Développement d'un Composant ESPHome
Partie 1 : Introduction

Présentation

La vocation première d'ESPHome est de permettre la création de **modules** électroniques par des utilisateurs *qui n'ont aucune connaissance en informatique*. Pour cela il fait appel à une bibliothèque de **composants** qu'il assemble en fonction des demandes de l'utilisateur et génère automatiquement pour lui le code nécessaire pour réaliser la fonction souhaitée. Ensuite l'utilisateur n'a qu'à assembler sur une carte électronique les différents composants dont il a besoin. Il faut ensuite la connecter à un ordinateur pour charger le code généré par ESPHome.

Une précision concernant la terminologie utilisée :

- Un « **module** » (**device**) sert à désigner un ensemble de **composants** grouper par un utilisateur pour réaliser une fonction. Dans ESPHome la description d'un module est définie dans un fichier yaml. Par exemple on peut définir un module pour mesurer la température d'une pièce. Pour cela on va utiliser un microcontrôleur ESP32 qui sera connecté par un bus I²C à un capteur de température.
- Un « composant » (**component**) sert à désigner un élément de base de la bibliothèque d'ESPHome. Les composants peuvent être des composants physiques comme par exemple un capteur, ou des composants logiques comme par exemple un bus I²C. Un composant est défini par un ensemble de fichiers en langage C++ et langage Python.

Un **utilisateur** normal d'ESPHome crée de nouveaux **modules.** Les modules sont décrits dans un fichier de configuration Yaml et à partir de cette description le programme ESPHome génère automatiquement du code C++. Ce code est compilé et un exécutable est généré. Le firmware résultant est chargé dans la mémoire du microcontrôleur et son exécution est lancée.

Un **développeur** pour ESPHome va lui créer de nouveaux **composants** pour la bibliothèque. C'est une opération beaucoup plus complexe car elle nécessite que le développeur décrive le fonctionnement du composant en utilisant les langages C++ et Python. Cela requiert une bonne connaissance du langage C++, et des notions de base du langage Python.

J'ai été confronté au problème de créer un **module** ESPHome qui utilisait des **composants** qui *n'existaient pas* dans la **bibliothèque** d'ESPHome. Et donc je me suis posé la question de savoir quelle procédure et quels outils utiliser pour créer ce nouveau composant.

À ma connaissance il existe beaucoup de tutoriels sur la création de modules dans ESPHome mais très peu sur la création de nouveaux composants. J'ai donc procédé par tâtonnements avec l'aide d'autres utilisateurs et au final je me suis défini un environnement et un processus à suivre. Comme j'ai appris pas mal de choses durant ce cheminement j'ai pensé qu'il serait intéressant de partager cette expérience. Je précise que je présente ici la façon dont moi je procède, mais il existe évidemment d'autres façons de procéder.

Vision à haut niveau d'ESPHome

Je vais d'abord vous présenter le fonctionnement d'ESPHome à un haut niveau d'abstraction tel que je le comprends.

Pour créer un nouveau **module** dans ESPHome, vous devez définir tous les composants que vous souhaitez utiliser et comment les interconnectés.

Par exemple pour créer un module de « mesure de température dans une pièce » vous allez indiquer que vous voulez utiliser : un microcontrôleur ESP32 installé sur une carte de développement esp32dev connecté à un capteur BMP085 au travers d'un bus I²C.

Le capteur mesure la température et la retourne à Home Assistant au travers de l'API native d'ESPHome en Wifi.

Le fichier Yaml va ressembler à cela :

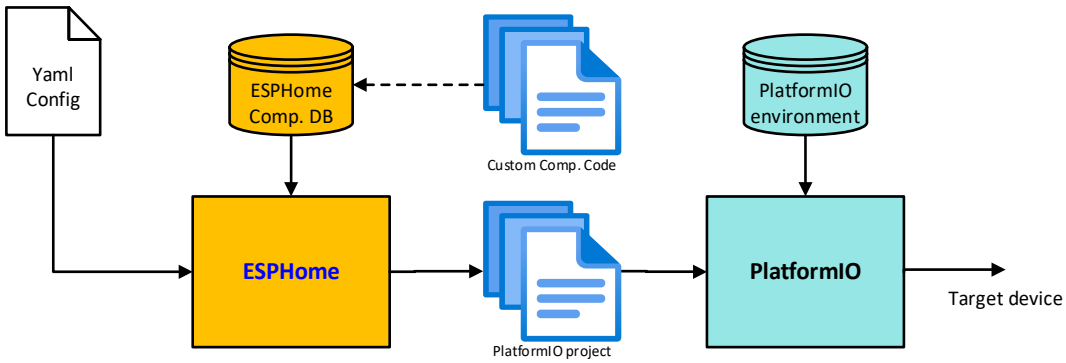
```
esphome :
  name: test-bmp085
  platform : esp32
  board: esp32dev

api :

wifi :
  ssid: !secret wifi_ssid
  password: !secret wifi_password

i2c:
  sda: 21
  scl: 22
  id: bus_i2c_id

sensor:
  - platform: bmp085
    i2c_id : bus_i2c
    address: 0x77
    temperature:
      name: "Outside Temperature"
```

Dans ce fichier on indique qu'on crée un nouveau module ESPHome appelé « test-bpm085 », que le microprocesseur utilisé est un « ESP32 », lequel est monté sur une carte de développement « esp32dev », que le module se connecte à votre réseau « Wi-Fi », qu'il communique avec Home Assistant grâce au « protocole api natif », qu'il utilise un bus « I2C », et qu'un « capteur BMP085 » connecté au bus I²C pour retourne la température.

À partir de cette description la *magie* d'ESPHome opère en exécutant les tâches suivantes :

- Dans un premier temps ESPHome va lire le fichier Yaml. Ce qui lui permet d'énumérer tous les composants utilisés et de connaitre toutes les interconnexions (création interne d'une netlist). Dans cette phase une vérification de la syntaxe et de la sémantique du fichier Yaml est effectuée. C'est la phase dite de préparation et de vérification.
- Dans un deuxième temps, ESPHome va rechercher dans sa bibliothèque tous les composants qui sont utilisés dans ce module. Par exemple : un composant wifi, un composant I2C, un composant Api, un composant BMP085, un composant capteur, etc. Il copie alors le code source de ces composants dans un répertoire créer spécifiquement pour le module en cours d'assemblage. Afin « d'instancier » et « de connecter » les composants entre eux, à partir de la netlist, il génère un fichier en C++ (main.cpp). C'est la phase de génération de code. Au cours de cette phase, ESPHome génère également tout l'environnement nécessaire à PlatformIO. Il lance ensuite PlatformIO qui va se charger de créer, télécharger, et lancer l'exécution du firmware résultant sur la cible matérielle.

Ce diagramme décrit le processus :



Donc pour résumer, après validation du fichier de configuration, les fichiers sources des différents composants utilisés sont copiés dans le projet PlatformIO, un fichier main.cpp ainsi que le fichier plaformio.ini sont générés et le contrôle est passé à PlatformIO.

PlatformIO vérifie que l'environnement de développement nécessaire, tel que les compilateurs, les bibliothèques, les utilitaires de programmation, etc. sont bien présents et à jour. Cet environnement est spécifié dans le fichier de configuration `platformio.ini` généré par ESPHome. PlatformIO va ensuite compiler et télécharger le code résultant dans l'appareil cible.

Dans cette série de présentations nous allons nous intéresser non pas à l'utilisation mais à la création de composants qui pourront par la suite être utilisés pour créer de nouveaux modules.

Le processus commence par :

- l'écriture d'une documentation qui indiquera à l'utilisateur comment utiliser notre nouveau composant.
- l'écriture de fichiers Python pour valider le module et générer du code C++ à partir de la configuration utilisateur.
- L'écriture de fichiers C++ pour « modéliser » le composant. Cette partie du développement est la plus longue et la plus complexe car elle nécessite d'analyser en détail les spécifications du composant, de comprendre un minimum l'architecture interne d'ESPHome, et d'écrire le code qui modélise fonctionnellement le composant.
- La soumission de votre nouveau composant à la bibliothèque d'ESPHome. C'est une phase parfois longue et difficile car le code et la documentation pour les composants d'ESPHome doivent respecter des contraintes très strictes. Cette phase nécessite de dialoguer avec la personne en charge de la validation de votre composant…
- Bien que cela ne soit mentionné nulle part, je pense que lorsque vous ajoutez un nouveau composant à ESPHome il est implicitement entendu que vous vous engagez à maintenir ce composant (bug-fix et enhancement). Pour cela votre nom de développeur GitHub est associé au composant.

Mais avant tout cela il faut commencer par installer l'environnement de développement…