

makePCB.py

Version: 1.1.0

Date: March 8th 2017

Author: Brian N Norman

Contents

1	Purpose	4
2	Background	4
3	License.....	4
4	Warranty	5
5	Where to find it.....	5
6	How it works	5
6.1	Standard Templates	5
6.2	Template Placeholders.....	6
7	The Position Data File	7
7.1	ASIS.....	8
7.2	CLRG/CLRL.....	8
7.3	DEFGROUP/ENDGROUP	8
7.4	DEFREPEAT/ENDREPEAT	9
7.5	DEFRING/ENDRING	9
7.6	ECHO	9
7.7	GROUP.....	11
7.8	KICADPCB	11
7.9	LIST/USELIST/ENDLIST.....	11
7.9.1	USELIST.....	12
7.9.2	ENDUSELIST.....	12
7.10	ORIGIN.....	13
7.11	REPEAT	13
7.12	RING	14
7.13	SAVEXY	15
7.14	SAVEONCEXY.....	16
7.15	SETDIRECTION.....	16
7.16	SHOWVARS	16
7.17	STOP	16
7.18	SETG/SETL	16
7.19	TEMPLATE	17

7.19.1	FIDUCIAL.....	17
7.19.2	GRAPHIC.....	18
7.19.3	KEEPOUT	19
7.19.4	MOUNT	19
7.19.5	SEGMENT or TRACK	19
7.19.6	TARGET.....	21
7.19.7	VIA.....	21
7.19.8	ZONE	21
8	Variables.....	25
8.1	Global variables.....	25
8.2	Local Variables	26
8.3	Using Variables and Expressions.....	26
9	Examples	26
10	Using the program	27
11	Error Messages.....	27
11.1	evalParm exception.....	27
11.1.1	evalNumericParam exception ("could not convert string to float: 'variable name'",)	28
11.1.2	evalNumericParam exception ('could not convert string to float: ',)	28
11.2	Missing or Extra parameters	28
11.2.1	ERROR at data file line NN error= ('not enough values to unpack (expected 3, got 2)',)	28
11.2.2	ERROR at data file line NN error= ('too many values to unpack (expected 3)',).....	28
11.3	Capitalisation	28
11.4	WARNING at line XX No template loaded for ID=<item> check your .pos file.	28
11.5	Invalid numerical parameters	28
11.6	Cannot convert %x% to float (or int)	29
11.7	Warnings	29
11.8	Other errors	32
12	Modifying MakePCB.....	32
12.1	makePCB.py	32
12.2	KiCadParser.py	32

1 Purpose

In a nutshell, makePCB is a Python program to auto-generate a kicad_pcb file from a list of positional details to take the tedium out of placing a lot of similar components. Thereafter you can hand edit the layout using pcbnew.

The positional data is fed in through a CSV data file which includes commands to do nice stuff like draw circular/arc track segments and various shapes of zones/keepout areas.

It will allow you to create circular/rectangular component placements. e.g. LED rings/arrays.

NOTE: makePCB doesn't require KiCad to be open as it uses templates which contain placeholders. The templates were created from the text placed into a kicad_pcb file by pcbnew.

You CAN also use the makePCB program to add to an existing kicad_pcb file.

2 Background

I was asked if I could design an WS2812B LED based logo board. The prototype was hand soldered into a 3D printed layout – so I knew where each LED went from my openSCAD designs. There were 180 LEDs and I didn't fancy laying out a PCB by hand.

In case there is demand for the logo board I thought it was time to look at designing a PCB (or two) and was horrified at the thought of positioning them LEDs through a GUI when I already had the co-ordinates. After finding KiCad had text data files I set about writing an initial script to suck in the co-ordinates and place the LEDs. That still left me with a lot of tracks to manually insert and as one LED arrangement was a nested circular pattern with something like 60 LEDs – all needing rotating.

I didn't create a schematic for this because the job just required power rails and signals going from DOUT to DIN on the WS2812B LEDs.

But then I came across people wanting different shaped zones and so the project developed further and I got very interested in developing code that would do that.

Primarily this program is for those times when you have lots and lots of the same components to place and can calculate the relative positions.

The program isn't intended to produce the finished article – you will probably want to (have to) hand edit some things like where the references appear etc.

3 License

MakePCB is Freeware , my contribution to the KiCad community in appreciation of the work done on KiCad – especially because it will allow me to make any size PCB.

If you improve makePCB (especially my use of Python) I'd like a copy. If you have suggestions for improvement I'd like to know. If you love it, tell the world and tell them I wrote it and where they can find it.

Don't sell it as your own work without making significant alterations and sending me some money (I am living on a meagre pension).

4 Warranty

None.

Always backup valuable work before using the program to modify it, in case it breaks, which it will if you are cavalier with your parameter values. For example creating a rounded rectangle asking for a corner radius greater than either the width or height of the rectangle.

5 Where to find it

GitHub – search for BNNorman/KiCad-makePCB

Use GitHub to report any issues. Be aware that it is easy to break if, for instance, you don't provide ALL the required parameters. But that's no problem; you will get an error message, fix the fault in your .pos data file and run it again. It only takes a few seconds on my quad core 64bit desktop machine running Windows 10.

6 How it works

It is all driven by a position data file (filename.pos) and a group of standard templates which have standard Ids. You can add your own templates for the components you want to use.

6.1 Standard Templates

The standard templates were obtained by adding items to a new kicad_pcb document and cutting the relevant section out then replacing variable values with %..% place holders (uppercase names which are case sensitive). You can edit the templates to your liking to create default settings.

Id	Template file (can be a relative path)	Comment
xxxxxxxxx	templates\Header.txt	This doesn't have a set ID. You can import this template using ASIS (see later) and name it however you want. I call it HEADER. It MUST be output first as it contains all the pcbnew settings (open a kicad_pcb file in a text editor and you will see them at the beginning. You can create different headers for different jobs and load whichever you want.
SEGMENT	templates\segment.txt	Used for creating segments of track. SEGMENT has a number of shape options: Arc, Circle, Grid, Line
GRLINE	templates\GRline.txt	Used to draw a graphics line normally on the Edge.Cuts layer. You can put this on a copper layer but KiCad won't let you edit it as a track. It will appear in the Gerber file

		as a track though.
GRCIRCLE	templates\GRcircle.txt	Similar to GRLINE but a circle.
GRTEXT	templates\grtext.txt	Used place text.
VIA	templates\via.txt	Used to add a VIA hole between two layers
TARGET	templates\target.txt	Used to place a target on the pcb
MOUNT	templates\mounting.txt	Used to create a mounting hole
ZONE	template\zone.txt	Used to create a filled zone. Zone has a number of options:- Poly, Circle, Cross, Donut, HollowRect, Pie, Rect and RoundedRect. With Poly you provide the coordinate pairs.
KEEPOUT	template\keepout.txt	Same as a zone but designated as a keepout area
FIDUCIAL	template\fiducial.txt	Used to add fiducials

User defined templates (e.g. components) take the same form. For program testing I was using just three components as I am interested in building custom LED arrays. You can easily create your own component templates by adding a component to a new pcb then save the document and copy the relevant (module) section into a text file (or delve into the .pretty libraries - harder).

Id	Template file (can be a relative path)	Comment
WS2812B	component\WS2812B.txt	The LEDs I want to use
CAP	component\Capacitors SMD.txt	An SMD capacitor template I want to use
JST	component\JST.txt	A 4 pin JST connector used in the LEDRING demo in the Examples\PCB folder

6.2 Template Placeholders

When making your own templates take a peek at mine first – you’ll get the idea – it’s very simple. Use pcbnew to place a rotated component so you can find out where to put the %ANGLE% placeholders because they are not present in an un-rotated component. The component and all it’s pads can be rotated, so there are usually quite a few places where you need to add %ANGLE% placeholders.

The placeholders used in the templates are as follows:-

Place holder	Comment
%ANGLE%	Used where needed for component and all pad orientations.
%ARCSEGMENTS%	Used in zones to control the zone fill. The pcbnew dialog allows 16 or 32.
%CLEARANCE%	Clearance dimension used in, for example, Zones and Fiducials.
%DRILL%	Drill size - used by some templates such as VIA, MOUNT.
%HATCHEDGE%	Used in zones and is the distance between hatch lines.
%HATCHTYPE%	Used in Zones. Usually replaced with values like ‘edge’, ‘full’ or ‘line’ (no quotes).

%INNERWIDTH%	Used in mounting holes.
%JUSTIFY%	Used in graphic text can be left/right/center (or centre)
%LAYER%	Used by most components. Interestingly I found I could specify a copper layer for a graphics circle or line and when opened in KiCad the lines were coloured correctly. What's more, they appeared on the copper layers in the Gerber files. However, the KiCad GUI won't let you edit them without moving them to a non-copper layer. But hey, if it's ok leave it alone. If it isn't ok, edit the position data file to correct it and regenerate the pcb file – it only takes a few seconds on my quad core 64bit PC.
%LAYERF%, %LAYERB%	Used by VIAs to indicate the front/back layers involved.
%MINTHICKNESS%	Also used in zones
%NET%	Netlist rules (I think). I personally haven't used any yet.
%NETNAME%	Used in Zones. Can be left blank.
%OUTERRADIUS%	Used in mounting holes
%OUTERWIDTH%	Used in mounting holes
%REF%	Component reference, where required. ALL user components have a reference based on their ID. MakePCB tracks the last reference number on a per ID basis. So my capacitors are numbered from 1 onwards as are my LEDs but the text for the references begin with L or C when the components are added as follows:- WS128B,L,0,50,60.... ->L1, L2, L3 etc, CAP,C,0,55,60.... -> C1, C2, C3 etc
%SHAPE%	Used in targets, Shape is 'plus' or 'x' (without the quotes).
%SIZE%	Used by some components like fiducials and vias.
%SIZE%	Used in graphic text and is the height of the text.
%THERMALBRIDGEWIDTH%	Used in zones.
%THERMALGAP%	Used in zones.
%THICKNESS%	Used in graphic text and is the thickness of the text.
%WIDTH%	Usually line width - Used where needed, for example in SEGMENT it is the copper track width.
%XPOS%, %YPOS%	X,Y coordinates in component templates - where only one co-ordinate is required
%XPOS1%, %YPOS1% and %XPOS2%, %YPOS2%	The SEGMENT template uses these to define start and end of linear track sections.
%XYPOINTS%	Used in zones and keepouts to define the polyshape.

7 The Position Data File

The data file is simply a text CSV document which can be exported from a spreadsheet. Using a spreadsheet lets you position items easily using simple formulas to calculate X,Y values.

Any empty line in the position file or one beginning with a hash (#) will be ignored for the purposes of commenting.

There are a number of examples in the Examples folder.

The remainder of this section discusses the commands which you can place in the data file.

7.1 ASIS

Syntax: ASIS,ID,Filename

Loads an ASIS template which must not contain placeholders – hence the name ASIS (unchanged). The kicad_pcb file header is a typical use for this.

e.g. ASIS,HEADER,templates\Header.txt

This loads the Header.txt template and gives it the id HEADER. The header can then be placed in the output using just the word HEADER in the position data file.

The kicad_pcb header MUST precede any commands which generate output. If you want to add to an existing kicad_pcb file use the KICADPCB command instead.

If you wanted to split the HEADER further (look at the contents of a kicad_pcb file) you could load different nets etc. just by changing the Filename.

7.2 CLRG/CLRL

Syntax: CLRx,varname

E.g. CLRG, xpos or CLRL, ypos

This deletes the named global or local variable. This could have adverse effects if your data file commands still use them. See the comments in the Using Variables section.

7.3 DEFGROUP/ENDGROUP

These commands allow you to define a group of components and place them where you will (even rotated). My requirement was to group a WS2812B led and a 104nF capacitor so I could place them together. Here is an example (You will have to read on to understand the values but I have highlighted the XY co-ordinates in red):-

```
DEFGROUP,LED_CAP
WS2812B,L,0,0,0,F.Cu
CAP,C,0,2.25,94.5,F.Cu
ENDGROUP
```

The X,Y coordinates of the group items should be relative to 0,0. Usually one of the components will be at 0,0 and probably the first in the list. The WS2812B and CAP are my component template Ids. The WS2812B was my 0,0 choice because the position of the LEDs was more critical than the capacitor – they had to be visible through holes which matched the LED positions.

7.4 DEFREPEAT/ENDREPEAT

These commands allow you to define a group of components and place them where you will in a repeating rectangular or diagonal pattern. Arrays can easily be built up.

```
DEFREPEAT,ARRAY1
WS2812B,L1,0,0,0,F.Cu
CAP,C1,0,2.25,94.5,F.Cu
ENDGROUP
```

As with GROUP , the X,Y coords should be relative to 0,0. Usually one of the components will be at 0,0. The above defines a group of components to be repeated in an array, wherever you want.

7.5 DEFRING/ENDRING

These commands allow you to define a group of components and place them where you will in a circular (ring) pattern.

```
DEFRING,LedRing
WS2812B,L1,0,0,0,F.Cu
CAP,C1,0,2.25,94.5,F.Cu
ENDRING
```

As with GROUP, the X,Y coords should be relative to 0,0. Usually one of the components will be at 0,0. The above defines a group of components to be placed in a ring wherever you want. See RING for information on creating the ring.

7.6 ECHO

Syntax: ECHO,Id,X,Y

Outputs a message to the console showing the co-ordinates of X,Y adjusted for the current origin and X/Y directions. In plain English it tells you what the absolute XY values are of a point so you can use the information to create connections between items in a block using something like SEGMENT.

Be aware that SAVEXY and SAVEONCEXY can be used instead so that you don't actually need to use ECHO anymore (I just left it in, in case someone had a need for it.)

ECHO is most useful in a command block (group/repeat/ring) for obtaining the calculated values of X and Y.

X and Y are the group/repeat/ring current X,Y positions.

In the following example I have defined a group using a WS2812B LED, a 104nf CAP(actor) with some tracks and highlighted the ECHO commands. You can see I want the adjusted XY coordinates of the end of a two tracks (The Din and Dout signals for a WS2812B LED.) so I can use SEGMENT to daisy chain the signal track between the LEDs. You'll find the full data file and kicad_pos for the LED ring in the Examples\Ring folder

```
# LED_CAP group
```

```
#
# groups my LED and CAP together
# the coordinates are relative to each other
# signal and power tracks added
DEFGROUP,LED_CAP
# LED,ref,X,Y,Angle,Layer
WS2812B,L,0,0,0,F.Cu
# CAP,ref,X,Y,Angle,Layer
CAP,C,-90,6,0,F.Cu
# add tracks
# pin 1 5V
SEGMENT,LINE,-2.5,2,-2.5,5,1,F.Cu,0
# pin 2 DOUT
SEGMENT,LINE,-2.5,-2,-2.5,0,.1,F.Cu,0
SEGMENT,LINE,-2.5,0,-4.5,0,.1,F.Cu,0
ECHO,DOUT,-4.5,0
# pin 4 DIN
SEGMENT,LINE,2.5,2,2.5,0,.1,F.Cu,0
SEGMENT,LINE,2.5,0,7.5,0,.1,F.Cu,0
ECHO,DIN,7.5,0
# pin 3 GND
SEGMENT,LINE,2.5,-2,2.5,-5,1,F.Cu,0
# capacitor 5V
SEGMENT,LINE,6,2,6,5,1,F.Cu,0
# capacitor GND
SEGMENT,LINE,6,-2,6,-5,1,F.Cu,0
ENDGROUP
```

Here's an example, printed on the console, from my LED ring project .

```
ECHO Id= DOUT X,Y= 150.0000 , 104.5000
ECHO Id= DIN X,Y= 150.0000 , 92.5000
ECHO Id= DOUT X,Y= 137.8058 , 133.0298
ECHO Id= DIN X,Y= 144.8592 , 123.3216
ECHO Id= DOUT X,Y= 111.1711 , 148.9434
ECHO Id= DIN X,Y= 122.5838 , 145.2352
ECHO Id= DOUT X,Y= 80.2694 , 146.1622
ECHO Id= DIN X,Y= 91.6821 , 149.8705
ECHO Id= DOUT X,Y= 56.9041 , 125.7487
ECHO Id= DIN X,Y= 63.9575 , 135.4569
ECHO Id= DOUT X,Y= 50.0000 , 95.5000
ECHO Id= DIN X,Y= 50.0000 , 107.5000
ECHO Id= DOUT X,Y= 62.1942 , 66.9702
ECHO Id= DIN X,Y= 55.1408 , 76.6784
ECHO Id= DOUT X,Y= 88.8289 , 51.0566
ECHO Id= DIN X,Y= 77.4162 , 54.7648
ECHO Id= DOUT X,Y= 119.7306 , 53.8378
ECHO Id= DIN X,Y= 108.3179 , 50.1295
ECHO Id= DOUT X,Y= 143.0959 , 74.2513
ECHO Id= DIN X,Y= 136.0425 , 64.5431
Finished normally - please check for any warnings.
Warnings: 0
Errors: 0

Process finished with exit code 0
```

In this project I needed to connect the data-in and data-out of my WS2812B LEDs. I copied the text lines above into a text editor and removed everything but the co-ordinates. I then saved it as a CSV and used an Excel spreadsheet to make up a paired list of start/end coordinate pairs to link every third pair (think about that). From there I created a set of SEGMENT commands for my data file. Job done.

7.7 GROUP

The above example in DEFGROUP defines a group called LED_CAP which comprises of two components and which can be placed using commands like this :-

```
GROUP,LED_CAP,50,50,0
GROUP,LED_CAP,60,60,0
GROUP,LED_CAP,70,70,0
```

The above places three LEDs with capacitors at positions 50/50, 60/60 and 70/70. The third parameter is a group rotation angle – in other words no rotation (0) in this case.

7.8 KICADPCB

Syntax: KICADPCB,ID,Filename

This command loads an existing kicad_pcb file and makes it available to use. For example:-

```
KICADPCB,myPCB,somefile.kicad_pcb
```

Then, before any other command which generates output, use the command:-

```
myPCB
```

MakePCB checks the start of 'somefile.kicad_pcb' for the signature '(kicad_pcb'. If not found the loading terminates with an error.

If all is well makePCB removes the leading bracket '(' and the trailing bracket ')'. All your other commands will then be added to the end of the kicad_pcb file and the brackets put back.

The 'somefile.kicad_pcb' file will not be overwritten unless you use the same name for your command data file AND it is in the same folder as the kicad_pcb you are loading. So, generally speaking, quite safe.

7.9 LIST/USELIST/ENDLIST

Syntax: LIST,ID,Sep,Filename

This lets you read in a txt file with a list of parameters for subsequent substitution into a block of commands. Id is your name for the list, it is unique to lists so will not conflict with any other template names.

Sep is the separator character you used in the data file. Sadly, you cannot use commas in this version of makePCB.

Filename is the path to the data file to load. Usually this will contain coordinates for drawing stuff but makePCB doesn't care it just replaces placeholders %0%,%1% etc with corresponding values from each line of the data file – if a parameter is missing then the placeholder is not replaced and the program will likely throw an error message like 'Cannot convert %5% to float'.

7.9.1 USELIST

Syntax: USELIST,<listId>

This indicates the start of a command list. MakePCB collects all commands up to the ENDUSELIST statement into a list for processing when it sees ENDUSELIST.

Within the commands you can insert placeholders for later substitution from the named list.

E.g. SEGMENT,LINE,%0%,%1%,%2%,%3%,0.01,F.Cu,0

See later for the SEGMENT syntax/parameters. The command above will use the first 4 parameters, from each line in the list, for the start and end coordinates of a track segment that is 0.01mm wide on the F.Cu layer using Net 0. So if your list contains 10 lines you end up with 10 tracks.

Your list file may contain expressions – they are evaluated just like any other numerical or string parameter.

Here's a stupid example which could just place a number of JST connectors and connect tracks to them:-

```
LIST,myList,somefile.txt

USELIST,MyList
SEGMENT,LINE,%0%,%1%,%2%,%3%,0.01,F.Cu,0
JST,ref,0,%0%,%1%,F.Cu
ENDLIST
```

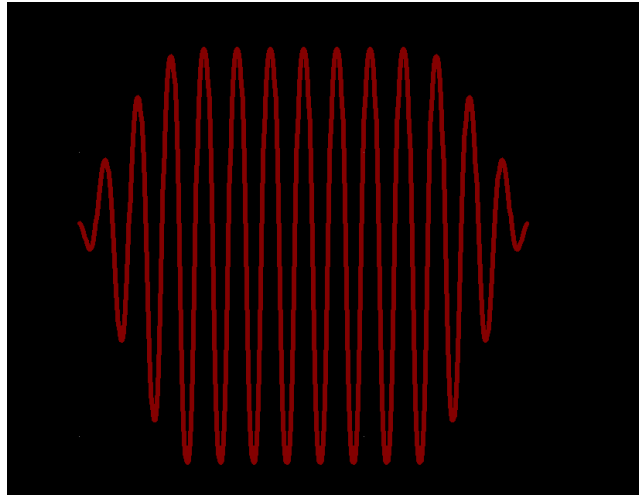
Remember that the above will be executed for every line in the LIST. I have highlighted the coordinates used to connect a track to Pin 1 of the JST.

7.9.2 ENDUSELIST

Syntax: ENDUSELIST

This causes makePCB to iterate over the command list, using the data list, substituting placeholders as it goes.

For example this track segment was created using the data provided with the KiCad microwave demo to create the track segment you can see in the screen shot below:-



7.10 ORIGIN

Syntax: ORIGIN,X,Y

e.g. ORIGIN,30,-190

KiCad locates its co-ordinate origin somewhere top-left just *outside* the boundaries of the drawing – which can flip your layout if you used a bottom-left scheme to work out your numbers.

Specifying the ORIGIN will cause the software to ADD to X,Y values to those in the position data file thus shifting your layout – if that's what you want.

The values you use will depend on your chosen page size. But for my A3 page (19.7cm x 42cm) specifying Y as 190 set the drawing origin to bottom left. Since the default origin is outside the drawing page a value of 30 for X will make sure your origin is within the page.

This MUST appear before any component placement entries which use them. Yes, that means you CAN redefine the ORIGIN as and when you wish. This is a useful tactic is you just want to shift a group of page elements without recalculating the individual XY values.

When makeBCP starts it creates a pair of global variables xOrigin and yOrigin which will both be zero until you change the origin. They are updated when you use the ORIGIN command. Beware, it is possible for you to overwrite the values of these using SETG,xOrigin,nnn or SETG,yOrigin,nnn.

7.11 REPEAT

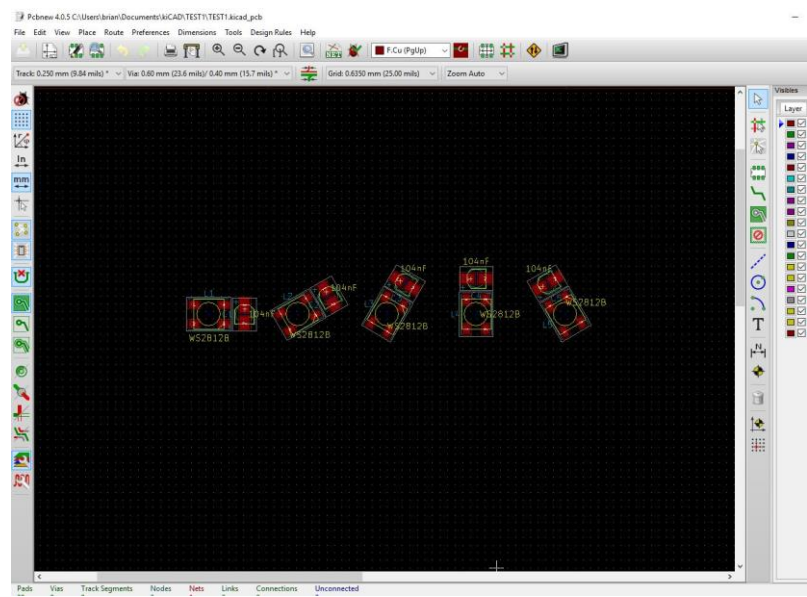
Syntax: REPEAT,repeatId, Xpos,Ypos,startAngle,Counter,stepX,stepY, stepAngle

Executes a repeat list created with DEFREPEAT.

e.g. REPEAT,ARRAY1,10,20,0,5,10,15,5

This will repeat ARRAY1 starting at co-ordinate 10,20 un-rotated (startAngle=0) and will step ARRAY1 5 times by 10mm in the X direction, 15mm in the Y direction and rotate ARRAY1 by 5 degrees at each

step (if you want to). The following screenshot shows it being used to create an array of WS2812B Leds whilst rotating the placements



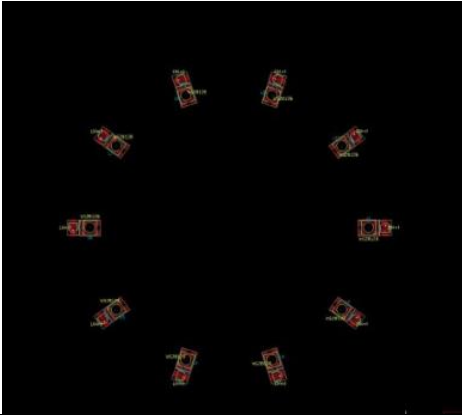
7.12 RING

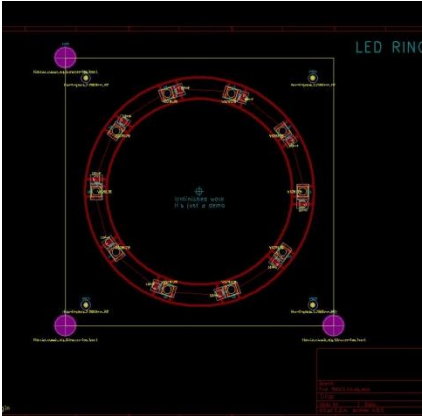

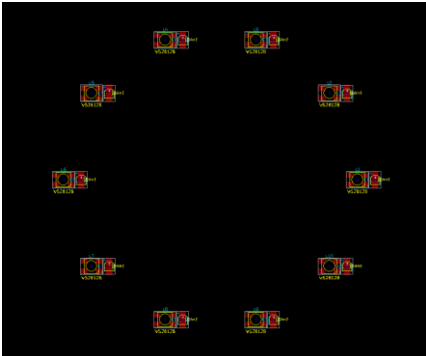
Syntax: RING, ringId, Xpos, Ypos, Radius, Mode, startAngle, Counter, stepAngle

e.g. RING,MyRing,100,100,50,0,0,10,36

This draws a ring of components at 100,100, on a radius of 50 using Mode=0, starting at 0 degrees placing 10 copies every 36 degrees (10x360=360 , simple relationship!).

The mode (an angle) controls how the components are placed. You can use any angle but 0, -90 and 90 make the most sense or use None if you don't want the components rotated..

Mode	Comments
0	Components are aligned along the radius (0 degrees) of the circle. 

-90	<p>Components are aligned tangential to the circumference of the circle facing right.</p> 
90	<p>Components are aligned tangential to the circumference of the circle facing left:-</p> 
None	<p>No component rotation:-</p> 

7.13 SAVEXY

Syntax: SAVEXY,ID,x-position,y-position

This creates a pair of saved variables called ID_X and ID_Y. When used as coordinates these variables are used without being transformed – i.e. they are absolute coordinates (as would appear in the kicad_pcb document).

When used within a ring/repeat they allow you to connect a point from the previous iteration to the current iteration. They also continue to exist after the ring has been completed so myou can use these to tell you where a track connection should be made. The LED RING demo in the

Examples\PCB folder uses this to connect the DIN/DOUT signals from one WS2812B LED to the next and uses the final values to link from a JST connector to the ring.

7.14 SAVEONCEXY

Syntax: SAVEONCEXY,varName,xx,yy

This is intended to be used in the same way as SAVEXY but the value will not be overwritten if the command is placed in a repeat/loop. This is handy for memorising a coordinate as it was at the start of a loop (E.g. where you want to connect to the signal input to).

7.15 SETDIRECTION

Syntax: SETDIRECTION,x,y

Set the current plotting direction. KiCad X coordinates run left to right as normal but the Y coordinates run top to bottom (Typical monitor direction). This is a bit counter intuitive for us humans.

By using -1 for y to indicate direction reversal we can allow users to enter XY coordinates based on the y values increasing bottom to top. The makePCB.py program will turn X,Y into KiCad friendly coordinates so everything is where you expect it to be.

Typically you would use ORIGIN and SETDIRECTION once only at the start of your data file e.g.

SETDIRECTION,1,-1	Y values increase bottom to top, X remains unchanged. In actual fact, you can use any +ve/-ve values for X,Y. The program sets them to -1 or 1 accordingly. If you use 0 the current direction setting is unchanged. By default, makePCB initialises with 1,-1 so you generally don't need to use this.
ORIGIN,20,197	Origin has been moved to somewhere near the bottom left of the drawing (if A3 which is 19.7cm x 42cm)

7.16 SHOWVARS

Syntax: SHOWVARS

A debugging aid. Dumps a list of the Global, Local and SAVEXY variables to the console.

7.17 STOP

Syntax: STOP

Stop has no parameters. MakePCB stops processing the position data file. Useful for debugging your layout.

The kicad_pcb file is still generated up to the STOP so you can look at it in KiCad's pcbnew to see if your changes worked. I found this useful for working out the positioning of the capacitors and tracks relative to my LED components.

7.18 SETG/SETL

Syntax: SETx,VARNAME,Value

E.g. SETG,Layer,F.Cu

These commands allow you to set a global or local variable. Local variables exist within the body of a GROUP, REPEAT or RING only.

They store Values adjusted for the position of the group.

One possible use is for reducing the amount of work to do if you make a changed which affects a lot of components (e.g. change Net or Layer).

7.19 TEMPLATE

Syntax: TEMPLATE,ID,Filename

This is intended for loading templates which use placeholders. There are a number of standard templates as mentioned earlier.

e.g. TEMPLATE,VIA,template\Via.txt

This loads the template file via.txt and gives it an id of VIA. You can then place a VIA at 10,20 in your layout by putting this in your position data file:-

VIA,10,20,4,3,F.Cu,B.Cu,0 (see later for parameters)

You could define lots of different VIA templates, load them all at the start, and place them at will.

TEMPLATE causes the program to load the templates you intend to use. TEMPLATE lines should appear before any commands which use them. I usually put all of them at the start of the data file.

Filename is a relative path and can be in a sub-folder. I keep the standard templates in a folder called templates and my components in a folder called components. It's a personal choice.

To use a template you specify its ID followed by any required parameters. IDs and parameters are case sensitive. So, for example, if you specify a layer like F.cu then the KiCad newpcb program will complain when you load the file. It should be F.Cu of course.

Component templates can have any name which isn't already taken.

e.g. TEMPLATE,WS2812B,component\WS2812B.txt

You can then place the WS2812B component using this Syntax:-

WS281B,ref,Angle,X,Y,Layer

This is the same for ALL components you use.

Ref would be something like LED. MakePCB will add a usage number for you (LED1,LED2 etc)

The following sections explain the parameters for each of the possible standard commands.

7.19.1 FIDUCIAL

Syntax: FIDUCIAL,REF,X,Y,Clearance,Line-Width,Layer

e.g.FIDUCIAL,FID1,0,0,1,0.15,Edge.Cuts

The default fiducial.txt template uses Fiducials:Fiducial_classic_big_SilkscreenTop_Type1. You should edit Fiducial.txt to change it. Chances are you will stick to one form or another for your pcb manufacturer.

7.19.2 GRAPHIC

Syntax: GRAPHIC,Shape,<shape-params>

This draws the chosen graphic shape. Shape is not case sensitive.

I have found that newpcb WILL draw graphics on a copper layer (if you set the layer to F.Cu or B.Cu) and will also generate a Gerber plot with it on the copper layer but the newpcb GUI will whinge if you try to edit it.

7.19.2.1 ArcAngle

Syntax: GRAPHIC,ArcAngle,X,Y,Radius,startAngle,stopAngle,Line-Width,Layer

Draws an arc centred at X, Y between the two Angles (anti-clockwise rotation). KiCad appears to use clockwise rotation which is the opposite of regular math circle coordinates.

7.19.2.2 Circle

Syntax: GRAPHIC,Circle,X,Y,Radius,Line-width,Layer

e.g. GRAPHIC,CIRCLE,30,30,5,0.15,Edge.Cuts

Draws a graphic circle.

7.19.2.3 Line

Syntax: GRAPHIC, Line, Xpos1, Ypos1, Xpos2, Ypos2, Width, Layer

Draws a graphic line.

7.19.2.4 Pie

Syntax: GRAPHIC,Pie,X,Y,Radius,startAngle,stopAngle,Line-Width,Layer

Draws a pie shape. If stopAngle is less than startAngle then 360 degrees are added to ensure the pie is drawn correctly – otherwise you would just get segments.

7.19.2.5 Rectangle

Syntax: GRAPHIC, RECT, X, Y, Width, Height, Line-width, Layer

Draws a rectangle at the given position and size and linewidth on the given layer.

7.19.2.6 RoundedRect

Syntax: GRAPHIC,RoundedRect,X, Y, width, height, corner-radius, Line-width, Layer

Draws a rectangle at the given position and size and linewidth on the given layer with corners of the given radius. If the radius exceeds either half the width or height it will not work as expected – use sensible values.

7.19.2.7 Text

Syntax: GRAPHIC,TEXT, JUSTIFY, X, Y, Angle, Size, Thickness, Layer, Text

Text is the last item so that it may contain commas. It can also include double quotes. JUSTIFY can be LEFT, RIGHT, CENTER or CENTRE (not case sensitive. Anything else defaults to centre.)

7.19.3 KEEPOUT

Syntax: KEEPOUT,Shape,<params>

This is the same as a zone (see later) but uses a different template which has a keep out section added.

7.19.4 MOUNT

Syntax: MOUNT, REF, X, Y, Drill-Radius, Outer-Radius, Inner-Width, Outer-Width

e.g.MOUNT,MTG,10,10,2.2,2.45,0.15,0.05

Creates a mounting hole at the specified location.

Drill-Radius is the value used in the template - use proper drill sizes. Outer-Radius sets the size of the copper surrounding the hole. Inner-width and outer-width are graphic line widths the values above came from the standard template.

7.19.5 SEGMENT or TRACK

Syntax: SEGMENT,Shape,<shape params> or TRACK,Shape,<shape params>

Create a track of the required shape. Shape is not case sensitive. If you prefer to use the word TRACK it does the same thing as SEGMENT (which is what a kicad_pcb track is called in the pcb document).

7.19.5.1 Arc

Syntax: SEGMENT,ARC,CX,CY,Radius,startAngle,stopAngle, NumSegments, Width, Layer ,Net

Draws an arc of a circle using short segments to approximate a smooth circle.

The arc is drawn anti-clockwise so if you want a circle with a gap in it specify a stopAngle less than startAngle.

Arc tracks drawn with segments are editable in the newpcb GUI.

7.19.5.2 Circle

Syntax: SEGMENT,CIRCLE,CX,CY,Radius,NumSegments,Width,Layer,Net=params.split(",")

e.g. SEGMENT,CIRCLE,100,100,50,360,1,F.Cu,0

Draws a circular track centred at CX,CY. The circle is created using short linear segments – if you want a spider’s web effect use a low number for NumSegments. I found 360 gave a smooth looking curve.

Circular tracks drawn like this are editable in newpcb so if you needed to add gaps you could delete some of the segments (manually).

7.19.5.3 Grid

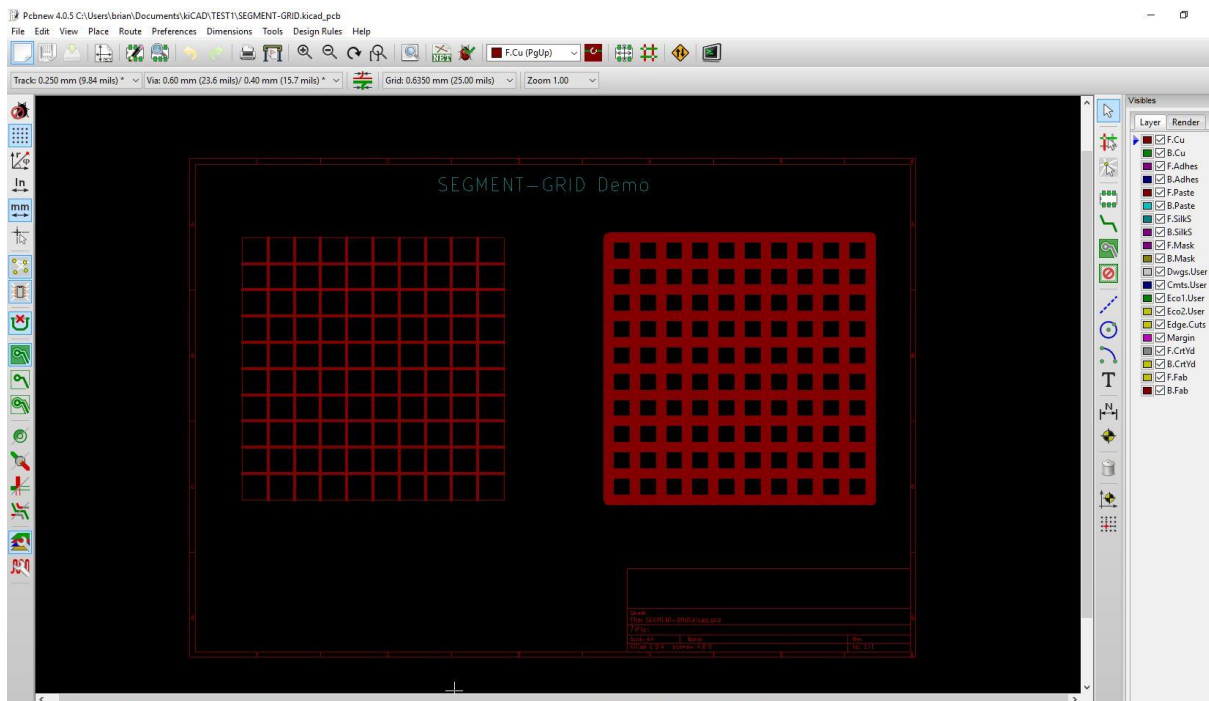
Syntax: SEGMENT, GRID, Xpos, Ypos, Width, Height, Hgaps, Vgaps, BorderWidth, LineWidth, Layer, Net

Xpos, Ypos, Width and Height set the size and position of the grid.

Hgaps and Vgaps is the number of, well, gaps. For example, if you have a 50mm height and you want a line every 10mm you would want 5 gaps.

BorderWidth sets the thickness of the lines used for the borders. LineWidth is the thickness of the grid lines. Using separate dimensions means you can have a nice tidy border even if the lines are very thick.

To create solid copper rectangular area you could use a line width twice the size of the gaps. If you do that the code only draws either the horizontal lines or vertical not both because that would be a waste of time. Here’s a demo you’ll find in the Examples/Tracks folder



7.19.5.4 Line

Syntax: SEGMENT,LINE,X1,Y1,X2,Y2,Width,Layer,Net

e.g. SEGMENT,LINE,10,100,20,19,2,F.Cu,0

Draws a linear track on layer F.Cu from 10,100 to 20,19 2mm thick (Good for a power bus?) using Net 0.

All other segments use Line as their base.

7.19.6 TARGET

Syntax: TARGET,Shape,X,Y,Size,Width,Layer

e.g. TARGET,plus,170,20,5,0.15,Edge.Cuts

Draws a target with specified shape (**plus** or **x**). Layer is case sensitive.

7.19.7 VIA

Syntax: VIA,X,Y,SIZE,DRILL,LayerF,LayerB,NET

e.g. VIA,10,20,4,3,F.Cu,B.Cu,0

Places a VIA at 10,20 between layers F.Cu and B.Cu.

7.19.8 ZONE

Syntax: ZONE,Shape,<params>

Creates a filled zone of the specified shape.

<params> depends on chosen shape – see following comments regarding the shapes:-

7.19.8.1 Poly

Params: Net,NetName,Layer,Hatch-Type,Hatch-Edge,clearance,min-thickness,fill-arc-segment,ThermalGap,fill-thermal-bridge,X0,Y0...Xn,Yn

This shape is the basis of all the other shapes. My code just creates the required coordinate lists.

All the other zone shapes use the same parameters – only the co-ordinate lists are changed (by calculation).

X0,Y0...Xn,Yn – is a list of co-ordinates for the zone outline. KiCad will complete the shape using straight lines so for a square you need 4 pairs of co-ordinates (or use the 'rect' shape).

If there is a missing X or Y you will see a warning in the program output like this:-

WARNING at line xx Zone has an odd number of co-ordinates. Zone ignored.

NetName MUST NOT include double quotes and may be left empty. If you specify a net name, but there's nothing in Header.txt, newpcb will warn you when you open the kicad_pcb document and will add an empty reference up in the nets section.

Hatch-Type can be 'edge', 'full' or 'line' (no quotes).

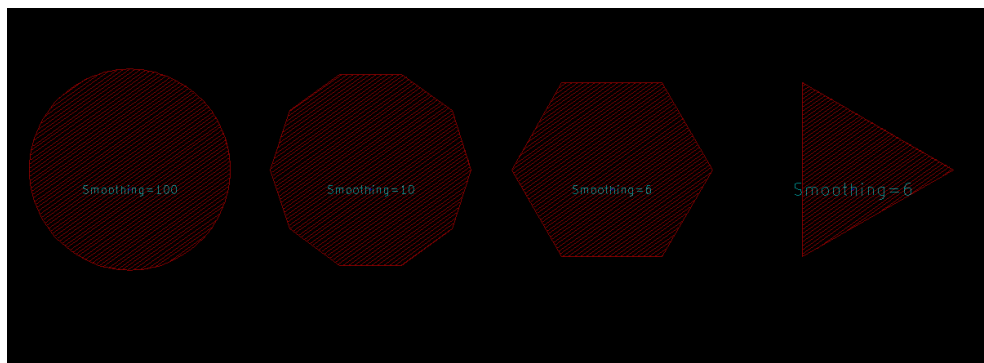
Hatch-Edge is the hatch dimension – it controls the spacing between the hatch lines.

You should set clearance, min-thickness, fill-arc-segment and fill-thermal-bridge to values you want to use – you can get these from the KiCad pcbnew dialogs. My templates just use the default values which were inserted when I added a zone to a new board.

7.19.8.2 Circle

params: X,Y,Radius,Smoothing, Net,NetName,Layer,Hatch-Type,Hatch-Edge,clearance,min-thickness,fill-arc-segment,thermal-gap,fill-thermal-bridge

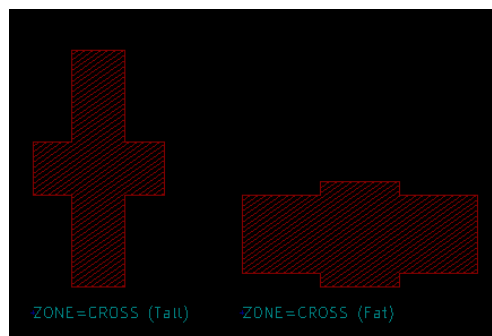
Smoothing defines the number of steps used to create the circle – smaller numbers will result in a spider's web effect.



7.19.8.3 Cross

Params: X,Y,width,height,thickness,Net,NetName,Layer,Hatch-Type,Hatch-Edge,clearance,min-thickness,fill-arc-segment,thermal-gap,fill-thermal-bridge

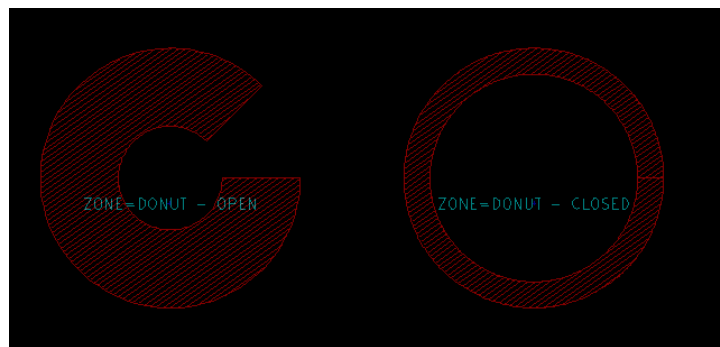
Draws a cross within a container rectangle (X,Y,W,H) with a cross of a given thickness



7.19.8.4 Donut

params: X, Y, InnerRadius, OuterRadius, startAngle, stopAngle, Smooth, Net, NetName, Layer, Hatch-Type, Hatch-Edge, clearance, min-thickness, fill-arc-segment, thermal-gap, fill-thermal-bridge

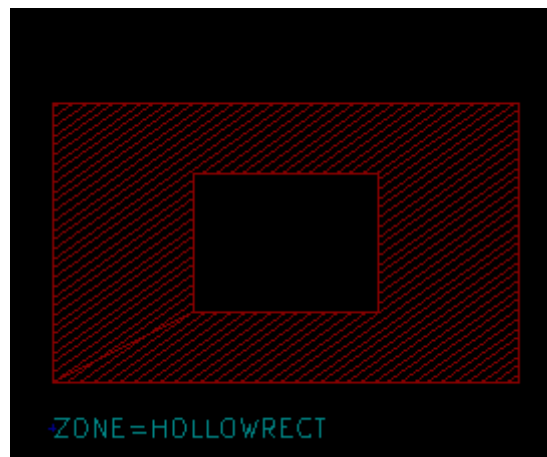
Draws a donut shaped zone. If you specify a start angle of 0 and an end angle of 360 you will get a closed donut as shown in the screen shot below. Higher values of Smooth give smoother curves, smaller values will give polygonal shapes



7.19.8.5 HollowRect

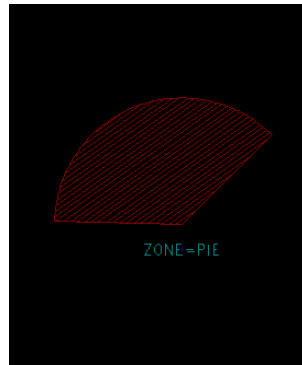
Params: X, Y, outerW, outerH, innerW, innerH, Net, NetName, Layer, Hatch-Type, Hatch-Edge, clearance, min-thickness, fill-arc-segment, thermal-gap, fill-thermal-bridge

Draws a rectangular zone with a centred hole in the middle like this:-



7.19.8.6 Pie

params: X,Y,Radius,startAngle,stopAngle,Smoothing,Net,NetName,Layer,Hatch-Type,Hatch-Edge,clearance,min-thickness,fill-arc-segment,fill-thermal-bridge

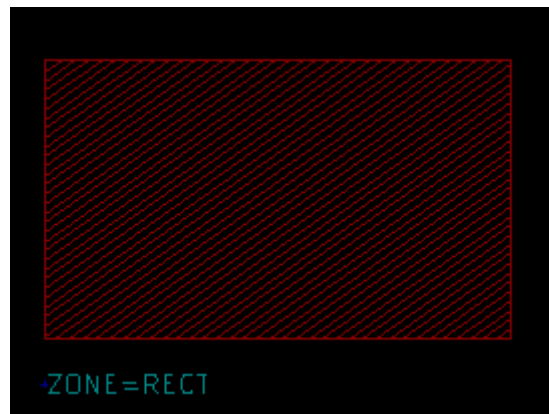


Similar to circle but the first coordinate pair is the centre of the circle so you get a pie effect. Smoothing is applied to the difference between the start and stop angles. So, if, for example, you chose startAngle=30 and endAngle=60 then Smoothing=15 you would get 2 degree separation $((60-30)/15)$ between the generated coordinates.

7.19.8.7 Rect

params: X,Y,Width,Height, Net,NetName,Layer,Hatch-Type,Hatch-Edge,clearance,min-thickness,fill-arc-segment,fill-thermal-bridge

Draws a rectangular zone as shown below:-

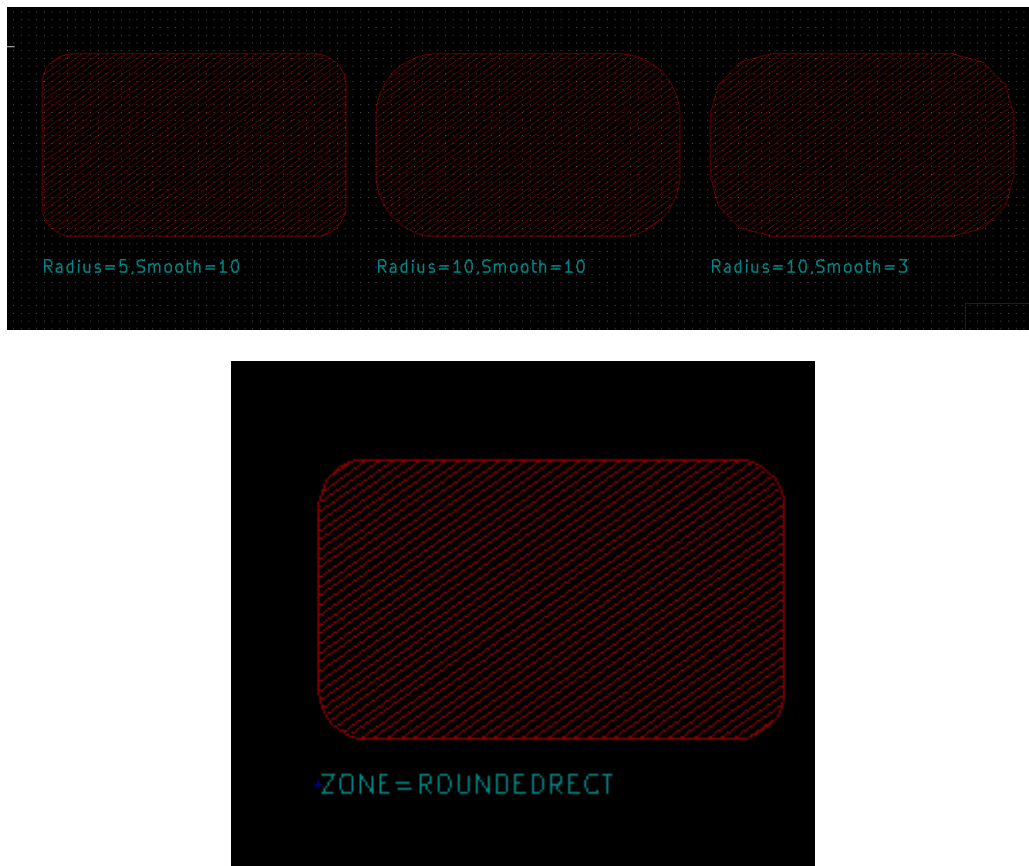


7.19.8.8 RoundedRect

Params: X,Y, Width,Height, Radius,Smooth,Net,NetName,Layer,Hatch-Type,Hatch-Edge, clearance,min-thickness,fill-arc-segment,thermal-gap,fill-thermal-bridge

Draws a rectangle with rounded corners. Smooth affects how smooth the curves are. Since the angle is 90 degrees you don't need very high smoothing values (see screenshots below).

Radius is the radius of the corners. Be sensible with radius, the value isn't checked. Theoretically, drawing a square of side S and using a radius of S/2 should result in a circle .



8 Variables

The program uses global and local variables. Local variables exist with a GROUP,RING or REPEAT block. Global variables can be set and used from anywhere and may be useful for specifying things like linewidth , layers etc which can be changed by just changing the variable name.

Variables are created as soon as you use the SETG or SETL commands and can be deleted with CLRG and CLRL

8.1 Global variables.

```
SETG,LineWidth,0.15
SETG,Layer,F.Cu
SEGMENT,LINE,0,0,50,50,LineWidth,Layer
SEGMENT,LINE,50,50,60,70,LineWidth,Layer
```

This would add two tracks to Layer F.Cu. If you wanted the tracks swapped from F.Cu to B.Cu you just need to change SETG,Layer,F.Cu to SETG,Layer,B.Cu. We could be talking lots of tracks so you can see the benefit of the variables – it saves you typing and reduces errors.

8.2 Local Variables

Outside of GROUP, REPEAT or RING blocks, local variables behave the same as global variables BUT local variables with the same name as a global variable will use the local value first so if you write this:

```
SETG,Layer,B.Cu
SETL,Layer,F.Cu
SEGMENT,LINE,0,0,50,50,LineWidth,Layer
```

The tracks will use Layer F.Cu. This is by design – nice cop out phrase that. Basically, if you place a GROUP inside a REPEAT or RING any pre-existing local variables are visible to the inner block.

However, if you change the value of a local variable inside a GROUP, REPEAT or RING block they will revert to their prior values when the END is reached. This is the same variable scoping as in most languages.

8.3 Using Variables and Expressions

makePCB only supports addition and subtraction for numerical parameters e.g.

```
SETG,CX,100
SETG,CY,100
SEGMENT,CX+50,CY+100,.....
SEGMENT,CX+55,CY+100,.....
```

Though not very useful, the above is an example of using simple arithmetic and using saved variable values. If we wanted to move the two tracks (SEGMENT) we would only need to change the SETG CX,CY values – very useful if there are a lot of tracks to move.

9 Examples

These can be found in the Examples folder. Each example has a corresponding data file and a resulting kicad_pcb file.

Folder	Comment
GRAPHICS	A page showcasing the GRAPHIC commands and their resulting shapes
ZONES	ZONES-MISC – a page showing the different zone shapes (Also used for Keepout). CIRCLES – a page showing the effect of smoothing. Use the command to create a triangle if you wish.
LIST	A page showing how to use a data file containing line co-ordinates to generate a wave shape. You could also use the data file to place components so if the coordinates come from, say, openSCAD you can accurately place components to match,
RING	Examples of using the RING command to create a ring of components
TRACKS	Examples showing the creation of different shaped tracks.
REPEAT	Examples of using the REPEAT command for making rectangular arrays
PCB	Just an example of PCB layout with tracks linking between LEDs and a JST connector.

10 Using the program

The code is written in Python and was developed with Python 3.5 using PyCharm. I have run it using 2.7.13. The only different was the print statements from V2.7x include brackets in the console output

The program runs standalone and does not need KiCad open. When finished you just need to open the kicad_pcb output file in KiCad's pcbnew program.

Here's an example run for the LED ring above. It will prompt you for the data file (highlighted red) which should have a .pos extension BUT the code only wants the filename part as it uses that for the kicad_pcb output filename and adds the .kicad_pcb for you.

```
"C:\Program Files (x86)\Python 3.5\python.exe"  
C:/Users/brian/PycharmProjects/kicad5/makePCB3.py  
makePCB.py Vsn 1.0.0  
  
Enter the name of the position data file without the .pos part. This will also be used as the name of  
the kicad_pcb output file.  
Position data file:-TEST1  
Opened position data file TEST1.pos  
Created kicad_pcb file TEST1.kicad_pcb  
INFO at line 4 ASIS ID=[HEADER]. Loaded ok.  
INFO at line 9 TEMPLATE ID=[SEGMENT]. Loaded ok.  
INFO at line 10 TEMPLATE ID=[VIA]. Loaded ok.  
INFO at line 11 TEMPLATE ID=[TARGET]. Loaded ok.  
INFO at line 12 TEMPLATE ID=[GRLINE]. Loaded ok.  
INFO at line 13 TEMPLATE ID=[FIDUCIAL]. Loaded ok.  
INFO at line 14 TEMPLATE ID=[GRCIRCLE]. Loaded ok.  
INFO at line 15 TEMPLATE ID=[MOUNT]. Loaded ok.  
INFO at line 16 TEMPLATE ID=[GRTEXT]. Loaded ok.  
INFO at line 17 TEMPLATE ID=[GRARC]. Loaded ok.  
INFO at line 18 TEMPLATE ID=[ZONE]. Loaded ok.  
INFO at line 19 TEMPLATE ID=[KEEPOUT]. Loaded ok.  
INFO at line 23 TEMPLATE ID=[WS2812B]. Loaded ok.  
INFO at line 24 TEMPLATE ID=[CAP]. Loaded ok.  
Finished normally - please check for any warnings.  
Warnings: 0  
Errors: 0  
  
Process finished with exit code 0
```

11 Error Messages

Most errors are generated by simple errors in the data file. The error message includes the line number, from the data file, where the erroneous parameter can be found.

11.1 evalParm exception

Generally this means you have either left out a parameter or used an undefined variable.

evalParm is the method used to interpret your numerical parameters. If you provide the name of a variable and it defined then the variable value is returned otherwise None is returned and methods replying on the value will fail with the result that, for example, a segment isn't drawn. However, the rest of the data file will be read and your kicad_pcb document will be created with everything that can be created.

11.1.1 evalNumericParam exception ("could not convert string to float: 'variable name',)

Caused by line NN.

Most likely an undefined variable 'variable name' used in line NN of your data file.

11.1.2 evalNumericParam exception ('could not convert string to float: ',)

Caused by line NN

Probably a missing parameter – look for two commas with nothing in between on line NN.

11.2 Missing or Extra parameters

If you miss off a parameter from the expected list you will get one of the following messages. You should check the syntax of the command on the specified line.

11.2.1 ERROR at data file line NN error= ('not enough values to unpack (expected 3, got 2)',)

The error line number is NN. This means you have less parameters than are needed.

11.2.2 ERROR at data file line NN error= ('too many values to unpack (expected 3)',)

Again, NN tells you which line was in error. This means you have more parameters than are needed.

11.3 Capitalisation

The first item on a command line is case sensitive. The built-in commands (TEMPLATE, VIA etc) are all in capital letters.

11.4 WARNING at line XX No template loaded for ID=<item> check your .pos file.

This indicates a probable typo in the tag (<item>) you used for the template OR you haven't loaded it yet. It's best to load all the templates you want at the start.

11.5 Invalid numerical parameters

Where a number is expected and you use text you will get messages like this :-

ERROR at data file line NN error= ("could not convert string to float: 'xxxx' ")

NN will be the line number in your data file. 'xxxx' will be the text from the data file that could not be converted. Make sure your data file has the required number of parameters.

11.6 Cannot convert %x% to float (or int)

This normally arises if you use a LIST with too few parameters on a line.

11.7 Warnings

These are all prepended with the text:

WARNING at line NN .

In no particular order (well, actually, it's the order in the parser, and I'm not sure I've nailed them all) they are:-

1. `One of the parameters on line NN resolved as None. Possibly an unassigned variable. The line will be skipped.`

The command on the offending line will be skipped but the rest of your input data will be processed. This will result in something missing from the pcb layout.

2. `Variable names should be strings. Ignored`

You probably tried to use something that looks like a number. There is no restriction on variable names but they should not contain a comma or look like a number.

3. `List 'Id' already loaded. Ignored`

Means you probably tried to load a second list using the same Id as one already loaded. Not a good ploy.

4. `Orphaned ENDUSELIST encountered`

You have ENDUSE with no prior USE,<list>

5. `Ring with Id 'xxxx' will be redefined.`

You have used DEFRING,xxxx twice. The first ring definition will be overwritten.

6. `Orphaned ENDRING encountered.`

You have used an ENDRING but no DEFRING.

7. `Ring ID 'xxxx' has no step angle. Ring ignored.`

You need to specify a stepAngle for a RING

8. `Ring ID 'xxxx' hasn't been defined. Ring ignored.`

You have a missing DEFRING,xxxx

9. `Repeat with Id 'xxxx' will be redefined.`

You have used REPEAT,xxxx twice. The first definition will be overwritten.

10. `Orphaned ENDREPEAT encountered`

ENDREPEAT without a DEFREPEAT.

11. `Repeat ID 'xxxx' hasn't been defined. Repeat ignored.`

You need to define a repeat list.

12. You cannot use DEFGROUP within a DEFGROUP. Previous ENDGROUP may be missing.

makePCB doesn't support defining groups within groups.

13. You cannot use DEFGROUP within a DEFREPEAT. Previous ENDREPEAT may be missing.

Define your group outside the DEFREPEAT-ENDREPEAT block – you can then use the GROUP inside the DEFREPEAT-ENDREPEAT

14. Group with Id 'xxxx' will be redefined

You have used DEFGROUP with the same Id as a previous DEFGROUP

15. Orphaned ENDGROUP encountered

Perhaps a missing DEFGROUP?

Group ID 'xxxx' hasn't been defined. Group ignored.

You have tried to enter the command GROUP,xxxx but haven't created a group with DEFGROUP,xxxx

16. No template loaded for ID='xxxx' check your data file.

Your data file should load templates something like this TEMPLATE,xxxx,<filename> before you use a command which uses it. For example the SEGMENT command expects a to use a template called SEGMENT.

17. Requested template 'xxxx' doesn't appear to be of type 'yyyy'. Ignored.

When a function wants to use a template it checks the beginning of the template to help identify it is the correct type. So, GRAPHIC,LINE will use the GRLINE template which should begin with the text '(gr_line'. If it doesn't you get this warning. So, if you inadvertently do this:-

```
TEMPLATE,GRLINE,templates\Segment.txt
```

you will get Requested template 'GRLINE' doesn't appear to be of type 'gr_line'. Ignored because Segment.txt begins with the text '(segment' not '(gr_line'.

18. Unsupported graphics xxxx. Ignored

xxxx isn't in the list of shapes that can be drawn.

19. Unsupported text justification [xxxx] using Centered.

GRAPHIC,TEXT accepts LEFT,RIGHT,CENTER (or CENTRE) only.

20. "Unknown zone/keepout shape 'xxxx'. Zone/keepout ignored."

Zones shapes in KiCadParser.py V1.0.0 can be POLY, RECT, CIRCLE, CROSS, PIE, ROUNDEDRECT, DONUT, HOLLOWRECT.

21. `Cross thickness must be less than width and height of the container rect. Ignored.`

You have asked the program to create a zone/keepout using a cross but the bars of the cross exceed the width/height dimensions. The command is, as it says, ignored.

22. `ArcSegments should be 16 or 32. Defaulting to 16`

When creating a zone the Arc Segments parameter can only be 16 or 32 – well, that’s all the pcbnew dialog will allow.

23. `Zone has an odd number of co-ordinates. Zone ignored. Length=NNN`

Zones and keepouts are created using a series of X,Y coordinates so there should always be an even number of them.

24. `Unsupported segment shape.`

The list of shapes available is LINE, CIRCLE, ARC, RECT and GRID. You may have a typo.

25. `SEGMENT segment_line_helper was passed a 'None' value in the parameter list. Ignored"`

None value is Python speak for, in this case, an unresolved/undefined variable used as a parameter. Segment_line_helper is a routine invoked from a number of segment shape handlers. The track will not be created. This is put to good use for chaining tracks in REPEAT/RING commands. See the section ‘Using Variables’

26. `One of the parameters on line NN resolved as None. Possibly an unassigned variable. The line will be skipped.`

I think the error message says it all.

11.8 Other errors

Please provide a copy of your data file, a copy of the error messages and version of this program for debugging.

12 Modifying MakePCB

MakePCB comprises of two python files makePCB and KiCadParser.py

12.1 makePCB.py

This script opens the position data file, skips comment lines, halts processing if a STOP command is encountered otherwise feeds each line to KiCadParser.py for processing.

The processed line must return a completed template (string) or None. If None is returned then makePCB.py doesn't write anything to the output file.

12.2 KiCadParser.py

This is the workhorse document. I apologise if I've broken any PEP rules but the code works ok with my examples.

The document begins with a StateVars class. These are the variables which are saved when a group/ring/repeat list is being executed. At the end of a group/ring/repeat the variables are restored to their original values.

```
class StateVars():
    def __init__(self):
        self.groupList=[]           # groups within groups is possible
        self.repeatList=[]         # nested repeats is possible
        self.ringList=[]           # nested rings are possible
        self.groupId=""            # current group Id
        self.repeatId=""
        self.ringId=""
        self.ringX=0.0             # X/Y on perimeter of ring
        self.ringY=0.0             # used for relative positioning
        self.ringCx=0.0            # centre of ring rotation
        self.ringCy=0.0            #
        self.ringRad=0.0           # ring radius
        self.ringAngle=0.0         # radial component rotation
        self.xOrigin=0.0           # rename as Origin???
        self.yOrigin=0.0
        self.angleOrigin=0.0
        self.repeatX=0.0           # for step and repeat operations
        self.repeatY=0.0
        self.repeatAngle=0.0
        self.groupX=0.0
        self.groupY=0.0
        self.groupAngle=0.0
```



```
self.localVars={}           # available in repeat loops etc
```

Following on from there is the KicadParser class. The `__init__` section defines the action table. You will recognise the commands – always in upper case. Here you can add an alias for a command as long as it doesn't clash with IDs for components – look at how I did this with SEGMENT and TRACK.

```
# action jump table - gets rid of a long if-elif chain
self.actions={}
self.actions["TEMPLATE"]=self.loadTemplate
self.actions["ASIS"]=self.loadAsis
self.actions["ORIGIN"]=self.origin
self.actions["DEFGROUP"]=self.defGroup
self.actions["ENDGROUP"]=self.endGroup
self.actions["GROUP"] = self.processGroup
self.actions["DEFREPEAT"]=self.defRepeat
self.actions["ENDREPEAT"]=self.endRepeat
self.actions["REPEAT"] = self.processRepeat
self.actions["DEFRING"]=self.defRing
self.actions["ENDRING"]=self.endRing
self.actions["RING"] = self.processRing
self.actions["TRACK"]=self.segment
self.actions["SEGMENT"]=self.segment
self.actions["GRAPHIC"]=self.graphic
self.actions["TARGET"]=self.target
self.actions["VIA"]=self.via
self.actions["FIDUCIAL"]=self.fiducial
self.actions["MOUNT"]=self.mounting
self.actions["ZONE"]=self.zone           # zones and keepouts do the same thing
self.actions["KEEPOUT"]=self.zone        # just use different templates
self.actions["SETDIRECTION"]=self.setDirection
self.actions["LIST"] = self.loadList
self.actions["USELIST"] = self.beginUseList
self.actions["ENDUSELIST"] = self.endUseList
self.actions["ECHO"] = self.echo
self.actions["SETG"] = self.setGlobal
self.actions["SETL"] = self.setLocal
self.actions["CLRL"] = self.clrLocal
self.actions["CLRG"] = self.clrGlobal
self.actions["SAVEXY"]=self.saveXY
self.actions["SAVEONCEXY"]=self.saveOnceXY
self.actions["SHOWVARS"]=self.showVars
self.actions["KICADPCB"]=self.loadKicadPcb
```

All the action methods MUST return either None or a completed template. None tells makePCB.py to ignore it. This is useful for commands which have no output to send to the kicad_pcb file you are creating.

Also in the `__init__` section is the formatting string for floating point numbers. Currently this is set to 4 decimal places, which represents .0001 of a millimetre or inch. That should be accurate enough for most purposes but if you need more just change this line:-

```
# formatting strings
self.fmtFloat="{:.4f}" # used for building X,Y coords strings
```

Zone shapes are handled with code like this:-

```
def zone(self,id,params):

    self.ZoneTemplate=id.strip().upper()

    # zone_poly is called by each alternative
    # and checks the template exists

    Shape,Remainder=params.split(",",1)
    Shape=Shape.strip().upper()

    if Shape=="RECT":          return self.zone_rect(Remainder)
    elif Shape=="POLY":        return self.zone_poly(Remainder)
    elif Shape=="CIRCLE":      return self.zone_circle(Remainder)
    elif Shape=="CROSS":       return self.zone_cross(Remainder)
    elif Shape=="PIE":         return self.zone_pie(Remainder)
    elif Shape=="ROUNDEDRECT": return self.zone_roundedrect(Remainder)
    elif Shape=="DONUT":       return self.zone_donut(Remainder)
    elif Shape=="HOLLOWRECT":  return self.zone_hollow_rect_centred(Remainder)

    self.Warning("Unknown zone/keepout shape '"+Shape+"'. Zone/keepout ignored.")
    return None
```

Notice that the shape is made upper case so that rect and Rect are the same thing.

Adding extra zone shapes is possible by following what I have done. Zones like HOLLOWRECT are possible because the ZONE entry in kicad_pcb uses a set of XY coordinates (xy <x> <y>) like this:-

```
(zone (net 0) (net_name "") (layer F.Cu) (tstamp 589D8DD7) (hatch full 0.5080)
(connect_pads (clearance 0.5080))
(min_thickness 0.2540)
(fill (arc segments 16) (thermal gap 0.5080) (thermal bridge width 0.5080))
(polygon
  (pts
    (xy 75.0000 70.0000) (xy 74.9507 71.5698) (xy 74.8029 73.1333) (xy 74.5572 74.6845)
  (xy 74.2146 76.2172)
  (xy 73.7764 77.7254) (xy 73.2444 79.2031) (xy 72.6207 80.6445) (xy 71.9077 82.0438) (xy
  71.1082 83.3957)
  (xy 70.2254 84.6946) (xy 69.2628 85.9356) (xy 68.2242 87.1137) (xy 67.1137 88.2242) (xy
  65.9356 89.2628)
  (xy 64.6946 90.2254) (xy 63.3957 91.1082) (xy 62.0438 91.9077) (xy 60.6445 92.6207) (xy
  59.2031 93.2444)
  (xy 57.7254 93.7764) (xy 56.2172 94.2146) (xy 54.6845 94.5572) (xy 53.1333 94.8029) (xy
  51.5698 94.9507)
  (xy 50.0000 95.0000) (xy 48.4302 94.9507) (xy 46.8667 94.8029) (xy 45.3155 94.5572) (xy
  43.7828 94.2146)
  (xy 42.2746 93.7764) (xy 40.7969 93.2444) (xy 39.3555 92.6207) (xy 37.9562 91.9077) (xy
  36.6043 91.1082)
  (xy 35.3054 90.2254) (xy 34.0644 89.2628) (xy 32.8863 88.2242) (xy 31.7758 87.1137) (xy
  30.7372 85.9356)
  (xy 29.7746 84.6946) (xy 28.8918 83.3957) (xy 28.0923 82.0438) (xy 27.3793 80.6445) (xy
  26.7556 79.2031)
  (xy 26.2236 77.7254) (xy 25.7854 76.2172) (xy 25.4428 74.6845) (xy 25.1971 73.1333) (xy
  25.0493 71.5698)
  (xy 25.0000 70.0000) (xy 25.0493 68.4302) (xy 25.1971 66.8667) (xy 25.4428 65.3155) (xy
  25.7854 63.7828)
  (xy 26.2236 62.2746) (xy 26.7556 60.7969) (xy 27.3793 59.3555) (xy 28.0923 57.9562) (xy
  28.8918 56.6043)
  (xy 29.7746 55.3054) (xy 30.7372 54.0644) (xy 31.7758 52.8863) (xy 32.8863 51.7758) (xy
  34.0644 50.7372)
  (xy 35.3054 49.7746) (xy 36.6043 48.8918) (xy 37.9562 48.0923) (xy 39.3555 47.3793) (xy
  40.7969 46.7556)
  (xy 42.2746 46.2236) (xy 43.7828 45.7854) (xy 45.3155 45.4428) (xy 46.8667 45.1971) (xy
  48.4302 45.0493)
  (xy 50.0000 45.0000) (xy 51.5698 45.0493) (xy 53.1333 45.1971) (xy 54.6845 45.4428) (xy
```

```
56.2172 45.7854)
(xy 57.7254 46.2236) (xy 59.2031 46.7556) (xy 60.6445 47.3793) (xy 62.0438 48.0923) (xy
63.3957 48.8918)
(xy 64.6946 49.7746) (xy 65.9356 50.7372) (xy 67.1137 51.7758) (xy 68.2242 52.8863) (xy
69.2628 54.0644)
(xy 70.2254 55.3054) (xy 71.1082 56.6043) (xy 71.9077 57.9562) (xy 72.6207 59.3555) (xy
73.2444 60.7969)
(xy 73.7764 62.2746) (xy 74.2146 63.7828) (xy 74.5572 65.3155) (xy 74.8029 66.8667) (xy
74.9507 68.4302)

)
```

Graphic shapes are handled in a similar fashion BUT KiCad doesn't appear to support using polygon XY coordinates so shapes can only be built using circles, arcs and lines. There is no FILL capability.

If you are into hacking software you can probably figure out the rest for yourself.