

# LibrePCB User Manual

## Table of Contents

1. About LibrePCB	2
1.1. Features	3
1.2. Project Status	3
2. Getting Started With LibrePCB	4
2.1. Installation	4
2.1.1. Online Installer	4
2.1.2. Portable Package	5
2.1.3. Other	5
2.2. Create a Workspace	6
2.3. Download Remote Libraries	6
2.4. Create Local Libraries	8
2.4.1. Component Category	11
2.4.2. Symbol	12
2.4.3. Component	13
2.4.4. Package Category	18
2.4.5. Package	19
2.4.6. Device	21
2.5. Create a New Project	23
2.6. Create Schematics	26
2.7. Create Boards	28
2.7.1. Draw Outlines	28
2.7.2. Place Devices	30
2.8. Order PCB	31
2.8.1. LibrePCB Fab	31
2.8.2. Generate Production Data	33
3. Library Conventions	34
3.1. Symbol Conventions	34
3.1.1. Generic vs. Specific	35
3.1.2. Naming	35
3.1.3. Origin	36
3.1.4. Outline	36
3.1.5. Pin Placement	36
3.1.6. Pin Naming	36
3.1.7. Text Elements	36
3.1.8. Grab Area	37
3.2. Package Conventions	38

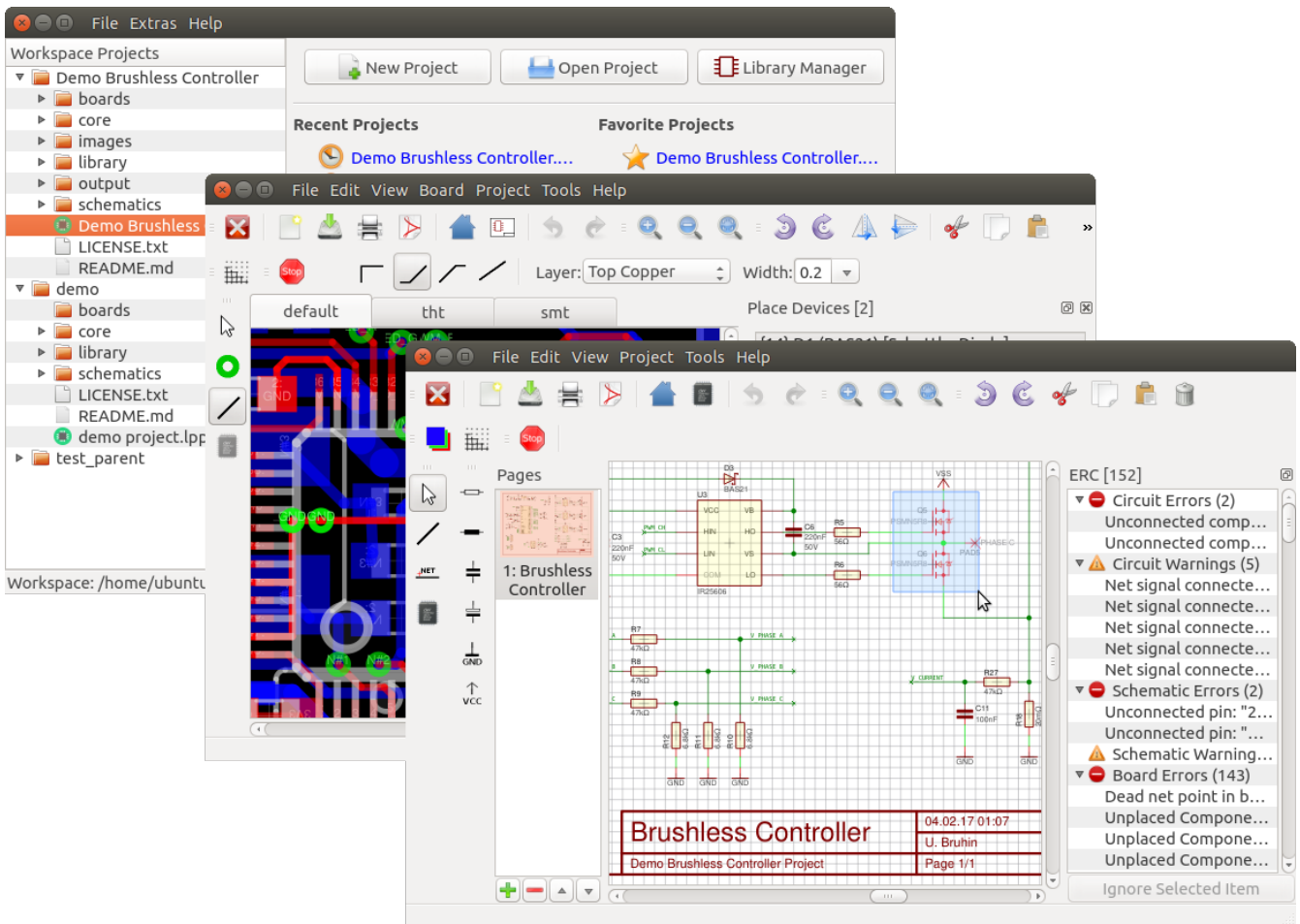
3.2.1. Scope .....	38
3.2.2. Naming .....	39
3.2.3. Pads .....	39
3.2.4. Footprint Variants .....	39
3.2.5. Origin .....	41
3.2.6. Orientation .....	41
3.2.7. Placement Layer .....	41
3.2.8. Documentation Layer .....	42
3.2.9. Text Elements .....	43
4. LibrePCB Command Line Interface .....	45
4.1. Installation .....	45
4.2. Running on Headless Linux .....	46
4.3. Usage .....	46
4.3.1. Command <code>open-library</code> .....	47
4.3.2. Command <code>open-project</code> .....	48

# 1. About LibrePCB

Welcome to the documentation of [LibrePCB](#)! LibrePCB is a free EDA software to develop printed circuit boards. It runs on Linux, Windows and Mac.



This documentation is currently under construction. Contributions are welcome!  
Check out <https://github.com/LibrePCB/librepcb-doc>



## 1.1. Features

- Cross-platform (Unix/Linux, Mac OS X, Windows)
- All-In-One: project management + library/schematic/board editors
- Intuitive, modern and easy-to-use graphical user interface
- Very powerful library design with some innovative concepts
- Human-readable file formats for both libraries and projects
- Multi-PCB feature (different PCB variants of the same schematic)
- Automatic netlist synchronisation between schematic and board

## 1.2. Project Status

LibrePCB is still under development and many features are not implemented yet. So here's an overview about the project status:

### What's working

- Downloading, creating and modifying libraries
- Drawing simple schematics with multiple sheets (incl. print or PDF export)
- Creating simple multilayer PCBs (incl. DRC and Gerber/Excellon/Pick&Place/PDF export)

- Exporting bill of materials (BOM)

## What's missing

- No hierarchical schematics [\[wishlist\]](#)
- No buses in schematics [\[wishlist\]](#)
- No 3D board viewer [\[wishlist\]](#)

Of course all these features will be implemented in future. Please consider to [contribute](#) or [donate](#) to support the development of LibrePCB!

# 2. Getting Started With LibrePCB

This chapter provides a quick introduction into LibrePCB, starting from where to download it and ending with how to generate PCB production data.

## 2.1. Installation



### *Disclaimer*

Please note that LibrePCB is still in early development and thus isn't as mature as other EDA tools. Use LibrePCB at your own risk, accepting that we don't take responsibility for any issues caused by software bugs or other reasons. For details, please read the full [license text](#).



### *Note for macOS users*

Because Apple doesn't provide the ability to run macOS without purchasing their hardware, we're not able to test LibrePCB on macOS. Don't hesitate to [open an issue](#) if LibrePCB doesn't work as expected.

### 2.1.1. Online Installer

The recommended way to install LibrePCB is to use the online installer. It provides the following features:

- Downloads the latest version from the Internet (no offline installation possible).
- Installs a maintenance tool to easily download and install updates.
- Creates start menu entries for LibrePCB and the maintenance tool.
- Optionally registers \*.lpp files, so LibrePCB projects can be opened with a double-click in the file manager.

### Windows

Download and run [librepcb-installer-0.1.7-windows-x86.exe](#).

## Linux

Download [librepcb-installer-0.1.7-linux-x86\\_64.run](#), make it executable and run it:

```
wget "https://download.librepcb.org/releases/0.1.7/librepcb-installer-0.1.7-linux-x86_64.run"
chmod +x ./librepcb-installer-0.1.7-linux-x86_64.run
./librepcb-installer-0.1.7-linux-x86_64.run
```

## Mac

Download and run [librepcb-installer-0.1.7-mac-x86\\_64.dmg](#). "Unverified" packages may have to be opened via right-click.

### 2.1.2. Portable Package

Alternatively you could run LibrePCB without installing it. But then you don't get an update mechanism, no start menu entries are created, and \*.lpp files will not be registered.

## Windows

Download and extract [librepcb-0.1.7-windows-x86.zip](#), then run the contained file `bin\librepcb.exe`.

## Linux

Download [librepcb-0.1.7-linux-x86\\_64.AppImage](#), make it executable and start it:

```
wget "https://download.librepcb.org/releases/0.1.7/librepcb-0.1.7-linux-x86_64.AppImage"
chmod +x ./librepcb-0.1.7-linux-x86_64.AppImage
./librepcb-0.1.7-linux-x86_64.AppImage
```

## Mac

Download [librepcb-0.1.7-mac-x86\\_64.dmg](#) and double-click it. Then drag and drop the app onto the "Applications" icon of Finder.

### 2.1.3. Other

#### FlatPak

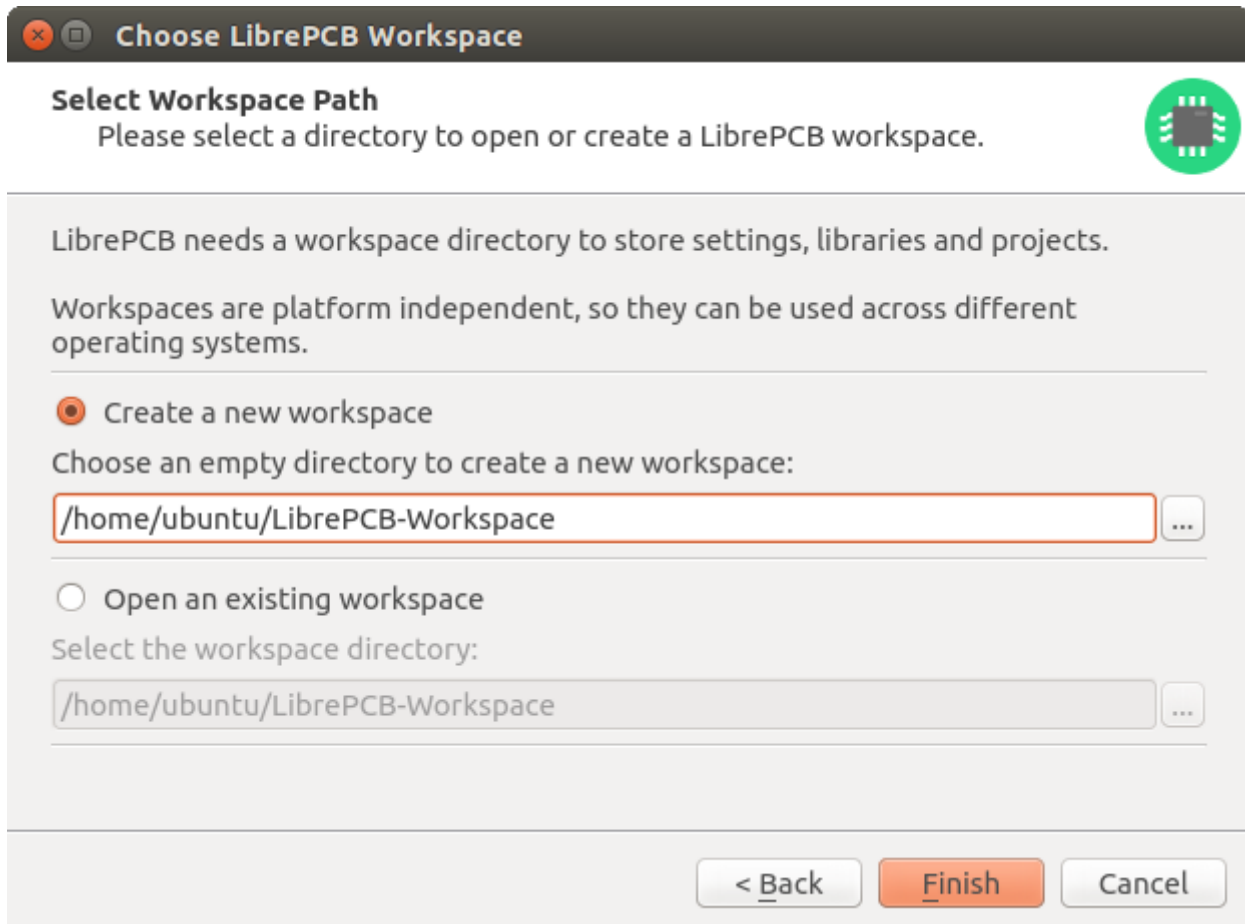
LibrePCB is also available as a [FlatPak](#) package from [FlatHub](#). Assuming you have followed [FlatPak setup](#), you can configure FlatHub and install LibrePCB as follows:

```
flatpak remote-add --if-not-exists flathub
https://flathub.org/repo/flathub.flatpakrepo
flatpak install flathub org.librepcb.LibrePCB
```

## 2.2. Create a Workspace

When starting LibrePCB the first time, a wizard asks you to open or create a Workspace. The Workspace is just a directory where settings, libraries and (optionally) projects will be stored. Once created, it can be used from all supported operating systems (i.e. it is platform independent) and in future it will also be usable with different LibrePCB versions.

You can just accept the default Workspace location (you could still move it to another location afterwards, if desired):

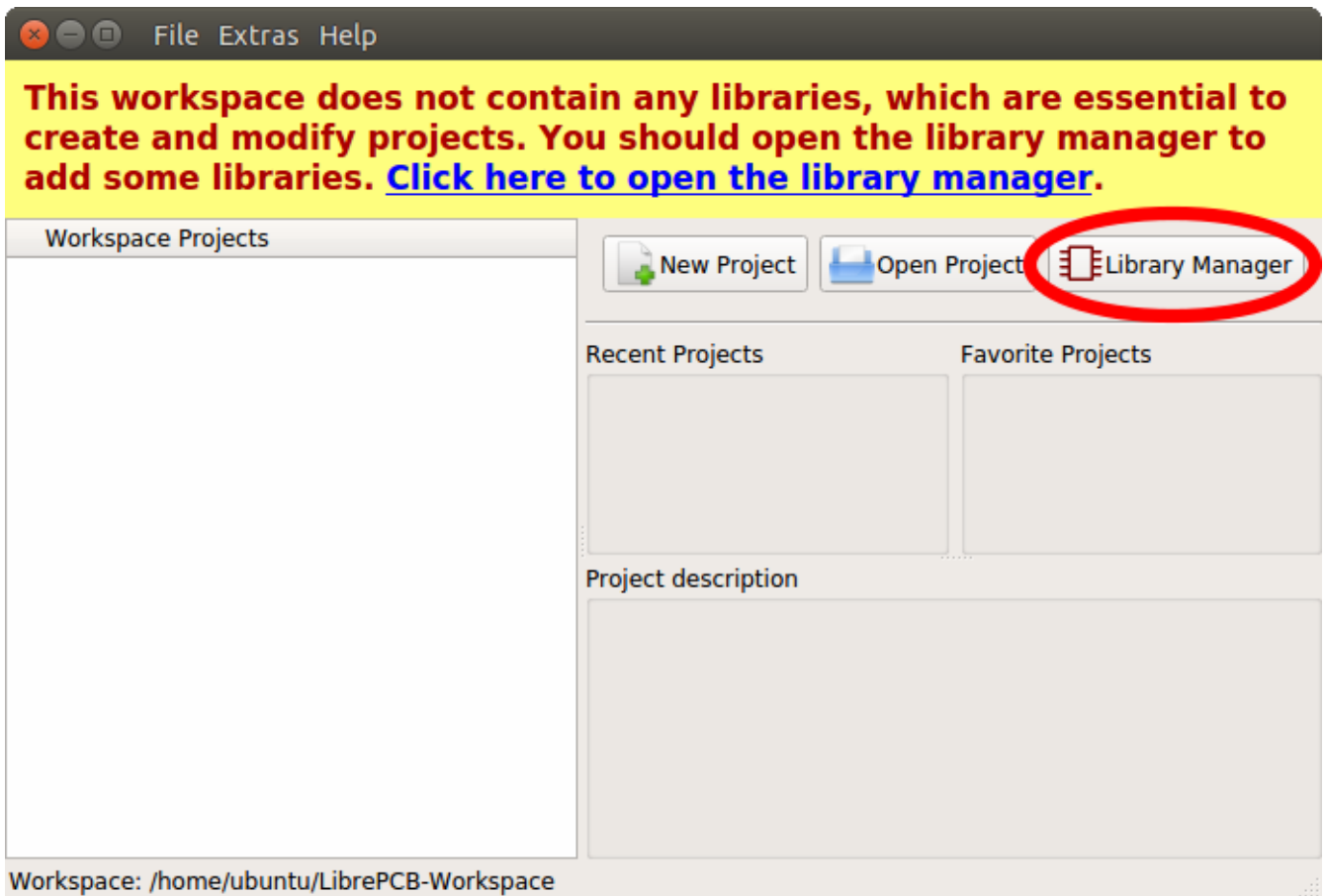


After clicking *Finish*, the Control Panel shows up and you're ready to start using LibrePCB!

## 2.3. Download Remote Libraries

Before you can start with creating new projects, you need to add some libraries to your Workspace. Libraries contain various kinds of elements which can be added to schematics and boards (e.g. symbols and footprints).

To open the Library Manager, click on the corresponding button in the Control Panel (or on the link in the shown warning, but the warning will disappear after you add a library):

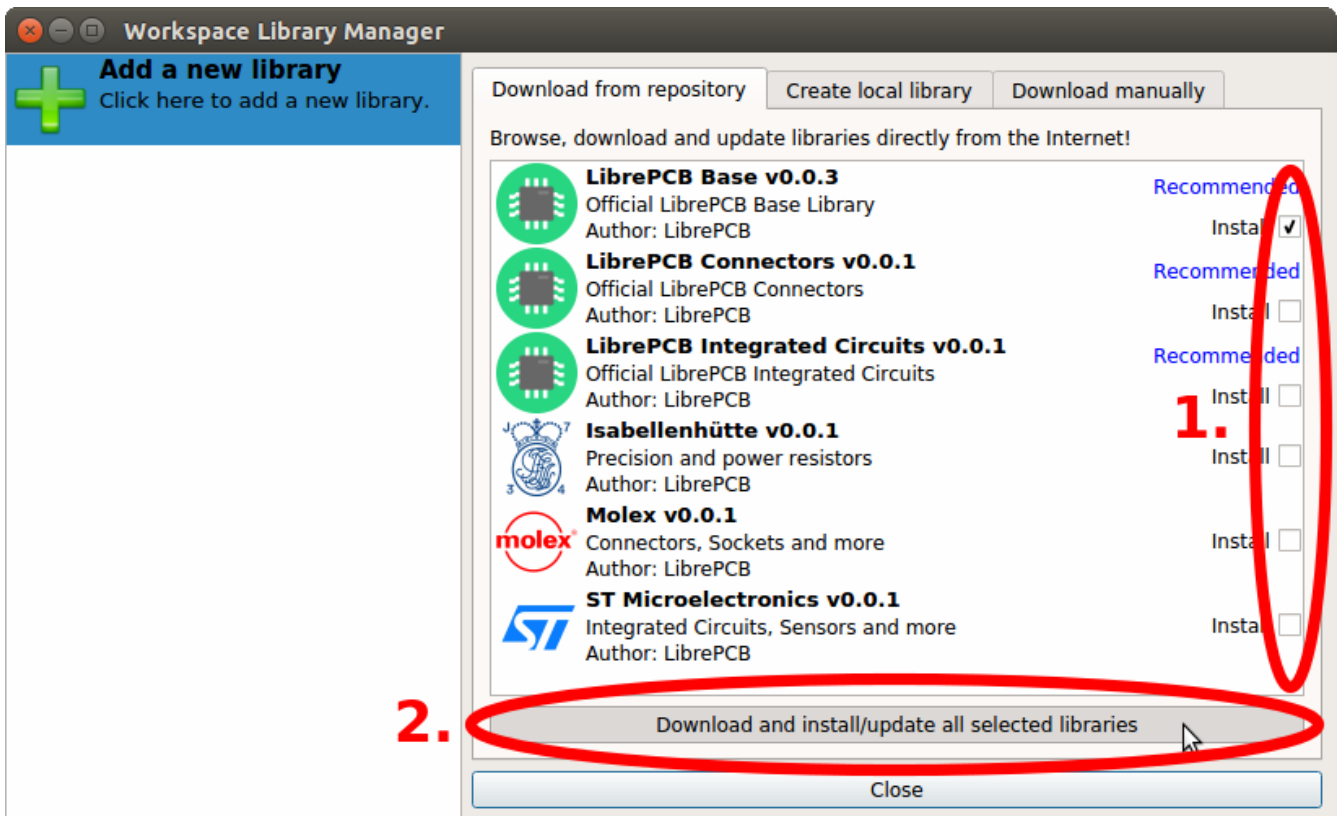


The Library Manager directly fetches the list of available libraries from the Internet. Most of these libraries are hosted at <https://github.com/LibrePCB-Libraries>.



The provided libraries currently contain only a few elements. You are welcome to contribute more elements by opening pull requests at <https://github.com/LibrePCB-Libraries>!

The most important library is the *LibrePCB Base* library because it contains the most commonly used library elements (like resistors or diodes). It is highly recommended to install this library. To do so, just select the checkboxes at the right and click on the *Download and install/update all selected libraries* button:



By the way, the same way you can also update already installed libraries to the latest version. You should regularly update all libraries to get more elements and bugfixes/improvements of existing elements.



Dependencies between different libraries are automatically taken into account when changing the selection. So for example if you select *LibrePCB Connectors*, the *LibrePCB Base Library* will automatically be selected too because the Connectors library depends on it.



Downloaded (so-called *remote-*) libraries are always read-only because otherwise local modifications could cause conflicts when updating the library the next time. But this should not be an issue, just follow the guide below to create your own local library. In a local library you can extend, or even overwrite existing library elements (by using a higher version number to enforce higher priority).

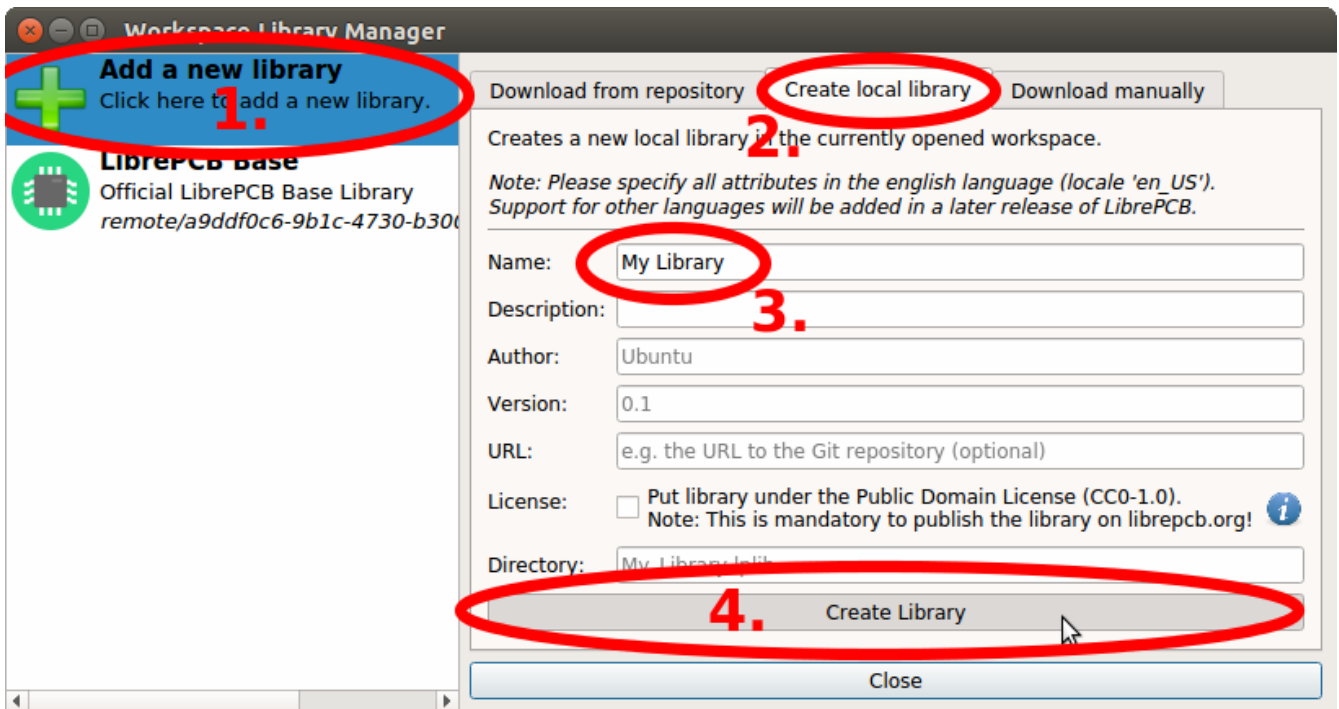


If you are familiar with Version Control Systems (e.g. *Git*) and want to use them to manage your libraries (instead of the Library Manager), just clone the libraries into the subdirectory `v0.1/libraries/local/` in your Workspace.

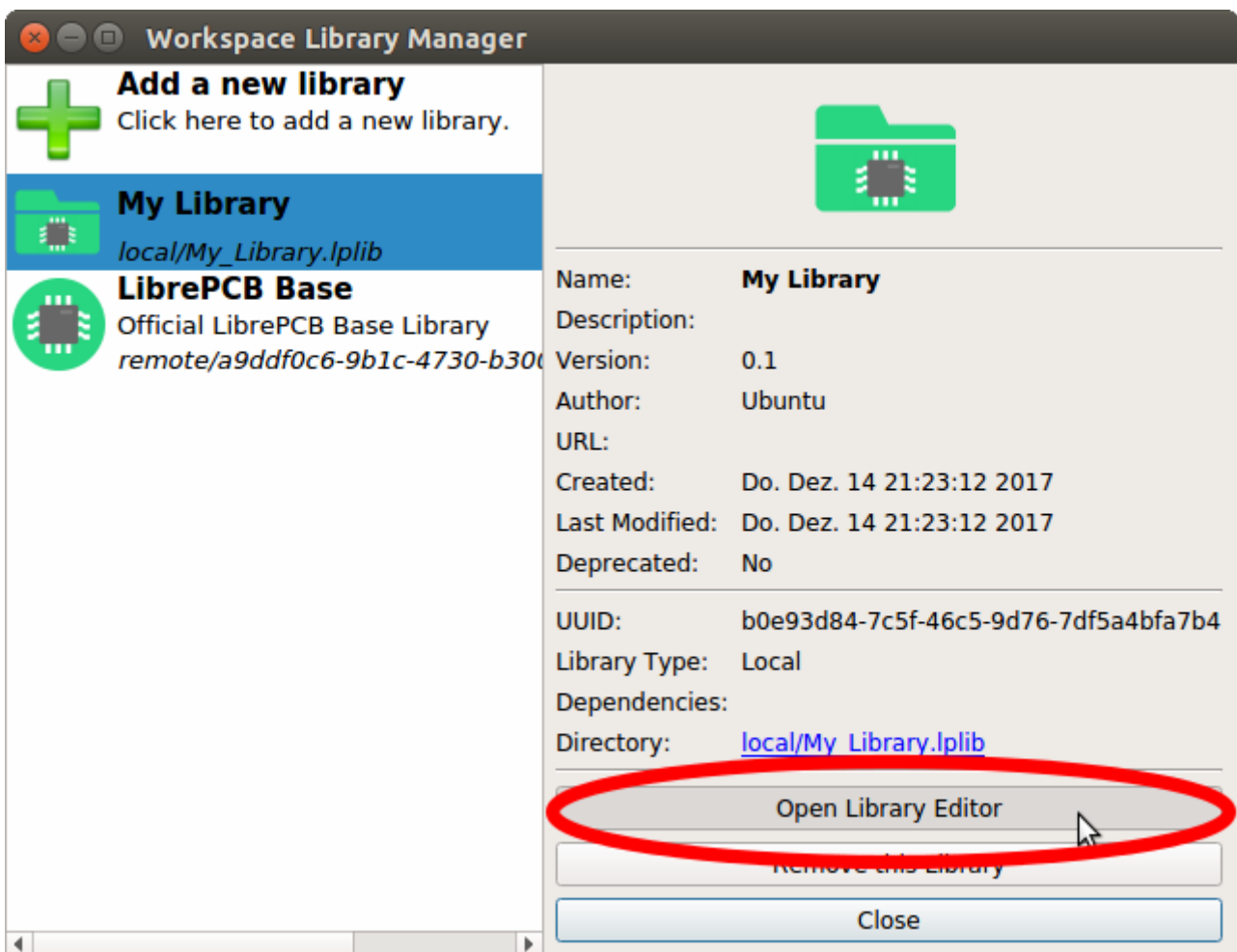
## 2.4. Create Local Libraries

Of course you can also create your own libraries. To do so, enter some metadata in the tab *Create local library* and click on *Create Library*:





Now you can open the library editor to create some symbols and footprints in your new library. Select your library on the left and then click on the *Open Library Editor* button:



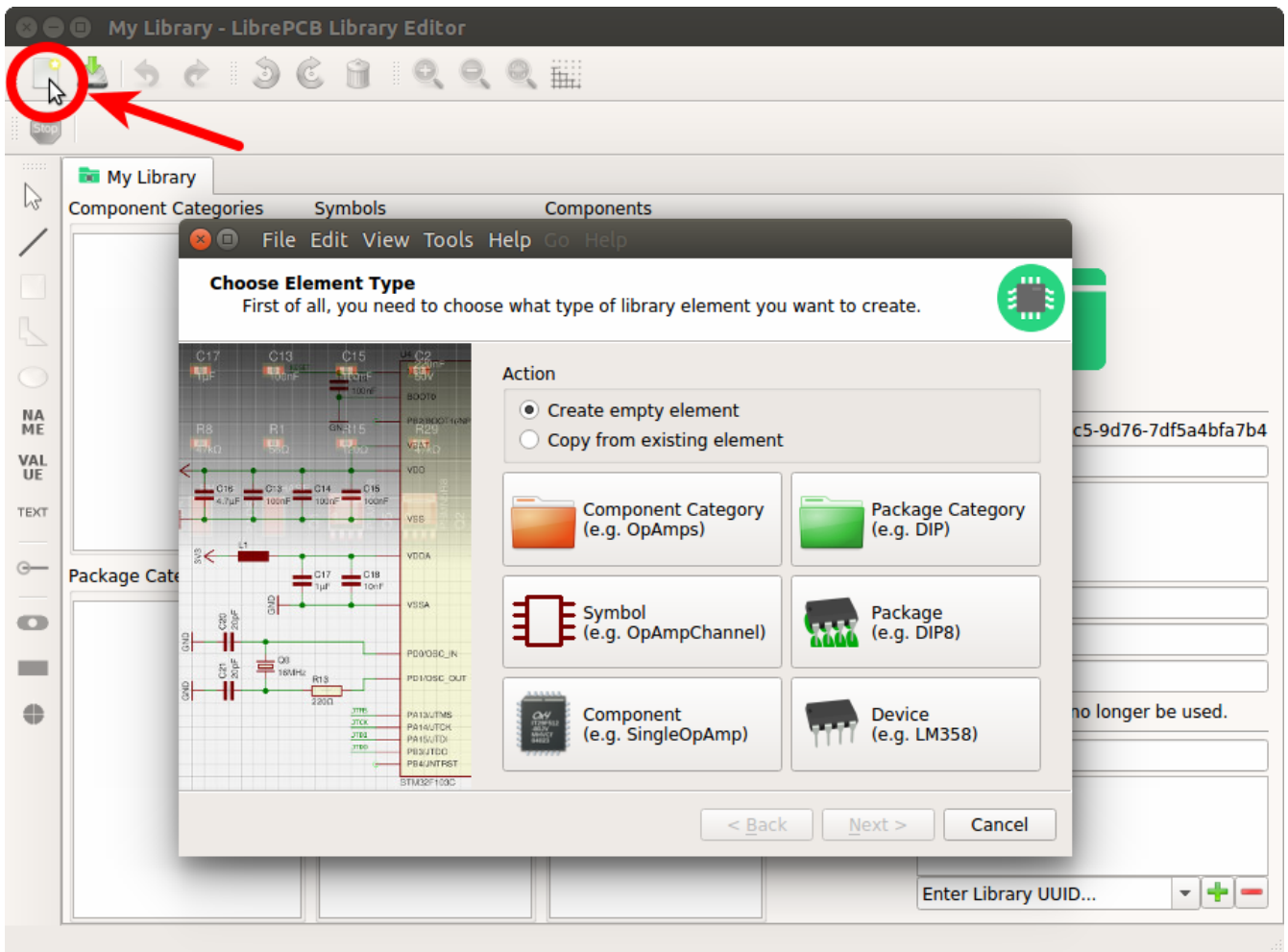
Now we need a crash course to understand the basics of LibrePCB's library concept ;) A library consists of several different elements:

- **Component Category:** These are basically "metadata-only" elements used to categorize the "real" library elements in a category tree. Every Symbol, Component and Device can be assigned to one or more categories so they will appear in the category tree. Examples: *Resistors*, *LEDs*, *Microcontrollers*
- **Symbol:** A symbol is the graphical representation of a Component (or parts of it) in a schematic. It consists of electrical pins and graphical objects like lines. Examples: *European Resistor*, *LED*, *1x10 Connector*
- **Component:** A Component basically represents a "generic" kind of electrical part. It's **not** a real part which you can buy, it's just theoretical. The Component defines the electrical interface of a part and how it is represented in the schematic (by referencing one or more Symbols). But it does not define how the part looks like in a board. Examples: *Resistor*, *Bipolar Capacitor*, *4-channel OpAmp*
- **Package Category:** Exactly the same as Component Categories, but for Packages instead of Components. Examples: *Chip Resistors*, *Axial Capacitors*, *DIP*
- **Package:** As the name suggests, packages represent the mechanical part of a "real" electronic part. It contains the footprint with their electrical pads and graphical objects which is then added to boards. Later a package may also contain a 3D model for the 3D board viewer. Examples: *TO220*, *DIP20*, *LQFP32*
- **Device:** The Device now represents a real electronic part which you can buy. It basically combines a Component with a Package to specify how a Component looks like on the board. Examples: *0805 Resistor*, *LM358D*, *STM32F103C*



The order of this list is actually also the order to follow when creating new library elements. For example a Device always needs to be created **after** the corresponding Component. The other direction is not possible because of the dependencies.

Ready to create your first library elements? At the top left of the library editor there is the entry point for every new library element. There you can choose which of the six library element type you want to create:



### Example: LMV321LILT

Let's say you want to create the part LMV321LILT from A to Z. We will create now all the necessary library elements for the LMV321LILT, though in practice you only need to create the elements which do not already exist. You can even use elements from other libraries, for example the Symbol from library A, the Component from library B and the Package from library C.

#### 2.4.1. Component Category

First you should create a Component Category for the LMV321LILT (if it doesn't already exist). Choose a suitable (generic!) name and select a parent category. You may first need to create the required parent categories.

File Edit View Tools Help

### Enter Metadata

Please specify some metadata about the new element.

Name:

Description:

Keywords:

Author:

Version:

Category:  ...

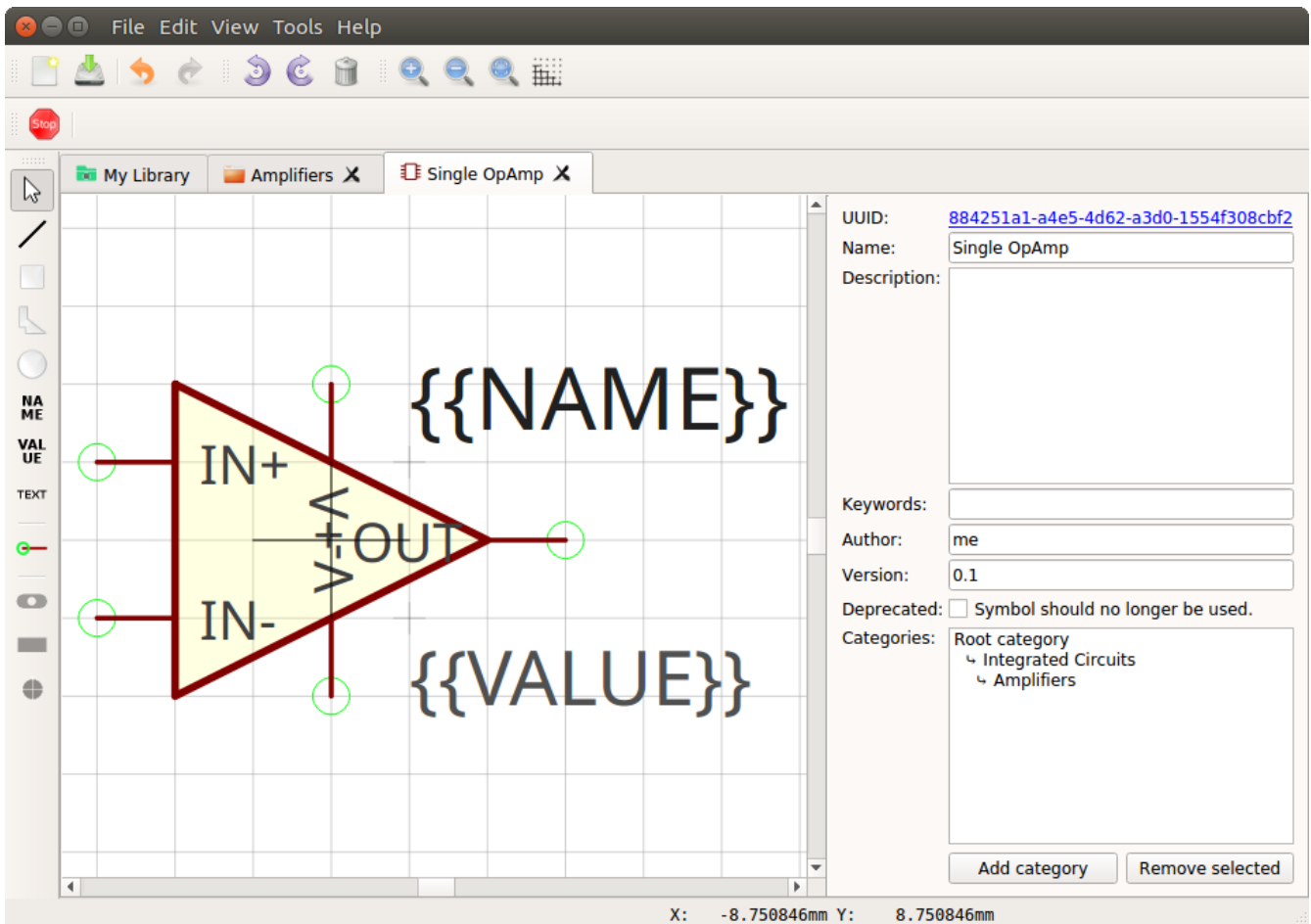
Root category => **Integrated Circuits**

< Back Finish Cancel

After clicking on *Finish*, your first Component Category is ready to be used :)

## 2.4.2. Symbol

Now we need to create a Symbol for the OpAmp. Click on the *Symbol* button in the *New Library Element* wizard, fill in some metadata and click on *Finish*. Now you can place the required pins (choose reasonable and unique names!), add the `{{NAME}}` and `{{VALUE}}` text elements and draw the outline. The result should look like this:



Yes, the overlapping pin texts look ugly, but let's ignore that for the moment ;)

### Symbol Conventions

For details about how symbols should be designed, please take a look at our [Symbol Conventions](#). The most important rules are:



- For generic components, create generic symbols (e.g. *Diode* instead of *1N4007*).
- The origin (coordinate *0,0*) should be in (or close to) the center of the symbol.
- Pins must represent the *electrical* interface of a part, not the *mechanical*. So don't add multiple pins with the same function (e.g. *GND*) and don't name pins according their location in the package. Name them according their electrical purpose (e.g. *IN+*, *IN-*, *OUT*) instead, or just use incrementing numbers (i.e. *1*, *2*, *3*, ...).
- Pins should be grouped by functionality and must be placed on the 2.54mm grid.
- There should be text elements for `{{NAME}}` and `{{VALUE}}`.

### 2.4.3. Component

The next element you need to create is the Component for a single OpAmp. Because it is still very generic (beside the LMV321LILT there are many other OpAmps with exactly the same functionality), so you should enter a generic name like *Single OpAmp*. In addition, it's really important to choose a Category for the new Component, otherwise it's hard to find it in the library

when you want to add it to a schematic.

**Enter Metadata**  
Please specify some metadata about the new element.

Name: Single OpAmp

Description:

Keywords: Comma separated list of keywords (en\_US, optional)

Author: me|

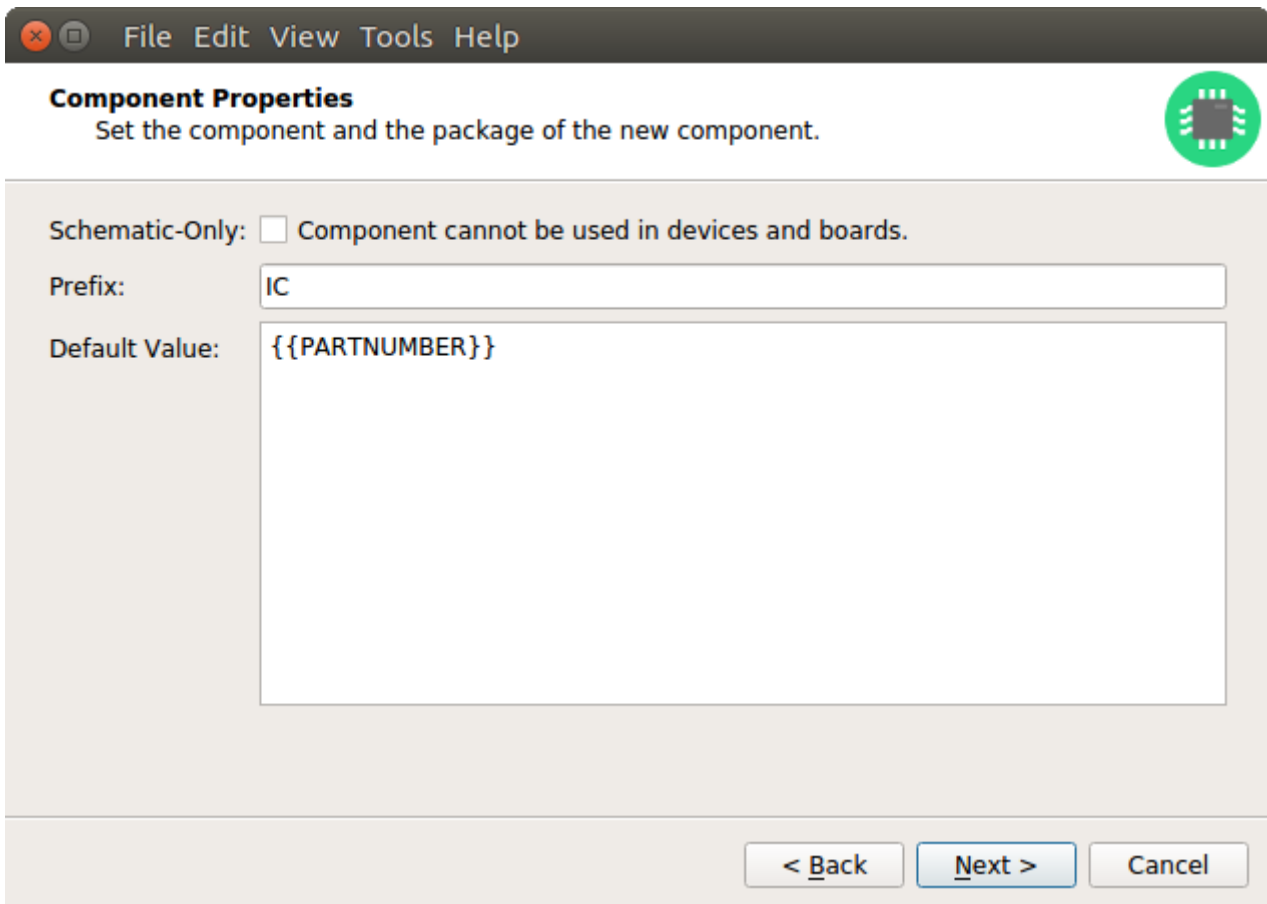
Version: 0.1

Category: 1204bf54-dbbb-4056-a771-5b13990ddd77 ...  
Root category => Integrated Circuits => **Amplifiers**

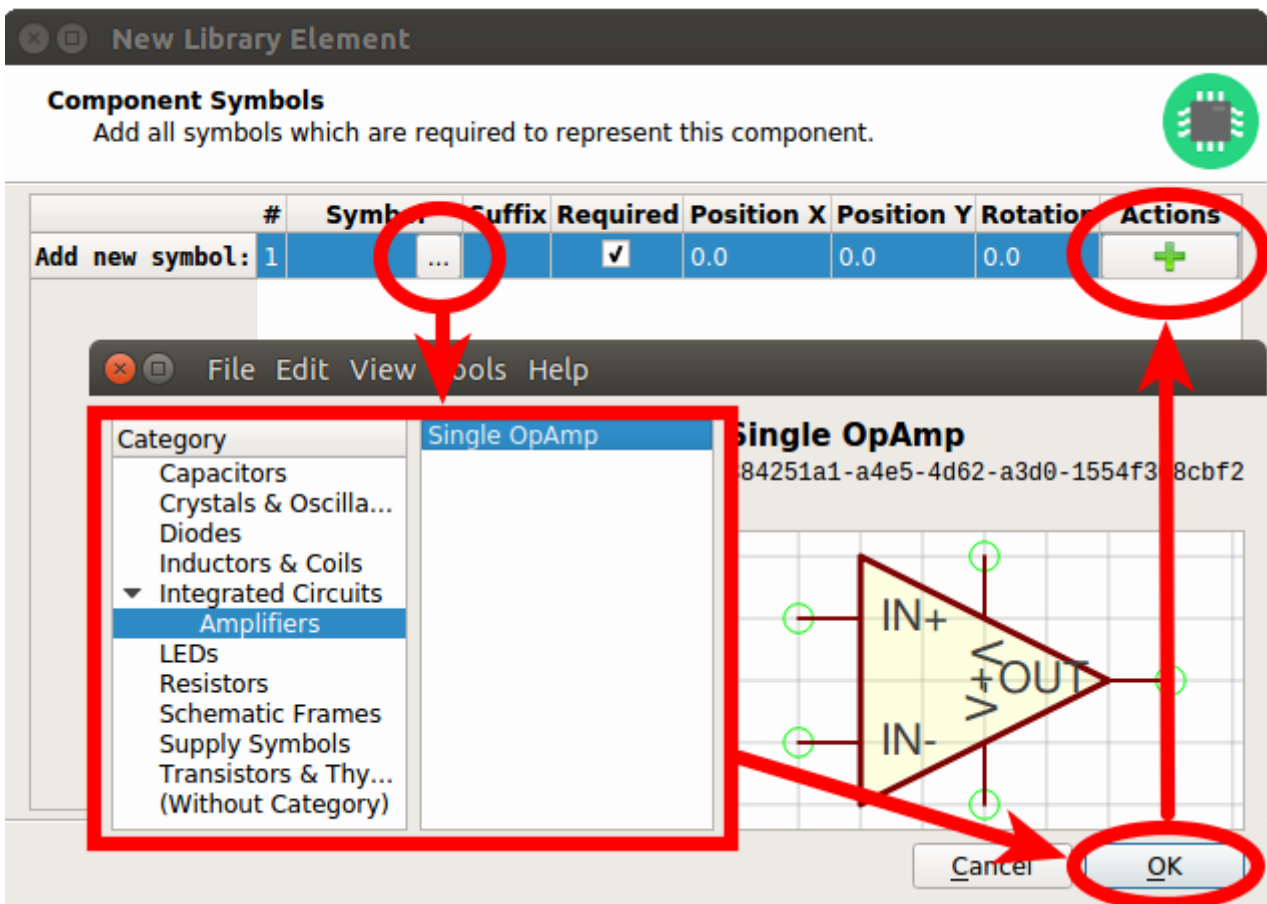
< Back Next > Cancel

Then you're asked to specify some properties of the Component:

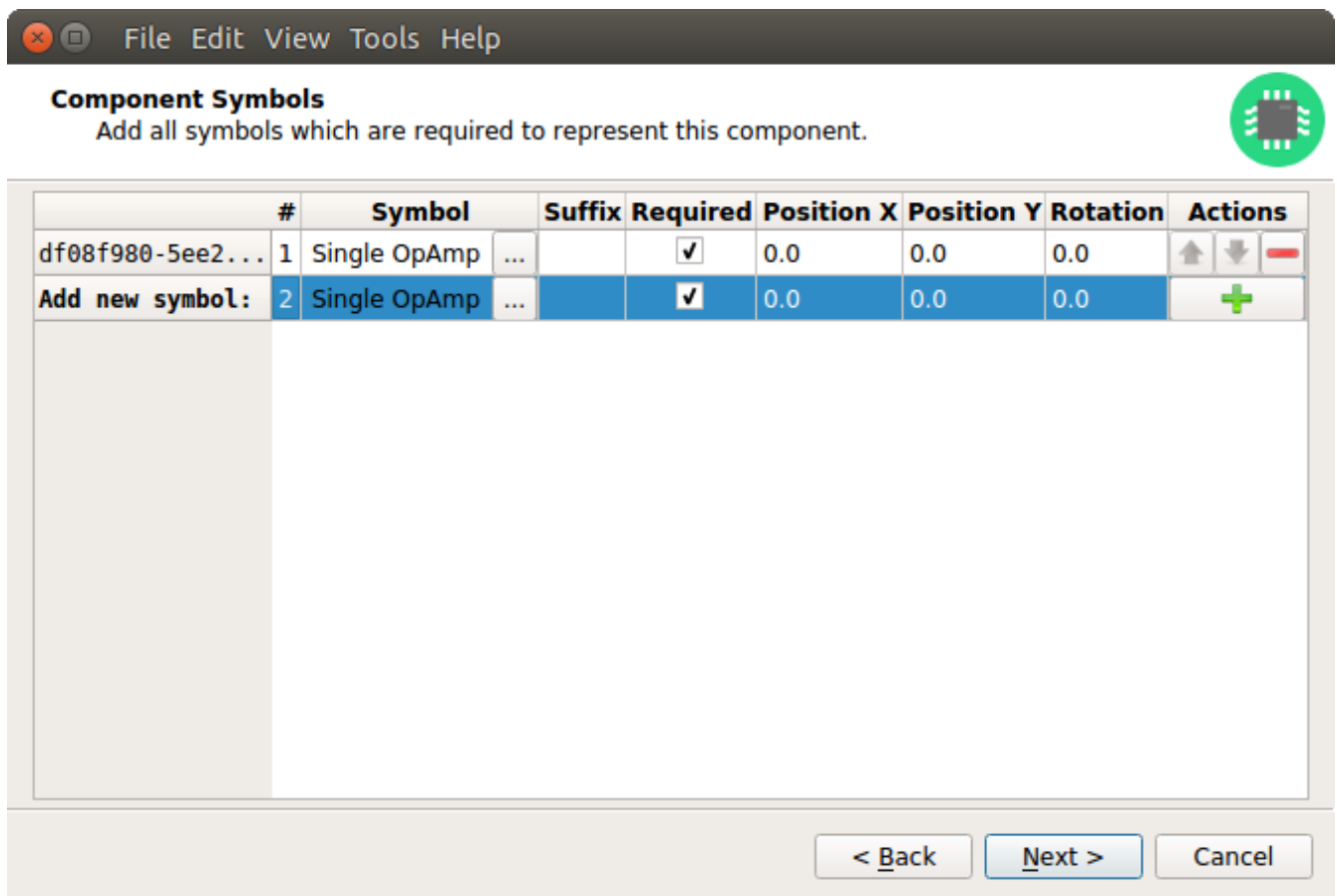
- **Schematic-Only:** This should be checked if the Component must not appear on a board, but only in the schematics. This is typically used for schematic frames (yes, they are also Components).
- **Prefix:** When adding the Component to a schematic, its name (designator) is automatically set to this value, followed by an incrementing number. So if you choose the prefix *R*, components added to a schematic will have the names *R1*, *R2*, *R3* and so on. The prefix should be very short and uppercase.
- **Default Value:** In addition to the name, Components also have a value assigned to it, which is typically also displayed in the schematic. For example a capacitor has its capacitance (e.g. *100nF*) set in its value. When adding a Component to a schematic, its value is initially set to the value specified here. The value can also be a placeholder, for example `{{PARTNUMBER}}`, `{{RESISTANCE}}` or `{{CAPACITANCE}}`. If you are unsure, just leave it empty, you can still change it later.



Now you need to choose the Symbols which represent the Component in schematics. Most Components have only one Symbol, but you can also add more than one, for example an OpAmp can have separate Symbols for power and amplifier.



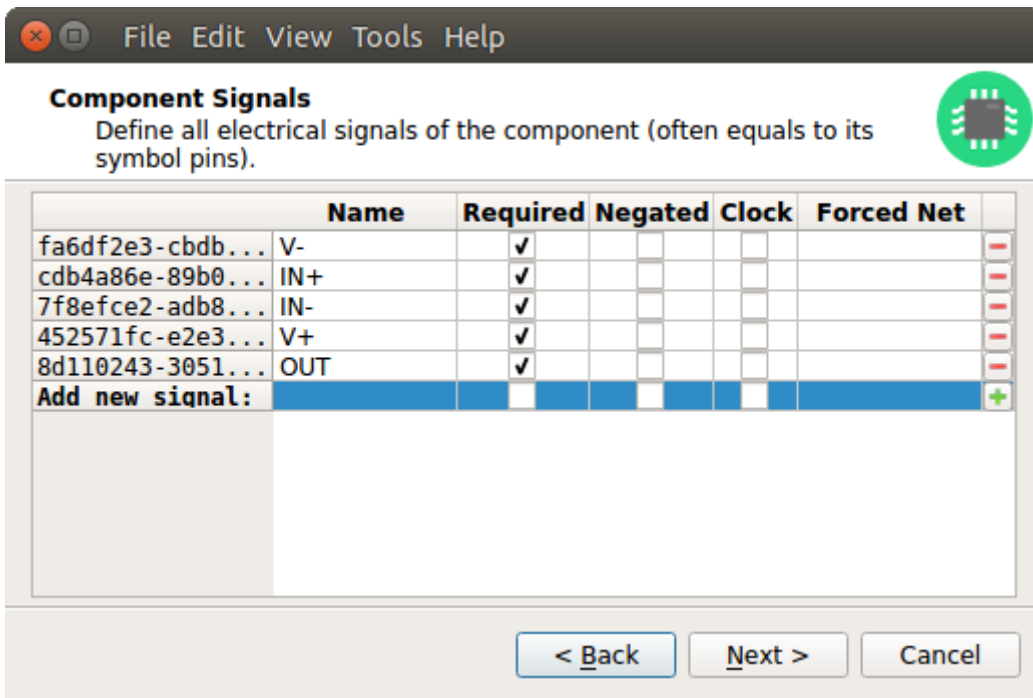
After adding the OpAmp Symbol, it should look like this:



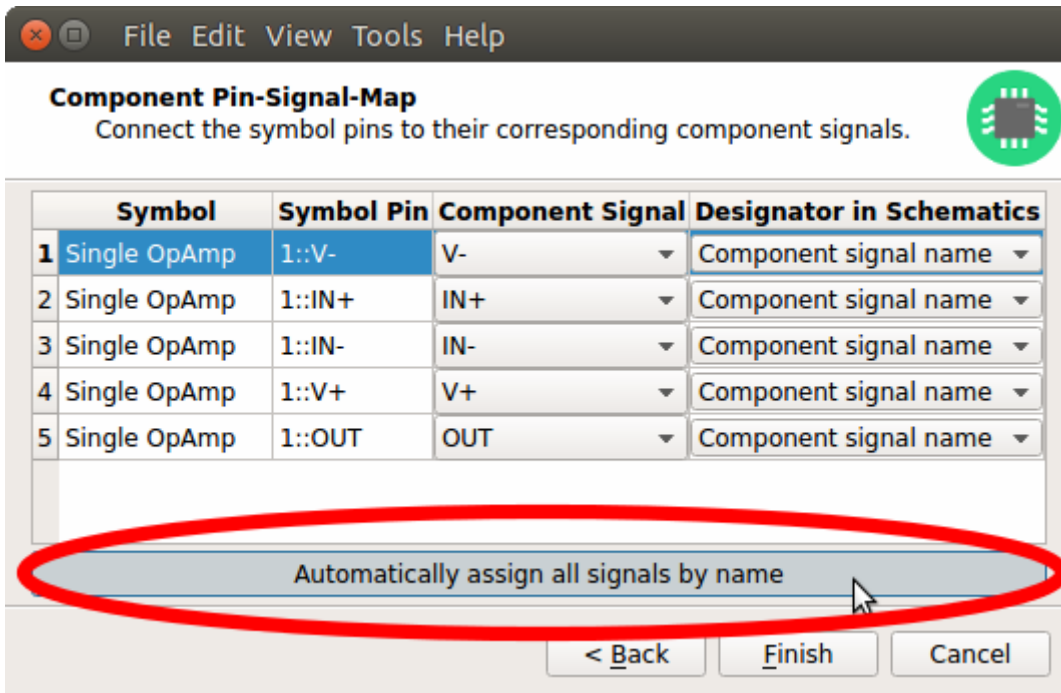
The next step is to define all so-called Signals of a Component. Signals represent the "electrical interface" of a Component. For example a transistor consists of the Signals *base*, *collector* and *emitter*. For a Component it's irrelevant whether the "real" transistor has multiple emitter pads, or an additional thermal pad and so on, the Component only specifies the three Signals.

LibrePCB automatically extracts the Signals from the Pins of the specified Symbols, so often we don't have to do this by hand. But sometimes you still should adjust the names or properties of these Signals. For the OpAmp, we check *Required* for all Signals, so the ERC will show a warning if these Signals are not connected to a net when the Component was added to a schematic:

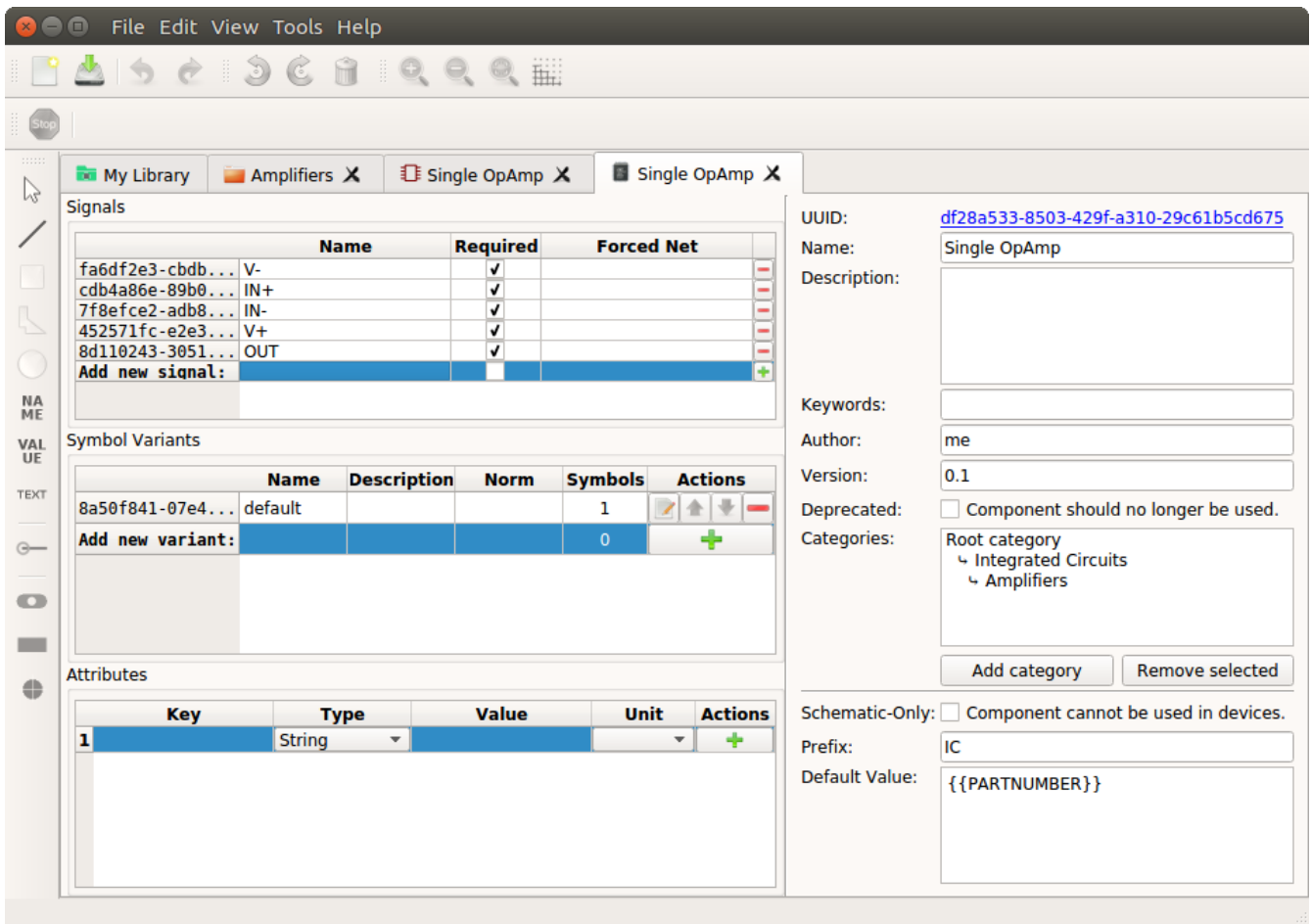




These Signals now need to be assigned to the corresponding Symbol Pins, but as they were automatically generated from the Pins, you can just click on the button below to automatically assign all Pins to their Signals:



That's it, the Component is now ready to be used:



For this simple example, this procedure may feel complicated. This is due to the broad flexibility of the librepcb library approach. The Component which we created actually only uses basic library features, but as soon as you understand the our library concepts, you will be able to easily create much more powerful library elements. We're sure you will love the flexibility of the library concept ;)



Following are the most important rules to create reusable Components:



- Create generic components whenever possible. Only create specific components for manufacturer-specific parts (like microcontrollers).
- Name signals according their electrical purpose (e.g. *Source*, *Drain*, *Gate*).
- Don't add multiple signals with the same name. Even for a microcontroller which has multiple GND pins, the component should have only one GND signal. A component represents the *electrical* interface of a part, not the *mechanical*!

## 2.4.4. Package Category

Before creating a Package for the LMV321LILT, you should (optionally) create a category for it. This is basically done exactly the same way as you already created the Component Category, so we won't explain it again. It could look like this:

File Edit View Tools Help

**Enter Metadata**  
Please specify some metadata about the new element.

Name:

Description:

Keywords:

Author:

Version:

Category:  ...  
Root category

< Back Finish Cancel

## 2.4.5. Package

Then you need to create the Package for the LMV321LILT, which is called *SOT23-5*. After specifying some metadata (like you already did for other library elements), you are asked to specify all pads of the Package. The *SOT23-5* has 5 pads which we just name from 1 to 5:

File Edit View Tools Help

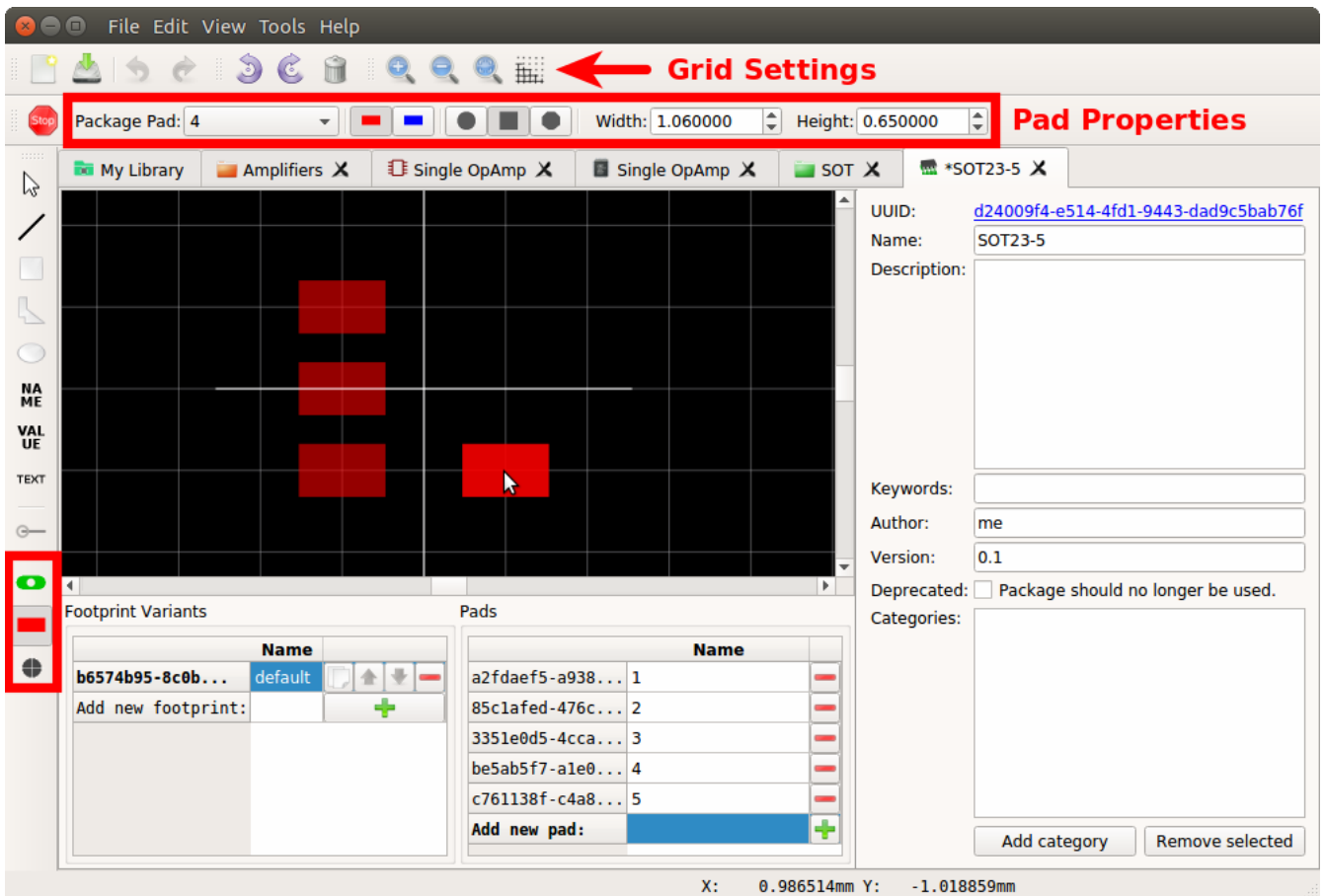
**Package Pads**  
Define all available pads of the package.

Name		
a2fdaef5-a938...	1	-
85c1afed-476c...	2	-
3351e0d5-4cca...	3	-
be5ab5f7-a1e0...	4	-
c761138f-c4a8...	5	-
<b>Add new pad:</b>		+

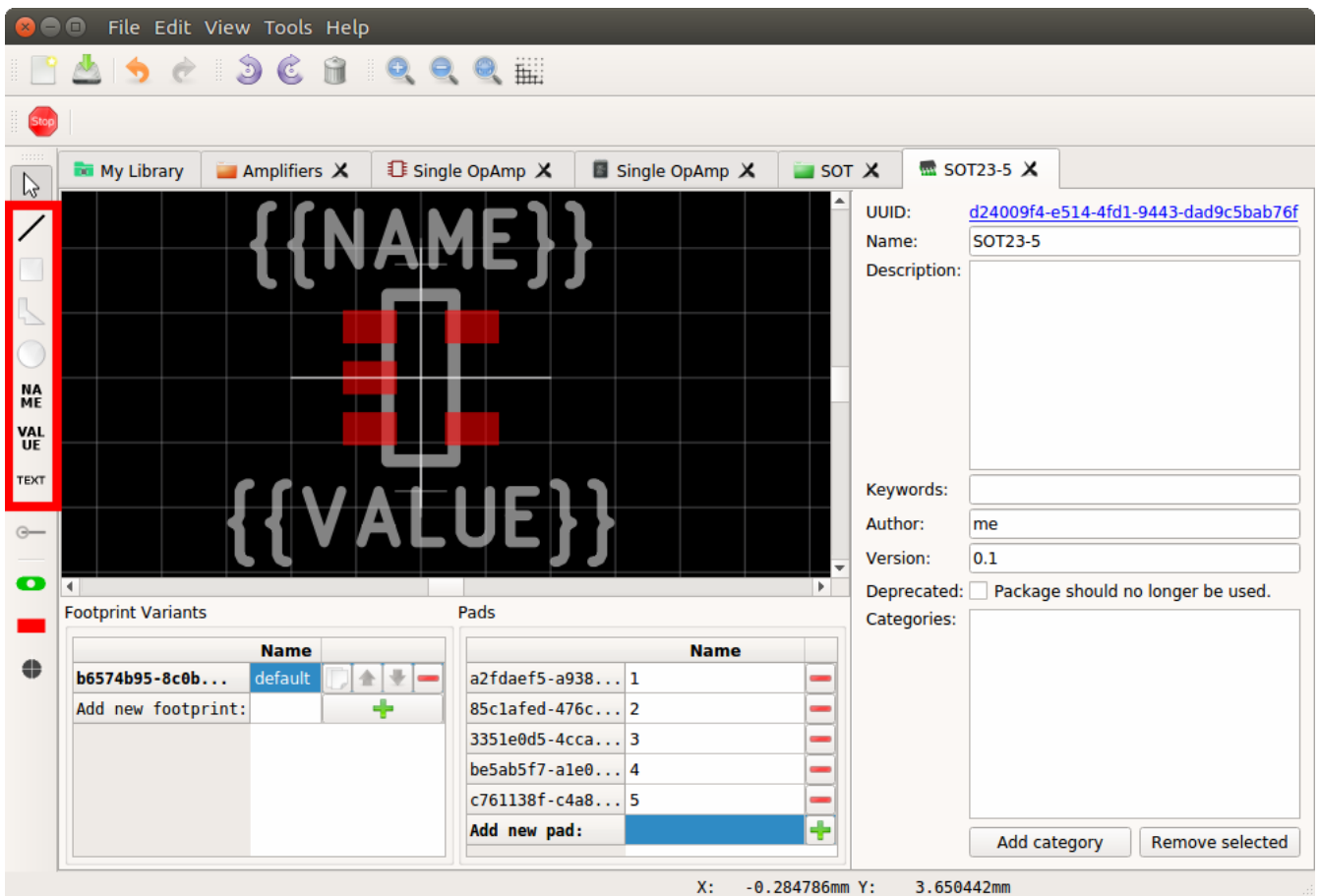
Note: You should also add pads which are not always required (e.g. thermal pads)!

< Back Finish Cancel

After that, you can start drawing the footprint of the Package. It's recommended to start with the pads:



And then add other graphical items like the outline, name and value:



That is already enough for a simple footprint.

### Package Conventions

For details about how Packages should be designed, please take a look at our [Package Conventions](#). The most important rules are:

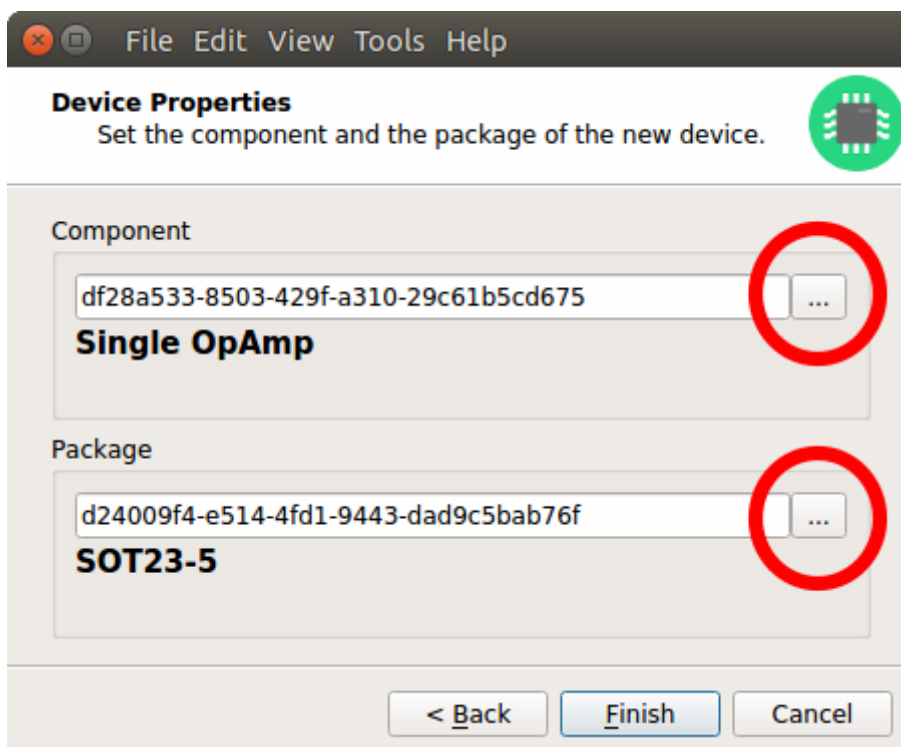


- Create generic packages, not specific ones. For example *DIP08* is *DIP08* — no matter whether it's an OpAmp, an EEPROM or a microcontroller.
- The origin (coordinate *0,0*) should be in the center of the Package body.
- Footprints must be drawn from the top-view. When a footprint needs to appear on the bottom of a board, this can be done in the board by mirroring it.
- Add **all** pads of a package, not only the one you currently need. For example if the package has a thermal pad, you should add it, even if you currently don't need it.
- Name pads according IPC-7351 (if applicable; see section Library Conventions for more information), typically just *1, 2, 3* etc. Only name pads according their electrical purpose (e.g. *Gate*) if the Package is very specific for a particular purpose.
- Pin 1 should always be at the top left.
- There should be text elements for `{{NAME}}` and `{{VALUE}}`.

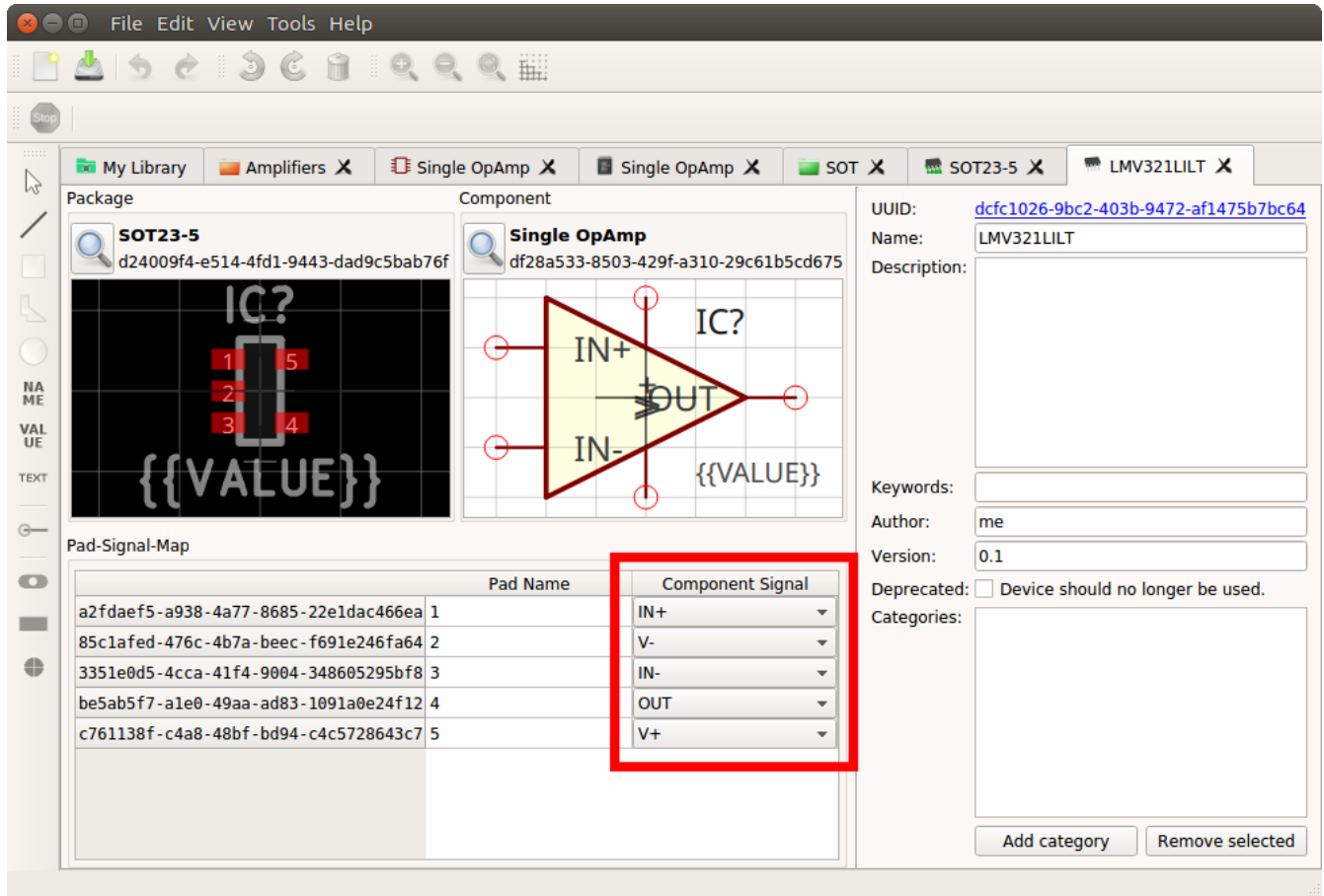
## 2.4.6. Device

The last library element you need to create is the Device which combines the Component *Single OpAmp* with the Package *SOT23-5*. This is actually the only library element which is specifically for LMV321LILT — all previously created elements are generic!

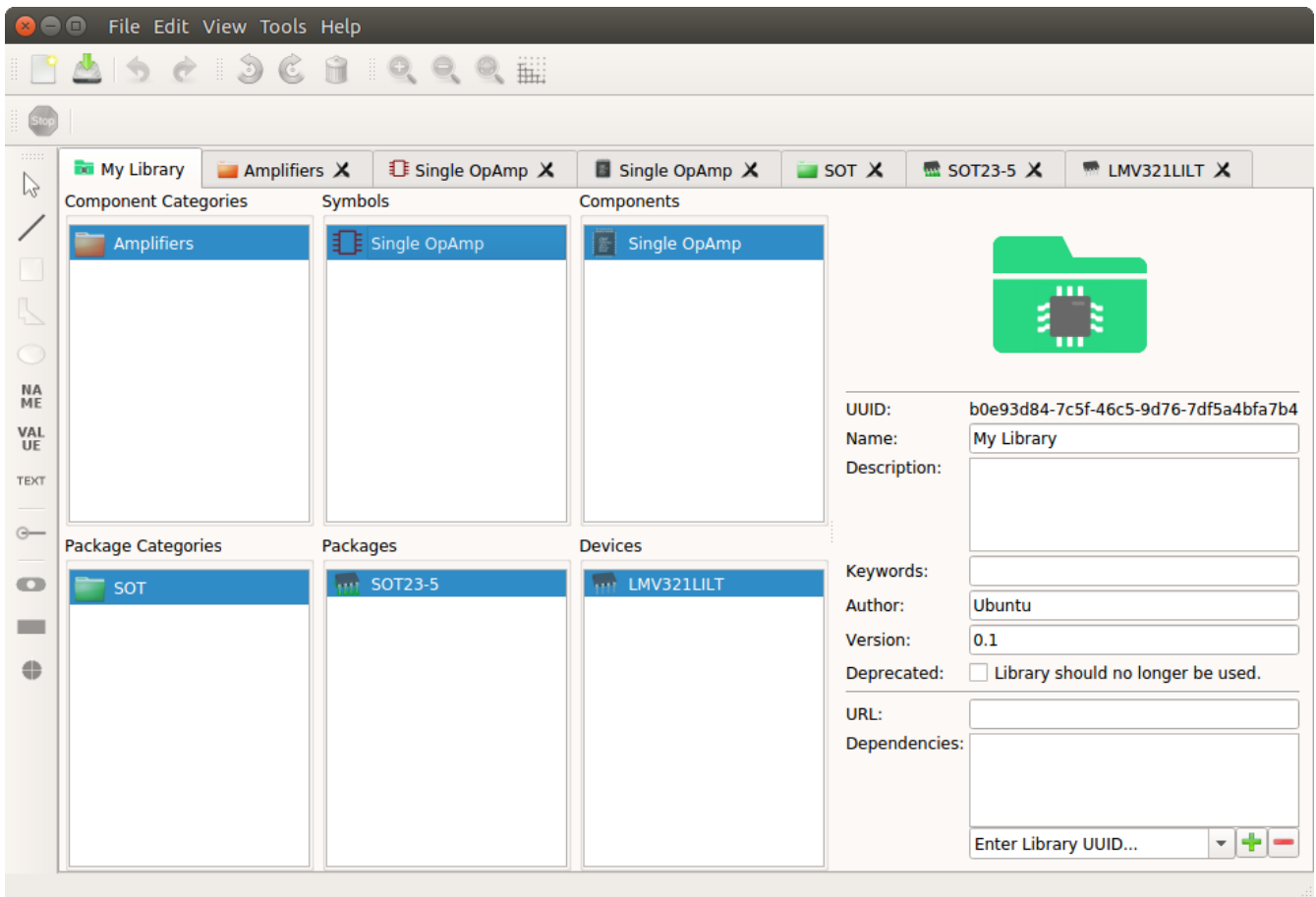
Again, specify some metadata about the Device first. Then you need to choose the Component and Package you want to assign:



Then you have to assign the Package pads to Component signals according to the datasheet of the LMV321LILT:



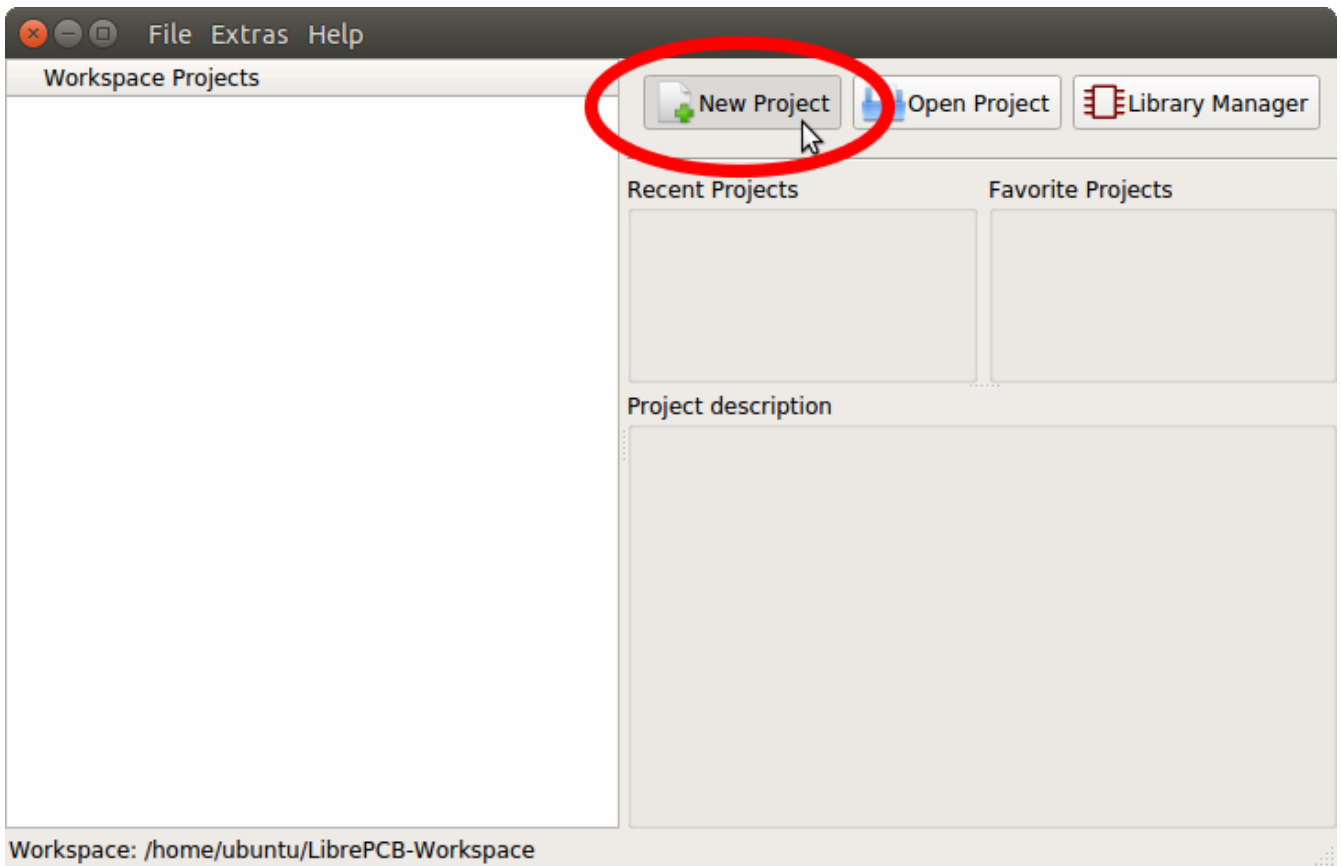
And that's it! In the library overview (the first tab in the Library Editor) you can see all the elements you have created:



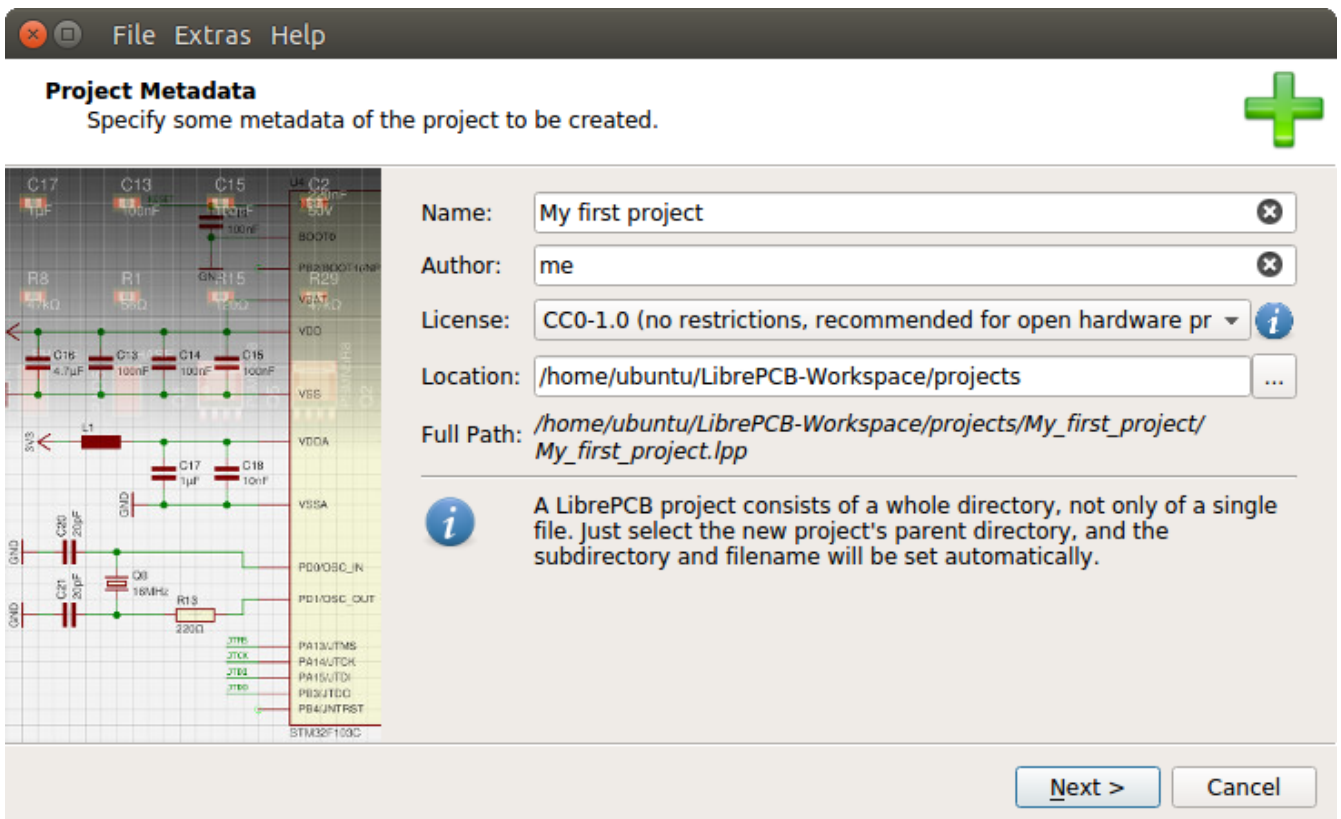
The LMV321LILT is now ready to be added to schematics and boards. And because the Categories, Symbol, Component and Package are quite generic, they can also be used for many other library elements :)

## 2.5. Create a New Project

In LibrePCB, schematics and boards are always part of a Project, so before creating schematics and boards you first need to create a Project for every PCB. This is really easy, just follow the *New Project* wizard:



Fill in some Project metadata:



For Open-Hardware projects it's highly recommended to select the license *CC0-1.0* as this allows everyone to use your project without restrictions.

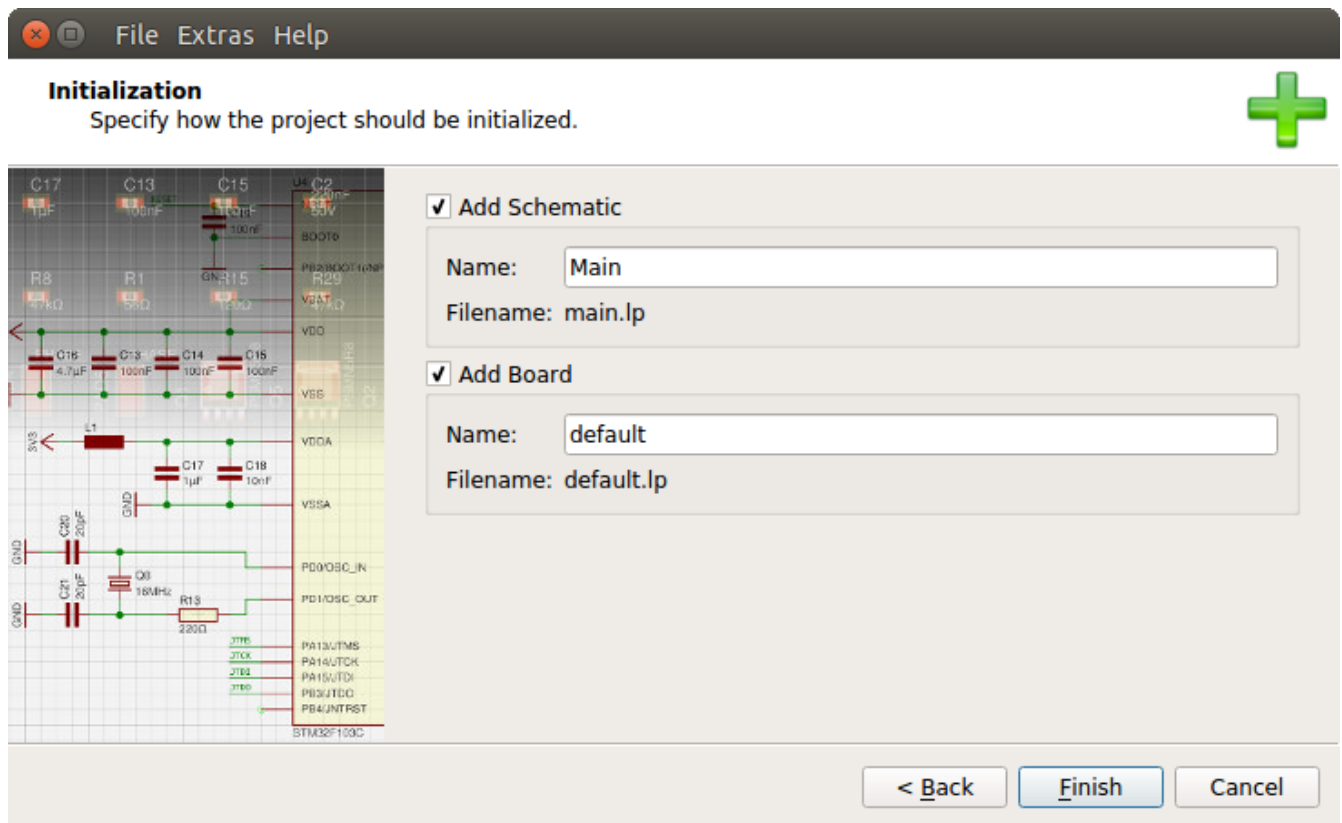


While it is possible to create the Project outside the Workspace directory, it's

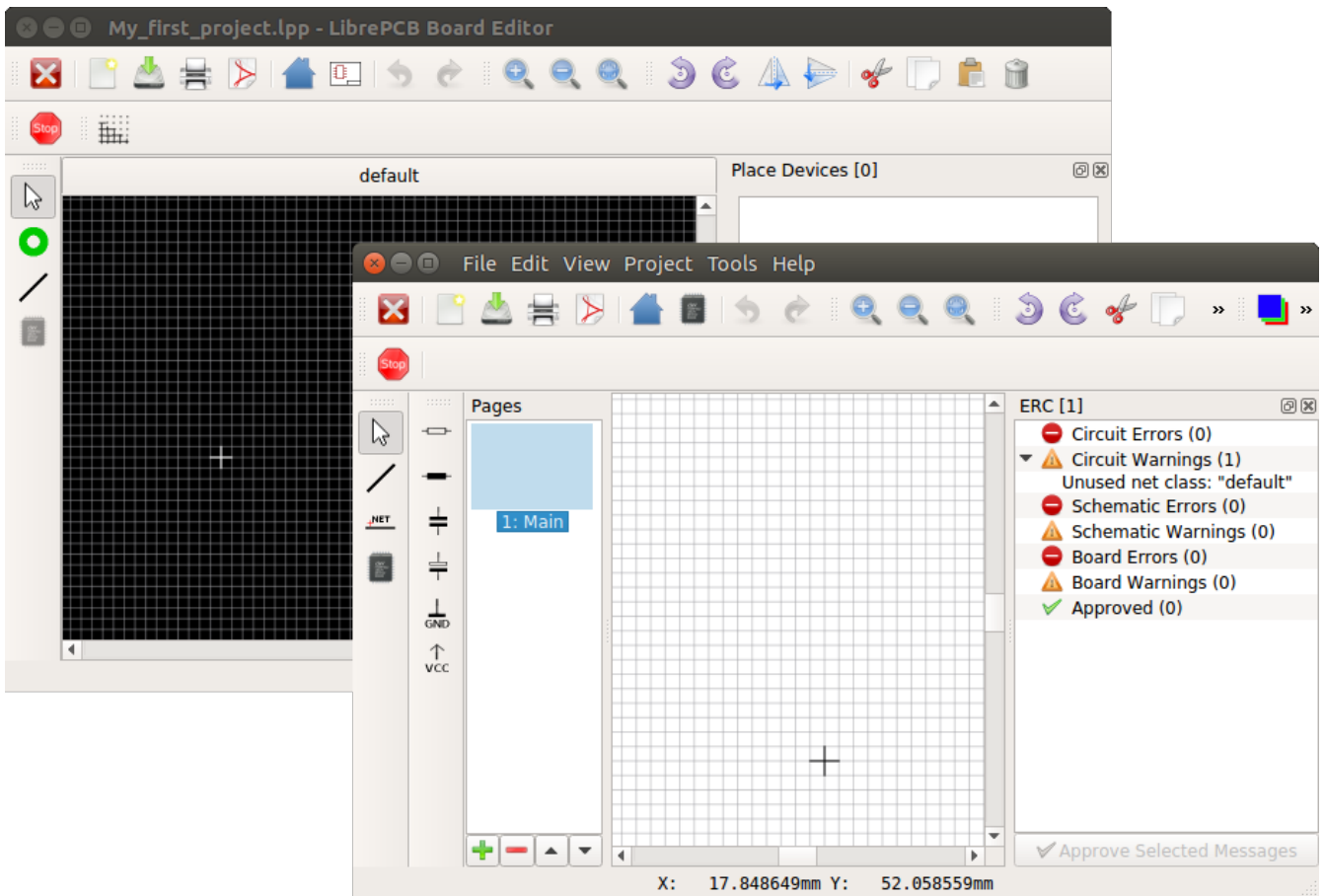


recommended to store all Projects within the Workspace because these are then shown in the Control Panel by default making them easy to locate and use. Projects outside the Workspace need to be opened by specifying their location in an open dialog.

Now you can choose whether the Project should be initialized with a first schematic page and board, and how they are named. If you are unsure, just accept the default values:

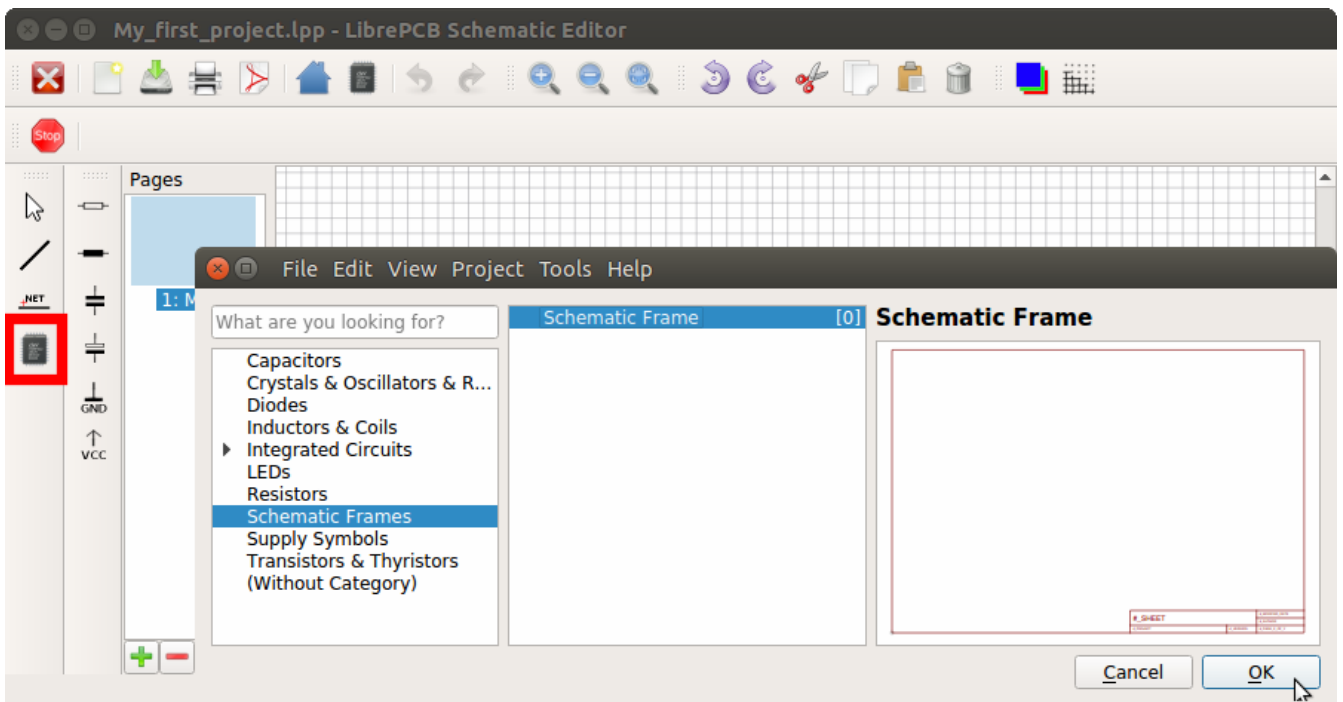


After clicking on *Finish*, the schematic and board editors show up and you're ready to draw the schematics:

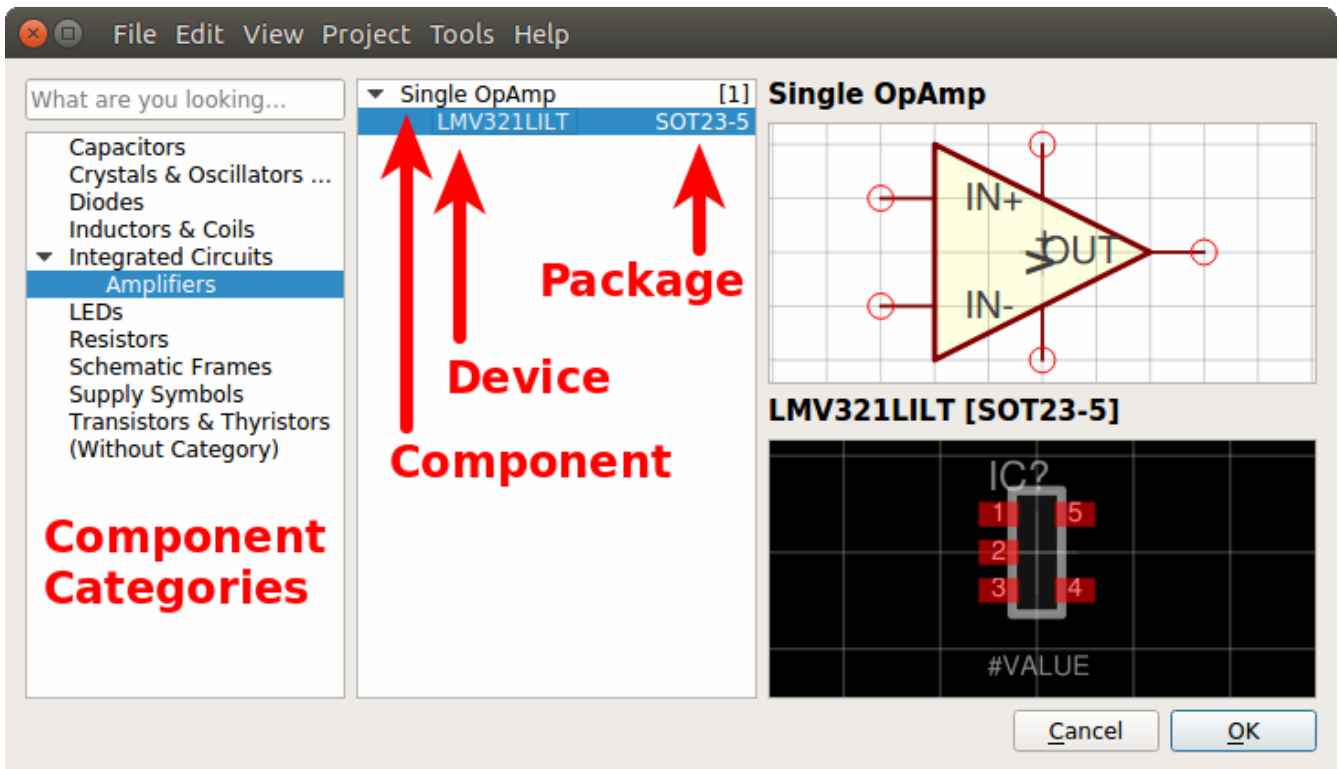


## 2.6. Create Schematics

First, you may want to add a frame to the schematic. This is done by choosing a schematic frame in the Component Chooser:

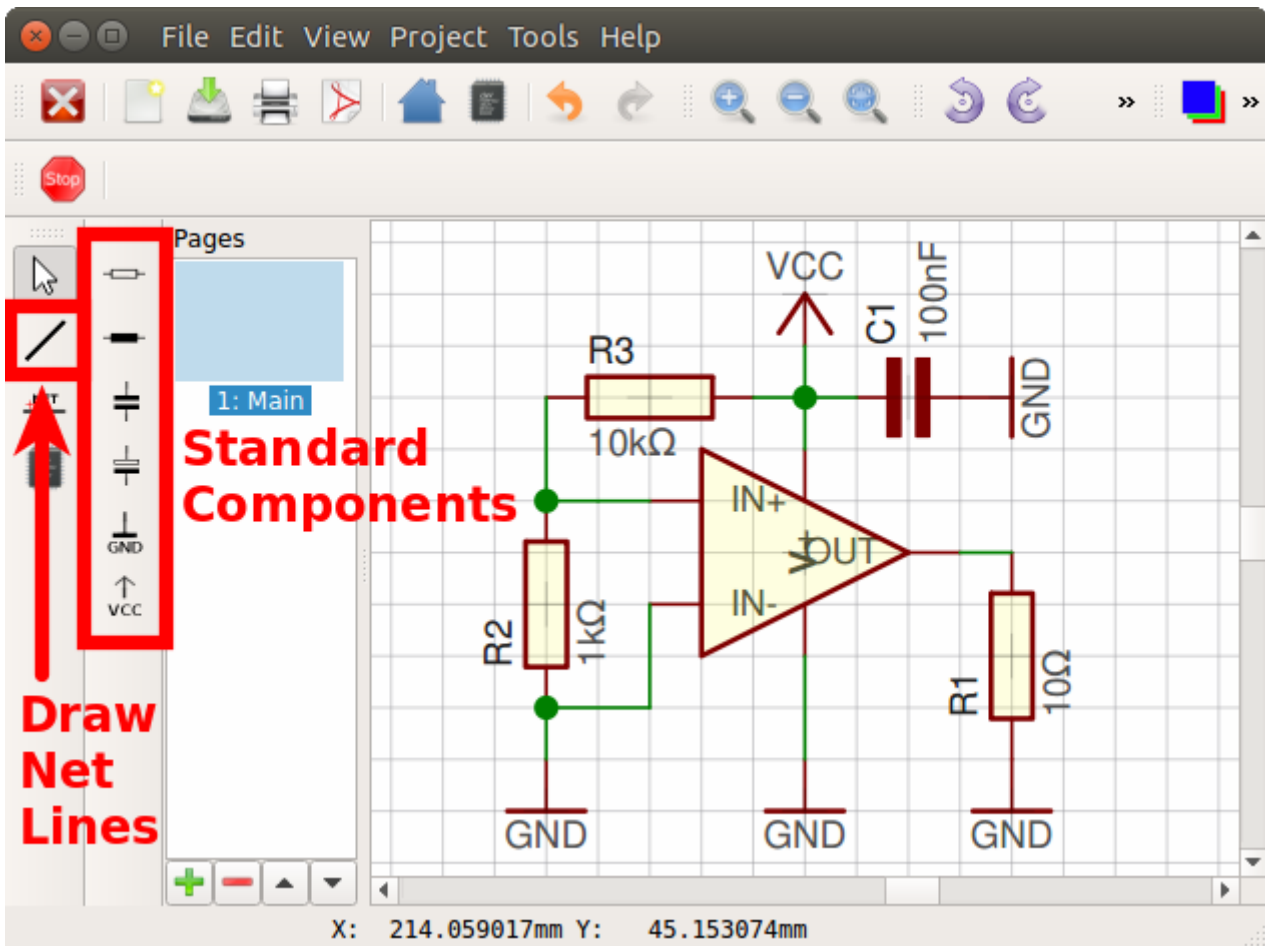


Then let's add the OpAmp we have created in our own library, again with the Component Chooser:



Here you can choose whether you want to add the Component *Single OpAmp* or the Device *LMV321LILT*. Yes, you can add either a Component or a Device to a schematic ;) Actually the schematic always contains Components (not Devices), but you can still (optionally) choose the Device which will be used later when adding the Component to the board (boards always contain Devices).

The most commonly used Components are also available in the toolbar on the left side:



Just place some Components and draw the net lines with the corresponding tool in the toolbar at the left side.

## 2.7. Create Boards

Once the schematic is (more or less) complete, you can start designing the PCB with the board editor.

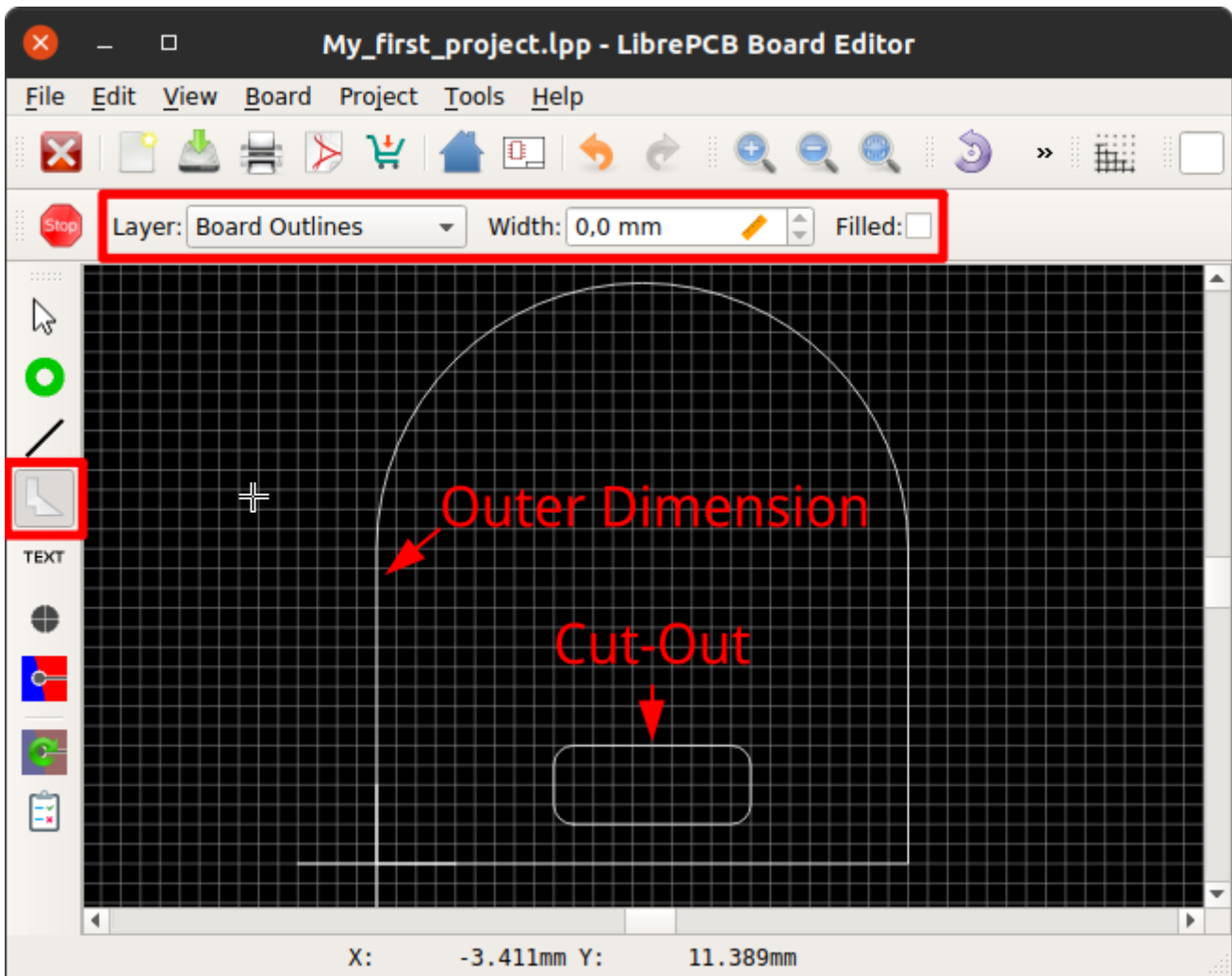
### 2.7.1. Draw Outlines


The most important thing of the board is its outline. Generally there must be a **single, closed polygon on the *Board Outlines* layer**. It is recommended to set its **line width to 0.0mm** since — in contrast to many other polygons — this polygon does not represent any actual material but only the outer dimension of the PCB.

If your PCB needs non-plated cut-outs (e.g. slots, windows, ...), just draw these polygons exactly the same way as the outer board outline. These polygons are implicitly considered as cut-outs since they are located *within* the outer polygon.



All polygons on the *Board Outlines* layer shall represent the actual board outlines (i.e. the edges), **NOT** the paths for the milling cutter! The PCB manufacturer will automatically offset the outline polygons to calculate the actual paths for the cutter.



 Keep in mind that inner edges can only be produced with a specific minimum radius (corresponding to the milling cutter diameter of the PCB manufacturer). Although PCB manufacturers may produce your PCB anyway even if it contains inner edges with no or too small radius, it's highly recommended to draw all inner edges with a proper radius. Often a radius of 1.2mm or more works fine, while a smaller radius might lead to additional cost.

To draw polygons with arcs, open *Properties* from the polygon's context menu and specify the vertex coordinates and angles manually.

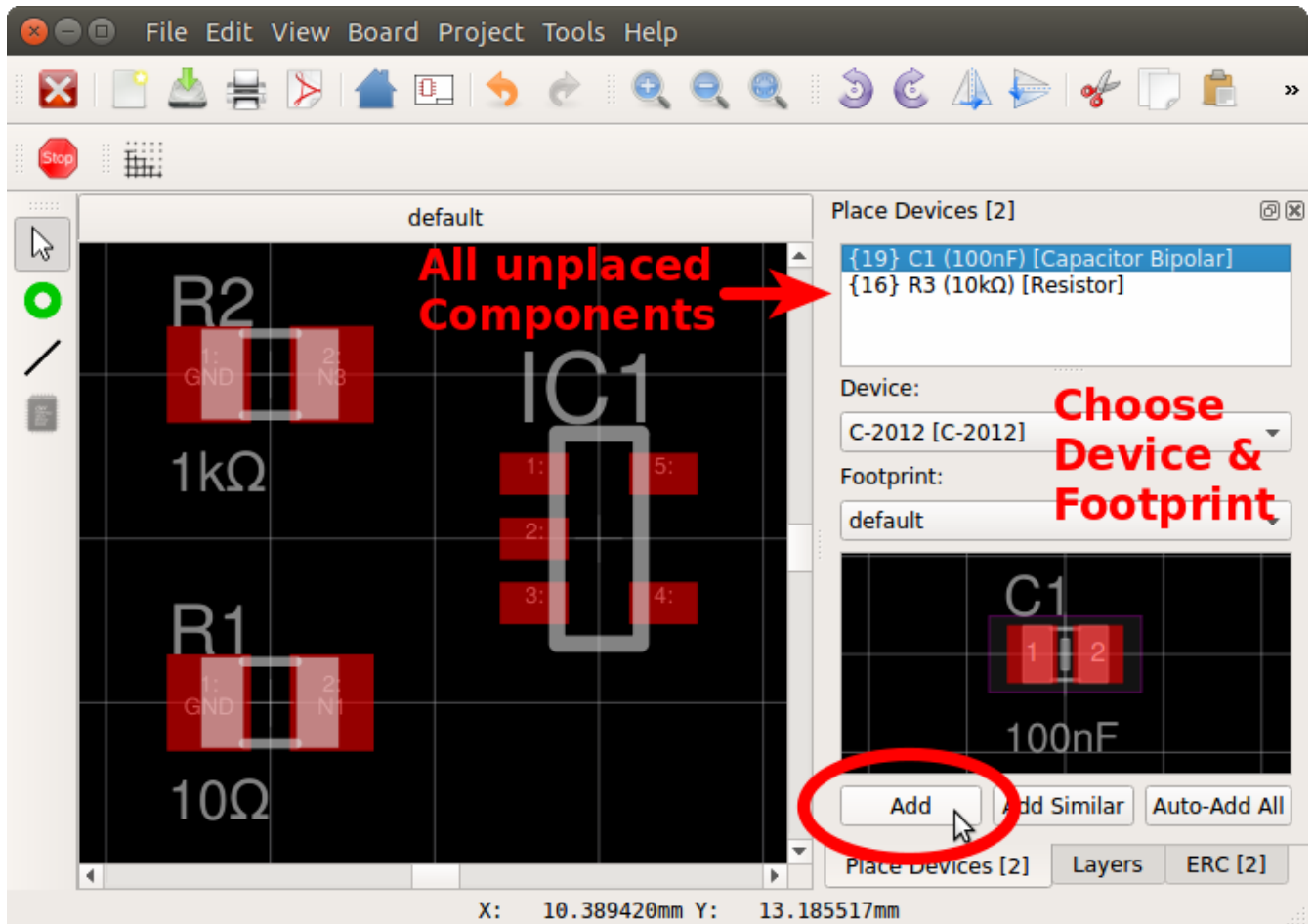
A correct board outline is really crucial to avoid problems during the PCB manufacturing process! Make sure to fulfil these rules:

- There's exactly one polygon to define the outer board outline
- Cut-out polygons (if there are any) are located fully inside the outer board outline
- There are no tangent or intersecting *Board Outlines* polygons
- Polygon layer is *Board Outlines* and line width is 0.0mm (the latter is optional, but recommended)
- Polygons are closed (start and end coordinates are exactly identical) and consisting of a single polygon object (**NOT** multiple joined lines!)

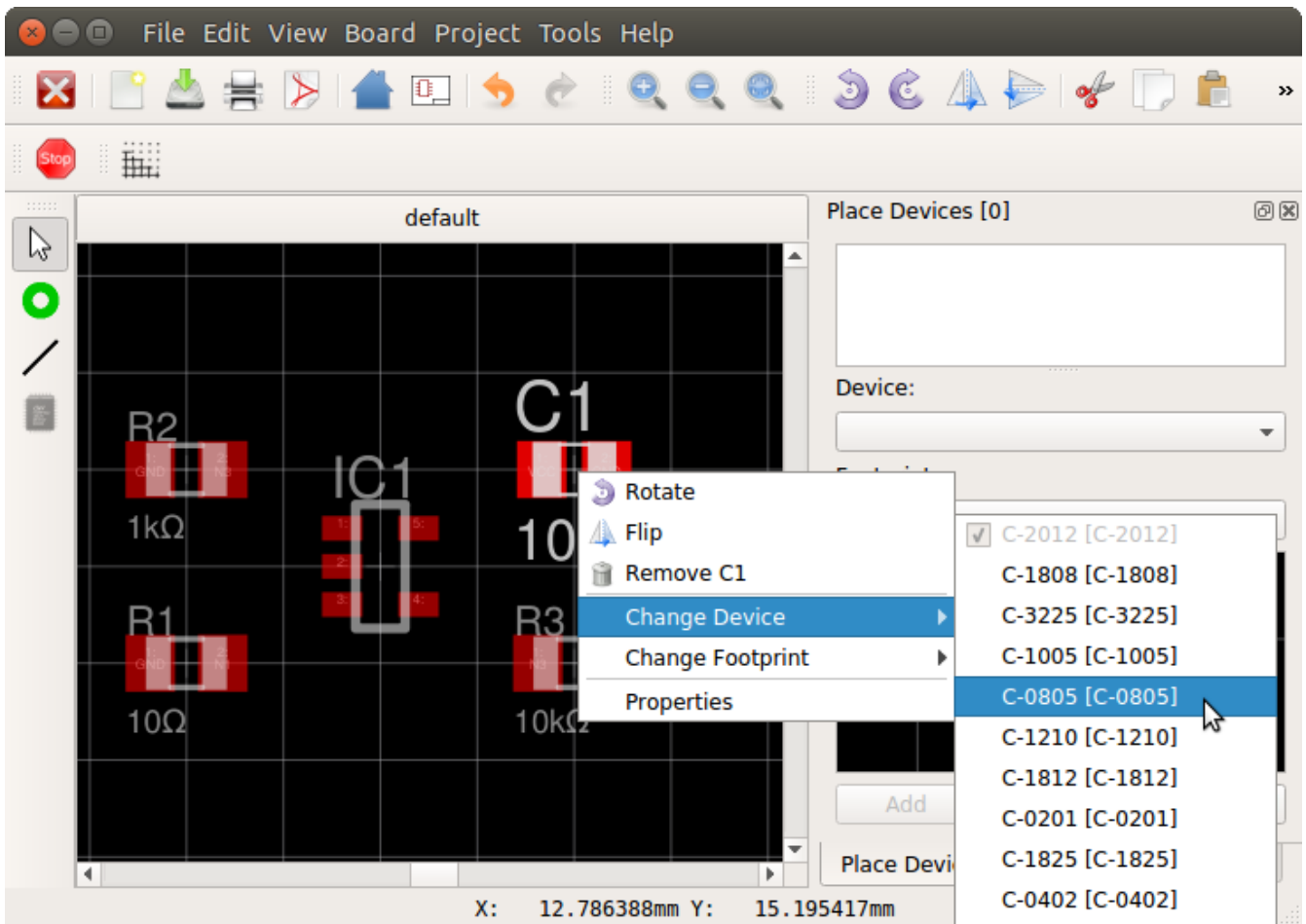
- There are no other objects on the *Board Outlines* layer

## 2.7.2. Place Devices

For every Component in the schematic, you need to place a Device in the board (except schematic-only Components, like the schematic frame). In the *Place Devices* dock on the right side of the board editor you can see all unplaced Components. For every Component you can choose its corresponding Device and add them to the board:



By the way, it's even possible to replace Devices after adding them to the board. For example you can replace a *C-1210* capacitor with a *C-0805* capacitor using the Device context menu:



Then you can start adding vias and traces to connect the pads together.

## 2.8. Order PCB

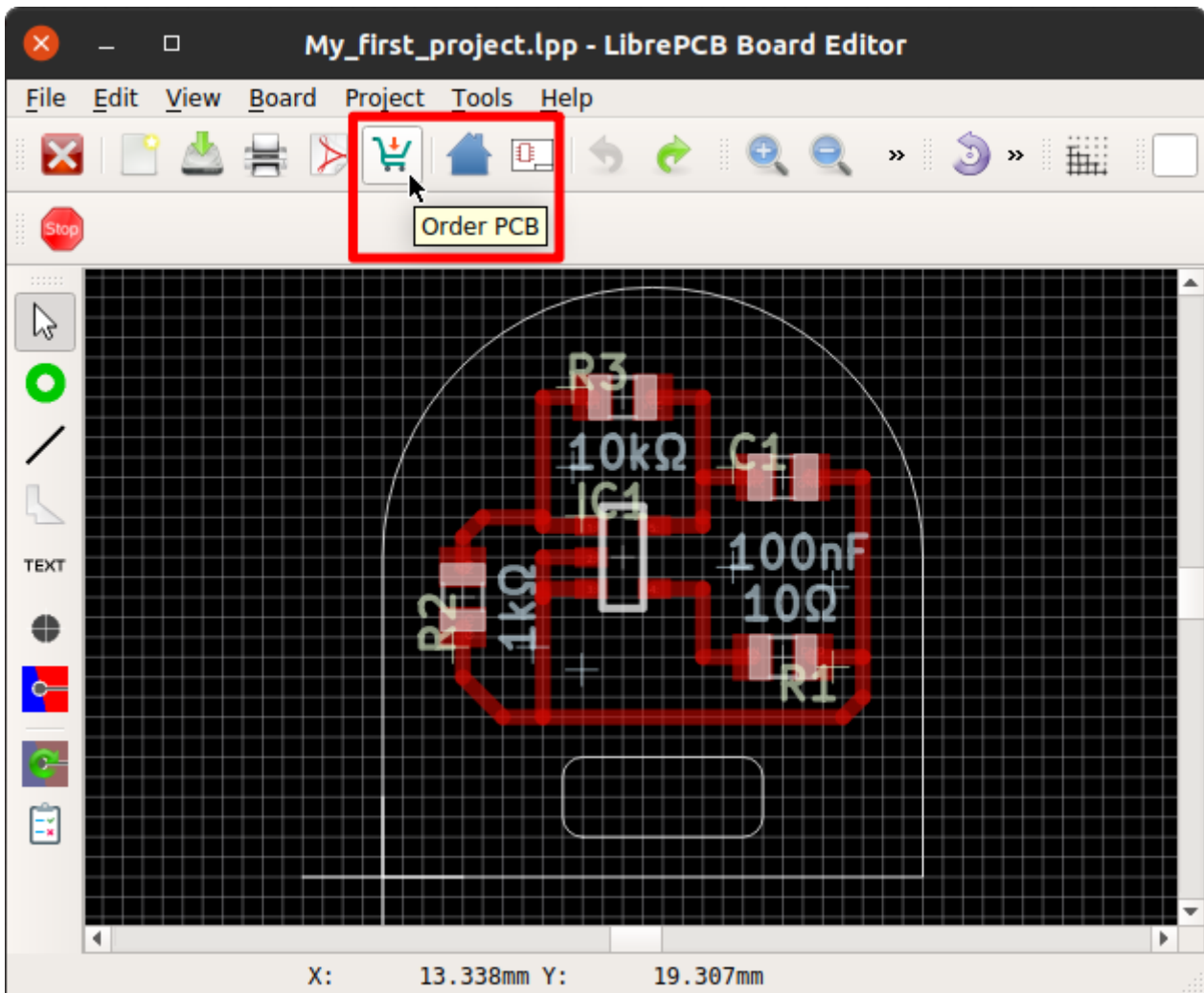
Once the board is finished, the easiest and fastest way to order the PCB is [LibrePCB Fab](#). It automatically exports and uploads all the necessary production data files without annoying you with the whole traditional production data workflow.



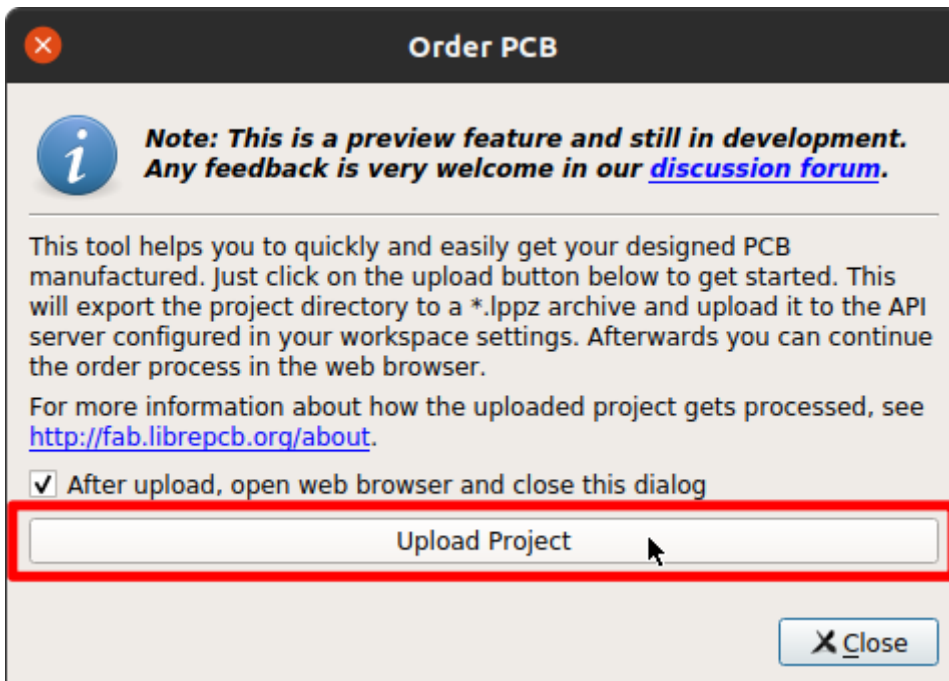
You prefer to manually generate the production data files? Or you want to use a PCB manufacturer not available at [LibrePCB Fab](#)? No problem! Just skip this section and go to [Generate Production Data](#).

### 2.8.1. LibrePCB Fab

To start the order process, open the corresponding dialog from either the schematic- or board editor:



Then just click on the button to upload the project to our fabrication service [fab.librepcb.org](http://fab.librepcb.org):



After the upload, the web browser is automatically opened where you can review and continue the order. More information about this service is available [here](#).

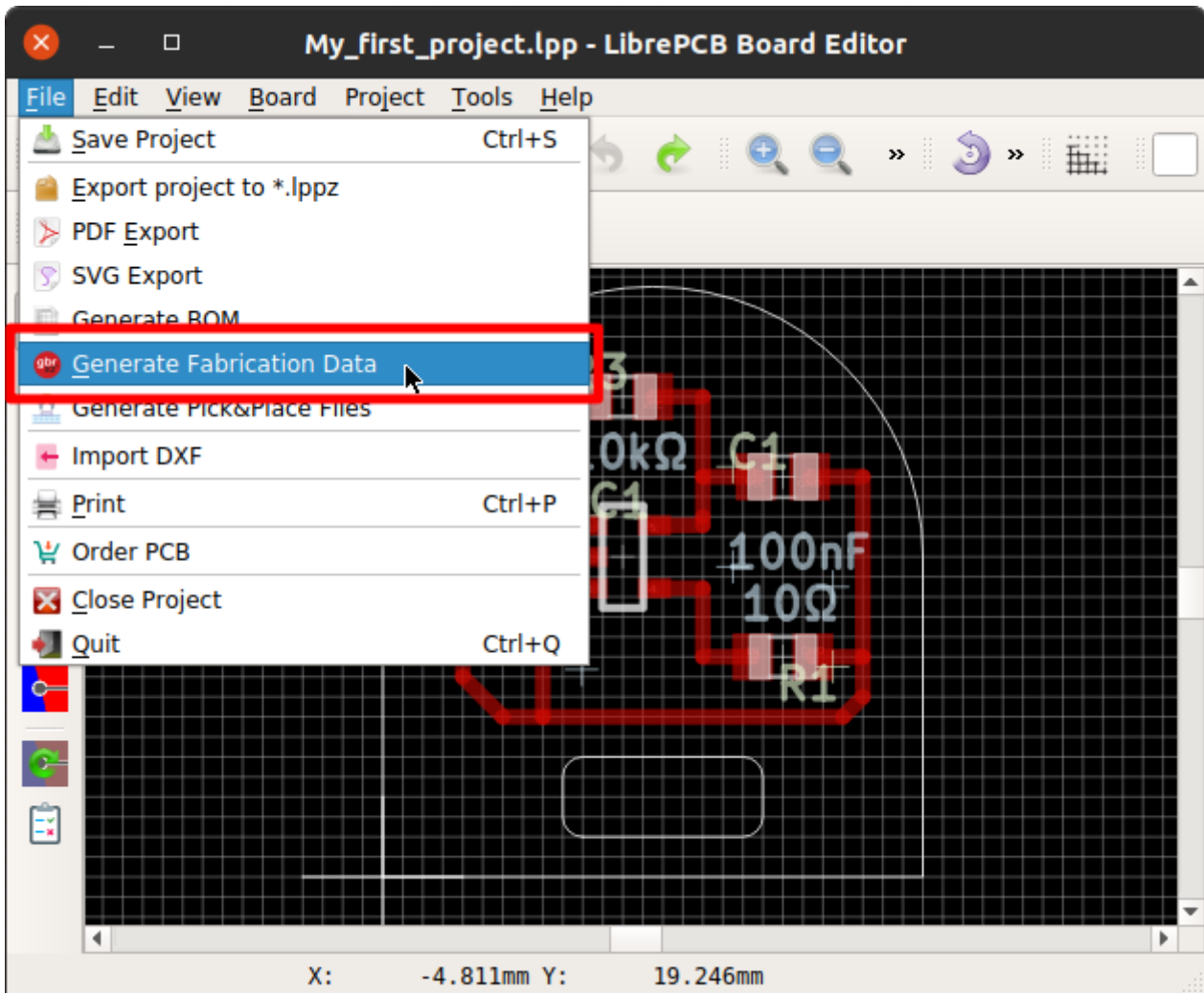




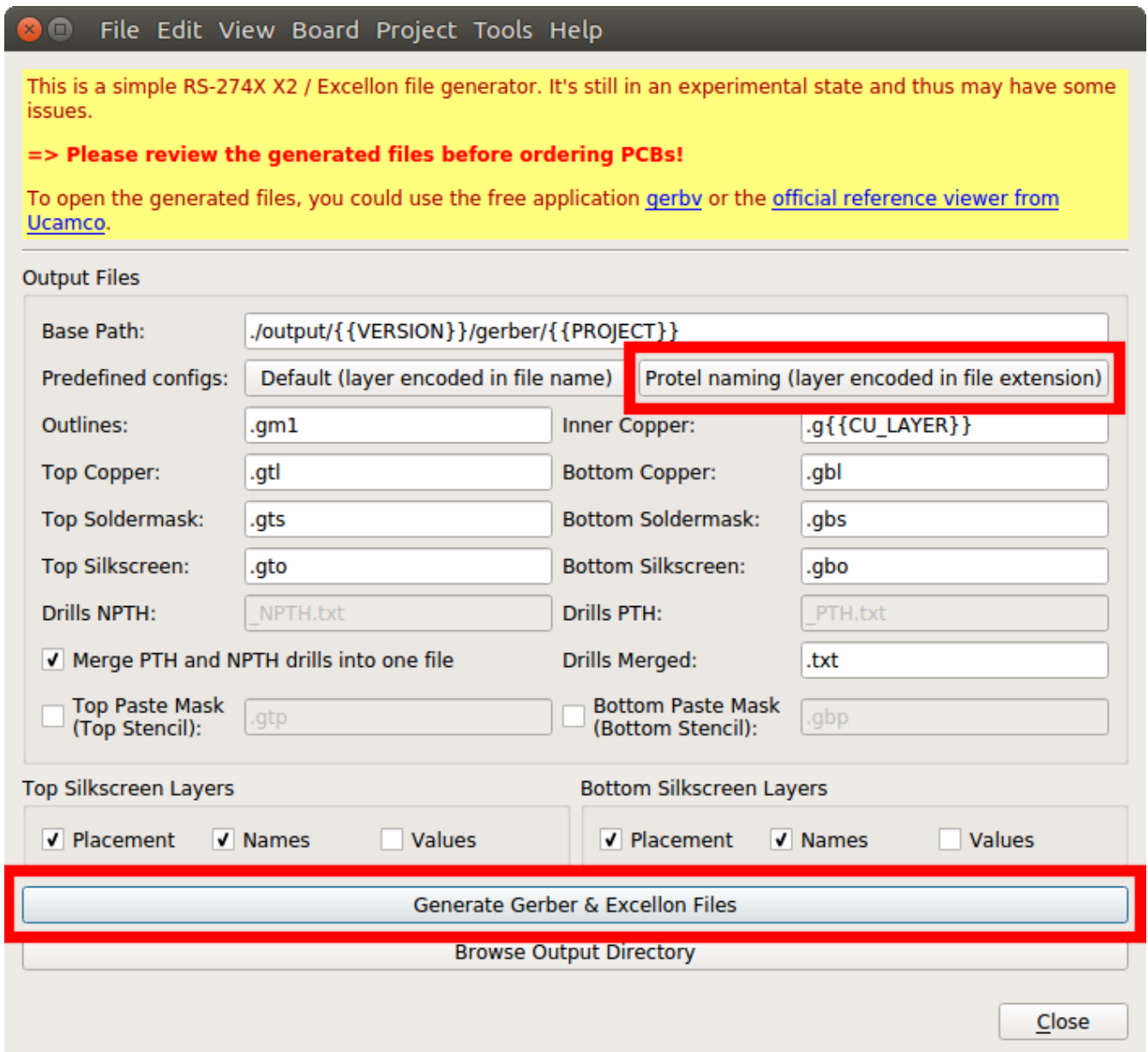
Alternatively you could also export your LibrePCB project as a \*.lppz archive (under the *File* menu item) and then upload this file with the web browser on [fab.librepcb.org](http://fab.librepcb.org). This procedure might be useful if for some reason the direct upload is not desired or doesn't work (e.g. due to a corporate firewall).

## 2.8.2. Generate Production Data

Instead of using [LibrePCB Fab](http://LibrePCB Fab), of course you can also generate the production data manually and forward these files to any PCB manufacturer you like:



Here you can adjust some settings of the Gerber export. Generally you should determine what format your PCB manufacturer accepts. Many manufacturers accept Protel-style Gerber files, so if you're unsure, click on the *Protel naming* button:



Then just click on the *Generate Gerber & Excellon Files* button and LibrePCB places the generated files in your project's *output* subdirectory. These files can then be sent to your PCB manufacturer for production.



It's highly recommended to cross-check the generated files with 3rd-party tools like [gerbv](#) or [the reference gerber viewer](#). LibrePCB is not responsible for any implications caused by wrong production data.

## 3. Library Conventions

Here we collect conventions / guidelines to be used when designing libraries.

### 3.1. Symbol Conventions



These guidelines are not yet complete. Help us create sensible conventions [on GitHub!](#)

### 3.1.1. Generic vs. Specific

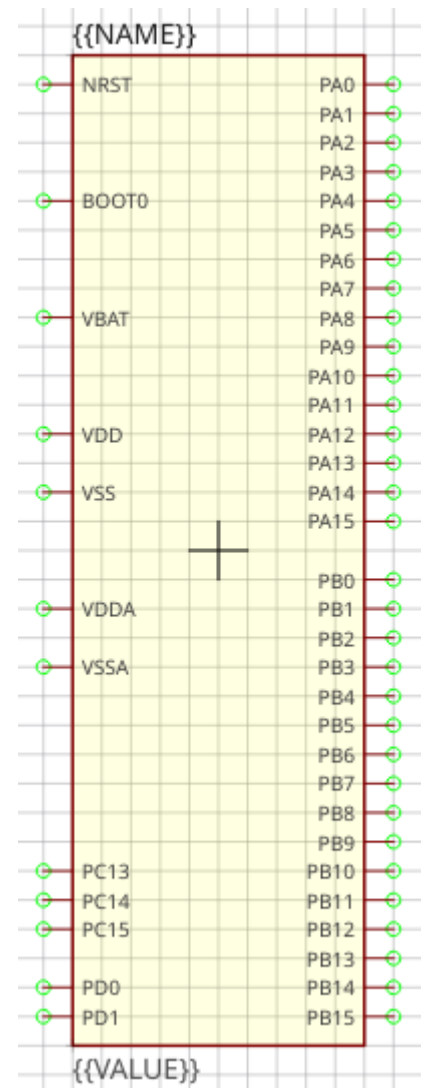
Generic components should have generic symbols. For example a diode (let's say *1N4007*) doesn't need its own symbol, a generic diode symbol is fine. So you should name it something like "Diode" and use the same symbol also for all other standard diodes. Of course every kind of diode (e.g. Zener) should have its own symbol because they look different.

On the other side, there are many very specific components, for example a microcontroller. Even if it's possible to also use generic symbols for them (e.g. "32-Pin IC"), you should create a symbol specific for that part instead. This way you can choose a reasonable [pin placement](#).

### 3.1.2. Naming

Following conventions apply to symbol names:

- Language must be American English (en\_US)
- Title case (e.g. "Capacitor Bipolar" instead of "Capacitor bipolar")
- Singular names, not plural (e.g. "Diode" instead of "Diodes")
- If reasonable, start with the generic term (e.g. "Supply GND" instead of "GND Supply") to improve navigation in sorted lists (all supply symbols are listed next to each other)



### 3.1.3. Origin

The origin (0, 0) must be at the center of the symbol (not including text elements). For non-symmetrical symbols it should be as close as possible to the center, but still on the 2.54mm grid.

### 3.1.4. Outline

The outline of a regular symbol should be drawn with a rectangle or a polygon. All vertices should be located on the 2.54mm grid and following properties should be used:

- **Layer:** *Outlines*
- **Line Width:** *0.2 mm*
- **Filled:** *no*
- **Grab Area:** *yes*

Special symbols (like a capacitor) might not have a regular outline, in such cases it's allowed to use different properties to draw the symbol geometry.

### 3.1.5. Pin Placement

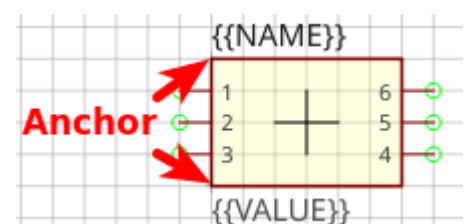
- For integrated circuit symbols (i.e. rectangular outline), generally **don't place pins at the top and bottom edges**, but only on the left and the right. This helps to get clear, easily readable schematics.
- **Group pins by functionality**, not by physical location of the leads or by datasheet. Always keep the typical application circuit in mind and choose pin locations which help to get clear schematics with only few crossed-over net lines. For example put *GND* exactly 5.08mm below the *VCC* pin if it's likely that capacitors need to be connected to them (capacitors have a height of 5.08mm). Or place *D+* and *D-* of a USB device right on top of each other (with the default distance of 2.54mm) as they are always used as a pair.
- **Use a pin length of 2.54mm** if possible. Other pin lengths should be used only in special cases.

### 3.1.6. Pin Naming

If the function of a pin is absolutely clear (e.g. anode/cathode of a diode), choose its abbreviated functionality as name (e.g. "A" for anode and "C" for cathode). If the functionality is not clear in the symbol (because it's defined by the component using that symbol), just use numbers starting with "1" at top left and increment them counterclockwise.

### 3.1.7. Text Elements

Typical symbols should have exactly two text elements: `{{NAME}}` and `{{VALUE}}`.



For rectangular symbols, the name should be placed at top left, aligned at bottom left to the corner of the symbol outlines. And the value should be placed at bottom left, aligned at the top left to the corner of the symbol outlines.



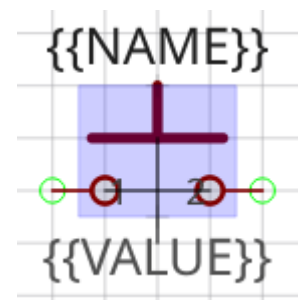
Irregularly shaped symbols may have text elements placed differently, see for example the crystal at the left. Keep in mind that the value of a component can consist of several lines, so there should always be enough space available for it.

Table 1. Typical text element properties

Property	Name text element	Value text element
Layer	<i>Names</i>	<i>Values</i>
Text	<code>{{NAME}}</code>	<code>{{VALUE}}</code>
Alignment	<i>Bottom Left</i>	<i>Top Left</i>
Height	<i>2.5mm</i>	<i>2.5mm</i>
Rotation	<i>0°</i>	<i>0°</i>

### 3.1.8. Grab Area

The grab area is the region of a symbol where it can be grabbed with the mouse (to move it, or to open the context menu). Symbols which have a single outline (like an IC) should typically have the "Grab Area" property set on the outline polygon (which makes the area filled with yellow color).



For symbols which have a more complex outline or which do not look nice with the yellow fill you should add an extra polygon to explicitly define the grab area. See the blue area of the push button for example. Ensure that the polygon doesn't overlap with pins and use following polygon properties:

- **Layer:** *Hidden Grab Areas* (will not be visible in the schematic editor)
- **Line Width:** *0.0 mm*
- **Filled:** *yes*
- **Grab Area:** *yes*



The origin cross of a symbol is always also an implicit grab area. So even if there is

no explicit grab area defined, the symbol can still be grabbed.

## 3.2. Package Conventions



These guidelines are not yet complete. Help us create sensible conventions [on GitHub!](#)

### 3.2.1. Scope

The most important thing to consider when creating a Package is the scope of it. Since LibrePCB handles footprints differently than other EDA tools, special attention is required here.

Think about the appearance of the part (the mechanical shape, dimension and color). If two parts look exactly (or *almost*) equal, they can use the same Package. If they look different, two separate Packages must be created.



**Don't think about the land pattern (i.e. footprint) of the part** — it's not relevant for this decision. Even if a package can be mounted differently on a PCB (e.g. a THT resistor can be mount horizontally or vertically) and thus require different footprints, only one Package is needed. Similarly, two different-looking parts that have the same land pattern (e.g. a SMD resistor and a SMD LED) should still be two separate Packages.

#### Example 1. Color (e.g. 0805 LED)

Even if a 0805 LED with a transparent lens has exactly the same footprint as a 0805 LED with a red lens, they should have **separate Packages because of the different color**. This way a Device can link to the Package with the proper color, and thus it will appear with the proper color in the 3D PCB preview (once LibrePCB supports 3D models).

#### Example 2. Height (e.g. SO-8)

Some packages are available in different heights. For instance, SO-8 is available with heights of 1.2mm and 1.4mm. **As the 3D models would be different, separate Packages are needed.**

*Note: To avoid creating too many Packages, a small tolerance is allowed. So for a device with a height of 1.3mm you might want to use the Package with a height of 1.4mm.*

#### Example 3. Mounting variants (e.g. TO220)

Many packages can be mounted either vertically or horizontally, for example the TO220. If mounted horizontally, there might be a hole in the PCB to screw the metal tab down to the PCB, or you may want to solder the tab to the PCB without a hole in it. For all these cases **only one Package is needed** — the different mounting variants should be handled by different footprint variants inside the Package.

### 3.2.2. Naming

The following conventions apply to Package names:

- We generally **follow IPC-7351 when naming packages** (e.g. "SOT23-5P95\_280X145L60" instead of "SOT23-5"). Alternative names (like "SOT23-5") should be added to the comma-separated keywords list and maybe to the description.
- For packages not covered by IPC-7351, use following naming conventions:
  - **Language must be American English** (en\_US), if applicable (many Packages have language-neutral names anyway).
  - **Size information must use metric units**, not imperial units.
  - For packages which are available with different pin counts, **append the pin count with a hyphen as separator and omit leading zeros** (e.g. "DIP-8" instead of "DIP08").
- For packages which are well known by their size in imperial units (e.g. "0805" which is "2012" in metric), it's recommended to write the well known name in parentheses. For example, a chip resistor could be named "RESC2012X70 (0805)".
- The name of manufacturer-specific packages should start with the manufacturers name (e.g. "Molex 53261-06"). *Note: Libraries do not act as namespaces for Package names, so you should start the Package name with the manufacturers name even if the Package is located in a manufacturer-specific library.*

### 3.2.3. Pads

- **Always add all pads of Packages**, even those which are not always connected. For example, the package "TO220" has a metal tab, so you should define it as a pad, no matter if it's often not connected (and even not connectable when mounted vertically).
- **Use pad names according IPC-7351** (if applicable). For Packages which are not covered by IPC-7351:
  - If the function of a pad is absolutely clear, choose its abbreviated functionality as name (e.g. "A" for anode and "C" for cathode).
  - Otherwise just use numbers starting with "1" at top left and increment them counterclockwise.

### 3.2.4. Footprint Variants

Within a Package there can be multiple footprint variants. They are intended to support the following use-cases:

- **Mounting variants:** For example, a THT resistor can be mounted either vertically or horizontally with various pad distances. Every common mounting variant should be available as footprint variants.
- **Soldering techniques:** Many packages can be soldered either by reflow-, wave- or hand-soldering, which usually require different land patterns. For every suitable soldering technique there could be a corresponding footprint variant.



- **Density levels:** [IPC-7351](#) specifies three different density levels for footprints:

- Density Level A: Maximum (Most) Land Protrusion
- Density Level B: Median (Nominal) Land Protrusion
- Density Level C: Minimum (Least) Land Protrusion

If applicable, these three density levels should also be added as footprint variants.

#### Combinations



As a given Package might support multiple of the use-cases above, all suitable combinations of them should be added. For example a Package which should have all three density levels as defined in IPC-7351 and can be mounted either vertically or horizontally would need six footprint variants to support all possible use-cases.

#### Set default footprint

**The first footprint is always the default footprint**, so you should move the most reasonable footprint to the top of the footprint list! The default footprint should fulfill these rules:





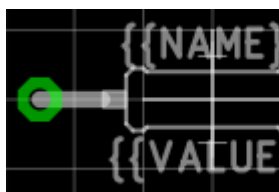
- Generic packages: Designed according to IPC density level B (if applicable)
- Manufacturer-specific packages: Designed according to datasheet
- Suitable for reflow soldering (if applicable)
- Most natural mounting variant (e.g. horizontal for THT resistors, or vertical for Transistor Outline packages)

Example 4. THT resistor 0207 footprint variants

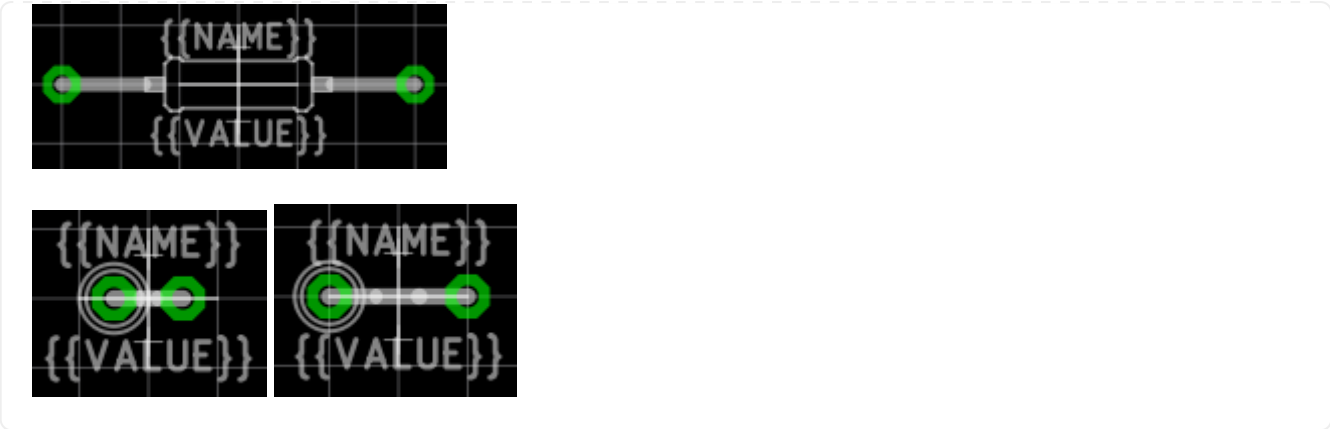
Footprint Variants

Default	Name	
63478879-99c7...	Horizontal 7.62mm	
b5afb23e-e9be...	Horizontal 10.16mm	
3cccaccf-5dc2...	Horizontal 12.7mm	
4b48782b-0e12...	Horizontal 15.24mm	
5658c521-17cd...	Vertical 2.54mm	
2f9b2f9d-2b17...	Vertical 5.08mm	
Add new footprint:		





### 3.2.5. Origin

The origin (0, 0) should be exactly at the center of the package body. It is used by pick and place machines.

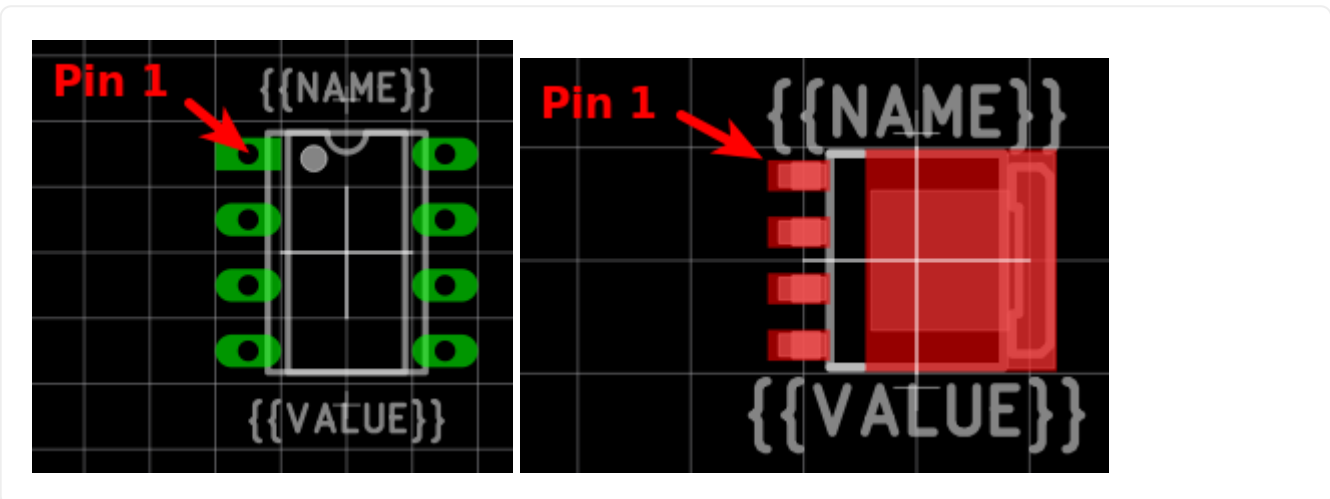
Some packages (especially those with non-symmetrical body) have the origin explicitly specified in the datasheet. In that case, use the origin from the datasheet.

### 3.2.6. Orientation

Footprints must be drawn from the top-view. When a footprint needs to appear on the bottom of a board, this can be done in the board editor by mirroring it.

Pin 1 should always be at the top left, as defined in [IPC-7351C "Level A", slide 22](#).

*Example 5. Footprint orientation examples*



### 3.2.7. Placement Layer

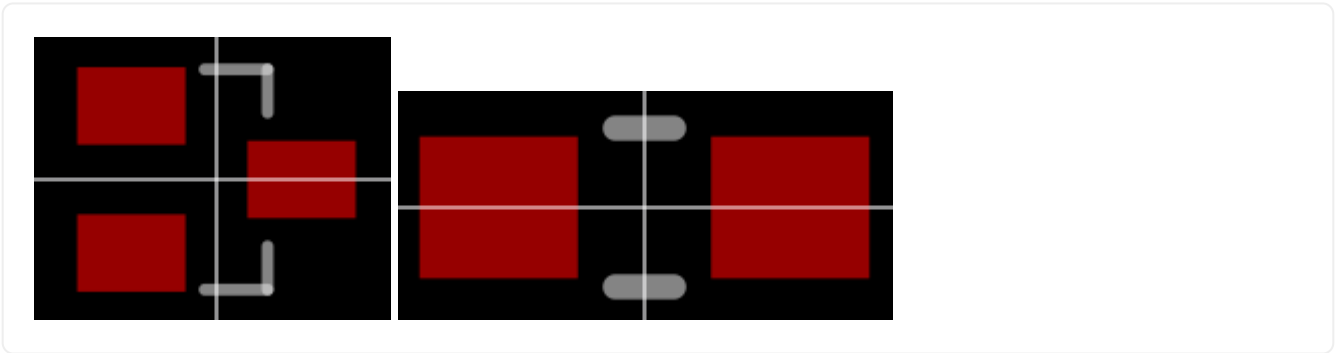
The *Top Placement* layer is intended to be printed on silkscreen and thus should contain information required for assembling the PCB. But don't put too many things on that layer as it would waste space on the PCB!

Typically this layer should only contain some lines and dots to indicate where and in which orientation the device gets assembled, for example an outline and a dot next to pin 1.

The placement should be drawn according to [IPC-7351C](#). The most important rules are the following:

- **It should stay visible after assembling the package** to allow reviewing positioning and orientation of assembled devices. In other words, the placement layer should primarily contain drawings *around* the package's body, but not *under* it.
- **Line width:** 0.2mm typical, 0.1mm minimum
- **Clearance to copper layers:** Equal or greater than the line width, but at least 0.15mm

*Example 6. Placement layer examples (only placement and copper layers shown)*



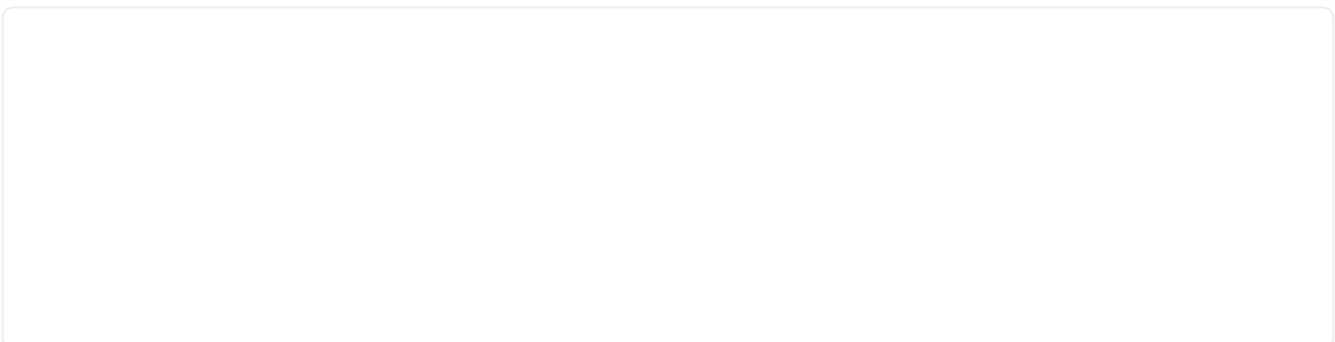
### 3.2.8. Documentation Layer

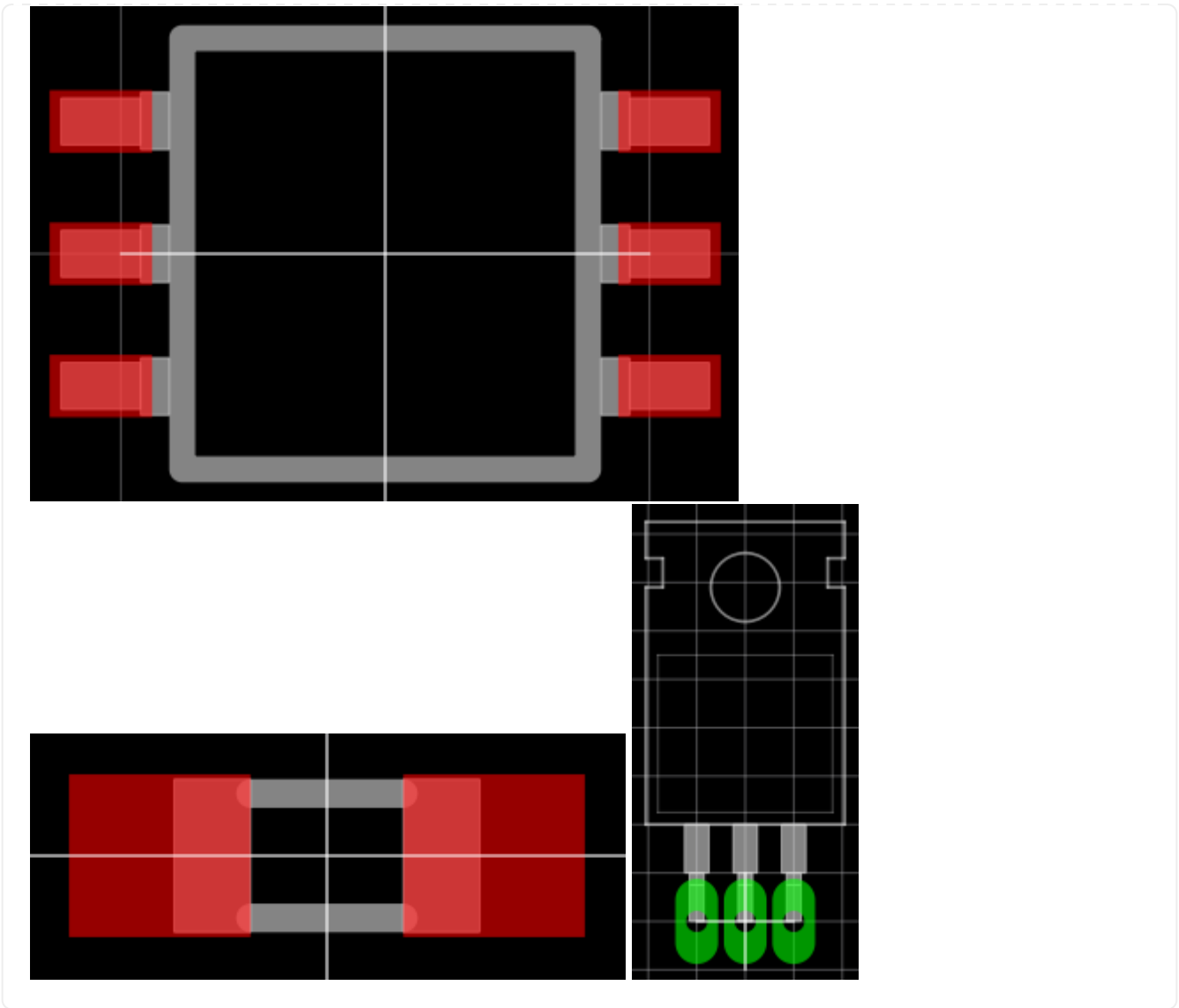
The layer *Top Documentation* should be used to draw the most important details of the package's appearance. It could be considered as an alternative to the 3D model of a package. But in contrast to the 3D model, the documentation layer is visible in the board editor while laying out the PCB.

Following things should be placed on the documentation layer:

- **The package's exact outline.** *Attention: The **outer** edges of the lines should correspond to the package's edges, **not** the middle of the lines! So, for example if the body is 5x5mm and the line width 0.2mm, you have to draw a 4.8x4.8mm rectangle.*
- **The top view of the leads/legs:** The leads or legs of both THT and SMT pads should be drawn from the top view, i.e. the vertical projection of them. This is needed to make packages look realistic on the documentation layer, as leads and legs are an important part of the appearance of packages.
- **The contact area of SMT leads:** The area where SMT leads touch the copper land pattern should be drawn as **filled polygons with a line width of 0mm**. This helps the PCB designer to see the expansion of the land pattern, i.e. how much copper is around the actual lead.

*Example 7. Documentation layer examples (only documentation and copper layers shown)*





### 3.2.9. Text Elements

Typical footprints should have exactly two text elements: `{{NAME}}` and `{{VALUE}}`.

The name should normally be placed at top of the package body, slightly above the outline and aligned at bottom center. The value should be placed at the bottom center, slightly below the package body and aligned at the top center.

**Always make sure that the text elements do not overlap with pads or with the placement layer.** Otherwise the text might be unreadable on silkscreen. In addition, text elements should usually be placed outside the Package body to still see them on silkscreen of an assembled PCB.

Keep in mind that the bottom-aligned anchor is placed on the text baseline. This means that some letters like "g" or "y" might extend slightly below the anchor.

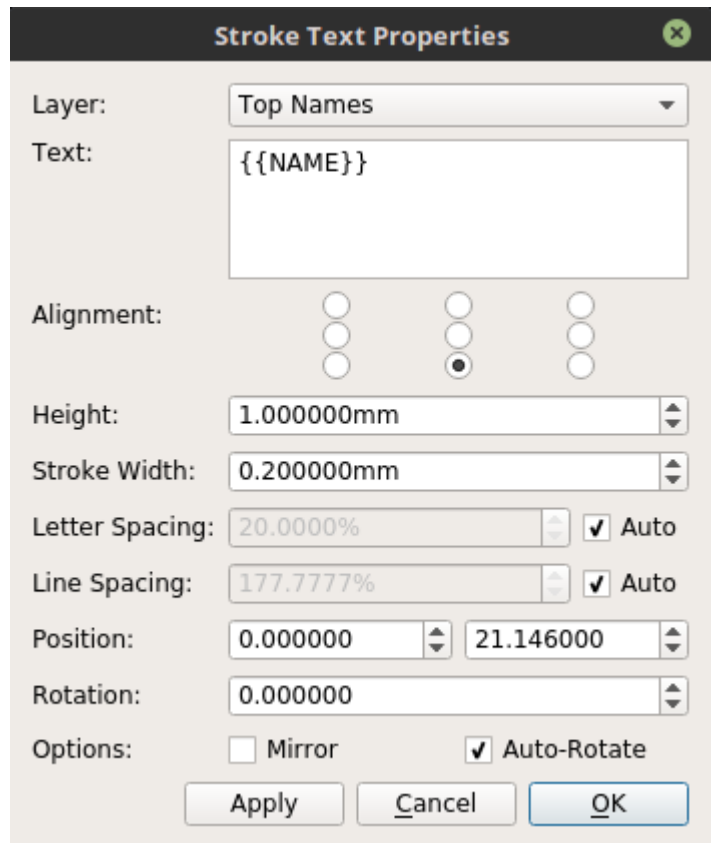


Figure 1. Typical footprint name properties

Table 2. Typical text element properties

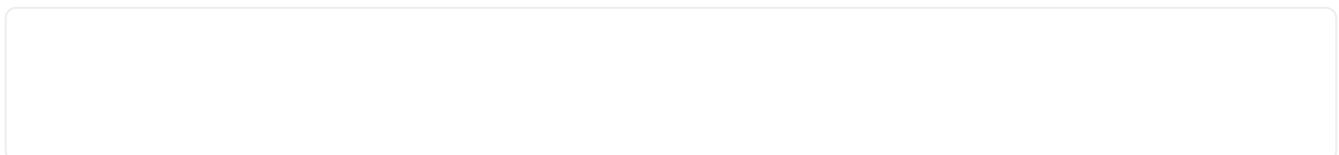
Property	Name text element	Value text element
Layer	Top Names	Top Values
Text	{{NAME}}	{{VALUE}}
Alignment	Bottom Center	Top Center
Height	1.0mm (or larger)	1.0mm (or larger)
Stroke Width	0.2mm (or thicker)	0.2mm (or thicker)
Letter Spacing	Auto	Auto
Line Spacing	Auto	Auto
Mirror	No	No
Auto-Rotate	Yes	Yes

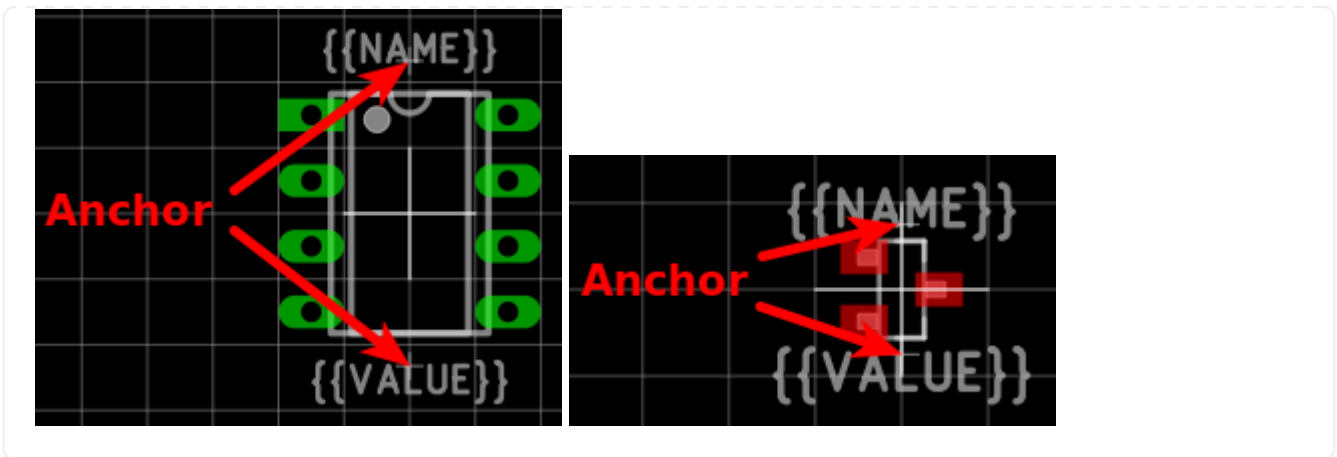


*Special cases*

These rules should be fine for many Packages, but probably not for all of them. For special cases it's allowed to have slightly different properties if they are more suitable.

Example 8. Footprint text element examples





## 4. LibrePCB Command Line Interface

LibrePCB also provides a command line interface (CLI). With that tool, you can automate some tasks, for example on Continuous Integration (CI) systems.

### 4.1. Installation

#### Online Installer

If you have installed LibrePCB with the installer (as described [here](#)), you should already have the `librepcb-cli` tool available in the installation directory.

#### Portable Package

Alternatively you could run `librepcb-cli` without installing it.

#### Windows

Download and extract [librepcb-0.1.7-windows-x86.zip](#), then run the contained file `bin\librepcb-cli.exe`.

#### Linux

##### AppImage

Download [librepcb-cli-0.1.7-linux-x86\\_64.AppImage](#), make it executable and start it:

```
wget "https://download.librepcb.org/releases/0.1.7/librepcb-cli-0.1.7-linux-x86_64.AppImage"
chmod +x ./librepcb-cli-0.1.7-linux-x86_64.AppImage
./librepcb-cli-0.1.7-linux-x86_64.AppImage
```

##### Binaries

Alternatively, you can download and extract [librepcb-0.1.7-linux-x86\\_64.tar.gz](#) which contains the

`librepcb-cli` executable:

```
wget "https://download.librepcb.org/releases/0.1.7/librepcb-0.1.7-linux-x86_64.tar.gz"
tar -xvzf ./librepcb-0.1.7-linux-x86_64.tar.gz
./librepcb-0.1.7-linux-x86_64/bin/librepcb-cli
```

## Docker

Or if you have [Docker](#) installed, you can use our official image `librepcb/librepcb-cli`:

```
docker run -it --rm -u $(id -u):$(id -g) -v `pwd`:/work librepcb/librepcb-cli:0.1.7
```

## Mac

Download [librepcb-cli-0.1.7-mac-x86\\_64.dmg](#) and double-click it. Then drag and drop the app onto the "Applications" icon of Finder.

## 4.2. Running on Headless Linux

Please note that (at this time) `librepcb-cli` requires a running X-server even if it doesn't open any windows. If your system doesn't have an X-server running, you can use [xvfb](#) instead:

```
$ xvfb-run -a librepcb-cli [args]
```

If the `librepcb-cli` executable still doesn't work, you may need to install some dependencies. On Debian/Ubuntu, following packages need to be installed:

```
$ apt-get install libfontconfig1 libglib2.0-0 libglu1-mesa
```

## 4.3. Usage

Usage instructions and available options can be shown with `--help`:

### Command

```
$ ./librepcb-cli --help
```

### Output

```
Usage: ./librepcb-cli [options] command
LibrePCB Command Line Interface

Options:
  -h, --help    Print this message.
```

```
-v, --version  Displays version information.
--verbose     Verbose output.
```

Arguments:

```
command      The command to execute (see list below).
```

Commands:

```
open-library  Open a library to execute library-related tasks.
open-project  Open a project to execute project-related tasks.
```

List command-specific options:

```
./librepcb-cli <command> --help
```

### 4.3.1. Command `open-library`

This command opens a LibrePCB library and lets you execute some tasks with it.

*Command*

```
$ ./librepcb-cli open-library --help
```

*Output*

```
Usage: ./librepcb-cli [options] open-library [command_options] library
LibrePCB Command Line Interface
```

Options:

```
-h, --help      Print this message.
-v, --version   Displays version information.
--verbose       Verbose output.
--all           Perform the selected action(s) on all elements contained in
                the opened library.
--save          Save library (and contained elements if '--all' is given)
                before closing them (useful to upgrade file format).
--strict        Fail if the opened files are not strictly canonical, i.e.
                there would be changes when saving the library elements.
```

Arguments:

```
open-library  Open a library to execute library-related tasks.
library       Path to library directory (*.lplib).
```

#### Example: Check and Upgrade File Format of Library Elements

This command is useful for Continuous Integration of LibrePCB libraries because it reports failure if you check in libraries with invalid or non-canonical S-Expression files.

*Command*

```
$ ./librepcb-cli open-library \
```

```
--all \  
--strict \  
MyLibrary.lplib
```

### Output

```
Open library 'MyLibrary.lplib'...  
Process 86 component categories...  
Process 44 package categories...  
Process 37 symbols...  
Process 492 packages...  
Process 34 components...  
Process 37 devices...  
SUCCESS
```

## 4.3.2. Command `open-project`

This command opens a LibrePCB project and lets you execute some tasks with it.

### Command

```
$ ./librepcb-cli open-project --help
```

### Output

```
Usage: ./librepcb-cli [options] open-project [command_options] project  
LibrePCB Command Line Interface  
  
Options:  
-h, --help                Print this message.  
-v, --version              Displays version information.  
--verbose                  Verbose output.  
--erc                       Run the electrical rule check, print all  
                             non-approved warnings/errors and report  
                             failure (exit code = 1) if there are  
                             non-approved messages.  
--export-schematics <file> Export schematics to given file(s).  
                             Existing files will be overwritten.  
                             Supported file extensions: pdf, svg, png  
--export-bom <file>        Export generic BOM to given file(s).  
                             Existing files will be overwritten.  
                             Supported file extensions: csv  
--export-board-bom <file> Export board-specific BOM to given  
                             file(s). Existing files will be  
                             overwritten. Supported file extensions: csv  
--bom-attributes <attributes> Comma-separated list of additional  
                             attributes to be exported to the BOM.  
                             Example: "MANUFACTURER, MPN"  
--export-pcb-fabrication-data Export PCB fabrication data
```



	(Gerber/Excellon) according the fabrication output settings of boards. Existing files will be overwritten.
<code>--pcb-fabrication-settings &lt;file&gt;</code>	Override PCB fabrication output settings by providing a *.lp file containing custom settings. If not set, the settings from the boards will be used instead.
<code>--export-pnp-top &lt;file&gt;</code>	Export pick&place file for automated assembly of the top board side. Existing files will be overwritten. Supported file extensions: csv, gbr
<code>--export-pnp-bottom &lt;file&gt;</code>	Export pick&place file for automated assembly of the bottom board side. Existing files will be overwritten. Supported file extensions: csv, gbr
<code>--board &lt;name&gt;</code>	The name of the board(s) to export. Can be given multiple times. If not set, all boards are exported.
<code>--board-index &lt;index&gt;</code>	Same as '--board', but allows to specify boards by index instead of by name.
<code>--remove-other-boards</code>	Remove all boards not specified with '--board[-index]' from the project before executing all the other actions. If '--board[-index]' is not passed, all boards will be removed. Pass '--save' to save the modified project to disk.
<code>--save</code>	Save project before closing it (useful to upgrade file format).
<code>--strict</code>	Fail if the project files are not strictly canonical, i.e. there would be changes when saving the project. Note that this option is not available for *.lppz files.
<b>Arguments:</b>	
<code>open-project</code>	Open a project to execute project-related tasks.
<code>project</code>	Path to project file (*.lpp[z]).

### Example: Check ERC Messages and Export Schematics & Boards

This command is useful for Continuous Integration of LibrePCB projects because it reports failure if you check in projects with non-approved ERC messages. In addition, it generates all production data so you don't have to do it manually.

#### Command

```
$ ./librepcb-cli open-project \
  --erc \
  --export-schematics="output/{{VERSION}}/{{PROJECT}}_Schematics.pdf" \
  --export-pcb-fabrication-data \
```

### Output

```
Open project 'MyProject.lpp'...
Run ERC...
  Approved messages: 7
  Non-approved messages: 2
    - [WARNING] Net signal connected to less than two pins: "CAN_RX"
    - [WARNING] Net signal connected to less than two pins: "JTCK"
Export schematics to 'output/{{VERSION}}/{{PROJECT}}_Schematics.pdf'...
=> 'output/v1/MyProject_Schematics.pdf'
Export PCB fabrication data...
Board 'default':
=> 'output/v1/gerber/MyProject_DRILLS-PTH.dr1'
=> 'output/v1/gerber/MyProject_OUTLINES.gbr'
=> 'output/v1/gerber/MyProject_COPPER-TOP.gbr'
=> 'output/v1/gerber/MyProject_COPPER-BOTTOM.gbr'
=> 'output/v1/gerber/MyProject_SOLDERMASK-TOP.gbr'
=> 'output/v1/gerber/MyProject_SOLDERMASK-BOTTOM.gbr'
=> 'output/v1/gerber/MyProject_SILKSCREEN-TOP.gbr'
=> 'output/v1/gerber/MyProject_SILKSCREEN-BOTTOM.gbr'
Finished with errors!
```

In this example, the application reported errors and exited with code 1 because there are non-approved ERC messages.