PCBmodE Documentation

Release 3.0

Saar Drimer

Dec 22, 2019

Contents

1 Introduction **3**
- 1.1 What is PCBmodE? . . . 3
- 1.2 How is PCBmodE different? . . . 3
- 1.3 What PCBmodE isn't . . . 3

2 Workflow **5**

3 Setup **7**
- 3.1 What you'll need . . . 7
- 3.2 Installation in a virtual environment . . . 7
- 3.3 Running PCBmodE . . . 8

4 Components **11**
- 4.1 Defining components . . . 11
- 4.2 Placing components and shapes . . . 16

5 Shapes **19**
- 5.1 Shape types . . . 19

6 Copper pours **23**
- 6.1 Defining pours . . . 23
- 6.2 Defining buffers . . . 24

7 Text **25**
- 7.1 Fonts . . . 25
- 7.2 Defining text . . . 25

8 Routing **27**
- 8.1 Adding routes . . . 27
- 8.2 Adding vias . . . 28

9 Extraction **29**

10 Layer control **31**

11 Indices and tables **33**

Contents:

CHAPTER 1

Introduction

1.1 What is PCBmodE?

PCBmodE is a Python script that takes input JSON files and converts them into an Inkscape SVG that represents a printed circuit board. *PCBmodE* can then convert the SVG it generated into Gerber and Excellon files for manufacturing.

1.2 How is PCBmodE different?

PCBmodE was conceived as a circuit design tool that allows the designer freedom to put any arbitrary shape on any layer of the board. While it is possible to add graphical elements to other PCB design tools by additional scripts and general arm-twisting, *PCBmodE* is purposefully designed and architected for that purpose.

PCBmodE uses open and widely used formats (SVG, JSON) together with open source tools (Python, Inkscape) without proprietary elements (Gerber is an exception, although the standard is public). It also provides a fresh take on circuit design and opens new uses for the circuit board manufacturing medium.

PCBmodE is free and open source under the GPL 3 license.

1.3 What PCBmodE isn't

PCBmodE isn't a general-purpose replacement for other PCB design tools like KiCad, EAGLE, Altium, etc. For example, it does not currently have a notion of a netlist or schematics, have design rule checks.

CHAPTER 2

Workflow

PCBmodE was originally conceived as a tool that enables the designer to precisely define and position design elements in a text file, and not through a GUI. For practical reasons, *PCBmodE* does not have a GUI of its own, and uses an unmodified Inkscape for visual representation and some editing that cannot practically be done textually.

A typical *PCBmodE* design workflow is the following:

1) Edit JSON files with a text editor
2) "Compile" the board using *PCBmodE*
3) View the generated SVG in Inkscape

Then, optionally

4) Make modifications in Inkscape
5) Extract changes using *PCBmodE*

and then

6) Back to step 1 or step 2

or

7) Generate production files using *PCBmodE*

Note: It is possible to design a complete circuit in a text editor without using Inkscape at all! This would only require generating, or hand crafting, SVG paths for the routing.

Tip: Inkscape does not reload the SVG when it is regenerated by *PCBmodE*. To reload quickly, press `ALT+f` and the `v`.

Tip: Until you get used to it, the extraction process may not do what you expect, so experiment first before designing something that will disappear when you reload the SVG. It might also be practical to design in a separate Inkscape window and then copy over the shapes to the design's SVG.

CHAPTER 3

Setup

We test *PCBmodE* with Python 3.7 under Linux, but it may or may not work on other operating systems.

It comes in the form of a installable tool called `pcbmode` which is run from the command line.

3.1 What you'll need

- Python 3.7+
- Inkscape 1.0+
- Text editor

3.2 Installation in a virtual environment

Use a virtual environment to keep *PCBmodE* in its own isolated environments, for example Python3's `venv`. If you don't have `venv`, get it like this:

```
sudo apt-get install python3-venv
```

These instructions describe how to build *PCBmodE* for use in a virtual environment. To be able to build python-lxml (one of *PCBmodE*'s dependencies) you need to install some system-level development packages. On Debian based systems these are installed like this:

```
sudo apt-get install libxml2-dev libxslt1-dev python-dev
```

Note: You're reading the documentation for version 5 of *PCBmodE*, 'Cinco'. The link below will get you that branch while we're working on it, and before its release.

Get the *PCBModE* source from GitHub.

```
git clone https://github.com/boldport/pcbmode/tree/cinco-master
```

Now run these commands to create a virtual environment, for example in the directory `pcbmode-env/` next to `pcbmode/`. Then create the virtual environment like this:

```
python3 venv -m pcbmode-env
source pcbmode-env/bin/activate
cd pcbmode
```

If you want to just run *PCBmodE*, run

```
python3 setup.py install
```

but if you want to develop it, run

```
python3 setup.py develop
```

After installation, *PCBmodE* will be available in your path as an executable `pcbmode`. But since it was installed in a virtualenv, the `pcbmode` command will only be available in your path after running `source pcbmode-env/bin/activate` and will no longer be in your path after running `deactivate`, which gets you out of the virtual environment. You will need to activate the virtualenv each time you want to run `pcbmode` from a new terminal window.

Nothing is installed globally, so to start from scratch you can just follow these steps:

```
deactivate          # skip if pcbmode-env is not active
rm -r pcbmode-env
cd pcbmode
git clean -dfX      # erases any untracked files (build files etc). Save your work!
```

3.3 Running PCBmodE

Tip: To see all the options that *PCBmodE* supports, use `pcbmode --help`

By default *PCBmodE* expects to find the board files under

```
boards/<board-name>
```

relative to the place where it is invoked.

Tip: Paths where *PCBmodE* looks for things can be changed in the config file `pcbmode_config.json`.

Here’s one way to organise the build environment

```
beautiful-pcbs/
  pcbmode-env/
  pcbmode/
  boards/
    my-board/                 # a PCB project
      my-board.json
      my-board_routing.json
```

(continues on next page)

(continued from previous page)

```
      components/
      shapes/
      docs/
        ...
    cordwood/                 # another PCB project
      ...
```

To make the `my-board` board, run *PCBmodE* within `beautiful-pcbs`

```
pcbmode -b my-board -m
```

Then open the SVG with Inkscape

```
inkscape beautiful-pcbs/boards/my-board/build/my-board.svg
```

If the SVG opens you're good to go!

Note: *PCBmodE* processes a lot of shapes on the first time it is run, so it will take a noticeable amount. This time will be dramatically reduced on subsequent invocations since *PCBmodE* caches the shapes in a datafile within the project's build directory.

CHAPTER 4

Components

Components are the building blocks of the board. In fact, they are used for placing any element on board, except for routes. A via is a 'component', and a copper pour is defined within a 'component' and then instantiated into the board's JSON file.

4.1 Defining components

Components are defined in their own JSON file. The skeleton of this file is the following

```
{
  "pins":
  {
  },
  "layout":
  {
    "silkscreen":
    {
    },
    "assembly":
    {
    }
  },
  "pads":
  {
  }
}
```

`pins` is where the pins of a components are 'instantiated'. `pads` contain what pads or pins are in terms of their shapes and drills. Each 'pin' instantiates a 'pad' from `pads''. ``layout` contain silkscreen and assembly shapes.

4.1.1 pins

Here's what a component with two pins looks like

```
{
  "pins":
  {
  "1":
    {
      "layout":
      {
        "pad": "pad",
        "location": [-1.27, 0],
        "show-label": false
      }
    },
  "2-TH":
    {
      "layout":
      {
        "pad": "pad",
        "location": [1.27, 0],
        "label": "PWR",
        "show-label": true
      }
    }
  }
}
```

Each pin has a unique key – `1` and `2-TH` above – that does not necessarily need to be a number. `pad` instantiates the type of landing pad to use, which is defined in the `pads'' section. ``location` is the position of the pin relative to the *centre of the component*.

PCBmodE can discreetly place a label at the centre of the pin (this is viewable when zooming in on the pin). The label can be defined using `label`, or if `label` is missing, the key will be used instead. To not place the label use `"show-label": false`.

4.1.2 pads

Pads define the shape of pins. Here's a definition for a simple throughole capacitor

```
{
  "pins": {
    "1": {
      "layout": {
        "pad": "th-sq",
        "location": [-2, 0]
      }
    },
    "2": {
      "layout":
      {
        "pad": "th",
        "location": [2, 0]
      }
    }
```

(continues on next page)

(continued from previous page)

```
  },
  "layout": {
    "silkscreen": {
      "shapes": [
        {
          "type": "path",
          "value": "m -10.515586,19.373448 c -0.214789,0.0199 -0.437288,0.01645 -0.
↪664669,-0.0017 m -0.514055,0.01247 c -0.202682,0.02292 -0.412185,0.02382 -0.626017,
↪0.01069 m 1.56129,1.209208 c -0.557685,-0.851271 -0.665205,-1.634778 -0.04126,-2.
↪443953 m -0.82831,2.449655 c -0.07502,-0.789306 -0.06454,-1.60669 1.98e-4,-2.441891
↪",
          "location": [0, 0],
          "style": "stroke"
        }
      ]
    },
    "assembly": {
      "shapes": [
        {
          "type": "rect",
          "width": 2.55,
          "height": 1.4
        }
      ]
    }
  },
  "pads": {
    "th": {
      "shapes": [
        {
          "type": "circle",
          "layers": ["top", "bottom"],
          "outline": 0,
          "diameter": 1.9,
          "offset": [0, 0]
        }
      ],
      "drills": [
        {
          "diameter": 1
        }
      ]
    },
    "th-sq": {
      "shapes": [
        {
          "type": "rect",
          "layers": ["top", "bottom"],
          "width": 1.9,
          "height": 1.9,
          "offset": [0, 0],
          "radii": { "tl": 0.3,"bl": 0.3,"tr": 0.3,"br": 0.3 }
        }
      ],
      "drills": [
        {
          "diameter": 1
```

(continues on next page)

(continued from previous page)

```
          }
        ]
      }
    }
}
```

This would result in this component

Here's a more complex footprint for a battery holder on an ocean-themed board

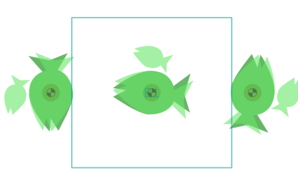
```
{
  "pins": {
    "POS-1": {
      "layout":
      {
        "pad": "pad",
        "location": [13.3, 0],
        "rotate": 95
      }
    },
    "NEG": {
      "layout": {
        "pad": "pad",
        "location": [0, 0]
      }
    },
    "POS-2": {
      "layout": {
        "pad": "pad",
        "location": [-13.3, 0],
        "rotate": -95
      }
    }
  },
  "layout": {
    "assembly": {
      "shapes": [
        {
          "type": "rect",
          "layers": ["top"],
          "width": 21.1,
          "height": 19.9,
          "offset": [0, 0]
        }
      ]
    }
  },
  "pads": {
    "pad": {
      "shapes": [
      {
        "type": "path",
        "style": "fill",
        "scale": 1,
        "layers": ["top"],
```

(continues on next page)

(continued from previous page)

```
        "value": "M 30.090397,29.705755 28.37226,29.424698 c 0,0 2.879054,-2.288897 4.991896,-2.270979 2.611383,0.02215 2.971834,2.016939 2.971834,2.016939 l 2.261927,-1.675577 -0.816738,2.741522 0.747218,2.459909 -2.119767,-1.518159 c 0,0 -0.605255,1.760889 -3.359198,1.739078 C 31.737346,32.90704 28.38105,30.56764 28.38105,30.56764 z",
        "soldermask": [
          {
            "type": "path",
            "style": "fill",
            "scale": 1,
            "rotate": 10,
            "layers": ["top"],
            "value": "M 30.090397,29.705755 28.37226,29.424698 c 0,0 2.879054,-2.288897 4.991896,-2.270979 2.611383,0.02215 2.971834,2.016939 2.971834,2.016939 l 2.261927,-1.675577 -0.816738,2.741522 0.747218,2.459909 -2.119767,-1.518159 c 0,0 -0.605255,1.760889 -3.359198,1.739078 C 31.737346,32.90704 28.38105,30.56764 28.38105,30.56764 z"
          },
          {
            "type": "path",
            "style": "fill",
            "scale": 0.5,
            "rotate": 20,
            "location": [0, 4.7],
            "layers": ["top"],
            "value": "M 30.090397,29.705755 28.37226,29.424698 c 0,0 2.879054,-2.288897 4.991896,-2.270979 2.611383,0.02215 2.971834,2.016939 2.971834,2.016939 l 2.261927,-1.675577 -0.816738,2.741522 0.747218,2.459909 -2.119767,-1.518159 c 0,0 -0.605255,1.760889 -3.359198,1.739078 C 31.737346,32.90704 28.38105,30.56764 28.38105,30.56764 z"
          }
        ]
      },
      {
        "type": "circle",
        "layers": ["bottom"],
        "outline": 0,
        "diameter": 2.3,
        "offset": [0, 0]
      }
    ],
    "drills": [
      {
        "diameter": 1.2
      }
    ]
    }
  }
}
```

This will what it looks like



Notice that you can define multiple shapes for the soldermask that are independent of the shape of the shape of the copper.

To control how soldermask shapes are placed, you have the following options:

- No `soldermask` definition will assume default placement. The buffers and multipliers are defined in the board's JSON file
- `"soldermask": []` will not place a soldermask shape
- `"soldermask": [{...},{...},...]` as above will place custom shapes

Defining custom solderpaste shapes works in exactly the same way except that you'd use `soldepaste` instead of `soldermask`.

4.1.3 layout shapes

4.2 Placing components and shapes

Footprints for components and shapes are stored in their own directories within the project path (those can be changed in the configuration file).

This is an example of instantiating a component within the board's JSON file

```
{
  "components":
  {
    "J2":
    {
      "footprint": "my-part",
      "layer": "top",
      "location": [
        36.7,
        0
      ],
      "rotate": -90,
      "show": true,
      "silkscreen": {
        "refdef": {
          "location": [
            -7.2,
            2.16
          ],
          "rotate": 0,
          "rotate-with-component": false,
          "show": true
        },
      "shapes": {
        "show": true
        }
      }
    }
  }
}
```

The key of each component – `J2` above – record is the component's reference designator, or in *PCBmodE*-speak, 'refdef'. Note that as opposed to `shape` types, here `layer` can only accept one layer.

`silkscreen` is optional, but allows control over the placement of the reference designator, and whether shapes are placed or not.

Note: The sharp-minded amongst you will notice that 'refdef' is not exactly short form of 'reference designator'. I noticed that fact only in version 3.0 of *PCBmodE*, way too far to change it. So I embraced this folly and it will forever be.

CHAPTER 5

Shapes

Shapes are the basic building blocks of *PCBmodE*. Here's an example of a shape type `path`:

```
{
  "type": "path",
  "layers": ["bottom"],
  "location": [3.1, -5.667],
  "stroke-width": 1.2,
  "style": "stroke",
  "value": "m -48.3,0 0,-5.75 c 0,-1.104569 0.895431,-2 2,-2 0,0 11.530272,-0.555504 17.300001,-0.5644445 10.235557,-0.015861 20.4577816,0.925558 30.6933324,0.9062128 C 10.767237,-7.4253814 19.826085,-8.3105055 28.900004,-8.3144445 34.703053,-8.3169636 46.3,-7.75 46.3,-7.75 c 1.103988,0.035813 2,0.895431 2,2 l 0,5.75 0,5.75 c 0,1.104569 -0.895431,2 -2,2 0,0 -11.596947,0.5669636 -17.399996,0.5644445 C 19.826085,8.3105055 10.767237,7.4253814 1.6933334,7.4082317 -8.5422174,7.3888865 -18.764442,8.3303051 -28.999999,8.3144445 -34.769728,8.305504 -46.3,7.75 -46.3,7.75 c -1.103982,-0.036019 -2,-0.895431 -2,-2 l 0,-5.75"
}
```

This will place an SVG path as a `stroke` with width `1.2 mm` at location `x=3.1` and `y=5.667`. The shape will be placed on the bottom layer of the PCB.

5.1 Shape types

For each shape a `type` must be defined. Below are the available shapes.

5.1.1 Rectangle

Below is an example of a filled rectangle with rounded corners except for the top left corner.

```
{
  "type": "rect",
  "layers": ["top"],
  "width": 1.7,
  "height": 1.7,
  "location": [6, 7.2],
  "radii": {"tl": 0,
            "tr": 0.3,
            "bl": 0.3,
            "br": 0.3},
  "rotate": 15,
  "style": "fill"
}
```

type `rect`: place a rectangle

layers (optional; default `["top"]`) list: layers to place the shape on (even if placing on a single layer, the definition needs to be in a form of a list)

width float: width of the rectangle

height float: height of the rectangle

location (optional; default `[0,0]`) list: `x` and `y` coordinates for where to place the shape

radii (optional) dict: radius of round corners `tl`: top left radius, `tr`: top right radius, `bl`: bottom left radius, `br`: bottom right radius,

rotate (optional; default `0`) float: rotation, clock-wise degrees

style (optional; default depends on sheet) `stroke` or `fill`: style of the shape

stroke-width (optional; default depends on sheet; ignored unless `style` is `stroke`) float: stroke width

buffer-to-pour (optional; defaults to global setting) float: custom buffer from shape to copper pour; 0 for no buffer

5.1.2 Circle

Below is an example of a circle outline of diameter 1.7 mm and stroke width of 0.23 mm

```
{
  "type": "circle",
  "layers": ["bottom"],
  "location": [-3.2, -6],
  "diameter": 1.7,
  "style": "stroke"
  "stroke-width": 0.23
}
```

type `circle`: place a circle

layers (optional; default `["top"]`) list: layers to place the shape on (even if placing on a single layer, the definition needs to be in a form of a list)

location (optional; default `[0,0]`) list: `x` and `y` coordinates for where to place the shape

diameter float: diameter of circle

style (optional; default depends on sheet) `stroke` or `fill`: style of the shape

stroke-width (optional; default depends on sheet; ignored unless `style` is `stroke`) float: stroke width

buffer-to-pour (optional; defaults to global setting) float: custom buffer from shape to copper pour; 0 for no buffer

5.1.3 Path

Other than simple shapes above, and SVG path can be placed.

```
{
  "type": "path",
  "layers": ["top","bottom"],
  "location": [3.1, 5.667],
  "stroke-width": 1.2,
  "style": "stroke",
  "rotate": 23,
  "scale": 1.2,
  "value": "m -48.3,0 0,-5.75 c 0,-1.104569 0.895431,-2 2,-2 0,0 11.530272,-0.555504␣
↪17.300001,-0.5644445 10.235557,-0.015861 20.4577816,0.925558 30.6933324,0.9062128 C␣
↪10.767237,-7.4253814 19.826085,-8.3105055 28.900004,-8.3144445 34.703053,-8.3169636␣
↪46.3,-7.75 46.3,-7.75 c 1.103988,0.035813 2,0.895431 2,2 l 0,5.75 0,5.75 c 0,1.
↪104569 -0.895431,2 -2,2 0,0 -11.596947,0.5669636 -17.399996,0.5644445 C 19.826085,8.
↪3105055 10.767237,7.4253814 1.6933334,7.4082317 -8.5422174,7.3888865 -18.764442,8.
↪3303051 -28.999999,8.3144445 -34.769728,8.305504 -46.3,7.75 -46.3,7.75 c -1.103982,-
↪0.036019 -2,-0.895431 -2,-2 l 0,-5.75"
}
```

type `path`: place an SVG path

value path: in SVG this is the `d` property of a `<path>`

layers (optional; default `["top"]`) list: layers to place the shape on (even if placing on a single layer, the definition needs to be in a form of a list)

location (optional; default `[0,0]`) list: `x` and `y` coordinates for where to place the shape

diameter float: diameter of circle

style (optional; default depends on sheet) `stroke` or `fill`: style of the shape

stroke-width (optional; default depends on sheet; ignored unless `style` is `stroke`) float: stroke width

rotate (optional; default `0`) float: rotation, clock-wise degrees

scale (optional; default `1`) float: scale factor to apply to the path

buffer-to-pour (optional; defaults to global setting) float: custom buffer from shape to copper pour; 0 for no buffer

5.1.4 Text

Placing a text shape is covered in *Text*.

CHAPTER 6

Copper pours

A copper pour covers the surface area of a board with copper while maintaining a certain buffer from other copper features, such as routes and pads. A 'bridge' can connect between a copper feature and a pour.

6.1 Defining pours

Pours are defined in their own section in the board's JSON under `shapes`

```
{
  "shapes": {
    "pours":
    [
      {
        "layers": [
          "bottom",
          "top"
        ],
        "type": "layer"
      }
    ]
  }
}
```

The above will place a pour over the entire top and bottom layer of the board. It's possible to pour a specific shape, and that's done just like any other shape definition.

> **Warning:** Since *PCBmodE* does not have a netlist, those bridges need to be added manually, and careful attention needs to be paid to prevent shorts – there's no DRC!

Tip: Even if you're pouring over a single layer, the `layers` definition only accepts a list, so you'd use `["bottom"]`, not `"bottom"`.

6.2 Defining buffers

The global settings for the buffer size between the pour and a feature is defined in the board's JSON file, as follows:

```
"distances": {
  "from-pour-to": {
    "drill": 0.4,
    "outline": 0.25,
    "pad": 0.4,
    "route": 0.25
  }
}
```

If this block, or any of its definitions, is missing, defaults will be used.

These global settings can be overridden for every shape and route. For routes, it's done using the `pcbmode:buffer-to-pour` definition, as described in *Routing*. For shapes it's done using the `buffer-to-pour` definition, as described in *Shapes*.

CHAPTER 7

Text

One of the unique features of *PCBmodE* is that any font – as long as it is in SVG form – can be used for any text on the board.

7.1 Fonts

SVG fonts have an SVG path for every glyph, and other useful information about how to place the font so the glyphs align. *PCBmodE* uses that information to place text on the board's layers.

The folder in which *PCBmodE* looks for a font is defined in the the configuration file `pcbmode_config.json`.

```
{
  "locations":
  {
    "boards": "boards/",
    "components": "components/",
    "fonts": "fonts/",
    "build": "build/",
    "styles": "styles/"
  }
}
```

When looking for a font file, *PCBmodE* will first look at the local project folder and then where `pcbmode.py` is.

Tip: When you find a font that you'd like to use, search for an SVG version of it. Many fonts at http://www.fontsquirrel.com have an SVG version for download.

7.2 Defining text

A text definition looks like the following

```
{
  "type": "text",
  "layers": ["bottom"],
  "font-family": "Overlock-Regular-OTF-webfont",
  "font-size": "1.5mm",
  "letter-spacing": "0mm",
  "line-height": "1.5mm",
  "location": [
    -32.39372,
    -33.739699
  ],
  "rotate": 0,
  "style": "fill",
  "value": "Your text\nhere!"
}
```

type `text`: place a text element

layers (optional; default `["top"]`) list: layers to place the shape on (even if placing on a single layer, the definition needs to be in a form of a list)

font-family text: The name of the font file, without the `.svg`

font-size float: font size in mm (the `mm` must be present)

value text: the text to display; use `\n` for newline

letter-spacing (optional; default `0mm`) float: positive/negative value increases/decreases the spacing. `0mm` maintains the natural spacing defined by the font

line-height (optional; defaults to `font-size`) float: the distance between lines; a negative value is allowed

location (optional; default `[0, 0]`) list: `x` and `y` to place the *center* of the text object

rotate (optional; default `0`) float: rotation, clock-wise degrees

style (optional; default depends on sheet) `stroke` or `fill`: style of the shape

stroke-width (optional; default depends on sheet; ignored unless `style` is `stroke`) float: stroke width

CHAPTER 8

Routing

Routing, of course, is an essential part of a circuit board. *PCBmodE* does not have an auto-router, and routing is typically done in Inkscape, although theoretically, routing can be added manually in a text editor. All routing shapes reside in the routing SVG layer of each PCB layer.

Important: Make sure that you place the routes and vias on the routing SVG layer of the desired PCB layer. To choose that layer either click on an element in the layer or open the layer pane by pressing `CTRL+SHIFT+L`.

Important: In order to place routes, make sure that Inkscape is set to 'optimise' paths by going to `File->Inkscape Preferences->Transforms` and choosing `optimised` under `Store transformation`.

8.1 Adding routes

Choose the desired routing SVG layer. Using the Bezier tool (`SHIFT+F6`) to draw a shape.

For a filled shape, make sure that it is a closed path and in the `Fill and stroke` pane (`SHIFT+CTRL+F`) click on the `flat color` button on the `Fill` tab, and the `No paint` (marked with an `X`) on the `Stroke point` tab.

For a stroke, in the `Fill and stroke` pane (`SHIFT+CTRL+F`) click on the `No paint` button on the `Fill` tab, and the `Flat color` on the `Stroke point` tab. Adjust the stroke thickness on the `Stroke style` tab.

Note: Shapes can be either stroke or fill, not both. If you'd like a filled and stroked shape, you'll need to create two shapes.

Finally, you *must* move the shape with the mouse or with the arrows.

Note: When creating a new shape Inkscape adds a matrix transform, which is removed when the shape is moved because of the `optimise` settings as described above. This minor inconvenience is a compromise that greatly simplifies the extraction process.

If the route is placed where there is a copper pour, it will automatically have a buffer around it that's defined in the board's configuration. Sometimes, it is desirable to reduce or increase this buffer, or eliminate it completely in order to create a bridge (for example when connecting a via to a pour). This is how it is done:

1) Choose the route
2) Open Inkscape's XML editor (`SHIFT+CTRL+X`)
3) On the bottom right, next to `set` remove what's there and type in `pcbmode:buffer-to-pour`
4) In the box below type in the buffer in millimeters (don't add 'mm') that you'd like, or `0` for none
5) Press `set` or `CTRL+ENTER` to save that property

Tip: Once you've created one route, you can simply cut-and-paste it and edit it using the node tool without an additional settings. You can even cut-and-paste routes from a different design.

8.2 Adding vias

Vias are components just like any other. There are placed just like other components, but in the routing file "<design_name>_routing.json", not the main board's JSON.

You can assign a unique key to the via, but that will be over-written by a hash when extracted.

Note: Since vias are components, anything could be a via, so if it makes sense to place a 2x2 0.1" header as a "via", that's possible.

Important: Don't forget to extract the changes!

CHAPTER 9

Extraction

One of the common steps of the *PCBmodE* workflow is extracting information from the SVG and storing it in primary JSON files.

The following will be extracted from the SVG:

- Routing shapes and location
- Vias' location
- Components' location and rotation
- Documentation elements' location
- Drill index location

That's it.

Note: It's quite likely that more information will be extracted in the future to make the design process require fewer steps. Architecturally, however, the use of a GUI is meant only to assist the textual design process, not replace it.

Other information needs to be entered manually with a text editor. A great tool in this process is Inkscape's built-in XML editor (open with `SHIFT+CTRL+X`) which allows you to see the path definition of shape (the `d` property) and copy it over to the JSON file.

Tip: Since some shapes (pours, silkscreen, etc.) are not extracted, it's sometimes a bit of a guesswork to get the location just right. To do that in a single iteration, use the XML editor to change the transform of the shape (press `CTRL+ENTER` to apply) until the position is right. Then copy over the coordinates for that shape to the JSON file. **Note** that Inkscape inverts the y-axis coordinate, so when entering it into the JSON invert it back.

CHAPTER 10

Layer control

When opening a *PCBmodE* SVG in Inkscape, the board's layers can be manipulated by opening the layer pane (CTRL+SHIFT+L). Each layer can then be set to be hidden/visible or editable/locked. The default for each layer is defined in utils/svg.py

```
layer_control = {
  "copper": {
    "hidden": False, "locked": False,
    "pours": { "hidden": False, "locked": True },
    "pads": { "hidden": False, "locked": False },
    "routing": { "hidden": False, "locked": False }
  },
  "soldermask": { "hidden": False, "locked": False },
  "solderpaste": { "hidden": True, "locked": True },
  "silkscreen": { "hidden": False, "locked": False },
  "assembly": { "hidden": False, "locked": False },
  "documentation": { "hidden": False, "locked": False },
  "dimensions": { "hidden": False, "locked": True },
  "origin": { "hidden": False, "locked": True },
  "drills": { "hidden": False, "locked": False },
  "outline": { "hidden": False, "locked": True }
}
```

but can be overridden in the board's configuration file. So, for example, if we wish to have the solderpaste layers visible when the SVG is generated, we'd add

```
{
  "layer-control":
  {
    "solderpaste": { "hidden": false, "locked": true }
  }
}
```

Or if we'd like the outline to be editable (instead of the default 'locked') we'd add

```
{
  "layer-control":
  {
    "solderpaste": { "hidden": false, "locked": true },
    "outline": { "hidden": false, "locked": false }
  }
}
```

Tip: The reason that some layers are locked by default – 'outline' is a good example – is because they are not edited regularly, but span the entire board so very often take focus when selecting objects. Locking them puts them out of the way until an edit is required.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search