
polychemprint3

Release 1.0

Feb 08, 2022

1	Program Overview	3
2	Installation and Setup	5
2.1	Requirements/Supported OS	5
2.2	Installing Anaconda (optional)	5
2.3	Installing PCP3 from PyPi via pip	6
2.4	Run from Source (from Github)	6
2.5	Setting up new Hardware	7
2.6	Modifying Marlin Firmware	7
3	Main Menu and Navigation	9
3.1	Reading Menus	10
3.2	Navigating Menus	10
3.3	Special Commands	10
3.4	Quit the Program	11
3.5	Ctrl + C, or Break	11
4	Configuration/About Menu	13
4.1	View Program Details and License Text	13
4.2	Change Level of Output Detail	14
4.3	Change Axes and Tool	15
5	Hardware Menu	17
5.1	GCODE Entry	17
5.2	Controlling Tools	18
5.3	Hotkeys for Jogging Axes	19
5.4	Clean and Raise Routines	20
5.5	T ? , / . Commands	21
5.6	Quit Hardware Menu	21
6	Sequence Menu	23
6.1	The Sequence Library	24
6.2	Importing GCode Sequences	32
7	Recipe Menu	37
7.1	Creating a new Recipe	37
7.2	Modifying/Saving Active Recipe	38

7.3	Browse/Load Stored Recipes	43
7.4	View Recipe Details	44
7.5	Execute Recipe Menu	44
8	PCP3 Package Overview	45
9	Sphinx Autodocumentation	47
9.1	polychemprint3.commandLineInterface	47
9.2	polychemprint3.data	49
9.3	polychemprint3.axes	49
9.4	polychemprint3.tools	53
9.5	polychemprint3.recipes	58
9.6	polychemprint3.sequence	60
9.7	polychemprint3.utility	69
	Python Module Index	73
	Index	75

PolyChemPrint3 is an [open source](#) benchtop additive manufacturing software developed at the University of Illinois by Bijal Patel and Dr. Ying Diao. For more information, please visit the [project homepage](#) and [Diao Research Group homepage](#).

This readthedocs.io page contains: A ‘user guide’ with instructions on setting up and operating the program.

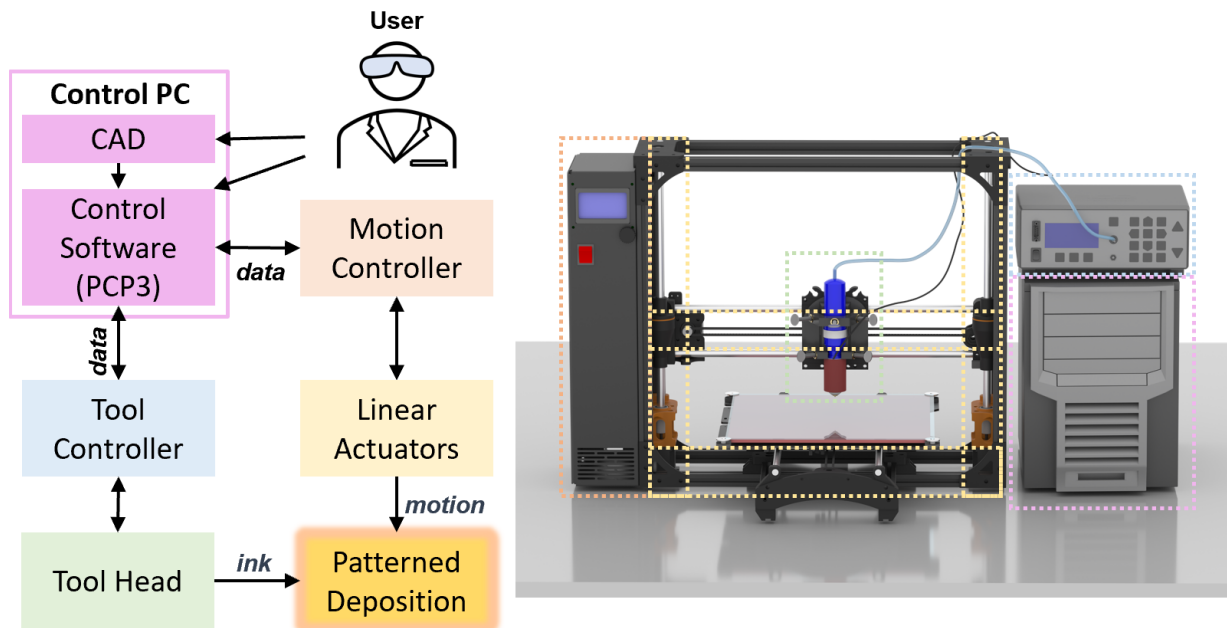
A ‘software guide’ intended as a programming aid that contains the organized python docstrings for the modules, classes, and methods of this object-oriented program.

Note: If you use this software/code, please cite the [original paper](#) listed on the project homepage! It really helps!

CHAPTER 1

Program Overview

PolyChemPrint3 (PCP3) is a command-line interface (CLI) Windows/Linux program that handles communication between the user and additive manufacturing(AM) hardware. In some ways, PCP3 offers overlapping functionality with 3D printer control software such as [pronterface](#) and 3D slicing programs such as [slic3r](#) or [Cura](#), but optimized for AM research with unconventional, non-FDM toolheads such as pneumatic (melt) extruders, LASERs, syringe pumps, etc.



At the most basic level, users can directly send commands to the motion axes and toolhead to execute GCode move sequences and simple tool on/off/ power set commands. The next level up is to use parameterized, hardcoded ‘sequences’ to execute specific 2D and 3D patterns such as meanderlines, cuboids, electrode patterns, etc. For more complex 2D/3D patterns, GCode files created by slicers such as GCodeTools in Inkscape and Cura/ Slic3r can be imported as sequences. Finally, any combination of sequences can be chained together into ‘Recipes’, offering a ‘code-free’

way to build-up complex patterns. Automatic data-logging exports print parameters to text files to optimize parameter screening.

Note: Even if you have identical hardware to the original developers, the software will need some initial setup - so pay careful attention to the “Installation and Setup” section. Best of luck!

2.1 Requirements/Supported OS

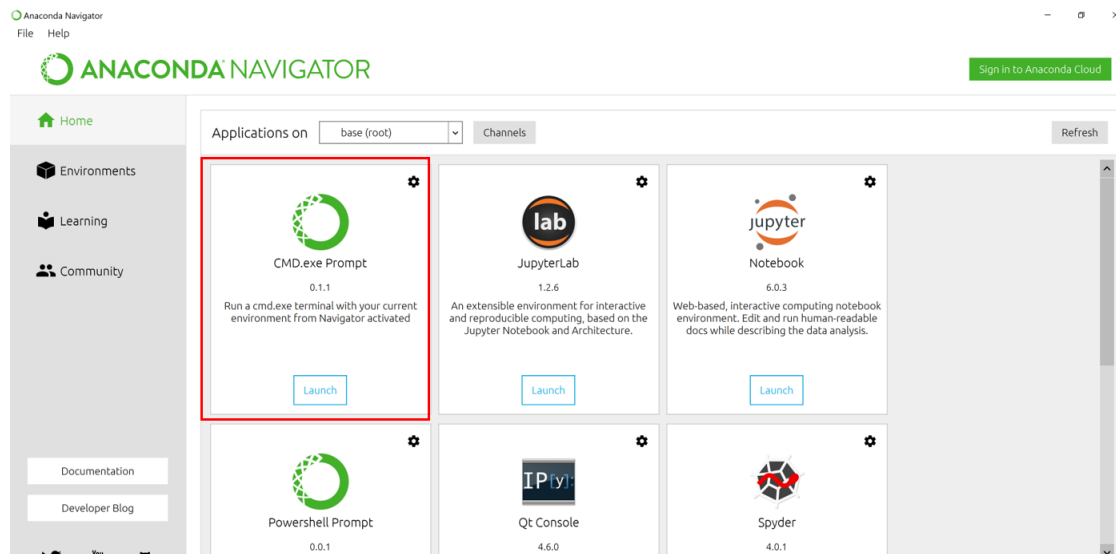
PCP3 is designed to run on Windows and Linux distributions with Python 3 on even very low spec hardware. At the end of the day, for just sending commands between hardware, there really isn't much you need in terms of specialized system specs. Just serial ports that can be connected to your desired hardware.

In our lab, we have run PCP3 on Debian 9/ Mint 19 Linux PCs and on Windows 10 using Anaconda as the python environment.

2.2 Installing Anaconda (optional)

To use polychemprint3, you need to have a python 3 environment set up. If you are on Linux, your distribution most likely has python 3 installed out of the box. If not, a simple way to set things up is to use [Anaconda](#), a free and open source python distribution commonly used in data science.

After successfully installing Anaconda, open Anaconda Navigator and launch anaconda prompt. Boxed in red below:



2.3 Installing PCP3 from PyPi via pip

After opening the appropriate terminal window (Anaconda Prompt/Terminal/Command Prompt), enter:

```
pip install --no-cache-dir --pre --upgrade polychemprint3
```

Press enter and polychemprint3 should install with any required dependencies automatically.

To run the program, just type

```
polychemprint3
```

into the terminal window and the program should launch

```
PolyChemPrint3 - Version:3.0   Revised: 2020/05/11
=====
## Main Menu
=====
(0) Configuration/About      | Software setup, options, choose Tool/Axes
(1) Hardware Control Menu   | Send commands directly to hardware
(2) Sequence Menu           | Configure/Execute predefined command sequences
(3) Recipe Menu             | Configure/Execute multi-sequence recipes
[T] Test Code               | Run test code
[q] Quit                    | Quit
[?] Repeat menu options     | Repeat menu options

[/] Repeat Last Command     |
[.] Repeat Saved Command    |
[,] Store Saved Command     | Will Prompt for command

Enter Command:>
```

2.4 Run from Source (from Github)

All source code (for PCP3 and this manual) is posted on github at <https://github.com/BijalBPatel/PolyChemPrint3> . Three branches are maintained:

- Master: The main stable release

- Beta: A test release which may have new features we are testing. This is the version we run in our lab.
- Dev: Testing release for new and semi/not working features.

If you are new to github, there are many quick tutorials online - such as [this](#).

2.5 Setting up new Hardware

PCP3 as written uses pySerial to communicate with hardware devices. To add a new tool, begin by cloning and renaming one of the existing tool.py files in the polychemprint3/tools directory. We will then go line by line and replace comment text and parameters such as device address, baudrate, etc with the values that correspond to your particular hardware. Here we highlight key parameters to change:

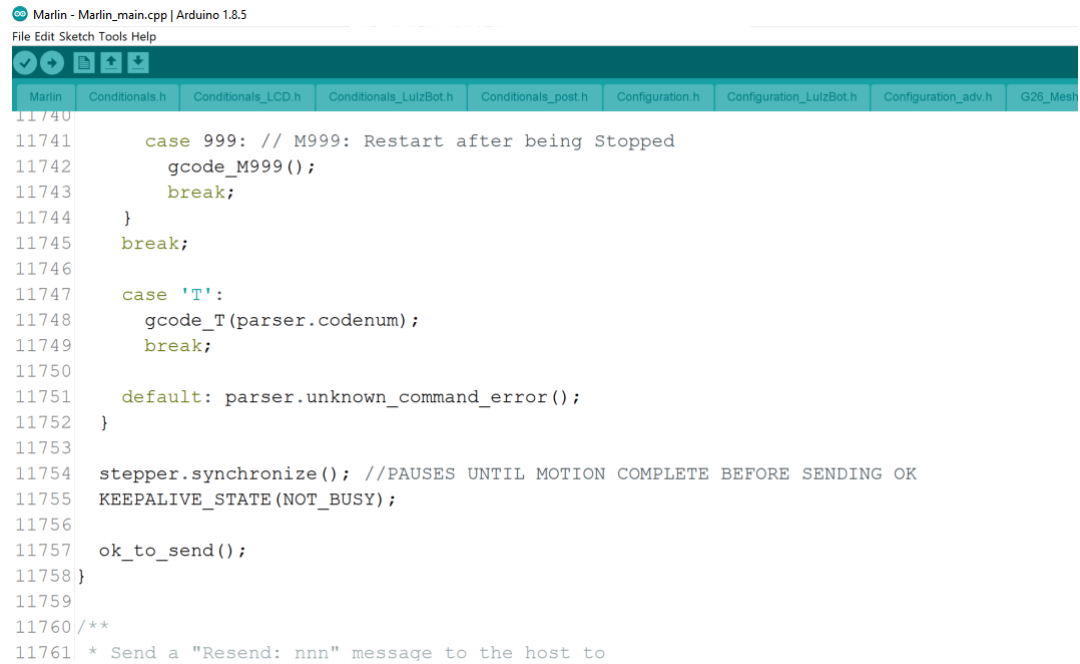
1. In the `__init__` method, set the `devAddress`, `baudRate`, `commsTimeOut`, and other parameters to reflect your hardware.
2. Next, go through each of the methods (`activate`, `deactivate`, `engage`, `disengage`, `setValue`, `startSerial`, etc), and write the necessary code to complete the communication loop with your hardware. If your device has a simple arduino based controller, these methods may be very simple (see `Laser6W.py`). If the device uses a special packet-based protocol, this can be more challenging, but see `ultimusExtruder.py` for a good example of this.
3. No matter what, make sure the methods specified in the `toolSpec.py` abstract base class are filled out in your new code file.
4. Once the `tool.py` file is complete, restart PCP3 and check that it properly is loaded [the starting load text will indicate “PASS” for both conditions.

2.6 Modifying Marlin Firmware

If you are using a consumer 3D printer for your motion axes, there is a high likelihood you will need to modify the stock Marlin firmware to work with PCP3. Our main goal is to force the command acknowledge statement “ok ...” to only be sent from the printer AFTER all motion steps are complete. If you are running on Linux, you may also need to change the firmware baudrate for compatibility. Here is how:

1. Download the Marlin firmware source files either from your printer manufacturer’s webpage, or from the main [Marlin Firmware webpage](#)
2. If you are getting firmware from the Marlin site, see if you can find the configuration files for your printer in the [MarlinFirmware Github folder](#) that corresponds to your printer.
3. Download arduinoIDE and from Tools -> Boards -> Board Manager install the RAMBo board files.
4. Open all of your Firmware files in arduino IDE by running the `Marlin.ino` file in the Marlin folder.
5. If necessary, in the `conditionals.h` file, set the baudrate to your desired value.
6. Navigate to the `Marlin_main.cpp` file and find the “`process_next_command()`” method. At the very end of this method (see image), add the following statement:

```
stepper.synchronize(); //PAUSES UNTIL MOTION COMPLETE BEFORE SENDING OK
```



```
11740
11741     case 999: // M999: Restart after being Stopped
11742         gcode_M999();
11743         break;
11744     }
11745     break;
11746
11747     case 'T':
11748         gcode_T(parser.codenum);
11749         break;
11750
11751     default: parser.unknown_command_error();
11752 }
11753
11754 stepper.synchronize(); //PAUSES UNTIL MOTION COMPLETE BEFORE SENDING OK
11755 KEEPALIVE_STATE(NOT_BUSY);
11756
11757 ok_to_send();
11758 }
11759
11760 /**
11761  * Send a "Resend: nnn" message to the host to
```

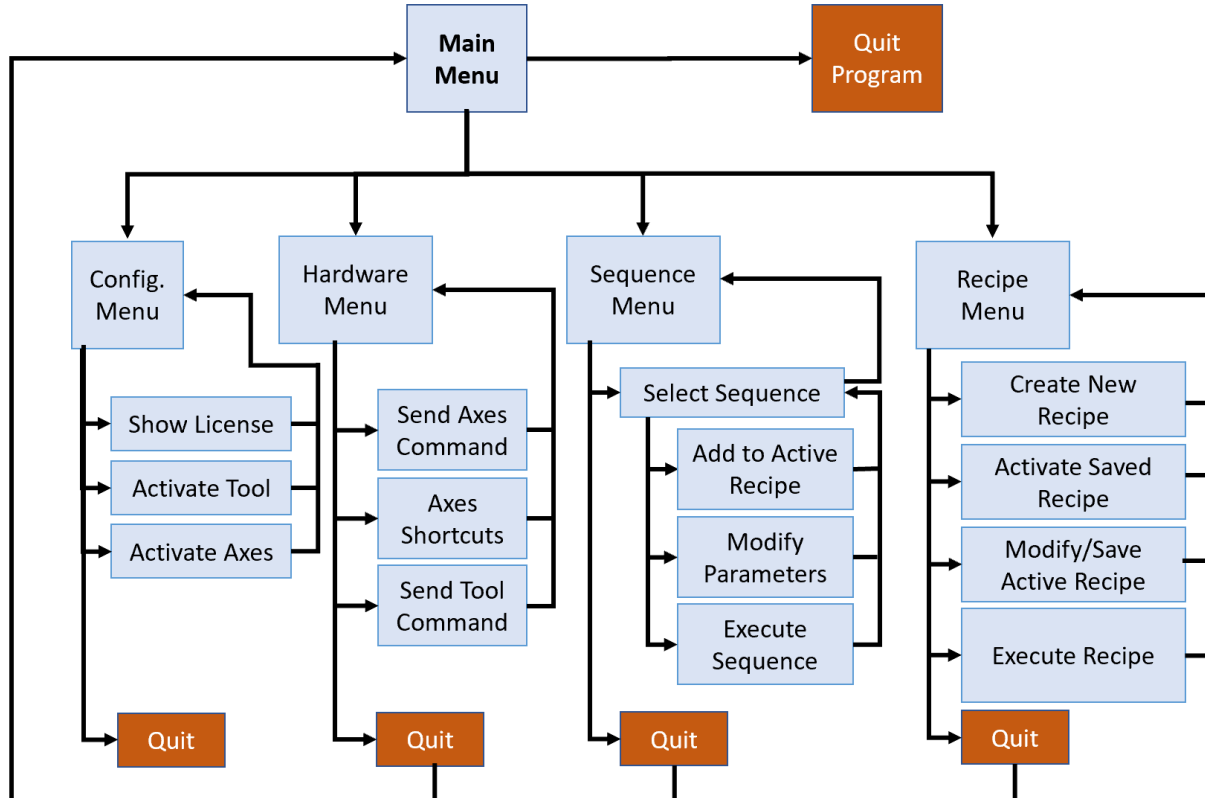
7. Compile as hex and export

8. Use a program such as cura to load your new firmware onto your printer.

Note: Be sure to save the old firmware, you will need it to go back to normal FDM 3D printing.

Main Menu and Navigation

PCP3 has a (hopefully) straightforward command-line interface (CLI) based on a series of menus and submenus. The Main Menu is the ‘root’ of the decision tree, from here you can navigate between the main functions of the program: configuration, manual control, sequences, and recipes. It should appear after the initial loading sequence and is a good place to start learning how to navigate the CLI.



3.1 Reading Menus

Here is a screenshot of the main menu in version 3.0:

```

PolyChemPrint3 - Version:3.0   Revised: 2020/05/11
=====
##      Main Menu
=====
(0) Configuration/About      | Software setup, options, choose Tool/Axes
(1) Hardware Control Menu    | Send commands directly to hardware
(2) Sequence Menu           | Configure/Execute predefined command sequences
(3) Recipe Menu             | Configure/Execute multi-sequence recipes
[T] Test Code               | Run test code
[q] Quit                   | Quit
[?] Repeat menu options     | Repeat menu options

[/] Repeat Last Command      |
[.] Repeat Saved Command     |
[,] Store Saved Command      | Will Prompt for command

Enter Command:>

```

As you can see, there are 2 columns separated by a vertical bar (“|”). On the left, is a command string (enclosed in brackets “[]” or parentheses “()”) and short name for the command. At right, there is an optional detailed description. For some commands, such as those near the end of the list (in teal) (“/”, “.”), the text on the right is initially blank and will be filled in based on the saved command in memory.

This scheme repeats throughout: in order to execute a command or change a data value, enter the command string at the extreme left (in brackets or parentheses).

At the bottom of the screen is the prompt.

3.2 Navigating Menus

```

PolyChemPrint3 - Version:3.0   Revised: 2020/05/11
=====
##      Main Menu
=====
(0) Configuration/About      | Software setup, options, choose Tool/Axes
(1) Hardware Control Menu    | Send commands directly to hardware
(2) Sequence Menu           | Configure/Execute predefined command sequences
(3) Recipe Menu             | Configure/Execute multi-sequence recipes
[T] Test Code               | Run test code
[q] Quit                   | Quit
[?] Repeat menu options     | Repeat menu options

[/] Repeat Last Command      |
[.] Repeat Saved Command     |
[,] Store Saved Command      | Will Prompt for command

Enter Command:>

```

As mentioned above, the leftmost column [boxed in red] contains the command strings. Type in **0** gives users access to Configuration/About Menu. Type in **1** gives you access to Hardware Control Menu. Type in **2** to go to Sequence Menu. Type in **3** to gain access to Recipe Menu.

3.3 Special Commands

Type in “T” to perform a test code run. This executes whatever code is in the `io_TestCode()` subroutine of the `__main__.py` file. By default the subroutine is left empty, but I found it handy when developing the code, so I left it in as an option to aid user testing.

Type “?” to repeat the current menu. Essentially it is just a handy refresh in case the screen gets cluttered with text. In this case, a new Main Menu will appear.

The “/” command works repeats the last command entered. This is mostly useful just when doing manual hardware control. In some menus this leads to meaningless behavior and so is disabled. For example, if the previous command typed in the terminal is “G0 X5”, typing “/” will send “G0 X5” to the printer again.

The “.” command will repeat the command that is save in Stored Saved Command.

The “,” command lets users save a command, so user can just type . to perform a long command instead of typing the long command every time. Again, a handy shortcut mostly for manual hardware control.

3.4 Quit the Program

In the main menu, “q” lets users quit the program. This triggers a confirmation prompt as follows:

```

##### Main Menu #####
(0) Configuration/About      | Software setup, options, choose Tool/Axes
(1) Hardware Control Menu    | Send commands directly to hardware
(2) Sequence Menu           | Configure/Execute predefined command sequences
(3) Recipe Menu             | Configure/Execute multi-sequence recipes
[T] Test Code               | Run test code
[q] Quit                   |
[?] Repeat menu options     |

[/] Repeat Last Command      | N
[.] Repeat Saved Command     |
[,] Store Saved Command      | Will Prompt for command

Enter Command:> q
Really quit (Y,q)? or internal reset? (N): >

```

Now, type “Y” or “q” will let users exist out of the program and returning to the terminal. (Note: In general, input to polychemprint3 is not case sensitive, unless being sent directly to hardware) Typing **N** will reset the program, which means restarting the program. Any work (recipes) that are not saved will be lost.

3.5 Ctrl + C, or Break

As of version 3.0, users can use the key combination “Ctrl + C” to shortcircuit ongoing processes and return to the last menu. This is invaluable for cases where you start a print sequence (with thousands of lines) and realize that you need to cancel out early. Note: **THIS WILL NOT** automatically turn off toolheads or suspend the current GCode command that is sent, it just prevents the PC from sending further commands. So be vigilant! The hardware off switch is still the last best fail-safe.

Configuration/About Menu

The configuration/about menu **Configure/About Menu** allows users to view program and license information about Polychemprint3. This menu also includes options to change the level of output details, and switch axes and tools the program is controlling. Here is what Configuration/About Menu looks like:

```
###      Configuration/About Menu
-----
(0) Info and License      | View Program Details and License Text
(1) Verbose               | Toggles level of output details
(2) Change Axes           | Current Axes: nullAxes
(3) Change Tool           | Current Tool: nullTool
[q]                       | Quit
[?]                       | List Commands

[/] Repeat Last Command   | T1
[.] Repeat Saved Command  | T
[,] Store Saved Command   | Will Prompt for command

Enter Command:>
```

4.1 View Program Details and License Text

By typing command **0** in **Configuration/About Menu**, program details and license text will be displayed on the terminal like the following:

```

Enter Command:> 0

PolyChemPrint3 v3.0
2020/05/11
By Bijal Patel bbpatel2@illinois.edu

Provided under the University of Illinois/NCSA
Open Source License

Copyright (c) <2019> <Bijal Patel - University of Illinois>. All rights reserved.

Developed by: <Diao Lab>
              <University of Illinois>
              <http://diao.scs.illinois.edu/Diao_Lab/Home.html>

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal with
the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to
do so, subject to the following conditions:
* Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimers.
* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimers in the documentation
  and/or other materials provided with the distribution.
* Neither the names of <Diao Lab>, <University of Illinois>,
  nor the names of its contributors may be used to endorse or promote products
  derived from this Software without specific prior written permission.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE
SOFTWARE

```

4.2 Change Level of Output Detail

By entering command **1** in **Configure/About Menu**, the amount of details presented in the program will be altered. Currently, only two levels of output details exist: “more” level and “less” level. Whether this alternation increases or reduce amount of information displayed depends on the current level of output details. If the program is in the “more” level, type in **1** will change to “less” level and vice versa.

```

### Configuration/About Menu
(0) Info and License | View Program Details and License Text
(1) Verbose          | Toggles level of output details
(2) Change Axes      | Current Axes: nullAxes
(3) Change Tool      | Current Tool: nullTool
[q]                  | Quit
[?]                  | List Commands

[/] Repeat Last Command | 0
[.] Repeat Saved Command | T
[,] Store Saved Command | Will Prompt for command

Enter Command:> 1
Fewer Details will be displayed.

### Configuration/About Menu
(0) Info and License | View Program Details and License Text
(1) Verbose          | Toggles level of output details
(2) Change Axes      | Current Axes: nullAxes
(3) Change Tool      | Current Tool: nullTool
[q]                  | Quit
[?]                  | List Commands

[/] Repeat Last Command | 1
[.] Repeat Saved Command | T
[,] Store Saved Command | Will Prompt for command

Enter Command:> 1
More Details will be displayed.

```

4.3 Change Axes and Tool

Command **2** and **3** in **Configure/About Menu** change the axes and tool that polychemprint3 is controlling. Axes represents the hardware in charge of movements, for example if polychemprint 3 is controlling a 3D printer, axes should be set to that 3D printer since it controls movement in x, y, and z directions. In order to change the axes, type **2** in the command line and terminal will display all the axes that is loaded into polychemprint3 and ask for user input to select the device need to be activate like the following(boxed in yellow):

```

### Configuration/About Menu

(0) Info and License      | View Program Details and License Text
(1) Verbose              | Toggles level of output details
(2) Change Axes          | Current Axes: nullAxes
(3) Change Tool          | Current Tool: nullTool
[q]                      | Quit
[?]                      | List Commands

[/] Repeat Last Command  | T1
[.] Repeat Saved Command | T
[,] Store Saved Command  | Will Prompt for command

Enter Command:> 2
Currently loaded Axes:
A0 lulzbotTaz6_BP
A1 nullAxes
Enter device to make active>

```

Right now, the program is loaded with two axes: lulzbotTaz6_BP and nullAxes. To select the desired axes, simply type in the corresponding commands (boxed in yellow) on the left of axes names, in this case **A0** or **A1**. Tools are hardware that does not control movement. For example, in laser cutting, the amount of energy emitted need to be controlled, so laser will be tool in this case. The change tool procedure works the same way as change axes: type **3** in command and select the desired tool using commands show on the left of tool name (boxed in yellow).

```

### Configuration/About Menu

(0) Info and License      | View Program Details and License Text
(1) Verbose              | Toggles level of output details
(2) Change Axes          | Current Axes: nullAxes
(3) Change Tool          | Current Tool: nullTool
[q]                      | Quit
[?]                      | List Commands

[/] Repeat Last Command  | 0
[.] Repeat Saved Command | 
[,] Store Saved Command  | Will Prompt for command

Enter Command:> 3
Currently loaded Tools:
(T0) laser6W
(T1) nulltool
(T2) ultimuxExtruder
Enter device to make active>

```


CHAPTER 5

Hardware Menu

This menu allows for the most basic communication to the hardware: line by line command entry and execution, along with some hotkeys for jogging the motion axes.

```
##      Hardware Menu
-----
From this menu you can directly send commands to the hardware. Be careful! There is limited error-checking!
Choose an execution option or directly enter a GCODE command for the axes:

T[Value]      | Sets the tool value
Toff           | Disengage Tool Dispense
Ton           | Engage Tool Dispense
(0) Clean Routine | Lift up 20 mm, lower on cmd
(1) Lift Tool   | Lift up 20 mm
a,d;r,f;s,w;x,z | Jog -+ 1mm (X; Y; Z; Z-0.1,-.01)
[q]           | Quit
[?]           | List Commands

[/] Repeat Last Command | T 100
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command
```

5.1 GCODE Entry

Direct GCODE commands are accepted as input in Hardware Menu. For example, if the axes need to be moved in the positive x direction for 20 mm, Gcode command “G0 X20” can be typed in to perform such task, as shown below.

```

##      Hardware Menu
-----
From this menu you can directly send commands to the hardware. Be careful! There is limited error-checking!
Choose an execution option or directly enter a GCODE command for the axes:

T[Value]          | Sets the tool value
Toff              | Disengage Tool Dispense
Ton              | Engage Tool Dispense
(0) Clean Routine | Lift up 20 mm, lower on cmd
(1) Lift Tool     | Lift up 20 mm
a,d;r,f;s,w;x,z  | Jog -+ 1mm (X; Y; Z; Z-0.1,-.01)
[q]              | Quit
[?]              | List Commands

[/] Repeat Last Command | T 100
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:> Go x20
Received: go x20
Null Axes move Command: 'GO X20\n'

```

5.2 Controlling Tools

Set Tool Value Type in **T[value]** under **Hardware Menu** allows users to directly set tool values. For example, to set tool value to 100, users can type in **T100** in terminal and program will give an output indicating value is set like the following:

```

##      Hardware Menu
-----
From this menu you can directly send commands to the hardware. Be careful! There is limited error-checking!
Choose an execution option or directly enter a GCODE command for the axes:

T[Value]          | Sets the tool value
Toff              | Disengage Tool Dispense
Ton              | Engage Tool Dispense
(0) Clean Routine | Lift up 20 mm, lower on cmd
(1) Lift Tool     | Lift up 20 mm
a,d;r,f;s,w;x,z  | Jog -+ 1mm (X; Y; Z; Z-0.1,-.01)
[q]              | Quit
[?]              | List Commands

[/] Repeat Last Command | T100
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:> T100
Null Tool Says: Value set: 100

```

Turning Tool On and Off Type in **Toff** or **Ton** to engage tool dispense or disengage tool dispense. It works just like a switch that turns the tool from off to on or on to off.

```

##      Hardware Menu
-----
From this menu you can directly send commands to the hardware. Be careful! There is limited error-checking!
Choose an execution option or directly enter a GCODE command for the axes:

T[Value]          | Sets the tool value
Toff              | Disengage Tool Dispense
Ton              | Engage Tool Dispense
(0) Clean Routine | Lift up 20 mm, lower on cmd
(1) Lift Tool     | Lift up 20 mm
a,d;r,f;s,w;x,z  | Jog +- 1mm (X; Y; Z; Z-0.1,-.01)
[q]              | Quit
[?]              | List Commands

[/] Repeat Last Command | Ton
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:> Toff
Disengaging Tool
Null Tool Says: Dispense Off
Dispense Off

```

But if the current state of tool is off and a Toff command is executed, program will give a warning message saying that “**Dispense already off**”. If the current state is on, Ton command is sent, program will also give error message saying “**Dispense already on**” These are handy for troubleshooting sequences/recipes, but are otherwise just for your information.

```

##      Hardware Menu
-----
From this menu you can directly send commands to the hardware. Be careful! There is limited error-checking!
Choose an execution option or directly enter a GCODE command for the axes:

T[Value]          | Sets the tool value
Toff              | Disengage Tool Dispense
Ton              | Engage Tool Dispense
(0) Clean Routine | Lift up 20 mm, lower on cmd
(1) Lift Tool     | Lift up 20 mm
a,d;r,f;s,w;x,z  | Jog +- 1mm (X; Y; Z; Z-0.1,-.01)
[q]              | Quit
[?]              | List Commands

[/] Repeat Last Command | Toff
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:> Toff
Disengaging Tool
Null Tool Error: Dispense already Off
Error: Dispense already off

```

5.3 Hotkeys for Jogging Axes

For convenience, the following commands jog axes for small distance movement in all direction. (Note that these are not case sensitive)

Command	Direction	Distance
a	-x	1mm
d	x	1mm
r	-y	1mm
f	y	1mm
s	-z	1mm
w	z	1mm
x	-z	0.1mm
z	-z	0.01mm

```
## Hardware Menu
-----
From this menu you can directly send commands to the hardware. Be careful! There is limited error-checking!
Choose an execution option or directly enter a GCODE command for the axes:

T[Value]          | Sets the tool value
Toff              | Disengage Tool Dispense
Ton              | Engage Tool Dispense
(0) Clean Routine | Lift up 20 mm, lower on cmd
(1) Lift Tool     | Lift up 20 mm
a,d;r,f;s,w;x,z  | Jog +- 1mm (X; Y; Z; Z-0.1,-.01)
[q]              | Quit
[?]              | List Commands

[/] Repeat Last Command | ton
[.] Repeat Saved Command
[,] Store Saved Command | Will Prompt for command

Enter Command:> a
Null Axes move Command: 'G0 X-1\n'
```

5.4 Clean and Raise Routines

Commands **1** and **0** provide convenient ways to lift the toolhead 20 mm. If command **0** is chosen, the terminal will prompt on whether to lower 20mm or not. Type in **Y**, axes will be lowered for 20 mm in Z direction. Type in **N**, axes will stay still.

```
## Hardware Menu
-----
From this menu you can directly send commands to the hardware. Be careful! There is limited error-checking!
Choose an execution option or directly enter a GCODE command for the axes:

T[Value]          | Sets the tool value
Toff              | Disengage Tool Dispense
Ton              | Engage Tool Dispense
(0) Clean Routine | Lift up 20 mm, lower on cmd
(1) Lift Tool     | Lift up 20 mm
a,d;r,f;s,w;x,z  | Jog +- 1mm (X; Y; Z; Z-0.1,-.01)
[q]              | Quit
[?]              | List Commands

[/] Repeat Last Command | N
[.] Repeat Saved Command
[,] Store Saved Command | Will Prompt for command

Enter Command:> 0
Raising Tool by 20 mm...
Null Axes move Command: 'G1 F2000 Z20\n'
Lower 20 mm?(Y/N):> Y
Null Axes move Command: 'G1 F2000 Z-15\n'
Null Axes move Command: 'G1 F100 Z-9\n'
Null Axes move Command: 'G1 F100 Z-1\n'
```


5.5 T ? , / . Commands

Functions of T ? , / and . have been described in user guide section **3.3**. Please see it for more information.

5.6 Quit Hardware Menu

To exit out of the **Hardware Menu**, type **q** in the prompt and you will be returned to the **Main Menu**.

CHAPTER 6

Sequence Menu

In this menu, users perform various operations with parameterized ‘sequence’ files that describe basic motion/tool paths.

```
##      Print Sequence Menu
-----
Sequences are pre-programmed, parameterized print routines stored as python files and loaded to RAM when the program launches.
Choose a sequence code to Edit/Execute:

(S0) basicMove      | PCP_CoreUtilities | Move Axes (relative or abs)
(S1) circle         | PCP_IDCore        | a circle
(S2) cuboid          | PCP_IDCore        | length in x,width in y, and height in z
(S3) gapline         | PCP_Electronics   | Lines in X-direction with a gap where tool raises
(S4) GCodeFile3DSlicer | PCP_CoreUtilities | Imported from GCodeFile
(S5) GCodeFileInkscape | PCP_CoreUtilities | Imported from GCodeFile
(S6) line            | PCP_IDCore        | Single Line in X/Y Direction
(S7) pause           | PCP_CoreUtilities | Pause execution
(S8) plate           | PCP_IDCore        | Meanderline Plate
(S9) pyrimad         | PCP_IDCore        | length in x,width in y, and height in z
(S10) rectangle      | PCP_IDCore        | a rectangle with length in x and width in y
(S11) setToolState   | PCP_CoreUtilities | Set Tool Value or Dispense State
(S12) triangle       | PCP_IDCore        | a triangle

[q]                  | Quit
[?]                  | List Commands

[/] Repeat Last Command | q
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>
```

Each sequence shown in the menu (labeled S#) corresponds to an individual python file that gives the ‘blueprint’ for the sequence. For example, entering “S6” opens the “Line” submenu as shown below:

```

-----
### Sequence: line
-----

From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | line                               |           | Change if modifying from default
(P2 ) Sequence Description | Single Line in X/Y Direction      |           | line.py
(P3 ) Creation Date    | 16/11/2019                        |           | dd/mm/yyyy
(P4 ) Created By      | Bijal Patel                       |           |
(P5 ) Owner           | PCP_IDCore                        |           | default: PCP_Core
(P6 ) Printing Speed   | 60                                |           |
(P7 ) Line direction   | X                                  |           | Along X or Y
(P8 ) Line Length      | 10                                 | mm        |
(P9 ) Tool ON Value    | 100                               | null      | Depends on tool loaded
(P10) Tool OFF Value   | 000                               | null      | Depends on tool loaded

[Add]                 | Add/Insert sequence as configured into active recipe
[q]                   | Quit
[GO]                  | Engage Print Sequence
[PRIME]               | Generate Print Commands
[VIEW]                | View Print Commands

[/] Repeat Last Command | S6
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:> |

```

From this submenu, the parameters of the sequence can be modified. These include basic notetaking parameters, such as the creation date of the sequence, description, name etc. and actual geometric/ execution parameters such as the printing speed, length, and tool on/off values.

After the sequence parameters have been finalized, the bracketed commands at the bottom can be used.

- ADD inserts the sequence into the active recipe at a given index
- PRIME generates the python commands that will be executed when the sequence is run and stores them in RAM
- VIEW displays the python commands in the terminal for user inspection
- GO begins execution of the program
- q quits the menu and returns to the main menu

6.1 The Sequence Library

Users can create and add sequences by cloning the python files in the polychemprint3/sequence folder and modifying parameters. By default, releases of polychemprint3 contain the following sequences built-in:

6.1.1 Basic Move Sequence

BasicMove is a sequence that control axes to move in x, y and z direction. Its menu looks like the following:

```

## Sequence: basicMove
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | basicMove | | Change if modifying from default
(P2 ) Sequence Description | Move Axes (relative or abs) | | basicMove.py
(P3 ) Creation Date    | 05/05/2020 | | dd/mm/yyyy
(P4 ) Created By      | Bijal Patel | |
(P5 ) Owner           | PCP_CoreUtilities | | default: PCP_Core
(P6 ) Positioning Mode | relative | absolute/relative | Versus current position or absolute origin
(P7 ) Axes Speed      | 60 | mm/min
(P8 ) X movement      | 5 | mm | distance/location to move in X
(P9 ) Y movement      | 5 | mm | distance/location to move in Y
(P10) Z movement      | 5 | mm | distance/location to move in Z

[Add] | Add/Insert sequence as configured into active recipe
[q] | Quit
[G0] | Engage Print Sequence
[PRIME] | Generate Print Commands
[VIEW] | View Print Commands

[/] Repeat Last Command | G0
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>

```

The top five commands (**P1** to **P5**) introduce the basic information of this sequence including the name, created date and etc. **P6** allows user to change the reference the command is execute from, whether relative to current position or the absolute reference, which is the origin. **P7** is the axes speed that controls how fast axes should move and the unit is in mm/min. **P8** through **P10** represent the distance axes is going to move move in x, y, and z direction. The unit is in mm.

PRIME,VIEW, and GO

After modifying a parameter, **GO** command allows the execution of the sequence and corresponding Gcode command will be displayed on the terminal. It is highly recommended to do **PRIME** and **VIEW** command before engaging printing sequence. **PRIME** command will generate the print commands without actual execution done by hardware and **VIEW** will show those commands on terminal for user to review. Thus, it is best to do **PRIME** and **VIEW** to check on print commands in case there is errors that might break hardware

6.1.2 Circle Sequence

Circle is a sequence that lets axes moves in a circle with radius set by user. Below is the menu of circle:

```

## Sequence: circle
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | circle | | Change if modifying from default
(P2 ) Sequence Description | a circle | | current positon of nozzle is center
(P3 ) Creation Date    | 16/11/2019 | | dd/mm/yyyy
(P4 ) Created By      | Yilong Chang | |
(P5 ) Owner           | PCP_10Core | | default: PCP_Core
(P6 ) Printing Speed   | 60 | mm/min
(P7 ) Radius           | 10 | mm
(P8 ) Steps in x and y | 0.5 | mm | smaller value lead to rounder circle
(P9 ) Tool ON Value    | 100 | null | Depends on tool loaded
(P10) Tool OFF Value   | 000 | null | Depends on tool loaded

[Add] | Add/Insert sequence as configured into active recipe
[q] | Quit
[G0] | Engage Print Sequence
[PRIME] | Generate Print Commands
[VIEW] | View Print Commands

[/] Repeat Last Command | s1
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>

```

P1 through **P5** inform users the basic information of this sequence. **P6** controls the speed of the axes movement and unit is in mm/min. **P7** command controls the radius of the circle. Since hardware like 3D printer is limited to straight move only, axes have to move step by step in x and y direction to create roughly round object. That is where **P8** comes from. It represents the step axes moves in x and y. A smaller **P8** value lead to a more circular shape.

Tool ON and OFF Value

P9 and **P10** control the on and off tool value. For example, when trying to do laser cutting, the energy of laser when print, on value, will be like 100 watts, and off value will be 0. Users should adjust the P8 and P9 based on the tool they are using.

6.1.3 Cuboid Sequence

Cuboid sequence allows a cuboid to be created. Here is what the menu looks like:

```
sequence: cuboid
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | cuboid | | Change if modifying from default
(P2 ) Sequence Description | length in x,width in y, and height in z | |
(P3 ) Creation Date   | 16/11/2019 | | dd/mm/yyyy
(P4 ) Created By     | Yilong Chang | |
(P5 ) Owner          | PCP_IDCore | | default: PCP_Core
(P6 ) Printing Speed  | 60 | |
(P7 ) Width          | 10 | mm | in y direction
(P8 ) Length         | 20 | mm | in x direction
(P9 ) Height         | 5 | mm | in z direction
(P10) Layer height   | 0.5 | mm | the amount nozzle moves up after each layer
(P11) Tool ON Value  | 100 | null | Depends on tool loaded
(P12) Tool OFF Value | 000 | null | Depends on tool loaded

[Add]                | Add/Insert sequence as configured into active recipe
[q]                  | Quit
[GO]                 | Engage Print Sequence
[PRIME]              | Generate Print Commands
[VIEW]               | View Print Commands

[/] Repeat Last Command | s2
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>
```

P1 through **P5** inform users the basic information of this sequence. **P6** controls the speed of the axes movement and unit is in mm/min. **P7** through **P9** controls the width, length, and height of the cuboid. The unit for those value is in mm. **P10** controls layer height. Since this is a 3D object, 2D layers needs to be build up to form 3D shapes. Layer height is the distance axes need to move in positive z direction when one layer is done printing.

Tool ON, **Tool OFF**, **PRIME**, **VIEW**, and **GO** commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information

6.1.4 GapLine Sequence

GapLine is a sequence that creates multiple rows of line segments in x direction with a gap between each segment. Here is what the menu looks like:

```

##      Sequence: gapLine
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) Sequence Name      | gapLine | | Change if modifying from default
(P2 ) Sequence Description | Lines in X-direction with a gap where tool raises | | gapLine.py
(P3 ) Creation Date      | 13/11/2019 | | dd/mm/yyyy
(P4 ) Created By        | Bijal Patel | |
(P5 ) Owner             | PCP_Electronics | | default: PCP_IDCore
(P6 ) Printing Speed     | 60 | mm/min |
(P7 ) Travel Speed      | 200 | mm/min |
(P8 ) x-seglength       | 10 | mm | Length of each X segment
(P9 ) X-Gap             | 2 | mm | Length of gap in X
(P10) numRows           | 3 | mm | Number of lines to print
(P11) Y-Spacing         | 3 | mm | Spacing (in y) between lines
(P12) Z-hop             | 1 | | Height to raise Z-axis for gap
(P13) Tool ON Value     | 100 | null | Depends on tool loaded
(P14) Tool Travel Value | 0 | null | Depends on tool loaded
(P15) Tool OFF Value    | 0 | null | Depends on tool loaded

[Add] | Add/Insert sequence as configured into active recipe
[q] | Quit
[GO] | Engage Print Sequence
[PRIME] | Generate Print Commands
[VIEW] | View Print Commands

[/] Repeat Last Command | go
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>

```

P1 through **P5** inform users the basic information of the gapLine sequence. **P6** controls printing speed which is how fast axes moves when materials are been printed. **P7** controls travel speed which is how fast axes move when no material is been deployed but the sequence is still running for example, during printing of the gap between lines. **P8** controls length of each line segment and **P9** represents the size of gap between line segments. **P10**, number of rows, is how many rows of line are there to be print. **P11** controls spacing between each row, which is the distance axes moves in y direction when done printing one row. **P12**, z-hop, value is the distance to move axes in positive z direction when printing gap.

Tool ON, **Tool OFF**, **PRIME**, **VIEW**, and **GO** commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information

6.1.5 Line Sequence

Line sequence allows single straight line to be created in either x or y direction. The menu looks like the following:

```

##      Sequence: line
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name              | line | | Change if modifying from default
(P2 ) Sequence Description | Single Line in X/Y Direction | | line.py
(P3 ) Creation Date      | 16/11/2019 | | dd/mm/yyyy
(P4 ) Created By        | Bijal Patel | |
(P5 ) Owner             | PCP_IDCore | | default: PCP_Core
(P6 ) Printing Speed     | 60 | |
(P7 ) Line direction     | X | | Along X or Y
(P8 ) Line Length       | 10 | mm |
(P9 ) Tool ON Value     | 100 | null | Depends on tool loaded
(P10) Tool OFF Value    | 000 | null | Depends on tool loaded

[Add] | Add/Insert sequence as configured into active recipe
[q] | Quit
[GO] | Engage Print Sequence
[PRIME] | Generate Print Commands
[VIEW] | View Print Commands

[/] Repeat Last Command | s6
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>

```

P1 through **P5** inform users the basic information of the line sequence. **P6** controls moving speed of axes when printing. **P7** lets user choose the direction of printed line. Two options are available: x or y direction. **P8**, line length,

represents distance axes move in printed direction. This unit is in mm.

Tool ON, **Tool OFF**, **PRIME**, **VIEW**, and **GO** commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information

6.1.6 Pause Sequence

Pause Sequence lets axes to pause for a certain amount time before precede to next movements. Usually pause is used to create time gap between sequences in a recipe. Here is what the pause menu looks like:

```
##      Sequence: pause
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | pause                               |                               | Change if modifying from default
(P2 ) Sequence Description | Pause execution                     |                               | pause.py
(P3 ) Creation Date   | 05/05/2020                         |                               | dd/mm/yyyy
(P4 ) Created By     | Bijal Patel                        |                               |
(P5 ) Owner          | PCP_CoreUtilities                  |                               | default: PCP_Core
(P6 ) pauseTime      | 1                                  | seconds                       | time to wait before next cmd sent
(P7 ) Prompt to Continue? | N                                  | (Y/N)                         | Will user be prompted to resume?

[Add]                | Add/Insert sequence as configured into active recipe
[q]                  | Quit
[GO]                 | Engage Print Sequence
[PRIME]              | Generate Print Commands
[VIEW]               | View Print Commands

[/] Repeat Last Command | s7
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>
```

P1 through **P5** inform users the basic information of the pause sequence. **P6**, pausetime, is the time to wait before next command is send. For example, when circle sequence and line sequence are written into recipe to be execute simultaneously, add pause sequence between them will allow axes to be stopped for certain time after completion of circle. **P7** gives user ability to control whether precede to next sequence after waiting time runs out. Typing Y in **P7** will let program ask for user prompt to resume when pause time expired, and typing N will have program automatically move on to next commands when pause time expired.

Tool ON, **Tool OFF**, **PRIME**, **VIEW**, and **GO** commands have been described in user guide **6.1.1 and 6.1.2. Please see them for more information

6.1.7 Plate Sequence

Plate Sequence will print one line after another with each line at different row and eventually forms a plate. Its menu is shown as follow:


```
## Sequence: plate
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | plate                               |                               | Change if modifying from default
(P2 ) Sequence Description | Meanderline Plate                 |                               | plate.py
(P3 ) Creation Date   | 13/11/2019                       |                               | dd/mm/yyyy
(P4 ) Created By     | Bijal Patel                      |                               |
(P5 ) Owner          | PCP_IDCore                       |                               | default: PCP_Core
(P6 ) Printing Speed  | 60                               |                               |
(P7 ) Line direction  | X                                 |                               | Along X or Y
(P8 ) Line Pitch      | 1                                 | mm                             |
(P9 ) Line Length     | 10                               | mm                             |
(P10) Number of lines | 5                                 | mm                             |
(P11) Tool ON Value   | 100                              | null                           | Depends on tool loaded
(P12) Value Increment | 0                                 |                               |
(P13) Value Operation | +                                 |                               |
(P14) Speed Increment | 0                                 |                               |
(P15) Speed Operation | +                                 |                               |

[Add]                | Add/Insert sequence as configured into active recipe
[q]                  | Quit
[GO]                 | Engage Print Sequence
[PRIME]              | Generate Print Commands
[VIEW]               | View Print Commands

[/] Repeat Last Command | view
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>
```

P1 through **P5** inform users the basic information of the line sequence. **P6** controls moving speed of axes when printing. **P7**, line direction, controls the direction each line is printed. If it is set x, straight lines will be printed in x direction and axes will move line pitch distance in y direction after the completion of each line. In order to form a plate with not no gap present between lines, **P8** should be set to the width of each line. If **P7** is set to y, straight lines will be printed in y direction with axes moving line pitch distancing in x direction after each line to create plate. **P9** controls how long each line should. **P10** is how many lines the sequence is going to print. The sequence also allows altering of print speed or tool value after completion of each line. For example, by setting **P14**, **speed increment**, to 10 and set **P15**, **speed operation**, to +, printing speed value will be added by 10 after each line. This works the same for tool value increment. Currently, four speed operations are available, +, *, -, /. But please note that having large value of increments could let to value out of bound and damage machine. Users should have a roughly estimating of the final value of speed and tool value before printing.

Tool ON, **Tool OFF**, **PRIME**, **VIEW**, and **GO** commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information.

6.1.8 Pyramid Sequence

Pyramid Sequence allows a pyramid to be created. Here is what the menu looks like:

```

Sequence: pyrimad
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | pyrimad | | | Change if modifying from default
(P2 ) Sequence Description | length in x,width in y, and height in z | | |
(P3 ) Creation Date   | 16/11/2019 | | | dd/mm/yyyy
(P4 ) Created By     | Yilong Chang | | |
(P5 ) Owner          | PCP_IDCore | | | default: PCP_Core
(P6 ) Printing Speed  | 60 | | |
(P7 ) Width          | 10 | mm | in y direction
(P8 ) Length         | 20 | mm | in x direction
(P9 ) Height         | 5 | mm | in z direction
(P10) layer height    | 0.5 | mm | the amount nozzle moves up after each layer
(P11) Tool ON Value   | 100 | null | Depends on tool loaded
(P12) Tool OFF Value  | 000 | null | Depends on tool loaded

[Add]                | Add/Insert sequence as configured into active recipe
[q]                  | Quit
[GO]                 | Engage Print Sequence
[PRIME]              | Generate Print Commands
[VIEW]              | View Print Commands

[/] Repeat Last Command | s9
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>

```

P1 through **P5** inform users the basic information of this sequence. **P6** controls the speed of the axes movement and unit is in mm/min. **P7** through **P9** control the width, length, and height of the cuboid. The unit for those value is in mm. **P10** controls layer height. Since this is a 3D object, 2D layers needs to be build up to form 3D shapes. Layer height is the distance axes need to move in positive z direction when one layer is done printing.

Tool ON, **Tool OFF**, **PRIME**, **VIEW**, and **GO** commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information

6.1.9 Rectangle Sequence

This sequence controls axes to moves following circumference of a rectangle, thus create a rectangle shape. Here is what the menu looks like:

```

### Sequence: rectangle
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | rectangle | | | Change if modifying from default
(P2 ) Sequence Description | a rectangle with length in x and width in y | | |
(P3 ) Creation Date   | 16/11/2019 | | | dd/mm/yyyy
(P4 ) Created By     | Yilong Chang | | |
(P5 ) Owner          | PCP_IDCore | | | default: PCP_Core
(P6 ) Printing Speed  | 60 | | |
(P7 ) Width          | 10 | mm | in y direction
(P8 ) Length         | 20 | mm | in x direction
(P9 ) Tool ON Value   | 100 | null | Depends on tool loaded
(P10) Tool OFF Value  | 000 | null | Depends on tool loaded

[Add]                | Add/Insert sequence as configured into active recipe
[q]                  | Quit
[GO]                 | Engage Print Sequence
[PRIME]              | Generate Print Commands
[VIEW]              | View Print Commands

[/] Repeat Last Command | s10
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>

```

P1 through **P5** inform users the basic information of the line sequence. **P6** controls moving speed of axes when printing. **P7** and **P8** controls the width and length of rectangle and the unit is in mm. Not width line is oriented in y direction and length line is oriented in x direction.

Tool ON, **Tool OFF**, **PRIME**, **VIEW**, and **GO** commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information

6.1.10 SetToolStatus Sequence

This sequence is typically implemented in a recipe between two sequences to change the status or the value of tool. Here is what it looks like:

```
## Sequence: setToolState
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | setToolState           | | | Change if modifying from default
(P2 ) Sequence Description | Set Tool Value or Dispense State | | | setToolState.py
(P3 ) Creation Date    | 05/05/2020            | | | dd/mm/yyyy
(P4 ) Created By      | Bijal Patel           | | |
(P5 ) Owner           | PCP_CoreUtilities     | | | default: PCP_Core
(P6 ) dispenseState    | off                   | | | On/Off/NoChange | Engage/Disengage/No Change
(P7 ) newVal          | NoChange              | | | New value to set: Number or NoChange | New Tool Value to Set

[Add]                | Add/Insert sequence as configured into active recipe
[q]                  | Quit
[GO]                 | Engage Print Sequence
[PRIME]              | Generate Print Commands
[VIEW]               | View Print Commands

[/] Repeat Last Command | view
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>
```

P1 through P5 informs users the basic information of the line sequence. P6 lets user decide on tool status. There choices are available: change tool status to on, off, or not change. P7 controls the new tool value. For example, if circle sequence has juts been complete with a tool on value of 100, setting P7 to 200, will alter the tool on value of the next sequence to be 200.

Not very certain what this sequence does

Tool ON, Tool OFF, PRIME, VIEW, and GO commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information

6.1.11 Triangle Sequence

Triangle Sequence allows a triangle shape to be created. Here is what the menu looks like:

```
### Sequence: triangle
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | triangle               | | | Change if modifying from default
(P2 ) Sequence Description | a triangle             | | | current positon of nozzle is center
(P3 ) Creation Date    | 16/11/2019            | | | dd/mm/yyyy
(P4 ) Created By      | Yilong Chang          | | |
(P5 ) Owner           | PCP_IDCore            | | | default: PCP_Core
(P6 ) Printing Speed   | 60                     | | |
(P7 ) Baseline length | 10                     | | | mm
(P8 ) Adjacent line length | 20                     | | | mm
(P9 ) Formed angle    | 60                     | | | degree | have to less than 180 degree
(P10) Steps in x and y | 0.5                    | | | mm | smaller value lead to better precision
(P11) Tool ON Value    | 100                    | | | null | Depends on tool loaded
(P12) Tool OFF Value   | 000                    | | | null | Depends on tool loaded

[Add]                | Add/Insert sequence as configured into active recipe
[q]                  | Quit
[GO]                 | Engage Print Sequence
[PRIME]              | Generate Print Commands
[VIEW]               | View Print Commands

[/] Repeat Last Command | s12
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:>
```

P1 through **P5** inform users the basic information of the line sequence. **P6** controls moving speed of axes when printing. The unit is in mm/min. In order to create a unique triangle, three pieces of information are needed: two sides length and the angle they formed. **P7** through **P9** control those information. For side lengths, units are in mm. For angle, the unit is in degree. **P10**, **steps in x and y**, is necessary due to the axes' limited ability of only creating straight

line. Since one side of triangle has to be diagonal line. To create it, axes needs to be move in small steps in x and y direction. The small the value is, the more accurate the triangle is going to be.

Tool ON, Tool OFF, PRIME, VIEW, and GO commands have been described in user guide 6.1.1 and 6.1.2. Please see them for more information

6.2 Importing GCode Sequences

PCP3 can natively import GCode generated by external slicer programs through specific GcodeFile sequences. For 2D (single layer) patterns, it is often easier to generate GCode from a vector image in the free software InkScape. For 3D (multilayer) patterns, Cura and slic3r are good options for converting .STL CAD files to Gcode. In both cases, it is very important to choose slicing settings that will ‘play nice’ with PCP3. For further details and tutorials, see the specific sequence pages below.

6.2.1 GcodeFileInkScape Sequence

This sequence allows locally stored Gcode file obtained through InkScape software to be imported into polychemprint3. Here is what menu of this sequence looks like:

```

### Sequence: GCodeFileInkScape
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name                | GCodeFileInkScape                |                               | Change if modifying from default
(P2 ) Sequence Description | Imported from GCodeFile           |                               |
(P3 ) Creation Date       | 06\18\2020                        |                               | dd/mm/yyyy
(P4 ) Created By         | Bijal Patel                       |                               |
(P5 ) Owner              | PCP_CoreUtilities                 |                               | default: PCP_Core
(P6 ) GCodeFilePath       | PathUnset                         |                               | Full File Path to target GCode File
(P7 ) Printing Speed      | 60                                | mm/min                       |
(P8 ) Travel Speed       | 61                                | mm/min                       |
(P9 ) Z Movement Speed   | 62                                | mm/min                       |
(P10) Z hop height        | 2                                  | mm                           |
(P11) Tool on Value       | 5                                  |                               | Tool value when dispensing
(P12) Tool off Value      | 5                                  |                               | Tool value when not dispensing
(P13) Tool travel Value   | 5                                  |                               | Tool value during travel moves

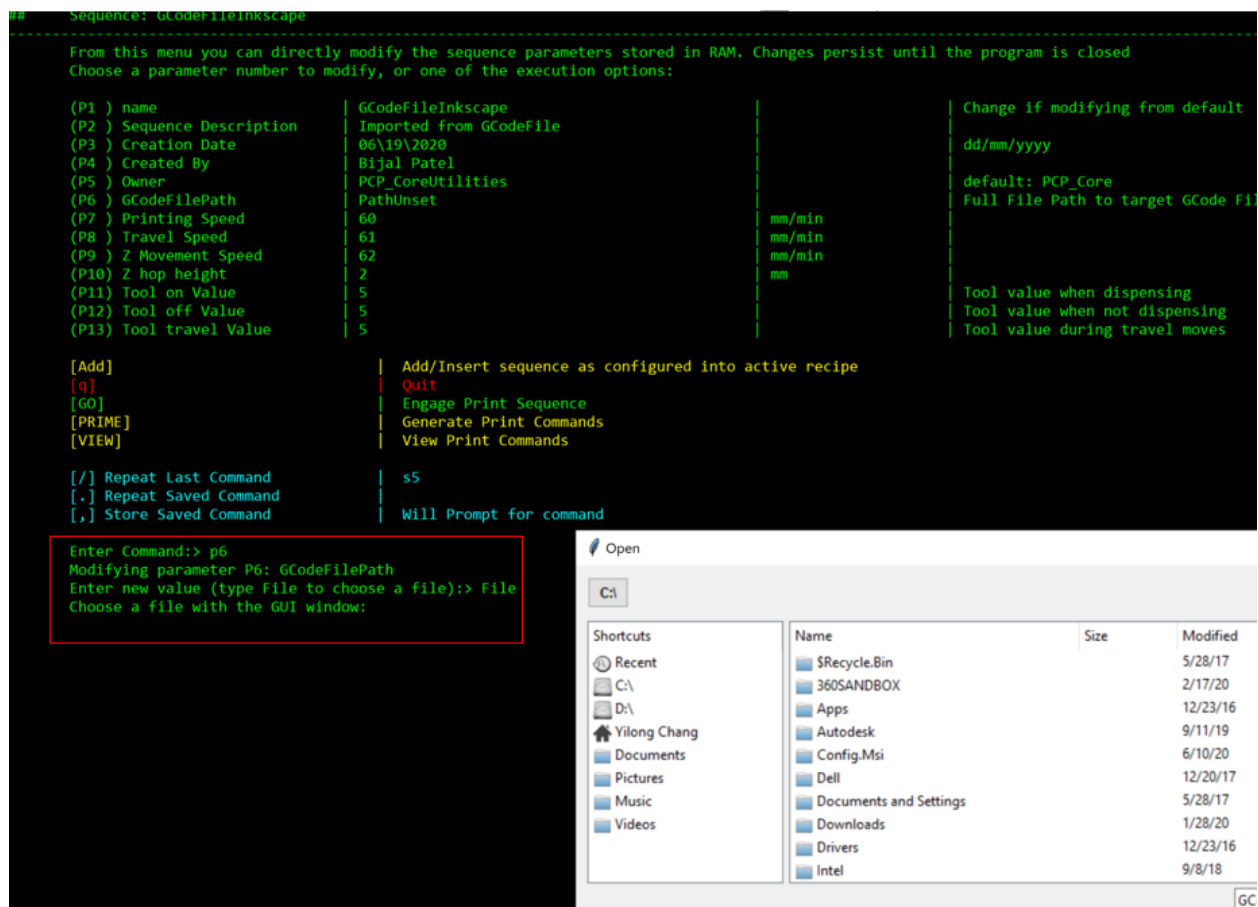
[Add]                    | Add/Insert sequence as configured into active recipe
[q]                      | Quit
[GO]                    | Engage Print Sequence
[PRIME]                 | Generate Print Commands
[VIEW]                 | View Print Commands

[/] Repeat Last Command  | s5
[.] Repeat Saved Command |
[,] Store Saved Command  | Will Prompt for command

Enter Command:>

```

P1 through **P5** inform user the basic information of the GcodeFileInkScape sequence. **P6** gives users options to select the objective file. Users can either type in the full file path or just type in File to opens up a GUI window and choose the matching file like follow:



P7 controls moving speed of axes in x and y direction when printing and **P8** represents the speed of axes when tool is not print but sequence is still running. **P9** is the speed of axes when traveling in z direction. The unit regarding speed is in mm/min. **P10, Z hop height**, controls the distance axes moves in positive z direction when two lines are crossed. Axes need to be move up to avoid collision of material, thus called hop height. The unit is in mm.

6.2.2 Inkscape 2D GCODE Tutorial

The free and open source vector image software [Inkscape](#) can be used to generate G-code for complex 2D patterns that can then be imported into polychemprint3. We will be using the plugin [GCodeTools](#) (included by default with Inkscape).

Below is a step-by-step tutorial for creating a vector image and generating PCP3-compatible GCode.

Step 1: Set Canvas Dimensions

- Open InkScape
- Click File -> Document Properties
- In the window that appears, set the page size and units. Note: the bottom left corner is the (0,0) coordinate.

Step 2: Draw your pattern (ex: text, shapes, and bezier curves)

- Create a layer for your pattern (shift + ctrl + L) to open the layer window.
- From the tools panel at left, choose either the text tool or one of the drawing tools and draw your pattern.
- For complex patterns, it can be helpful to paste a picture into a layer below your working layer, set it to ~60% opacity, lock it, and then 'trace' a pattern on your working layer with bezier curves.

- d) Note: There is technically a way to auto-generate paths from non-vector drawings, but I have not been able to get it to work. After import an image into Inkscape, you can click Path-> Trace Bitmap. In the window that appears, choose a tracing mode and your path will be generated. Unfortunately, all modes generate 'double-paths' except centerline tracing, which just crashes (as of 12/17/20).

Step 3: Convert to Path

- a) Select all of your 'artwork' in the working layer.
- b) From the top toolbar, choose Path -> Object to path
- c) Before you proceed, go to node view (n) and try to simplify the nodes as much as you can while maintaining pattern fidelity. Having a huge number of nodes will lead to having many many tiny steps that the printer will execute, increasing print time and reducing stability. If you have overlapping nodes (e.g., two lines coming to a point), use the 'join nodes' tool to combine them.
- d) At this stage, your artwork is complete. Clone this layer to a new layer above this one and hide and lock all sublayers.

Step 4: Setup Orientation Points

- a) From the top toolbar, choose Extensions -> GCodeTools -> Orientation Points.
- b) In the window that appears:
 1. Choose 2-points mode
 2. Set Z surface to 0.000
 3. Set Z-depth to -1.
- c) Click Apply when you are done and you should see two coordinates point appearing on your drawing sheet. One on the bottom left corner with coordinates of (0,0; 0,0; 0,0). The other one on the on the bottom margin of drawing sheet with coordinates of (100.0; 0.0; -1.0)

Step 5: Setup the Tool

- a) From the top toolbar, choose Extensions -> GCodeTools -> Tools Library.
- b) In the window that appears, select the 'default' tool and press Apply.
- c) A text panel will appear on top of your drawing. Move it to the side with the selection tool (S) and then select the text editing tool (T).
- d) Edit the following parameters in the text panel.
 1. Set diameter to your tool diameter (optional).
 2. Set feed to 9999.
 3. Set penetration feed to 9998.
 4. Set passing feed to 1000.

Step 6: Enter G-code processing parameters

- a) From the top toolbar, choose Extensions -> GCodeTools -> Path-to-Gcode. A window with 4 tabs will appear.
- b) In the "Path to GCode" tab, set the cutting order to "pass by pass" and use depth function d.
- c) In the "Options" tab, set the 'Offset along Z axis' to 1.00. Also, check the "Select all paths if nothing is selected" checkbox.
- d) In the "Preferences" tab:
 1. Enter the filename for your exported G-code file.
 2. Enter the full path to the export directory in the 'directory' field.

3. Set 'Z safe height for G00 move over blank' to 2.00'
- e) Generate log files if you would like.
- f) At this stage, everything is ready to generate a gcode file. Save another copy of this file and delete all sublayers.

Step 7: Generate and export G-code File

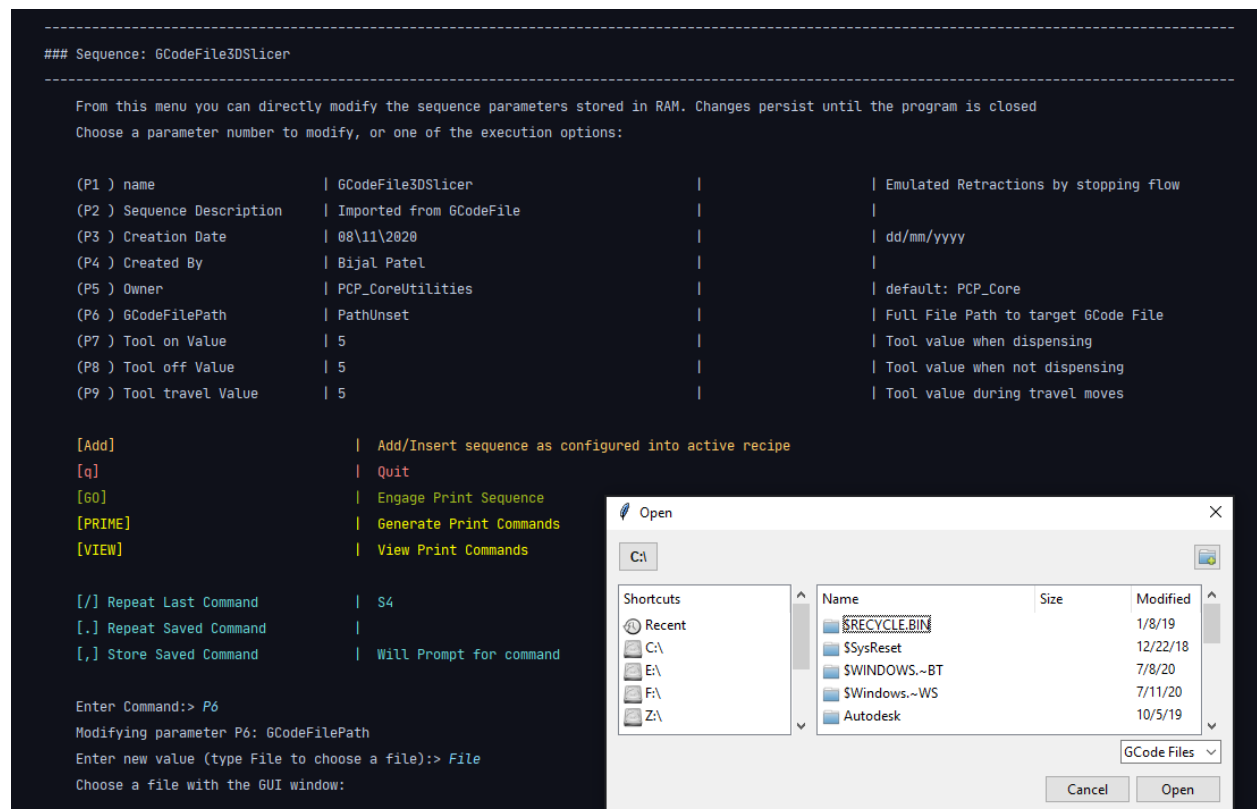
- a) With the "Path-to-Gcode" tab of the "Path-to-Gcode" panel open, click apply.
- b) If a warning appears that no paths were selected, just press ok and GCodeTools will attempt to use all paths.

Step 8: Validate GCode File [Strongly recommended]

- a) Open the G-code file that you have generated and look through it for obvious errors such as:
 1. No/ very few commands -> Likely the plugin didnt select your drawing, or your drawing wasnt in the top layer.
 2. Printing steps aren't at Z0, travel steps arent at Z3. -> you have made a mistake in steps 4 or 6.
- b) Use a program like [CAMotics](#) or [NC Viewer](#) to visually inspect the toolpath BEFORE you try it on the printer.

6.2.3 GcodeFile3DSlicer Sequence

This sequence allows locally stored Gcode file obtained through InkScape software to be imported into polychemprint3. Here is what menu of this sequence looks like:



P1 through **P5** are basic information describing the sequence. **P6** gives users options to select the .gcode file. Users can either type in the full file path or type in "File" to open up a GUI window and choose the matching file like follow:

P7 - P9 Allow the user to set the tool on, off, and travel values. Upon importing FDM Gcode, the program automatically converts from extruder positions to tool on/off triggers.

6.2.4 Cura 3D GCode Tutorial

Cura is a free software that converts CAD files (commonly .STL) into GCode for FDM printers. We do not give an exhaustive tutorial of Cura here, but instead briefly outline the process and note the parameters you must select in order for PCP3 to properly import GCode.

Step 0: Printer/ Program Settings

Either create a “Custom FFF Printer” or modify your 3D printer’s profile to reflect the following.

1. Delete any start or end Gcode.
2. Set the origin at center of the bed.
3. Select Marlin as the Gcode flavor.
4. Set the nozzle diameter to the diameter of the nozzle you are using (or feature size of LASER etch lines etc.)

Step 1: Import STL file

1. Click File on the top bar and select Open Files
2. Select the STL file you want to generate G-code from .
3. Properly adjust orientation of your object. Make sure it is in contact with the bed.
4. In the print settings make the following changes:
 - a. Set the layer height to the height of your material under the desired printing conditions
 - b. Set the print speed (for all parts) to be the desired printing speed.
 - c. Turn off build plate adhesion tools like skirts or brims (unless you want them)

Step 2: Slice Object

1. After done adjusting parameters, the bottom right corner will state “Ready to Slice”
2. Please wait until it states “Ready to Save to File” and precede to step 3

Step 3: Generate G-code

1. Click “Save to File” button on the bottom right corner
2. Select the desired directory and click save
3. GCODE file is generated.

CHAPTER 7

Recipe Menu

Recipe are a combination of sequences to be executed in series. The constituent sequences each can take on their own parameters and duplicates are allowed. This can be very useful for combinatorial screening, or even just for printing a batch of identical samples in one long run, without user intervention. It is also possible to include ‘interrupt’ sequences that prompt for user input, e.g. to confirm the nozzle is clean/ swap out substrates/ etc. between prints. Because recipes can become quite long, they are stored as text files (.yaml) and only one recipe is fully loaded into RAM at a time, the “active recipe”. For convenience, basic details of all recipes can be seen from the loading screen.

Note: Recipes are NOT meant to be created outside of PCP3. Although it is possible to edit .yaml files, it is not encouraged.

The recipe menu is shown below:

```
##      Recipe Menu
-----
Recipes are chains of sequences stored as yaml files and only loaded into RAM when active
Active Recipe: e| e| 03:06PM on June 19, 2020
Choose an option from the list below:
(1) Browse/Load Stored Recipes | Search through recipe folder for recipe to activate
(2) Modify/Save Active Recipe  | Remove/Reorder sequences, change parameters, and save to yaml file
(3) Build a New Recipe         | Start a new recipe from scratch
[q]                            | Quit
[GO]                           | Begin recipe execution
[PRIME]                        | Build active recipe into python code
[VIEW]                         | View active recipe details
[?]                            | List Commands

[/] Repeat Last Command       | 3
[.] Repeat Saved Command      |
[,] Store Saved Command       | Will Prompt for command

Enter Command:>
```

At program launch, no recipe is loaded and thus sequences cannot be added.

7.1 Creating a new Recipe

To build a new recipe, type **3, Build a New Recipe**, in **Recipe Menu** and the program will ask for user input regarding the name and description of the new recipe. The newly created recipe will be empty of any sequences. Picture below

shows the procedure on create a new recipe (commands entered are boxed by red). By default, the new recipe becomes the active recipe.

```
##      Recipe Menu
-----
Recipes are chains of sequences stored as yaml files and only loaded into RAM when active
Active Recipe: Third Try| w| 02:34PM on June 04, 2020
Choose an option from the list below:
(1) Browse/Load Stored Recipes      | Search through recipe folder for recipe to activate
(2) Modify/Save Active Recipe       | Remove/Reorder sequences, change parameters, and save to yaml file
(3) Build a New Recipe              | Start a new recipe from scratch
[q]                                 | Quit
[G0]                                | Begin recipe execution
[PRIME]                             | Build active recipe into python code
[VIEW]                              | View active recipe details
[?]                                 | List Commands

[/] Repeat Last Command             | 3
[.] Repeat Saved Command            |
[,] Store Saved Command              | Will Prompt for command

Enter Command:> 3
      Enter new recipe name:> User Menu
      Enter new recipe description:> Test for User Menu
      Successfully created new recipe!
```

7.2 Modifying/Saving Active Recipe

Type 2 in **Recipe Menu** to access the **Modify/Save Active Recipe** menu and users can adjust that recipe by remove/add sequences, change parameters, or saving to the yaml file.

```
##      Modify/Save Active Recipe:
-----
From this menu you can modify the active recipe stored in RAM. Changes persist until the program is closed, unless saved to file
Choose an edit/save operation:
(SAVE) Save Recipe to File          | Writes to yaml file in recipe folder
(0) Edit Recipe Parameters          | Edit Name, Description, Update Date
(1) Add Sequence                    | Inserts sequence at specified position
(2) Edit Sequence                   | Edit a sequence that is already present
(3) Remove Sequence                 | Removes one or more sequences
(4) Reorder Sequences               | Change the order of sequence execution
[q]                                 | Quit

(P0) Active Recipe Name: Third Try
(P1) Description: w
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List
-----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | line          | PCP_IDCore    | Single Line in X/Y Direction |
| ( S1 ) | circle        | PCP_IDCore    | a circle    |
| ( S2 ) | plate         | PCP_IDCore    | Meanderline Plate |
End Sequence List
Enter Command:>
```

Edit basic information One bottom portion of **Modify/Save Active Recipe** menu, information about activated recipe is shown. Description information includes name, description, creation date are shown. To modify these information, type 0 in terminal to edit text in P0 through P2(commands entered are boxed by red).

```

##      Modify/Save Active Recipe:
-----
From this menu you can modify the active recipe stored in RAM. Changes persist until the program is closed, unless saved to file
Choose an edit/save operation:
(SAVE) Save Recipe to File      | Writes to yaml file in recipe folder
(0) Edit Recipe Parameters      | Edit Name, Description, Update Date
(1) Add Sequence                | Inserts sequence at specified position
(2) Edit Sequence               | Edit a sequence that is already present
(3) Remove Sequence             | Removes one or more sequences
(4) Reorder Sequences           | Change the order of sequence execution
[q]                             | Quit

(P0) Active Recipe Name: Third Try
(P1) Description: w
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List
-----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | line          | PCP_1DCore    | Single Line in X/Y Direction |
| ( S1 ) | circle        | PCP_1DCore    | a circle    |
| ( S2 ) | plate         | PCP_1DCore    | Meanderline Plate |
-----
End Sequence List
Enter Command:> 0
Which parameter to edit? (P0-P2): > p1
Enter new description: > User Menu

```

Add Sequence To add a sequence, type **1** in the terminal and user will be brought to print sequence menu to choose a sequence to add. Type the code of the sequence user want and the matching sequence menu will be displayed for user to edit parameters. (For more detailed information about sequences, please see sequence menu). After finishing modifying parameters, type in **ADD** in terminal to add that sequence to sequence list, program will ask for user input about which index should the new sequence be occupying. This decides on the executing order sequences. Pictures below reveals process of add line sequence to a recipe (commands entered are boxed by red).

1. Type in **1** to add sequence command

```

##      Modify/Save Active Recipe:
-----
From this menu you can modify the active recipe stored in RAM. Changes persist until the program is closed, unless saved
Choose an edit/save operation:
(SAVE) Save Recipe to File      | Writes to yaml file in recipe folder
(0) Edit Recipe Parameters      | Edit Name, Description, Update Date
(1) Add Sequence                | Inserts sequence at specified position
(2) Edit Sequence               | Edit a sequence that is already present
(3) Remove Sequence             | Removes one or more sequences
(4) Reorder Sequences           | Change the order of sequence execution
[q]                             | Quit

(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List
-----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | plate         | PCP_1DCore    | Meanderline Plate |
| ( S1 ) | circle        | PCP_1DCore    | a circle    |
-----
End Sequence List
Enter Command:> 1
Sending to Sequence Menu...

```

2. Select a sequence(in this example, select line sequence)

```
##      Print Sequence Menu
-----
Sequences are pre-programmed, parameterized print routines stored as python files and loaded to RAM when the program launches.
Choose a sequence code to Edit/Execute:

(S0) basicMove          | PCP_CoreUtilities      | Move Axes (relative or abs)
(S1) circle             | PCP_IDCore            | a circle
(S2) cuboid             | PCP_IDCore            | length in x,width in y, and height in z
(S3) gapLine           | PCP_Electronics       | Lines in X-direction with a gap where tool raises
(S4) GCodeFile3DSlicer  | PCP_CoreUtilities     | Imported from GCodeFile
(S5) GCodeFileInkscape  | PCP_CoreUtilities     | Imported from GCodeFile
(S6) line               | PCP_IDCore            | Single Line in X/Y Direction
(S7) pause             | PCP_CoreUtilities     | Pause execution
(S8) plate              | PCP_IDCore            | Meanderline Plate
(S9) pyrimad            | PCP_IDCore            | length in x,width in y, and height in z
(S10) rectangle         | PCP_IDCore            | a rectangle with length in x and width in y
(S11) setToolState      | PCP_CoreUtilities     | Set Tool Value or Dispense State
(S12) triangle          | PCP_IDCore            | a triangle

[q]                     | Quit
[?]                     | List Commands

[/] Repeat Last Command | 1
[.] Repeat Saved Command
[,] Store Saved Command | Will Prompt for command

Enter Command:> s6
```

3. Modify line sequence parameter and type in command add

```
##      Sequence: line
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1) name               | line                  |          | Change if modifying from default
(P2) Sequence Description | Single Line in X/Y Direction | line.py
(P3) Creation Date       | 16/11/2019           |          | dd/mm/yyyy
(P4) Created By          | Bijal Patel          |          |
(P5) Owner               | PCP_IDCore           |          | default: PCP_Core
(P6) Printing Speed      | 60                   |          |
(P7) Line direction      | X                    |          | Along X or Y
(P8) Line Length         | 10                   | mm       |
(P9) Tool ON Value       | 100                  | null     | Depends on tool loaded
(P10) Tool OFF Value      | 000                  | null     | Depends on tool loaded

[Add]                   | Add/Insert sequence as configured into active recipe
[q]                     | Quit
[G0]                    | Engage Print Sequence
[PRIME]                 | Generate Print Commands
[VIEW]                  | View Print Commands

[/] Repeat Last Command | s6
[.] Repeat Saved Command
[,] Store Saved Command | Will Prompt for command

Enter Command:> add
```

4. Select the index to be occupied by new sequence

```
Enter Command:> add
Current state of Recipe:
(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Thir d Try.yaml
Begin Sequence List
-----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | plate        | PCP_IDCore    | Meanderline Plate |
| ( S1 ) | circle       | PCP_IDCore    | a circle          |
End Sequence List
Enter index to be occupied by new sequence: > 0
New state of Recipe:
(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Thir d Try.yaml
Begin Sequence List
-----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | line         | PCP_IDCore    | Single Line in X/Y Direction |
| ( S1 ) | plate        | PCP_IDCore    | Meanderline Plate |
| ( S2 ) | circle       | PCP_IDCore    | a circle          |
End Sequence List
```

Edit Sequence To edit a sequence that is already in sequence list, type 2 in terminal under Modify/Save Active Recipe menu. The terminal will ask for user input of sequence code (in the form of S#). Select the sequence wanted

to be modified and the matching sequence menu will be brought up. Edit parameters and type in q to quit the sequence menu. The editing process of sequence is complete. Following pictures shows a process of editing line sequence (commands entered are boxed by red).

1. Select Edit Sequence

```
##      Modify/Save Active Recipe:
-----
From this menu you can modify the active recipe stored in RAM. Changes persist until the program is closed, unless saved to file
Choose an edit/save operation:
(SAVE) Save Recipe to File      | Writes to yaml file in recipe folder
(0) Edit Recipe Parameters      | Edit Name, Description, Update Date
(1) Add Sequence                | Inserts sequence at specified position
(2) Edit Sequence               | Edit a sequence that is already present
(3) Remove Sequence             | Removes one or more sequences
(4) Reorder Sequences           | Change the order of sequence execution
[q]                             | Quit

(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List
-----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | line          | PCP_1DCore    | Single Line in X/Y Direction |
| ( S1 ) | plate         | PCP_1DCore    | Meanderline Plate |
| ( S2 ) | circle        | PCP_1DCore    | a circle |
-----
End Sequence List
Enter Command:> 2
Enter Index (S#) for Sequence to modify:> s0
```

2. Modify parameters

```
##      Sequence: line
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name           | line | | | Change if modifying from default
(P2 ) Sequence Description | Single Line in X/Y Direction | | | line.py
(P3 ) Creation Date    | 16/11/2019 | | | dd/mm/yyyy
(P4 ) Created By       | Bijal Patel | | |
(P5 ) Owner            | PCP_1DCore | | | default: PCP_Core
(P6 ) Printing Speed   | 100 | | |
(P7 ) Line direction   | X | | | Along X or Y
(P8 ) Line Length      | 100 | mm |
(P9 ) Tool ON Value    | 100 | null | Depends on tool loaded
(P10) Tool OFF Value   | 000 | null | Depends on tool loaded

[Add]                 | Add/Insert sequence as configured into active recipe
[q]                   | Quit
[G0]                  | Engage Print Sequence
[PRIME]               | Generate Print Commands
[VIEW]                | View Print Commands

[/] Repeat Last Command | s0
[.] Repeat Saved Command
[,] Store Saved Command | Will Prompt for command

Enter Command:> p10
Modifying parameter P10: Tool OFF Value
Enter new value (type File to choose a file):> 10000
Value changed from 000 to 10000
```

3. Quit the sequence menu

```

## Sequence: line
-----
From this menu you can directly modify the sequence parameters stored in RAM. Changes persist until the program is closed
Choose a parameter number to modify, or one of the execution options:

(P1 ) name | line | | | Change if modifying from default
(P2 ) Sequence Description | Single Line in X/Y Direction | | line.py
(P3 ) Creation Date | 16/11/2019 | | dd/mm/yyyy
(P4 ) Created By | Bijal Patel | |
(P5 ) Owner | PCP_IDCore | | default: PCP_Core
(P6 ) Printing Speed | 100 | |
(P7 ) Line direction | X | | Along X or Y
(P8 ) Line Length | 100 | mm |
(P9 ) Tool ON Value | 100 | null | Depends on tool loaded
(P10) Tool OFF Value | 10000 | null | Depends on tool loaded

[Add] | Add/Insert sequence as configured into active recipe
[q] | Quit
[GO] | Engage Print Sequence
[PRIME] | Generate Print Commands
[VIEW] | View Print Commands

[/] Repeat Last Command | 10000
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:> q

```

Remove Sequence To remove a sequence in sequence list, type **3, Remove sequence**, in terminal under **Modify/Save Active Recipe** menu and then type in the index of sequence that needs to be removed. The following picture shows the removal of line sequence (commands entered are boxed by red).

```

(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List -----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | line | PCP_IDCore | Single Line in X/Y Direction |
| ( S1 ) | plate | PCP_IDCore | Meanderline Plate |
| ( S2 ) | circle | PCP_IDCore | a circle |
End Sequence List -----
Enter Command:> 3
Enter index of sequence to remove (S#), q to cancel: > S0

Modify/Save Active Recipe:
-----
From this menu you can modify the active recipe stored in RAM. Changes persist until the program is closed, unless saved to file
Choose an edit/save operation:
(SAVE) Save Recipe to File | Writes to yaml file in recipe folder
(0) Edit Recipe Parameters | Edit Name, Description, Update Date
(1) Add Sequence | Inserts sequence at specified position
(2) Edit Sequence | Edit a sequence that is already present
(3) Remove Sequence | Removes one or more sequences
(4) Reorder Sequences | Change the order of sequence execution
[q] | Quit

(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List -----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | plate | PCP_IDCore | Meanderline Plate |
| ( S1 ) | circle | PCP_IDCore | a circle |
End Sequence List -----
Enter Command:>

```

Recorder Sequence To change the order of sequence in recipe, type **4, Recorder Sequence**, in terminal under **Modify/Save Active Recipe** menu and type the index of sequence that need to be changed and then typed the index that sequence is going to occupy. A single re-order process is complete. This re-order procedure will be continued until user type **q** to quit the re-order sequence function. The following picture shows the change order process (commands entered are boxed by red).

```

(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List -----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | plate         | PCP_1DCore    | Meanderline Plate |
| ( S1 ) | circle        | PCP_1DCore    | a circle           |
End Sequence List -----
Enter Command: > 4
Enter index of sequence to move (S#), q to finish: > s0
Enter index you would like the sequence to occupy (S#), q to cancel: > s1
New state:
(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List -----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | circle        | PCP_1DCore    | a circle           |
| ( S1 ) | plate         | PCP_1DCore    | Meanderline Plate |
End Sequence List -----
Enter index of sequence to move (S#), q to finish: > s1
Enter index you would like the sequence to occupy (S#), q to cancel: > s0
New state:
(P0) Active Recipe Name: Third Try
(P1) Description: User Menu
(P2) Creation Date: 02:34PM on June 04, 2020
Full Path: c:\users\yilong chang\anaconda3\lib\site-packages\polychemprint3\recipes\Third Try.yaml
Begin Sequence List -----
| Index | Sequence Name | Sequence Type | Description |
| ( S0 ) | plate         | PCP_1DCore    | Meanderline Plate |
| ( S1 ) | circle        | PCP_1DCore    | a circle           |
End Sequence List -----
Enter index of sequence to move (S#), q to finish: > q

```

Save Modifications After finishing editing recipe, users must save the modified sequence list to yaml file in recipe folder. Otherwise, if the program is closed or reset, changes made will not no longer exist. To save a recipe, type **save** in terminal under **Modified/Save Active** menu. (Note, after adding a sequence and quit the Sequence Menu, program will not return to **Modified/Save Active** menu, please remember to return to **Main Menu**, and go to **Recipe Menu**, and go to **Modified/Save Active** Recipe menu in order to do save command)

Quit Modify/Save Active Recipe Menu To exit out of the Hardware Menu, type **q** in the command and you will be in the **Recipe Menu**.

7.3 Browse/Load Stored Recipes

Type command **1** in the **Recipe Menu** will allow program to search through polychemprint3/recipe folder and display names, creation dates and descriptions of recipes stored locally.

```

###  Recipe Menu
-----
Recipes are chains of sequences stored as yaml files and only loaded into RAM when active
Active Recipe: Third Try | w | 02:34PM on June 04, 2020
Choose an option from the list below:
(1) Browse/Load Stored Recipes | Search through recipe folder for recipe to activate
(2) Modify/Save Active Recipe | Remove/Reorder sequences, change parameters, and save to yaml file
(3) Build a New Recipe | Start a new recipe from scratch
[q] | Quit
[GO] | Begin recipe execution
[PRIME] | Build active recipe into python code
[VIEW] | View active recipe details
[?] | List Commands

[/] Repeat Last Command | ?
[.] Repeat Saved Command |
[,] Store Saved Command | Will Prompt for command

Enter Command:> 1
Refreshing Recipe Stub List...
| Code | Name | Date Created | Description |
| (1) | First Try | 02:19PM on June 04, 2020 | d |
| (2) | Larry's try | 02:25PM on June 17, 2020 | 2 |
| (3) | Second Try | 02:28PM on June 04, 2020 | w |
| (4) | Third Try | 02:34PM on June 04, 2020 | w |
Choose a recipe to activate, or q to cancel: > 4
Attempting to load full recipe from yaml file in recipe folder...
New recipe activated!

```

By entering a recipe code(1,2,3,4,etc.), users can choose to activate a recipe in the program. The current active recipe name will also be shown in recipe menu (boxed in yellow).

7.4 View Recipe Details

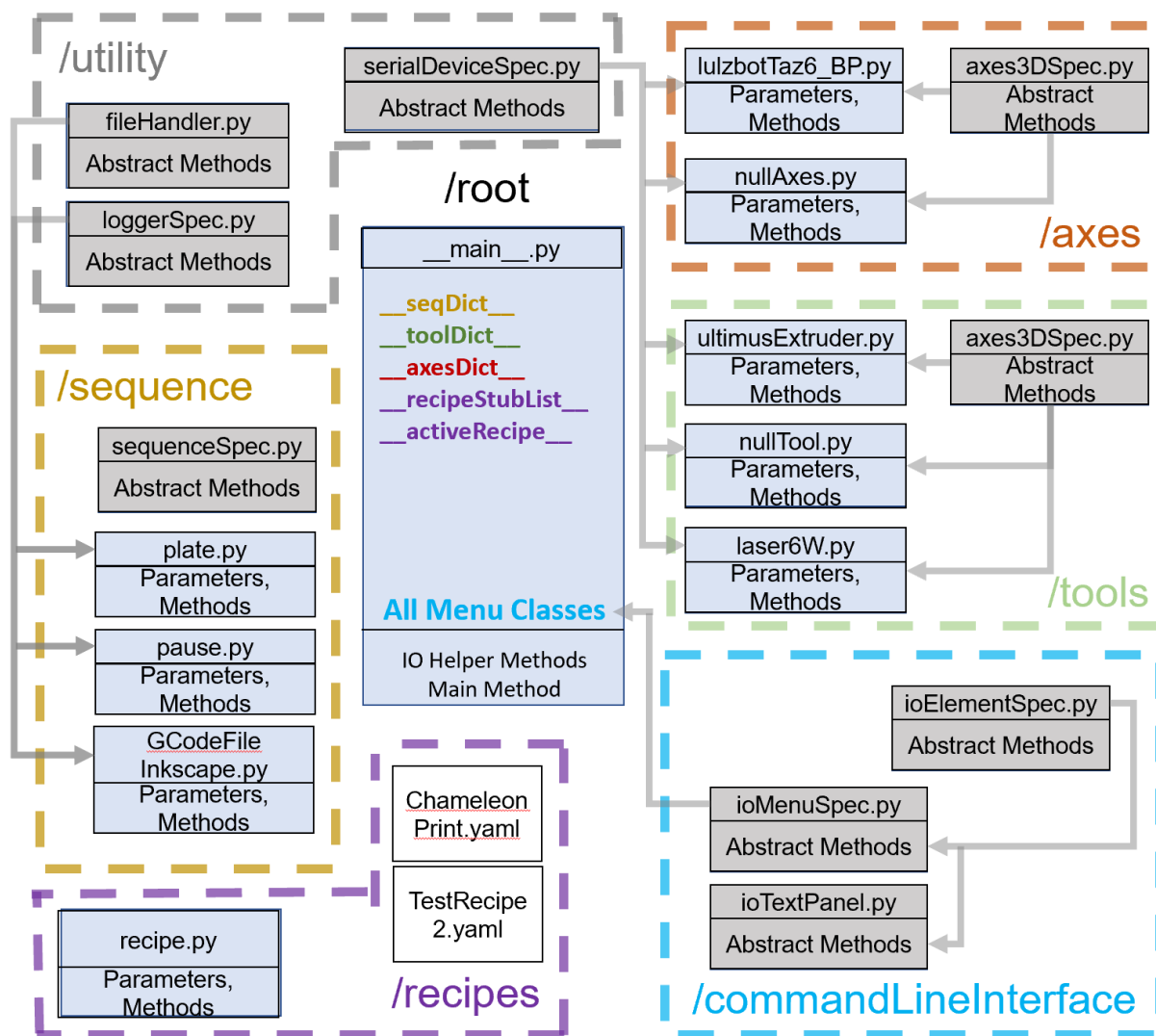
To view information on activated recipe, type **view** under **Recipe Menu** and terminal will deliver information of name, description, and also the contents of the active recipe.

7.5 Execute Recipe Menu

To execute a recipe, first type **Prime** in terminal under **Recipe Menu** to convert active recipe into python code. Then type in **GO** to begin recipe execution. Before the recipe begins, it will prompt for a log file name to deposit full recipe details into. Logs are saved as text files in the polychemprint3/Logs folder.

PCP3 Package Overview

PCP3 is written in Python following object-oriented programming (OOP) principles, meaning that data and methods are organized in terms of distinct “objects” with clearly defined attributes and behaviors, which are only instantiated (given specific values) in the `__main__.py` method. This modular approach (see Figure below) greatly simplifies the process of adding additional hardware and sequences by isolating these blocks of code into distinct submodules (folders) that are separate from the more complex user interface and main methods. In order to add a new sequence to the program, for example, the user can simply clone one of the existing light blue `.py` files, rename it and modify the contents to contain the desired motion/tool commands, and place it in the sequence folder. On startup, PCP3 automatically attempts to load any new sequences, axes, or tool `.py` files, after checking for compiler errors.



PCP3 also takes advantage of the OOP concept of inheritance to streamline addition of new code files to the program. The grey boxes in the Figure are effectively blueprints ('Abstract Base Classes', in Python) for the class declarations (light blue .py files) that inherit from them (signified by grey arrows from the parent to the child class). This system enforces a standardized format for each type of code file. All 'sequences' are required to behave following the rules of the parent `sequenceSpec` class to compile properly, or else they are not loaded when PCP3 starts. In addition to minimizing runtime errors, this approach provides another way to isolate the user from having to deal with repetitive boilerplate code common across many different objects. For example, there is no need to explicitly write logging methods for each sequence, because they already inherit them from the `loggerSpec` Abstract Base Class. Finally, the main method contains the data structures which hold all of the instantiated objects as well as containing all Menu classes and driving the user interface.

Documentation generated using the sphinx autodoc method.

9.1 polychemprint3.commandLineInterface

9.1.1 polychemprint3.commandLineInterface.ioElementSpec module

Contains ioElementSpec Abstract Base Class.

First created on Sun Oct 20 00:03:21 2019

Revised (dd/mm/yyyy): 20/12/2020 - BP

Author: Bijal Patel

```
class polychemprint3.commandLineInterface.ioElementSpec.ioElementSpec (name,  
                                                                **kwargs)
```

Bases: abc.ABC

Specifies the interface for CLI menus/text/etc.

io_Operate ()

Do the primary purpose of the CLI element.

Returns an optional flag which either reflects how operation went, or is direction for future CLI operations.

Return type str

9.1.2 polychemprint3.commandLineInterface.ioMenuSpec module

Contains ioMenuSpec Abstract Base Class.

First created on Sun Oct 20 00:03:21 2019

Revised (dd/mm/yyyy): 20/12/2020 - BP

Author: Bijal Patel

```
class polychemprint3.commandLineInterface.ioMenuSpec.ioMenuSpec(menuTitle,
                                                                menuItems,
                                                                menuDesc="",
                                                                menuIn-
                                                                struc='Choose
                                                                from the
                                                                following
                                                                menu items:',
                                                                lastCmd="",
                                                                memCmd="",
                                                                **kwargs)

Bases: polychemprint3.commandLineInterface.ioElementSpec.ioElementSpec, abc.
ABC
```

Specifies the interface for CLI menus.

ioMenu_printMenu (*showStoredCmds=True*)
Prints formatted menu options from menuItems dict.

ioMenu_updateStoredCmds (*lastCmd*, *memCmd*)
Updates stored commands local to this menu item from inputs.

Parameters

- **lastCmd** (*str*) – specifying the last command entered
- **memCmd** (*str*) – specifying the command saved to memory

io_Operate ()
Performs menu operations and loops on user input. :returns: Title of next menu to present. :rtype: str

9.1.3 polychemprint3.commandLineInterface.ioTextPanel module

Contains ioTextPanelSpec Abstract Base Class.

First created on Sun Oct 20 00:03:21 2019

Revised: 20/10/2019 00:34:27

Author: Bijal Patel

```
class polychemprint3.commandLineInterface.ioTextPanel.ioTextPanel(panelTitle,
                                                                full-
                                                                FilePath,
                                                                **kwargs)

Bases: polychemprint3.utility.fileHandler.fileHandler, polychemprint3.
commandLineInterface.ioElementSpec.ioElementSpec

Specifies the interface for CLI menus.

io_Operate ()
    Prints formatted text from file.
```

9.2 polychemprint3.data

9.2.1 Subpackages

polychemprint3.data.TextPanels package

Module contents

9.2.2 Module contents

9.3 polychemprint3.axes

9.3.1 polychemprint3.axes.axes3DSpec module

Specifies 3D Axes classes, implementations are for specific printers/stages.

First created on Sat Oct 19 20:39:58 2019

Revised: 23/10/2019 14:06:59

Author: Bijal Patel

```
class polychemprint3.axes.axes3DSpec.Axes3DSpec (name, __verbose__=0, pos-
Mode='absolute', **kwargs)
    Bases: polychemprint3.utility.loggerSpec.loggerSpec, abc.ABC
```

Abstract Base Class for 3D Axes.

activate ()

Makes required connections and returns status bool.

Returns True if ready to use False if not ready

Return type bool

deactivate ()

Closes communication and returns status bool.

Returns True if ready to use False if not ready

Return type bool

getAbsPosXY ()

Gets the current position (absolute) and return XY positions.

Parameters **command** (*String*) – Gcode to write to axes

Returns [X, Y] X and Y positions as strings

Return type String

move (*gcodeString*)

Moves to the specified gcodeString position.

Parameters **gCodeString** (*String*) – Motion command in terms of Gcode G0/G1/G2/G3 supported

poll (*command*)

Sends message to axes and returns response.

Parameters **command** (*String*) – to write to axes

Returns Response from axes

Return type String

sendCmd (*command*)

Writes command to axes device when ready.

Parameters **command** (*String*) – to write to axes

setPosMode (*newPosMode*)

Sets positioning mode to relative or absolute.

Parameters **newPosMode** (*String*) – Positioning mode to use for future move cmds

setPosZero ()

Sets the current position (absolute) to (0,0,0).

9.3.2 polychemprint3.axes.lulzbotTaz6_BP module

Implements axes3DSpec for lulzbot taz 6 with modified firmware.

First created on Sat Oct 19 20:39:58 2019

Revised: 23/10/2019 14:06:59

Author: Bijal Patel

```
class polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP (name='LulzbotTaz6',
                                                    posMode='relative', dev
                                                    vAddress='/dev/ttyACM0',
                                                    baudRate=115200, comm-
                                                    sTimeOut=0.001, __ver-
                                                    bose__=1,      firmware-
                                                    Vers='BP')
```

Bases: `polychemprint3.utility.serialDeviceSpec.serialDeviceSpec,`
`polychemprint3.axes.axes3DSpec.Axes3DSpec`

Implemented interface for Lulzbot Taz 6 with BP modified firmware.

activate ()

Makes required connections and returns status bool.

Returns True if ready to use False if not ready

Return type bool

deactivate ()

Closes communication and returns status bool.

Returns True if closed succesfully False if failed

Return type bool

getAbsPosXY ()

Gets the current position (absolute) and return XY positions.

Parameters **command** (*String*) – Gcode to write to axes

Returns [X, Y] X and Y positions as strings

Return type String

handShakeSerial ()

Perform communications handshake with serial device.

Returns

- [1, "Handshake Successful"] – success occurred
- [0, 'Handshake Failed, Rcvd + message received'] – failure occurred
- [-1, "Error (Handshake with Tool Failed + error text)"] – Error received

move (gcodeString)

Moves axes by set amount.

Parameters

- **gCodeString** (String) – Motion command in terms of Gcode G0/G1/G2/G3 supported
- **Returns** () –
- **none** () –

poll (command)

Sends message to axes and parses response.

Parameters **command** (String) – to write to axes

Returns Response from axes

Return type String

readTime ()

Reads in from serial device until timeout.

Returns All text read in, empty string if nothing

Return type String

sendCmd (command)

Writes command to axes device when ready.

Parameters **command** (String) – to write to axes

Returns Response from axes

Return type String

setPosMode (newPosMode)

Sets positioning mode to relative or absolute.

Parameters **newPosMode** (String) – Positioning mode to use for future move cmds

setPosZero ()

Sets current axes position to absolute (0,0,0).

startSerial ()

Creates pySerial device.

Returns

- [1, "Serial Device Started successfully"] – started successfully
- [-1, 'Failed Creating pySerial...'] – could not start

stopSerial()

Closes serial devices.

Returns

- [1, "Terminated successfully"] – started successfully
- [-1, "Error (Serial Device could not be stopped + error text)"]

waitReady()

Looks for "ok" in input, waits indefinitely.

Returns All text read in, empty string if nothing

Return type String

writeReady(command)

Sends command only after receiving ok message.

Parameters

command, string to write to axes

Returns

inp, String read in

9.3.3 polychemprint3.axes.nullAxes module

Implements axes3DSpec as null axes (returns successful to all).

First created on Sat Oct 19 20:39:58 2019

Revised: 23/10/2019 14:06:59

Author: Bijal Patel

```
class polychemprint3.axes.nullAxes.nullAxes (name='nullAxes', posMode='relative',  
                                             __verbose__=0)
```

Bases: `polychemprint3.axes.axes3DSpec.Axes3DSpec`

Implementing axes3D for null case.

activate()

Makes required connections and returns status bool.

Returns True if ready to use False if not ready

Return type bool

deactivate()

Closes communication and returns status bool.

Returns True if closed successfully False if failed

Return type bool

getAbsPosXY()

Gets the current position (absolute) and return XY positions.

Parameters `command` (*String*) – Gcode to write to axes

Returns [X, Y] X and Y positions as strings

Return type String

move (*gcodeString*)

Initializes Axes3D object.

Parameters

- **gCodeString** (*String*) – Motion command in terms of Gcode G0/G1/G2/G3 supported
- **Returns** () –
- **none** () –

poll (*command*)

Sends message to axes and parses response.

Parameters `command` (*String*) – to write to axes

Returns Response from axes

Return type String

sendCmd (*command*)

Writes command to axes device when ready.

Parameters `command` (*String*) – to write to axes

setPosMode (*newPosMode*)

Sets positioning mode to relative or absolute.

Parameters `newPosMode` (*String*) – Positioning mode to use for future move cmds

setPosZero ()

Sets current axes position to absolute (0,0,0).

9.4 polychemprint3.tools

9.4.1 polychemprint3.tools.laser6W module

Implements the Tool base class for Danny's arduino-uno controlled 6W LASER.

First created on Wed Feb 12 2020

Revised: 12/02/2020

Author: Bijal Patel

```
class polychemprint3.tools.laser6W.laser6W (name='BlueLASER6W',      units='percent',
                                           devAddress='/dev/ttyACM1',    baudRate=115200,
                                           commsTimeOut=0.001,
                                           __verbose__=1, **kwargs)

Bases:      polychemprint3.utility.serialDeviceSpec.serialDeviceSpec,
           polychemprint3.tools.toolSpec.toolSpec
```

Implements the Tool base class for Danny's 6W LASER.

activate()

Makes required connections and returns status bool.

Returns True if ready to use False if not ready

Return type bool

checkIfSerialConnectParamsSet()

Goes through connection parameters and sees if all are set.

Returns True if all parameters are set, false if any unset

Return type bool

deactivate()

Closes communication and returns status bool.

Returns True if deactivated False if not deactivated

Return type bool

disengage()

Toggles Dispense off.

Returns

- [1, "Dispense Off"]
- [0, "Error (Dispense already off)"]
- [-1, 'Failed engaging dispense ' + inst.__str__()]

engage()

Toggles Dispense on.

Returns

- [1, "Dispense On"]
- [0, "Error (Dispense Already On)"]
- [-1, 'Failed engaging dispense ' + inst.__str__()]

getState()

Returns active state of tool.

Parameters

none

Returns

- [1, "Tool On"]
- [0, "Tool Off"]
- [-1, "Error: Tool activation state cannot be determined + Error"]

handShakeSerial()

Perform communications handshake with Tool.

Returns

- [1, "Handshake Successful"]

- [0, 'Handshake Failed, Received (+ message received)'] – if unexpected input received
- [-1, "Error (Handshake with Tool Failed + error text)"]

loadLogSelf (*jsonString*)
loads json log back into dict.

Parameters *jsonString* (*String*) – json string to be loaded back in

readTime ()
Reads in from serial device until timeout.

Returns

- [1, *inp String* of all text read in, empty string if nothing]
- [0, 'Read failed + Error' if exception caught]

setValue (*value*)
Set Laser PWM value in percent 1-100.

Parameters *value* (*String*) – New value of pressure out of 100

Returns

- [output of *writeSerialCommand*]
- [-1, "Error (value could not be set for LASER + error text)"]

startSerial ()
Creates and connects pySerial device.

Returns

- [1, "Connected Succesfully to Serial Device"]
- [0, 'Not all connection parameters set']
- [-1, "Error (Could not connect to serial device: + error text)"]

stopSerial ()
Terminates communication.

Returns

- [1, "Terminated successfully"]
- [-1, "Error (Tool could not be stopped + error text)"]

writeLogSelf ()
Generates json string containing dict to be written to log file.

Returns *logJson* – log in json string format

Return type *String*

9.4.2 polychemprint3.tools.nullTool module

Implements the Tool base class for a null Tool [no action, returns true].

First created on 13/11/2019 13:33:28

Revised: 17/10/20

Author: Bijal Patel

```
class polychemprint3.tools.nullTool.nullTool (name='nullTool', units='null', devAd-  
dress='unset', baudRate='unset', comm-  
sTimeOut=0.5, __verbose__=0, **kwargs)
```

Bases: `polychemprint3.tools.toolSpec.toolSpec`

Implements the toolSpec abstract base class for a null tool, a virtual hardware device that only writes to the terminal.

activate()

To be called in main.py to load as active tool. Makes required serial connections and returns status as True/False.

Returns True if tool serial connection made and tool is ready to use False if error generated and tool is not ready for use

Return type bool

deactivate()

To be called in main.py to unload as active tool. Closes serial communication and returns status as True/False.

Returns True if tool serial connection destroyed and tool is succesfully disabled. False if error generated and serial communication could not be suspended.

Return type bool

disengage()

Turn tool primary action off (stops dispense/LASER beam off, etc).

Returns

status – First element (int) indicates whether disengage was successful (1), already off (0), or error (-1).

Second element (String) provides text explanation.

Return type two-element list

engage()

Turn tool primary action on (dispense/LASER beam on, etc).

Returns

status – First element (int) indicates whether engage was successful (1), already on (0) or error (-1)

Second element (String) provides text explanation.

Return type two-element list

getState()

Returns the current dispense/action state (on/off).

Returns

status – First element indicates whether tool is on(1) or off(0) or error(-1).

Second element provides text explanation.

Return type two-element list

loadLogSelf(yamlString)

loads json log back into dict.

Parameters **yamlString** (*String*) – yaml string to be loaded back in

setValue (*value*)

Set the primary tool action value (e.g., Laser power, extruder pressure, etc.).

Parameters **value** (*String*) – The new value of the parameter as a string, expressed at arbitrary precision/ without leading zeros. Conversion to hardware specific format occurs internally. e.g., (use 23.456 NOT 0234”)

Returns

status – First element (int) indicates whether value setting was successful (1) or error (-1).

Second element provides text explanation.

Return type two-element list

writeLogSelf ()

Generates yaml string containing dict to be written to log file.

Returns **logyaml** – log in yaml string format

Return type String

9.4.3 polychemprint3.tools.toolSpec module

Contains toolSpec Abstract Base Class.

First created on Sun Oct 20 00:03:21 2019

Revised: 10/17/2020

Author: Bijal Patel

class polychemprint3.tools.toolSpec.**toolSpec** (*name, units, __verbose__, **kwargs*)

Bases: *polychemprint3.utility.loggerSpec.loggerSpec*, *abc.ABC*

Abstract Base Class for all dispensing/writing tool drivers.

activate ()

To be called in main.py to load as active tool. Makes required serial connections and returns status as True/False.

Returns True if tool serial connection made and tool is ready to use False if error generated and tool is not ready for use

Return type bool

deactivate ()

To be called in main.py to unload as active tool. Closes serial communication and returns status as True/False.

Returns True if tool serial connection destroyed and tool disabled. False if error generated and serial communication not suspended.

Return type bool

disengage ()

Turn tool primary action off (stops dispense/LASER beam off, etc).

Returns **status** – First element (int) indicates whether disengage was successful (1), already off (0), or error (-1). Second element (String) provides text explanation.

Return type two-element list

engage ()

Turn tool primary action on (dispense/LASER beam on, etc).

Returns status – First element (int) indicates whether engage was successful (1), already on (0) or error (-1) Second element (String) provides text explanation.

Return type two-element list

getState ()

Returns the current dispense/action state (on/off).

Returns status – First element indicates whether tool is on(1), off(0) or error(-1). Second element provides text explanation.

Return type two-element list

loadLogSelf (yamlString)

Loads yaml log back into __dict__.

Parameters yamlString (String) – yaml string to be loaded back in

setValue (value)

Set the primary tool action value (e.g., Laser power, extruder pressure, etc.).

Parameters value (String) – The new value of the parameter as a string, expressed at arbitrary precision/ without leading zeros. Conversion to hardware specific format occurs internally. e.g., (use 23.456 NOT 0234”)

Returns status – First element (int) indicates whether value setting was successful (1) or error (-1). Second element provides text explanation.

Return type two-element list

writeLogSelf ()

Generates yaml string containing __dict__ to be written to log file.

Returns log in yaml string format

Return type String

9.4.4 polychemprint3.tools.ultimusExtruder module

9.5 polychemprint3.recipes

9.5.1 polychemprint3.recipes.recipe module

Specifies modular recipe protocol to link series of sequences

First created on Mon May 11 17:27:00 2020

Revised:

Author: Bijal Patel

```

class polychemprint3.recipes.recipe.recipe (name: str = 'NoRecipeNameSet', descrip-
tion: str = 'NoRecipeDescriptionSet',
dateCreated: str = 'NoDateSet', axes: <mod-
ule 'polychemprint3.axes.axes3DSpec' from
'/home/docs/checkouts/readthedocs.org/user_builds/polychemprint3/ch
= <polychemprint3.axes.nullAxes.nullAxes
object>, tool: poly-
chemprint3.tools.toolSpec.toolSpec = <poly-
chemprint3.tools.nullTool.nullTool object>,
seqList=None, __verbose__: bool = 0,
**kwargs)

```

Bases: `polychemprint3.utility.fileHandler.fileHandler`, `polychemprint3.utility.loggerSpec.loggerSpec`

Class for recipes - a series of sequences joined together

addSeq (beforeIndex: int, newSeq: polychemprint3.sequence.sequenceSpec.sequenceSpec)

Adds a copy of the provided sequence to the seqList. :param beforeIndex: :type beforeIndex: int :param newSeq: :type newSeq: sequenceSpec

deleteSeq (index: int)

Adds a copy of the provided sequence to the seqList. :param index: :type index: int

genRecipe ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (String) – log string to be loaded back in

operateRecipe (axesIn, toolIn)

Performs print sequence. :returns: Whether recipe successfully completed or not :rtype: bool

reorderSeq (currentIndex: int, newIndex: int)

Moves sequence from currentIndex to newIndex in seqList. :param currentIndex: :type currentIndex: int :param newIndex: :type newIndex: int

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

```

class polychemprint3.recipes.recipe.recipeStub (name: str = 'NoRecipeNameSet', de-
scription: str = 'NoRecipeDescription-
Set', dateCreated: str = 'NoDateSet',
**kwargs)

```

Bases: `polychemprint3.utility.fileHandler.fileHandler`

Class for recipe stubs, just the name, description, path info

9.6 polychemprint3.sequence

9.6.1 polychemprint3.sequence.sequenceSpec module

Specifies modular pre-written motion and dispense sequences for common prints.

First created on Sun Oct 20 00:08:15 2019

Revised (dd/mm/yyyy): 01/18/2021 - BP

Author: Bijal Patel

```
class polychemprint3.sequence.sequenceSpec.seqParam(name, value, unit, helpString)
```

Bases: object

Base Class for parameters used in sequences.

```
class polychemprint3.sequence.sequenceSpec.sequenceSpec(axes:                poly-  
chemprint3.axes.axes3DSpec.Axes3DSpec  
=                                <poly-  
chemprint3.axes.nullAxes.nullAxes  
object>, tool: poly-  
chemprint3.tools.toolSpec.toolSpec  
=                                <poly-  
chemprint3.tools.nullTool.nullTool  
object>, dictParams: dict  
= None, __verbose__:  
bool = 0, tool2: poly-  
chemprint3.tools.toolSpec.toolSpec  
=                                <poly-  
chemprint3.tools.nullTool.nullTool  
object>, tool3: poly-  
chemprint3.tools.toolSpec.toolSpec  
=                                <poly-  
chemprint3.tools.nullTool.nullTool  
object>, **kwargs)
```

Bases: `polychemprint3.utility.loggerSpec.loggerSpec`, `abc.ABC`

Abstract Base Class for predefined print sequences.

```
genSequence ()
```

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

```
loadLogSelf (logString)
```

loads log back into dict.

Parameters `logString` (*String*) – log string to be loaded back in

```
operateSeq (**kwargs)
```

Performs print sequence. :returns: Whether sequence successfully completed or not :rtype: bool

```
updateParams ()
```


writeLogSelf()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.2 polychemprint3.sequence.basicMove module

Moves Axes a set distance in X,Y,Z at set speed

First created on 05/05/2020

Revised: 10/8/2020

Author: Bijal Patel

```
class polychemprint3.sequence.basicMove.basicMove (axes:      <module      'poly-
chemprint3.axes.axes3DSpec' from
'/home/docs/checkouts/readthedocs.org/user_builds/polychemprint3/axes.axes3DSpec'
=      <poly-
chemprint3.axes.nullAxes.nullAxes
object>,      tool:      poly-
chemprint3.tools.toolSpec.toolSpec
=      <poly-
chemprint3.tools.nullTool.nullTool
object>, **kwargs)
```

Bases: `polychemprint3.sequence.sequenceSpec.sequenceSpec`

Sequence for a basic translation in a given direction

genSequence()

Loads print sequence into a list into cmdList attribute.

Returns Whether Command Generation Sequence reaches the end or not.

Return type bool

loadLogSelf(logString)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.3 polychemprint3.sequence.pause module

Sequence for introducing a pause. The length of time can be set, or it can resume on user input.

First created (dd/mm/yyyy): 05/05/2020

Revised (dd/mm/yyyy): 17/12/2020 - BP

Author: Bijal Patel

```
class polychemprint3.sequence.pause.pause (axes: polychemprint3.axes.axes3DSpec.Axes3DSpec
                                         = <polychemprint3.axes.nullAxes.nullAxes
                                         object>, tool: poly-
                                         chemprint3.tools.toolSpec.toolSpec = <poly-
                                         chemprint3.tools.nullTool.nullTool object>,
                                         **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Sequence for introducing a pause. The length of time can be set, or it can resume on user input.

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.4 polychemprint3.sequence.setToolState module

Sequence for changing tool value or dispense state.

First created (dd/mm/yyyy): 05/05/2020

Revised (dd/mm/yyyy): 17/12/2020 - BP

Author: Bijal Patel

```
class polychemprint3.sequence.setToolState.setToolState (axes: poly-
                                                         chemprint3.axes.axes3DSpec.Axes3DSpec
                                                         = <poly-
                                                         chemprint3.axes.nullAxes.nullAxes
                                                         object>, tool: poly-
                                                         chemprint3.tools.toolSpec.toolSpec
                                                         = <poly-
                                                         chemprint3.tools.nullTool.nullTool
                                                         object>, **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Sequence for changing tool value or dispense state.

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (*logString*)
loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()
Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.5 polychemprint3.sequence.gapLine module

Predefined print sequence for gapLines.

First created on 13/11/2019 14:41:31

Revised: 5/3/20

Author: Bijal Patel

```
class polychemprint3.sequence.gapLine.gapLine (axes:          <module          'poly-
chemprint3.axes.axes3DSpec'          from
'/home/docs/checkouts/readthedocs.org/user_builds/polychemprint3/
=          <poly-
chemprint3.axes.nullAxes.nullAxes
object>,          tool:          poly-
chemprint3.tools.toolSpec.toolSpec          =
<polychemprint3.tools.nullTool.nullTool
object>, **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Implemented print sequence for gapLines.

genSequence ()
Generates the list of python commands to execute for this sequence (shape).

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (*logString*)
loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()
Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.6 polychemprint3.sequence.line module

While dispensing, moves axes a set distance in X,Y,Z at set speed

First created on 13/11/2019 14:41:31
Revised (dd/mm/yyyy): 17/12/2020 - BP
Author: Bijal Patel

```
class polychemprint3.sequence.line.line (axes: polychemprint3.axes.axes3DSpec.Axes3DSpec
                                         = <polychemprint3.axes.nullAxes.nullAxes
                                         object>, tool: poly-
                                         chemprint3.tools.toolSpec.toolSpec = <poly-
                                         chemprint3.tools.nullTool.nullTool
                                         object>,
                                         **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Implemented print sequence for single lines.

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.7 polychemprint3.sequence.circle module

9.6.8 polychemprint3.sequence.cuboid module

3D Cuboid with base along the XY axes

First created (dd/mm/yyyy): 05/06/2020
Revised (dd/mm/yyyy): 17/12/2020 - BP
Author: Yilong Chang

```
class polychemprint3.sequence.cuboid.cuboid (axes: poly-
                                              chemprint3.axes.axes3DSpec.Axes3DSpec =
                                              <polychemprint3.axes.nullAxes.nullAxes
                                              object>, tool: poly-
                                              chemprint3.tools.toolSpec.toolSpec =
                                              <polychemprint3.tools.nullTool.nullTool
                                              object>, **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

3D Cuboid with base along the XY axes

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.9 polychemprint3.sequence.pyramid module

3D Pyramid with base along XY axes.

First created (dd/mm/yyyy): 05/06/2020

Revised (dd/mm/yyyy): 17/12/2020 - BP

Author: Yilong Chang

```
class polychemprint3.sequence.pyramid.pyramid (axes: poly-
chemprint3.axes.axes3DSpec.Axes3DSpec
= <poly-
chemprint3.axes.nullAxes.nullAxes
object>, tool: poly-
chemprint3.tools.toolSpec.toolSpec =
<polychemprint3.tools.nullTool.nullTool
object>, **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Implemented print sequence for circle.

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.10 polychemprint3.sequence.rectangle module

2D Rectangle along the XY axes

First created (dd/mm/yyyy): 05/06/2020

Revised (dd/mm/yyyy): 17/12/2020 - BP

Author: Bijal Patel

Author: Yilong Chang

```
class polychemprint3.sequence.rectangle.rectangle (axes:                poly-
                                                    chemprint3.axes.axes3DSpec.Axes3DSpec
                                                    =                <poly-
                                                    chemprint3.axes.nullAxes.nullAxes
                                                    object>,          tool:                poly-
                                                    chemprint3.tools.toolSpec.toolSpec
                                                    =                <poly-
                                                    chemprint3.tools.nullTool.nullTool
                                                    object>, **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Implemented print sequence for rectangle.

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.11 polychemprint3.sequence.triangle module

9.6.12 polychemprint3.sequence.plate module

Predefined print sequence for plates.

First created on 13/11/2019 14:41:31

Revised:

Author: Bijal Patel

```
class polychemprint3.sequence.plate.plate (axes: polychemprint3.axes.axes3DSpec.Axes3DSpec
                                         = <polychemprint3.axes.nullAxes.nullAxes
                                         object>, tool: poly-
                                         chemprint3.tools.toolSpec.toolSpec = <poly-
                                         chemprint3.tools.nullTool.nullTool
                                         object>,
                                         **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Implemented print sequence for plates.

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.13 polychemprint3.sequence.GCodeFileInkscape module

Parameterized code for reading in a gcode file and reprocessing for PCP3

First created on 2020/05/14 18:16:00

Revised: 2020/12/17

Author: Bijal Patel

```
class polychemprint3.sequence.GCodeFileInkscape.GCodeFileInkscape (axes: poly-
                                                                    chemprint3.axes.axes3DSpec.Axes3D
                                                                    = <poly-
                                                                    chemprint3.axes.nullAxes.nullAxes
                                                                    object>,
                                                                    tool: poly-
                                                                    chemprint3.tools.toolSpec.toolSpec
                                                                    = <poly-
                                                                    chemprint3.tools.nullTool.nullTool
                                                                    object>,
                                                                    **kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Sequence template for importing GCODE motion commands and tool triggers into PCP Recipe framework

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns Whether successfully reached the end or not.

Return type bool

importFromFile()

Attempts to read line by line from GcodeFile at GcodeFilePath and return the read lines as a list.

Returns

- *bool* – True if read from file without an error.
- *list of str* – A list containing each line read in as a separate str element.

insertToolCode (*procGlines*)

Augments procGlines with tool on/off/trv values based on the z-carriage height.

Returns **fullLines** – The combined list of Gcode and tool commands.

Return type list of str

loadLogSelf (*logString*)

loads log back into dict.

Parameters **logString** (*String*) – log string to be loaded back in

processGCode (*Glines*)

Parses each line in Glines to remove unusable commands and reconstitutes motion, feed strings with the rates the user provides in the CLI.

Returns **procGlines** – A list of GCode lines processed to remove garbage and include user-specified feeds, z height.

Return type list of str

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.6.14 polychemprint3.sequence.GCodeFile3DSlicer module

Parameterized code for reading in a gcode file for 3D printing and reprocessing for PCP3

First created on 05/14/2020 18:16:00

Revised: 11/11/21

Author: Bijal Patel

```
class polychemprint3.sequence.GCodeFile3DSlicer.GCodeFile3DSlicer (axes: poly-  
chemprint3.axes.axes3DSpec.Axes3D  
= <poly-  
chemprint3.axes.nullAxes.nullAxes  
object>,  
tool: poly-  
chemprint3.tools.toolSpec.toolSpec  
= <poly-  
chemprint3.tools.nullTool.nullTool  
object>,  
**kwargs)
```

Bases: *polychemprint3.sequence.sequenceSpec.sequenceSpec*

Sequence template for importing 3D GCODE motion commands and tool triggers into PCP Recipe framework

genSequence ()

Loads print sequence into a list into cmdList attribute.

Returns whether successfully reached the end or not

Return type bool

importFromFile ()

Attempts to read line by line from GcodeFile at GCodeFilePath into memory

insertToolCode (procGlines)

loadLogSelf (logString)

loads log back into dict.

Parameters **logString** (String) – log string to be loaded back in

processGCode (GLines)

Attempts to filter line by line from GLines to remove garbage and substitute values

writeLogSelf ()

Generates log string containing dict to be written to log file.

Returns log in string format

Return type String

9.7 polychemprint3.utility

9.7.1 polychemprint3.utility.fileHandler module

Specifies interface for classes that will handle rw 'data' files.

First created on Sun Oct 20 00:03:21 2019

Revised: 20/10/2019 00:34:27

Author: Bijal Patel

class polychemprint3.utility.fileHandler.**fileHandler** (fullFilePath=None, **kwargs)

Bases: object

Class for objects that can read/write to file

appendToFile (outString)

Appends to file with new content from outString.

Parameters **outString** (String) – the string to write to the file

Returns True/False if writing passes/fails + errors

Return type bool

overwriteToFile (outString)

Completely overwrites file with new content from outString.

Parameters **outString** (String) – the string to write to the file

Returns True/False if writing passes/fails + errors

Return type bool

peekFile (*numLines*)

Reads numLines from file and returns.

Parameters **numLines** (*int*) – number of lines to read in from file

Returns

- *bool* – True/ False if read successful
- [*lines*] – array of strings read in or [“Failed”]

readFullFile ()

Reads the entire file into memory as a list of strings.

Returns

- *bool* – True/False if read passes/fails + errors
- [*lines*] – array of strings read in or [“Failed”]

testFileIO (*modeString*)

Tests if file can be opened and closed.

Parameters **modeString** (*String*) – mode with which to open file (“r,w,r+,a”)

Returns True/False if test passes/fails + errors

Return type bool

9.7.2 polychemprint3.utility.loggerSpec module

Specifies interface for all classes to read/write themselves to string.

First created on Sun Oct 20 00:03:21 2019

Revised: 20/10/2019 00:34:27

Author: Bijal Patel

class polychemprint3.utility.loggerSpec.**loggerSpec** (***kwargs*)

Bases: abc.ABC

Abstract Base Class for objects that can generate log strings.

loadLogSelf (*yamlString*)

loads yaml log back into dict.

Parameters **yamlString** (*String*) – yaml string to be loaded back in

writeLogSelf ()

Generates yaml string containing dict to be written to log file.

Returns log in yaml string format

Return type String

9.7.3 polychemprint3.utility.serialDeviceSpec module

Interface for all Serial Device objects (extruders/lasers/axes/etc).

First created on Sun Oct 20 00:03:21 2019

Revised: 20/10/2019 00:34:27

Author: Bijal Patel

```
class polychemprint3.utility.serialDeviceSpec.serialDeviceSpec (devAddress,
                                                                baudRate,
                                                                commsTimeOut,
                                                                **kwargs)
```

Bases: abc.ABC

Abstract Base Class for all objects using serial device.

checkIfSerialConnectParamsSet ()

Goes through connection parameters and sees if all are set.

Returns True if all parameters are set, false if any unset

Return type bool

handShakeSerial ()

Perform communications handshake with serial device.

Returns

- [1, "Handshake Successful"] – success occurred
- [0, 'Handshake Failed, Rcvd + message received'] – failure occurred
- [-1, "Error (Handshake with Tool Failed + error text)"] – Error received

readTime ()

Reads in from serial device until timeout.

Returns All text read in, empty string if nothing

Return type String

startSerial ()

Creates pySerial device.

Returns

- [1, "Terminated successfully"] – started successfully
- [-1, "Error (error text)"] – could not start

stopSerial ()

Terminates communication.

Returns

- [1, "Terminated successfully"] – started successfully
- [-1, "Error (Serial Device could not be stopped + error text)"] – could not start

p

- `polychemprint3.axes.axes3DSpec`, 49
- `polychemprint3.axes.lulzbotTaz6_BP`, 50
- `polychemprint3.axes.nullAxes`, 52
- `polychemprint3.commandLineInterface.ioElementSpec`, 47
- `polychemprint3.commandLineInterface.ioMenuSpec`, 47
- `polychemprint3.commandLineInterface.ioTextPanel`, 48
- `polychemprint3.data`, 49
- `polychemprint3.data.TextPanels`, 49
- `polychemprint3.recipes.recipe`, 58
- `polychemprint3.sequence.basicMove`, 61
- `polychemprint3.sequence.cuboid`, 64
- `polychemprint3.sequence.gapLine`, 63
- `polychemprint3.sequence.GCodeFile3DSlicer`, 68
- `polychemprint3.sequence.GCodeFileInkscape`, 67
- `polychemprint3.sequence.line`, 63
- `polychemprint3.sequence.pause`, 61
- `polychemprint3.sequence.plate`, 66
- `polychemprint3.sequence.pyramid`, 65
- `polychemprint3.sequence.rectangle`, 66
- `polychemprint3.sequence.sequenceSpec`, 60
- `polychemprint3.sequence.setToolState`, 62
- `polychemprint3.tools.laser6W`, 53
- `polychemprint3.tools.nullTool`, 55
- `polychemprint3.tools.toolSpec`, 57
- `polychemprint3.utility.fileHandler`, 69
- `polychemprint3.utility.loggerSpec`, 70
- `polychemprint3.utility.serialDeviceSpec`, 71

A

activate() (*polychemprint3.axes.axes3DSpec.Axes3DSpec* method), 49

activate() (*polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP* method), 50

activate() (*polychemprint3.axes.nullAxes.nullAxes* method), 52

activate() (*polychemprint3.tools.laser6W.laser6W* method), 53

activate() (*polychemprint3.tools.nullTool.nullTool* method), 56

activate() (*polychemprint3.tools.toolSpec.toolSpec* method), 57

addSeq() (*polychemprint3.recipes.recipe.recipe* method), 59

appendToFile() (*polychemprint3.utility.fileHandler.fileHandler* method), 69

Axes3DSpec (class in *polychemprint3.axes.axes3DSpec*), 49

B

basicMove (class in *polychemprint3.sequence.basicMove*), 61

C

checkIfSerialConnectParamsSet() (*polychemprint3.tools.laser6W.laser6W* method), 54

checkIfSerialConnectParamsSet() (*polychemprint3.utility.serialDeviceSpec.serialDeviceSpec* method), 71

cuboid (class in *polychemprint3.sequence.cuboid*), 64

D

deactivate() (*polychemprint3.axes.axes3DSpec.Axes3DSpec* method), 49

deactivate() (*polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP* method), 50

deactivate() (*polychemprint3.axes.nullAxes.nullAxes* method), 52

deactivate() (*polychemprint3.tools.laser6W.laser6W* method), 54

deactivate() (*polychemprint3.tools.nullTool.nullTool* method), 56

deactivate() (*polychemprint3.tools.toolSpec.toolSpec* method), 57

deleteSeq() (*polychemprint3.recipes.recipe.recipe* method), 59

disengage() (*polychemprint3.tools.laser6W.laser6W* method), 54

disengage() (*polychemprint3.tools.nullTool.nullTool* method), 56

disengage() (*polychemprint3.tools.toolSpec.toolSpec* method), 57

E

engage() (*polychemprint3.tools.laser6W.laser6W* method), 54

engage() (*polychemprint3.tools.nullTool.nullTool* method), 56

engage() (*polychemprint3.tools.toolSpec.toolSpec* method), 57

F

fileHandler (class in *polychemprint3.utility.fileHandler*), 69

G

gapLine (class in *polychemprint3.sequence.gapLine*), 63

GCodeFile3DSlicer (class in polychemprint3.sequence.GCodeFile3DSlicer), 68

GCodeFileInkscape (class in polychemprint3.sequence.GCodeFileInkscape), 67

genRecipe () (polychemprint3.recipes.recipe.recipe method), 59

genSequence () (polychemprint3.sequence.basicMove.basicMove method), 61

genSequence () (polychemprint3.sequence.cuboid.cuboid method), 64

genSequence () (polychemprint3.sequence.gapLine.gapLine method), 63

genSequence () (polychemprint3.sequence.GCodeFile3DSlicer.GCodeFile3DSlicer method), 69

genSequence () (polychemprint3.sequence.GCodeFileInkscape.GCodeFileInkscape method), 67

genSequence () (polychemprint3.sequence.line.line method), 64

genSequence () (polychemprint3.sequence.pause.pause method), 62

genSequence () (polychemprint3.sequence.plate.plate method), 67

genSequence () (polychemprint3.sequence.pyramid.pyramid method), 65

genSequence () (polychemprint3.sequence.rectangle.rectangle method), 66

genSequence () (polychemprint3.sequence.sequenceSpec.sequenceSpec method), 60

genSequence () (polychemprint3.sequence.setToolState.setToolState method), 62

getAbsPosXY () (polychemprint3.axes.axes3DSpec.Axes3DSpec method), 49

getAbsPosXY () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 50

getAbsPosXY () (polychemprint3.axes.nullAxes.nullAxes method), 52

getState () (polychemprint3.tools.laser6W.laser6W method), 54

getState () (polychemprint3.tools.nullTool.nullTool method), 56

getState () (polychemprint3.tools.toolSpec.toolSpec method), 58

H

handShakeSerial () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51

handShakeSerial () (polychemprint3.tools.laser6W.laser6W method), 54

handShakeSerial () (polychemprint3.utility.serialDeviceSpec.serialDeviceSpec method), 71

I

importFromGFile () (polychemprint3.sequence.GCodeFile3DSlicer.GCodeFile3DSlicer method), 69

importFromGFile () (polychemprint3.sequence.GCodeFileInkscape.GCodeFileInkscape method), 68

insertToolCode () (polychemprint3.sequence.GCodeFile3DSlicer.GCodeFile3DSlicer method), 69

insertToolCode () (polychemprint3.sequence.GCodeFileInkscape.GCodeFileInkscape method), 68

io_Operate () (polychemprint3.commandLineInterface.ioElementSpec.ioElementSpec method), 47

io_Operate () (polychemprint3.commandLineInterface.ioMenuSpec.ioMenuSpec method), 48

io_Operate () (polychemprint3.commandLineInterface.ioTextPanel.ioTextPanel method), 48

ioElementSpec (class in polychemprint3.commandLineInterface.ioElementSpec), 47

ioMenu_printMenu () (polychemprint3.commandLineInterface.ioMenuSpec.ioMenuSpec method), 48

ioMenu_updateStoredCmds () (polychemprint3.commandLineInterface.ioMenuSpec.ioMenuSpec method), 48

ioMenuSpec (class in polychemprint3.commandLineInterface.ioMenuSpec), 48

ioTextPanel (class in polychemprint3.commandLineInterface.ioTextPanel), 48

L

laser6W (class in polychemprint3.tools.laser6W), 53
 line (class in polychemprint3.sequence.line), 64
 loadLogSelf () (polychemprint3.recipes.recipe.recipe method), 59
 loadLogSelf () (polychemprint3.sequence.basicMove.basicMove method), 61
 loadLogSelf () (polychemprint3.sequence.cuboid.cuboid method), 65
 loadLogSelf () (polychemprint3.sequence.gapLine.gapLine method), 63
 loadLogSelf () (polychemprint3.sequence.GCodeFile3DSlicer.GCodeFile3DSlicer method), 69
 loadLogSelf () (polychemprint3.sequence.GCodeFileInkscape.GCodeFileInkscape method), 68
 loadLogSelf () (polychemprint3.sequence.line.line method), 64
 loadLogSelf () (polychemprint3.sequence.pause.pause method), 62
 loadLogSelf () (polychemprint3.sequence.plate.plate method), 67
 loadLogSelf () (polychemprint3.sequence.pyramid.pyramid method), 65
 loadLogSelf () (polychemprint3.sequence.rectangle.rectangle method), 66
 loadLogSelf () (polychemprint3.sequence.sequenceSpec.sequenceSpec method), 60
 loadLogSelf () (polychemprint3.sequence.setToolState.setToolState method), 63
 loadLogSelf () (polychemprint3.tools.laser6W.laser6W method), 55
 loadLogSelf () (polychemprint3.tools.nullTool.nullTool method), 56
 loadLogSelf () (polychemprint3.tools.toolSpec.toolSpec method), 58
 loadLogSelf () (polychemprint3.utility.loggerSpec.loggerSpec method), 70
 loggerSpec (class in polychemprint3.utility.loggerSpec), 70

lulzbotTaz6_BP (class in polychemprint3.axes.lulzbotTaz6_BP), 50

M

move () (polychemprint3.axes.axes3DSpec.Axes3DSpec method), 49
 move () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 move () (polychemprint3.axes.nullAxes.nullAxes method), 53

N

nullAxes (class in polychemprint3.axes.nullAxes), 52
 nullTool (class in polychemprint3.tools.nullTool), 56

O

openRecipe () (polychemprint3.recipes.recipe.recipe method), 59
 openSeq () (polychemprint3.sequence.sequenceSpec.sequenceSpec method), 60
 overWriteToFile () (polychemprint3.utility.fileHandler.fileHandler method), 69

P

pause (class in polychemprint3.sequence.pause), 62
 peekFile () (polychemprint3.utility.fileHandler.fileHandler method), 70
 plate (class in polychemprint3.sequence.plate), 66
 poll () (polychemprint3.axes.axes3DSpec.Axes3DSpec method), 49
 poll () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 poll () (polychemprint3.axes.nullAxes.nullAxes method), 53
 polychemprint3.axes.axes3DSpec (module), 49
 polychemprint3.axes.lulzbotTaz6_BP (module), 50
 polychemprint3.axes.nullAxes (module), 52
 polychemprint3.commandLineInterface.ioElementSpec (module), 47
 polychemprint3.commandLineInterface.ioMenuSpec (module), 47
 polychemprint3.commandLineInterface.ioTextPanel (module), 48
 polychemprint3.data (module), 49
 polychemprint3.data.TextPanels (module), 49
 polychemprint3.recipes.recipe (module), 58
 polychemprint3.sequence.basicMove (module), 61

polychemprint3.sequence.cuboid (module), 64
 polychemprint3.sequence.gapLine (module), 63
 polychemprint3.sequence.GCodeFile3DSlicer (module), 68
 polychemprint3.sequence.GCodeFileInkscape (module), 67
 polychemprint3.sequence.line (module), 63
 polychemprint3.sequence.pause (module), 61
 polychemprint3.sequence.plate (module), 66
 polychemprint3.sequence.pyramid (module), 65
 polychemprint3.sequence.rectangle (module), 66
 polychemprint3.sequence.sequenceSpec (module), 60
 polychemprint3.sequence.setToolState (module), 62
 polychemprint3.tools.laser6W (module), 53
 polychemprint3.tools.nullTool (module), 55
 polychemprint3.tools.toolSpec (module), 57
 polychemprint3.utility.fileHandler (module), 69
 polychemprint3.utility.loggerSpec (module), 70
 polychemprint3.utility.serialDeviceSpec (module), 71
 processGCode () (polychemprint3.sequence.GCodeFile3DSlicer.GCodeFile3DSlicer method), 69
 processGCode () (polychemprint3.sequence.GCodeFileInkscape.GCodeFileInkscape method), 68
 pyramid (class in polychemprint3.sequence.pyramid), 65

R

readFullFile () (polychemprint3.utility.fileHandler.fileHandler method), 70
 readTime () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 readTime () (polychemprint3.tools.laser6W.laser6W method), 55
 readTime () (polychemprint3.utility.serialDeviceSpec.serialDeviceSpec method), 71
 recipe (class in polychemprint3.recipes.recipe), 58
 recipeStub (class in polychemprint3.recipes.recipe), 59
 rectangle (class in polychemprint3.sequence.rectangle), 66
 reorderSeq () (polychemprint3.recipes.recipe.recipe method), 59

S

sendCmd () (polychemprint3.axes.axes3DSpec.Axes3DSpec method), 50
 sendCmd () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 sendCmd () (polychemprint3.axes.nullAxes.nullAxes method), 53
 seqParam (class in polychemprint3.sequence.sequenceSpec), 60
 sequenceSpec (class in polychemprint3.sequence.sequenceSpec), 60
 serialDeviceSpec (class in polychemprint3.utility.serialDeviceSpec), 71
 setPosMode () (polychemprint3.axes.axes3DSpec.Axes3DSpec method), 50
 setPosMode () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 setPosMode () (polychemprint3.axes.nullAxes.nullAxes method), 53
 setPosZero () (polychemprint3.axes.axes3DSpec.Axes3DSpec method), 50
 setPosZero () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 setPosZero () (polychemprint3.axes.nullAxes.nullAxes method), 53
 setToolState (class in polychemprint3.sequence.setToolState), 62
 setValue () (polychemprint3.tools.laser6W.laser6W method), 55
 setValue () (polychemprint3.tools.nullTool.nullTool method), 56
 setValue () (polychemprint3.tools.toolSpec.toolSpec method), 58
 startSerial () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 startSerial () (polychemprint3.tools.laser6W.laser6W method), 55
 startSerial () (polychemprint3.utility.serialDeviceSpec.serialDeviceSpec method), 71
 stopSerial () (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 51
 stopSerial () (polychemprint3.tools.laser6W.laser6W method), 55

stopSerial() (poly-chemprint3.utility.serialDeviceSpec.serialDeviceSpec method), 71

T

testFileIO() (poly-chemprint3.utility.fileHandler.fileHandler method), 70

toolSpec (class in polychemprint3.tools.toolSpec), 57

U

updateParams() (poly-chemprint3.sequence.sequenceSpec.sequenceSpec method), 60

W

waitReady() (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 52

writeLogSelf() (poly-chemprint3.recipes.recipe.recipe method), 59

writeLogSelf() (poly-chemprint3.sequence.basicMove.basicMove method), 61

writeLogSelf() (poly-chemprint3.sequence.cuboid.cuboid method), 65

writeLogSelf() (poly-chemprint3.sequence.gapLine.gapLine method), 63

writeLogSelf() (poly-chemprint3.sequence.GCodeFile3DSlicer.GCodeFile3DSlicer method), 69

writeLogSelf() (poly-chemprint3.sequence.GCodeFileInkscape.GCodeFileInkscape method), 68

writeLogSelf() (polychemprint3.sequence.line.line method), 64

writeLogSelf() (poly-chemprint3.sequence.pause.pause method), 62

writeLogSelf() (poly-chemprint3.sequence.plate.plate method), 67

writeLogSelf() (poly-chemprint3.sequence.pyramid.pyramid method), 65

writeLogSelf() (poly-chemprint3.sequence.rectangle.rectangle method), 66

writeLogSelf() (poly-chemprint3.sequence.sequenceSpec.sequenceSpec method), 60

writeLogSelf() (poly-chemprint3.sequence.setToolState.setToolState method), 63

writeLogSelf() (poly-chemprint3.tools.laser6W.laser6W method), 55

writeLogSelf() (poly-chemprint3.tools.nullTool.nullTool method), 57

writeLogSelf() (poly-chemprint3.tools.toolSpec.toolSpec method), 58

writeLogSelf() (poly-chemprint3.utility.loggerSpec.loggerSpec method), 70

writeReady() (polychemprint3.axes.lulzbotTaz6_BP.lulzbotTaz6_BP method), 52