Mongoose OS

An open source operating system for hardware that support javascript. It follows event-driven architecture with a non-blocking I/O model.

Features

* Supported microcontrollers: ESP32, ESP8266, CC3220, CC3200
* based on Mongoose library for networking
* APIs for GPIO, PWM and other peripherals
* Built in support for IoT cloud integration
* Manage devices using Remote Procedure Call(RPC)
* Write firmware in Javascript or in C
* Supports Over the Air(OTA) firmware update

For further information on Mongoose OS is available on the following link

Mongoose OS components

- **mos tool** - device management and firmware building
- **build toolchain (only for building firmware offline)** - Docker image for building a mongoose OS app (normally happens in cloud unless specified otherwise)
- **ready-to-use apps and libraries**

Hardware and Software

- Hardware required: ESP8266 or ESP32
- Software required: mos tool

Installing mos tool

Operating System	**Installation Procedure**
Windows	Create `C:\mos` folder. Right-click on this mos.exe link, choose "Save link as", save "`mos.exe`" into the `C:\mos` folder. Double-click on `mos.exe` to start a Web UI. If it does not start, open command prompt, enter `cd c:\mos` and then `mos --start-webview=false`

Ubuntu Linux	`sudo add-apt-repository ppa:mongoose-os/mos`
	`sudo apt-get update`
	`sudo apt-get install mos`
	`mos`

For a list of other operating systems and their installation procedure follow this link https://mongoose-os.com/docs/mongoose-os/quickstart/setup.md#1-download-and-install-mos-tool

Running the demo-js app with mongoose tool

Using Mongoose OS is easy. Follow steps 2 to 7 from below link to become familiar with mongoose to flash a firmware.
https://mongoose-os.com/docs/mongoose-os/quickstart/setup.md#2-start-mos-tool

Mongoose OS app structure

1. mos.yml
2. fs/init.js (if app is developed using JavaScript)
3. src/main.c (if app is developed using C)

Important links to get a better understanding of mos

1. **Building a custom app unlike demo-js in javascript**
   https://mongoose-os.com/docs/mongoose-os/quickstart/develop-in-js.md
2. **Concepts to know** https://mongoose-os.com/docs/mongoose-os/userguide/intro.md and other links in the user guide. Important ones are RPC Mechanism, Device Config, Build Process.
3. **GPIO** https://mongoose-os.com/docs/mongoose-os/api/core/mgos_gpio.h.md
4. **Config** https://mongoose-os.com/docs/mongoose-os/api/core/mgos_sys_config.h.md
5. **Timers** https://mongoose-os.com/docs/mongoose-os/api/core/mgos_timers.h.md
6. **MQTT** https://mongoose-os.com/docs/mongoose-os/api/net/mqtt.md
7. Finally our **repository** https://github.com/manjrekarom/iot-workshop

Using mos tool

Command for building app

```
mos build --arch esp8266
```

This will trigger a remote build process and create a new **build** folder with the fs(filesystem) and fw(firmware) folders required for the app.

Flashing the firmware and filesystem

Connect ESP/NodeMCU to USB port with the provided cable. New serial device will be detected in host system, generally as **/dev/ttyUSB0**. To flash the code type following command in the terminal.

```
sudo mos flash
```

In Linux, for flashing the firmware mos tool uses **ttyUSB0** serial port of the system.
To see the available serial ports in host system, you can use `ls /dev/ttyUSB` command.
If device is detected other than **ttyUSB0**, change the default serial port by specifying the value manually with **--port** flag.

```
sudo mos flash --port /dev/ttyUSB1
```

More mos commands
Help on use of various mos commands: `mos --help`
Setting the wifi connection with access point: `mos wifi` \<Wifi SSID\> \<Wifi Password\>
Viewing the console log of running app for debugging: `mos console`
List the filesystem in the device: `mos ls`
List the configuration setting for all peripherals and app: `mos config-ge`t
Changing the value in the configuration file: `mos config-set \<config tag = value\>`
e.g. for enabling mqtt - `mos config-set mqtt.enable=true`