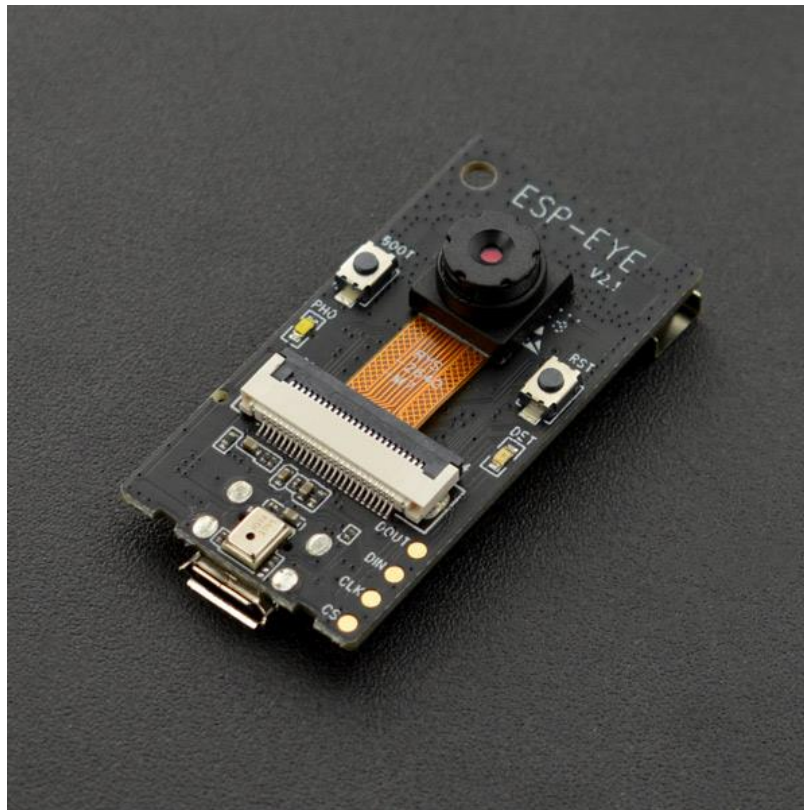


# ESP-EYE Development Board

SKU:DFR0620

---



## Introduction

---

ESP-EYE is a development board for image recognition and speech signal processing, on which it integrates an on-board ESP32 chip, a 2 megapixel camera, and a digital microphone, 8MByte PSRAM and 4MByte flash. The board also supports image transmission via Wi-Fi and debugging through a Micro-USB port.

What's more, there is a complete AIoT solution for the product, including the ESP-EYE development board, ESP-WHO AI frameworks, and ESP-IDF software SDK, which enables users to explore the benefits of combining of AI technology and IoT applications.

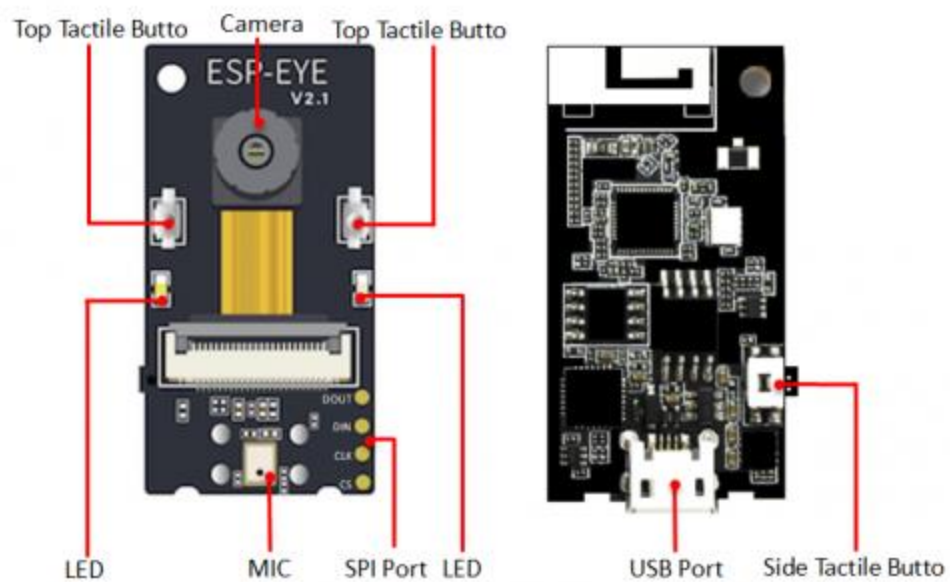
## Specification

---

- Micro-controller: ESP32 Series
- Power/Debug: Micro USB
- Dimension: 41.00x21.00x6.50mm/1.61x0.83x0.26"
- Operating Voltage: 3.3V
- Operating Current: 0.5A
- Total Current Output from I/O: 1200mA
- Operating Temperature: -40~125°C
- Storage Temperature: -50~150°C

## Overview

---



| Name     | Description                |
|----------|----------------------------|
| USB Port | Debug/Power Interface      |
| Camera   | Face Detection/Recognition |

| Name                | Description  |
|---------------------|--|
| Side Tactile Button | Function Key   |
| Top Tactile Button  | Reset/BOOT Button (We recommend that you do not configure this button for other uses.)   |
| LED                 | Red and White indicators.For indicating different statues of the board, e.g. waking up, networking, face detection, face recognition, face enrollment. |
| MIC                 | A digital microphone for voice control functions.  |
| SPI Port            | A reserved port for data transmission  |

## ESP-EYE User Guide For Windows 10

---

### Requirements

- **Hardware**
  - ESP-EYE Development Board V2.1 x 1
  - Micro USB Cable x 1
  - PC(Windows10) ×1
- **Software**
  - Toolchain to compile code for ESP32
  - Build tools - CMake and Ninja to build a full Application for ESP32
  - ESP-IDF that essentially contains API (software libraries and source code) for ESP32 and scripts to operate the Toolchain
  - Text editor to write programs (Projects) in C, e.g.,

# Setting up Development Environment

## *Step 1. Install Prerequisites*

- IDE Tools Installer

[https://dl.espressif.com/dl/esp32\\_win32\\_msys2\\_environment\\_and\\_toolchain-20190611.zip](https://dl.espressif.com/dl/esp32_win32_msys2_environment_and_toolchain-20190611.zip)

- ESP-IDF Tools Installer

The easiest way to install ESP-IDF's prerequisites is to download the ESP-IDF Tools installer from this URL: <https://dl.espressif.com/dl/esp-idf-tools-setup-2.0.exe>

The installer includes the cross-compilers, OpenOCD, cmake and Ninja build tool, and a configuration tool called mconf-idf. The installer can also download and run installers for Python 3.7 and Git For Windows if they are not already installed on the computer.

The installer also offers to download one of the ESP-IDF release versions.

- Using the Command Prompt

For the remaining Getting Started steps, we're going to use the Windows Command Prompt.

ESP-IDF Tools Installer creates a shortcut in the Start menu to launch the ESP-IDF Command Prompt. This shortcut launches the Command Prompt(cmd.exe) and runs export.bat script to set up the environment variables (PATH, IDF\_PATH and others). Inside this command prompt, all the installed tools are available.

Note that this shortcut is specific to the ESP-IDF directory selected in the ESP-IDF Tools Installer. If you have multiple ESP-IDF directories on the computer (for example, to work with different versions of ESP-IDF), you have two options to use them:

1. Create a copy of the shortcut created by the ESP-IDF Tools Installer, and change the working directory of the new shortcut to the ESP-IDF directory you wish to use.
2. Alternatively, run cmd.exe, then change to the ESP-IDF directory you wish to use, and run export.bat. Note that unlike the previous option, this way requires Python and Git to be present in PATH. If you get errors related to Python or Git not being found, use the first option.

### *Step 2. Get ESP-WHO*

To build applications for the ESP32, you need the software libraries provided by Espressif in ESP-WHO repository.

To get ESP-WHO, navigate to your installation directory and clone the repository with **git clone**, following instructions below specific to your operating system.

Open Terminal, and run the following commands:

```
mkdir%userprofile%\esp
cd %userprofile%\esp
git clone --recursive https://github.com/espressif/esp-who.git
```

### *Step 3. Set up the Tools*

Aside from the ESP-IDF, you also need to install the tools used by ESP-IDF, such as the compiler, debugger, Python packages, etc.

ESP-IDF Tools Installer for Windows introduced in Step 1 installs all the required tools.

If you want to install the tools without the help of the ESP-IDF Tools Installer, open the Command Prompt and follow these steps:

```
cd %userprofile%\esp\esp-idf
install.bat
```

- Customizing the tools installation path

The scripts introduced in this step install compilation tools required by ESP-IDF inside the user home directory:

**\$HOME/.espressif** on Linux and macOS, **%USERPROFILE%.espressif** on Windows. If you wish to install the tools into a different directory, set the environment variables **IDF\_TOOLS\_PATH** before running the installation scripts. Make sure that your user has sufficient permissions to read and write this path.

If changing the **IDF\_TOOLS\_PATH**, make sure it is set to the same value every time the **install.bat/install.sh** and **export.bat/export.sh** scripts are executed.

#### *Step 4. Set up the Environment Variables*

The installed tools are not yet added to the PATH environment variable. To make the tools usable from the command line, some environment variables must be set. ESP-IDF provides another script which does that.

ESP-IDF Tools Installer for Windows creates an "ESP-IDF Command Prompt" shortcut in the Start Menu. This shortcut opens the Command Prompt and sets up all the required environment variables. You can open this shortcut and proceed to the next step.

Alternatively, if you want to use ESP-IDF in an existing Command Prompt window, you can run:

```
%userprofile%\esp\esp-idf\export.bat
```

#### *Step 5. Start a Project*

Now you are ready to prepare your applications for ESP32. You can start with `single_chip/recognition_solution` project from `examples` directory in IDF.

Copy `single_chip/recognition_solution` to `~/esp` directory:

```
cd %userprofile%\esp  
xcopy /e /i %IDF_PATH%\examples\ single_chip/recognition_solution
```

There is a range of example projects in the `examples` directory in ESP-WHO. You can copy any project in the same way as presented above and run it.

It is possible to build examples in-place, without copying them first.

#### *Step 6. Connect Your Device*

Now connect your ESP32 board to the computer and check under what serial port the board is visible.

Serial ports have the following patterns in their names:

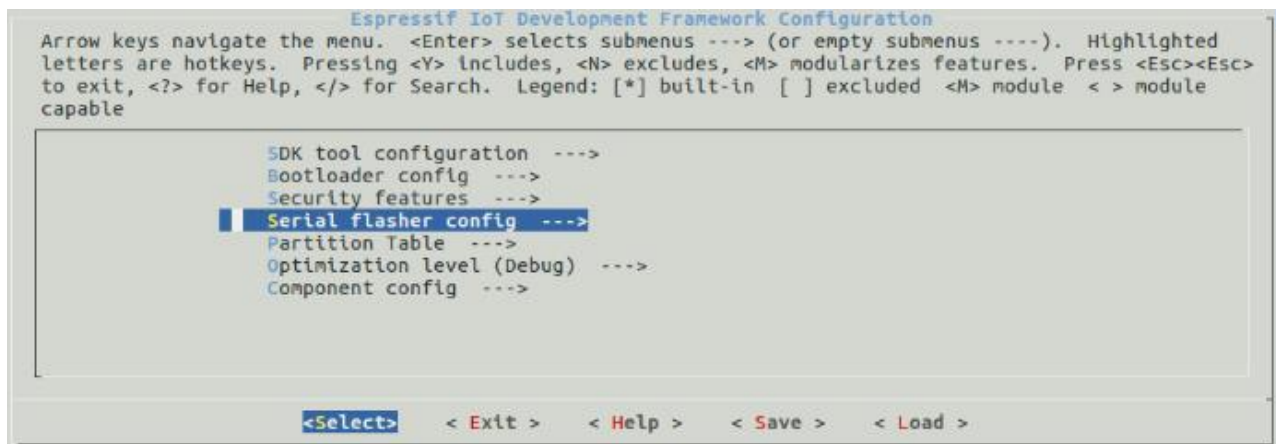
Windows : names like COM1

### Step 7. Configure

Navigate to your **recognition\_solution** directory from Step 5 Start a Project and run the project configuration utility menuconfig.

```
cd %userprofile%\esp\ recognition_solution
idf.py menuconfig
```

If the previous steps have been done correctly, the following menu appears:



### Step 8. Build the Project

Build the project by running:

```
idf.py build
```

This command will compile the application and all ESP-IDF components, then it will generate the bootloader, partition table, and application binaries.

```
$ idf.py build
Running cmake in directory /path/to/ recognition_solution /build
Executing "cmake -G Ninja --warn-uninitialized /path/to/ recognition_solution "...
Warn about uninitialized values.
-- Found Git: /usr/bin/git (found version "2.17.0")
-- Building empty aws_iot component due to configuration
-- Component names: ...
-- Component paths: ...
```

... (more lines of build system output)

```
[527/527] Generating hello-world.bin
esptool.py v2.3.1
```

Project build complete. To flash, run this command:

```
../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600 write_flash
--flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
recognition_solution.bin build 0x1000 build/bootloader/bootloader.bin 0x8000
build/partition_table/partition-table.bin
or run 'idf.py -p PORT flash'
```

If there are no errors, the build will finish by generating the firmware binary.bin file.

### *Step 9. Flash onto the Device*

Flash the binaries that you just built onto your ESP32 board by running:

```
idf.py -p PORT [-b BAUD] flash
```

Replace PORT with your ESP32 board's serial port name from Step 6 Connect Your Device.

You can also change the flasher baud rate by replacing BAUD with the baud rate you need. The default baud rate is 460800.



```
Running esptool.py in directory [...]esp/ recognition_solution
Executing "python [...]esp-idf/components/esptool_py/esptool/esptool.py -b 460800 write_flash
@flash_project_args"...
esptool.py -b 460800 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x1000
bootloader/bootloader.bin      0x8000      partition_table/partition-table.bin      0x10000
recognition_solution.bin
esptool.py v2.3.1
Connecting....
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 22992 bytes to 13019...
Wrote 22992 bytes (13019 compressed) at 0x00001000 in 0.3 seconds (effective 558.9 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 5789.3 kbit/s)...
Hash of data verified.
Compressed 136672 bytes to 67544...
Wrote 136672 bytes (67544 compressed) at 0x00010000 in 1.9 seconds (effective 567.5 kb|t/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

If there are no issues by the end of the flash process, the module will be reset and the "recognition\_solution" application will be running.

### Requirements

- **Hardware**
  - ESP-EYE Development Board x 1
  - Micro USB Cable x 1
  - Raspberry Pi ×1
- **Software**
  - Toolchain to compile code for ESP32
  - Build tools - CMake and Ninja to build a full Application for ESP32
  - ESP-IDF that essentially contains API (software libraries and source code) for ESP32 and scripts to operate the Toolchain
  - Text editor to write programs (Projects) in C, e.g (Eclipse).

### Setting up Development Environment

*Step 1. Install Prerequisites*

**To compile with ESP-IDF you need to get the following packages:**

- CentOS 7:

```
sudo yum install git wget ncurses-devel flex bison gperf python pyserial python-pyelftools cmake ninja-build ccache
```

- Ubuntu and Debian:

```
sudo apt-get install git wget libncurses-dev flex bison gperf python python-pip python-setuptools python-serial python-click python-cryptography python-future python-pyparsing python-pyelftools cmake ninja-build ccache
```

- Arch

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python2-pip python2-pyserial python2-click python2-cryptography python2-future python2-pyparsing python2-pyelftools cmake ninja ccache
```

## Permission issues /dev/ttyUSB0

With some Linux distributions you may get the Failed to open port /dev/ttyUSB0 error message when flashing the ESP32. This can be solved by adding the current user to the dialout group.

### *Step 2. Get ESP-WHO*

To build applications for the ESP32, you need the software libraries provided by Espressif in ESP-WHO repository.

To get ESP-WHO, navigate to your installation directory and clone the repository with **git clone**, following instructions below specific to your operating system.

Open Terminal, and run the following commands:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-who.git
```

### *Step 3. Set up the Tools*

Aside from the ESP-IDF, you also need to install the tools used by ESP-IDF, such as the compiler, debugger, Python packages, etc.

ESP-IDF Tools Installer for Windows introduced in Step 1 installs all the required tools.

If you want to install the tools without the help of the ESP-IDF Tools Installer, open the Command Prompt and follow these steps:

```
cd ~/esp/esp-idf
./install.sh
```

- Customizing the tools installation path

The scripts introduced in this step install compilation tools required by ESP-IDF inside the user home directory:

**\$HOME/.espressif** on Linux and macOS, **%USERPROFILE%.espressif** on Windows. If you wish to install the tools into a different directory, set the environment variables **IDF\_TOOLS\_PATH** before running the installation scripts. Make sure that your user has sufficient permissions to read and write this path.

If changing the **IDF\_TOOLS\_PATH**, make sure it is set to the same value every time the **install.bat/install.sh** and **export.bat/export.sh** scripts are executed.

#### *Step 4. Set up the Environment Variables*

The installed tools are not yet added to the PATH environment variable. To make the tools usable from the command line, some environment variables must be set. ESP-IDF provides another script which does that.

ESP-IDF Tools Installer for Windows creates an "ESP-IDF Command Prompt" shortcut in the Start Menu. This shortcut opens the Command Prompt and sets up all the required environment variables. You can open this shortcut and proceed to the next step.

Alternatively, if you want to use ESP-IDF in an existing Command Prompt window, you can run:

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

Note the space between the leading dot and the path!

You can also automate this step, making ESP-IDF tools available in every terminal, by adding this line to your `.profile` or `.bash_profile` script.

#### *Step 5. Start a Project*

Now you are ready to prepare your applications for ESP32. You can start with `single_chip/recognition_solution` project from examples directory in IDF.

Copy `single_chip/recognition_solution` to `~/esp` directory:

```
cd ~/esp
xcopy /e /i %IDF_PATH%\examples\ single_chip/recognition_solution
```

There is a range of example projects in the examples directory in ESP-WHO. You can copy any project in the same way as presented above and run it.

It is also possible to build examples in-place, without copying them first.

#### *Step 6. Connect Your Device*

Now connect your ESP32 board to the computer and check under what serial port the board is visible.

Serial ports have the following patterns in their names:

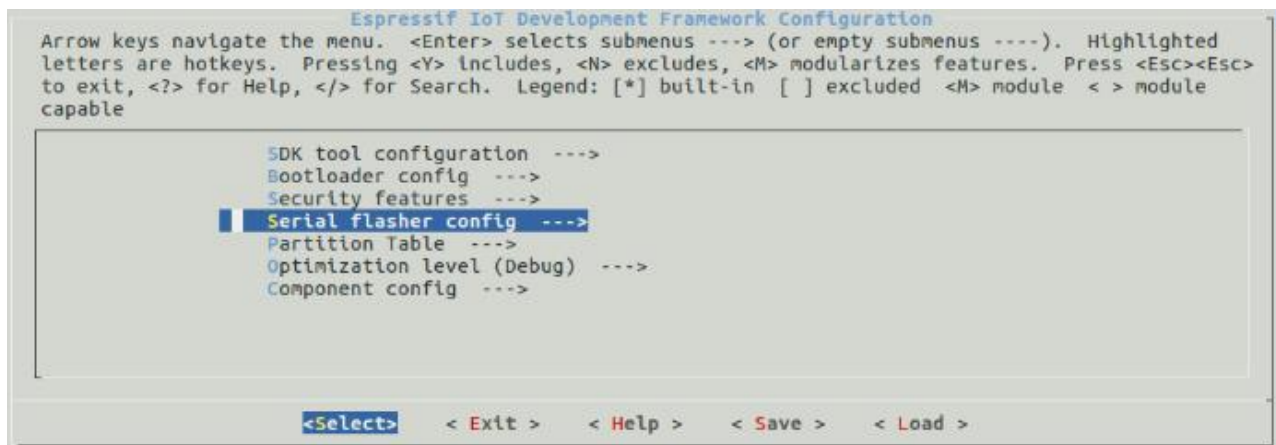
Linux: starting with /dev/tty

#### *Step 7. Configure*

Navigate to your **recognition\_solution** directory from Step 5 Start a Project and run the project configuration utility menuconfig.

```
cd ~\esp\ recognition_solution
idf.py menuconfig
```

If the previous steps have been done correctly, the following menu appears:



```
Espressif IoT Development Framework Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc>
to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module
capable

  SDK tool configuration --->
  Bootloader config --->
  Security features --->
  Serial flasher config --->
  Partition Table --->
  Optimization level (Debug) --->
  Component config --->

<Select>  < Exit >  < Help >  < Save >  < Load >
```

#### *Step 8. Build the Project*

Build the project by running:

```
idf.py build
```

This command will compile the application and all ESP-IDF components, then it will generate the bootloader, partition table, and application binaries.

```
$ idf.py build
```

```
Running cmake in directory /path/to/ recognition_solution /build
```

```
Executing "cmake -G Ninja --warn-uninitialized /path/to/ recognition_solution "...
```

```
Warn about uninitialized values.
```

```
-- Found Git: /usr/bin/git (found version "2.17.0")
```

```
-- Building empty aws_iot component due to configuration
```

```
-- Component names: ...
```

```
-- Component paths: ...
```

```
... (more lines of build system output)
```

```
[527/527] Generating hello-world.bin
```

```
esptool.py v2.3.1
```

Project build complete. To flash, run this command:

```
../..../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600 write_flash  
--flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/  
recognition_solution.bin build 0x1000 build/bootloader/bootloader.bin 0x8000  
build/partition_table/partition-table.bin  
or run 'idf.py -p PORT flash'
```

If there are no errors, the build will finish by generating the firmware binary.bin file.

### *Step 9. Flash onto the Device*

Flash the binaries that you just built onto your ESP32 board by running:

```
idf.py -p PORT [-b BAUD] flash
```

Replace PORT with your ESP32 board's serial port name from Step 6 Connect Your Device.

You can also change the flasher baud rate by replacing BAUD with the baud rate you need. The default baud rate is 460800.

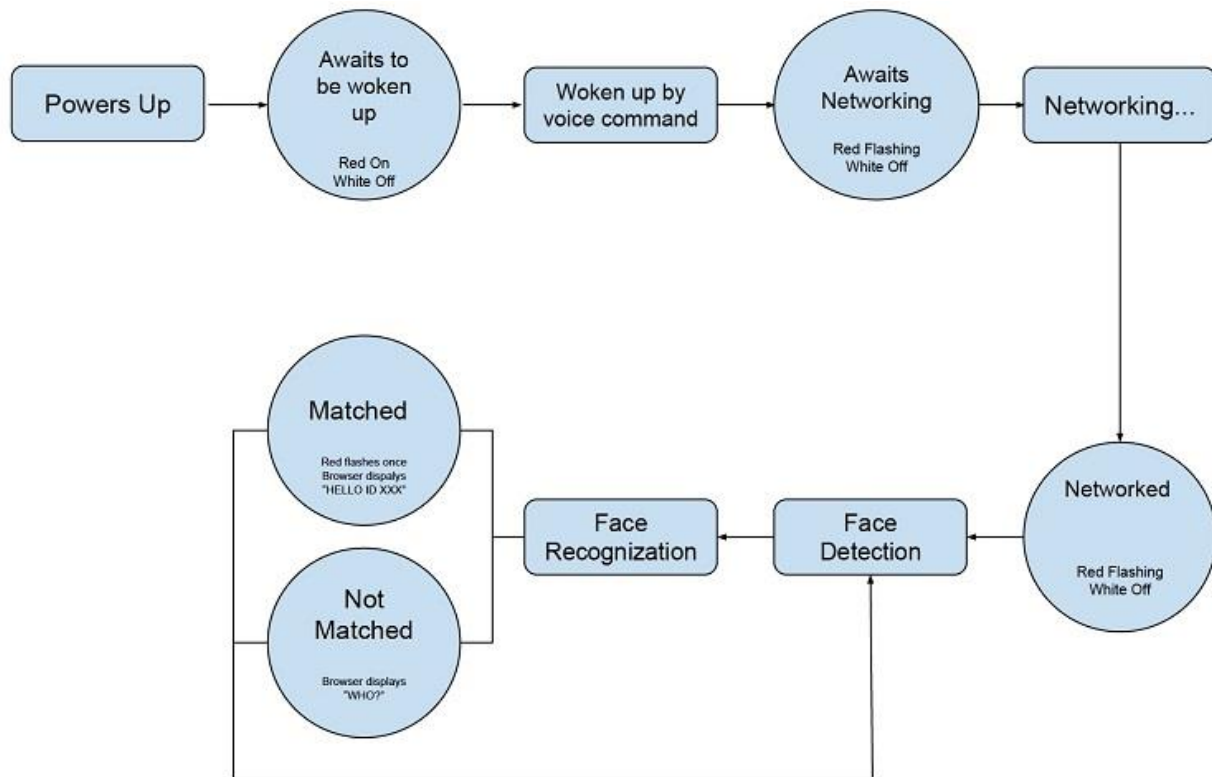
```
Running esptool.py in directory [...]esp/ recognition_solution
Executing "python [...]esp-idf/components/esptool_py/esptool/esptool.py -b 460800 write_flash
@flash_project_args"...
esptool.py -b 460800 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x1000
bootloader/bootloader.bin      0x8000      partition_table/partition-table.bin      0x10000
recognition_solution.bin
esptool.py v2.3.1
Connecting....
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 22992 bytes to 13019...
Wrote 22992 bytes (13019 compressed) at 0x00001000 in 0.3 seconds (effective 558.9 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 5789.3 kbit/s)...
Hash of data verified.
Compressed 136672 bytes to 67544...
Wrote 136672 bytes (67544 compressed) at 0x00010000 in 1.9 seconds (effective 567.5 kb|t/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

If there are no issues by the end of the flash process, the module will be reset and the "recognition\_solution" application will be running.

## Function Test

After Flash onto the Device, The development board will run normally. Development board workflow as shown in the figure.



### Wake up through voice

After all the previous steps are completed, the board will be powered on, and the board will enter the "Wait for Wake-up" state (the red light is always on and the white light is off), and the user needs to wake up by voice.

Support "Hi Le Xin" wake up, when the user says "Hi Le Xin", the development board wakes up and enters the "waiting for networking" state (red light flashes, white light often off). At this point, the user can perform networking operations.



## Connect to Network

Users can connect to Wi-Fi hotspots created by ESP-EYE through PC and mobile. The default information for this hotspot is as follows:

- Username: esp-eye-xxxx (xxxx is the device MAC address)
- Password: No password

## Face Detection

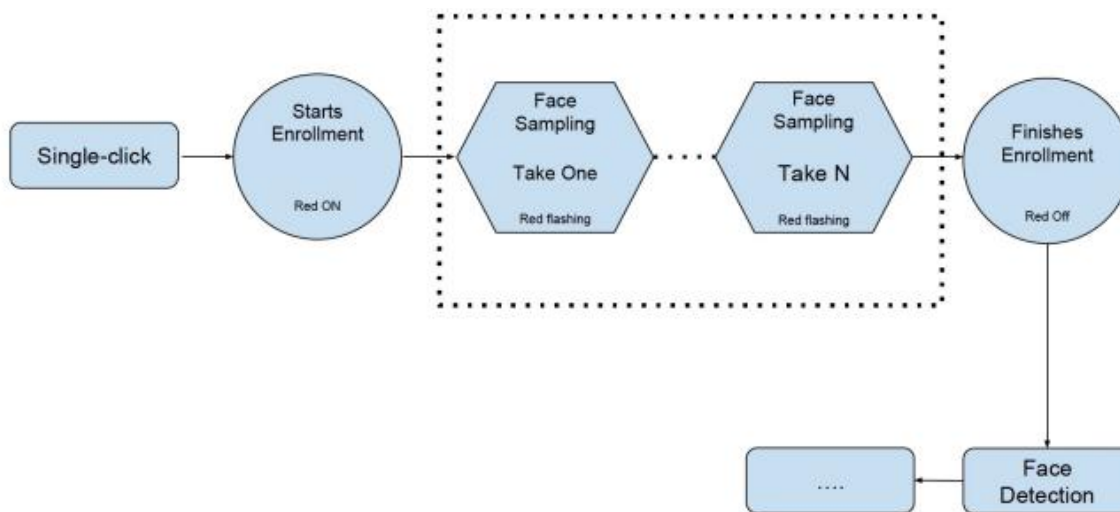
After successful networking, ESP-EYE will perform "face detection". Users can open a browser and enter the address 192.168.4.1/face\_stream to see live image information on the web page. At this point, the development board red light is off and the white light is always on.

## Face identification

When the development board detects a human face, the development board will perform "face recognition".

- Matching success: Red light flashes 1 time, the web page displays HELLO ID XXX
- Match failed: Web page shows WHO?

## Entry the Face ID



- The user clicks the side touch button to enter the "Enter Face ID" (the red light is always on), and the web page displays: START ENROLLING;
- The user faces the camera and starts to collect portraits.
- Each time a successful acquisition, the development board will flash red, and the web page will display the corresponding number of acquisitions, such as THE 1st SAMPLE.
- By default, users need to capture 3 portraits (configurable) for each Face ID entered.
- During the portrait collection process, if the red light does not flash for a long time, it is recommended that the user adjust the posture and angle, and then try again;
- After the portrait is collected, the red light on the development board is always off, indicating that the Face ID has been entered.
- At this time, the web page displays: ENROLLED FACE ID xxx; after the Face ID is successfully entered, the system will return "Face Detection".

## Delete the Face ID

- The user double-clicks the side touch button to enter the "delete FACE ID";
- After double-clicking, the white light of the development board flashes, and the system will automatically delete the oldest FACE ID existing in the system. The terminal displays: XXX ID(S) LEFT

## Other Situations

When there is a situation such as "network disconnection" or "network timeout", the development board will return to the "waiting for wakeup" status.

## FAQ

---

For any questions, advice or cool ideas to share, please visit the [DFRobot Forum](#)

## More Documents

---

[-ESP-EYE Getting Started Guide](#)