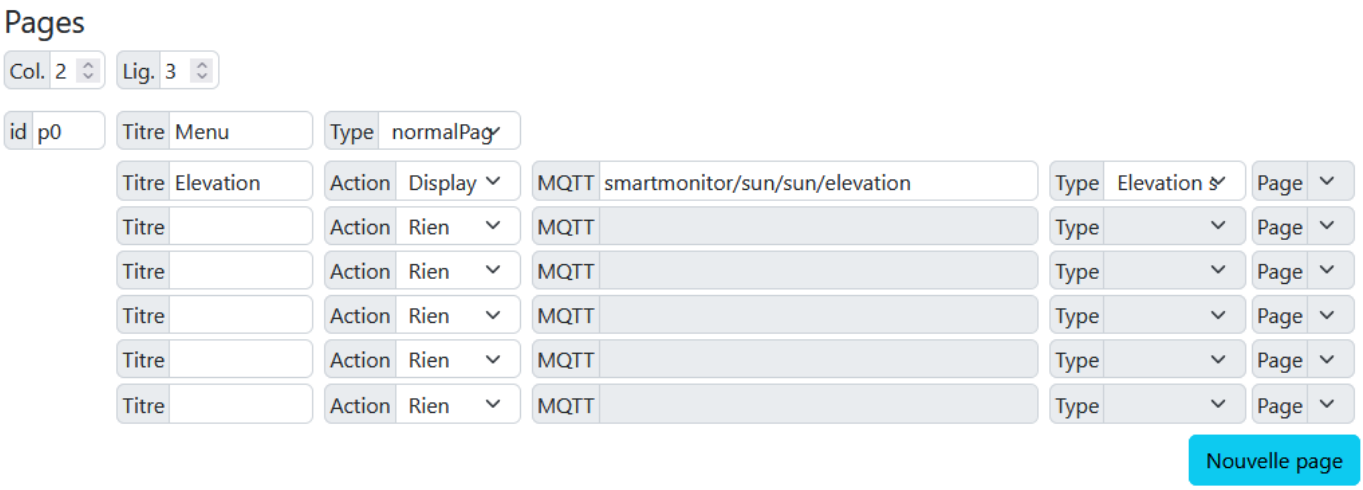
En premier lieu il faut choisir le nombre de Zone d'affichage. Par défaut la configuration est de 2x3 (2 colonnes, 3 lignes) qui est la plus optimisée pour un afficheur 480x320, à l'horizontale, avec des longueurs de texte raisonnables.

L'ordre d'affichage des zones sur le module est le suivant :

ZONE 1	ZONE 2
ZONE 3	ZONE 4
ZONE 5	ZONE 6

⚠ Si vous souhaitez changer le nombre de zone, il faut le faire au tout début, **Valider** cette simple modification et relancez le module (bouton **Reset** en bas de page).

Description des paramètres



Vous constaterez que l'affichage de la page est effectué en plusieurs fois, en effet, il peut y avoir beaucoup de données affichées si il y a plusieurs pages et la mémoire de l'ESP32 (même si elle est importante) ne permet pas de traiter un gros volume; l'affichage a donc été segmenté en parties élémentaires, et envoyées par Websocket (un peu de technique). Laissez bien la page s'afficher complètement avant de réaliser le paramétrage.

⌦ Problème connu : lors de certains affichages, la page peut ne pas se rafraîchir complètement ou rester bloquée au début du chargement. Je n'ai pas encore réussi à trouver d'où venait le problème ; en général un simple rafraîchissement du navigateur suffit à recharger la page correctement.

Définition d'une page

ID est un code unique de page qui va permettre la gestion du changement de page (lecture seule).

Titre est un libellé simple qui permet de s'y retrouver durant la configuration (sera peut être utilisé dans le futur comme titre de page).

Type à ce jour seul *normalPage* est utilisable; keypadPage est visible mais non sélectionnable, la page du digicode est crée implicitement lors de la configuration de *Alarm_control_panel*.

Définition des zones d'affichage

Titre il s'agit du titre d'un Bouton, du nom d'un Trigger (visible dans H.A.) ou simplement d'un libellé dans le cas d'affichage simple.

Action détermine la fonction de la zone:

- **Rien**, quand la zone doit rester vide
- **HAtrigger**, quand on veut définir un « trigger » qui sera utilisable par une automatisation dans H.A.; le titre devient le *subType* (le nom) du Trigger visible par H.A. avec un type *button_short_press*.
- **changePage** permet de définir la zone comme un bouton qui va permettre d'ouvrir une nouvelle page. Le cas particulier du digicode est traité dans le chapitre de l'alarme.
- **Display** définit un simple affichage de valeur.
- **Command** définit une action possible, par exemple l'ouverture/fermeture de volet ou la gestion d'un switch.

Mqtt permet de définir le Topic qui sera utilisé pour récupérer les données; ou les envoyer.
Type définit la façon dont sont affichées les données. Les premiers de la liste avec le nom qui commence par **Etat** permet d'afficher une icône correspondant au **State** de l'entité suivi du **Titre** choisi. Les autres affichent une icône correspondant à l'entité suivi de la valeur reçue.

- L'unité est pour le moment codée "en dur", mais dans la TODO list il est prévu de récupérer l'information depuis le serveur Mqtt.

Page permet de choisir la page qui sera affichée lors du choix de l'action **changePage**.

- Par défaut la liste ne contient que "kp" (keypad), il faut créer une nouvelle page pour que le complément de liste soit mis à jour.

Exemples de configurations

Ci-dessous, dans Mqtt, les termes en **gras** sont ceux que vous avez choisi dans H.A.

<u>Affichage de la température extérieure fournie par une intégration météo:</u>

- Titre : Météo
- Action : Display
- MQTT : smartmonitor/weather/**maison**/temperature
- Type : Température
- Page : *inactif*

<u>Pilotage d'un volet, ou groupe de volets:</u>

- Titre : Volets nuit
- Action : Command

- MQTT : smartmonitor/cover/**volet_nuit**;/state;/set
- Type : Etat volet
- Page : *inactif*

Vous noterez la présence de « ; » dans Mqtt qui permet de séparer la base du topic, le terme qui permet la lecture (**state**) et celui qui permet l'écriture (**set**). Utilisez MQqt explorer pour vérifier cela si vous rencontrez un problème de pilotage et/ou d'affichage.

<u>Pilotage d'un interrupteur 3 boutons (bouton de droite)</u>

- Titre : Cuisine
- Action : Command
- MQTT : zigbee2mqtt/**Inter_Cuisine**;/#|state_right ;/right/set
- Type : Etat lampe
- Page : *inactif*

Vous noterez que dans cet exemple, c'est zigbee2mqtt qui est utilisé, en effet, mqtt_statestream n'est utilisé que pour ajouter les entités souhaitées et non disponible par ailleurs.

Dans ce cas de figure, il s'agit d'un interrupteur qui publie ses données sous forme de json.


```
{
  "backlight_mode": "low",
  "linkquality": 65,
  "power_on_behavior": null,
  "power_on_behavior_left": "off",
  "state_center": "OFF",
  "state_left": "OFF",
  "state_right": "OFF"
}
```


La valeur est récupérée sur l'attribut **state_right** ; lorsque la commande sera activée par l'écran, la valeur est envoyée avec le topic **right/set**.

Pour connaître le topic de commande, vous pouvez aller dans MQTT INFO de l'appareil de H.A.

Descriptif de Mqtt :

`<base topic>;<state topic>|<attribute state>;<command topic>`

Trigger permettant l'ouverture d'un portail:

- Titre : Portail
- Action : HAtrigger
- MQTT : *inactif*
- Type : *inactif*
- Page : *inactif*

Dans H.A. vous trouverez un déclencheur attaché au device **smartmon0** ayant le nom **button_short_press_Portail** sur lequel vous pourrez déclencher une automatisation.

Exemple de changement de page:
Il faut d'abord créer la nouvelle page en utilisant le bouton prévu à cet effet (en bas à droite de la zone Pages). Par exemple une page regroupant toutes les informations de consommation/production d'énergie.

- Titre : Info énergie
- Action : changePage
- MQTT : *inactif*
- Type : *inactif*
- Page : p1

Lors d'un changement de page, celle-ci est affichée 10s, réarmés lors de chaque « tap » ; au-delà de cette durée, la page par défaut « p0 » sera ré-activé.

- Quand Mqtt n'est pas renseigné, le titre est affiché dans le bouton permettant le changement de page.
- Quand Mqtt est renseigné par un topic/state, la valeur ou l'icône sont affichés. On peut par exemple afficher un récapitulatif de consommation/production d'énergie dans le bouton et lorsque l'on fait un « tap » dessus, on affiche la page de détail de chaque production/consommation.

Gestion de l'alarme

- Titre : Alarme
- Action : changePage
- MQTT : smartmonitor/alarm_control_panel/**alarme**/state
- Type : Etat lalarme
- Page : p1

Dans ce cas, on combine le changement de page avec l'affichage du statut de l'alarme 

En tapant sur l'icône, on change de page pour gérer le nouveau statut de l'alarme.

Conformément à la définition de H.A., l'alarme peut prendre les statuts suivants:

Désactivée	

Activée absent	
Activée nuit	
Armement	
Délai avant déclenchement (c'est la durée durant laquelle vous allez tapez le code pour désarmer)	
Déclenchement	

Pour le moment, seuls les modes `ALARM_AWAY` et `ALARM_NIGHT` sont disponibles, les autres sont dans la TODO list.

Du coté de H.A. il faut que l'alarme soit configurer en tant que « manual ».

```
alarm_control_panel:
  - platform: manual
    name: "Alarme"
    code: 1234
    code_arm_required: false
    # temps en seconde pour partir après mise en mache
    arming_time: 30
    # temps en seconde pour saisir le code pour désarmer
    delay_time: 60
    # temps en seconde du décenchement (pour la durée sirène par exemple)
    trigger_time: 30
```

Lorsque un bouton de changement de mode est pressé, un Trigger est envoyé contenant le l'action demandée et le code, par exemple:

```
{action: ARM_NIGHT, code=""}
```

ou:

```
{action: DISARM, code="1234"}
```

Automatisation du changement de mode

L'automatisation est crée initialement dans son éditeur afin de pouvoir choisir simplement le device et son trigger correspondant : **button_short_press_AlarmKeypad**

Exemple simple de changement de mode sans vérification de capteur:

```
mode: single
trigger:
  - platform: device
```

```
    domain: mqtt
    device_id: 3cdce4ea751b50e2f5fe1b94a1710430 # defini par UI
    type: button_short_press
    subtype: AlarmKeypad
    discovery_id: a4e57ce081cc button_short_press_AlarmKeypad
condition: []
action:
  - service: alarm_control_panel.{{ trigger.payload_json.action }}
    target:
      entity_id:
      - alarm_control_panel.alarme
    data:
      code: '{{ trigger.payload_json.code | int(0) }}'
```

Dans cet autre exemple ci-dessous, il y a une vérification de l'état d'un groupe de contact de fenêtre, l'alarme n'est activée que si toutes les fenêtres sont fermées (groupe **contacts_ouvertures**); sinon un message d'information est envoyé au module.

```
- id: sm_trigger_alarm_change_mode # uniquement si édition du fichier yaml
  alias: 'Alarme: Changement de mode'
  description: ''
  trigger:
  - platform: device
    domain: mqtt
    device_id: 284cfad79d5e602d26418e26554a4d6b #défini par UI
    type: button_short_press
    subtype: AlarmKeypad
    discovery_id: a4e57cdf15e4 button_short_press_AlarmKeypad
  condition: []
  action:
  - if:
    - condition: or
      conditions:
      - condition: state
        entity_id: binary_sensor.contacts_ouvertures
        state: 'off'
      - condition: not
        conditions:
        - condition: state
          entity_id: alarm_control_panel.alarme
          state: disarmed
    then:
    - service: alarm_control_panel.{{ trigger.payload_json.action }}
      target:
        entity_id:
        - alarm_control_panel.alarme
      data:
        code: '{{ trigger.payload_json.code | int(0) }}'
    else:
    - service: input_text.set_value
      data:
        value: '{% set defauts = ["Fenêtre(s) ouverte(s):"] +
```

```
expand(''binary_sensor.contacts_ouvertures'')
          | selectattr(''state'',''in'',[''on'',''open'',''unknown'']) |
map(attribute=''name'')
          | list %} {%- if defauts | length > 1 -%} {%- set json = {"text":
defauts}
          -%} {{ json }} {%- endif -%}
          '
      target:
        entity_id: input_text.notification_text
  mode: single
```

La configuration du message temporaire est décrite ci-dessous.

Messages temporaires sur l'écran

Le module peut recevoir un message temporaire, par exemple pour informer de l'état des capteurs lors d'une mise en alarme (si une fenêtre est restée ouverte).
Pour cela il faut créer un input_text dans H.A. :
« Paramètres » / « Appareils et services » / « Entrées »

Créez une nouvelle entrée de type **Texte** que vous nommerez par exemple : **notification_text**

Lors que l'on désire envoyer un message au module, il suffit d'affecter une valeur à **notification_text**; une automatisation va publier le message et réinitialiser la valeur après un certain temps.

Automatisation de message temporaire

```
- id: sm_raz_published_message  #id uniquement si vous éditez le
fichier .yaml
  alias: raz message smartmonitor
  description: ''
  trigger:
  - platform: state
    entity_id:
    - input_text.notification_text
  condition:
  - condition: template
    value_template: '{{ states(''input_text.notification_text'')|length !=
0 }}'
  action:
  - service: mqtt.publish
    data:
      topic: smartmonitor/alarm_message
      payload_template: '{{ states(''input_text.notification_text'') }}'
  - delay:
      hours: 0
      minutes: 0
      seconds: 2
      milliseconds: 0
  - service: input_text.set_value
    data:
      value: ''
```

```
    target:
      entity_id: input_text.notification_text
  - delay:
      hours: 0
      minutes: 0
      seconds: 1
      milliseconds: 0
  - service: mqtt.publish
    data:
      topic: smartmonitor/alarm_message
      payload_template: '{{ states(''input_text.notification_text'') }}'
  mode: single
```

to be continued... soon 😁

TODO list

- [x] ajouter tous les types de capteurs connus par H.A.
- [x] gestion des publications d’attributs
- [ ] récupération automatique des unités d'affichage
- [ ] compléter la liste des écrans TFT dans display_setup.h
- [ ] affichage du forecast météo (icônes sur n jours)
- [ ] gérer tous les modes d'alarme supplémentaires (present, vacation, custom)
- [ ] amélioration du message temporisation pour devenir une notification
- [ ] finir GPIO
- [ ] traduction