Proof-Of-Concept - Contrôle device Matter

Target release	
System	
Document status	READY
Document owner	Utilisateur inconnu (gregory.lemercier@leroymerlin.fr) Utilisateur inconnu (leo.segretain@leroymerlin.fr) Utilisateur inconnu (ismayl.abdouissakou@leroymerlin.fr)
Developers	Enki -
Formation	Matter Lighting-app (ESP32-WROOM-32D)
Contents	
Last updated	9 mai 2022
Programme de Formation	

Lancer un exemple de projet Matter avec une ESP32

Lexique

Clé	Description
chip-tool	L'outil `chip-tool` est un controller Matter qui permet de mettre connecter un device Matter dans le réseau et de communiquer avec lui à l'aide de messages Matter. Ces messages peuvent encoder des commandes de cluster.

Pour bien suivre ce tutoriel, vous aurez besoin de préparer votre espace de travail et installer quelques outils nécessaires, notamment **Espressif IoT Development** Framework (ESP-IDF).

Configuration des l'espace de travail et installation des outils

Depuis votre votre terminal, vous pouvez exécuter les commandes suivantes :

```
mkdir ${HOME}/tools
cd ${HOME}/tools
git clone https://github.com/espressif/esp-idf.git
cd esp-idf
git checkout v4.4.1
git submodule update --init
./install.sh
cd ~/
sudo apt-get install git gcc g++ python pkg-config libreadline-dev minicom libssl-dev libdbus-1-dev libglib2.0-dev libavahi-client-dev ninja-build python3-venv python3-dev python3-pip unzip libgirepository1.0-dev libcairo2-dev
```

Certains des outils installés ne sont pas encore ajoutés à la variable d'environnement `PATH`. Pour se faire(n'oubliez pas le point 😉):

```
. $HOME/esp/esp-idf/export.sh
```

Comme vous aurez besoin d'utiliser cette commande régulièrement, vous pouvez créer un alias en rajoutant la ligne suivante dans votre fichier de configuration (`~/.bashrc` sous Linux) :

```
alias get_idf='. $HOME/tools/esp-idf/export.sh'
```

ainsi vous n'aurez qu'à exécuter la commande `get_idf` pour configurer ou actualiser l'environnement ESP-IDF.

Vous pouvez à présent cloner ce dépôt https://github.com/project-chip/connectedhomeip

```
git clone https://github.com/project-chip/connectedhomeip
```

Vous aurez ainsi un répertoire nommé `connectedhomeip` qui contient plein d'exemples de projet Matter.

Il faudra maintenant télécharger, installer et activer certains packages avec les commandes suivantes :

```
cd ${HOME}/connectedhomeip
source ./scripts/bootstrap.sh
source ./scripts/activate.sh
```

si ces packages sont déjà installés, vous n'avez qu'à les activer avec la deuxième commande.

Dans la suite, nous allons utiliser l'exemple **lighting-app** du répertoire `~/connectedhomeip/lighting-app/esp32/` pour allumer et éteindre une ampoule.

Flasher sa carte ESP32

Vous pouvez maintenant brancher votre devkit ESP32 avec un câble data micro-USB. Vous retrouverez le nom du device qui commence par `tty` dans le répertoire `/dev` (sous Linux en général c'est `ttyUSB0` et sous mac `tty.SLAB_USBtoUART`).

Utilisez la commande suivante :

```
ls /dev/tty*
```

Certains utilisateurs doivent installer **VCP driver** avant que le device n'apparaisse sur `dev/tty`.

L'utilisateur actuellement connecté doit avoir un accès en lecture et en écriture au port série via USB. Sur la plupart des distributions Linux, cela se fait en ajoutant l'utilisateur au groupe `dialout`(instructions 1 à 3).

1. Vérifiez que le groupe existe avec la commande `groups` sinon il faudra le créer avec `newgrp dialout`. Si vous avez du créer le groupe il faudra exécuter la commande `bash` pour qu'il soit pris en compte.
2. Exécutez la commande `sudo usermod -a -G dialout $USER` pour avoir un accès en lecture et en écriture au port série via USB.
3. Vous pouvez lancer `minicom` pour vérifier les log du device.
4. Assurez vous aussi d'avoir lancé l'exécutable `~/tools/esp-idf/export.sh` afin qu'il rajoute certains outils au `PATH`.

   ```
   . $HOME/esp/esp-idf/export.sh
   ```

   ou `get_idf`si vous avez défini un alias. (Attention !! les commandes `idf.py ...` ne s'exécutent que dans des répertoires contenant un fichier `C MakeLists.txt`)
5. Définir ESP32 comme cible et créer les exécutables(compiler) avec la commande suivante

```
cd ~/connectedhomeip/examples/lighting-app/esp32
idf.py set-target esp32
idf.py build
```

Le dossier build crée, il ne reste plus qu'à flasher la carte!

```
idf.py -p /dev/ttyUSB0 flash monitor
```

Une fois le flash terminé, vous pouvez passer à l'étape suivante.

Construire le chip-tool

Afin de mettre votre device Matter sur le réseau et le commander vous aurez besoin de créer le `chip-tool`

```
cd ${HOME}/connectedhomeip
scripts/examples/gn_build_example.sh examples/chip-tool examples/lighting-app/esp32/build/
```

Ainsi le `chip-tool` sera généré dans le dossier `examples/lighting-app/esp32/build/` .

Mise en Service du device via Bluetooth BLE.

Pour connecter votre carte au réseau wifi, aller dans le répertoire contenant le `chip-tool` et exécuter la commande suivante.

```
./chip-tool pairing ble-wifi 0x11 ${nom_wifi} ${mot_de_passe_wifi} 20202021 3840
```

Où `0x11` désigne l'identifiant (`NODE_ID`) du device que vous voulez mettre sur le réseau. Vous pouvez lui donner n'importe quelle valeur (hexa ou décimale) si vous voulez.

Pour reste, il s'agit simplement des infos additionnels sur le device. Vous pouvez trouver ces valeurs dans les log de l'appareil lorsqu'il démarre mais en générale, ça reste les mêmes.

S'il n'arrive pas à identifier votre device (i.e qu'il vous affiche *does not look like a chip device*) n'hésiter pas à appuyer sur le bouton reset de la carte une ou deux fois lorsqu'il est en cours de scan.

Allumer et éteindre votre device

Une fois votre device sur le réseau, vous allez utiliser à nouveau le `chip-tool` pour faire du `onoff`.

pour faire du **ON**

```
./chip-tool onoff on ${Id} 1
```

pour faire du **OFF**

```
./chip-tool onoff off ${Id} 1
```

Ou encore vous pouvez utiliser `toggle` pour basculer de on à `off` ou inversement.

```
./chip-tool onoff toggle ${Id} 1
```

Où :

- la valeur `${Id}` répresente le `NODE_ID` que vous avez attribué à votre device lors de la connexion,
- la valeur `1` répresente le point de terminaison (`endpoint` en anglais) qui est une valeur comprise entre 1 et 240.

A présent vous savez flasher une ESP32 et commander un device Matter 😉 . Vous pouvez retrouver pleins d'autres exemples si cela vous intéresse !

https://github.com/project-chip/connectedhomeip/tree/master/examples/light-switch-app/esp32

https://github.com/project-chip/connectedhomeip/tree/master/src/android/CHIPTool