mariner

Luiz Ribeiro

Feb 21, 2022

CONTENTS

1 Features **3**

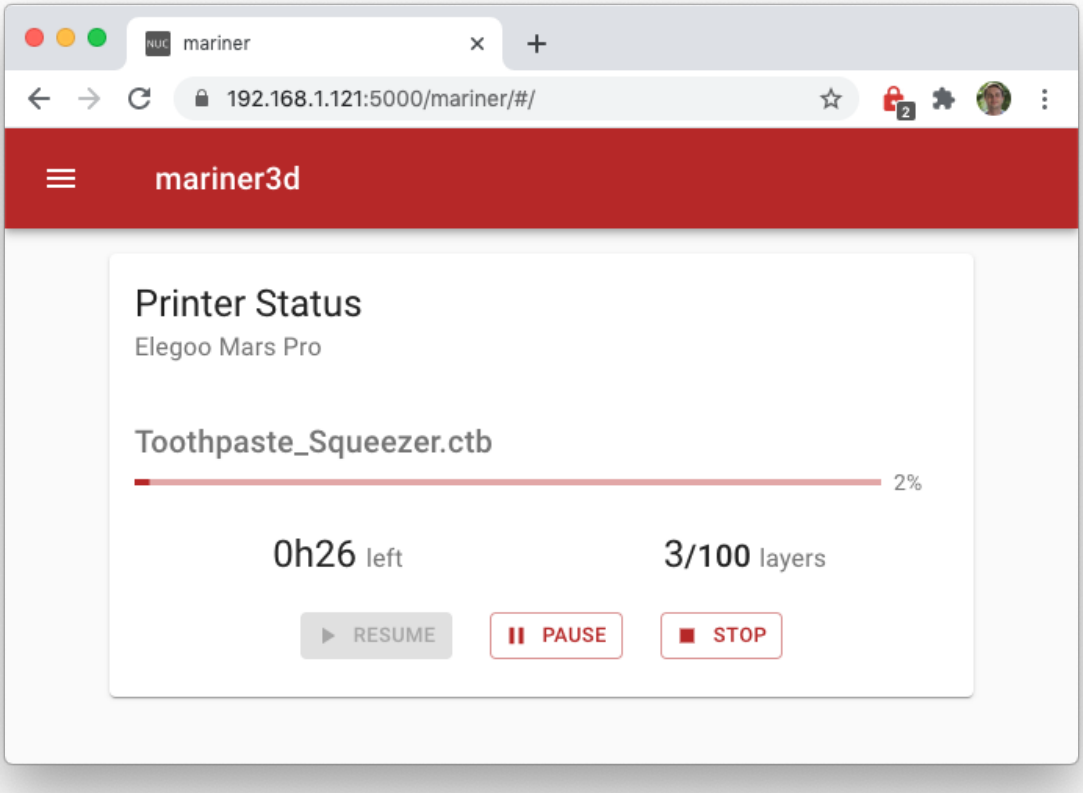
1.1 Supported Printers 3

1.2 Installation Guide 4

1.3 Troubleshooting 6

1.4 Contributing 8

Mariner is a web interface for controlling MSLA 3D Printers based on ChiTu controllers. These are controllers commonly used on 3D Printers by many brands such as Elegoo and Phrozen, making mariner compatible with a wide range of printers.



CHAPTER

ONE

FEATURES

Mariner provides the following features:

- Wide range of *supported MSLA 3D printers*.
- Web interface with support for both desktop and mobile.
- Upload files to be printed through the web UI over WiFi!
- Remotely check print status: progress, current layer, time left.
- Remotely control the printer: start prints, pause/resume and stop.
- Browse files available for printing.
- Inspect `.ctb`, `.cbddlp` and `.fdg` files: including image preview, print time, slicing settings and other metadata.

1.1 Supported Printers

The following printers have been confirmed to work with mariner by our community:

- Anycubic Photon
- Elegoo Mars
- Elegoo Mars Pro
- Elegoo Mars 2
- Elegoo Mars 2 Pro
- Elegoo Saturn
- Phrozen Sonic Mighty 4K
- Phrozen Sonic Mini 4K
- Creality LD-002H
- Creality LD-002R
- Voxelab Proxima
- Peopoly Phenom L
- EPAX E10/X10

More printers probably work with mariner without any changes to the software. Let us know if you succeed on using mariner with any other printers!

1.2 Installation Guide

This document will guide you through the process of setting up mariner with your printer: from the hardware setup, installing the mariner software, and setting up Linux to share files with the printer and use the serial port.

Note: This software is provided "as is", without warranty of any kind. There is no guarantee this will work on your printer, nor that it won't damage it. Use at your own risk.

If you are ready to begin, head on to *Requirements* to get started!

1.2.1 Requirements

Before you get started, make sure you have the following:

- **Raspberry Pi Zero W, Zero WH, 3A+ or 4B**. See Wikipedia's article about the Raspberry Pi specifications for a list of which models support USB OTG.
- **A supported printer.** Most MSLA 3D printers with a ChiTu board should work. We keep a list of tested 3D Printers on our documentation.
- **A USB cable** to connect between Pi and printer mainboard, please use:
  - Micro-USB to USB-A male *if using a Pi Zero W*
  - USB-C to USB-A male *if using a Pi 4B*
  - USB-A male to USB-A male *if using a Pi 3A+*
- **Power Supply for the Pi**, or a 12V to 5V converter to power it from printer's power supply. Some users have also have had success with using 5V pins from the printer mainboard itself. However, that sometimes creates issues with poor WiFi signal, making uploading files too slow or it can also cause serial connection drops.
- **Micro-SD card** faster is better and size is depending on how much space you want to have for the printed files. 2GB would likely work but 4GB or 8GB would probably me a better start.
- **An hour or two** and some basic linux knowledge :-)

1.2.2 Hardware setup

These are high level instructions for the hardware setup:

1. Setup a Raspberry Pi Zero W, Pi 4B or Pi 3A+ with ssh access over WiFi
2. Wire it to the serial port on the ChiTu mainboard. Do not connect the 5V line unless you want your printer's board to power your Pi (might work with Pi Zero W, other boards probably won't)
3. Connect the USB OTG port on the Pi to the USB port of the mainboard. Do not connect the 5V line. You can put some tape on the connector to isolate the 5V line of the USB cable. If using a Pi 3A+, you will need a USB A male to USB A male cable.
4. Connect Pi's USB PWR port to a power supply. You can also use a 12V to 5V converter from the printer's power supply. Do not connect if you plan on powering from the printer's mainboard.

For more detailed instructions for the Elegoo Mars Pro, refer to this blog post. The setup for other printers should be almost identical.

Once you have your hardware setup, it's time to *setup your software*.

1.2.3 Software Setup

Once your *hardware setup* done, you will have to:

1. Install the `mariner3d` Debian package
2. Setup the USB Gadget driver so that the printer can see uploaded files
3. Enable the serial port so the Raspberry Pi can send commands to the printer

This section will guide you through those steps.

Installing package

First, enable the repository:

```
$ curl -sL gpg.l9o.dev | sudo apt-key add -
$ echo "deb https://ppa.l9o.dev/raspbian ./" | sudo tee /etc/apt/sources.list.d/l9o.list
$ sudo apt update
```

Then install mariner:

```
$ sudo apt install mariner3d
```

USB Gadget Setup

In order to make the printer see the files uploaded to mariner, we need to setup the USB Gadget driver as a Mass Storage device. This section will guide you through that process.

Enable USB driver for gadget modules by adding this line to `/boot/config.txt`:

If you are using a Pi Zero W or a Pi 4B add:

```
dtoverlay=dwc2
```

If you are using a Pi 3A+, there is a little variant as these supports device mode or host mode, but not "true" OTG which is auto-sensing between host and device (AKA gadget). So, for the Pi 3A+ you have to add:

```
dtoverlay=dwc2,dr_mode=peripheral
```

Enable the dwc2 kernel module, by adding this to your `/boot/cmdline.txt` just after `rootwait`:

```
modules-load=dwc2
```

Setup a container file for storing uploaded files, the `count=` is in MB, use multiples of 1024 to get the number of GBs you want:

```
$ sudo dd bs=1M if=/dev/zero of=/piusb.bin count=2048
$ sudo mkdosfs /piusb.bin -F 32 -I
```

Create the mount point for the container file:

```
$ sudo mkdir -p /mnt/usb_share
```

Add the following line to your `/etc/fstab` so the container file gets mounted on boot:

```
/piusb.bin /mnt/usb_share vfat users,gid=mariner,umask=002 0 2
```

Finally, make `/etc/rc.local` load the `g_mass_storage` module. If that file doesn't exist yet, create it with the following contents:

```
#!/bin/sh -e

modprobe g_mass_storage file=/piusb.bin stall=0 ro=1

exit 0
```

If the file exists, you should simply add the `modprobe` line to it.

Once you restart the pi (or potentially run `sudo mount -a`), the printer should start seeing the contents of `/mnt/usb_share`.

Setting up the serial port

First, enable UART by adding this to `/boot/config.txt`:

```
enable_uart=1
```

In order for the Pi to communicate with the printer's mainboard over serial, you also need to disable the Pi's console over the serial port:

```
$ sudo systemctl stop serial-getty@ttyS0
$ sudo systemctl disable serial-getty@ttyS0
```

Lastly, remove the console from `cmdline.txt` by removing this from it:

```
console=serial0,115200
```

1.2.4 Wrapping up

Reboot the Pi and you should be all set. Again these are rough instructions for now :)

You can check that the `mariner3d` service is running with:

```
$ sudo systemctl status mariner3d
```

If it is, you should be able to access it by opening `http://<pi ip address>:5000/` on your browser.

1.3 Troubleshooting

1.3.1 Common Issues

Slow WiFi

If uploading or accessing mariner is too slow, the following might help:

- If you are powering your Raspberry Pi from a 5V pin on the printer's mainboard, try using an external power supply or a buck converter from the printer's 12V line. WiFi reception has been reported to be weaker when powering the RPi directly from the printer's mainboard.
- Use an external antenna.
- Try moving the RPi to outside of the printer case.
- Try moving your RPi closer to your router, consider using ethernet or a different model such as the RPi 4B, which has onboard ethernet.

Disk init fail

If your printer is returning an error such as `//Disk init fail!\r\n`, it means your Raspberry Pi is not properly configured to use Linux's USB Gadget mode and your printer is failing to see the virtual disk shared by the RPi. Please refer to the *USB Gadget Setup* section of the documentation and check your setup carefully.

Often, this can be one of these issues:

- **Check for bad USB cables.** Your Micro USB cable should be able to transfer data. There are a lot of USB cables out there that don't have the D+/D- data lines connected and only have the 5V and GND pins connected. Check that your USB cable works with other devices!
- Your `/etc/rc.local` isn't setup correctly with the `modprobe g_mass_storage` line (see Issue #412 for an example).
- Make sure `dtoverlay=dwc2` is under the `[all]` section of your `/boot/config.txt`.
- Make sure your Raspberry Pi supports USB OTG.

Unexpected Printer Response

If you are seeing the "Unexpected Printer Response" while printing from the web interface (but printing is working fine), then you are running into Issue #180, which is being worked on.

This has no performance effect and won't cause issues with your prints. It should cause nothing but cosmetic issues on the user interface and can safely be ignored. Just refresh the page if you run into it.

No such file or directory: /dev/serial0

If you are seeing this on the logs for the `mariner3d` service on `journalctl`, this means your serial port is not setup properly. Please refer to the *Setting up the serial port* section of the installation guide.

Error: File is not existed or empty

If your printer is returning responses such as `Error: File is not existed or empty`, make sure to upgrade to the latest version of mariner. This is an old issue which was fixed on `v0.2.0`. See Issue #311.

Printer stops during print

If your printer is stopping during the print process, make sure to upgrade to the latest version of mariner. This is an old issue which was fixed on `v0.2.0`. See Issue #311.

1.3.2 Other Issues

If your issue isn't listed above, check the `mariner3d` logs for clues:

```
$ sudo journalctl -ub mariner3d.service
```

Also check the message buffer of the Linux kernel for errors:

```
$ sudo dmesg
```

Carefully read the *Installation Guide* once again and, if none of that helps, file an issue on GitHub!

1.4 Contributing

Contributions to mariner are welcome! Both code and documentation are hosted on our GitHub repository. If you are not familiar with GitHub and Pull Requests, we recommend for you to read GitHub's documentation.

This document will guide you through setting up your development environment to *develop mariner itself* and *write documentation*.

Before diving into the sections below, make sure you have a checkout of the mariner git repository:

```
$ git clone https://github.com/luizribeiro/mariner.git
```

1.4.1 Development

Mariner is developed largely on top of Python, TypeScript and React.

If you are interested to help with development, first make sure that you have Python 3.7 (or newer) installed. We also use Poetry for dependency management, so you should have that installed as well.

If you would like to do frontend development, you should have node.js and the yarn package manager.

Once you have those dependencies installed, you can get your virtual environment setup with all the required Python dependencies with `poetry`:

```
$ poetry install
```

Backend

The mariner backend code is responsible for serving HTTP requests from the frontend and communicating with the printer. It's the core of everything and written in Python. All of the code is within the `mariner` directory of the Git repository.

If you would like to run mariner locally, you can do so with the following command:

```
$ poetry run mariner
```

Note: Before running mariner locally, you will want to build the frontend's static resources first. Make sure to follow the instructions from the *Frontend* section below.

The Pi Zero is a bit too slow for development, so we recommend you to do development from another machine or use a Raspberry Pi 4 on your printer which will help considerably, since those models support OTG.

We use green for running backend tests. It is installed by Poetry along with the development dependencies, so you can simply run the backend tests with this command:

```
$ poetry green
```

In addition to running tests, you will want to make sure your code:

- Passes all Flake8 checks: `poetry run flake8`
- Is auto-formatted with Black: `poetry run black --check .`
- Is type-checked with pyre: `poetry run pyre`

If you are not familiar with Python type-checking it is recommended for you to get familiarized with it first, as the mariner code is strictly typed. PEP 484 offers a good overview of the basic functionality and the `typing` module documentation is a great resource as well.

Frontend

The mariner frontend code is largely written in TypeScript with React. Most of the frontend is built with Material-UI components. HTTP requests are made to the backend API endpoints with Axios. We use yarn to manage our frontend dependencies, so make sure to have that installed before you move forward.

All of the frontend source code lives in the `frontend` directory. So `cd` into that directory and install the JS dependencies with:

```
$ yarn install
```

In order to run `mariner` locally, you will want to package the frontend code into a static resource that can be served by the backend. Do this with:

```
$ yarn build
```

If successful, a `main.js` will be generated under the `dist` directory. Now you should be able to run mariner locally with `poetry run mariner` as described on the *Backend* section.

You can run the frontend tests with the following command:

```
$ yarn --cwd frontend test
```

Just like on the backend, we use a few tools to hold the style guidelines and best practices of the codebase. So make sure to do these checks on your code:

- ESLint issues with: `yarn lint`
- Formatting issues with prettier: `yarn run prettier --cwd src`
- Typing issues with: `yarn run tsc`
- Passes all Flake8 checks: `poetry run flake8`
- Is auto-formatted with Black: `poetry run black --check .`
- Is type-checked with pyre: `poetry run pyre`

Pre-commit Hooks

We run tests, style and typing checks automatically on all commits and pull requests with a GitHub Action. However, you can also automatically run those locally before every commit you make with pre-commit. See pre-commit's installation instructions for more information. Once you have `pre-commit` installed, setup the mariner hook by running this from your local repository:

```
$ pre-commit install
```

Once the pre-commit hook is installed, any commits to your local mariner repository will automatically run all backend and frontend checks that would be run on the GitHub Action.

1.4.2 Documentation

All of mariner's documentation (including this document!) is hosted on the mariner repository itself under the `docs/` directory. We use Read the Docs to host our documentation, which is formatted with reStructuredText.

If you would like to make changes to the documentation, you can make those from the GitHub UI itself and submit those as Pull Requests without requiring any local development.

You can also make changes to the documentation and preview them locally. All you need is a local checkout of the `mariner` repository from GitHub, Python 3.7 (or newer), and Poetry installed. For more information on that setup, refer to the *Backend* section as the instructions are the same.

To build the documentation locally, just run the following commands:

```
$ cd docs
$ make html
```

Then just open `docs/_build/html/index.html` on your browser.

If you are building the documentation on Windows, there is a `make.bat` file which you can run instead of `make html`.