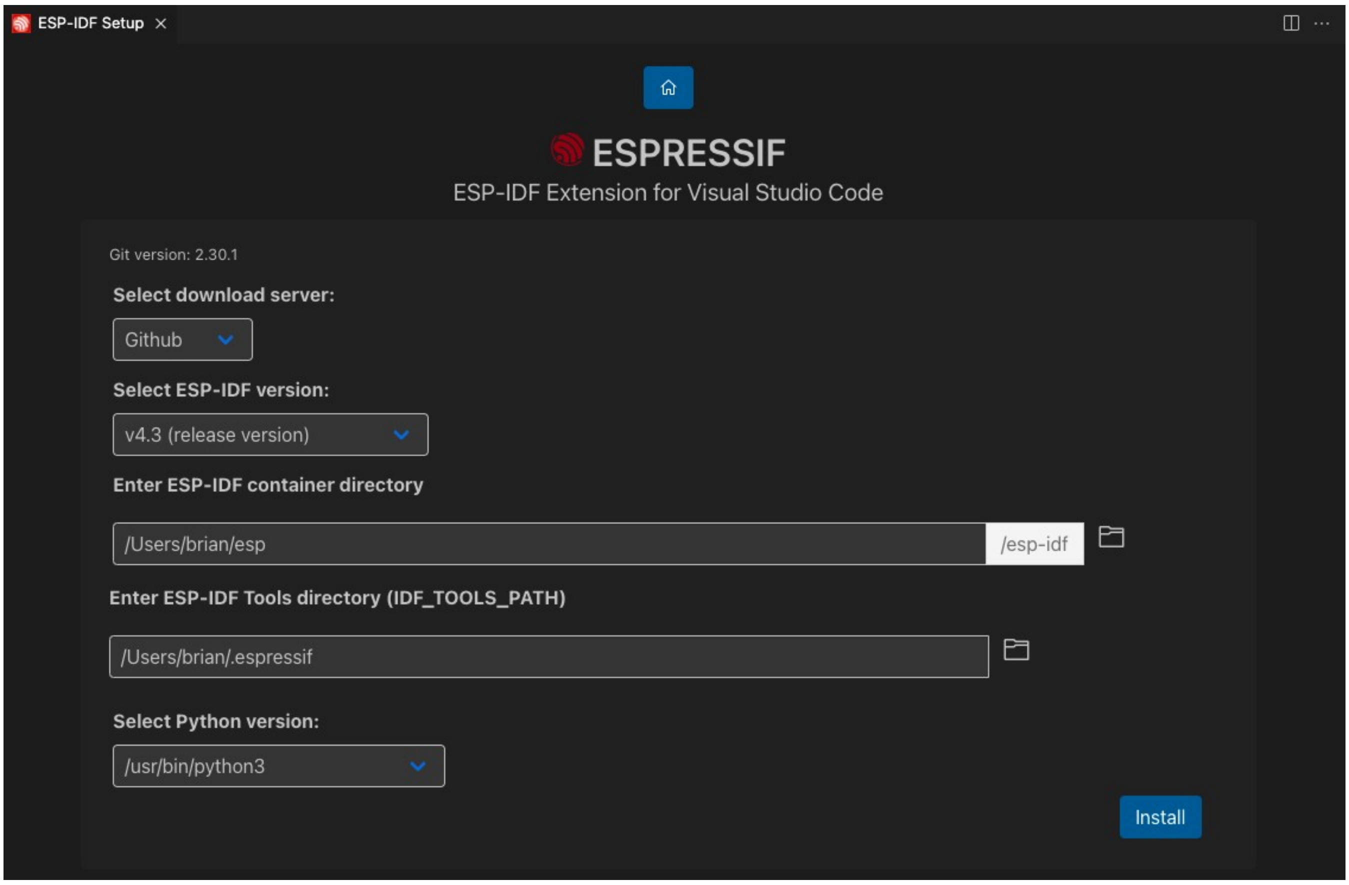
How to use

3.Install the extension.

4.(OPTIONAL) Press `F1` and type **ESP-IDF: Select where to Save Configuration Settings**, which can be User Settings (global), Workspace Settings or Workspace Folder Settings. Default is User settings.

> **NOTE:** Please take a look at Working with multiple projects for more information.

5.In Visual Studio Code, select menu "View" and "Command Palette" and type `configure esp-idf extension`. After, choose the **ESP-IDF: Configure ESP-IDF Extension** option. You can also choose where to save settings in the setup wizard.



Git version: 2.30.1

Select download server: Github

Select ESP-IDF version: v4.3 (release version)

Enter ESP-IDF container directory: /Users/brian/esp /esp-idf

Enter ESP-IDF Tools directory (IDF_TOOLS_PATH): /Users/brian/.espressif

Select Python version: /usr/bin/python3

8.Choose **Express** and select the download server:

•Espressif: Faster speed in China using Espressif Download servers links.

•Github: Using github releases links.

9.Pick an ESP-IDF version to download or the `find ESP-IDF in your system` option to search for existing ESP-IDF directory.

10.Choose the location for ESP-IDF Tools (also known as `IDF_TOOLS_PATH`) which is `$HOME\.espressif` on MacOS/Linux and `%USERPROFILE%\.espressif` on Windows by default.

11. If your operating system is Linux or MacOS, choose the python executable to create ESP-IDF virtual environment.

> **NOTE:** Windows users don't need to select a python executable since it is part of the setup. **NOTE:** Make sure that `IDF_TOOLS_PATH` doesn't have any spaces to avoid any build issues. Also make sure that `IDF_TOOLS_PATH` is not the same directory as `IDF_PATH`.

12. The user will see a page showing the setup progress status showing ESP-IDF download progress, ESP-IDF Tools download and install progress as well as the creation of a python virtual environment.
13. If everything is installed correctly, the user will see a message that all settings have been configured. You can start using the extension. Otherwise check the Troubleshooting section if you have any issues.
14. Press `F1` and type **ESP-IDF: Show Examples Projects** to create a new project from ESP-IDF examples. Select ESP-IDF and choose an example to create a new project from.
15. (OPTIONAL) Configure the `.vscode/c_cpp_properties.json` as explained in C/C++ Configuration.

Note: For code navigation the Microsoft C/C++ Extension or Clangd extension can be used for C/C++ language support. By default, projects created with **ESP-IDF: Create Project from Extension Template** or **ESP-IDF: Show Examples Projects** include a template for Microsoft C/C++ extension `c_cpp_properties.json` configuration file and doesn't need to be configured. Run **ESP-IDF: Run idf.py reconfigure task** to generate the compile_commands.json file so language support works.

16. Set the serial port of your device by pressing `F1`, typing **ESP-IDF: Select Port to Use:** and choosing the serial port your device is connected.
17. Select an Espressif target (esp32, esp32s2, etc.) with the **ESP-IDF: Set Espressif Device Target** command.
18. Use the **ESP-IDF: Select OpenOCD Board Configuration** to choose the openOCD configuration files for the extension openOCD server.
19. Next configure your ESP-IDF project by pressing `F1` and typing **ESP-IDF: SDK Configuration Editor** command (`CTRL E G` keyboard shortcut) where the user can modify the ESP-IDF project settings. After all changes are made, click save and close this window.
20. When you are ready, build your project by pressing `F1` and typing **ESP-IDF: Build your Project**.
21. Flash to your device by pressing `F1` and typing **ESP-IDF: Select Flash Method and Flash** to select either `UART`, `DFU` or `JTAG` depending on your serial connection.

NOTE: You can also use the **ESP-IDF: Flash (UART) your Project** or **ESP-IDF: Flash (with JTag)** directly.

22. Start a monitor by pressing `F1` and typing **ESP-IDF: Monitor Device** which will log the device activity in a Visual Studio Code terminal.

23. To make sure you can debug your device, select your board configuration by pressing `F1` and typing **ESP-IDF: Select OpenOCD Board Configuration**. You can test the connection by pressing `F1` and typing **ESP-IDF: OpenOCD Manager**. The output is shown in the menu `View` -> `Output` and choose `ESP-IDF` from the dropdown list.

    NOTE: The user can start or stop the OpenOCD from Visual Studio Code using the **ESP-IDF: OpenOCD Manager** command or from the `OpenOCD Server (Running | Stopped)` button in the visual studio code status bar.

24. If you want to start a debug session, just press `F5` (make sure you had at least build, flash and openOCD is connecting correctly so the debugger works correctly).

Check the Troubleshooting section if you have any issues.

Tutorials

1. Install and setup the extension.
2. Create a project from ESP-IDF examples, Build, flash and monitor.
3. Debugging with steps to configure OpenOCD and debug adapter.
4. Heap tracing
5. Code coverage
6. Developing on Docker Container
7. Developing on WSL

Check all the tutorials here.

Table of content

Check all the documentation.

Available commands

Click `F1` to show Visual studio code actions, then type **ESP-IDF** to see all possible actions.

Category	**Command Description**	**Description**	**Keyboard Shortcuts (Mac)**	**Keyboard Shortcuts (Windows/ Linux)**
Configuration	Add OpenOCD rules file (For Linux users)	Add OpenOCD permissions to /etc/udev/rules.d to allow OpenOCD execution.		
	Add Docker Container Configuration	Add the **.devcontainer** files to the currently opened project directory, necessary to use a ESP-IDF project in a Docker container with Visual Studio Code Remote - Containers extension		
	Add vscode configuration folder	Add **.vscode** files to the currently opened project directory. These include launch.json (for debugging), settings.json and c_cpp_properties.json for syntax highlight.		
	Configure ESP-IDF extension	Open a window with a setup wizard to install ESP-IDF, IDF Tools and python virtual environment.		
	Select output and notification mode	This extension shows many notifications and output in the Output window **ESP-IDF**. This command allows the user to set if to show notifications, show output, both or none of them.		
	Select where to save configuration settings	In Visual Studio Code settings can be saved in 3 places: User Settings (global settings), workspace (.code-workspace file) or workspace folder (.vscode/settings.json). More information in working with multiple projects.		
	Pick a workspace folder	when using a Visual Studio Code workspace with multiple workspace folders, this command allow you to select which workspace folder to use for this extension commands. More information in working with multiple projects.		
	Basic	Show Examples Projects	Launch UI to show	

Category	Command Description	Description	Keyboard Shortcuts (Mac)	Keyboard Shortcuts (Windows/ Linux)
			examples from selected framework and allow the user to create a project from them. This command will show frameworks already configured in the extension so if you want to see ESP-Rainmaker examples you need to run the **Install ESP-Rainmaker** first (or set the equivalent setting idf.espRainmakerPath) and then execute this command to see the examples.	
		Set Espressif device target	This will set the target for the current project (IDF_TARGET). Similar to **idf.py set-target**. For example if you want to use ESP32 or ESP32-C3 you need to execute this command.	
SDK Configuration editor		Launch a UI to configure your ESP-IDF project settings. This is equivalent to **idf.py menuconfig**	`⌘ I G`	`Ctrl E G`
Build your project		Build your project using \`CMake\` and \`Ninja-build\` as explained in [ESP-IDF Build System Using Cmake Directly](). You could modify the behavior of the build task with **idf.cmakeCompilerArgs** for Cmake configure step and **idf.ninjaArgs** for Ninja step. For example, using **[-j N]** where N is the number of jobs run in parallel.	`⌘ I B`	`Ctrl E B`
Size analysis of the binaries		Launch UI with the ESP-IDF project binaries size information.	`⌘ I S`	`Ctrl E S`
Select port to use		Select which serial port to use for ESP-IDF tasks like flashing or monitor your device.	`⌘ I P`	`Ctrl E P`

Category	Command Description	Description	Keyboard Shortcuts (Mac)	Keyboard Shortcuts (Windows/ Linux)
Flash your project		Write binary data to the ESP's flash chip from your current ESP-IDF project. This command will use either UART, DFU or JTAG based on **idf.flashType**	⌘ I F	Ctrl E F
Monitor device		This command will execute idf.py monitor to start serial communication with Espressif device. Please take a look at the IDF Monitor Documentation.	⌘ I M	Ctrl E M
Open ESP-IDF Terminal		Launch a terminal window configured with extension ESP-IDF settings. Similar to export.sh script from ESP-IDF CLI.	⌘ I T	Ctrl E T
Select OpenOCD Board Configuration		Select the openOCD configuration files that match your Espressif device target. For example if you are using DevKitC or ESP-Wrover-Kit. This is necessary for flashing with JTAG or debugging your device.		
Build, Flash and start a monitor on your device		Build the project, write binaries program to device and start a monitor terminal with a single command. Similar to `idf.py build flash monitor`	⌘ I D	Ctrl E D
Project creation	Show Examples Projects	Launch UI to show examples from selected framework and allow the user to create a project from them. This command will show frameworks already configured in the extension so if you want to see ESP-Rainmaker examples you need to run the **Install ESP-Rainmaker** first (or set the equivalent setting idf.espRainmakerPath) and then execute this command to see the examples.		
	Create project from Extension Template	Create ESP-IDF using one of the extension template projects.	⌘ I C	Ctrl E C
	Create New ESP-IDF Component	Create a new component in the current directory based on ESP-IDF component template		
	Import ESP-IDF	Import an existing ESP-IDF		

Category	Command Description	Description	Keyboard Shortcuts (Mac)	Keyboard Shortcuts (Windows/ Linux)
	Project	project and add .vscode and .devcontainer files to a new location and also able to rename the project.		
	New Project	Launch UI with a ESP-IDF project creation wizard using examples templates from ESP-IDF and additional frameworks configured in the extension.	⌘ I N	Ctrl E N
	Select Flash Method	Select which flash method to use for **Flash your project** command. It can be DFU, JTAG or UART.		
	Flash your project	Write binary data to the ESP’s flash chip from your current ESP-IDF project. This command will use either UART, DFU or JTAG based on **idf.flashType**	⌘ I F	Ctrl E F
	Flash (DFU) your project	Write binary data to the ESP’s flash chip from your current ESP-IDF project using DFU. Only for ESP32-S2 and ESP32-S3.		
Flashing	Flash (UART) your project	Write binary data to the ESP’s flash chip from your current ESP-IDF project using esptool.py		
	Flash (with JTag)	Write binary data to the ESP’s flash chip from your current ESP-IDF project using OpenOCD JTAG		
	Encrypt and Flash your Project	Execute flashing the project program to device while adding **--encrypt** for partitions to be encrypted.		
	Erase Flash Memory from Device	Execute esptool.py erase_flash command to erase flash chip (set to 0xFF bytes)	⌘ I R	Ctrl E R
Code coverage	Add Editor coverage	Parse your project GCOV Code coverage files to add color lines representing code coverage on currently opened source code file		
	Configure Project SDKConfig for Coverage	Set required values in your project SDKConfig to enable Code Coverage		
	Get HTML	Parse your project GCOV Code		

Category	Command Description	Description	Keyboard Shortcuts (Mac)	Keyboard Shortcuts (Windows/ Linux)
	Coverage Report for project	coverage files to generate a HTML coverage report.		
	Remove Editor coverage	Remove editor colored lines from **Add Editor coverage** command		
Additional frameworks	Install ESP-ADF	Clone ESP-ADF inside the selected directory and set **idf.espAdfPath** (**idf.espAdfPathWin** in Windows) configuration setting.		
	Add Arduino ESP32 as ESP-IDF Component	Add Arduino-ESP32 as a ESP-IDF component in your current directory (**${CURRENT_DIRECTORY}/components/arduino**).		
	Install ESP-IDF Python Packages (DEPRECATION NOTICE)	Install extension python packages. Deprecated will be removed soon.		
	Install ESP-MDF	Clone ESP-MDF inside the selected directory and set **idf.espMdfPath** (**idf.espMdfPathWin** in Windows) configuration setting.		
	Install ESP-Matter	Clone ESP-Matter and set **idf.espMatterPath**. The **ESP-IDF: Set ESP-MATTER Device Path (ESP_MATTER_DEVICE_PATH)** is used to define the device path for ESP-Matter. ESP-Matter is not supported in Windows. Make sure to install Matter system prerequisites first.		
	Set ESP-MATTER Device Path (ESP_MATTER_DEVICE_PATH)	The **ESP-IDF: Set ESP-MATTER Device Path (ESP_MATTER_DEVICE_PATH)** is used to define the device path for ESP-Matter. ESP-Matter is not supported in Windows.		
	Install ESP-Rainmaker	Clone ESP-Rainmaker and set **idf.espRainmakerPath** (**idf.espRainmakerPathWin** in Windows) configuration setting.		

Category	Command Description	Description	Keyboard Shortcuts (Mac)	Keyboard Shortcuts (Windows/ Linux)
	Install ESP-HomeKit-SDK	Clone ESP-HomeKit-SDK inside the selected directory and set **idf.espHomeKitSdkPath** (**idf.espHomeKitSdkPathWin** in Windows) configuration setting.		
eFuse	Get eFuse Summary	Get list of eFuse and values from currently serial port chip.		
	Clear eFuse Summary	Clear the eFuse Summary tree from ESP Explorer EFUSEEXPLORER		
QEMU	Launch QEMU Server	As described in QEMU documentation this command will execute ESP32 QEMU from the project Dockerfile with the current project binaries.		
	Launch QEMU Debug Session	As described in QEMU documentation this command will start a debug session to ESP32 QEMU from the project Dockerfile with the current project binaries.		
	Monitor QEMU Device	As described in QEMU documentation this command will start a terminal to monitor the ESP32 QEMU from the project Dockerfile with the current project binaries.		
Monitoring	Monitor device	This command will execute idf.py monitor to start serial communication with Espressif device. Please take a look at the IDF Monitor Documentation.	`⌘ I M`	`Ctrl E M`
	Launch IDF Monitor for CoreDump / GDB-Stub Mode	Launch ESP-IDF Monitor with websocket capabilities. If the user has configured the panic handler to gdbstub or core dump, the monitor will launch a post mortem debug session of the chip.		
	Monitor QEMU Device	As described in QEMU documentation this command will start a terminal to monitor the ESP32 QEMU from the project Dockerfile with the current project binaries.		
Editors	NVS Partition Editor	Launch UI to create a CSV file		

Category	Command Description	Description	Keyboard Shortcuts (Mac)	Keyboard Shortcuts (Windows/ Linux)
		for ESP_IDF Non Volatile Storage		
	Partition Table Editor	Launch UI to manage custom partition table as described in ESP_IDF Partition Table		
	SDK Configuration editor	Launch a UI to configure your ESP-IDF project settings. This is equivalent to **idf.py menuconfig**	`⌘ I G`	`Ctrl E G`
Unit Testing	Unit Test: Build and flash unit test app for testing	Copy the unit test app in the current project, build the current project and flash the unit test application to the connected device. More information in Unit testing documentation		
	Unit Test: Install ESP-IDF PyTest requirements	Install the ESP-IDF Pytest requirements packages to be able to execute ESP-IDF Unit tests. More information in		
Scripts and Tools	Run idf.py reconfigure task	This command will execute **idf.py reconfigure** (CMake configure task). Useful when you need to generate compile_commands.json for the C/C++ language support.		
	Erase Flash Memory from Device	Execute esptool.py erase_flash command to erase flash chip (set to 0xFF bytes)	`⌘ I R`	`Ctrl E R`
	Dispose Current SDK Configuration Editor Server Process	If you already executed the SDK Configuration editor, a cache process will remain in the background for faster re opening. This command will dispose of such cache process.		
	Doctor Command	Run a diagnostic of the extension setup settings and extension logs to provide a troubleshooting report.		
	Troubleshoot Form	Launch UI for user to send a troubleshoot report with steps to reproduce, run a diagnostic of the extension setup settings and extension logs to send to telemetry backend.		
	Run ESP-IDF-SBOM vulnerability check	Creates Software bill of materials (SBOM) files in the Software Package Data Exchange (SPDX) format for applications generated		

Category	Command Description	Description	Keyboard Shortcuts (Mac)	Keyboard Shortcuts (Windows/ Linux)
		by the Espressif IoT Development Framework (ESP-IDF).		
	Save Default SDKCONFIG file (save-defconfig)	Generate sdkconfig.defaults files using the project current sdkconfig file.		
	Show Ninja Build Summary	Execute the Chromium ninja-build-summary.py		
	Search in documentation...	Select some text from your source code file and search in ESP-IDF documentation with results right in the vscode ESP-IDF Explorer tab.	`⌘ I Q`	`Ctrl E Q`
	Search Error Hint	Type some text to find a matching error from ESP-IDF hints dictionary.		
Clear ESP-IDF Search Results	Clear results from ESP Explorer Documentation Search Results			
Clear Saved ESP-IDF Setups	Clear existing esp-idf setups saved by the extension.			

About commands

1. The **Add Arduino-ESP32 as ESP-IDF Component** command will add Arduino-ESP32 as a ESP-IDF component in your current directory (`${CURRENT_DIRECTORY}/components/arduino`).

   NOTE: Not all versions of ESP-IDF are supported. Make sure to check Arduino-ESP32 to see if your ESP-IDF version is compatible.

2. You can also use the **ESP-IDF: Create Project from Extension Template** command with `arduino-as-component` template to create a new project directory that includes Arduino-ESP32 as an ESP-IDF component.

3. The **Install ESP-ADF** will clone ESP-ADF inside the selected directory and set `idf.espAdfPath` (`idf.espAdfPathWin` in Windows) configuration setting.

4. The **Install ESP-Matter** will clone ESP-Matter inside the selected directory and set `idf.espMatterPath` configuration setting. The **ESP-IDF: Set ESP-MATTER Device Path (ESP_MATTER_DEVICE_PATH)** is used to define the device path for ESP-Matter. **ESP-Matter is not supported in Windows**. Make sure to install Matter system prerequisites first.

5. The **Install ESP-MDF** will clone ESP-MDF inside the selected directory and set `idf.espMdfPath` (`idf.espMdfPathWin` in Windows) configuration setting.

6. The **Install ESP-HomeKit-SDK** will clone ESP-HomeKit-SDK inside the selected directory and set `idf.espHomeKitSdkPath` (`idf.espHomeKitSdkPathWin` in Windows) configuration setting.

7. The **Show Examples Projects** command allows you create a new project using one of the examples in ESP-IDF, ESP-ADF, ESP-Matter, ESP-HomeKit-SDK or ESP-MDF directory if related configuration settings are correctly defined.

Commands for tasks.json and launch.json

We have implemented some utilities commands that can be used in tasks.json and launch.json that can be used like:

```
"miDebuggerPath": "${command:espIdf.getToolchainGdb}"
```

- `espIdf.getExtensionPath`: Get the installed location absolute path.
- `espIdf.getOpenOcdScriptValue`: Return the value of OPENOCD_SCRIPTS from `idf.customExtraVars` or from system OPENOCD_SCRIPTS environment variable.
- `espIdf.getOpenOcdConfig`: Return the openOCD configuration files as string. Example `-f interface/ftdi/esp32_devkitj_v1.cfg -f board/esp32-wrover.cfg`.
- `espIdf.getProjectName`: Return the project name from current workspace folder `build/project_description.json`.
- `espIdf.getToolchainGcc`: Return the absolute path of the toolchain gcc for the ESP-IDF target given by `idf.adapterTargetName` configuration setting and `idf.customExtraPaths`.
- `espIdf.getToolchainGdb`: Return the absolute path of the toolchain gdb for the ESP-IDF target given by `idf.adapterTargetName` configuration setting and `idf.customExtraPaths`.

See an example in the debugging documentation.

Available Tasks in tasks.json

A template Tasks.json is included when creating a project using **ESP-IDF: Create Project from Extension Template**. These tasks can be executed by running `F1`, writing `Tasks: Run task` and selecting one of the following:

1. `Build` - Build Project
2. `Set Target to esp32`
3. `Set Target to esp32s2`

4.`Clean` - Clean the project

5.`Flash` - Flash the device

6.`Monitor` - Start a monitor terminal

7.`OpenOCD` - Start the openOCD server

8.`BuildFlash` - Execute a build followed by a flash command.

Note that for OpenOCD tasks you need to define `OPENOCD_SCRIPTS` in your system environment variables with openocd scripts folder path.

Troubleshooting

If something is not working please check for any error on one of these:

> **NOTE:** Use `idf.openOcdDebugLevel` configuration setting to 3 or more to show debug logging in OpenOCD server output.

> **NOTE:** Use `logLevel` in your /.vscode/launch.json to 3 or more to show more debug adapter output.

1.In Visual Studio Code select menu **View** -> **Output** -> **ESP-IDF**. This output information is useful to know what is happening in the extension.

2.In Visual Studio Code select menu **View** then click **Command Palette...** and type `ESP-IDF: Doctor Command` to generate a report of your environment configuration and it will be copied in your clipboard to paste anywhere.

3.Check log file which can be obtained from:

•Windows: `%USERPROFILE%\.vscode\extensions\espressif.esp-idf-extension-VERSION\esp_idf_vsc_ext.log`

•Linux & MacOSX: `$HOME/.vscode/extensions/espressif.esp-idf-extension-VERSION/esp_idf_vsc_ext.log`

4.In Visual Studio Code, select menu **Help** -> `Toggle Developer Tools` and copy any error in the Console tab related to this extension.

5.Make sure that your extension is properly configured as described in JSON Manual Configuration. Visual Studio Code allows the user to configure settings at different levels: **Global (User Settings)**, **Workspace** and **Workspace Folder** so make sure your project has the right settings. The `ESP-IDF: Doctor command` result might give the values from user settings instead of the workspace folder settings.

6.Review the OpenOCD troubleshooting FAQ related to the `OpenOCD` output, for application tracing, debug or any OpenOCD related issues.

If there is any Python package error, please try to reinstall the required python packages with the **ESP-IDF: Install ESP-IDF Python Packages** command. Please consider that this extension install ESP-IDF, this extension's and ESP-IDF Debug Adapter python packages when running the **ESP-IDF: Configure ESP-IDF Extension** setup wizard.

> **NOTE:** When downloading ESP-IDF using git cloning in Windows if you receive errors like "unable to create symlink", enabling `Developer Mode` while cloning ESP-IDF could help resolve the issue.

If the user can't resolve the error, please search in the github repository issues for existing errors or open a new issue here.

Code of Conduct

This project and everyone participating in it is governed by the Code of Conduct. By participating, you are expected to uphold this code. Please report unacceptable behavior to vscode@espressif.com.

License

This extension is licensed under the Apache License 2.0. Please see the LICENSE file for additional copyright notices and terms.