

# Développement de Composants ESPHome

## Partie 2 : Environnement de Développement logiciel

### Environnement de développement logiciel sous Windows

Nous allons maintenant installer les différents logiciels nécessaires pour faire un développement directement sous **Windows**. Par rapport à l'utilisation de containers, cette façon de faire est facile à comprendre et les performances sont meilleures. Néanmoins il y a certaines opérations que je n'ai pas réussi à faire fonctionner correctement sous Windows (comme clang-tidy) c'est pourquoi j'ai une section de ce document qui indique comment faire les développements en utilisant le **Dev Container** d'ESPHome. Nous en verrons l'utilité quand je décrirai le processus de soumission de code dans **GitHub**.

Chaque développeur a ses habitudes et ses outils qui lui sont propres et par conséquent ce que je présente ici correspond à mes préférences. Il existe de nombreux environnements alternatifs qui peuvent être utilisés en fonction des goûts de chacun.

Les pièces maîtresses dont vous aurez besoin sont :

- Un éditeur de code : j'ai choisi d'utiliser Visual Studio Code (**VS Code**).
- Un environnement intégré de développement (IDE) : j'ai choisi **PlatformIO**.
- Un système de contrôle de version : j'ai choisi **Git**

Cette combinaison apporte de nombreuses fonctionnalités comme la complétion et la vérification intelligente du code C/C++ lors de l'écriture, le débogage intégré, la gestion des dépendances...

Noter que s'il est possible de choisir ce que vous voulez comme éditeur de code, par contre l'utilisation de PlatformIO et de Git est **obligatoire** si vous voulez intégrer votre code dans la bibliothèque d'ESPHome. Noter également qu'il est nécessaire que vous ayez un compte GitHub pour intégrer votre composant dans ESPHome.

Comme les logiciels que j'ai choisis sont largement utilisés, il existe déjà beaucoup de tutoriels qui expliquent en détail la façon de les installer et de les utiliser. Je vais donc me contenter d'en faire une présentation succincte.

### Logiciels pour le développement du code

Voici la liste des logiciels que j'ai installés sur ma machine de développement pour créer ou modifier des composants d'ESPHome. Je décris ici l'installation sur un PC avec Windows, mais ces logiciels sont disponibles sur Linux et Mac.

**Remarque importante sur les installations** : je recommande de sélectionner « installez pour vous seulement » plutôt que « installez pour tous » quand on vous donne le choix. Cela permettra à d'ESPHome d'installer ses logiciels dans le répertoire « C:\Users\<login\_name>\AppData\... » qui vous appartient et donc ouvert en écriture et non pas dans « C:\Program Files » qui lui est « read only ».

#### **Git - Système de contrôle de version distribué (must have)**

Un système de contrôle de version est indispensable lorsque l'on fait du développement de code.

Pour pouvoir intégrer votre code dans ESPHome il faut impérativement installer et utiliser le logiciel [Git](#). L'installation est standard et non décrite ici.

Après l'installation il faut configurer Git en remplissant **au minimum** les variables « **user.name** » et « **user.email** » qui permettront de vous identifier. Pour cela on peut lancer un « Git bash » et exécuter les commandes suivantes :

```
git config --global user.name "Your Name"
git config --global user.email youremail@yourdomain.com
git config --list
```

### **Visual Studio Code (VS Code) - Editeur de code (must have)**

Comme éditeur de code j'installe « [Visual Studio Code](#) ». C'est un éditeur de code open source très populaire pour sa polyvalence et ses fonctionnalités, notamment l'intégration de Git, un terminal intégré, des outils de débogage, IntelliSense et l'existence de milliers d'extensions. Nous personnalisons VS Code pour créer un environnement de développement adapté à ESPHome qui permettra de faire des vérifications et du formatage automatique de votre code afin de suivre les règles imposées par ESPHome. L'installation est standard et non décrite ici.

### **Installation d'extensions de Visual Studio Code**

Il existe énormément d'extension pour VS Code. Certaines de ces extensions seront automatiquement installées par PlatformIO lorsque vous importerez le repository d'ESPHome dans votre environnement.

Je recommande l'installation des extensions suivantes :

- **C/C++, C/C++ Extension Pack, C++ Themes,**
- **CMake, CMake Tools, Makefile tools,**
- Python, Pylance, PyLint, Python indent, **Black formatter**
- Code Spell Checker, Code Runner, GitHub Pull Requests and Issues, GitLens, Remote SSH, Yaml

### **PlatformIO IDE - Environnement de développement intégré (must have)**

[PlatformIO](#), qui est compatible avec de nombreux microcontrôleurs, fournit un environnement de développement unifié pour plusieurs langages de programmation tels que C ++, Python. Il offre des outils de débogage, une gestion des bibliothèques, des tests unitaires et des tas d'autres fonctionnalités pour gérer un projet. C'est l'environnement qui est utilisé par les équipes de développement d'ESPHome. PlatformIO est installé en tant qu'extension de VS Code. L'installation se fait dans VS Code elle est standard et non décrite ici.

### **Installation d'ESPHome (must have)**

Il est indispensable d'installer ESPHome pour pouvoir vérifier, compiler, et uploader votre composant directement depuis votre machine de développement.

Pour cela il faut **cloner le repository d'ESPHome** sur votre machine de développement. Cette opération est de toute façon nécessaire si vous voulez créer et soumettre un nouveau composant. Nous verrons plus en détail le processus complet nécessaire mais en résumant vous devez faire un fork d'ESPHome sur votre compte GitHub et puis cloner celui-ci localement sur votre machine de développement.

Dans VS Code ouvrez le répertoire qui contient ESPHome. PlatformIO va immédiatement initialiser le projet et installer toutes les extensions qui sont nécessaires. Cela peut prendre pas mal de temps.

C'est là que ça se complique un peu car la documentation « [Setting Up Development Environment](#) » ne s'applique pas à Windows mais à Unix. Donc à la place de la commande **script/setup** il faut en réalité taper la commande :

```
pip install -e .
```

Cette commande va installer les différents utilitaires nécessaires au bon fonctionnement d'ESPHome et si tout se passe bien vous devriez voir apparaître un message comme celui-ci :

```
Successfully installed esphome-2023.5.0.dev0
```

Pour vérifier qu'ESPHome s'exécute correctement tapez la commande suivante :

```
esphome version
```

Félicitation vous allez maintenant pouvoir compiler le code de votre composant et l'envoyer sur votre carte de développement directement depuis votre système y compris en lançant une session de débogage matérielle.

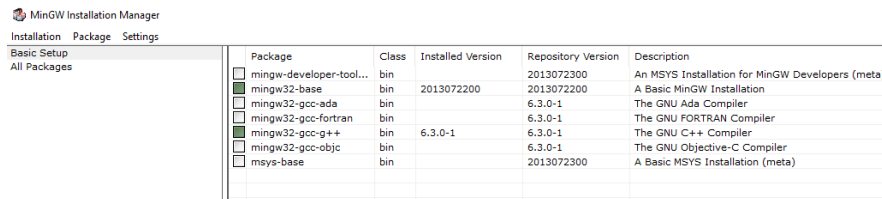
### **Python (recommended)**

Il est recommandé (mais pas indispensable) d'installer [Python](#). Lors de l'installation vérifiez que Python soit disponible à partir de n'importe quel emplacement en modifiant la variable de PATH de votre système. L'installation est classique et non décrite ici.

### **C++ compiler & debugger (optional)**

#### **MinGW**

Cette suite d'applications permet de compiler et d'exécuter des programmes écrits en C++. Il n'est pas nécessaire de l'installer si vous ne voulez pas faire de développement en C++ en dehors d'ESPHome. Personnellement j'utilise MinGW que j'installe à partir de « [mingw-get-setup](#) » en choisissant les packages « mingw32-base » et « mingw32-gcc-g++ ».



| Package   | Class | Installed Version | Repository Version | Description                                      |
|---|-------|-------------------|--------------------|--|
| <input type="checkbox"/> mingw-developer-tool...        | bin   |                   | 2013072300         | An MSYS Installation for MinGW Developers (meta) |
| <input checked="" type="checkbox"/> mingw32-base        | bin   | 2013072200        | 2013072200         | A Basic MinGW Installation                       |
| <input checked="" type="checkbox"/> mingw32-gcc-ada     | bin   |                   | 6.3.0-1            | The GNU Ada Compiler                             |
| <input checked="" type="checkbox"/> mingw32-gcc-fortran | bin   |                   | 6.3.0-1            | The GNU FORTRAN Compiler                         |
| <input checked="" type="checkbox"/> mingw32-gcc-g++     | bin   | 6.3.0-1           | 6.3.0-1            | The GNU C++ Compiler                             |
| <input checked="" type="checkbox"/> mingw32-gcc-objc    | bin   |                   | 6.3.0-1            | The GNU Objective-C Compiler                     |
| <input type="checkbox"/> msys-base                      | bin   |                   | 2013072300         | A Basic MSYS Installation (meta)                 |

Cette installation est directement reconnue par Visual studio code ce qui permet de lancer ou de déboguer des programmes en C++ directement dans VS Code.

#### *Visual Studio Community Edition (optional)*

Une autre alternative si vous faites des développements plus conséquents en C++ est d'installer le logiciel gratuit de Microsoft : **Visual Studio Community Edition**. C'est un outil fantastique pour faire des développements dans de nombreux langages comme C++, Python, C#, etc. L'installation de Visual Studio installe automatiquement l'environnement de développement C++ de Microsoft. L'installation est classique et n'est pas décrite ici.

Si vous voulez utiliser l'environnement de développement C++ directement dans VS Code il faut au préalable ouvrir un « Developer Command Prompt for VS » et lancer VS Code à partir de là en tapant

**VS Code** .

### **CMake Builder (optional)**

Ce logiciel complète l'installation de MinGW C++ et de Visual Studio. [CMake](#) est utilisé pour contrôler le processus de compilation du logiciel à l'aide de fichiers de configuration simples, indépendants de la plate-forme et du compilateur, et pour générer des fichiers **makefiles** et des espaces de travail natifs qui peuvent être utilisés dans l'environnement du compilateur de votre choix.

### **Interface graphique de bureau pour Git (optional)**

Il est agréable d'avoir une application graphique pour visualiser l'arborescence de ses repositories et de pouvoir exécuter les commandes Git/GitHub à partir de menu. Il existe de nombreux programmes pour faire cela, par exemple « [GitHub desktop](#) » ou « Git Gui », mais personnellement j'utilise depuis de nombreuses années « [Git Extensions](#) ».

### **Makefile (optional)**

**TODO**

## **Logiciels pour la documentation**

L'installation des logiciels pour la documentation est facultative mais il est utile pour :

- Générer de la documentation sur le code que vous écrivez,
- Vous aidez dans l'écriture et la mise au point de la documentation qui est exigée pour ajouter un composant à ESPHome.

### **Génération de documentation pour le code C++ (recommended)**

J'ai pour habitude de toujours documenter un minimum mon code source et pour cela j'utilise le générateur de documentation [Doxygen](#). Doxygen est un logiciel open source de génération de documentation de code pour de nombreux langages de programmation tels que C++, C, Python, etc. Il génère de la documentation directement à partir du code source, ce qui facilite la compréhension

des API, des fonctionnalités et des détails d'implémentation. Visual studio code utilise les mêmes balises que Doxygen ce qui permet de visualiser la documentation en temps réel, grâce à la technologie IntelliSense des classes, des méthodes, des paramètres, etc. Doxygen génère également des graphes tel que des **diagrammes d'héritage et d'association des classes** ce qui aide à la compréhension du code. J'installe également [Graphviz](#) pour la génération des graphiques et [Microsoft HTML Help Workshop](#) pour la génération de documents en HTML compressé (plus pratique que des centaines de fichiers HTML).

*Notez que les développeurs d'ESPHome utilisent Doxygen pour générer la documentation de l'API d'ESPHome (disponible à [ESPHome API](#)) mais malheureusement le code source d'ESPHome est très (mais vraiment très) peu documenté !*

### **Génération de la documentation d'un composant (recommended)**

La documentation des composants de la bibliothèque d'ESPHome est disponible directement depuis le site [ESPHome](#). Il existe une page (en html) par composant. Lorsque vous créez un nouveau composant il faut écrire une documentation. Cette documentation doit être écrite au format [reStructuredText](#) qui utilise [Sphinx](#) pour générer directement le fichier HTML à partir de votre texte. Pour vous aider à écrire, à vérifier, et à visualiser le rendu exacte de votre documentation il existe plusieurs solutions dans VS Code.

Personnellement j'ai installé dans VS Code le serveur de langage [esbonio](#). Il vous indique en temps réel si vous faites des erreurs lors de l'écriture de la documentation Sphinx, vous suggère les mots clés à utiliser et bien d'autres fonctionnalités comme de vous permettre de visualiser en temps réel le rendu HTML final dans VS Code.

Noter qu'il existe une autre extension populaire dans VS Code qui s'appelle [reStructuredText](#) mais je n'ai pas réussi à la faire fonctionner correctement et donc j'utilise Esbonio qui fonctionne très bien.

Avant de pouvoir l'utiliser cette extension vous devez avoir installé [Python](#). Ensuite dans VS Code vous installez les deux extensions : [esbonio](#) (qui installe Sphinx automatiquement) et « [reStructuredText Syntax highlighting](#) ».

### **Extensions de Visual Studio Code pour la documentation**

En plus de Esbonio/Sphinx il existe des centaines d'extensions VS Code qui peuvent être utiles pour générer de la documentation :

- DotUML, Doxygen Document Generator, ...

### **Création de diagrammes UML (optional)**

Il est intéressant de documenter les classes du code C++ avec des diagrammes UML. Pour cela je recommande d'installer le logiciel [Visual Paradigm Community Edition](#).

## Environnement de développement logiciel Dev Container

Cette section explique le développement dans VS code en utilisant le **Dev Container** d'ESPHome. L'avantage de développer dans le Dev Container est que vous n'avez pratiquement aucun logiciel à installer et que vous êtes dans le même environnement que les développeurs d'ESPHome. Pour utiliser cet environnement il n'est pas indispensable d'avoir des connaissances sur l'utilisation de Docker mais il est quand même recommandé d'avoir des notions de base. La section [références](#) de ce document pointe vers des articles que je recommande de lire.

Pendant la phase de **développement** je ne conseille pas l'utilisation de containers. Mais pendant la phase de **validation** finale du code il y a un certain nombre de logiciels (en particulier clang-tidy) que je n'ai pas réussi à faire fonctionner correctement dans l'environnement Windows. Et donc dans cette phase je préconise l'utilisation du **Dev Containeur** d'ESPHome dans VS Code. Ceci permet de faire *tourner tous les tests de validation localement*. Nous verrons cela plus en détail dans le chapitre concernant la « procédure de soumission de son code ».

Pour utiliser le Dev Containeur d'ESPHome dans VS Code il faut préalablement installer un certain nombre de programmes et d'extensions VS Code.

### WSL2 (must have)

Normalement [l'installation est extrêmement simple](#) quand tout se passe bien.

Il suffit d'ouvrir un terminal **PowerShell** en tant qu'administrateur et de taper la commande :

```
wsl --install
```

Par contre si cela ne se passe pas bien, et c'est souvent le cas, alors cela peut devenir compliqué. Pour résoudre les problèmes je vous recommande de [lire cet article de Microsoft](#).

### Docker Desktop (must have)

une fois que vous avez installé **WSL2** [l'installation de Docker Desktop](#) devrait se faire très simplement. Il suffit [d'exécuter le programme d'installation](#) et de suivre les instructions.

### VS Code extensions

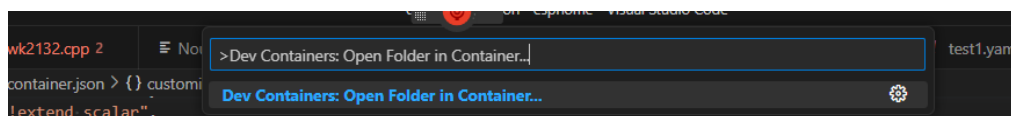
Dans VS Code vous devez installer les extensions : **Dev Container**.

L'extension VS Code **Dev Containers** vous permet d'utiliser un conteneur comme environnement de développement complet en profitant de l'ensemble des fonctionnalités de Visual Studio Code. Les fichiers de l'espace de travail sont montés à partir du système de fichiers local dans le conteneur. Les extensions sont installées et exécutées dans le conteneur, où elles ont un accès complet aux outils, à la plate-forme et au système de fichiers. Cela signifie que vous pouvez changer votre environnement de développement pour celui d'ESPHome.

### Utilisation du Dev Containeur d'ESPHome

Le répertoire d'ESPHome contient toutes les informations nécessaires pour pouvoir utiliser ESPHome dans un Dev Container. Cela donne accès à un environnement Linux Debian prédéfini ou pratiquement tout ce dont vous avez besoin a déjà été préinstallé par les équipes d'ESPHome. A la première ouverture du Dev Containeur VS Code va vérifier que tout l'environnement nécessaire est présent et à jour sinon il fera le nécessaire.

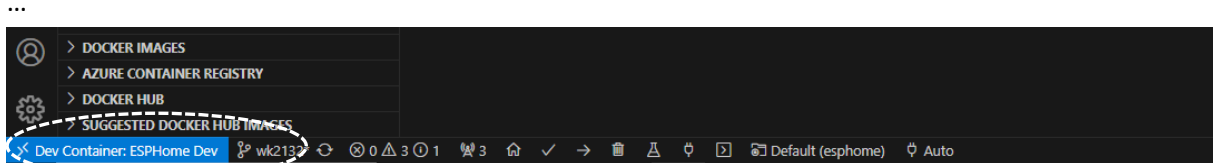
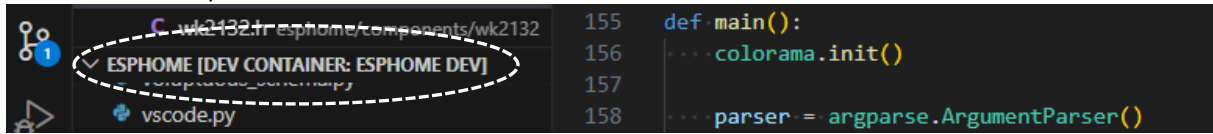
Pour utiliser le Dev Containeur : lancez en premier **Docker Desktop** et lancez **VS Code**, exécutez la commande « **Dev Containers : Open folder in container...** » à partir de la palette de commandes (F1), et sélectionnez le dossier d'ESPHome.



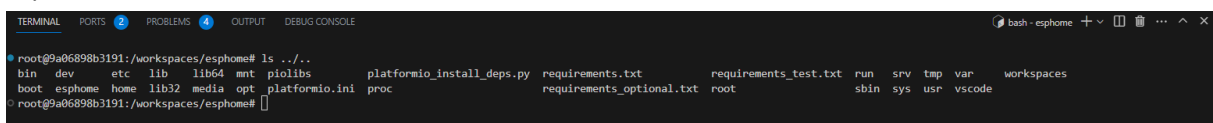
La fenêtre VS Code se recharge et commence à construire le conteneur de développement. Vous ne devez construire le conteneur de développement que la première fois que vous l'ouvrez ; l'ouverture

du dossier après la première construction réussie sera beaucoup plus rapide. Une fois la construction terminée, VS Code se connecte automatiquement au conteneur de développement.

Vous devez voir que VSCode utilise le conteneur d'ESPHome.



Si vous ouvrez une fenêtre « terminal » vous êtes maintenant dans un environnement Linux avec le répertoire d'ESPHome monté dans `/workspaces/esphome`.



Pour compléter l'installation de ESPHome tapez la commande suivante :

```
script/setup
```

L'exécution du **script de setup** peut prendre beaucoup de temps, donc soyez patient.

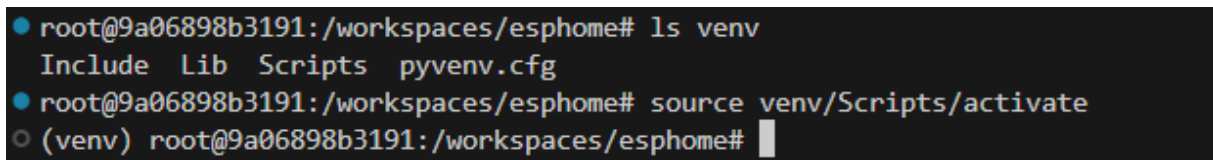
À la fin pour vérifier qu'ESPHome est correctement installé tapez

```
esphome version
```

Noter que la documentation vous dit d'exécuter la commande

```
source venv/Script/activate
```

Mais en réalité, comme nous sommes dans un conteneur il n'y a pas lieu de lancer un environnement venv.



Noter qu'après la première commande le prompt indique que vous êtes maintenant dans l'environnement **venv**.

Voilà ESPHome est installé et configuré sous Linux. Nous verrons dans le chapitre procédure GitHub comment l'utiliser.

Noter que le fichier `.devcontainer/devcontainer.json` inclus dans votre projet indique à VS Code comment accéder au conteneur de développement avec une pile d'outils et d'exécutions bien définie.

## Installation de la suite LLVM dans ESPHome DevContainer

Il manque un certain nombre d'utilitaires liés à **clang** (au minimum **clang-tidy-14** et **clang-apply-replacements-14**) dans le container fourni par ESPHome. Pour pallier à ce problème le plus simple est d'installer la suite **LLVM** à l'aide du script qu'il fournisse.

Le Linux que l'on exécute dans le Dev Container est un Debian GNU/Linux 11. Pour le vérifier vous pouvez taper les commandes suivantes :

```
cat /etc/issue
cat /etc/os-release
```

Et pour obtenir le shell d'installation de LLVM tapez les commandes :

```
apt update
apt upgrade
apt install wget
wget -O - https://apt.llvm.org/llvm.sh > llvm.sh
chmod +x llvm.sh
apt install lsb-release wget software-properties-common gnupg
```

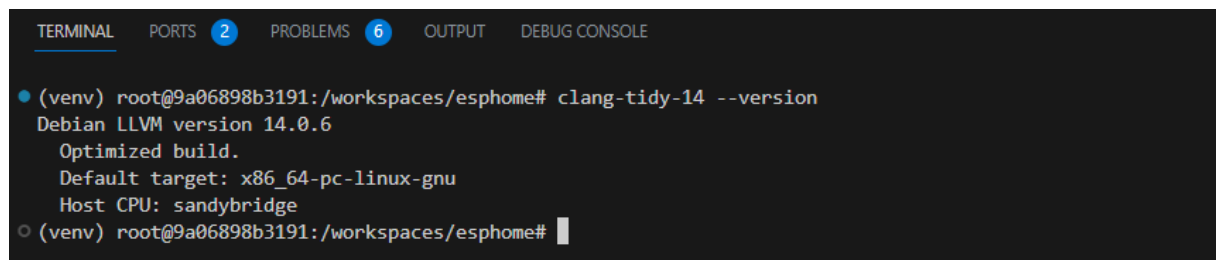
Si on veut la version 14 qui est celle utilisée dans ESPHome et ce qui évite de modifier les scripts, on tape la commande :

```
./llvm.sh 14 all
```

Noter que le script détecte que l'on est sur une version Debian et applique les workarounds nécessaires à cette version (voir <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=1038383> ).

Pour vérifier l'installation on tape :

```
Clang-tidy-14 --version
```



```
TERMINAL  PORTS 2  PROBLEMS 6  OUTPUT  DEBUG CONSOLE

• (venv) root@9a06898b3191:/workspaces/esphome# clang-tidy-14 --version
Debian LLVM version 14.0.6
Optimized build.
Default target: x86_64-pc-linux-gnu
Host CPU: sandybridge
○ (venv) root@9a06898b3191:/workspaces/esphome#
```

# Personnalisation (réglages) des logicielles

## Git

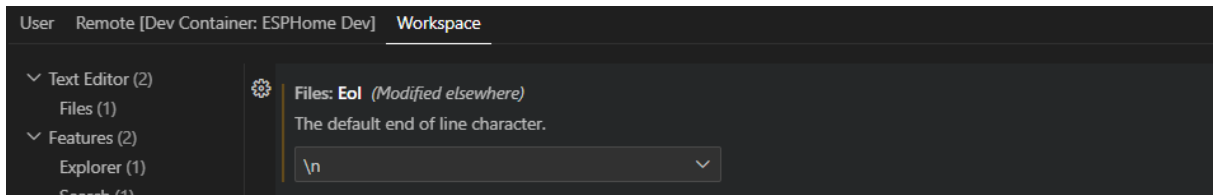
Nous avons déjà vu qu'il faut spécifier les variables `user name` et `user email`.

Il est aussi important de vérifier comment Git va traiter **les fins de ligne**. Je recommande, surtout si vous utilisez les conteneurs, de régler cette option sur le mode **Unix** :

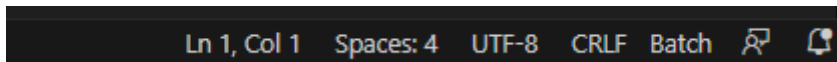
```
git config --global core.autocrlf input
```

**Attention si vous changez le mode cela ne s'applique que sur les nouveaux fichiers que vous ajoutez mais pas sur les fichiers existants.**

In VS Code : allez dans **setting** et chercher **Eol**. Le régler sur `\n`



Quand un fichier est ouvert il indique en bas à droite le mode **Eol** de ce fichier :



Ici on est en mode CRLF (mode Windows). Si vous voulez le passer en mode LF (mode Unix) cliquez sur ce texte et changez directement le mode.

Noter que normalement vous ne devriez pas avoir à vous soucier de ce problème car ESPHome possède un fichier **.gitattributes** qui contient :

```
# Normalize line endings to LF in the repository
* text eol=lf
*.png binary
```

Si vous voulez fonctionner dans les deux environnements vous pouvez modifier ce fichier avec :

```
# Normalize line endings to LF in the repository
* text=auto eol=lf
*.{cmd,[cC][mM][dD]} text eol=crlf
*.{bat,[bB][aA][tT]} text eol=crlf
*.png binary
```



## VS Code

Il est recommandé de faire un certain nombre de réglages dans VS Code.

Il y a des réglages communs à tous les projets comme par exemple la sauvegarde automatique des fichiers (perso je le règle sur sauvegarder quand vous changer de fenêtre). Ces réglages sont stockés dans le fichier :

```
C:\Users\<YourLoginName>\AppData\Roaming\Code\User\settings.json
```

Ou <YourLoginName> est votre nom de login.

Voilà par exemple mon fichier de setting

```
{
  "C_Cpp.default.compilerPath": "C:\\MinGW\\bin\\gcc.exe",
  "cSpell.userWords": [
    "codegen",
    "CODEOWNERS",
    "GPIO",
    "uart",
    "apiref",
    "Datasheet",
    "ghedit",
    "GPIO",
    "Mbit",
    "pmsx",
    "pullup",
    "UART"
  ],
  "cSpell.ignoreWords": [
    "toctree"
  ],
  "cSpell.diagnosticLevel": "Hint",
  "cSpell.language": "en,en-US",
  // "code-runner.runInTerminal": true,
  "editor.renderWhitespace": "all",
  "editor.formatOnSave": true,
  "editor.guides.bracketPairs": "active",
  "files.autoSave": "onFocusChange",
  "[python]": {
    "editor.formatOnType": true,
    "editor.defaultFormatter": "ms-python.black-formatter"
  },
}
```

Et il y a les réglages qui sont propres au projet.

Pour le projet ESPHome il n'y a rien de particulier à noter

Pour le projet ESPHome-docs je spécifie certaines valeurs pour que l'extension esbonio construise les fichiers HTML dans un répertoire `_buid` (spécifié dans `.gitignore`). Ce fichier est situé dans le projet :

```
.\.vscode\settings.json
```

Voilà son contenu :

```
{
  "esbonio.sphinx.confDir": "",
  "esbonio.sphinx.buildDir": "C:\\Work\\Projects\\esphome-docs\\_buid",
  "esbonio.sphinx.configOverrides": {},
}
```

## Conseils et astuces...

### Raccourcis utiles dans VS Code

Ouverture/fermeture fenêtres :

- Afficher la Palette : F1 ou Ctrl Shift p
- Afficher Settings : Ctrl ,
- Afficher panel : Ctrl j ou Ctrl `
- Afficher sidebar : Ctrl b
- afficher rechercher : Ctrl f – remplacer : Ctrl h
- Find symbols in file: Ctrl Shift o – in workspace: Ctrl T
- afficher suggestions : Ctrl Espace ou Ctrl I
- montrer les actions recommandées : Ctrl ;

Déplacement curseur :

- Jump to definition: F12 – Peek definition: Alt F12
- Jump matching bracket : Ctrl Shift \*
- Jump to line : Ctrl G

Edition :

- Déplacer une ligne : Alt Haut ou Bas
- Commenter / dé-commenter : Ctrl /
- Indentation : Tab – indentation arrière : Shift Tab
- Supprimer une ligne : Ctrl Shift K
- Copier une ligne : Shift Alt Haut ou Bas
- Insérer un bloc de commentaire : Shift Alt A
- formater le document : Shift Alt F – la sélection : Ctrl K Ctrl F

Sélections :

- Sélectionner une ligne : Click numéro de ligne
- Ajouter une ligne à la sélection (utile pour line-staging) : Alt Click numéro de ligne
- Ajouter à la sélection le prochain même mot : Ctrl D – tous les mêmes mots : Ctrl Shift L
- Sélection d'une boîte = Alt Shift Gauche ou Droite
- Multi curseurs : Alt Click
- Colonne de curseurs : Ctrl Alt Haut ou Bas
- inclusion block (sélection étendue): Alt Shift Droite ou Gauche

## Références

### DevContainer

- [GitHub Codespaces](#) (GitHub)
- [Introduction to dev containers](#) (GitHub)
- [Developing inside a Container](#) (MS)
- [Remote Development Tips and Tricks](#) (MS)

### Docker

- [Install Docker Desktop on Windows](#) (docker)
- [Docker in Visual Studio Code](#) (MS)

### ESPHome

- [Contributing to ESPHome](#) (ESPHome)
- [Generic Custom Component](#) (ESPHome)
- [Custom Sensor Component](#) (ESPHome)
- [Submit your work](#) (HA)
- [Catching up with Reality](#) (HA)

### Espressif / Arduino – Programming/Reference

- [ESP-IDF Programming Guide](#)
- [ESP32 Technical Reference Manual](#)
- [Arduino Language Reference](#)
- [Arduino MultiSpeed I2C Scanner](#)
- [I<sup>2</sup>C Scanner code](#)
- [Working with I2C Devices](#)
- [How many Devices can you Connect to the I2C Bus?](#)
- [PlatformIO Tutorial for Arduino and ESP – First Steps with Visual Studio Code](#) (Video EN)
- [Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266](#)
- [Comprendre le bus I2C](#) (Video FR)

### Git/GitHub

- [Contributing to projects](#) (GitHub)
- [Git and GitHub for Beginners - Crash Course](#) (Video EN)
- [Working with GitHub in VS Code](#)
- [Git](#)
- [Using Git with VS Code and PlatformIO](#) (Video EN)

### VS Code

- [Snippets in Visual Studio Code](#) (MS)
- [C/C++ for Visual Studio Code](#)
- [Configure VS Code for Microsoft C++](#)
- [How to run a C program in Visual Studio Code?](#)
- [VS Code documentation](#)
- [Visual Studio Code Crash Course](#) (Video EN)

### WSL

- [Installer Linux sur Windows avec WSL](#) (MS)
- [Troubleshooting Windows Subsystem for Linux](#)
- [Set up a WSL development environment](#)