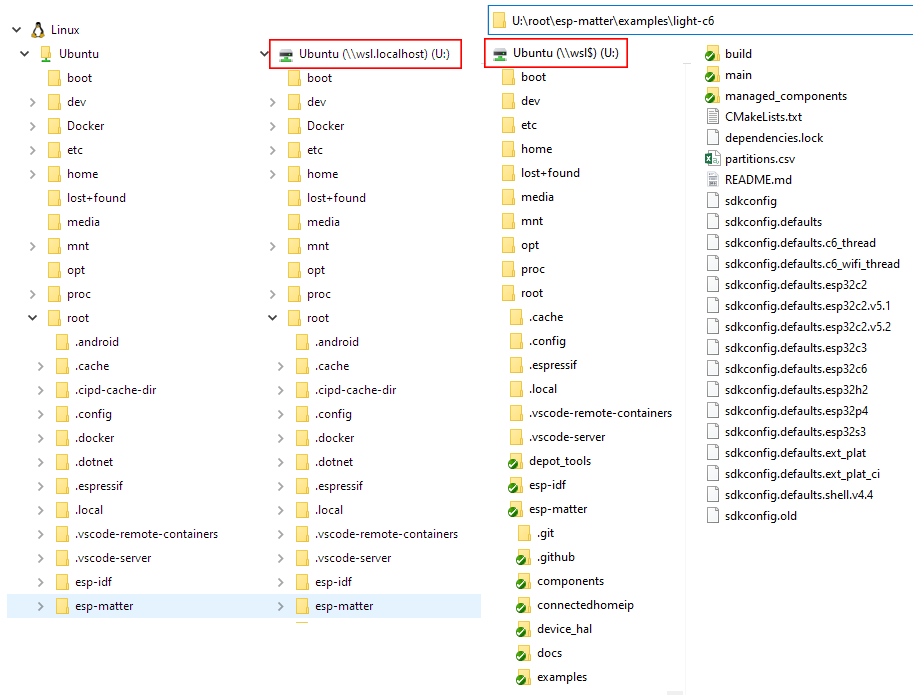
**How to setup and work with ESP-IDF and ESP-Matter**  
URL: https://github.com/mozolin/matter-thread  
  
Windows Powershell (with Administrator rights):  
~~~  
systeminfo  
~~~  
OS Name: Microsoft Windows 10 Pro  
OS Version: 10.0.19045 N/A Build 19045  
***Works successfully in this version of Windows!***  
~~~  
systeminfo  
~~~  
OS Name: Microsoft Windows 10 Enterprise  
OS Version: 10.0.19045 N/A Build 19045  
***Works without sharing COM-ports in this version of Windows!***

**# 1. How to install Ubuntu on Windows with WSL**Recommended Ubuntu 22.04…  
https://learn.microsoft.com/en-us/windows/wsl/installCheck Ubuntu version:~~~lsb\_release -a~~~If asked to add a new user, just do it, but after that we should switch to root.Windows Powershell:~~~  
ubuntu config --default-user root #-- default instance  
ubuntu2204 config --default-user root #-- another instance  
~~~  
  
**# 2. Ubuntu on Windows App Store**https://apps.microsoft.com/search?query=ubuntu&hl=en-us&gl=US  
Map network resource **\\wsl.localhost\Ubuntu** or **\\wsl$\Ubuntu** on disk **U** (for instance). So, the project examples can be found in **U:\root\esp-matter\examples** folder. If we are using another WSL instance, then the path should be something like **\\wsl$\Ubuntu-22.04**

  
*ubuntu\_wsl.png*  
**# 3. ESP-IDF Prerequisites (Ubuntu only)**https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/linux-macos-setup.html  
https://wiki.seeedstudio.com/xiao\_idf/  
~~~  
sudo apt update  
sudo apt upgrade  
sudo apt-get install git wget flex bison gperf python3 python3-pip python3-venv cmake ninja-build ccache libffi-dev libssl-dev dfu-util libusb-1.0-0  
~~~

**# 4. ESP-IDF Setup (Windows and Ubuntu)**https://docs.espressif.com/projects/esp-matter/en/latest/esp32/developing.html  
Windows Powershell (with Administrator rights):  
~~~  
D:  
cd /  
mkdir Espressif  
cd Espressif  
~~~  
Both, Windows and Ubuntu (esp-idf v5.2.4 is recommended):  
~~~  
git clone --recursive https://github.com/espressif/esp-idf.git

cd esp-idf  
git checkout v5.2.4  
git submodule update --init --recursive

./install.sh

source ./export.sh  
#./install.bat #-- Windows instance

#./export.bat #-- Windows instance

~~~  
To /root/.bashrc add (Ubuntu only):~~~#-- Alias for setting up the ESP-IDF environmentalias get\_idf='. ~/esp-idf/export.sh'~~~Than run:~~~source ~/.bashrc~~~  
Now it is possible to run get\_idf to set up or refresh the esp-idf environment in any terminal session.

**# 5. Matter Prerequisites (Ubuntu only)**https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/BUILDING.md#prerequisites  
https://wiki.seeedstudio.com/xiao\_esp32\_matter\_env/  
  
~~~  
sudo apt-get install git gcc g++ pkg-config 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 libreadline-dev default-jre  
~~~  
  
**# 6. ESP Matter Setup (Ubuntu only)**https://docs.espressif.com/projects/esp-matter/en/latest/esp32/developing.html#esp-matter-setup  
https://wiki.seeedstudio.com/xiao\_esp32\_matter\_env/

~~~  
cd esp-idf

source ./export.sh

cd ..

git clone --depth 1 https://github.com/espressif/esp-matter.git

cd esp-matter

git submodule update --init --depth 1

cd ./connectedhomeip/connectedhomeip

./scripts/checkout\_submodules.py --platform esp32 linux --shallow

cd ../..

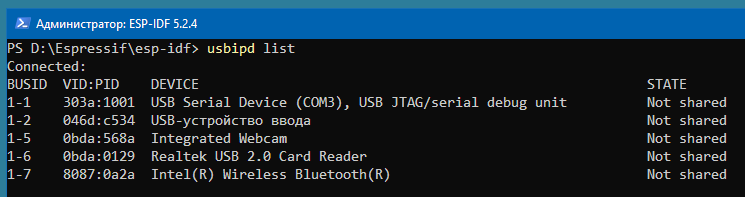
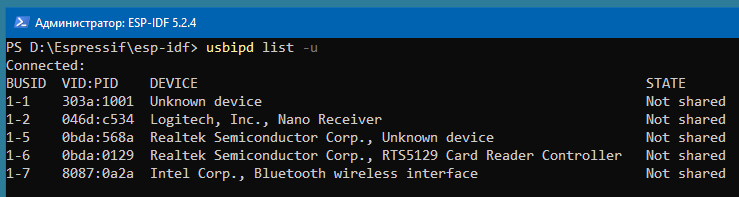
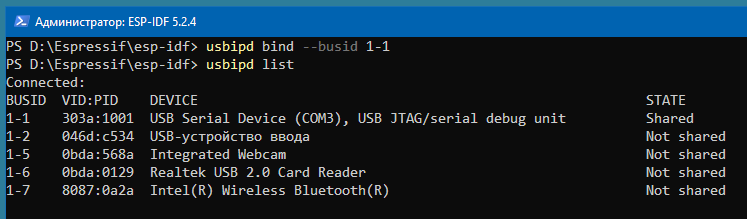
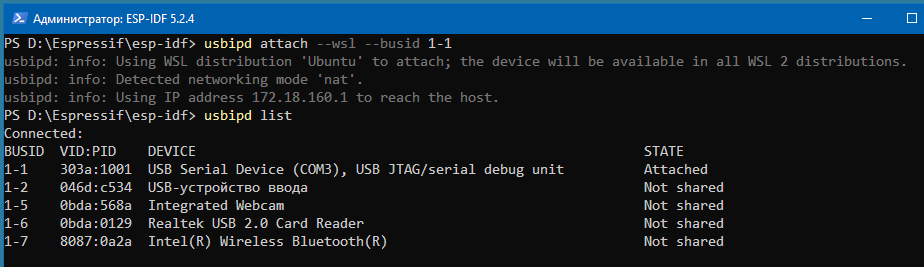
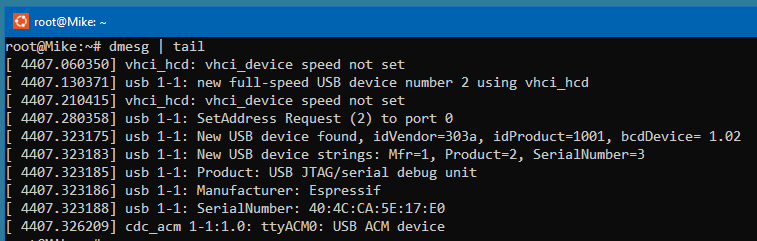
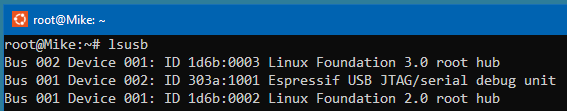
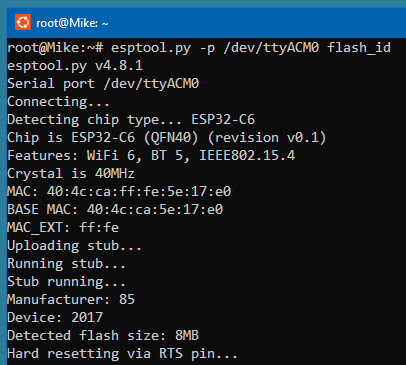
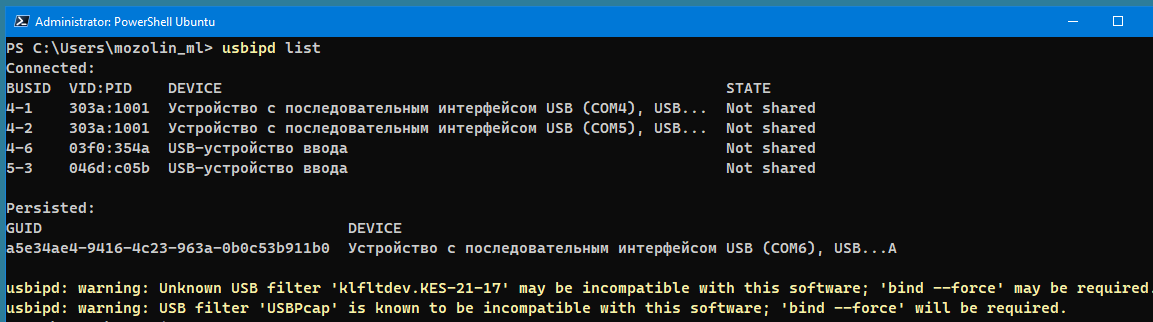
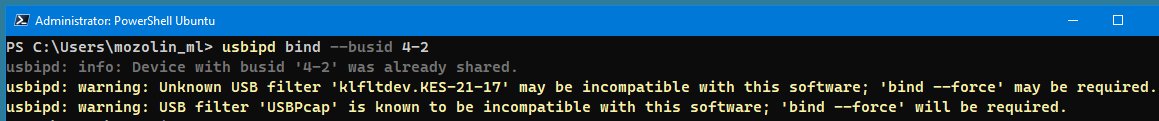
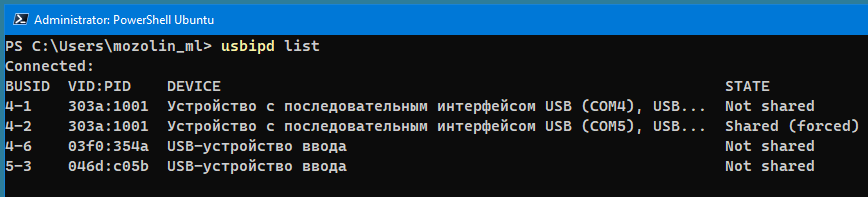
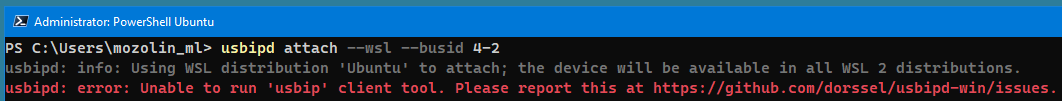
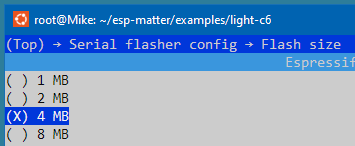
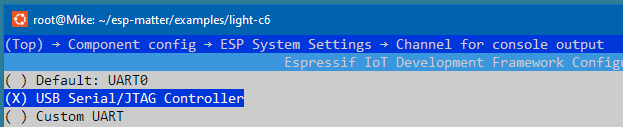
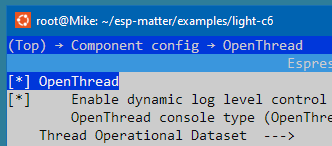
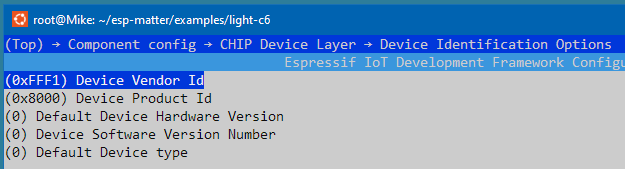
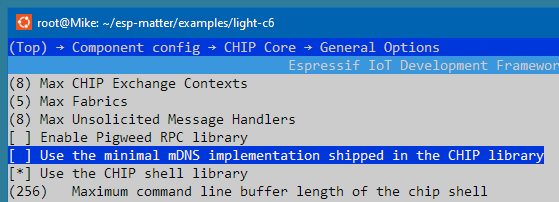
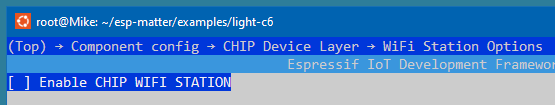
./install.sh

cd ..

cd esp-idf; source ./export.sh; cd ..

cd esp-matter; source ./export.sh; cd ..  
  
export IDF\_CCACHE\_ENABLE=1  
~~~  
To /root/.bashrc add:  
~~~  
#-- Alias for setting up the ESP-Matter environment  
alias get\_matter='. ~/esp-matter/export.sh'  
  
#-- Enable ccache to speed up compilation  
alias set\_cache='export IDF\_CCACHE\_ENABLE=1'  
~~~  
Than run:  
~~~  
source ~/.bashrc  
~~~  
Now it is possible to run get\_matter and set\_cache to set up or refresh the esp-matter environment in any terminal session.  
  
Note:

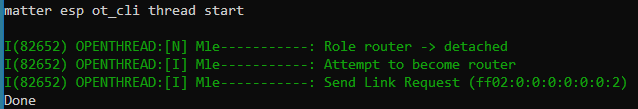
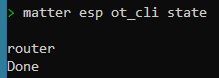
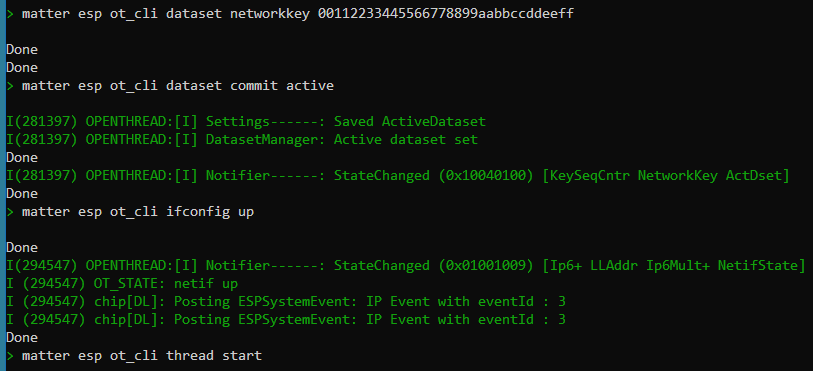
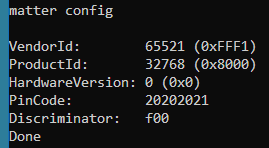
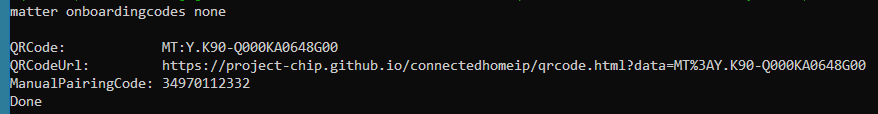
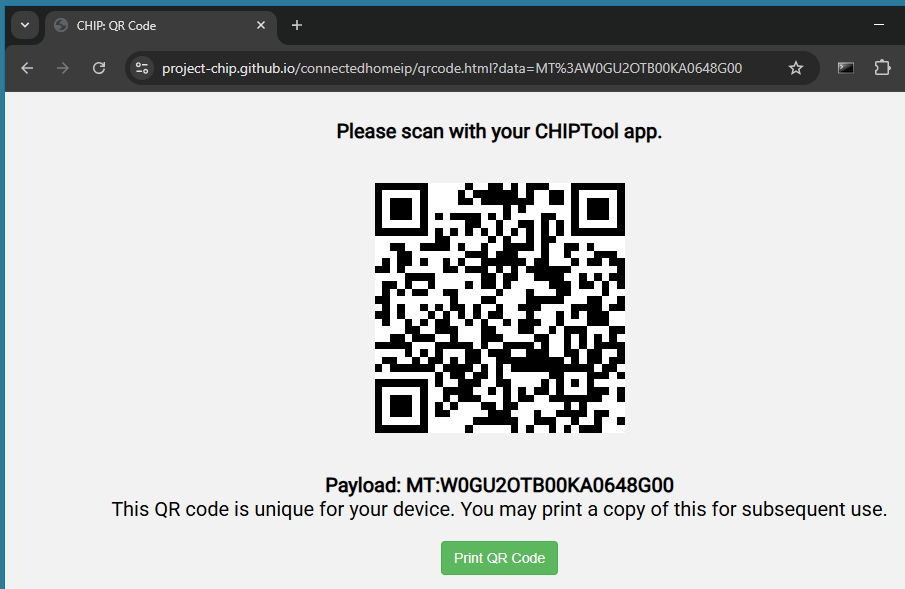
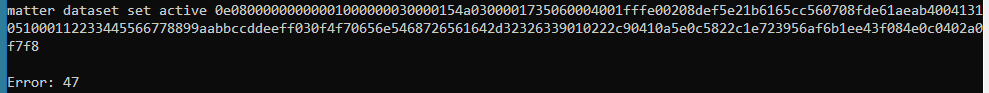
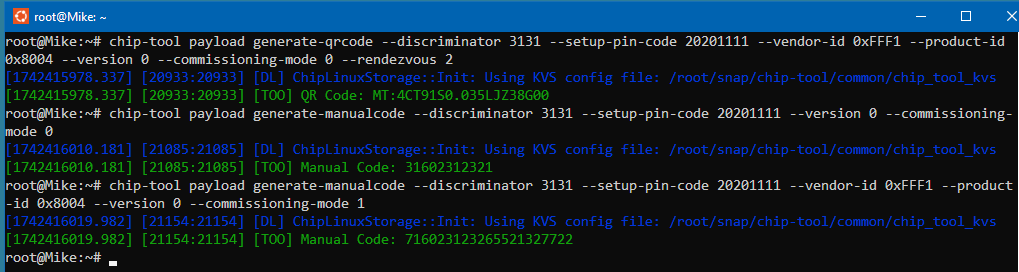
"A complete installation of Ubuntu, ESP-IDF and ESP-Matter takes up about 20 GB of disk space on drive C."

**# 7. Install Visual Studio Code**https://code.visualstudio.com/ **# 8. Install Remote WSL extension in Visual Studio Code**https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/additionalfeatures/wsl.html#install-remote-wsl-extension-in-visual-studio-code  
  
**# 9. Install usbipd-win**https://github.com/dorssel/usbipd-win/releases  
https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/additionalfeatures/wsl.html#usbipd-win-in-wslWindows Powershell:  
~~~  
usbipd list  
usbipd list –u  
usbipd bind --busid 1-3  
usbipd attach --wsl --busid 1-3  
~~~  
  
  
*usbipd\_01.png*  
  
  
*usbipd\_02.png*  
  
  
*usbipd\_03.png*  
  
  
*usbipd\_04.png*  
  
Ubuntu:  
~~~  
dmesg | tail  
lsusb  
~~~  
  
  
*usbipd\_05.png*  
  
  
*usbipd\_06.png*  
  
  
*usbipd\_07.png*  
  
There may be issues with shared access to COM ports in Windows 10 Enterprise. Therefore, it will not be possible to use Ubuntu ports for flashing and monitoring the firmware. In this case, we should make a build in VSCode using a remote WSL or in Ubuntu, and then use a shared drive (**U**). Here are a couple of examples of BAT files for flashing and monitoring: **/D/Espressif/idf\_build\_c6.cmd** and **/D/Espressif/idf\_build\_h2.cmd**  
  
  
*usdipd\_error\_01.png*  
  
  
*usdipd\_error\_02.png*  
  
  
*usdipd\_error\_03.png*  
  
  
*usdipd\_error\_04.png*  
  
**# 10. Make an example project (all the settings are made for ESP32-C6 development board, Ubuntu only)**- Make a copy of /root/esp-matter/examples/**light** folder to ../**light-c6**  
- Open this folder in VSCode using a remote WSL or in Ubuntu  
~~~  
cd /root/esp-matter/examples/light-c6 #-- Navigate to the light example directory  
rm -rf build/ #-- Clean previous build files  
idf.py set-target esp32c6 #-- Set the build target to ESP32-C6  
idf.py menuconfig #-- Enter the configuration menu  
~~~  
  
*CONFIG\_OPENTHREAD\_ENABLED=****y*** *CONFIG\_ENABLE\_WIFI\_STATION=****n*** *CONFIG\_USE\_MINIMAL\_MDNS=****n***  
  
  
*esp\_menuconfig\_01.png*  
  
  
*esp\_menuconfig\_02.png*  
  
  
*esp\_menuconfig\_03.png*  
  
  
*esp\_menuconfig\_04.png*  
  
  
*esp\_menuconfig\_05.png*  
  
  
*esp\_menuconfig\_06.png*  
  
~~~  
idf.py -p /tty/ACM0 build flash monitor #-- Building, flashing and monitoring  
~~~  
  
**JOIN THE THREAD NETWORK VIA NETWORKKEY**  
OpenThread Border Router (see: “How\_to\_setup\_and\_work\_with\_OpenThread\_Border\_Router.docx”):  
~~~  
dataset active -x  
~~~  
0e08000000000001000000030000154a0300001735060004001fffe00208def5e21b6165cc560708fde61aeab4004131051000112233445566778899aabbccddeeff030f4f70656e5468726561642d32326339010222c90410a5e0c5822c1e723956af6b1ee43f084e0c0402a0f7f8  
Done  
  
~~~  
networkkey  
~~~  
00112233445566778899aabbccddeeff  
Done  
  
Thread End Device:  
~~~  
matter esp ot\_cli dataset set active  
0e08000000000001000000030000154a0300001735060004001fffe00208def5e21b6165cc560708fde61aeab4004131051000112233445566778899aabbccddeeff030f4f70656e5468726561642d32326339010222c90410a5e0c5822c1e723956af6b1ee43f084e0c0402a0f7f8

~~~  
Error 7: InvalidArgs  
Why is that? There is a difference between the results of the datasets:

Here:

*> matter esp ot\_cli dataset set active 0e08000000000001000000030000154a0300001735060004001fffe00208def5e21b6165cc560708fde61aeab4004131051000112233445566778899aabbccddeeff030f4f70656e5468726561642d32326339010222c90410a5e0c5822c1e723956af6b1ee43f084e****0c0402a0f7f8***   
  
Example "Join the OTBR network" (https://openthread.io/codelabs/openthread-border-router:

*> matter esp ot\_cli dataset set active 0e080000000000010000000300001235060004001fffe002083d3818dc1c8db63f0708fda85ce9df1e662005101d81689e4c0a32f3b4aa112994d29692030f4f70656e5468726561642d35326532010252e204103f23f6b8875d4b05541eeb4f9718d2f40c0302a0ff*The second one is 12 characters shorter!  
So, we pair the end device using the Thread Router network key.  
  
~~~  
matter esp ot\_cli dataset networkkey 00112233445566778899aabbccddeeff  
matter esp ot\_cli dataset commit active  
matter esp ot\_cli ifconfig up  
matter esp ot\_cli thread start  
matter esp ot\_cli state  
~~~  
  
  
*esp\_join\_openthread\_01.png*  
  
  
*esp\_join\_openthread\_02.png*  
  
  
*esp\_join\_openthread\_03.png*  
  
  
*esp\_join\_openthread\_04.png*  
  
  
*esp\_join\_openthread\_05.png*  
  
  
*matter\_esp\_ot\_cli.png*  
  
**# 11. ??? Test Setup (CHIP Tool)**https://docs.espressif.com/projects/esp-matter/en/latest/esp32c6/developing.html#test-setup-chip-tool  
  
  
**# 12. ??? Working with the CHIP Tool**https://github.com/project-chip/connectedhomeip/blob/master/docs/development\_controllers/chip-tool/chip\_tool\_guide.md **# 13. Matter Shell Reference**https://project-chip.github.io/connectedhomeip-doc/examples/chef/README\_SHELL.htmlhttps://docs.espressif.com/projects/esp-matter/en/latest/esp32/developing.html#device-console ****  
*esp\_matter\_01.png*  
  
****  
*esp\_matter\_02.png*  
  
  
*qrcode\_01.png*  
  
****  
*esp\_matter\_03.png* **# 14. How to generate Matter Onboarding Codes (QR Code and Manual Pairing Code)**https://docs.espressif.com/projects/esp-matter/en/latest/esp32/faq.html#a1-9-how-to-generate-matter-onboarding-codes-qr-code-and-manual-pairing-code  
~~~  
#-- Generate the QR Code  
chip-tool payload generate-qrcode --discriminator 3131 --setup-pin-code 20201111 --vendor-id 0xFFF1 --product-id 0x8004 --version 0 --commissioning-mode 0 --rendezvous 2  
  
#-- Generates the short manual pairing code (11-digit)  
chip-tool payload generate-manualcode --discriminator 3131 --setup-pin-code 20201111 --version 0 --commissioning-mode 0  
  
#-- To generate a long manual pairing code (21-digit) that includes both the vendor ID and product ID, --commissioning-mode parameter must be set to either 1 or 2, indicating a non-standard commissioning flow  
chip-tool payload generate-manualcode --discriminator 3131 --setup-pin-code 20201111 --vendor-id 0xFFF1 --product-id 0x8004 --version 0 --commissioning-mode 1  
~~~  
  
*esp\_chiptool\_qrcode.png*