



5/4/2018

# VandalForge Printer Software

Team Vulcan – Full Manual



Jonathan Buch, Tim Clemans, Michael Madsen,  
James Young  
UNIVERSITY OF IDAHO

# Table of Contents

Executive Summary .....	4
Background.....	5
Problem Definition .....	5
Project Plan.....	6
Objectives .....	6
Requirements .....	6
Specifications.....	6
Concepts Considered.....	7
Cura Ultimaker (Slic3r Alternative).....	7
Smoothieware Web Client (OctoPrint Alternative).....	8
Concept Selection.....	9
Slic3r .....	9
OctoPrint .....	9
System Architecture .....	10
Future Work .....	11
Slic3r .....	11
OctoPrint .....	11
Appendices .....	12
A: Revised Installation Tutorial for Slic3r (On Windows Platforms) v1.3.....	12
1) Installing a Perl for Windows.....	13
2) Installing the dependencies .....	16
3) Getting the Slic3r source .....	21
4) Building Slic3r .....	23
5) Starting Slic3r.....	28
B: Slic3r – Adding GCode Flavors.....	30
README.md .....	30
slic3r.pl.....	30
utils/zsh/functions/_slic3r.....	30
xs/src/libslc3r/GCodeWriter.cpp.....	30
xs/src/libslc3r/PrintConfig.cpp .....	30
xs/src/libslc3r/PrintConfig.hpp.....	31
C: Altering GCode output for use with new User Inputs.....	32

Slic3r/xs/src/libslc3r/PrintConfig.hpp .....	32
Slic3r/xs/src/libslc3r/PrintConfig.cpp.....	33
Slic3r/xs/src/libslc3r/GCodeWriter.cpp .....	34
lib/Slic3r/GUI/PresetEditor.pm .....	35
D: OctoPrint Documentation.....	37
Helpful Links .....	37
Install .....	37
Add Config File.....	37
To Change to University of Idaho Colors .....	37
More Documentation / Manual .....	37
E: Perl Dependencies.....	39

# Executive Summary

## Overview

The three major parts of the System Design are the Slic3r software, the GCode output, and the OctoPrint Hosting software. The Slic3r software generates the GCode file that is then uploaded for use with OctoPrint.

## Slic3r

We worked on four distinct parts of Slic3r: GCode flavors, graphical user interface (GUI), dependencies, and the algorithm. For the GCode Flavors, we first modified RepRap so we would have a stable starting point. Once we understood how to make our own GCode flavors, we created the MigWelder and TigWelder flavors for the Mark I and Mark II printers respectively. Both of them rely on GCodeWriter.cpp for changing the file output when exporting GCode. With respect to the GUI, the main pieces are PresetEditor.pm which handles the GUI Settings options, PrintConfig.hpp, and PrintConfig.cpp which define the variables used by the GUI Settings options. Moving to the dependencies, there are two sets that are needed for Slic3r to run, the Boost 1.63.0 dependencies and the Perl dependencies stored in local-lib. We took a cursory look at the Boost libraries, but found they weren't necessary for altering the Slic3r. The Perl dependencies, on the other hand, were researched to potentially identify changes that could be made to ease the transition into Python at some point.

## GCode

The file format used by the metal printer that is generated by Slic3r and sent to OctoPrint. The specific changes we created for use with our metal printer flavors include using G0 for the travel-to commands and G1 for the extrude-to commands. For the Mark II printer we added in the I0-100 and C1-9 commands for use with the nine potentiometers.

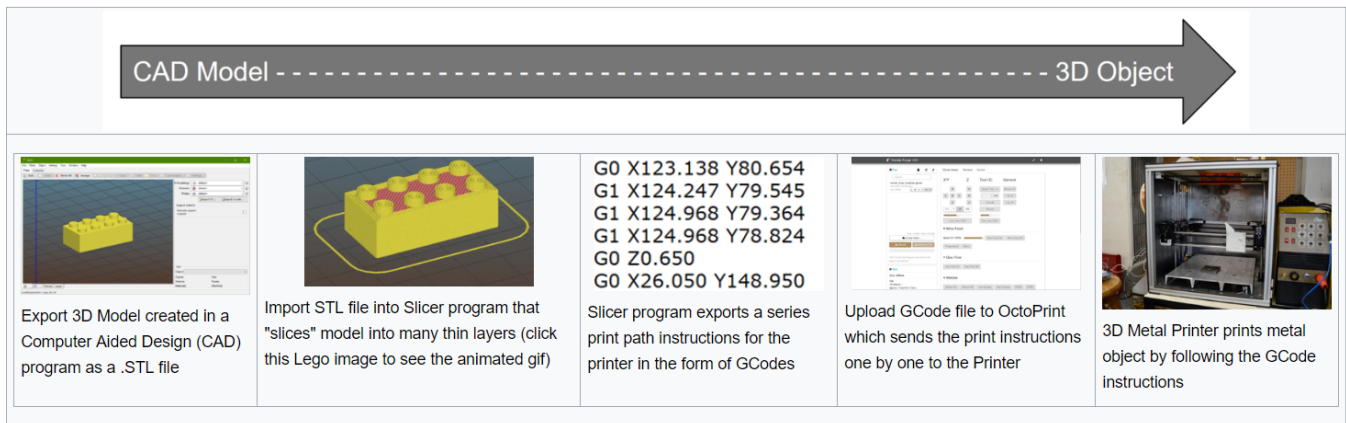
## OctoPrint

The part of OctoPrint that was changed for our project was the GUI. Specifically, this was done through the creation of the VandalPrint plugin and config.yaml file. The VandalPrint plugin is responsible for the changes to the color scheme to match the University of Idaho colors by injecting CSS styles into the HTML GUI via the navigation bar. The config.yaml file adds the TIG welder commands used by the Mark II printer. This includes the nine new digital potentiometers that were added in the Mark II version.

# Background

Current printing software is not customized for a 3D metal printer. Print instructions created by default slicer software contains extra unused commands and does not provide custom commands needed to run a 3D Metal printer. Last year's team had to manually modify the output of Plastic Printing software, which was a slow and laborious process. OctoPrint, the interface between the print instructions and the printer, was originally designed for plastic printers and does not have the necessary features for metal printing. The Mark I and Mark II 3D Metal Printers require a custom print instructions file.

## Problem Definition



# Project Plan

## Objectives

To improve the quality of the VandalForge Software while extending usability to allow further customization for future project endeavors.

- Slic3r produces custom GCode commands
- Octoprint accepts custom commands
- GUI's utilize custom commands
- ForgeWare Integration
- Easy to use
- Create excellent documentation for future teams

## Requirements

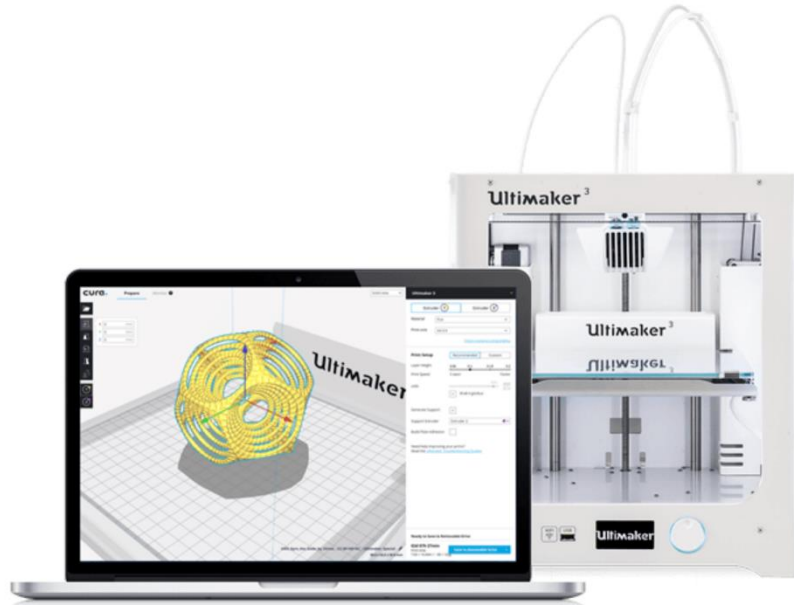
- Use Open Source software
- Modifications must work for both versions of the printer
- Modified software should be user friendly
- Carefully document process so future teams can easily add improvements

## Specifications

No.	Software	Need	Importance
1	GCODE FILE	easily understood	2
2	GCODE FILE	updated with customer variables	5
3	Slic3r	able to handle errors	4
4	Slic3r	extendable by non-CS majors	4
5	Slic3r	well commented	3
6	Slic3r	easily understood	3
7	Slic3r UI	clean/uniform in design	2
8	Slic3r UI	easily understood	4
9	Slic3r UI	visually appealing	3
10	Slic3r UI	easy to navigate	4
11	Host Software	interpret and convey temperature info in real time	3
12	Host Software	able to make use of an IR camera	1
13	Host Software	temperature history visualization	2

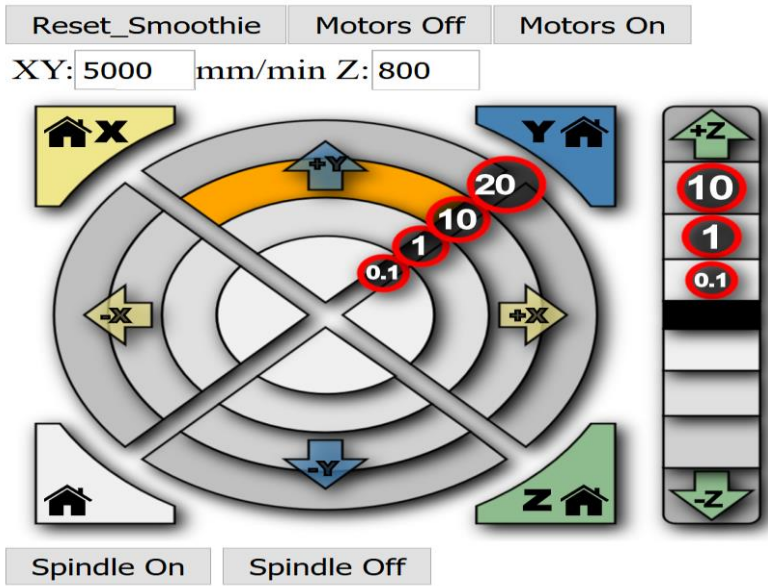
# Concepts Considered

## Cura Ultimaker (Slic3r Alternative)



- **Pro**
  - Written in Python
  - Well documented and developed
  - Sleek user interface
- **Con**
  - Direction of development controlled by Ultimaker Company
  - No guarantees that Ultimaker's continued development of Cura would be useful to the project
  - Designed specifically for Ultimaker Plastic Printers

# Smoothieware Web Client (OctoPrint Alternative)



## XYZ Axis Control

		Wpos	Mpos	Over	
X Axis:	Zero X	0	0	0	Override X
Y Axis:	Zero Y	0	0	0	Override Y
Z Axis:	Zero Z	0	0	0	Override Z

- **Pro**
  - Already exists on the Smoothieboard
  - Simple setup (a single html file)
- **Con**
  - No guarantee it works with other board types
  - Very limited feature set



## Concept Selection

### Slic3r

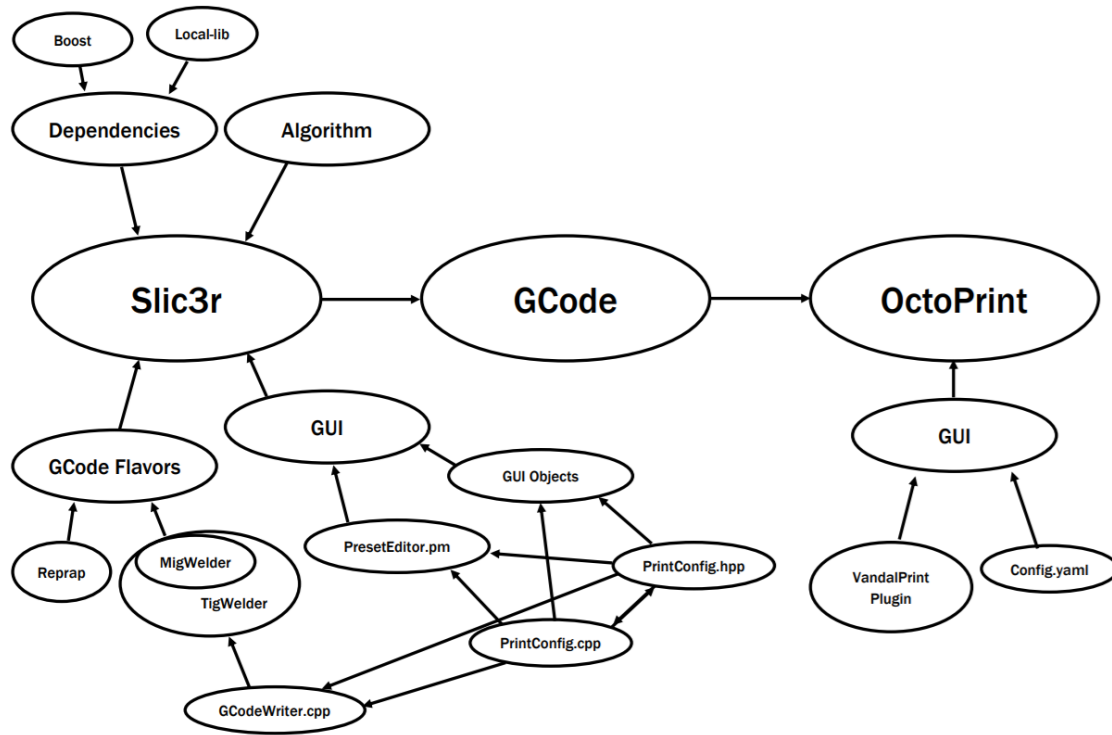
Slic3r was chosen because it is open source, works with a variety of printers and GCode flavors. The previous team had selected Slic3r as the dedicated slicing software for the 3D metal printer. Slic3r also works very closely with our chosen hosting software (OctoPrint).

### OctoPrint

OctoPrint was chosen because it is very popular hosting software, works with dozens of different boards, and is very easy to modify and tailor to your needs. OctoPrint makes it trivial to certain types of improvements, but also allows you to fully customize and tweak any part of it.

# System Architecture

## System Diagram



# Future Work

## Slic3r

- New Print-Path Algorithm
  - Slic3r creates a specific toolpath that functions well with plastic printers but does not work with metal prints. This is due to the large difference between extruding plastic and welding metal materials. Understanding and modifying the printing algorithm will allow Slic3r to create toolpaths that will work well with welding.
- Automatically change print parameters based on current print progress
  - Throughout a print, Slic3r is capable of changing the speed of the print depending on if it is near an edge or the center of the object. We can get more complex prints if the potentiometers could also change their settings dynamically when exporting GCode.

## OctoPrint

- GCodeViewer 2D output of print
  - OctoPrint creates live 2D outputs of plastic prints, if there are no plastic extrusion GCode commands there is no output.
  - In the future we would like to modify the GCodeViewer.js file to recognize our custom metal extrusion commands (G1) and create output based on these. See OctoPrint documentation for location of GCodeViewer.js file.
- Web Camera to watch prints
  - OctoPrint has a plugin available to connect to a web camera.
  - In the future we would like to attach a web camera inside the printer so that OctoPrint could have a live feed of active prints.
- Create 3D wire-frame model of print with temperature/heat map from infrared camera
  - Have OctoPrint create a 3D wireframe preview of what the print will look like, use a thermal camera to show the temperature of the model during the print.

# Appendices

## A: Revised Installation Tutorial for Slic3r (On Windows Platforms) v1.3

by James Young

This document will contain segments from the provided Slic3r tutorial, edited when and where needed, as well as additional notes to aid in the process of installation. Segments taken from the tutorial will be denoted by a box, with additional notes provided underneath. Additionally, this installation is from the standpoint of Windows 10, but the ideas, concepts, and actions taken are general enough for any modern version of Windows.

## 1) Installing a Perl for Windows

You can choose between Citrus Perl or Strawberry Perl. Either will do. They include MinGW which facilitates installing additional Perl modules.

Citrus Perl installs cleanly into a standalone folder and won't touch the Windows system in any ways. MinGW is downloaded separately (though automatically). At the time of writing, 3D preview won't work with Citrus Perl on 64-bit Windows because its MinGW lacks FreeGLUT.

Strawberry Perl comes with the MinGW packaged in an MSI installer, and touch the environment variable such that integrates itself into the Windows shell. Side by side installation of different versions would be difficult if not impossible.

Because of a [known issue](#) that is a combination of newer gcc versions, SJLJ exceptions, and Perl exceptions, use the following custom Strawberry Perl 5.24 build that packages GCC 6.3.0 using SEH exceptions. If using 32-bit, then use regular Strawberry Perl instead.

Strawberry Perl 5.26 should be compatible (as it uses GCC7 and SEH exceptions).

As this section describes, you can use either Citrus or Strawberry Perl for compiling Slic3r, but both have downsides, in either missing functionality or running into problems. For all intents and purposes, the VandalForge project uses the Slic3r Perl variant, which is a modified version of Strawberry Perl. Because of this, this tutorial shall only focus on dealing with Slic3r Perl, but should you need to do otherwise, refer to the source documentation.

### 1.1) Slic3r Perl

- Download [Slic3r Perl](#).
- Install it, following the on screen instruction. Basically that means keep clicking Next.
- Now your command prompt is loaded with Perl. You start the command prompt with...
  - In the Start Menu, All Programs, then Accessories, finally Command Prompt
  - Press Win+R, type cmd, then Enter.
  - For Vista/Windows 7: open the Start menu, type cmd in the search box, then Enter.

As listed, following these instructions, and the correct version of Slic3r Perl<sup>1</sup> should be installed on your machine. Additional screenshots below for clarity.

---

<sup>1</sup> Slic3r Perl URL [https://bintray.com/lordofhyphens/Slic3r/download\\_file?file\\_path=slic3r-perl-5.24.1.4-64bit.msi](https://bintray.com/lordofhyphens/Slic3r/download_file?file_path=slic3r-perl-5.24.1.4-64bit.msi)

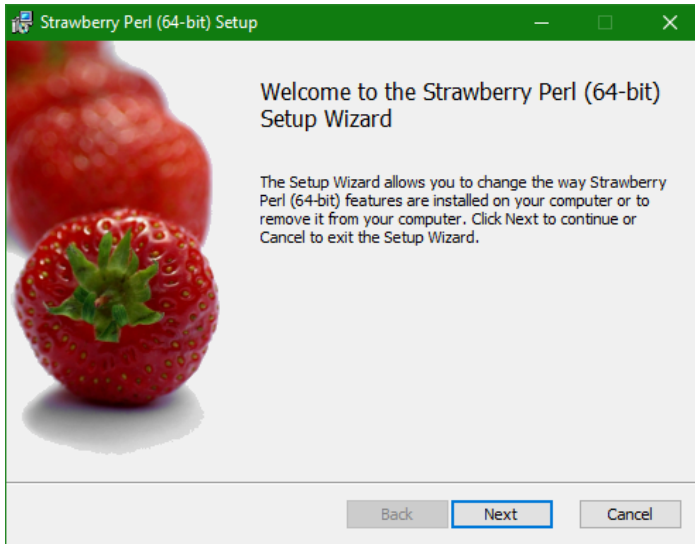


Fig. 1.1) Beginning of the installation process

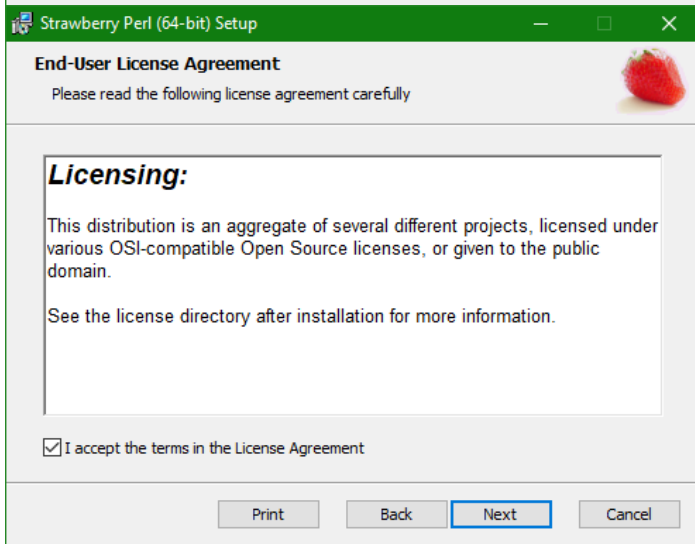


Fig. 1.2) Licensing information

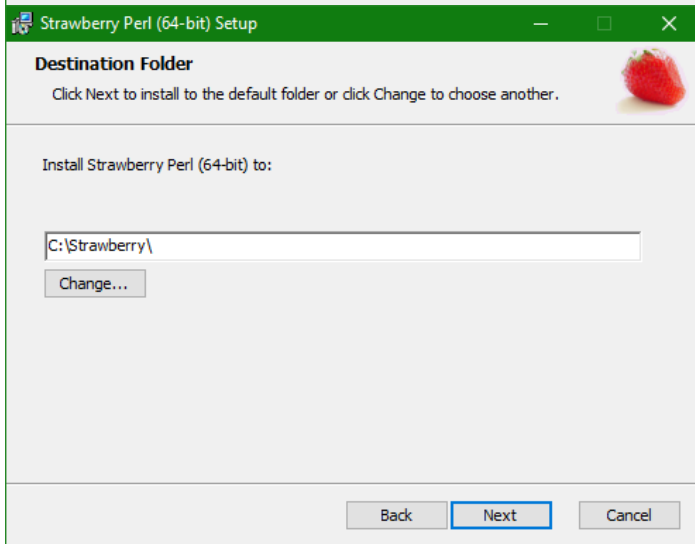


Fig. 1.3) Selecting an installation directory.

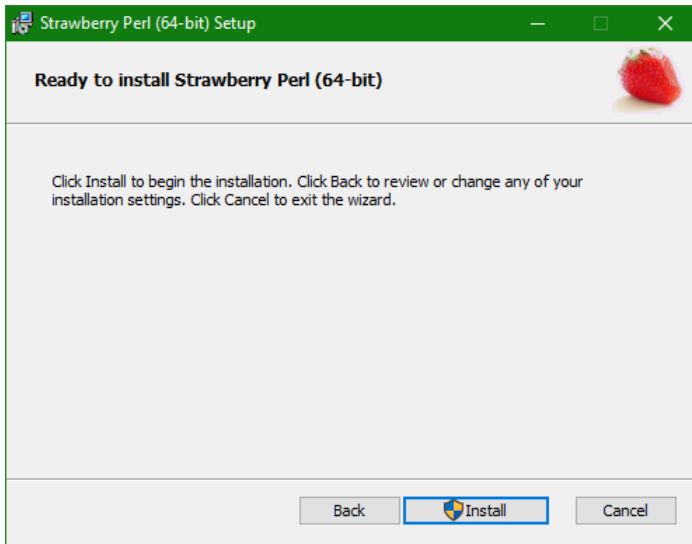


Fig. 1.4) Begin installation

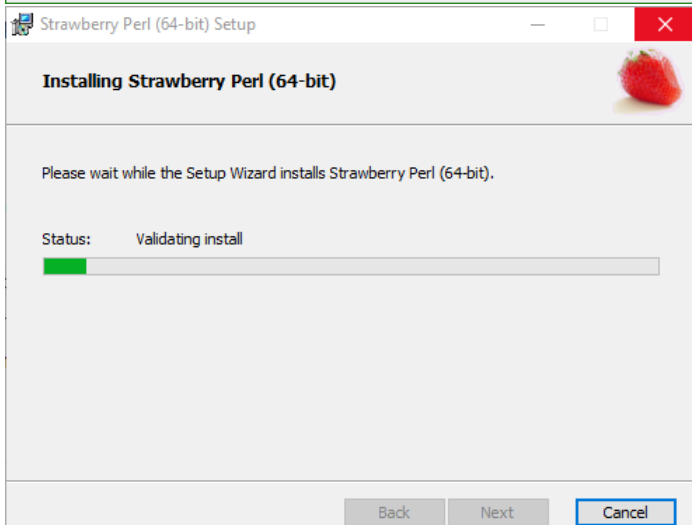


Fig. 1.5) Perl being installed

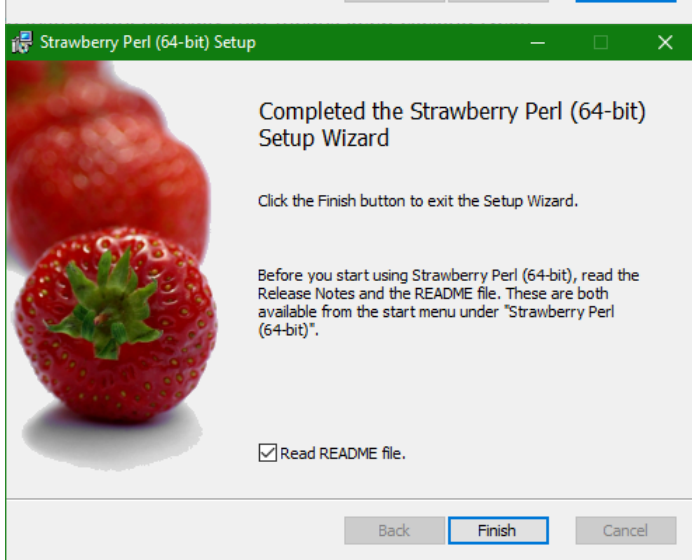


Fig. 1.6) Perl installation complete

## 2) Installing the dependencies

### 2.1) Installing the Boost libraries

Slic3r is known to work with [Boost 1.63.0](#). Note: Dev version prior to 27/03/2017 worked ok with boost 1\_59\_0. If you have that installed, you need to erase the folder c:\dev\boost\_1.59\_0\ and then install and compile the new boost libraries, following the next instructions

The overall information in this section has little to no effect on this project. The only important piece of information is the hyperlink provided for the necessary Boost libraries<sup>2</sup>, which are needed to compile and run Slic3r.

Download them from the Boost website. The official binaries won't work since they're compiled for MSVC. Extract the sources to C:\dev\boost\_1\_63\_0. Run the following instructions from the Perl command prompt

```
cd C:\dev\boost_1_63_0
bootstrap.bat gcc
b2 toolset=gcc cxxflags="-std=c++11" link=static runtime-link=static
variant=release threading=multi
```

A few things to note here, the first being the extract location. The only requirements regarding the extraction point of the files is that the files must be in dev\boost\_1\_63\_0, but you may nest this file path within any other folder or drive, just so long as you can access it via command prompt. Additionally, make sure to remember where these files are, as we may potentially need to know it later.

Another point of note are the commands.

```
cd C:\dev\boost_1_63_0
```

This changes you to the directory where the Boost files were extracted.

```
bootstrap.bat gcc
```

This prepares for the next line of code.

```
b2 toolset=gcc cxxflags="-std=c++11" link=static runtime-link=static
variant=release threading=multi
```

This is all one line, and it will begin the mass compilation of all the libraries required by Slic3r. Again, this is all one line, so **make sure** that it is entered as one in the command prompt.

After these commands are entered, it will take quite some time to compile. While mileage may vary, this can take as long as 30 minutes even on higher end computers. Often, it will look like it has frozen, but do not cancel the process as it is still working. To check this, best way is to open the task manager, view more details, and ensure that there is CPU activity from the command prompt.

---

<sup>2</sup> Boost Libraries URL [https://sourceforge.net/projects/boost/files/boost/1.63.0/boost\\_1\\_63\\_0.7z/download](https://sourceforge.net/projects/boost/files/boost/1.63.0/boost_1_63_0.7z/download)



```
Command Prompt
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\James>cd C:\dev\boost_1_63_0

C:\dev\boost_1_63_0>
```

Fig. 2.1) Begin the installation process by navigating to the boost folder.

```
Command Prompt
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\James>cd C:\dev\boost_1_63_0

C:\dev\boost_1_63_0>bootstrap.bat gcc
```

Fig. 2.2) Run the command to begin building the required engine.

```
Command Prompt - bootstrap.bat gcc
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\James>cd C:\dev\boost_1_63_0

C:\dev\boost_1_63_0>bootstrap.bat gcc
Building Boost.Build engine
```

Fig. 2.3) Expected output.

```
Command Prompt - bootstrap.bat gcc
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\James>cd C:\dev\boost_1_63_0

C:\dev\boost_1_63_0>bootstrap.bat gcc
Building Boost.Build engine
execcmd.c: In function 'onintr':
execcmd.c:120:5: warning: implicit declaration of function 'out_printf' [-Wimplicit-function-declaration]
    out_printf( "...interrupted\n" );
    ^~~~~~
execnt.c: In function 'maxline':
execnt.c:524:12: warning: type defaults to 'int' in declaration of 'result' [-Wimplicit-int]
    static result;
    ^~~~~~
filent.c: In function 'file_archscan':
filent.c:358:10: warning: implicit declaration of function 'filelist_empty' [-Wimplicit-function-declaration]
    if ( filelist_empty( archive->members ) )
    ^~~~~~
filent.c:360:14: warning: implicit declaration of function 'file_collect_archive_content_' [-Wimplicit-function-declaration]
    if ( file_collect_archive_content_( archive ) < 0 )
    ^~~~~~
```

Fig. 2.4) These warnings can be safely ignored, as they do not impact compiling Slic3r.

```
Command Prompt
make.c: In function 'make0':
make.c:735:13: warning: implicit declaration of function 'out_flush' [-Wimplicit-function-declaration]
    out_flush();
    ^~~~~~
fileys.c: In function 'file_archivescan_impl':
fileys.c:360:10: warning: implicit declaration of function 'filelist_empty' [-Wimplicit-function-declaration]
    if ( filelist_empty( archive->members ) )
    ^~~~~~
modules/path.c: In function 'path_exists':
modules/path.c:16:12: warning: implicit declaration of function 'file_query' [-Wimplicit-function-declaration]
    return file_query( list_front( lol_get( frame->args, 0 ) ) ) ?
    ^~~~~~

Bootstrapping is done. To build, run:

    .\b2

To adjust configuration, edit 'project-config.jam'.
Further information:

- Command line help:
  .\b2 --help

- Getting started guide:
  http://boost.org/more/getting_started/windows.html

- Boost.Build documentation:
  http://www.boost.org/build/doc/html/index.html

C:\dev\boost_1_63_0>
```

Fig. 2.5) Building process finished

```
Command Prompt
make.c:735:13: warning: implicit declaration of function 'out_flush' [-Wimplicit-function-declaration]
    out_flush();
    ^~~~~~
fileys.c: In function 'file_archivescan_impl':
fileys.c:360:10: warning: implicit declaration of function 'filelist_empty' [-Wimplicit-function-declaration]
    if ( filelist_empty( archive->members ) )
    ^~~~~~
modules/path.c: In function 'path_exists':
modules/path.c:16:12: warning: implicit declaration of function 'file_query' [-Wimplicit-function-declaration]
    return file_query( list_front( lol_get( frame->args, 0 ) ) ) ?
    ^~~~~~

Bootstrapping is done. To build, run:

    .\b2

To adjust configuration, edit 'project-config.jam'.
Further information:

- Command line help:
  .\b2 --help

- Getting started guide:
  http://boost.org/more/getting_started/windows.html

- Boost.Build documentation:
  http://www.boost.org/build/doc/html/index.html

C:\dev\boost_1_63_0>b2 toolset=gcc cxxflags="-std=c++11" link=static runtime-link=static variant=release threading=multi
```

Fig. 2.6) Begin the compilation of boost files. This will take some time.

```
Command Prompt - b2 toolset=gcc coxflags="-std=c++11" link=static runtime-link=static variant=release threading=multi
oost_atomic-mgw63-mt-s-1_63.a
common.copy stage\lib\libboost_atomic-mgw63-mt-s-1_63.a
bin.v2\libs\atomic\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static\threading-multi\libboost_atomic-
mgw63-mt-s-1_63.a
    1 file(s) copied.
common.mkdir bin.v2\libs\system
common.mkdir bin.v2\libs\system\build
common.mkdir bin.v2\libs\system\build\gcc-mingw-6.3.0
common.mkdir bin.v2\libs\system\build\gcc-mingw-6.3.0\release
common.mkdir bin.v2\libs\system\build\gcc-mingw-6.3.0\release\link-static
common.mkdir bin.v2\libs\system\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static
common.mkdir bin.v2\libs\system\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static\threading-multi
gcc.compile.c++ bin.v2\libs\system\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static\threading-multi\
error_code.o
gcc.archive bin.v2\libs\system\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static\threading-multi\libb
oost_system-mgw63-mt-s-1_63.a
common.copy stage\lib\libboost_system-mgw63-mt-s-1_63.a
bin.v2\libs\system\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static\threading-multi\libboost_system-
mgw63-mt-s-1_63.a
    1 file(s) copied.
common.mkdir bin.v2\libs\chrono
common.mkdir bin.v2\libs\chrono\build
common.mkdir bin.v2\libs\chrono\build\gcc-mingw-6.3.0
common.mkdir bin.v2\libs\chrono\build\gcc-mingw-6.3.0\release
common.mkdir bin.v2\libs\chrono\build\gcc-mingw-6.3.0\release\link-static
common.mkdir bin.v2\libs\chrono\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static
common.mkdir bin.v2\libs\chrono\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static\threading-multi
gcc.compile.c++ bin.v2\libs\chrono\build\gcc-mingw-6.3.0\release\link-static\runtime-link-static\threading-multi\
chrono.o
```

Fig. 2.7) Midway through the compilation process.

```
Command Prompt
- regex : building
- serialization : building
- signals : building
- system : building
- test : building
- thread : building
- timer : building
- type_eraser : building
- wave : building

...patience...
...patience...
...patience...
...patience...
...patience...
...found 11033 targets...

The Boost C++ Libraries were successfully built!

The following directory should be added to compiler include paths:

C:\dev\boost_1_63_0

The following directory should be added to linker library paths:

C:\dev\boost_1_63_0\stage\lib

C:\dev\boost_1_63_0>
```

Fig. 2.8) Boost compilation finished. While no problems have been seen regarding the file and linker paths, remember to record them in case of a failure later on.

## 2.2) Getting the Perl Module Dependencies

For our custom Perl package of 5.24, use [the prebuilt dependency archive](#). It includes wxWidgets 3.1.0, because wxWidgets 3.0.2 does not build with newer GCC versions. See [Building wxWidgets 3.1.0](#) for more information about custom-building wxWidgets if you *really* want to build it yourself.

For Strawberry Perl 5.24.0 (32bit) and 5.18.3 (64bit), there is an archive of the local-lib directory, which speeds things up markedly and avoids some common issues with building dependencies: [https://bintray.com/lordofhyphens/Slic3r/Slic3r\\_Dependencies/Dependencies-1.3.0-dev](https://bintray.com/lordofhyphens/Slic3r/Slic3r_Dependencies/Dependencies-1.3.0-dev)

This portion of the installation process caused a lot of confusion in the initial stages of the project. The important piece to pull out of this is the first link, the prebuilt dependency archive. Download the zip file<sup>3</sup>, and hold onto it for now, as it needs to be put into a specific location mentioned in the next section.

---

<sup>3</sup> Prebuilt Dependency Archive URL [https://bintray.com/lordofhyphens/Slic3r/download\\_file?file\\_path=slic3r-perl-dependencies-5.24.0-win-seh-gcc6.3.0-x64.7z](https://bintray.com/lordofhyphens/Slic3r/download_file?file_path=slic3r-perl-dependencies-5.24.0-win-seh-gcc6.3.0-x64.7z)

### 3) Getting the Slic3r source

The VandalForge code is located on the private GitHub repository established by team Vulcan. You **SHOULD NOT** retrieve the code from the current Slic3r project, as it will not have the work done by our team. However, should it be necessary (as it likely will in the future), all the base Slic3r source code can be pulled from the [Slic3r GitHub](#)<sup>4</sup>.

Once the source code has been obtained, the prebuilt dependency can now be placed within. The local-lib folder should be placed at the top level directory of Slic3r, and it will handle the Wx graphical components for the GUI.

---

<sup>4</sup> Base Slic3r Github, (NOT VandalForge Slic3r) <https://github.com/alexrij/Slic3r/>

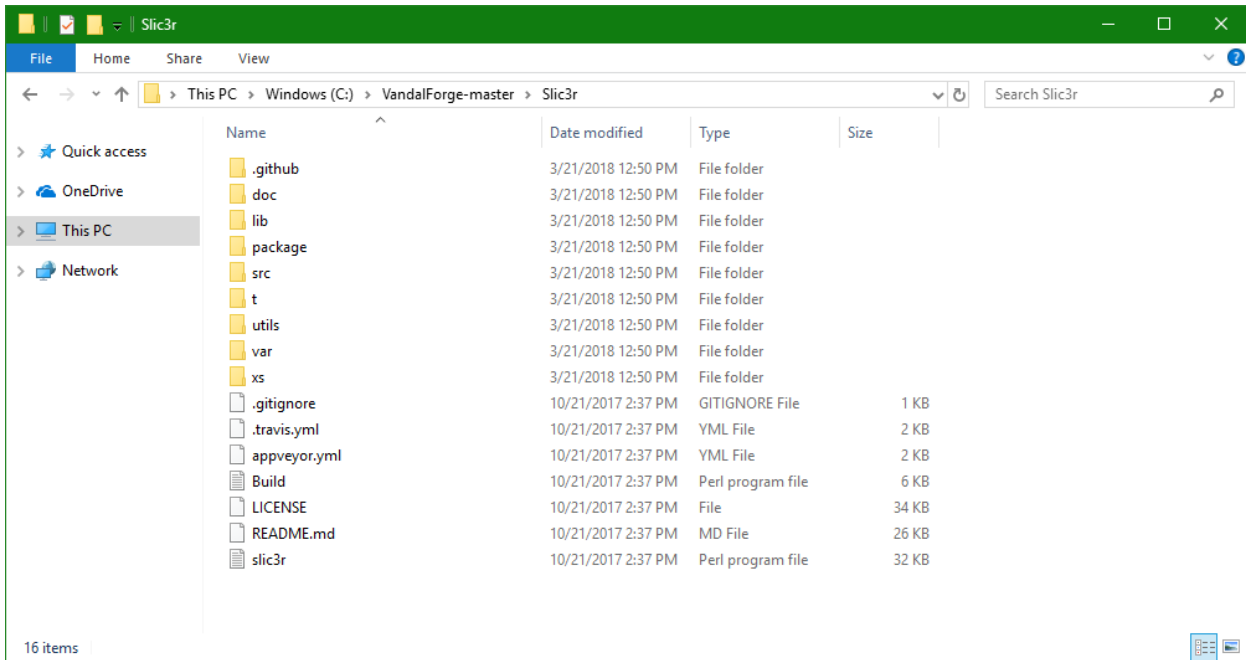


Fig. 3.1) Navigate to this directory...

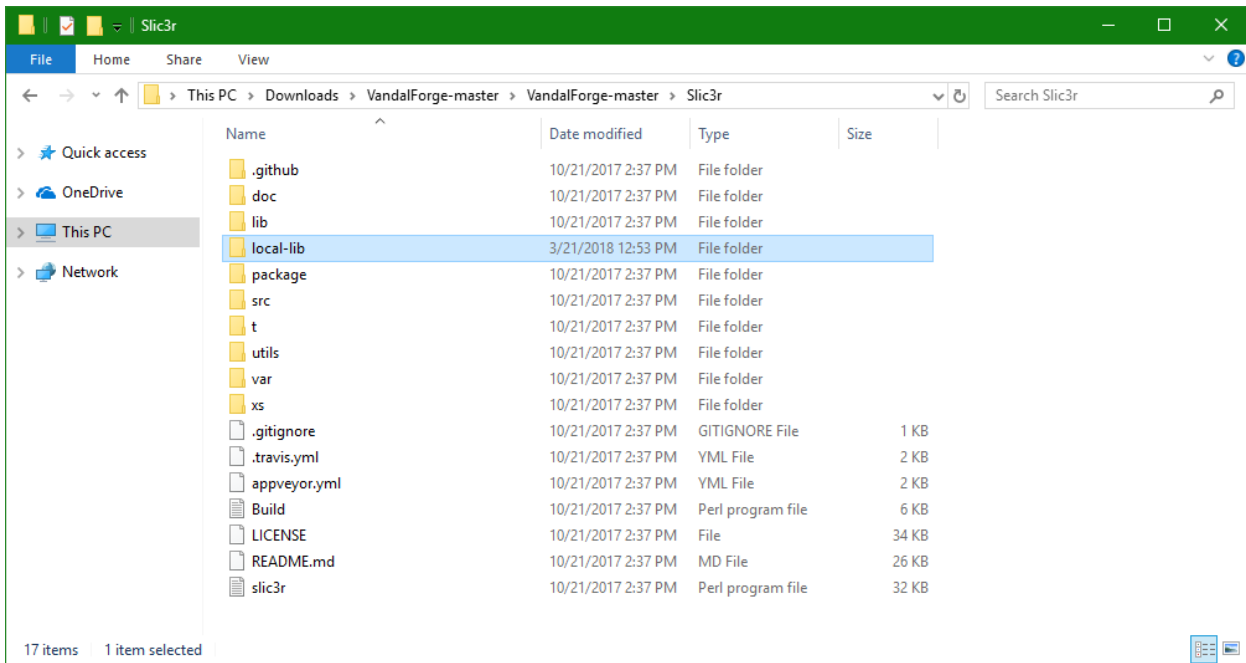


Fig. 3.2) ... and place local-lib here.

## 4) Building Slic3r

- After getting the sources, you will have a slic3r folder created, inside this folder extract the content of prebuilt dependency archive, and then continue with next step. Don't try to build the GUI without this dependencies, it won't work
- First, verify your Perl command environment is working. Start the prompt as stated above, then type `perl -v`. You should get some meaningful message. Check your perl installation if not.
- Now `cd` to the folder where Slic3r source code is cloned, and proceed with the build process to verify the dependence installed:

```
cd Slic3r
perl Build.pl
perl Build.pl --gui
```

If you want to have a simple exe file to run Slic3r from (instead of invoking perl), open a Powershell window and `cd` to package/win and run `& .\compile_wrapper.ps1 524`. This will put 3 exe files in the same directory; either copy these files to the root Slic3r directory and/or run `& .\package_win32.ps1` to make a zip file (similar to how our build server works). The former approach is fine if you aren't going to distribute Slic3r to machines that don't have slic3r-perl installed.

At this point, Slic3r is almost at a point where it can be modified, compiled, and run. However, before it reaches this point, there is two more things to do. Recall back to the section regarding the Perl Module dependencies; this is where these come into play. Before building the gui version, first extract from the zip file the folder titled "local-lib". Take this folder and place it within the Slic3r directory. Next, run this line of code:

```
cpan App::cpanminus
```

This is just a required module for perl, and will quickly be installed

Now Slic3r is ready to be compiled. Following the lines provided in the base tutorial...

```
cd Slic3r
```

This will move you to the Slic3r directory, which is required for building the program.

```
perl Build.pl
```

This will run the build sequence for Slic3r without a GUI. While this does not build the GUI, it is absolutely needed for the GUI portion to run, so do not skip this step. This will take quite some time, depending on the machine compiling, so be aware.

```
perl Build.pl --gui
```

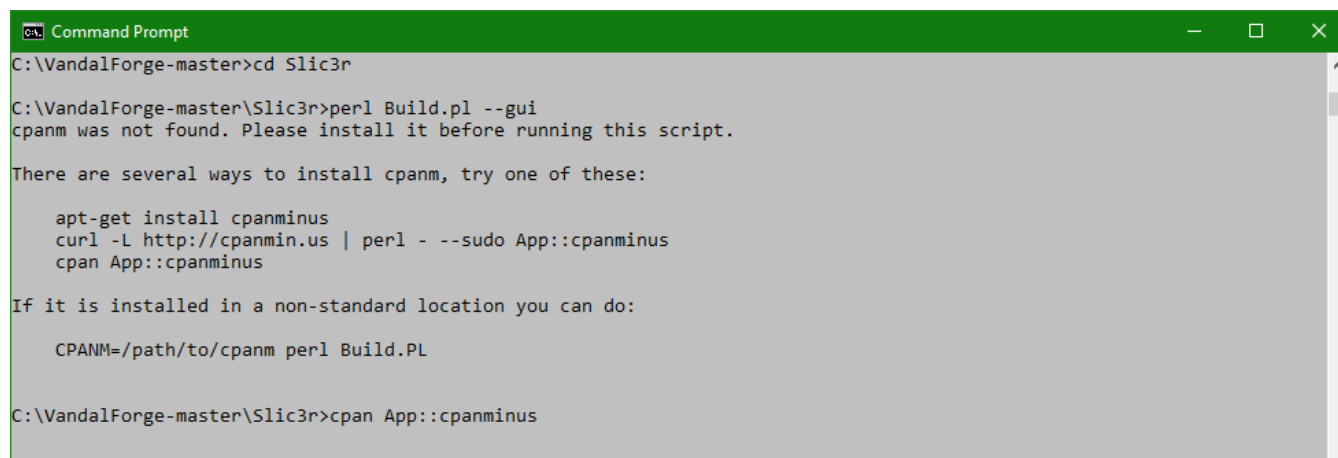
This will run the build sequence for Slic3r with a GUI. While this is what you want, there are a few special notes regarding the build process. First, be sure to allow Perl access to your network, as it is required for one of the tests. Second, it will take a very long time much like the previous boost

dependencies. Finally, it **will** say that the compilation failed, specifically the Wx components. This is SAFE to ignore, as these are included in the local-lib folder moved in earlier.

NOTE: If you receive an error regarding cpanm not being installed, follow the provided instructions and run the command:

```
cpan App::cpanminus
```

NOTE: While the compilation will take a long time, it should speed itself up on future compilations. Additionally, be sure to only compile what needs to be compiled, or in other words, if working on the GUI component, only recompile the GUI side of Slic3r.



```
Command Prompt
C:\VandalForge-master>cd Slic3r
C:\VandalForge-master\Slic3r>perl Build.pl --gui
cpanm was not found. Please install it before running this script.

There are several ways to install cpanm, try one of these:

    apt-get install cpanminus
    curl -L http://cpanmin.us | perl - --sudo App::cpanminus
    cpan App::cpanminus

If it is installed in a non-standard location you can do:

    CPANM=/path/to/cpanm perl Build.PL

C:\VandalForge-master\Slic3r>cpan App::cpanminus
```

Fig. 4.1) Potential error when attempting to compile Slic3r. Fix this by following the instructions and entering in: `cpan App::cpanminus`



```
Command Prompt
Writing MYMETA.yml and MYMETA.json
MIYAGAWA/App-cpanminus-1.7043.tar.gz
C:\Strawberry\perl\bin\perl.exe Makefile.PL -- OK
Running make for M/MI/MIYAGAWA/App-cpanminus-1.7043.tar.gz
cp lib/App/cpanminus.pm blib\lib\App\cpanminus.pm
cp lib/App/cpanminus/fatscript.pm blib\lib\App\cpanminus\fatscript.pm
"C:\Strawberry\perl\bin\perl.exe" -MExtUtils::Command -e cp -- bin\cpanm blib\script\cpanm
pl2bat.bat blib\script\cpanm
MIYAGAWA/App-cpanminus-1.7043.tar.gz
C:\STRAWB~1\c\bin\dmake.exe -- OK
Running make test
"C:\Strawberry\perl\bin\perl.exe" "-MExtUtils::Command::MM" "-MTest::Harness" "-e" "undef *Test::Harness::Switches; test
_harness(0, 'blib\lib', 'blib\arch')" t/*.t
t/happy_cpantesters.t .. 1/1 # App::cpanminus/1.7043
t/happy_cpantesters.t .. ok
All tests successful.
Files=1, Tests=1, 1 wallclock secs ( 0.06 usr + 0.03 sys = 0.09 CPU)
Result: PASS
MIYAGAWA/App-cpanminus-1.7043.tar.gz
C:\STRAWB~1\c\bin\dmake.exe test -- OK
Running make install
Installing C:\STRAWB~1\perl\site\lib\App\cpanminus.pm
Installing C:\STRAWB~1\perl\site\lib\App\cpanminus\fatscript.pm
Installing C:\STRAWB~1\perl\site\bin\cpanm
Installing C:\STRAWB~1\perl\site\bin\cpanm.bat
Appending installation info to C:\STRAWB~1\perl\lib\perllocal.pod
MIYAGAWA/App-cpanminus-1.7043.tar.gz
C:\STRAWB~1\c\bin\dmake.exe install UNINST=1 -- OK
C:\VandalForge-master\Slic3r>
```

Fig. 4.2) After installing cpanm

```
Command Prompt
C:\Users\James\Downloads\VandalForge-master\VandalForge-master\Slic3r>perl Build.PL
```

Fig. 4.3) Begin compiling non-GUI components of Slic3r

```
Command Prompt - perl slic3r.pl
t\combineinfill.t ..... ok
t\config.t ..... ok
t\cooling.t ..... ok
t\custom_gcode.t ..... ok
t\fill.t ..... ok
t\flow.t ..... ok
t\gaps.t ..... ok
t\gcode.t ..... ok
t\geometry.t ..... ok
t\layers.t ..... ok
t\loops.t ..... ok
t\multi.t ..... ok
t\perimeters.t ..... ok
t\polyclip.t ..... ok
t\pressure.t ..... ok
t\print.t ..... ok
t\retraction.t ..... ok
t\shells.t ..... ok
t\skirt_brim.t ..... ok
t\slice.t ..... skipped: temporarily disabled
t\speed.t ..... ok
t\support.t ..... ok
t\svg.t ..... ok
t\thin.t ..... ok
t\threads.t ..... ok
t\vibrationlimit.t ..... ok
All tests successful.
Files=35, Tests=589, 148 wallclock secs ( 0.30 usr + 0.19 sys = 0.48 CPU)
Result: PASS
If you also want to use the GUI you can now run `perl Build.PL --gui` to install the required modules.
```

Fig. 4.4) End of the non-GUI component of Slic3r compilation.

```
Command Prompt
C:\VandalForge-master>cd Slic3r
C:\VandalForge-master\Slic3r>perl Build.pl --gui
```

Fig. 4.5) Begin building the GUI component of Slic3r

```
Command Prompt - perl Build.pl --gui
C:\VandalForge-master\Slic3r>perl Build.pl --gui
--> Working on local::lib
Fetching http://www.cpan.org/authors/id/H/HA/HAARG/local-lib-2.000024.tar.gz ... OK
Configuring local-lib-2.000024 ... OK
Building and testing local-lib-2.000024 ... OK
Successfully installed local-lib-2.000024
1 distribution installed
App::cpanminus is up to date. (1.7043)
--> Working on Class::Accessor
Fetching http://www.cpan.org/authors/id/K/KA/KASEI/Class-Accessor-0.51.tar.gz ... OK
Configuring Class-Accessor-0.51 ... OK
Building and testing Class-Accessor-0.51 ... OK
Successfully installed Class-Accessor-0.51 (upgraded from 0.34)
1 distribution installed
--> Working on Growl::GNTP
Fetching http://www.cpan.org/authors/id/M/MA/MATTN/Growl-GNTP-0.21.tar.gz ... OK
Configuring Growl-GNTP-0.21 ... OK
==> Found dependencies: Crypt::CBC, Data::UUID
--> Working on Crypt::CBC
Fetching http://www.cpan.org/authors/id/L/LD/LDS/Crypt-CBC-2.33.tar.gz ... OK
Configuring Crypt-CBC-2.33 ... OK
Building and testing Crypt-CBC-2.33 ... OK
Successfully installed Crypt-CBC-2.33
--> Working on Data::UUID
Fetching http://www.cpan.org/authors/id/R/RJ/RJBS/Data-UUID-1.221.tar.gz ... OK
Configuring Data-UUID-1.221 ... OK
Building and testing Data-UUID-1.221 ...
```

Fig. 4.6) Expected output after starting GUI compile

```
Command Prompt
Building and testing IO-Socket-SSL-2.056 ... OK
Successfully installed IO-Socket-SSL-2.056
Building and testing LWP-Protocol-https-6.07 ... OK
Successfully installed LWP-Protocol-https-6.07
Building and testing Alien-wxWidgets-0.69 ... FAIL
! Installing Alien::wxWidgets failed. See C:\Users\James\.cpanm\work\1521611750.10200\build.log for details. Retry with
--force to force install it.
! Installing the dependencies failed: Module 'Alien::wxWidgets' is not installed
! Bailing out the installation for Wx-0.9932.
2 distributions installed
--> Working on Wx::GLCanvas
Fetching http://www.cpan.org/authors/id/M/MB/MBARBON/Wx-GLCanvas-0.09.tar.gz ... OK
==> Found dependencies: Wx::build::MakeMaker
--> Working on Wx::build::MakeMaker
Fetching http://www.cpan.org/authors/id/M/MD/MDOOTSON/Wx-0.9932.tar.gz ... OK
==> Found dependencies: Alien::wxWidgets
--> Working on Alien::wxWidgets
Fetching http://www.cpan.org/authors/id/M/MD/MDOOTSON/Alien-wxWidgets-0.69.tar.gz ... OK
Configuring Alien-wxWidgets-0.69 ... OK
Building and testing Alien-wxWidgets-0.69 ... FAIL
! Installing Alien::wxWidgets failed. See C:\Users\James\.cpanm\work\1521612483.8576\build.log for details. Retry with
--force to force install it.
! Installing the dependencies failed: Module 'Alien::wxWidgets' is not installed
! Bailing out the installation for Wx-0.9932.
! Installing the dependencies failed: Module 'Wx::build::MakeMaker' is not installed
! Bailing out the installation for Wx-GLCanvas-0.09.
Don't worry, this module is optional.
The following prerequisites failed to install: Wx
C:\VandalForge-master\Slic3r>
```

Fig. 4.7) End of GUI compile, note the failure to install Wx. This is normal, and should be expected

## 5) Starting Slic3r

If you got the dependencies installed, now you could fire up the Slic3r with still in the command prompt by:

```
perl slic3r.pl
```

To start slic3r with a single click You should create a shortcut with a target that shows to the full path of the "wperl.exe" file and then the full path to Your "slicer.pl" file. For example,

```
C:\dev\CitrusPerl\bin\wperl.exe
```

```
C:\Users\YourUsername\Documents\GitHub\Slic3r\slic3r.pl
```

**Happy Slicing!**

After Slic3r is done building, it is finally able to be run. As mentioned in the base tutorial, all that is needed now to run Slic3r is the line:

```
perl slic3r.pl
```

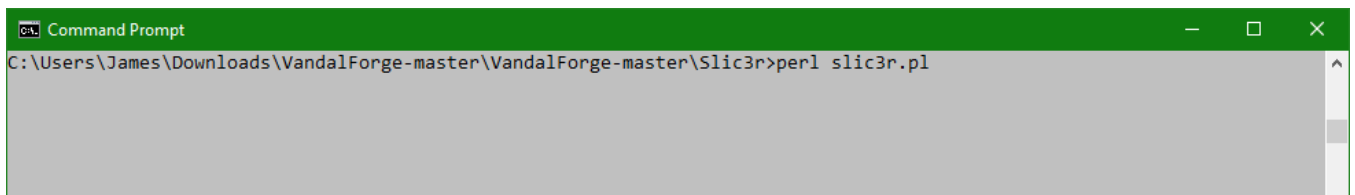


Fig. 5.1) Run the line of code...

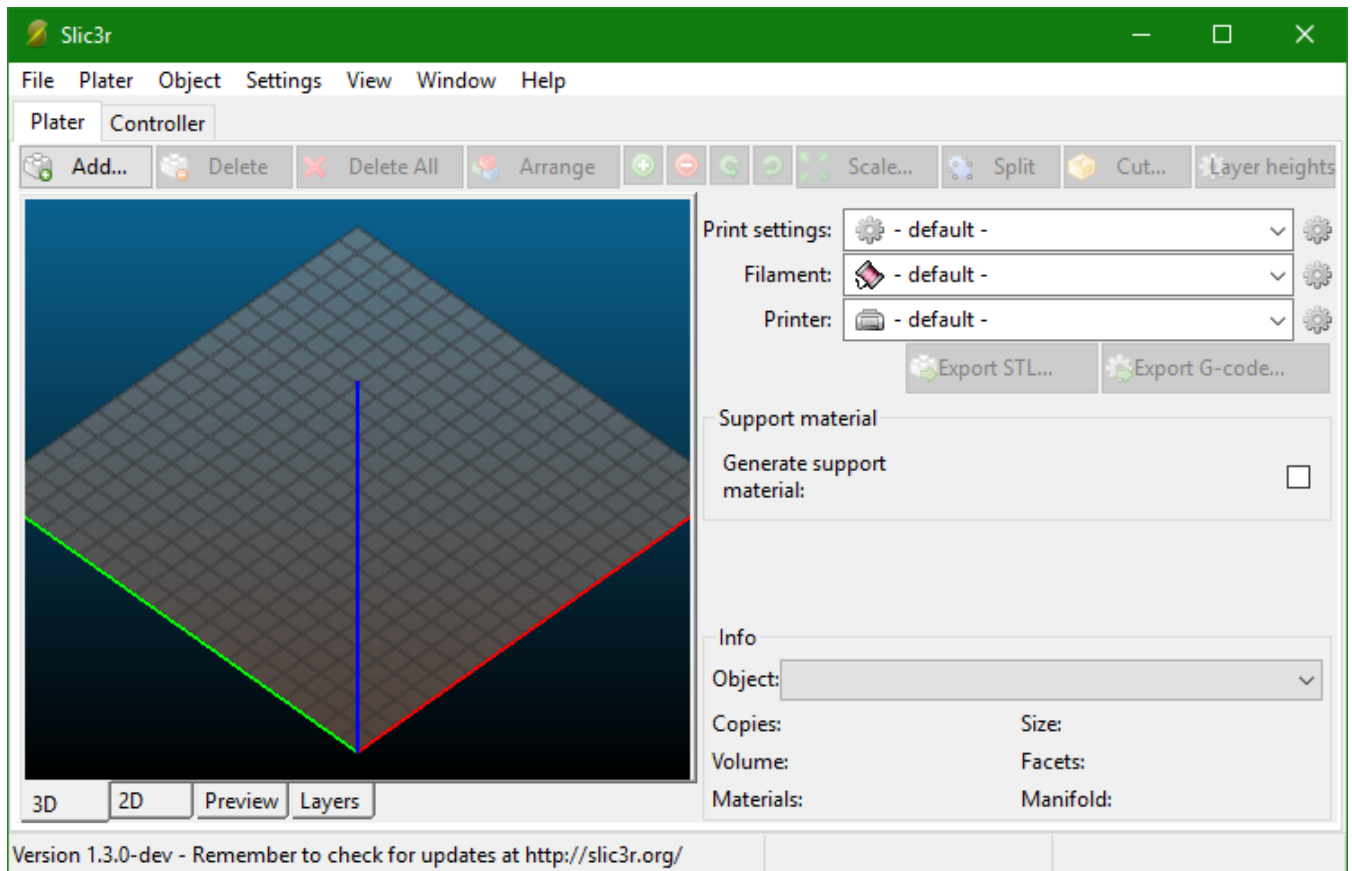


Fig. 5.2) ... and Slic3r should start up!

This concludes the specialized tutorial for the building of the VandalForge variant of Slic3r. Questions can be directed to James Young, contact info below.

James Young  
E-mail: [youn5393@vandals.uidaho.edu](mailto:youn5393@vandals.uidaho.edu)  
Cellphone: 208-881-6318

## B: Slic3r – Adding GCode Flavors

---

Adding a GCode flavor to the existing options requires changing these 6 files. You should see something similar to the subtractions of code highlighted in red, change the code to look like the green highlighted additions.

### README.md

```
140 - --gcode-flavor The type of G-code to generate
    (reprap/teacup/repetier/makerware/sailfish/mach3/machinekit/smoothie/no-extrusion,
140 + --gcode-flavor The type of G-code to generate
    (reprap/teacup/repetier/makerware/sailfish/mach3/machinekit/smoothie/no-extrusion/tigwelder/migwelder,
```

### slic3r.pl

```
363 - --gcode-flavor The type of G-code to generate (reprap/teacup/repetier/makerware/sailfish/mach3/machinekit/smoothie/no-extrusion,
363 + --gcode-flavor The type of G-code to generate (reprap/teacup/repetier/makerware/sailfish/mach3/machinekit/smoothie/no-extrusion/tigwelder/migwelder,
```

### utils/zsh/functions/\_slic3r

```
25 - '--gcode-flavor[specify the type of G-code to generate]:G-code flavor:(reprap teacup repetier makerware sailfish mach3 machinekit no-extrusion)' \
25 + '--gcode-flavor[specify the type of G-code to generate]:G-code flavor:(reprap teacup repetier makerware sailfish mach3 machinekit no-extrusion tigwelder migwelder)' \
```

### xs/src/lib slic3r/GCodeWriter.cpp

If you are adding a GCode flavor that will reuse a portion of code then simply add in the flavor in the conditional statement.

```
218 - if (FLAVOR_IS(gcfRepetier)) {
255 + if (FLAVOR_IS(gcfRepetier) || (FLAVOR_IS(gcfRepRap))) {
```

If you want to add in new output for your GCode then create a new conditional statement with your custom code.

```
136 + if(FLAVOR_IS(gcfTigWelder) || FLAVOR_IS(gcfMigWelder)){
137 +     gcode << "G0 X75 Y100 Z20\n";
138 +     gcode << "M84\n";
139 + }
140 +
```

### xs/src/lib slic3r/PrintConfig.cpp

Define your values and labels to be used with your new flavor ensure they match throughout the rest of your files.

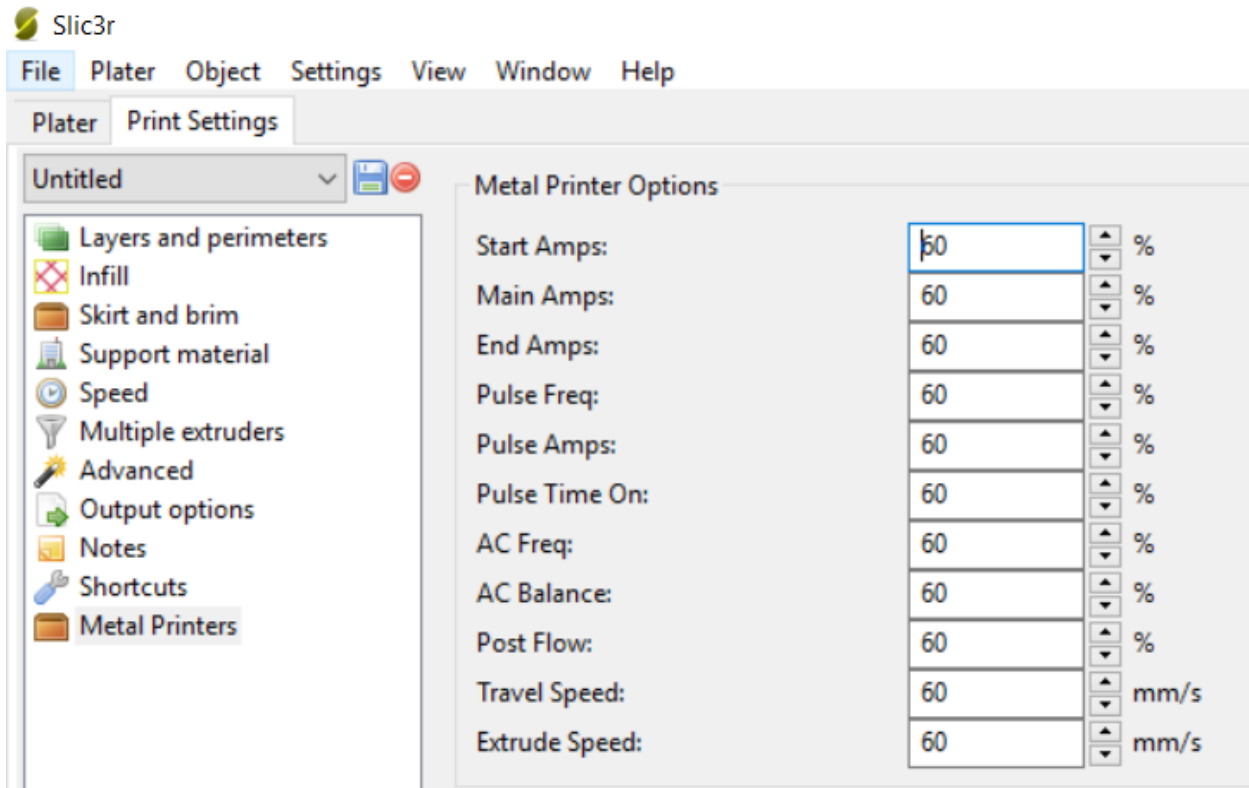
```
653 + def->enum_labels.push_back("TigWelder");
654 + def->enum_labels.push_back("MigWelder");
642 + def->enum_values.push_back("tigwelder");
643 + def->enum_values.push_back("migwelder");
```

## xs/src/libslc3r/PrintConfig.hpp

Add your new flavor into the GCodeFlavor user defined data type and map the key for your new flavor.

```
- gcfRepRap, gcfTeacup, gcfMakerWare, gcfSailfish, gcfMach3, gcfMachinekit, gcfNoExtrusion, gcfSmoothie, gcfRepetier,
29 + gcfRepRap, gcfTeacup, gcfMakerWare, gcfSailfish, gcfMach3, gcfMachinekit, gcfNoExtrusion, gcfSmoothie, gcfRepetier, gcfTigWelder, gcfMigWelder,
62 + keys_map["tigwelder"] = gcfTigWelder;
63 + keys_map["migwelder"] = gcfMigWelder;
```

## C: Altering GCode output for use with new User Inputs.



Altering the GCode output requires changing these 4 files. To see all the changes to these files go to the VandalForge Github repository and view commit 9ec987e “Pushing Metal Printer Changes” committed on 5/3/2018.

[Slic3r/xs/src/libslic3r/PrintConfig.hpp](https://github.com/VandalForge/slic3r/blob/master/src/libslic3r/PrintConfig.hpp)

This file defines the value types for the user inputs in the PrintConfig.cpp file.

To be able to create new user defined variables you must first declare them in this file for use in PrintConfig.cpp.

To define user defined variables for use with GCode you place your variable inside of public below the GCodeConfig class. For example to create new integer user input variables for use with the Metal Printers Potentiometers you would declare them like this.

```
316 class GCodeConfig : public virtual StaticPrintConfig
317 {
318     public:
319         ConfigOptionString        before_layer_gcode;
320         ConfigOptionString        between_objects_gcode;
321         ConfigOptionString        end_gcode;
322         ConfigOptionStrings       end_filament_gcode;
323         ConfigOptionString        extrusion_axis;
324         ConfigOptionFloats        extrusion_multiplier;
```



```

354 + ConfigOptionInt      start_amps;
355 + ConfigOptionInt      main_amps;
356 + ConfigOptionInt      end_amps;
357 + ConfigOptionInt      pulse_freq;
358 + ConfigOptionInt      pulse_amps;
359 + ConfigOptionInt      pulse_time_on;
360 + ConfigOptionInt      ac_freq;
361 + ConfigOptionInt      ac_balance;
362 + ConfigOptionInt      post_flow;
363 + ConfigOptionInt      travel_to_speed;
364 + ConfigOptionInt      extrude_to_speed;

```

You must also define these as new options for use in the PresetEditor.pm file.

```

407 + OPT_PTR(start_amps);
408 + OPT_PTR(main_amps);
409 + OPT_PTR(end_amps);
410 + OPT_PTR(pulse_freq);
411 + OPT_PTR(pulse_amps);
412 + OPT_PTR(pulse_time_on);
413 + OPT_PTR(ac_freq);
414 + OPT_PTR(ac_balance);
415 + OPT_PTR(post_flow);
416 + OPT_PTR(travel_to_speed);
417 + OPT_PTR(extrude_to_speed);

```

### Slic3r/xs/src/lib slic3r/PrintConfig.cpp

This file defines the default values and value types for the print, filament, and printer settings options.

Creating new elements and assigning their default values requires several specific fields you must create. You must define what value type the new input will have, what the Label will be to tell the user what the value is, you may optionally define a category to sort your newly defined value, create a tooltip for when the user hovers over the input, define what the text will look like right next to the input field, define the Command Line Interface option for this new value, define min and max of this value, and set the defaults. There are a few more options as well but I do not fully understand their use or implementation. This example creates a new user defined integer value that defaults to 60.

```

1689 + def = this->add("start_amps", coInt);
1690 + def->label = "Start Amps";
1691 + def->category = "Metal Printers";
1692 + def->tooltip = "TBD";
1693 + def->sidetext = "%";
1694 + def->cli = "start-amps=i";
1695 + def->min = 0;
1696 + def->max = 100;
1697 + def->default_value = new ConfigOptionInt(60);

```

Start Amps:  %

### Slic3r/xs/src/lib slic3r/GCodeWriter.cpp

This file handles the output and formatting when exporting GCode files.

If you want to apply your changes to specific GCode flavors then you will put a conditional if statement with brackets around the desired code.

```

136 + if (FLAVOR_IS(gcfTigWelder) || FLAVOR_IS(gcfMigWelder)){
137 +     gcode << "G0 X75 Y100 Z20\n";
138 +     gcode << "M84\n";
139 + }
140 +

```

If you want to apply your changes to all but specific GCode flavors then you will put a conditional if statement with brackets around the desired code. And if you remove all “gcode <<” statements for specific GCode flavors when the line return gcode.str() is called make sure that at the end you write gcode << “”; or the program will not compile.

```

357 + if (FLAVOR_IS_NOT(gcfTigWelder) && FLAVOR_IS_NOT(gcfMigWelder))
358 + {
317 359     gcode << "G1 F" << F;
318 360     COMMENT(comment);
319 361     gcode << cooling_marker;
320 362     gcode << "\n";
363 + }
364 + gcode << "";
321 365     return gcode.str();

```

If you want to use the user defined integer values you created in PresetEditor.pm then you must call it with XYZF\_NUM(this->config.start\_amps.value).

```

101 +   if(FLAVOR_IS(gcfTigWelder))
102 +   {
103 +       gcode << "I" << XYZF_NUM(this->config.start_amps.value) << " C1\n";
104 +       gcode << "I" << XYZF_NUM(this->config.main_amps.value) << " C2\n";
105 +       gcode << "I" << XYZF_NUM(this->config.end_amps.value) << " C3\n";
106 +       gcode << "I" << XYZF_NUM(this->config.pulse_freq.value) << " C4\n";
107 +       gcode << "I" << XYZF_NUM(this->config.pulse_amps.value) << " C5\n";
108 +       gcode << "I" << XYZF_NUM(this->config.pulse_time_on.value) << " C6\n";
109 +       gcode << "I" << XYZF_NUM(this->config.ac_freq.value) << " C7\n";
110 +       gcode << "I" << XYZF_NUM(this->config.ac_balance.value) << " C8\n";
111 +       gcode << "I" << XYZF_NUM(this->config.post_flow.value) << " C9\n";
112 +   }

```

If the user did not change the defaults then this value will be what you set in printconfig.cpp

[lib/Slic3r/GUI/PresetEditor.pm](#)

This files handles how the graphical user interface will display the print, filament, and printer settings.

The corresponding positions in the code are found beneath these three titles

sub title { 'Print Settings' }

sub title { 'Filament Settings' }

sub title { 'Printer Settings' }

If you want to add a new user defined GUI element to one of these settings you must ensure you correctly modify both the sub options found directly underneath the sub title line and alter or add an additional page you want the option to appear on.

For example adding in the Metal Printers options to the Print Settings tab in this file is done in the following way.

```

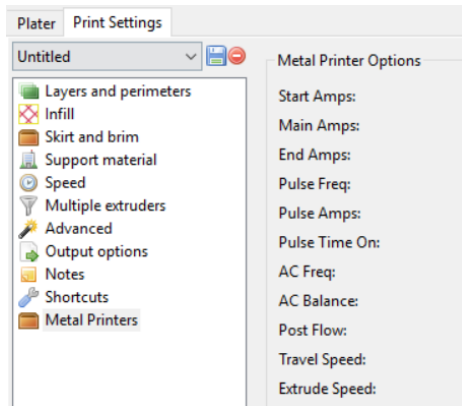
478 +   start_amps
479 +   main_amps
480 +   end_amps
481 +   pulse_freq
482 +   pulse_amps
483 +   pulse_time_on
484 +   ac_freq
485 +   ac_balance
486 +   post_flow
487 +   travel_to_speed
488 +   extrude_to_speed

```

```

792 + {
793 +     my $page = $self->add_options_page('Metal Printers', 'box.png');
794 +     {
795 +         my $optgroup = $page->new_optgroup('Metal Printer Options');
796 +         $optgroup->append_single_option_line($_, undef, width => 100)
797 +             for qw(start_amps main_amps end_amps pulse_freq pulse_amps
798 +                 pulse_time_on ac_freq ac_balance post_flow travel_to_speed extrude_to_speed
799 +             );
800 +     }
801 + }

```



## D: OctoPrint Documentation

### Helpful Links

Download and Install OctoPrint: <https://github.com/foosel/OctoPrint>

OctoPrint Documentation: <http://docs.octoprint.org/en/master/>

Config.yaml documentation: [http://docs.octoprint.org/en/master/configuration/config\\_yaml.html](http://docs.octoprint.org/en/master/configuration/config_yaml.html)

### Install

Download the latest OctoPrint from <https://github.com/foosel/OctoPrint> and follow the Installation instructions for your operating system (windows, mac, Linux, etc) (<https://github.com/foosel/OctoPrint#installation>)

Once you have OctoPrint installed and you have started the server (instructions are in the Install instructions) you can reach OctoPrint at the default addresses <https://127.0.0.1:5000> or `https:0.0.0.0:5000`

### Add Config File

To use the VandalForge config file, just replace the default OctoPrint Config file with the VandalForge one and then restart OctoPrint. The official OctoPrint config.yaml documentation has more detailed instructions for the different operating systems (depending on your OS the default config file is stored in a different place).

### To Change to University of Idaho Colors

For this you need to install a plugin. The plugin is very simple. It just injects some css classes to change the colors. The plugin is called OctoPrint-VandalPrint. All you need to do is copy the OctoPrint-VandalPrint folder into the directory where you installed OctoPrint, if you followed the documentation that folder should be called “OctoPrint”. Next, install the plugin from inside the OctoPrint-VandalPrint folder, then restart OctoPrint to see the changes.

1. Copy “OctoPrint-VandalPrint” folder into the folder containing OctoPrint
2. Change directory to inside the OctoPrint-VandalPrint folder:

```
user@linux:~/OctoPrint$ cd OctoPrint-VandalPrint
```

3. Install the VandalPrint plugin:

```
user@linux:~/OctoPrint/OctoPrint-VandalPrint$ ./venv/bin/python setup.py install
```

### More Documentation / Manual

#### *How to Install OctoPrint*

Here's install instructions directly from the GitHub:

4. Checkout OctoPrint: `git clone https://github.com/foosel/OctoPrint.git`
5. Change into the OctoPrint folder: `cd OctoPrint`
6. Create a user-owned virtual environment therein: `virtualenv venv`

7. Install OctoPrint into that virtual environment: `./venv/bin/pip install .`

You may then start the OctoPrint server via `/path/to/OctoPrint/venv/bin/octoprint serve`

#### *How to Start OctoPrint*

- Start Server (make sure using virtual environment, see the install instructions)
- Open Browser to Default address: <https://0.0.0.0:5000> or <https://127.0.0.1:5000>

#### *Adding Tool Path to GCode Viewer Notes:*

- Octoprint's GcodeViewer code is in the folder:  
OctoPrint/src/octoprint/static/gcodeviewer/js/
- The Worker.js file should be the file that controls the GcodeViewer print behavior based on the GCode values that the javascript parses as the GCode is sent to the Printer.
- You should be able to change some of the default extrude commands to the custom Metal Printer commands. (G1 and G0 I think).
- You cannot do a “dry run” on OctoPrint without connecting it to a board of some kind. You'll have to test it with OctoPrint connected to a printer.

## E: Perl Dependencies

1. Each tab is a new folder/level deeper
  2. Nearly all of the files are in Perl, some are in XS, and the last few are batch files
  3. Be careful when messing with any of the dependencies, one change can cause a series of errors
  4. Many of the batch files are dependent on their respective data file
  5. Most of the dependencies have detailed explanations inside of their comments
- /local-lib
    - Bin
      - Scandeps.batch – batch file created by scandeps.pl
      - Scandeps.pl – scans file prerequisites and dependencies
      - Use-devel-checklib – deprecated script
      - Use-devel-checklib.batch – batch file created by Use-devel-checklib
      - wxperl\_overload – finds overload declarations
      - wxperl\_overload.batch – batch file created by wxperl\_overload
      - xspp – handles the preprocessor
      - xspp.batch – batch file created by xspp
      - xsubpp – compiler used to run the make files for other perl modules
      - xsubpp.batch – batch file created by xsubpp
    - Lib
      - Perl5
        - 5.24.1
          - MSWin32-x64-multi-thread
            - (empty)
        - Crypt
          - CBC.pm – Details cipher block chaining of messages used by slic3r
        - Devel
          - CheckLib.pm
            - Checks if a certain C library is available
        - Digest

- HMAC.pm – Message hashing function
- HMAC\_MD5.pm – Message hashing function using MD5
- HMAC\_SHA1.pm – Message hashing function using SHA1
- ExtUtils
  - ParseXS – Details file specifics
    - Constants.pm
    - CountLines.pm
    - Eval.pm
    - Utilities.pm
  - Typemap – Depreciated headers for varying files
    - STL
      - String.pm
      - Vector.pm
    - Basic.pm
    - Default.pm
    - ObjectMap.pm
    - STL.pm
  - Typemaps – More complex version of typemap
    - STL
      - List.pm
      - String.pm
      - Vector.pm
    - Basic.pm
    - Cmd.pm
    - Default.pm
    - InputMap.pm
    - ObjectMap.pm
    - OutputMap.pm
    - STL.pm
    - Type.pm



- XSpp – Files related to XSpp functionality
  - Exception
    - Code.pm
    - Object.pm
    - Perlcode.pm
    - Simple.pm
    - Stdmessage.pm
    - Unknown.pm
  - Node – Ignore for now
  - Plugin
    - Feature
      - Default\_xs\_typemap
  - Typemap
    - Parsed.pm
    - Reference.pm
    - Simple.pm
    - Wrapper.pm
  - Cmd.pm
  - Driver.pm
  - Exception.pm
  - Grammar.pm
  - Lexer.pm
  - Node.pm
  - Parser.pm
  - Plugin.pod
  - Typemap.pm
- CppGuess.pm – Guesses cpp files and flags
- ParseXS.pod – Extremely long file dealing with XS code
- Typemaps.pm – Change perl/XS typemap files
- XSpp.pm
- XSpp.pod – Layer over XS, similar to C and C++

- Xsubpp – Compiler that converts XS code to C code
- Getopt
  - ArgvFile.pm – Integrates script options in command line or other array
- Growl
  - GNTTP.pm – Perl implementation of GNTTP Protocol
- IO
  - AtomicFile.pm – Writes atomically updated file
  - CaptureOutput.pm – Gets the output from Perl, XS, or subprocesses
  - InnerFile.pm – Define a file inside of another file
  - Lines.pm – Used for reading or writing an array of lines
  - Scalar.pm – Used for reading or writing a scalar
  - ScalarArray.pm – Used for reading or writing an array of scalars
  - Stringy.pm – Deals with IO for objects, ex. Strings and arrays
  - Wrap.pm – Wraps file handles
  - WrapTie.pm – Wraps “tieable” objects
- Module
  - Build
    - WithXSpp.pm – Used to make wrapping C++ with XS++ easier
  - ScanDeps
    - Cache.pm – details with caching
  - ScanDeps.pm – Very long files that generates executables from scripts that contain prerequisite modules
- MSWin32-x64-multithread – Ignore for now
- Net
  - Bonjour

- Entry.pm – Used to handle Apple’s Bonjour program
  - DNS – Ignore for now, deals with the network info
  - Rendezvous
    - Entry.pm - DEPRECATED
  - Bonjour.pm – Module for DNS service discovery
  - DNS.pm – Advanced DNS Queries
  - Rendezvous.pm - DEPRECATED
- PAR – Perl Archive Toolkit
  - Dist.pm – Creates and manipulates PAR distributions
  - Environment.pod – Environment Variables for PAR
  - FAQ.pod – FAQ for PAR
  - Heavy.pm – Dynamic Inclusion of XS modules
  - SetupPrognome.pm – Setup for the environment variables
  - SetupTemp.pm – Setup for the PAR\_TEMP environment variable
  - Tutorial.pod – A tutorial on PAR
- Parse
  - Binary – Conversions for hashes
    - FixedFormat
      - Variants.pm – Converts between variant recodes and hashes
    - FixedFormat.pm – Converts between fixed-length fields and hashes
  - Binary.pm - Unpack binary data structures into object hierarchies
- Test
  - Builder
    - IO
      - Scalar.pm – Copy of the above IO scalar.pm
    - Tester

- Color.pm – Responsible for color output in debug
      - Formatter.pm – Converts events into TAP
      - Module.pm – Helps in creating test modules
      - Tester.pm – Tests the testing modules that were built
      - TodoDiag.pm – Diagnose tool to be used on TODO – Not used directly
    - Tester – Assists with testing built test modules
      - Capture.pm – Assists with tester.pm in testing built modules
      - CaptureRunner.pm – Able to adjust monitoring of test results
      - Delegate.pm
    - Use
      - Ok.pm
    - Builder.pm – The backend for building test libraries, extremely long
    - Differences.pm – Simply checks if two strings or structures are different
    - More.pm – Framework for writing test scripts
    - Simple.pm – Basic module for writing tests for CPAN modules
    - Tester.pm – Basic testing on a test module built with Test::Builder
    - Tutorial.pod – How to use the files in this subdirectory; a tutorial for writing tests
  - Test2
    - API
      - Breakage.pm – Finds what modules are broken
      - Context.pm – Sets the context for a test
      - Instance.pm – Not used directly

- Stack.pm – Manages hub instances in Test2
- Event
  - TAP
    - Version.pm – Event that gives the TAP version
  - Bail.pm – Halts testing in current file
  - Diag.pm – Diagnose event type
  - Encoding.pm – Changes the encoding for the output
  - Exception.pm – Handles exception events
  - Generic.pm – Template for custom event creation
  - Info.pm – Enables additional information formatting
  - Note.pm – Standard out note event
  - Ok.pm – Runs when a test produces a result
  - Plan.pm – Plan event
  - Skip.pm – Skips an event
  - Subtest.pm – Event for subtest types
  - Waiting.pm – Informs all processes and threads to complete
- Formatter
  - TAP.pm – Turns events into TAP
- Hub
  - Interceptor
    - Terminator.pm – Exception class for Interceptor
  - Interceptor.pm – Hub that grabs results
  - Subtest.pm – Hub used to route events for subtests
- IPC
  - Driver
    - Files.pm – Responsible for sending events between processes and threads

- Driver.pm – Class for Test2 IPC drivers
  - Tools
    - Tiny.pm – Tiny set of tools as an alternative to Test2::Suite
  - Util
    - ExternalMeta.pm – Enables 3<sup>rd</sup> party tools to attach metadata to instances
    - HashBase.pm – DON'T MODIFY! It is generated from hashbase\_inc.pl
    - Trace.pm – Debug for events
  - API.pm – Used for writing Test2 testing tools
  - Event.pm – Base class for events
  - Formatter.pm – EMPTY, namespace for formatters
  - Hub.pm – Event processor that passes events to the formatter
  - IPC.pm – Turns on IPC, useful for threading and forking support
  - Transition.pod – Notes on Test2, specifically when upgrading from Test
  - Util.pm – Tools for use by Test2 and its derivatives
- Win32
  - IPHelper.pm – Perl wrapper for Win32 IP helper functions
- XML – Ignore for now, deals with Pureperl and the XML portions of slic3r
- Enum.pm – Enumerated types in C stlye for use in Perl
- Ok.pm – Runs when a test produces a result
- PAR.pm – The Perl Archive toolkit, long description in comments
- Test2.pm – Framework for testing coordinating tools