Development of ESPHome Components
Part 2: Software Development Environment

Windows software development environment

*We are now going to see how to install the different software needed to do development directly under **Windows**. Compared to using **containers**, this way of doing things is easy to understand and the performance is better. However, there are some operations that I couldn't get to work properly on Windows (like clang-tidy) that's why I have a section of this document that shows how to do development using the Dev Container of ESP Home. We'll see the usefulness of this when I describe the code submission process in GitHub.*

Each developer has their own habits and tools and therefore what I present here corresponds to my preferences. The key pieces you will need are:

- A version control system: I chose **`Git`**
- A code editor: I chose to use **`Visual Studio Code`** (VS Code).
- An integrated development environment (IDE): I chose **`PlatformIO`**.

This combination brings many functionalities like the completion and the intelligent checking of the C/C++ code during the writing, the integrated debugging, the management of the dependencies…

Also note that you need a GitHub account to integrate your component into ESPHome.

Since the software I have chosen is widely used, there are already a lot of tutorials that explain in detail how to install and use it. I will therefore content myself with giving a brief presentation.

Software for code development

Here is the list of software I have installed on my development machine to create or modify ESPHome components. I describe here the installation on a PC with Windows, but this software is available on Linux and Mac.

Git - Distributed version control system (must have)

A version control system is essential when doing code development. To be able to integrate your code in ESPHome it is imperative to install and use the software Git.

After installation you must configure Git by filling in at least the variables “user.name” and “user.email” which will allow you to be identified. To do this, you can launch a “Git bash” and execute the following commands:

```
git config --global user.name “Your Name”
git config --global user.emailyouremail@yourdomain.com
git config --list
```

Visual Studio Code (VS Code) - Code editor (must have)

It is a very popular open-source code editor for its versatility and features, including Git integration, an integrated terminal, debugging tools, IntelliSense, and the existence of thousands of extensions. We will customize VS Code to create a development environment adapted to ESPHome which will make it possible to make checks and automatic formatting of your code in order to follow the rules imposed by ESPHome.

PlatformIO IDE - Integrated Development Environment (must have)

`PlatformIO`, which is compatible with many microcontrollers, provides a unified development environment for multiple programming languages such as C++, Python. It offers debugging tools, library management, unit testing, and loads of other features to manage a project. This is the environment that is used by the ESPHome development teams. PlatformIO is installed as an extension of VS Code. The installation is done in VS Code it is standard and not described here.

Installing Visual Studio Code Extensions

There are a lot of extensions for VS Code. Some of these extensions will be automatically installed by PlatformIO when you import the ESPHome repository into your environment.

I recommend installing the following extensions:

- ***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

Installing ESPHome (must have)

It is essential to install ESPHome to be able to check, compile, and upload your component directly from your development machine.

Nous verrons plus tard le détail du processus complet pour créer et soumettre un nouveau composant. Mais en simplifiant vous devez faire un fork du repository d'ESPHome sur votre compte GitHub et puis le cloner localement sur votre machine de développement.

In VS Code open the directory that contains ESPHome. PlatformIO will immediately initialize the project and install any extensions that are needed. It can take quite a while.

This is where things get a little complicated, because the Setting Up Development Environment" documentation doesn't apply to a Windows environment, but to a Unix environment. So you don't need to use the script/setup, which is replaced by the command:

```
pip install -e .
```

This command will install the various utilities necessary for the proper functioning of ESPHome and if all goes well you should see a message like this:

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

To verify that ESPHome is running correctly type the following command:

```
esphome version
```

Congratulations, you will now be able to compile the code for your component and send it to your development board directly from your system, including by launching a hardware debugging session.

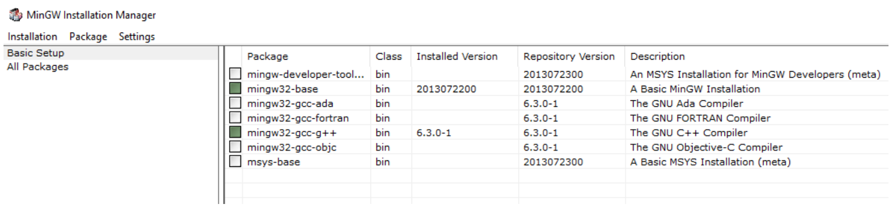
Python (recommended)

It is recommended (but not essential) to install Python. During installation verify that Python is available from any location by changing your system's PATH variable. The installation is classic and not described here.

C++ compile & debug (optional)

MinGW

This suite of applications allows you to compile and run programs written in C++. There is no need to install it if you don't want to do C++ development outside of ESPHome. Personally, I use MinGW which I install from mingw-get-setup by choosing the packages “mingw32-base” and “mingw32-gcc-g++”.



	Package	Class	Installed Version	Repository Version	Description
☐	mingw-developer-tool...	bin		2013072300	An MSYS Installation for MinGW Developers (meta)
🟩	mingw32-base	bin	2013072200	2013072200	A Basic MinGW Installation
☐	mingw32-gcc-ada	bin		6.3.0-1	The GNU Ada Compiler
☐	mingw32-gcc-fortran	bin		6.3.0-1	The GNU FORTRAN Compiler
🟩	mingw32-gcc-g++	bin	6.3.0-1	6.3.0-1	The GNU C++ Compiler
☐	mingw32-gcc-objc	bin		6.3.0-1	The GNU Objective-C Compiler
☐	msys-base	bin		2013072300	A Basic MSYS Installation (meta)

This installation is directly recognized by Visual studio code which allows you to launch or debug C++ programs directly in VS Code.

Visual Studio Community Edition (optional)

Another alternative if you are doing more substantial development in C++ is to install Microsoft's free software: Visual Studio Community Edition. It is a fantastic tool for doing development in many languages like C++, Python, C#, etc. Installing Visual Studio automatically installs Microsoft's C++ development environment. The installation is standard and is not described here.

If you want to use the C++ development environment directly in VS Code, you must first open a “Developer Command Prompt for VS” and launch VS Code from there by typing

```
VSCode.
```

CMake Builder (optional)

This software completes the installation of MinGW C++ and Visual Studio. CMake is used to control the software compilation process using simple, platform- and compiler-independent configuration files, and to generate native makefiles and workspaces that can be used in the environment of the compiler of your choice.

Desktop GUI for Git (optional)

It is nice to have a graphical application to visualize the tree structure of its repositories and to be able to execute Git/GitHub commands from the menu. There are many programs to do this, for example GitHub desktop or Git Gui, but personally I've been using for many years” Git Extensions ".

Software for documentation

Installing documentation software is optional but useful for:

- Generate documentation on the code you write,
- You help in writing and debugging the documentation that is required to add a component to ESPHome.

Generating documentation for C++ code (recommended)

I usually always document a minimum of my source code and for that I use the documentation generator Doxygen. Doxygen is an open source code documentation generation software for many programming languages such as C++, C, Python, etc. It generates documentation directly from the source code, making it easy to understand APIs, features, and implementation details. Visual studio code uses this information to visualize the documentation of classes, methods, parameters, etc. in real time, thanks to IntelliSense technology. Doxygen also generates graphs such as inheritance and class association diagrams which help in understanding the code. I also install Graphviz for generating graphics and Microsoft HTML Help Workshop for generating documents in compressed HTML (more practical than having hundreds of HTML files).

> ***Note that the ESPHome developers use Doxygen to generate the ESPHome API documentation (ESP Home API) but unfortunately the source code of ESPHome is very (but really very) poorly documented!***

Generating component documentation (recommended)

The documentation of the components of the ESPHome library is available directly from the site ESP Home. There is one page (in html) per component. When you create a new component, you have to write documentation. This documentation should be written in the format reStructuredText who uses Sphinx to directly generate the HTML file from your text. To help you write, check, and visualize the exact rendering of your documentation there are several solutions in VS Code.

Personally I installed in VS Code the language server esbonio.It tells you in real time if you make mistakes when writing Sphinx documentation, suggests the keywords to use and many other features like allowing you to visualize in real time the final HTML rendering in VS Code .

Note that there is another popular extension in VS Code called reStructuredTextbut I couldn't get it to work properly so I use Esbonio which works fine.

In VS Code you install the two extensions:esbonio (which installs Sphinx automatically) and " reStructuredText Syntax highlighting ".

Visual Studio Code extensions for documentation

In addition to Esbonio/Sphinx, there are hundreds of VS Code extensions that can be useful for generating documentation:

- DotUML, Doxygen Document Generator, …

Creation of UML diagrams (optional)

It is interesting to document the classes of C++ code with UML diagrams. For this I recommend installing the software Visual Paradigm Community Edition.

Dev Container software development environment

This section explains development in VS code using ESPHome's Dev Container. The advantage of developing in the Dev Container is that you have virtually no software to install and are in the same environment as ESPHome developers. To use this environment, basic knowledge of Docker is not essential, but is recommended. The references section of this document points to articles I recommend reading.

During the development phase, I do not recommend the use of containers. But during the final code validation phase there are a number of pieces of software (especially clang-tidy) that I couldn't get to work properly in the Windows environment. And so, in this phase I advocate the use of ESPHome's Dev Container in VS Code. This allows all validation tests to be run locally. We will see this in more detail in the chapter on the “code submission procedure”.

To use the ESPHome Dev Container in VS Code, you must first install a number of VS Code programs and extensions.

WSL2 (must have)

Normally installation is extremely simple when everything is going well.

Just open a PowerShell terminal as administrator and type the command:

```
wsl --install
```

On the other hand, if it does not go well then it can become complicated. To solve the problems, I recommend that you read this Microsoft article.

Docker Desktop (must have)

once you have installed WSL2 installing Docker Desktop should be done very simply. It is enough run the installer and follow the instructions.

VS Code plugins

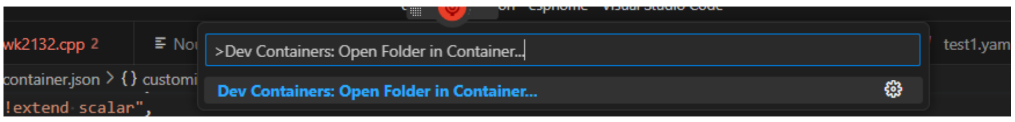
In VS Code you need to install the extensions: Dev Container.

The VS Code Dev Containers extension allows you to use a container as a complete development environment taking advantage of the Visual Studio Code feature set. Workspace files are mounted from the local file system into the container. Extensions are installed and run in the container, where they have full access to the tools, platform, and file system. This means that you can change your development environment to that of ESPHome.

Using the ESPHome Dev Container

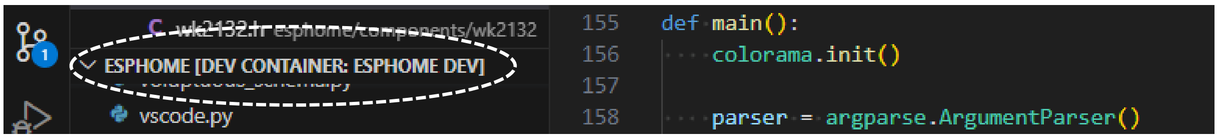
The ESPHome directory contains all the information needed to be able to use ESPHome in a Dev Container. This gives access to a predefined Debian Linux environment where almost everything you need has already been preinstalled by the ESPHome teams. At the first opening of the Dev Container VS Code will check that all the necessary environment is present and up to date otherwise it will do the necessary.

To use the Dev Container: first launch Docker Desktop and launch VS Code, run the command, “`Dev Containers: Open folder in container...`” from the command palette (F1), and select the ESPHome folder.


wk2132.cpp 2
>Dev Containers: Open Folder in Container...
test1.yam
container.json > {} customi
!extend scalar",
Dev Containers: Open Folder in Container...

The VS Code window reloads and begins building the development container. You only need to build the development container the first time you open it; opening the folder after the first successful build will be much faster. When the build is complete, VS Code automatically connects to the development container.

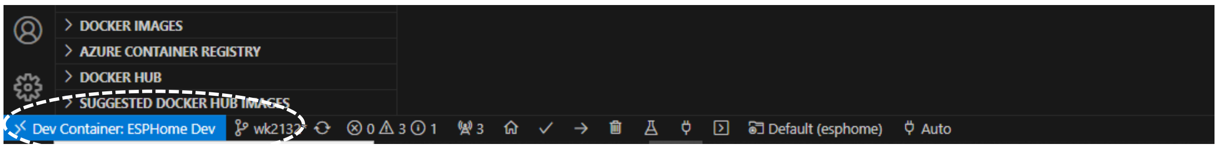
You should see that VSCode uses the ESPHome container.



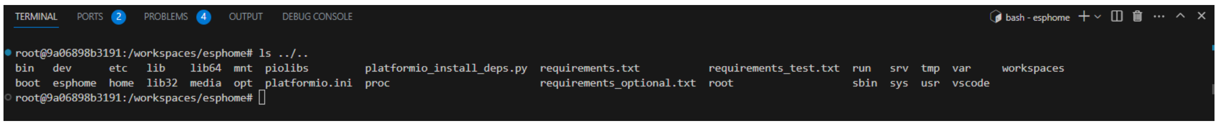
```
def main():
    colorama.init()

    parser = argparse.ArgumentParser()
```

...



If you open a “terminal” window, you are now in a Linux environment with the ESPHome directory mounted in /workspaces/esphome.



```
root@9a06898b3191:/workspaces/esphome# ls ../..
bin   dev      etc   lib    lib64  mnt  piolibs        platformio_install_deps.py  requirements.txt           requirements_test.txt  run   srv  tmp  var     workspaces
boot  esphome  home  lib32  media  opt  platformio.ini  proc                        requirements_optional.txt  root                   sbin  sys  usr  vscode
root@9a06898b3191:/workspaces/esphome#
```

To complete the installation of ESPHome type the following command:

```
script/setup
```

Running the setup script can take a long time, so be patient.

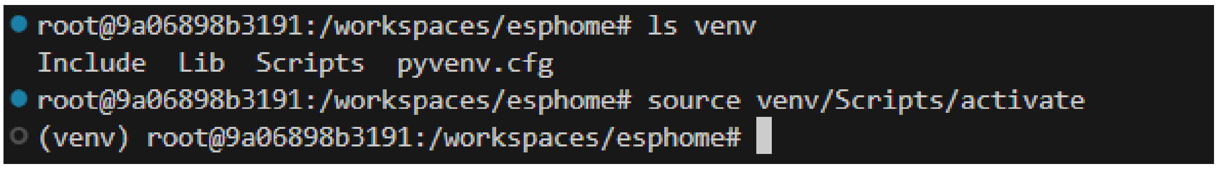
At the end to check that ESPHome is correctly installed type

```
esphome version
```

Note that the documentation tells you to run the command

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

But in reality, as we are in a container there is no need to launch a venv environment.



```
root@9a06898b3191:/workspaces/esphome# ls venv
Include  Lib  Scripts  pyvenv.cfg
root@9a06898b3191:/workspaces/esphome# source venv/Scripts/activate
(venv) root@9a06898b3191:/workspaces/esphome#
```

Note that after the first command the prompt indicates that you are now in the venv environment.

That's ESPHome installed and configured on Linux. We will see in the GitHub procedure chapter how to use it.

Note that the .devcontainer/devcontainer.json file included in your project tells VS Code how to access the development container with a well-defined stack of tools and runtimes.

Installing LLMV Suite in ESPHome DevContainer

A number of clang-related utilities (at least clang-tidy-14 and clang-apply-replacements-14) are missing from the container provided by ESPHome. To overcome this problem, the easiest way is to install the following **LLVM** using the script he provides.

The Linux running in the Dev Container is Debian GNU/Linux 11. To verify this, you can type the following commands:

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

And to get the LLVM installation shell type the commands:

```
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
```

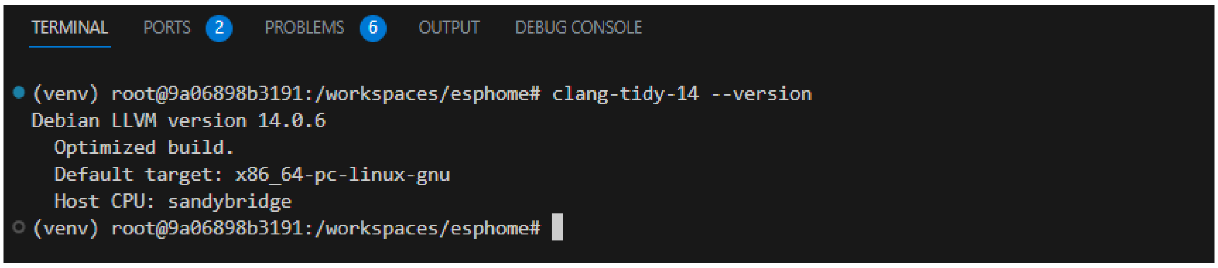
If we want version 14 which is the one used in ESPHome and which avoids modifying the scripts, we type the command:

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

Note that the script detects that you are on a Debian version and apply the necessary workarounds to this version (seehttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=1038383).

To verify the installation, type:

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



```
(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#
```

Customization (settings) of software

Git

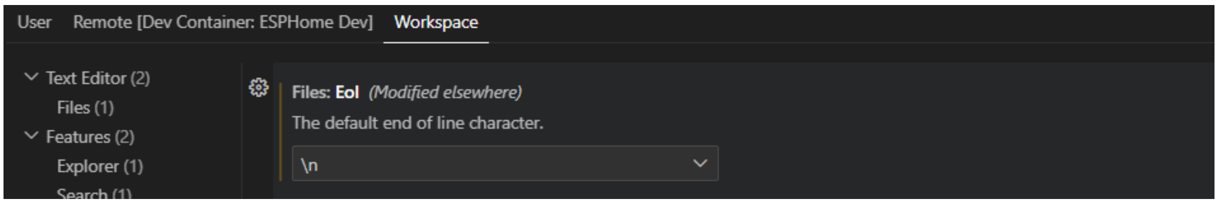
We have already seen that it is necessary to specify the variables user name and user email.

It is also important to check how Git will handle line endings. I recommend, especially if you use containers, to set this option to Unix mode:

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

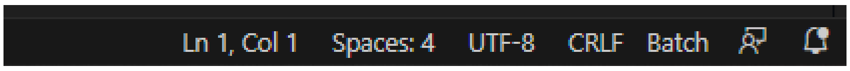
Be careful if you change the mode this only applies to new files that you add but not to existing files.

In VS Code: go to setting and search for Eol. Set it to \n



Files: Eol *(Modified elsewhere)*
The default end of line character.
\n

When a file is opened, it indicates at the bottom right the Eol mode of this file:



Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Batch

Here we are in CRLF mode (Windows mode). If you want to switch it to LF mode (Unix mode) click on this text and change the mode directly.

Note that normally you shouldn't have issues because ESPHome has a .gitattributes file that contains:

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

If you want to work in both environments, you can modify this file with:

```
# 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
```

VSCode

It is recommended to make a number of adjustments in VS Code.

There are settings common to all projects such as automatic saving of files (personally I set it to save when you change windows). These settings are stored in the file:

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

Where <YourLoginName> is your login name.

Here is for example my settings file

```
{
“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”
},
}
```

And there are settings that are specific to the project.

For the ESPHome project, there is nothing special to note

For the ESPHome-docs project, I specify some values for the esbonio extension to build the HTML files in a _buid directory (specified in .gitignore). This file is located in the project:

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

Here is its content:

```
{
“esbonio.sphinx.confDir”: “”,
“esbonio.sphinx.buildDir”: “C:\\Work\\Projects\\esphome-docs\\_build”,
“esbonio.sphinx.configOverrides”: {},
}
```

Tips and tricks…

Useful Shortcuts in VS Code

Opening/closing windows:

- Show Palette: F1 or Ctrl Shift p
- Show Settings: Ctrl ,
- Show panel: Ctrl j or Ctrl ù
- Show sidebar   Ctrl b
- show find: Ctrl f – replace: Ctrl h
- Find symbols in file: Ctrl Shift o – in workspace: Ctrl T
- show suggestions: Ctrl Space or Ctrl I
- show recommended actions: Ctrl;

Cursor movement:

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

Editing :

- Move a row: Alt Up or Down
- Comment / uncomment: Ctrl /
- Indent: Tab – back indent: Shift Tab
- Delete a row: Ctrl Shift K
- Copy a line: Shift Alt Up or Down
- Insert comment block: Shift Alt A
- format the document: Shift Alt F – the selection: Ctrl K Ctrl F

Selections:

- Select a line: Click line number
- Add line to selection (useful for line-staging): Alt Click line number
- Add to selection the next same word: Ctrl D – all the same words: Ctrl Shift L
- Selection of a box = Alt Shift Left or Right
- Multi Cursors: Alt Click
- Cursor column: Ctrl Alt Up or Down
- inclusion block (extended selection): Alt Shift Right or Left

References

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)

ESP Home

- Contributing to ESP Home(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^{2}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
- Understanding the I2C bus (Video EN)

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)

VSCode

- 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

WSL

- Install Linux on Windows with WSL(MS)
- Troubleshooting Windows Subsystem for Linux
- Set up a WSL development environment