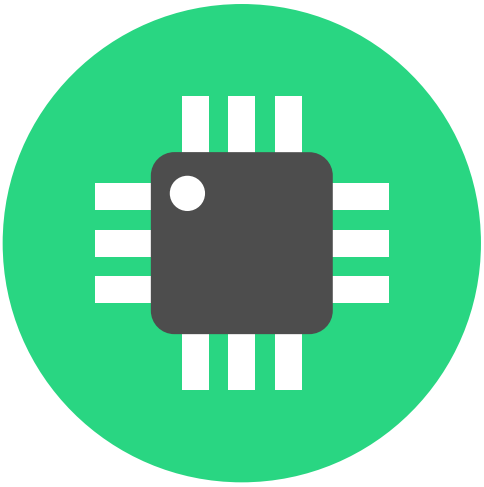


LibrePCB Documentation

2026-06-12

Table of Contents

Welcome to the documentation of LibrePCB 2.1.1!

The documentation is still work in progress. Help us writing beautiful documentation on GitHub!

Video Tutorials

In addition to this documentation in written form, there are also video tutorials available on our YouTube channel.

Offline Documentation

For offline- or printable documentation, use the PDF download link at the bottom left of the page.

Chapters:

Didn't find what you're looking for? Contact us!

Installation

Official Binaries

We provide official binary releases for the following operating systems:

- Windows
- Linux
- macOS

Distribution Packages

In addition, we are officially maintaining the following packages:

- Snap on Snapcraft
- Flatpak on Flathub

For other systems, a LibrePCB package might be provided by a package maintainer, either partially related or unrelated to the LibrePCB developers. We are aware of the following packages:

- Chocolatey Package (Windows)
- Homebrew Cask Package (MacOS)
- Arch Linux AUR Package (builds from source)
- NixOS Package
- Gentoo Package
- OpenPandora Package
- Void Linux Package

As these packages are not under our control, we cannot guarantee their genuineness and correctness.

You're a LibrePCB package maintainer? Ask us to list your package here!

Build From Sources

Since LibrePCB is a free & open-source application, you can compile it by yourself if you like. This allows to run LibrePCB even on systems where no pre-built binaries are available. See instructions at Build From Sources.

On Windows

Installer

The recommended way to install LibrePCB is to use the installer.

Just download and run librepcb-installer-2.1.1-windows-x86_64.exe. Afterwards you'll find LibrePCB in your start menu.

Starting from LibrePCB 2.0, our official releases are signed by **Scheibling Consulting AB** as you may see when running LibrePCB the first time. This code signing service is thankfully sponsored by OSSign.

For automated (unattended) installation, please check out the command-line parameters of the Inno Setup framework here (uninstall):

```
librepcb-installer-2.1.1-windows-x86_64.exe /VERYSILENT
/SUPPRESSMSGBOXES
```

Portable Package

Alternatively you could run LibrePCB without installing it. But then you don't get start menu entries and LibrePCB file extensions won't be registered so you can't open LibrePCB projects with a double-click in the file manager.

Download and extract librepcb-2.1.1-windows-x86_64.zip, then run the contained file `bin\librepcb.exe`.

On Linux

Due to the diversity of the Linux ecosystem, there are many different ways to install LibrePCB. The order of the options provided below do not reflect any recommendation.

If you're unsure, here our recommendations:

- On Ubuntu: Snap Package
- On a Raspberry Pi: Flatpak
- Everywhere else: Portable AppImage

Portable AppImage (x86_64)

The AppImage is a single-file portable package which runs on most Linux distributions. It is fully functional without installing anything on your system, but it does not provide an update mechanism.

Download librepcb-2.1.1-linux-x86_64.AppImage, make it executable and run it:

```
wget "https://download.librepcb.org/releases/2.1.1/librepcb-2.1.1-linux-
```

```
x86_64.AppImage"
chmod +x ./librepcb-2.1.1-linux-x86_64.AppImage
./librepcb-2.1.1-linux-x86_64.AppImage
```

If you're not familiar with the terminal: Right-click on the downloaded file and then check something like *Allow executing file as program* or *Run as executable*. Afterwards double-click the file to run it.

If LibrePCB silently doesn't start, run the AppImage from a terminal to see any error messages. It is possible that you need to install some system libraries.

Especially from Ubuntu 26.04 onwards, it seems that OpenGL needs to be installed in order to get LibrePCB functional: `sudo apt install libglu1-mesa`

Snap Package (multi-arch)

For distrubutions like Ubuntu which use the Snap package manager, probably the easiest way is to install the LibrePCB Snap package.

On Ubuntu, just open the *Ubuntu Software* application (app store), search for LibrePCB and install it. Alternatively, run this command from in the terminal:

```
sudo snap install librepcb
```

Some users reported that LibrePCB crashes when installed as a Snap package. It seems to be a problem related to fonts and Snap. If you experience this issue, the following workaround might help:

```
sudo rm /var/cache/fontconfig/*
rm ~/.cache/fontconfig/*
fc-cache -r
```

For more information about Snap, check out its documentation.

Flatpak (multi-arch)

LibrePCB is also available as a Flatpak package from Flathub. Assuming you have followed the Flatpak setup steps, you can configure Flathub and install LibrePCB as follows:

After installing Flatpak itself, make sure to **reboot the computer** before executing the follwing commands! Otherwise LibrePCB might not appear in your application launcher.

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

```
flatpak install flathub org.librepcb.LibrePCB
```

Online Installer (abandoned)

Note that starting with LibrePCB 1.0, we do no longer provide an installer for Linux. If you installed a previous LibrePCB release with the installer, please uninstall it with the *LibrePCB Maintenance Tool* and install the latest release with a different installation method instead.

On macOS

Portable Package

To install LibrePCB, download the portable `*.dmg` file matching your CPU architecture:

- **Intel (x86_64)**: librepcb-2.1.1-mac-x86_64.dmg
- **Apple Silicon (arm64)**: librepcb-2.1.1-mac-arm64.dmg

Double-click the downloaded file in Finder. Then drag and drop the LibrePCB app onto the "Applications" folder in Finder. Afterwards you'll find LibrePCB in the Launchpad.

Starting from LibrePCB 2.x, our official releases are signed by **Scheibling Consulting AB** as you may see when running LibrePCB the first time. This code signing service is thankfully sponsored by OSSign.

Build From Sources

Requirements

To compile LibrePCB, you need to install the following tools & libraries first:

- g++, MinGW or Clang (any version with C++20 support should work)
- Rust >= 1.88 toolchain (GNU, not MSVC)
- Qt >= 6.2 (make sure the imageformats plugin is installed too as it will be needed at runtime)
- OpenCASCADE OCCT or OCE (optional, OCCT highly preferred)
- OpenGL Utility Library GLU (optional)
- OpenSSL
- CMake 3.22 or newer

Get the Sources

It is very important to use the correct sources:

- **Do NOT clone any branch (e.g. `master`) from our repository on GitHub!** These sources are not compatible with the stable file format of LibrePCB.

> - Do NOT use the archives provided at the GitHub Releases page. These do not include the submodules and thus can't be compiled.
> - It's fine to clone the official release **tag** (current: `2.1.1`) from our repository on GitHub, just keep in mind to pass `--recursive` to also get all the submodules.

For convenience, we provide an official source archive which contains all the required files (including submodules) and has stripped any unnecessary files: librepcb-2.1.1-source.zip

```
wget "https://download.librepcb.org/releases/2.1.1/librepcb-2.1.1-source.zip"
unzip ./librepcb-2.1.1-source.zip
cd ./librepcb-2.1.1
```

Build LibrePCB

Within the downloaded source directory, execute the following commands:

```
mkdir build && cd build
cmake ..
make -j8
```

Additional Resources

These are just the most important commands. For more details (e.g. the available configuration flags), check out the following resources:

- `README.md` within the source archive
- Build instructions on our developers documentation

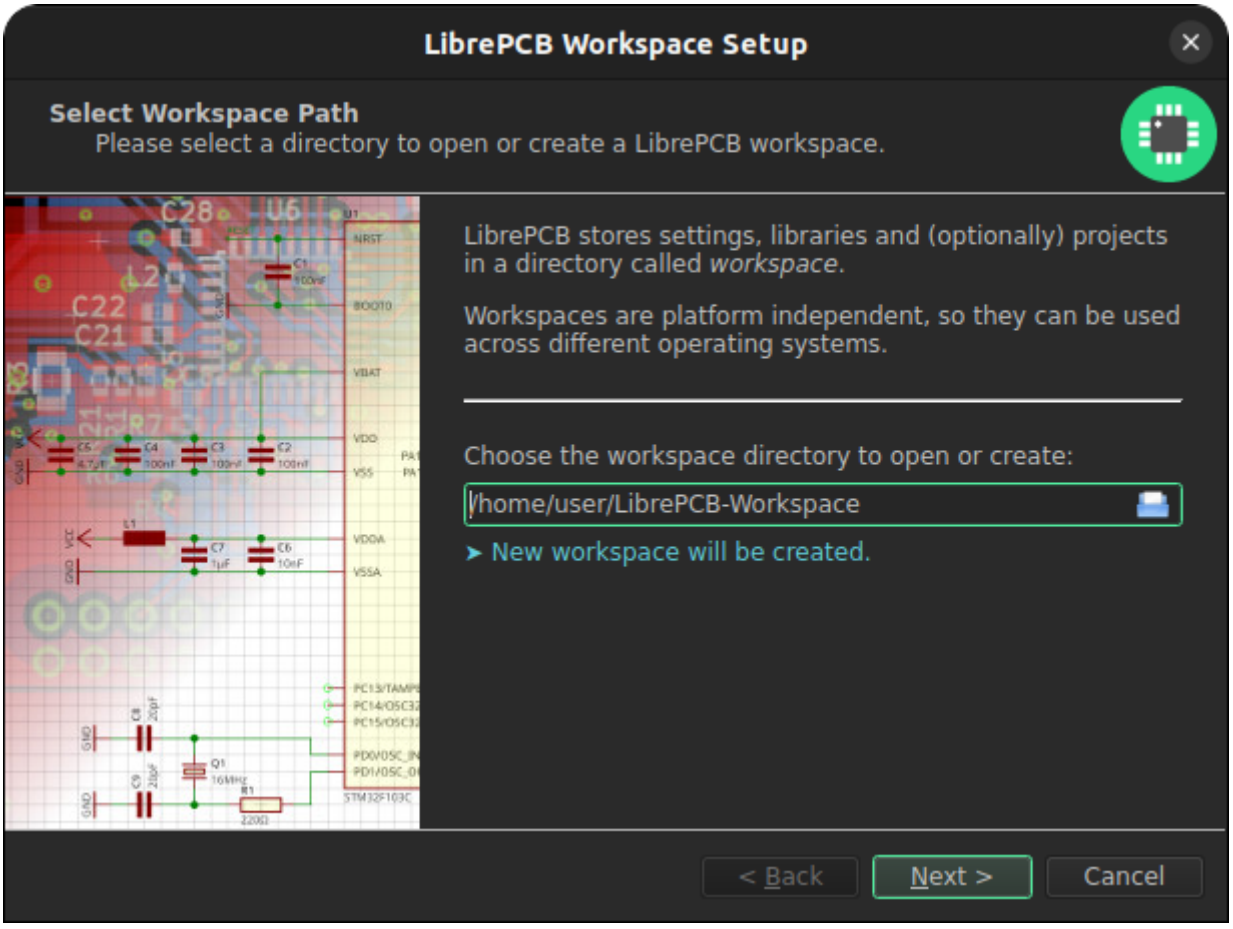
Quickstart Tutorial

This chapter provides a quick introduction into LibrePCB, starting from workspace initialization and ending with how to order the designed PCB.

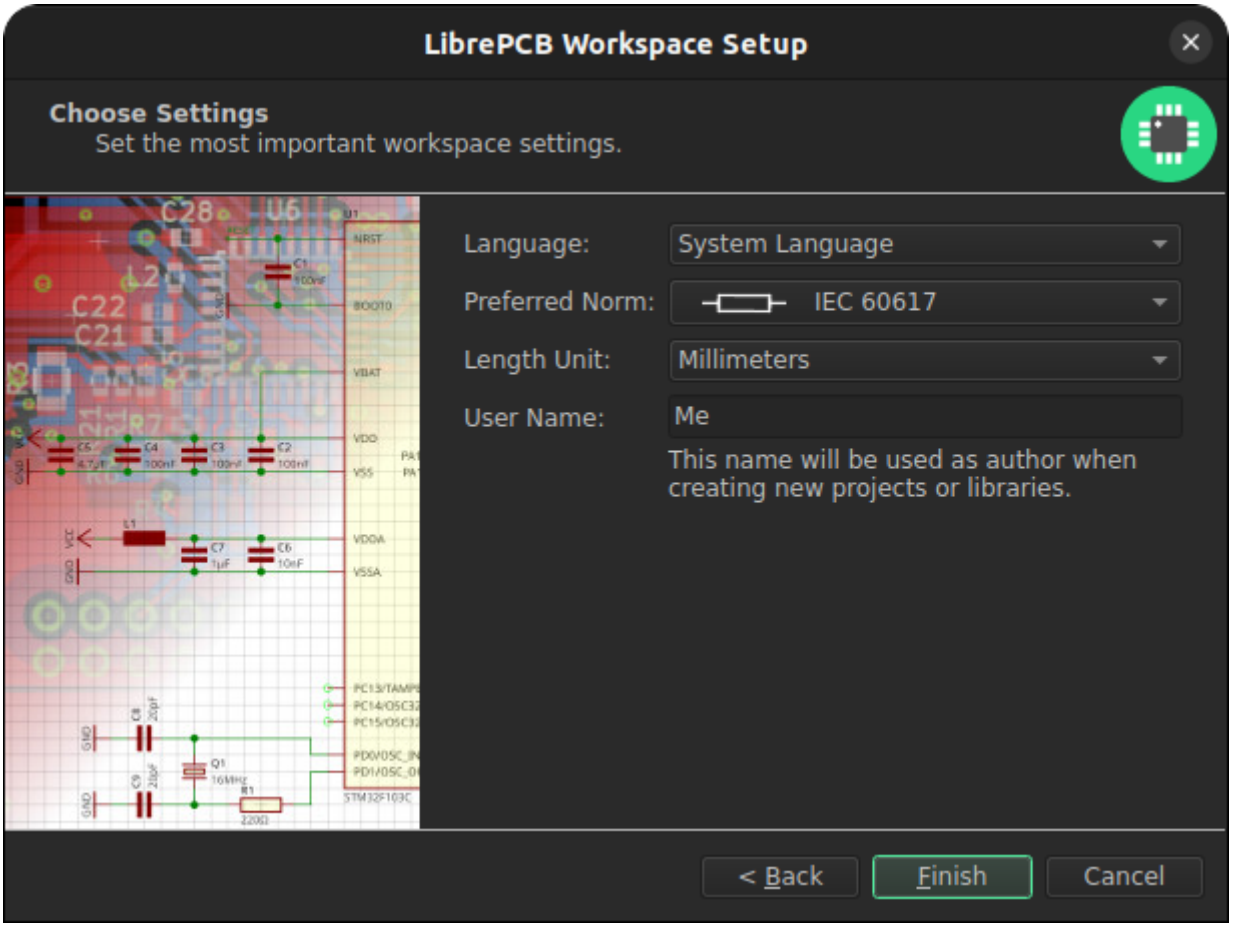
Create a Workspace

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 from any LibrePCB version.

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



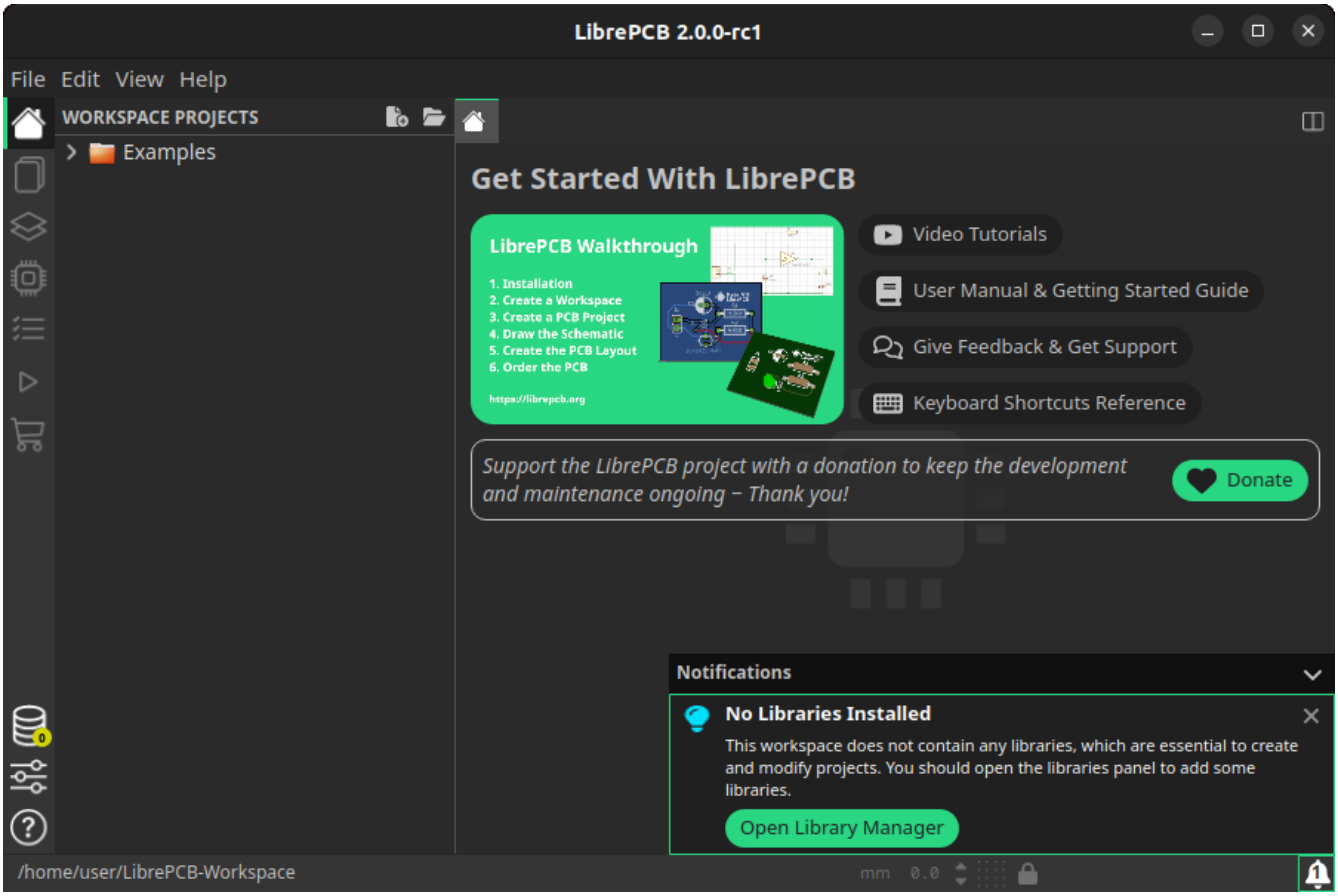
If the selected path does not contain a workspace yet, clicking on **[Next]** will show a page to choose the most important settings:



It is recommended to select at least your **preferred norm** and **length unit** since these usually depend on where you're living.

You can change these settings at any time later in the main window under **File › Workspace Settings**.

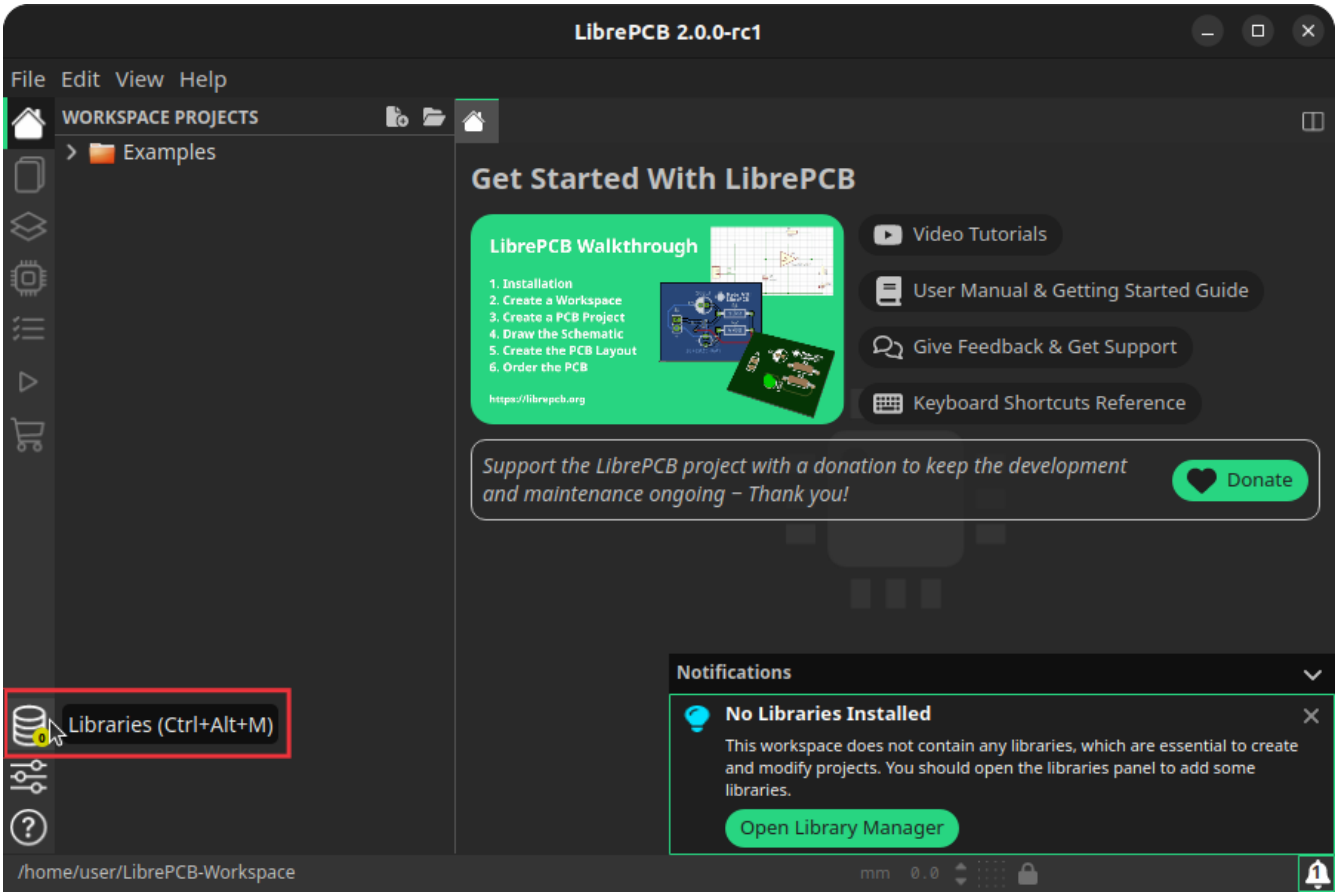
After clicking **[Finish]**, the main window shows up and you're ready to start using LibrePCB!



Install Remote Libraries

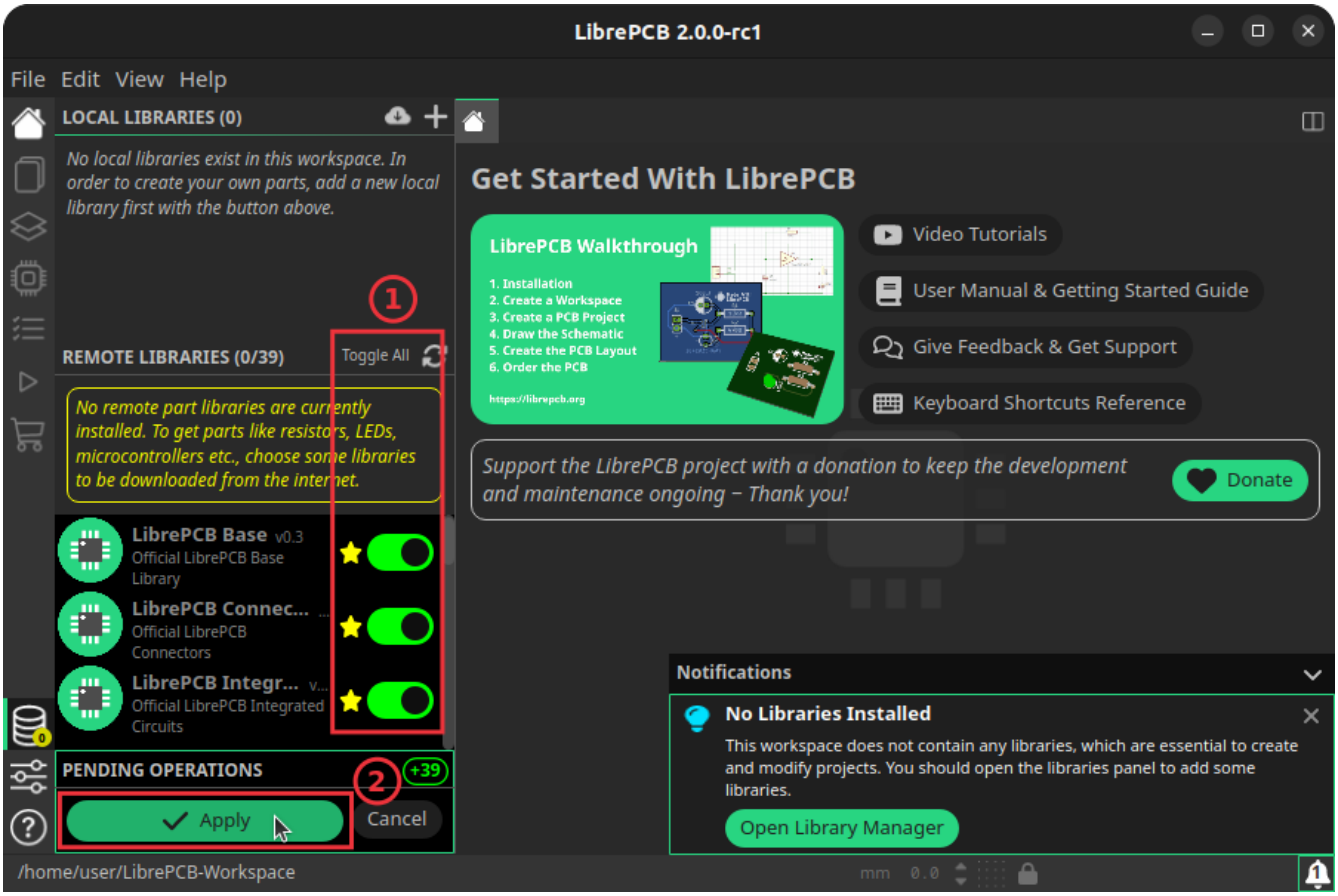
Before you can start creating new projects, you need to add some libraries to your workspace. Libraries contain various kinds of elements which can be added to schematics and boards (e.g. symbols, footprints and devices).

Click on **[Libraries]** in the side bar:



LibrePCB then immediately fetches the list of available libraries from the Internet. Most of these libraries are hosted at github.com/LibrePCB-Libraries.

Then you can select the libraries you like to install. The most important library is *LibrePCB Base* because it contains commonly used library elements like resistors or diodes. It is highly recommended to install at least this library. However, you can even simply install all the available libraries at once by clicking on **[Toggle All]**:



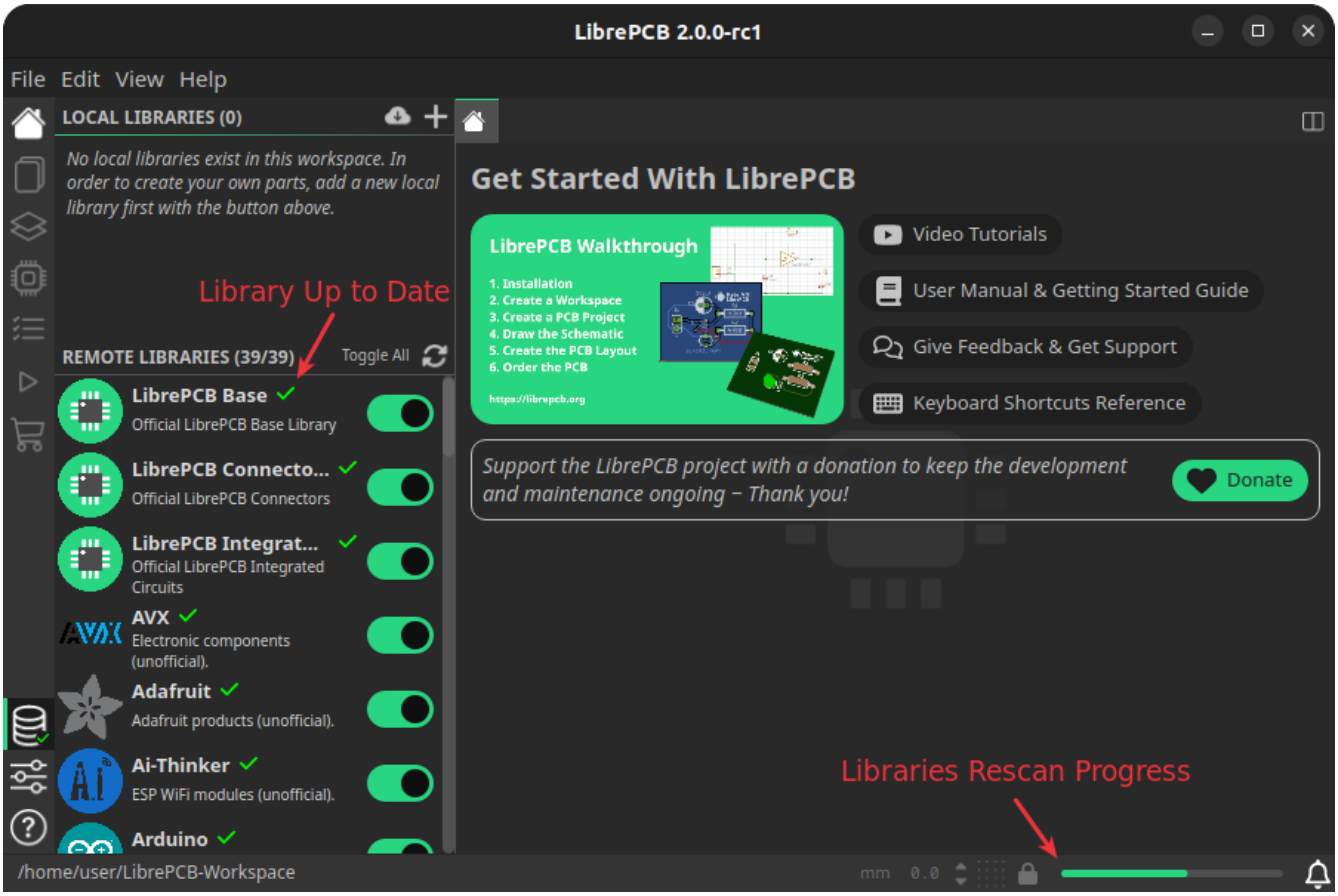
Later you can keep the installed libraries up to date the same way — once you open the libraries panel, all outdated libraries will automatically be marked for update and you only need to click on the **[Apply]** button.

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 is no problem, just follow this tutorial to create your own local library later. In a local library you can use or even override library elements from remote libraries by specifying a higher version number.

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 `data/libraries/local/` in your workspace.

After the selected libraries have been installed, they will have a green check mark to indicate that they are up to date:

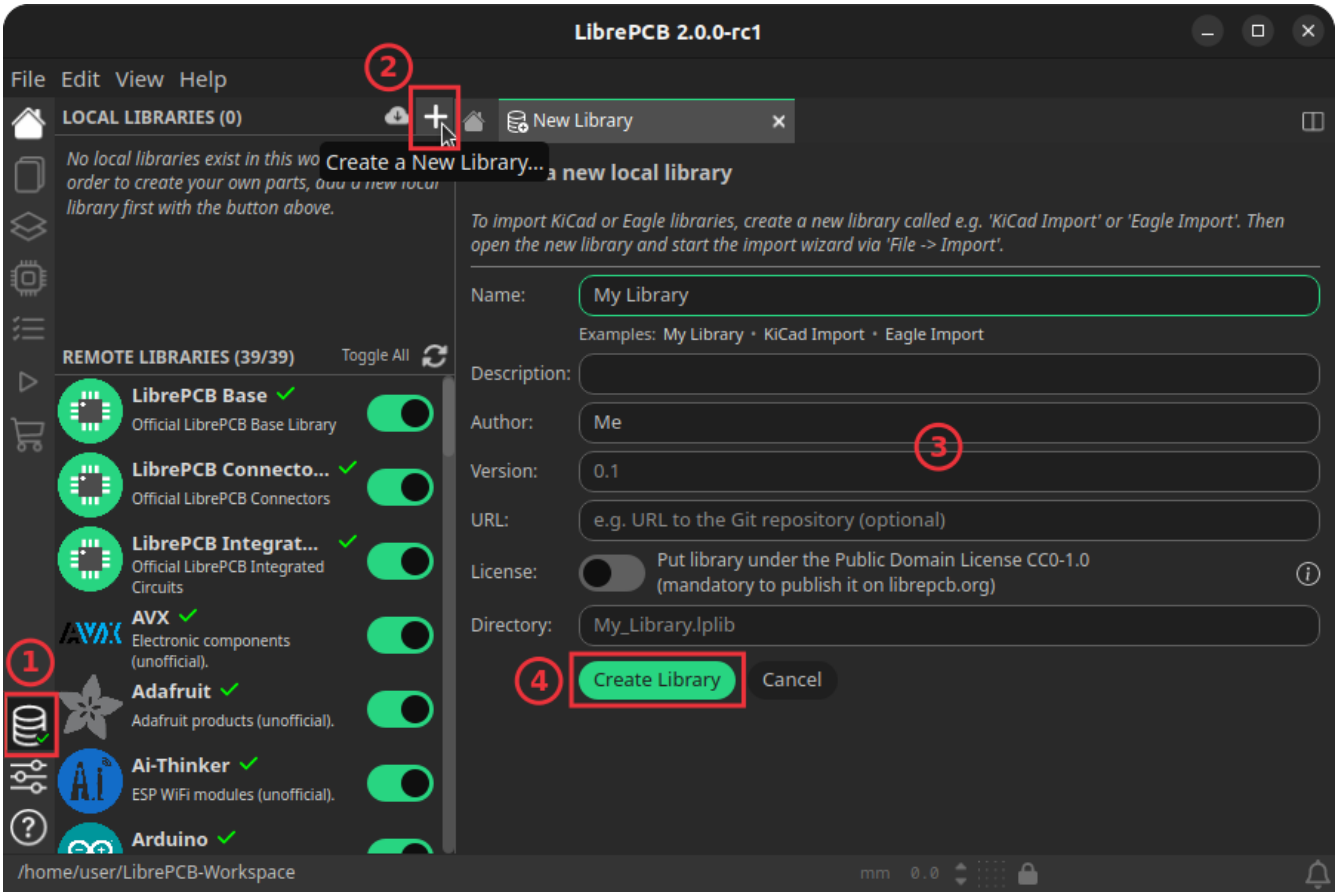


Note that after the libraries were installed, it takes a moment to create an index of all the contained elements. This process automatically runs in background and is indicated with a progress bar at the bottom right of all main window. The installed libraries are ready to use once the progress bar disappears.

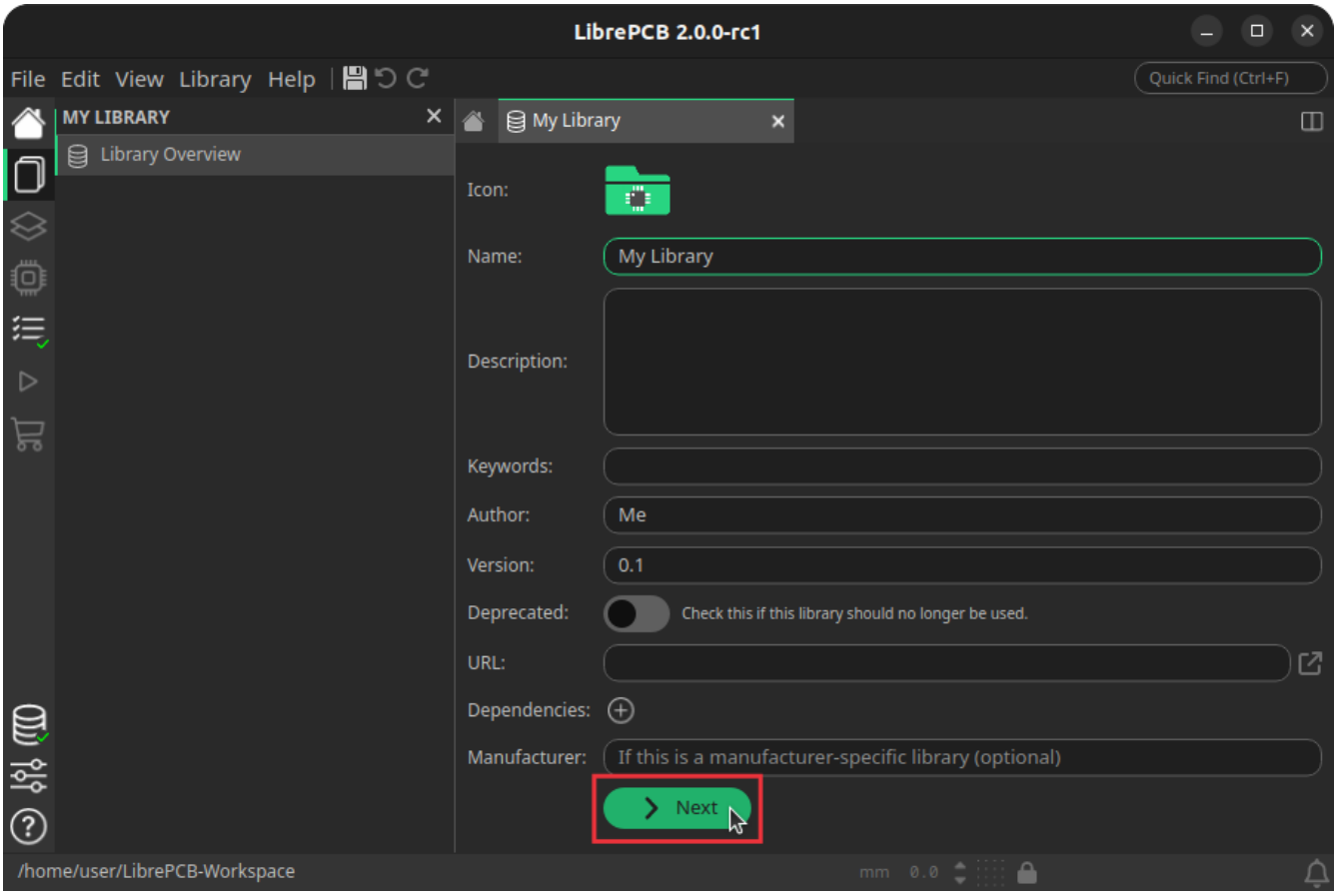
Create a Local Library

In addition to the (read-only) remote libraries, you should create a personal, so-called *local* library. This is the place where you'll add your own symbols, footprints etc. later.

To do so, click on the to the **[Create a New Library]** button, optionally enter some metadata (default values are usually good enough) and click on **[Create Library]**:

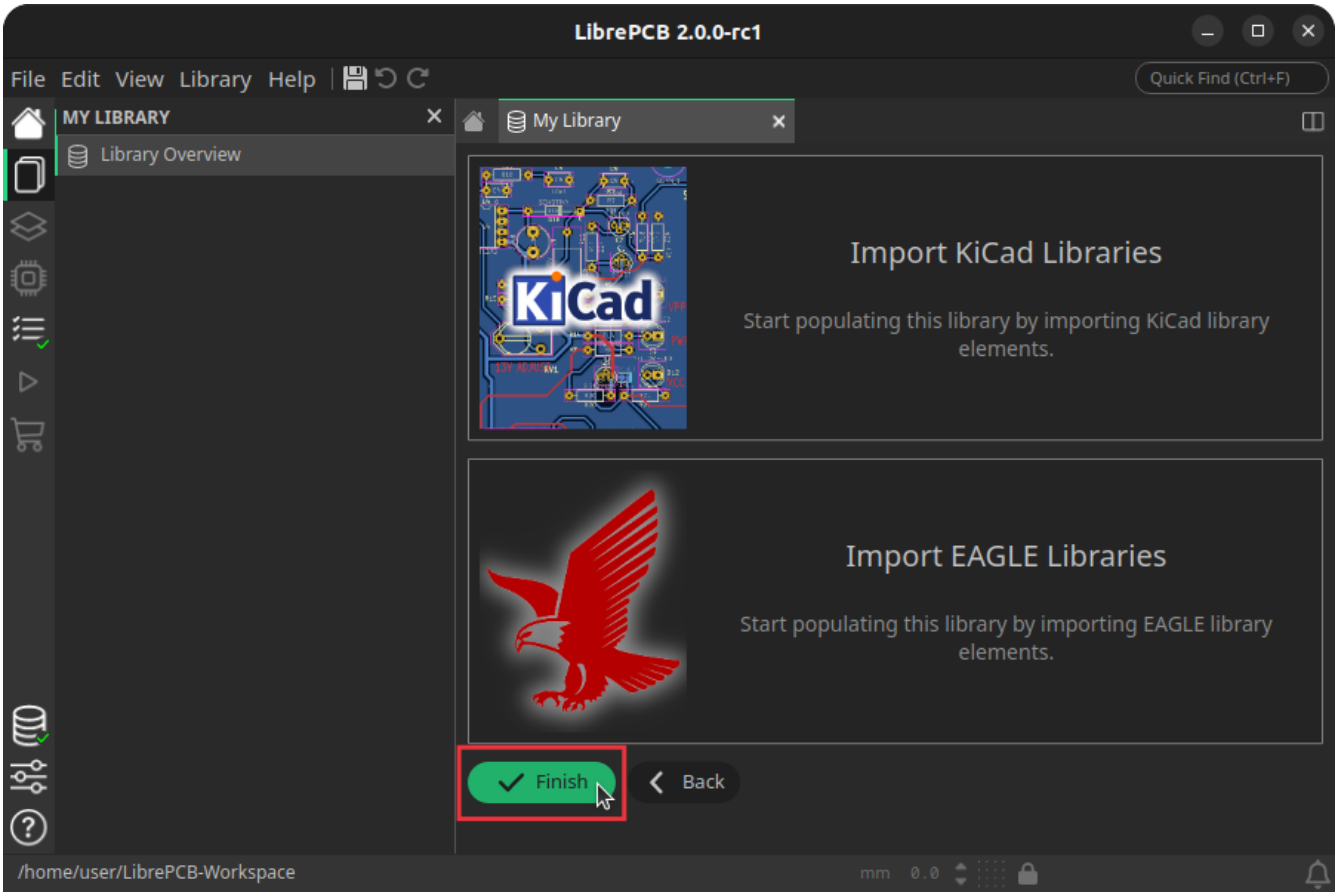


Afterwards you could enter more metadata of the library, but that's usually not required so you can just click on **[Next]**:

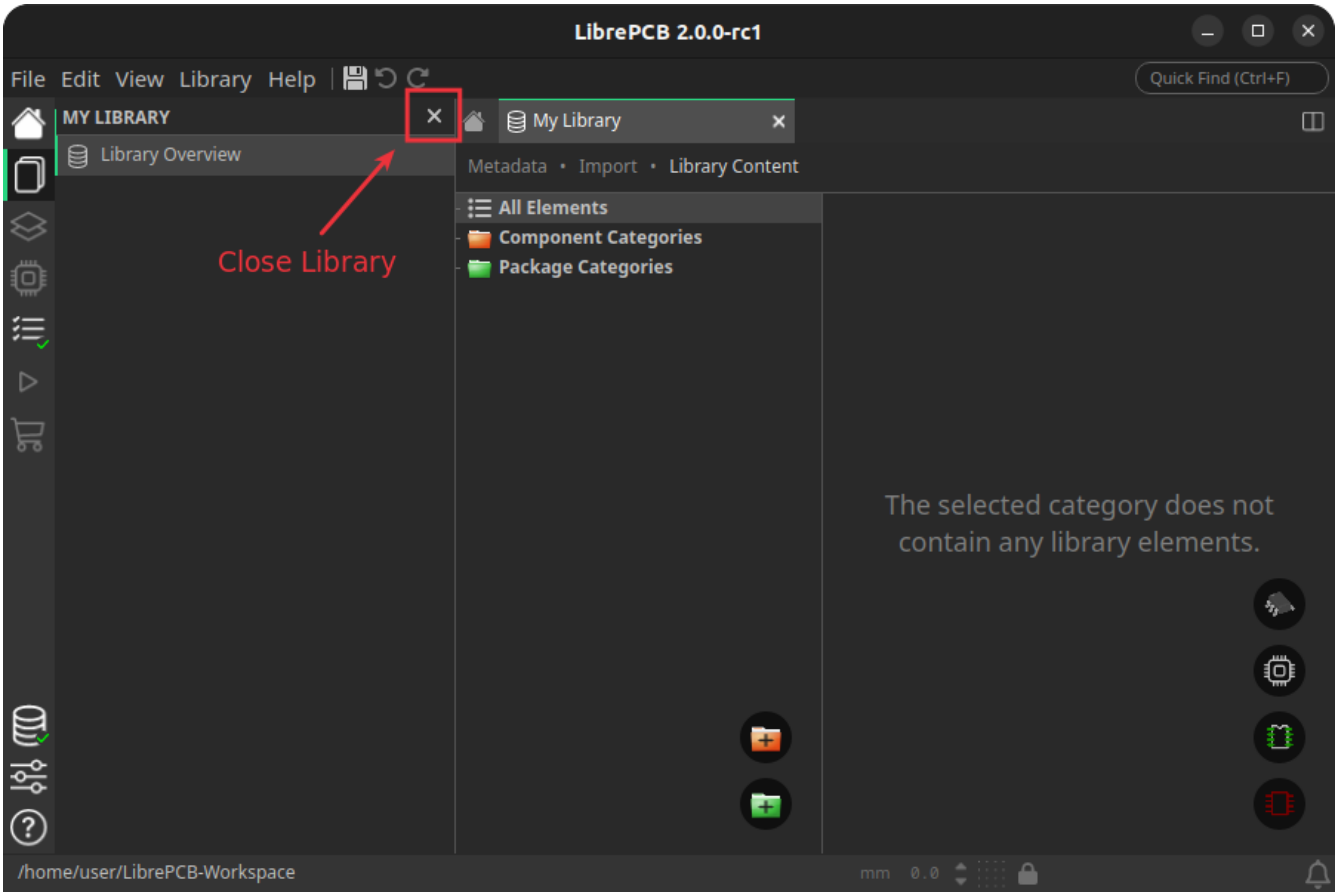


If you switched from KiCad or Eagle to LibrePCB, you could import your existing libraries now.

However, this can also be done at any time later, so you can just choose **[Finish]** for now:



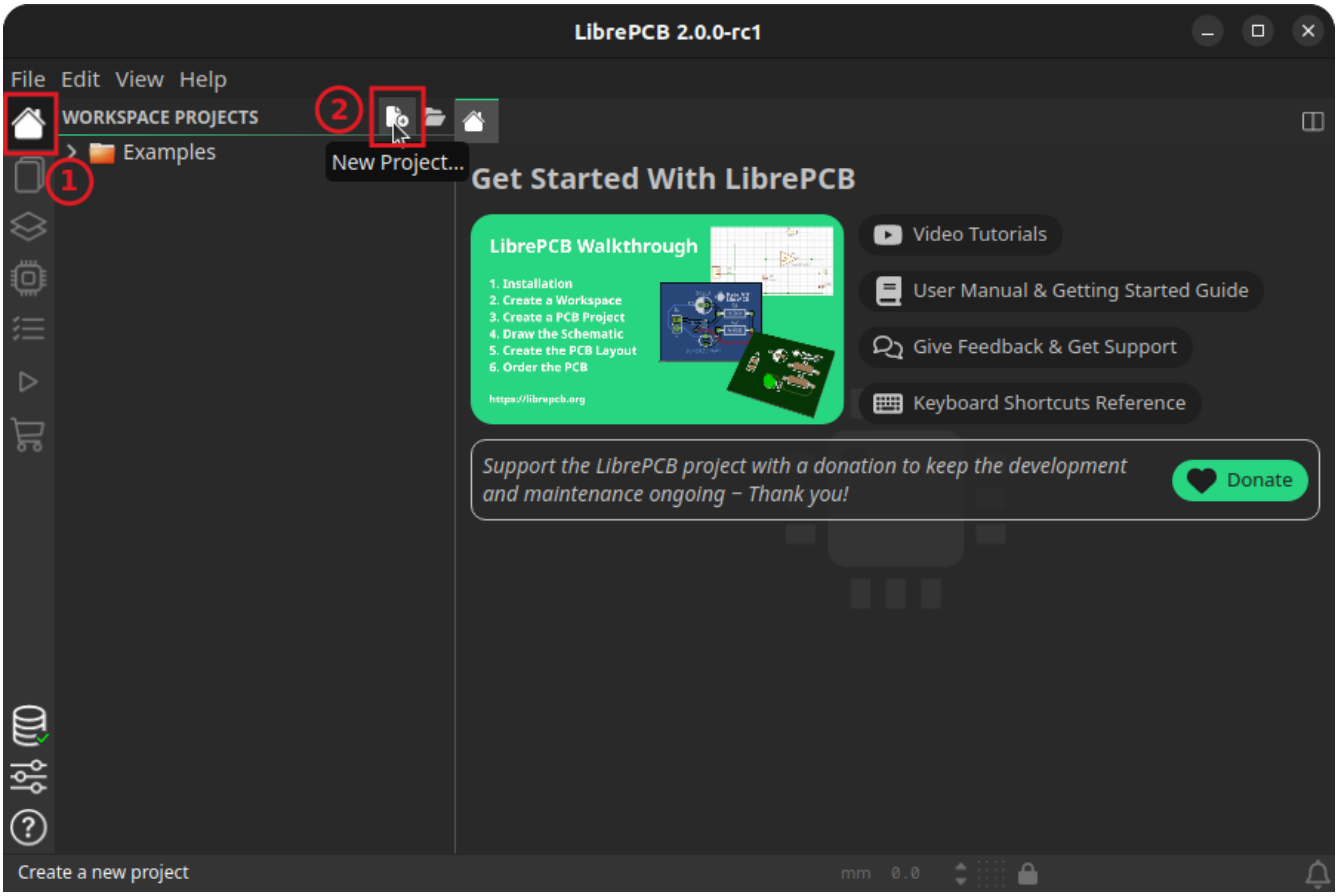
That's it, your library is now created and is ready to get some library elements added to it, as explained in a later chapter. At the moment you will just see an empty library. At the moment we don't need the library editor, anymore, so you can close it with the **[X]** button in the *Open Documents* panel:



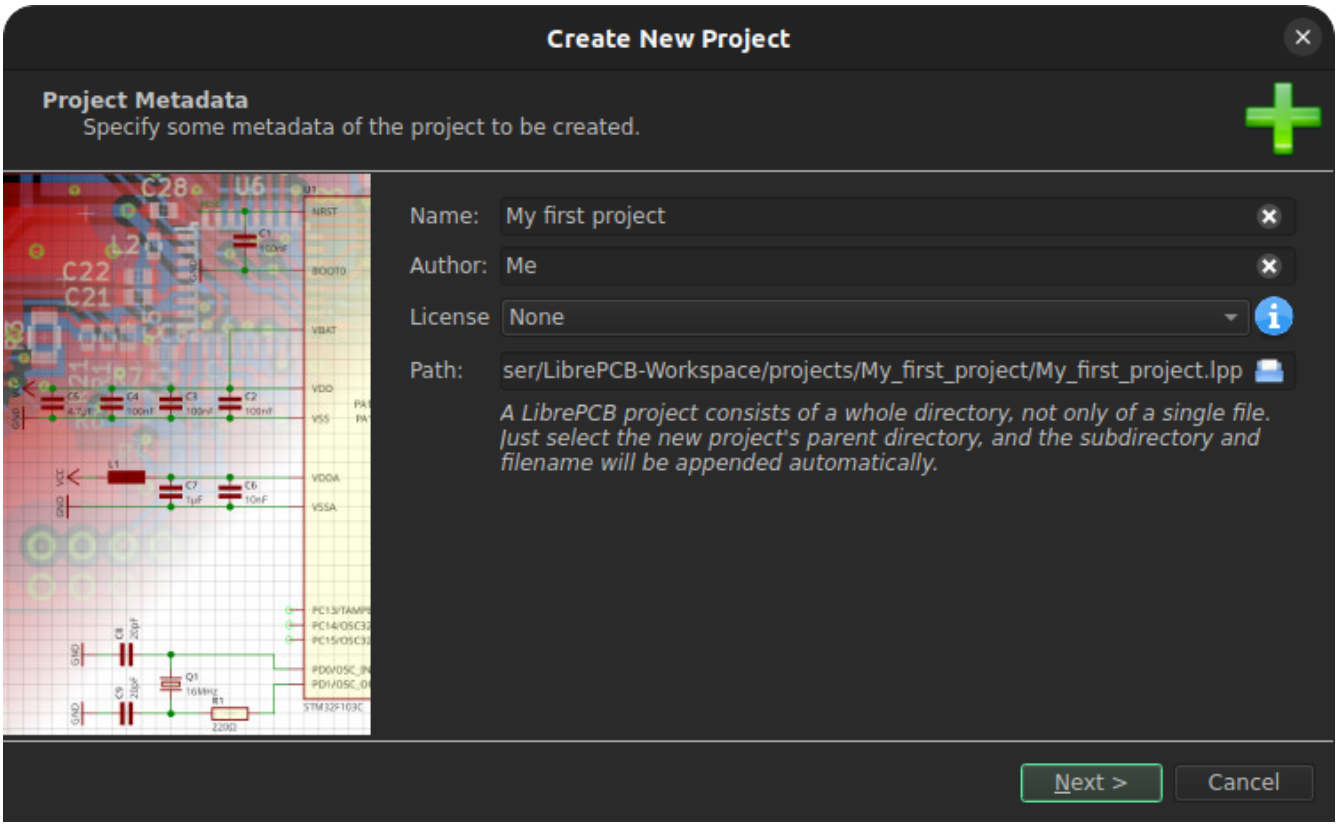
Your workspace setup is now complete and ready to start creating your first PCB project! We'll come back to the library editor later when we need to create our own library elements.

Create a PCB 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. Click on **[New Project]** in the *Home* panel:



Then specify some project metadata:

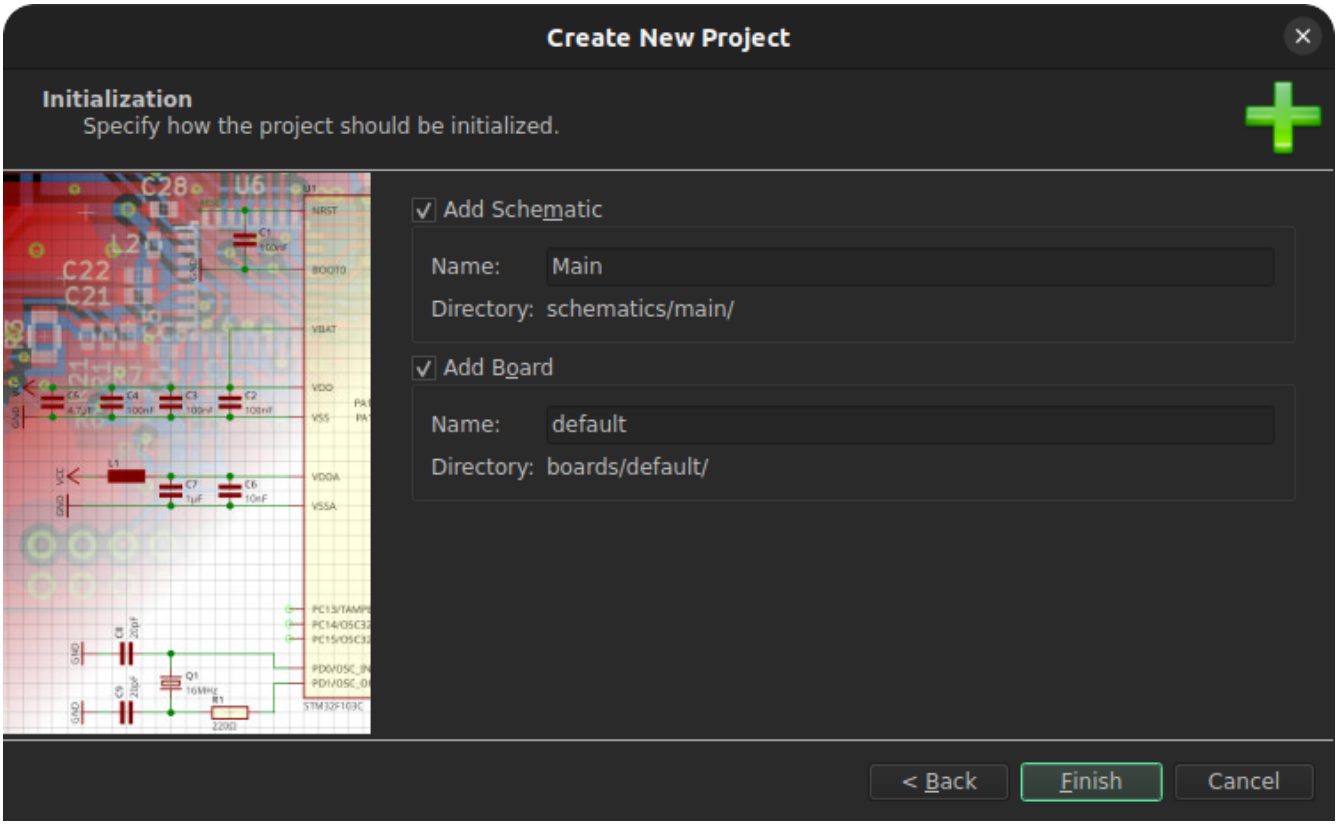


It's recommended to store projects within the workspace subdirectory named `projects` (the default location suggested by the wizard) because these projects are then shown in the control panel file explorer, making them easy to locate and use.

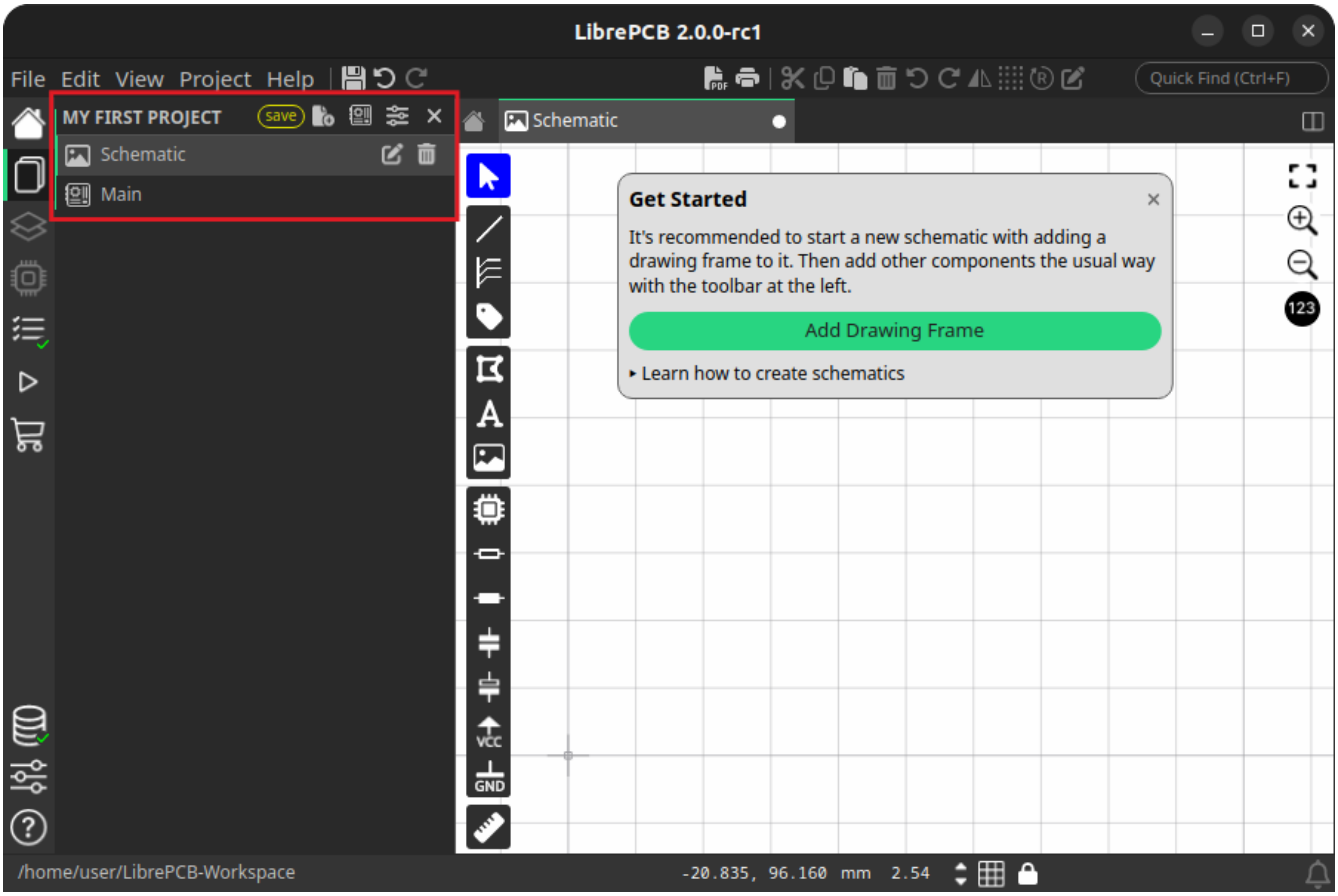
But of course projects can be created at any other location as well.

A LibrePCB project consists of a whole directory on the file system. While it is possible to manually add/modify files in that directory, generally you should avoid adding large files (e.g. datasheets) since this *could* slow down some operations. It's better to store unrelated files outside of the project directory.

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 project is opened in the *Open Documents* panel and the schematic editor shows up:

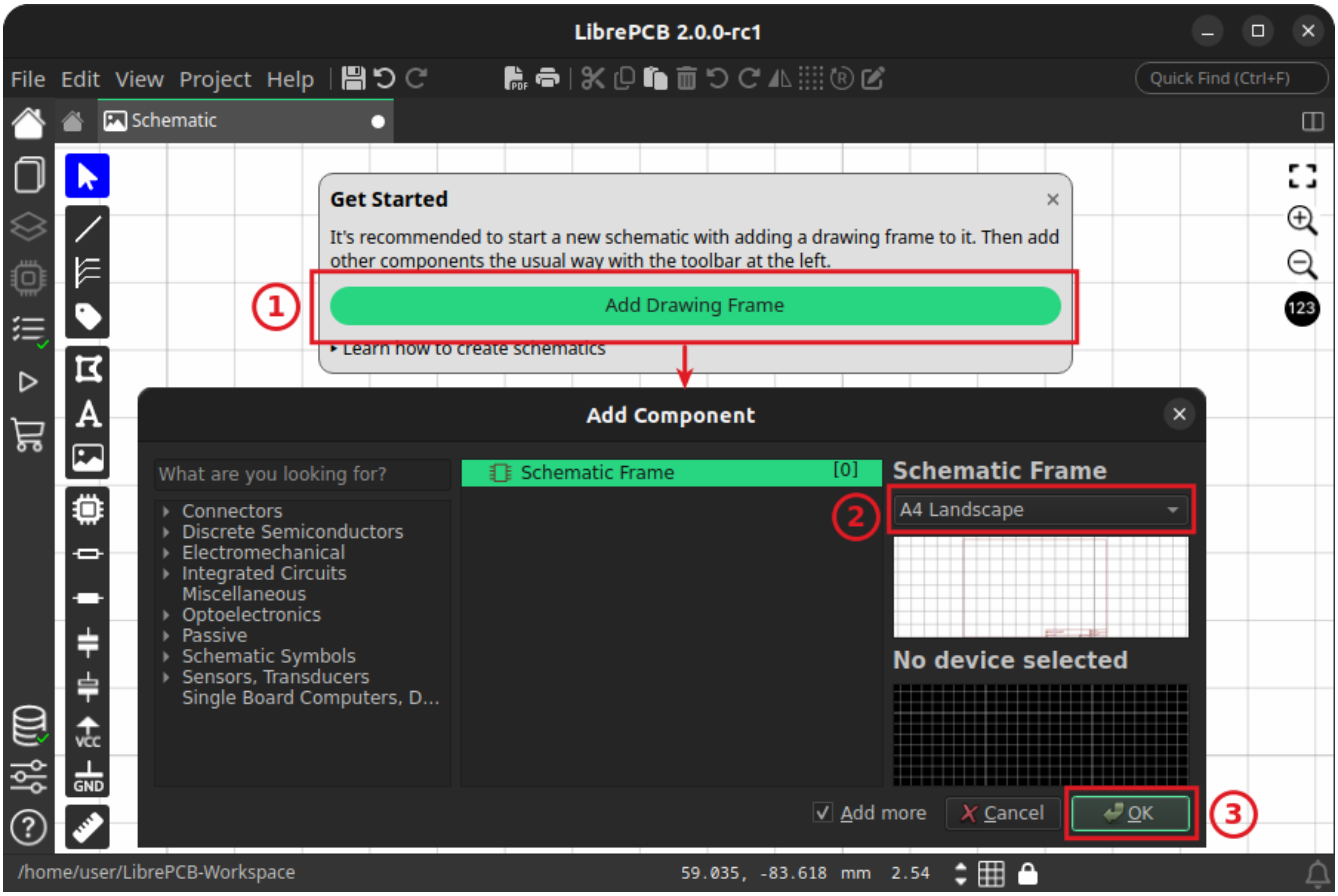


Create Schematics

Before starting with the board layout, a schematic will be needed. At the moment, it is not possible to create boards without creating a schematic first (we may change that in future, though we recommend anyway to always create a schematic first). So let's see how to draw a schematic.

Add Drawing Frame

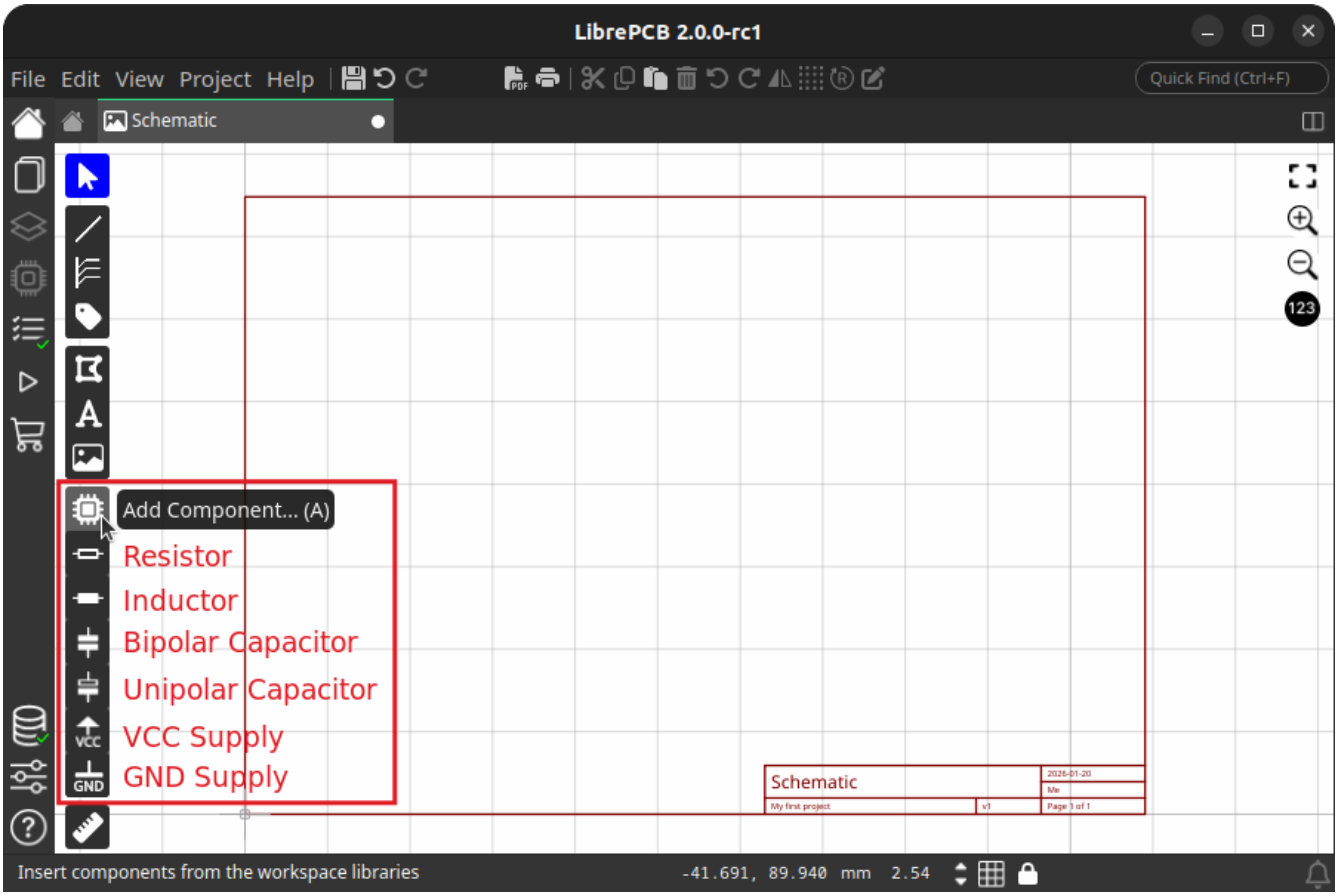
First, you may want to add a drawing frame to the schematic. Click on the **[Add Drawing Frame]** button which is already suggested, or alternatively use the **[Add Component]** button from the toolbar. Then choose the desired paper format and click **[OK]** to add the frame to the schematic:



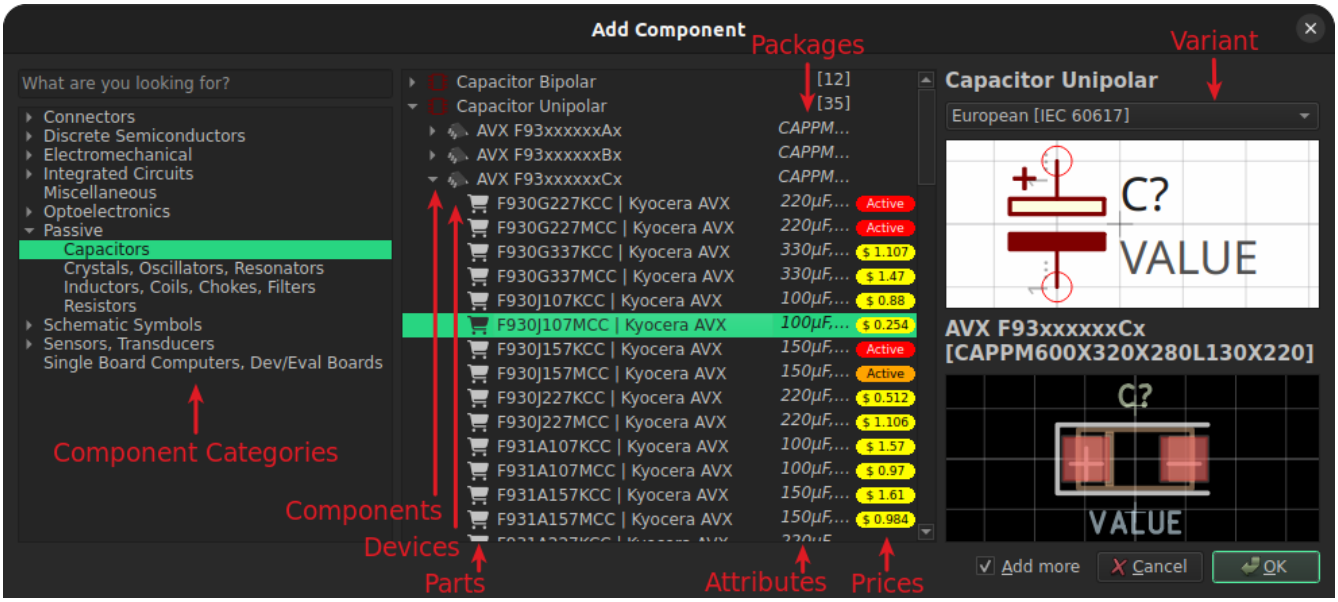
Note that we now collapsed the left side panel to have more space for the drawing area.

Add Components/Devices

Now add all the resistors, capacitors, ICs etc. with the corresponding tool buttons to your schematic:



However, for real parts (in contrast to the schematic frame), the *Add Component* dialog lets you select a concrete device. Here an explanation about the displayed information:



You can choose between adding a **component** a **device** or a **part**:

- **Component**: Defines the schematic symbol and netlist signals. It's all you need in a schematic, but it does not represent a concrete part and does not specify the package to be placed on the board.
- **Device**: Defines the package to be used in the board. Basically it's the combination of a component and a package with a particular pinout.
- **Part**: Represents a real, orderable part. In addition to defining the package, it also defines the

exact MPN[1] which will appear in the BOM[2].

To add something to a board, you need to choose a **device** or a **part**. However, it's your choice whether to select it *now* or *later* when starting with the board layout. This allows to draw the complete schematics even if various packages and devices do not exist yet in your libraries.

After clicking on **[OK]**, the selected component is attached to the cursor. Click on a point in your schematics to place the component. Press `Esc` to finish the placement. The *Add Component* dialog pops up again to choose the next component. Press `Esc` again to leave the tool.

While placing components, press `R` to rotate or `M` to mirror. With `Tab` the focus is moved into the toolbar to allow specifying a value.

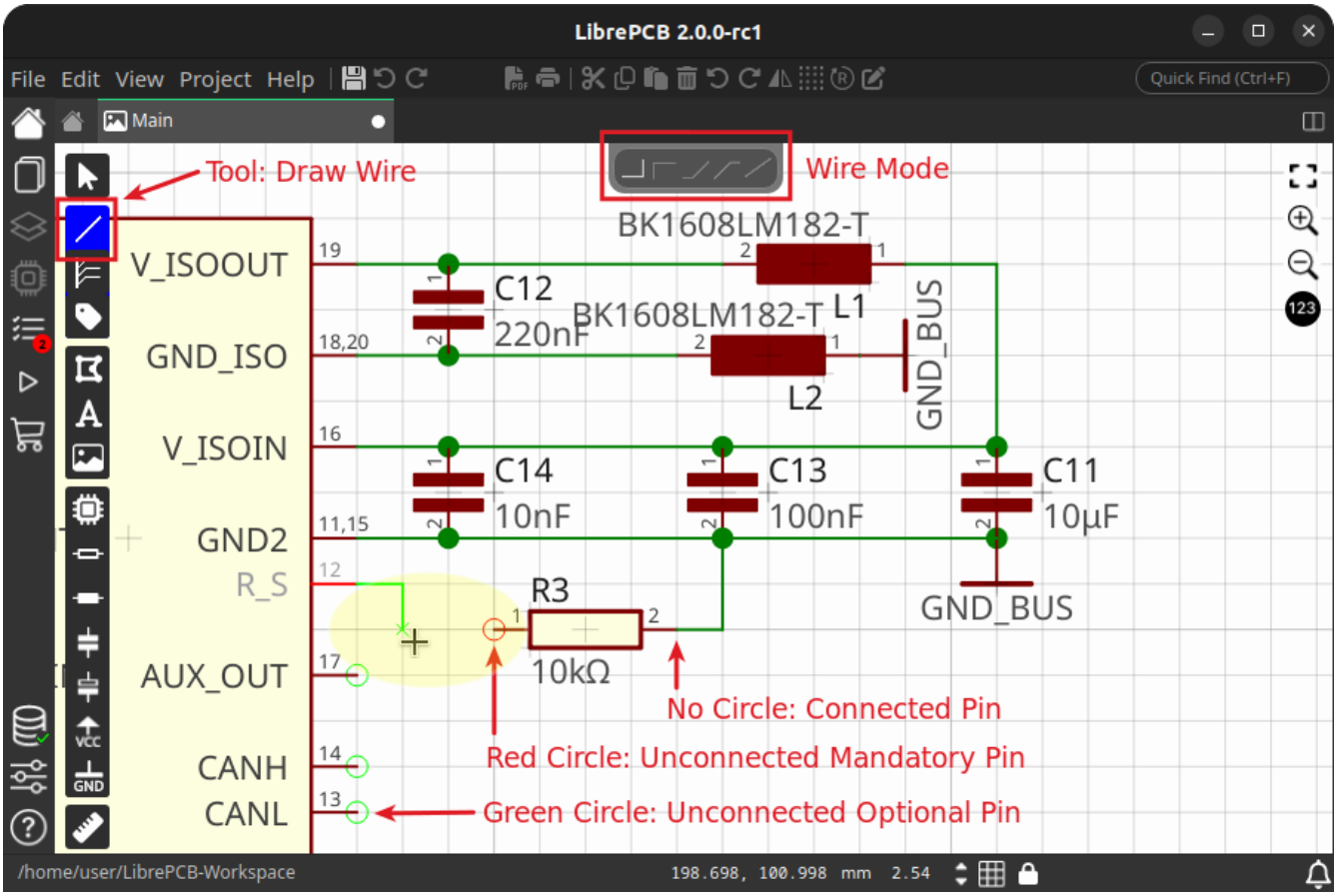
Supply symbols like VCC or GND are added exactly the same way since these are ordinary library elements as well. However, they are also provided in a dedicated toolbar for a quick access to the most commonly used elements.

The *Add Component* dialog lists all the components, devices and parts available in the libraries you have installed in your workspace. **If you are missing something, you either need to install more libraries or create your own library elements.**

To create your own library elements, follow the linked tutorial. You can keep the project open while working in the library editor. Afterwards, **wait for the background library scan to complete** (indicated as a progress bar at the bottom right of the window). Then the new library elements will appear in the *Add Component* dialog and are ready to be used.

Draw Wires

Once your schematic contains some components, the pins can be connected with the **[Draw Wire]** tool. In this tool, just click on a pin to start a new wire:



Pay attention to the circles around the pins. If a wire appears to be starting at a pin, but the circle is visible, it is **not** connected.

The color of the pin circles even provide some more context:

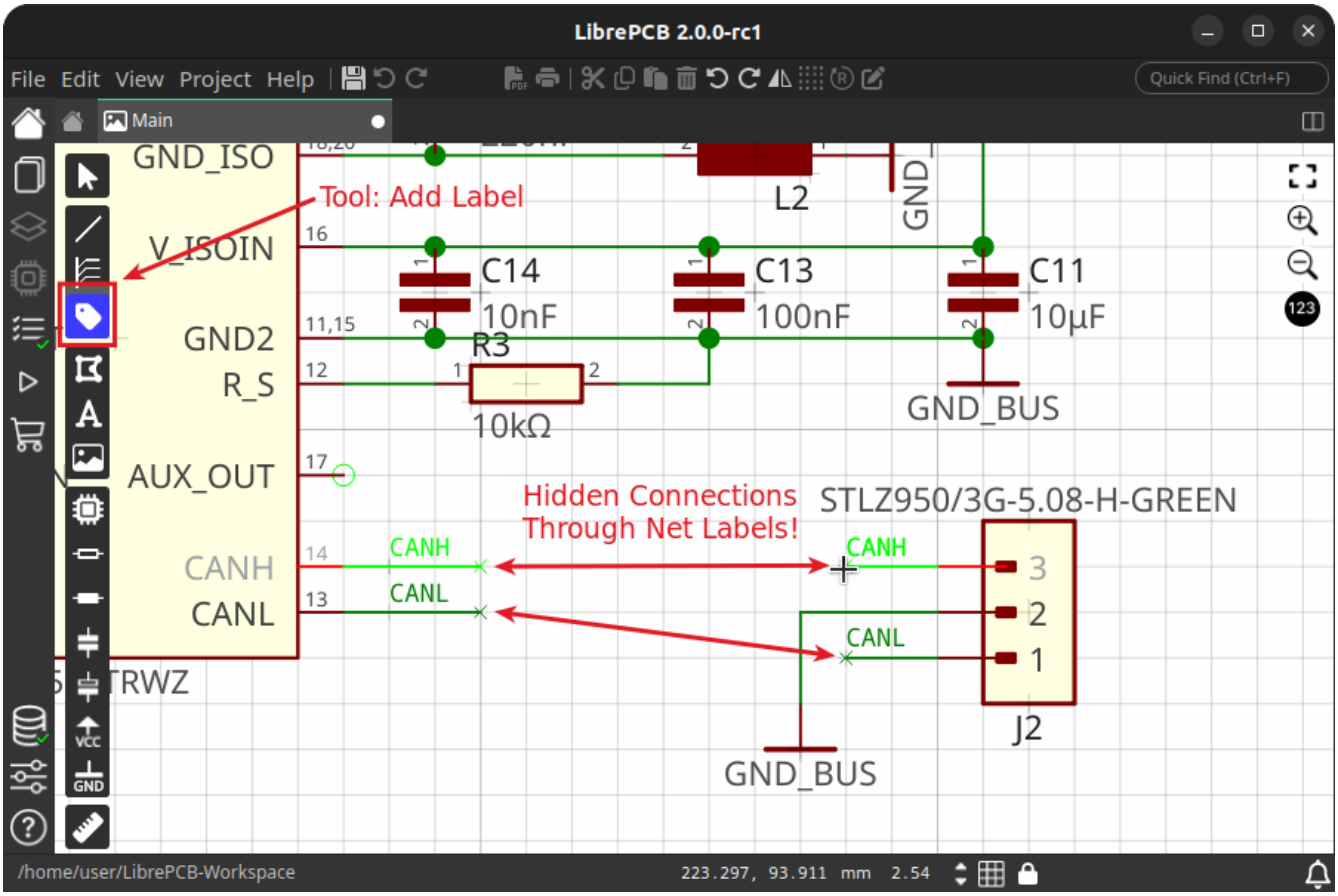
- **Red**: Mandatory pin, i.e. needs to be connected to a wire (if not, an ERC[3] warning is raised).
- **Green**: Optional pin, i.e. may or may not be connected, depending on the use-case. No ERC error will be raised if left unconnected.

Add Net Labels

To keep schematics clean and readable, net labels may be added. They allow to explicitly specify net names, and to create hidden connections between wires of the same net name.

1. Start the tool **[Add Label to Net or Bus]**.
2. click on the wire where to attach the label.
3. Click to specify the label position.

While placing labels, press `R` to rotate.



All wires in the whole project which have the same name assigned will automatically be connected, even accross schematic pages.

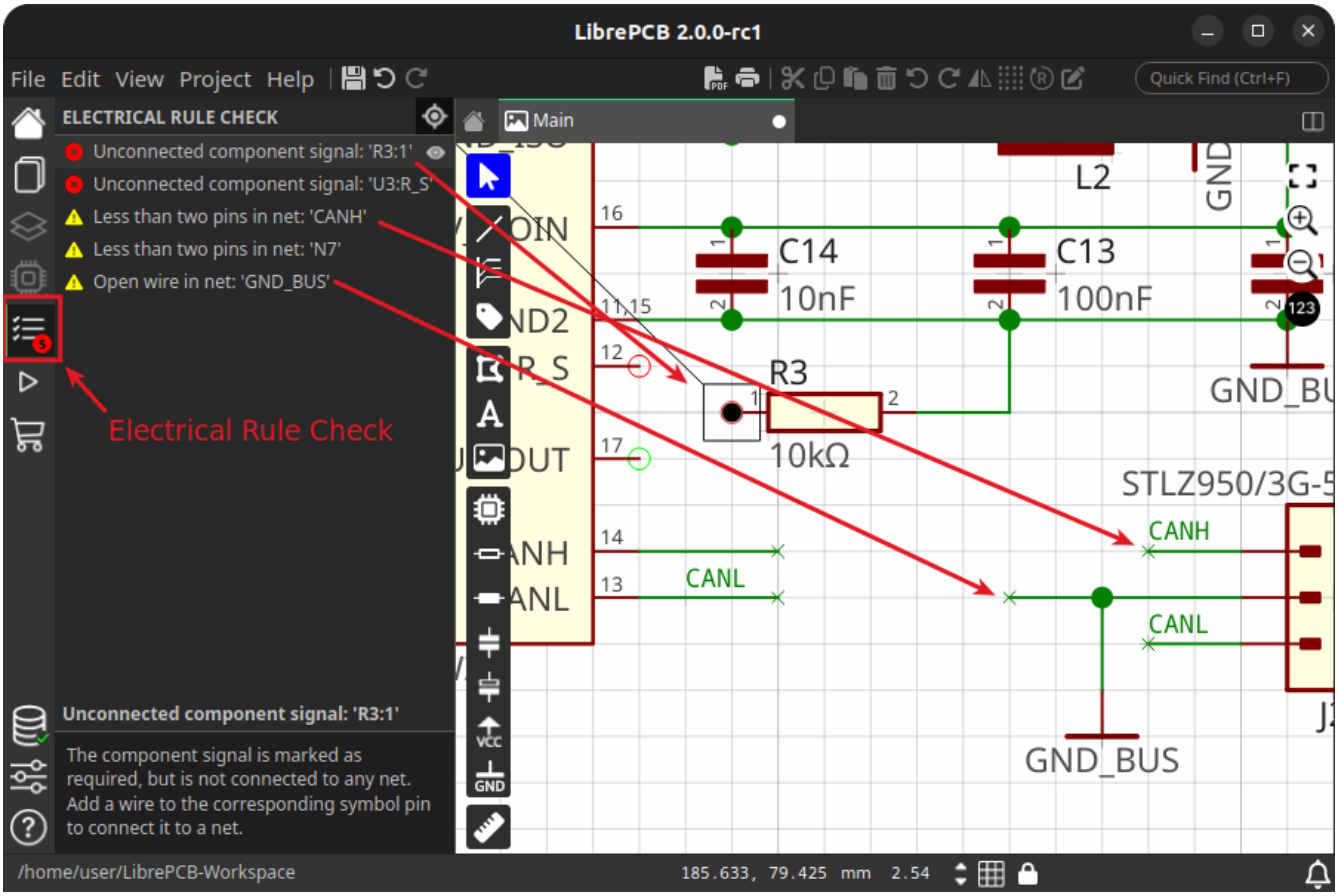
Add More Sheets

For larger projects, you may want to split the schematics into multiple sheets for better readability. Just add more sheets with **Project › New Sheet**, then add a frame and devices the same way. Use supply symbols and net labels to connect nets across pages.

Electrical Rule Check

At latest when you're finished with the schematics, you should check if there are no critical ERC messages. The ERC does not need to be triggered since it is automatically updated.

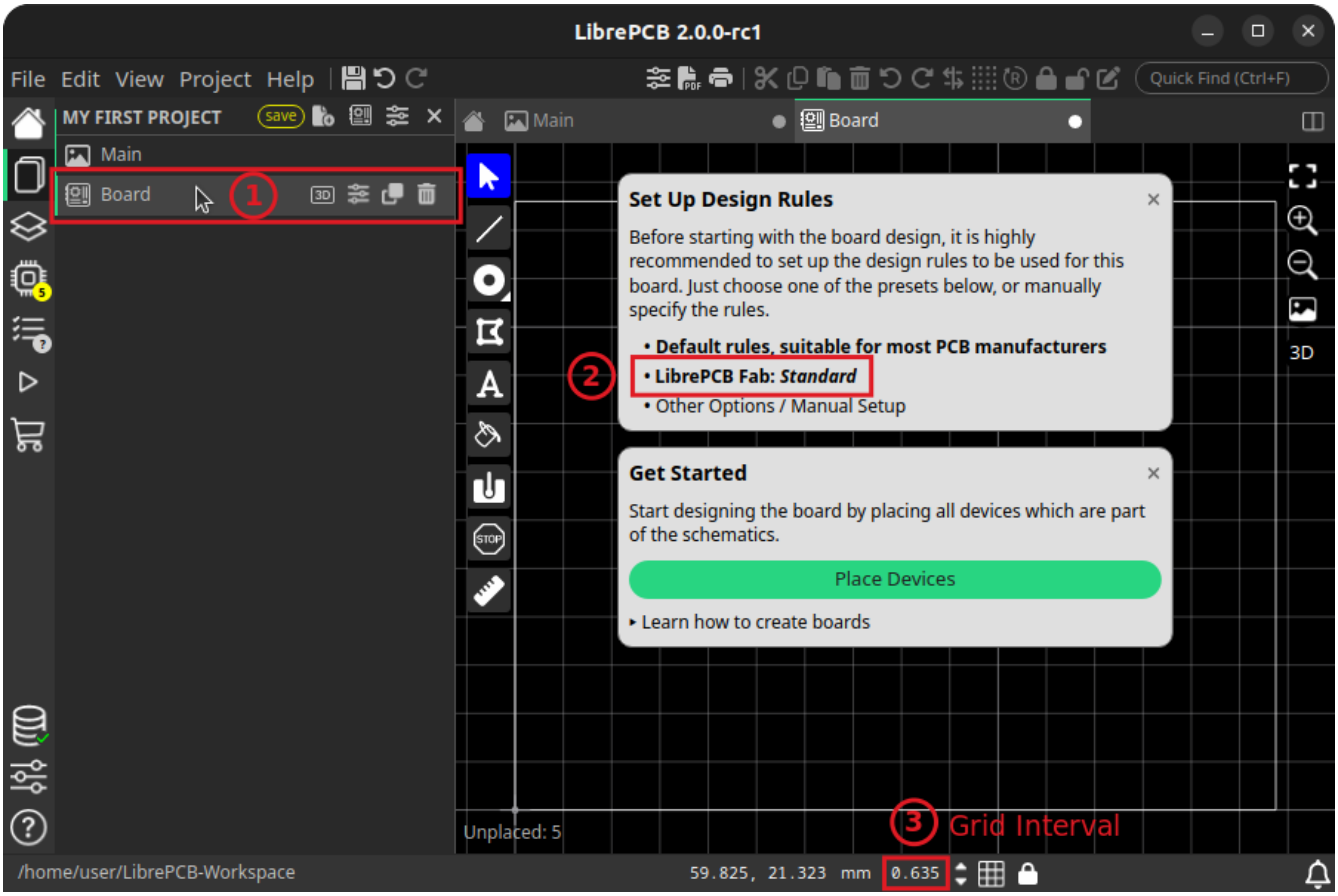
Open the ERC panel with the corresponding sidebar button while a schematic is opened:



Click on a message to get more information about it and to see its location (if available) in the schematics. If you're sure a message is not relevant, you could approve it with the eye symbol next to them, but usually warnings/errors should be fixed instead of approved.

Create Board

Once the schematic is (more or less) complete, you can start designing the PCB in the board editor. For this, open the board in the *Open Documents* panel and set up some initial settings as described below:



Setup Design Rules

Different PCB manufacturers have different capabilities about what structures they are able to produce (e.g. minimum copper trace width). LibrePCB can verify if your board conforms to the manufacturers capabilities, but for this you need to set up the so-called *Design Rules*. If you intend to order your PCB at LibrePCB Fab, you can choose the corresponding rules (see screenshot above). If you are unsure, just go with the default rules.

Set Grid Properties

All board editor tools (e.g. the *Draw Trace* tool) work on a particular grid interval, i.e. the cursor snaps on a multiple of that value. The value might depend on the task you're working on, so probably you'll need to change it several times while working on the board.

You can change it at any time in the status bar, see screenshot above (press `F4` to edit, or hover with the mouse and scroll up/down).

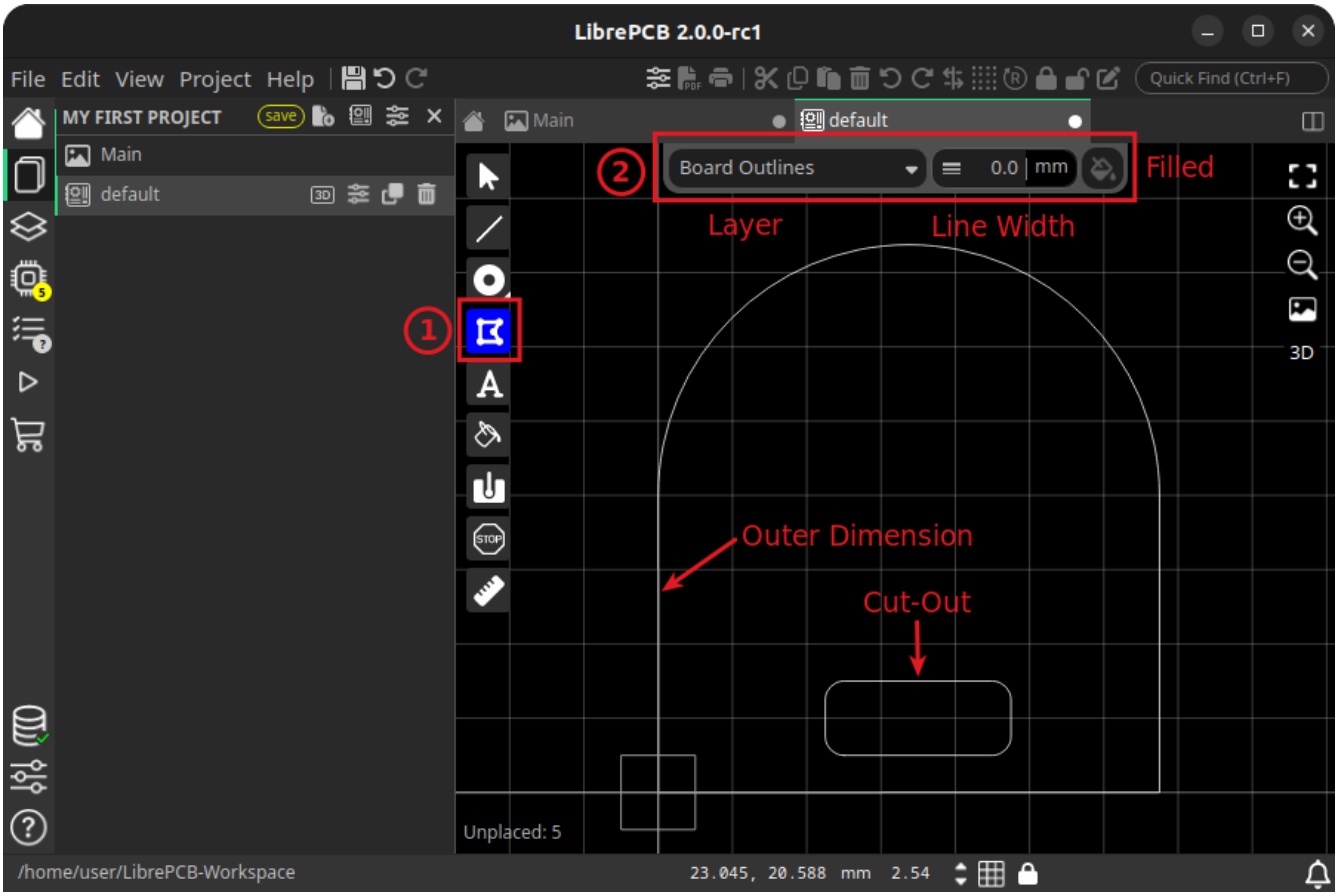
Draw Outlines

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, ...), draw these polygons on the *Board Cutouts* layer with a width of 0.0mm.

A simple board outline polygon is automatically added by LibrePCB when creating a new project or board! So usually the only thing you need to do is to **resize it to the desired size**. The instructions here are intended only to explain more complicated scenarios and in case you want to re-draw the outline from scratch.

All polygons on the *Board Outlines* and *Board Cutouts* layers 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 (right-click) 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 on the *Board Outlines* layer.

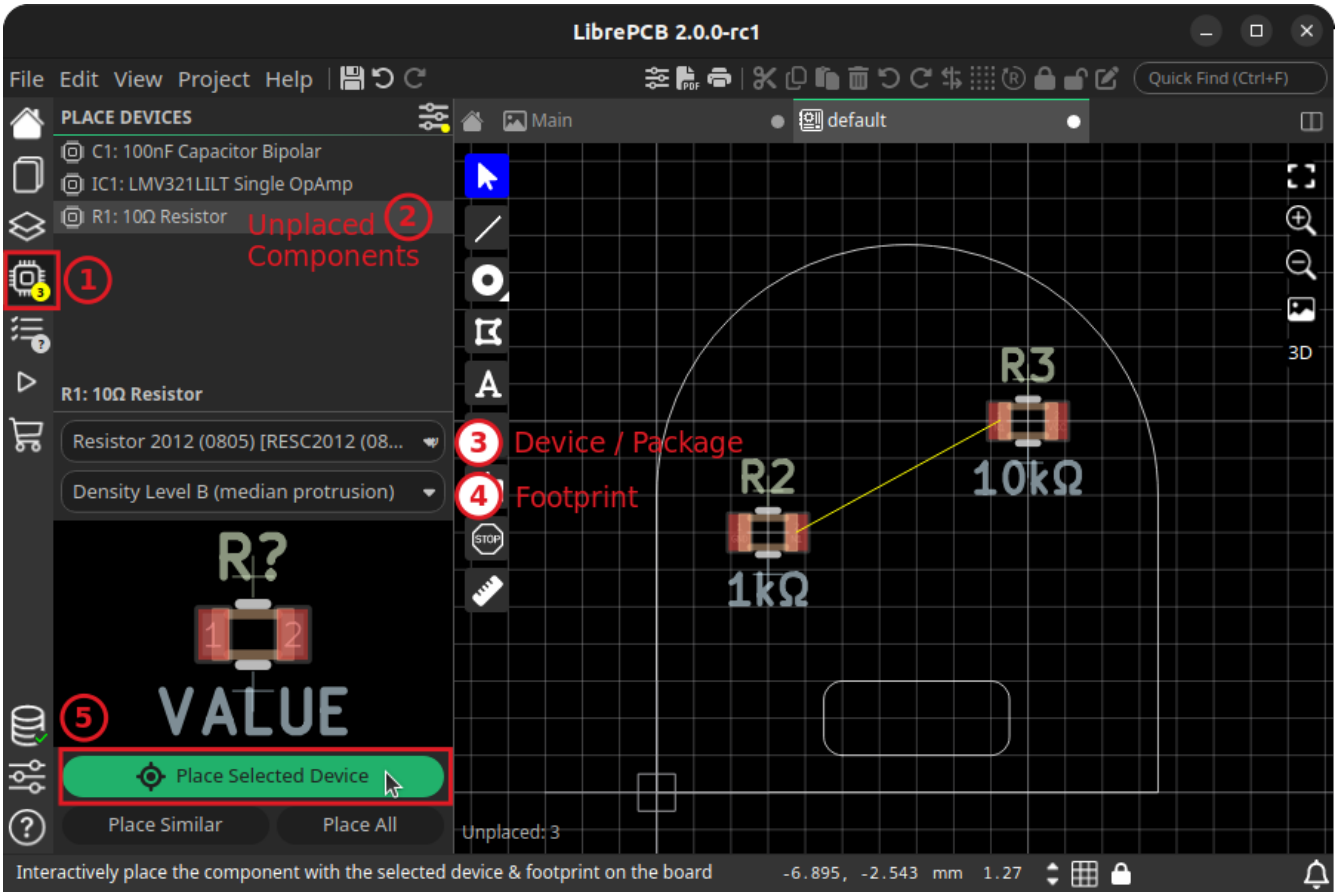
- Cut-out polygons (if there are any) are on the *Board Cutouts* layer and located fully inside the outer board outline.
- There are no tangent or intersecting polygons on these two layers.
- The line width of those polygons is 0.0mm (optional, but recommended).
- Polygons are closed (start and end coordinates are exactly identical) and consisting of a single polygon object (**NOT** multiple joined lines!).
- There are no other objects on these two layers.

An easy way to check if the board outline is valid is to review the PCB in the 3D viewer. For that, open **View › Toggle 2D/3D Mode** or press `Ctrl` + `3`.

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).

1. Open the *Place Devices* panel in the sidebar (`Ctrl` + `Alt` + `P`).
2. Select a component to place.
3. Select the desired device for that component (not needed if the device is already specified in the schematics).
4. Choose the exact footprint to place, if there are multiple. Most packages have only one footprint — if not, the default footprint is pre-selected.
5. Click **[Place Selected Device]** and place the device with the cursor on the board. Press `R` to rotate or `F` to flip to the other board side while moving.

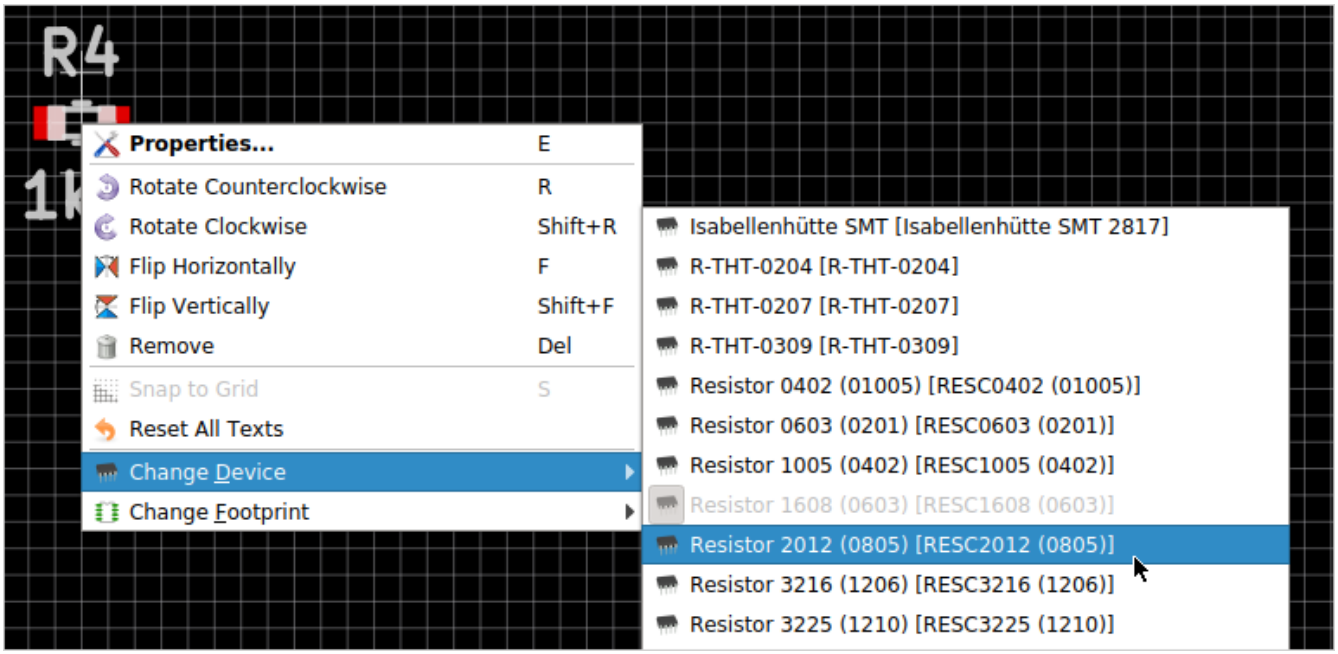


Repeat these steps until there are no more unplaced components.

If you want to use the same device and footprint for all instances of a particular component, use the **[Place Similar]** button to add all at once.

If you can't find the desired device for a component (or the device dropdown is completely empty), you need to add the device to your local library first. Continue with the library element creation tutorial and come back to the board editor once the device is created.

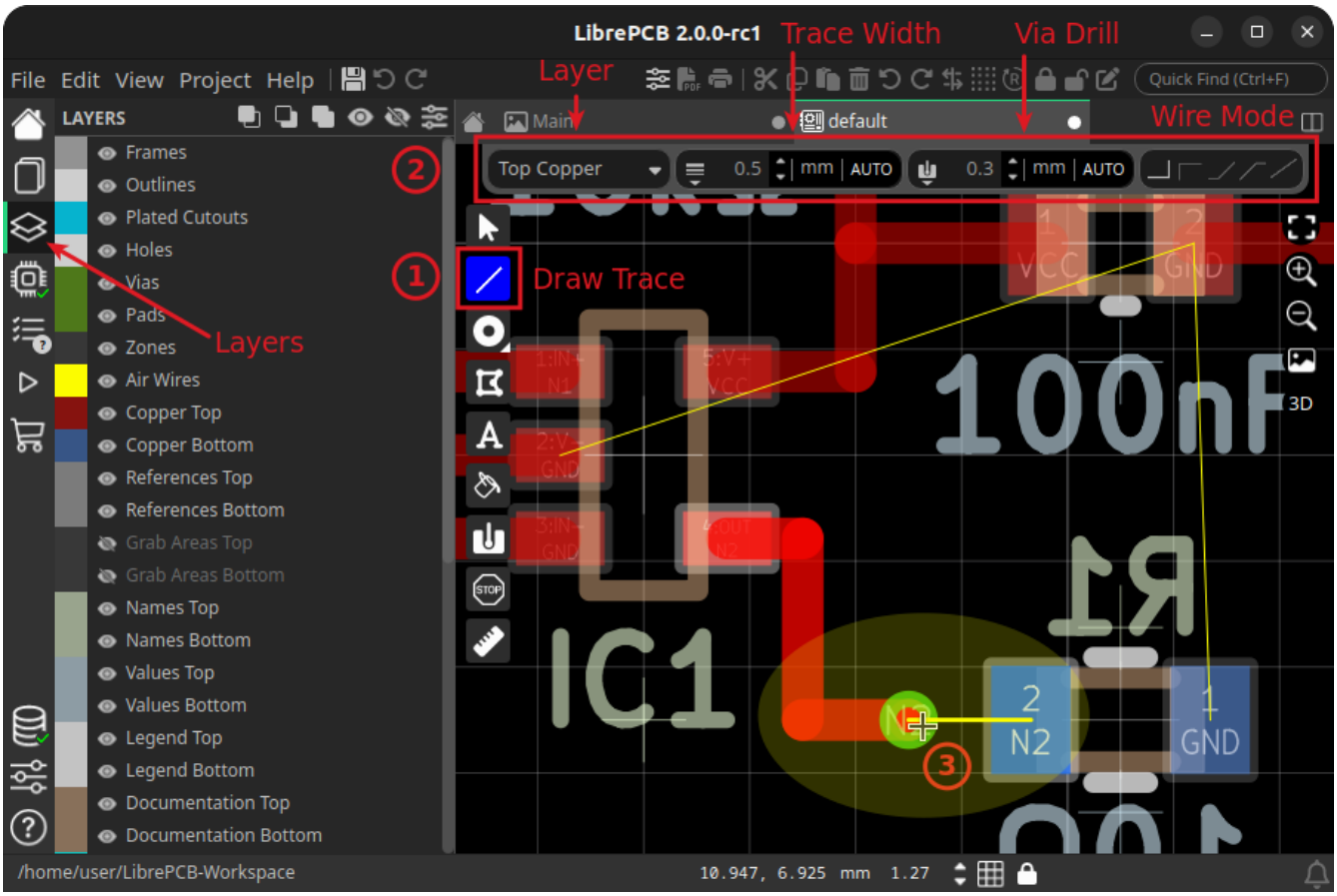
By the way, it's even possible to replace devices after adding them to the board. For example you can replace a 0603 resistor by a 0805 resistor using the **[Change Device]** context menu item (right-click):



Exactly the same way you can switch to a different footprint, just use the **[Change Footprint]** context menu item instead.

Draw Traces

As soon as you add devices to the board, airwires will appear to show the missing traces. Start the **[Draw Trace]** tool and specify the trace settings in the toolbar. Then click on a pad to start a new trace:



In the *Layers* panel (in the left sidebar) you may show or hide individual layers to keep the working

area clear of irrelevant objects during trace drawing.

The cursor automatically snaps on objects of the same net. If this is not desired, hold `Shift` while drawing.

With the right mouse button you can cycle through the different routing modes.

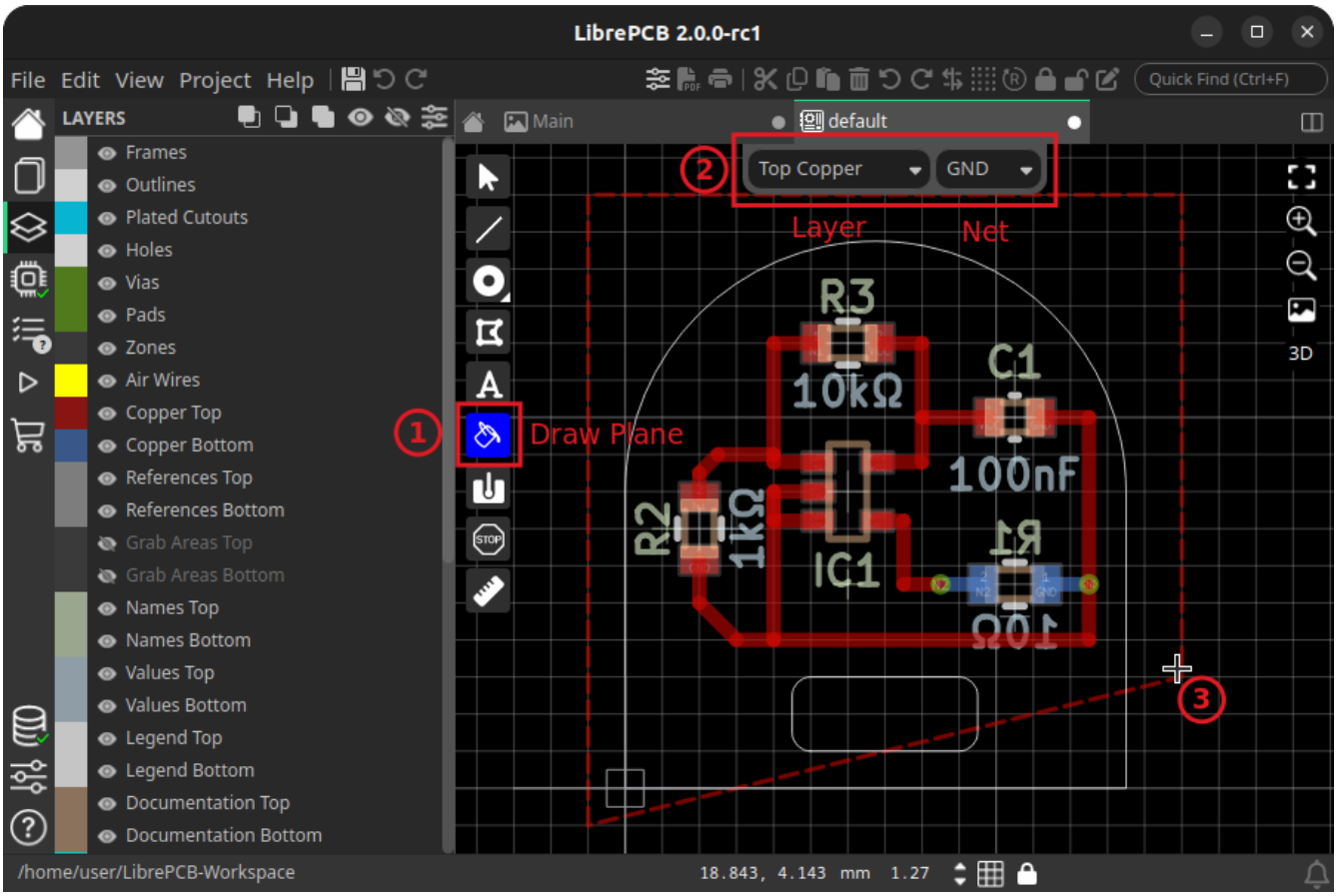
To switch to a different copper layer while drawing a trace, press `Page Down` (next lower layer) or `Page Up` (next higher layer). This will automatically insert a via if needed.

There are also shortcuts to change trace & via properties, see **Help › Keyboard Shortcuts Reference** for details.

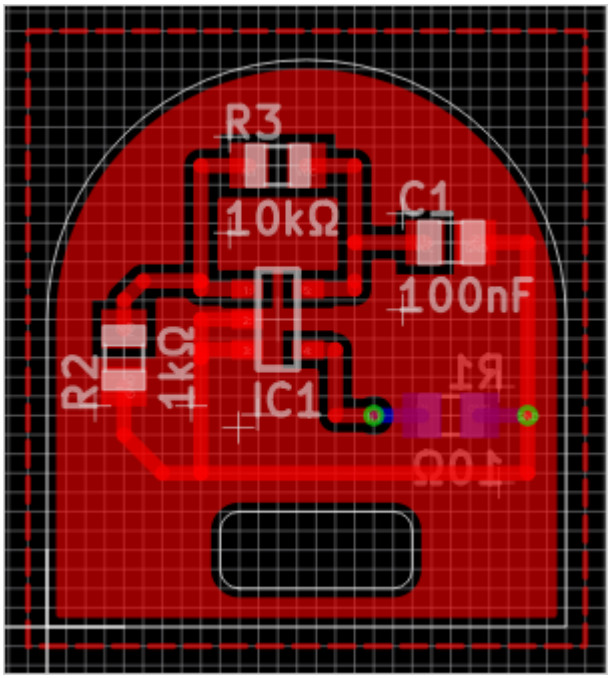
Add Planes (Copper Pours)

If you need planes (also known as *copper pours*, i.e. filled copper areas to create electrical connections), proceed as follows:

1. Start the **[Draw Plane]** tool.
2. Specify the copper layer and the electrical net in the toolbar.
3. Add vertices with mouse clicks. To fill the whole board, an approximate outline is good enough since it will be clipped automatically.



One the plane area is calculated, it appears with a filled area. As you can see, the area is automatically clipped to the board outline:



In case your plane does not get filled, make sure:

- The board outline polygon exists and fulfils all the rules listed above.
- The plane is located *within* the board outlines.
- There is at least one copper element of the same net located within the plane area — e.g. a via, pad or trace. **Plane areas which are not connected to any copper element are automatically discarded** to avoid electrically "floating" copper areas on the board. If you prefer to add these copper areas anyway, open **[Properties]** from the plane context menu (right-click) and check the *Keep Islands* option.

To avoid plane areas cluttering up the view too much, they can be hidden with **View › Hide All Planes**. They will still be there, they are just hidden on the screen.

To interconnect planes on different copper layers, just place vias with the **[Add Via]** tool within the plane areas. Make sure the vias have the same net as the plane. Vias will also prevent plane fragments from disappearing if there's no other copper element within the plane and the *Keep Islands* option is disabled.

Add Non-Plated Holes

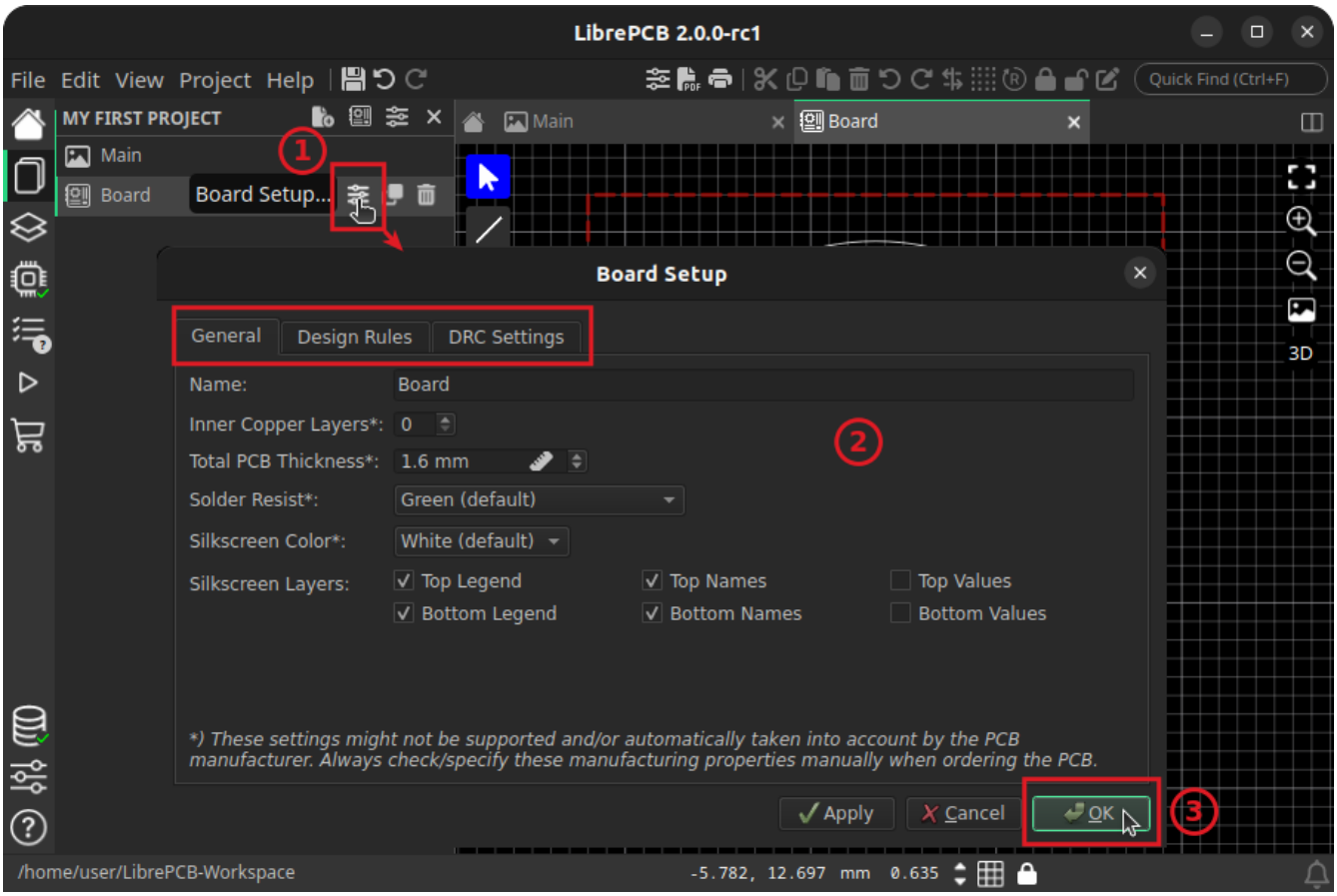
Non-plated holes can be added to the board with the **[Add Hole]** tool. Just specify the diameter and click on the desired position. Afterwards, use the **[Properties]** context menu item to specify the exact position if needed (e.g. if not located on the grid interval).

Design Rule Check

Once your design is complete, you should run the design rule check (DRC) to ensure there are no critical mistakes. The check will report missing connections, short circuits and many more possible issues.

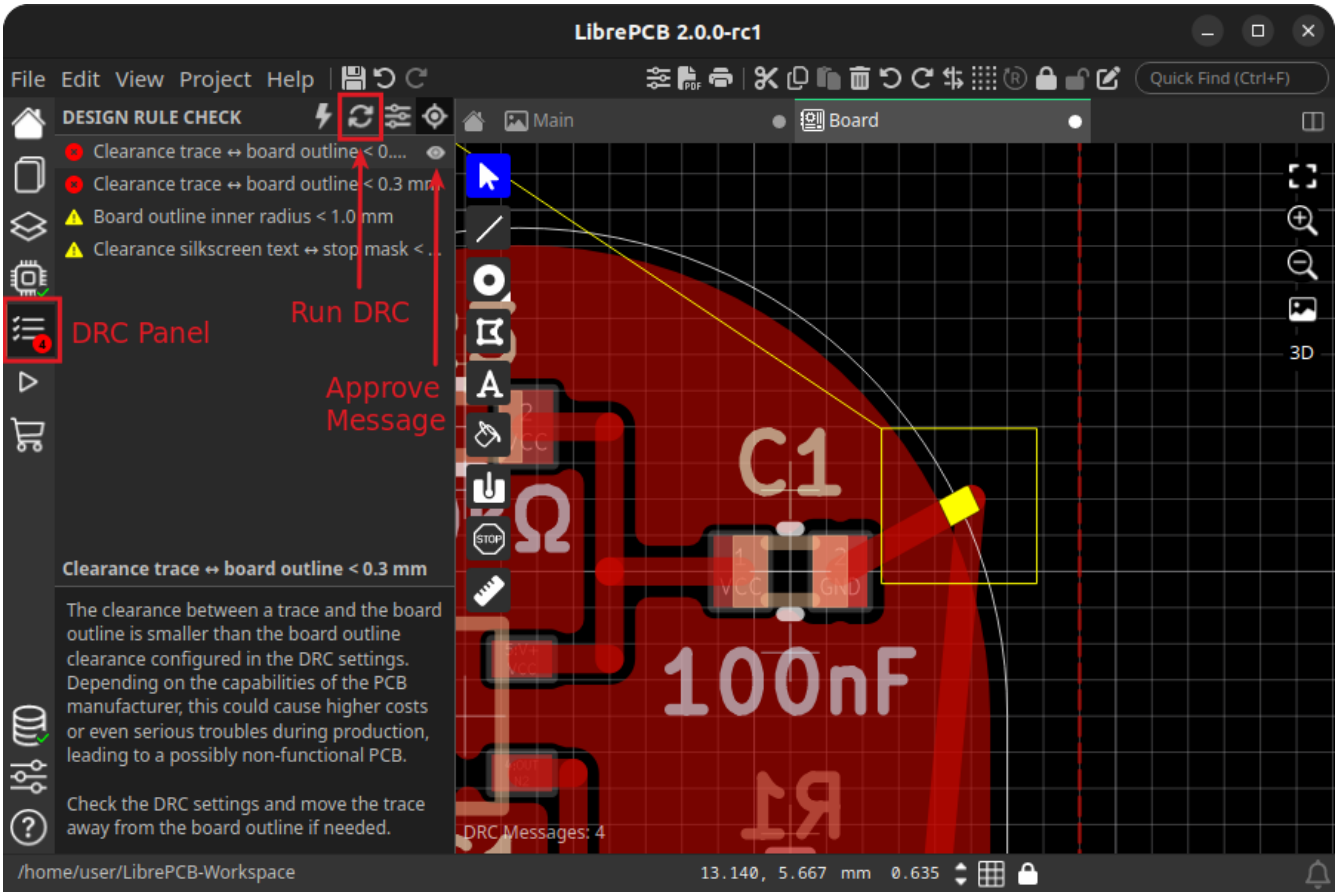
But first you should review/adjust the board setup parameters, including the design rules and DRC settings. The design rules (second tab) are used to calculate for example the via/pad restrings and cream/stop mask clearances, while the DRC settings (third tab) are used only for the verification of the design. This sounds a bit complicated, but no worries, it's often not really important so you can just keep all the default values if you are unsure (you might have initialized some settings anyway already by choosing a PCB manufacturer earlier).

The board setup is available from the *Open Documents* panel (or with `F7`):



Actually it's better to set all these settings *before* drawing traces and adding planes since they affect the clearances. It is only moved to the end of the boards tutorial to keep the focus on the design workflow. Fortunately, usually the default values are fine, so it's not critical to derive from the standard workflow for your first PCB design.

Once all settings are configured, trigger **Project › Run Design Rule Check** or press `F8` to run the DRC. This can take some time. The DRC panel should automatically appear to display the result:

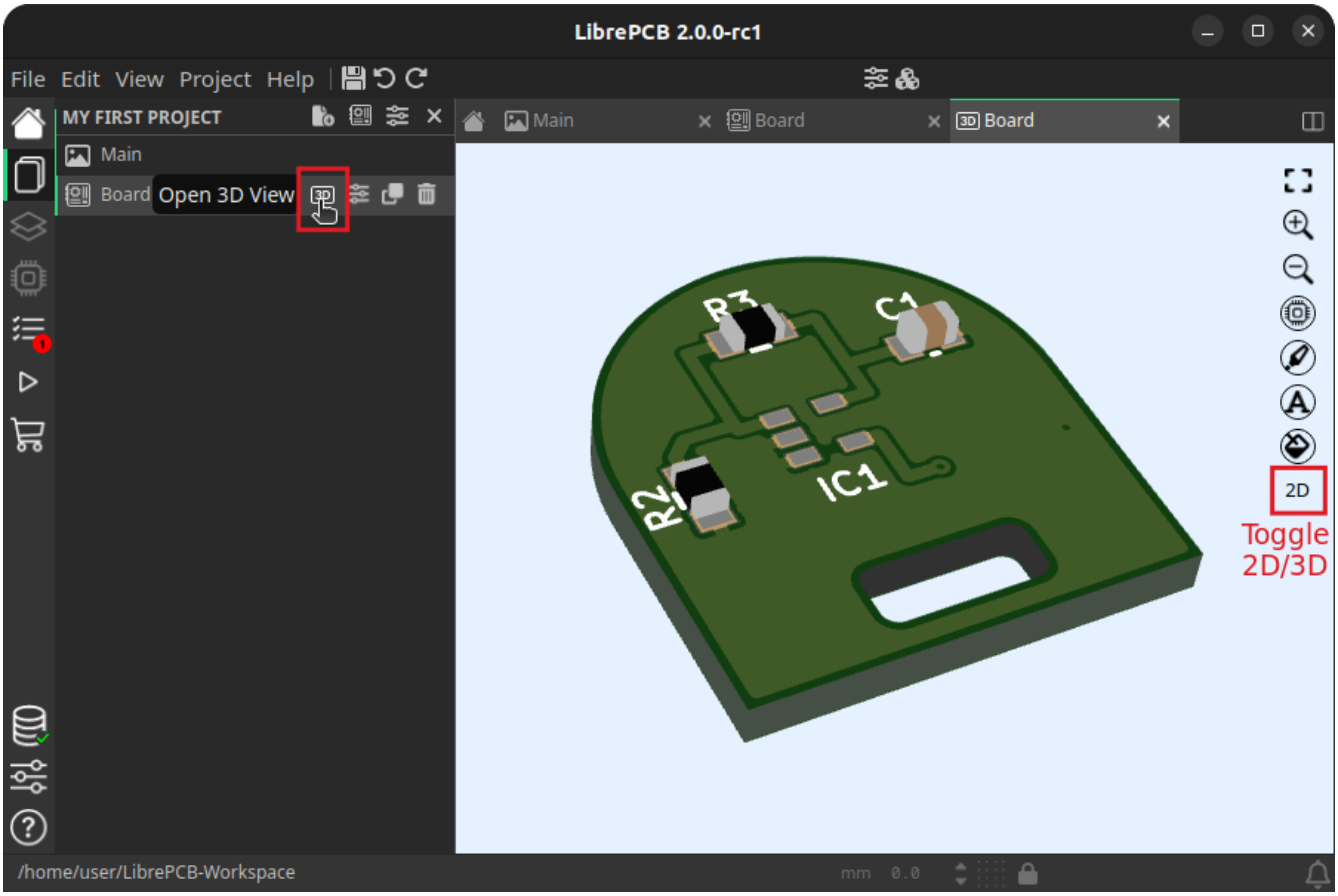


Then just click on a message to highlight the issue in the board editor. If you're sure a message is not relevant, you could approve it with the eye symbol but usually warnings/errors should be fixed instead of approved.

There's also a tool named **Quick Check** which runs only the most important checks of the DRC and is therefore much faster. It is intended to be run regularly while working on the layout and can be triggered with `Shift` + `F8`.

3D Preview

Once you fixed all ERC issues, it's highly recommended to review the PCB in the 3D viewer. If anything with the board outline, the device placement or something like that is not correct, chances are high you will notice that in the 3D view. Click on the board's **[3D]** button or press `Ctrl` + `3` to open it:



Note that not all packages have a 3D model assigned, like the OpAmp in our example. But no worries, this does not cause any issues.

If everything looks as expected, you're ready to order the PCB!

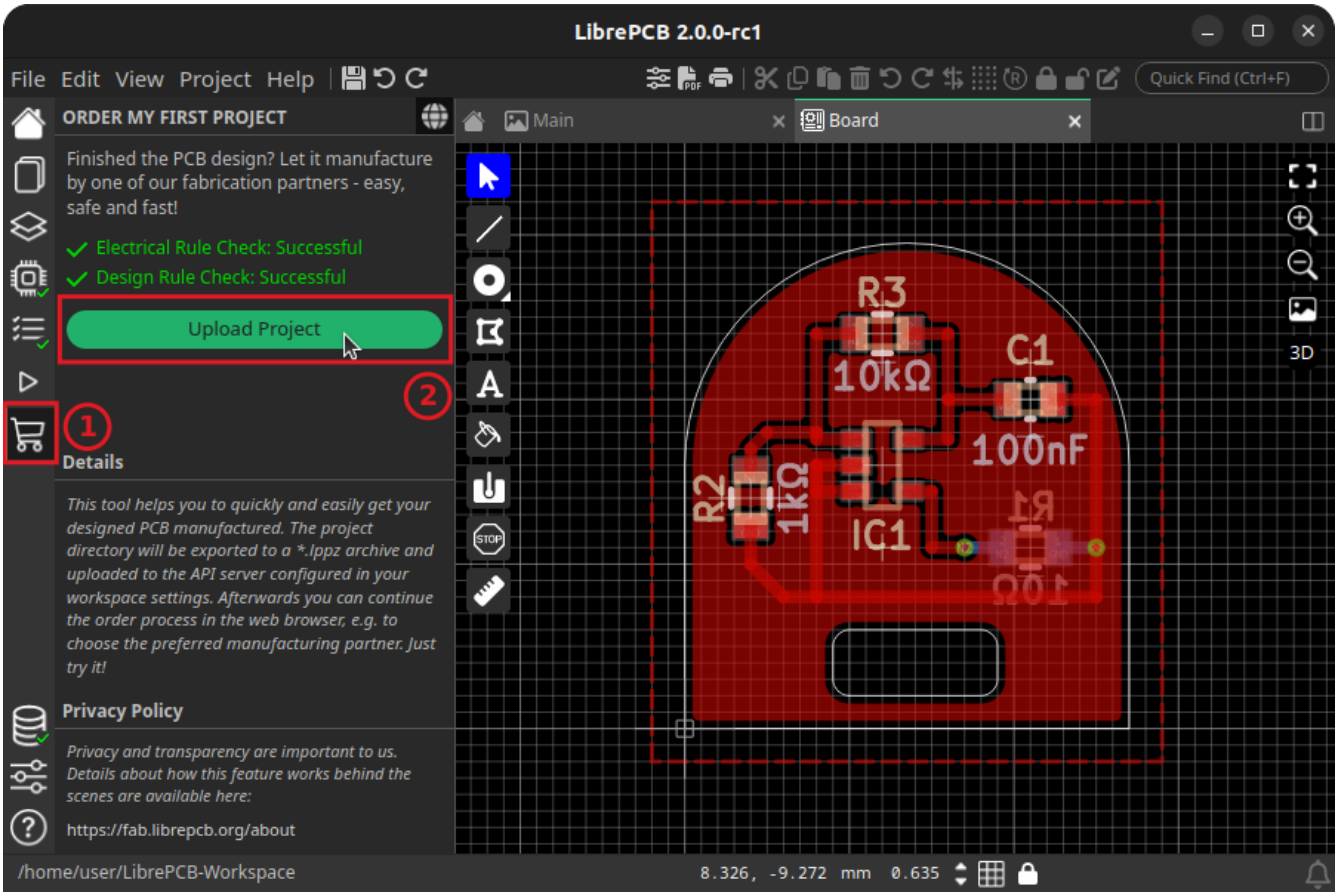
Order PCB

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. See fab.librepcb.org/about for more information.

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.

LibrePCB Fab

To start the order process, switch to the *Order PCB* panel (`F12`):

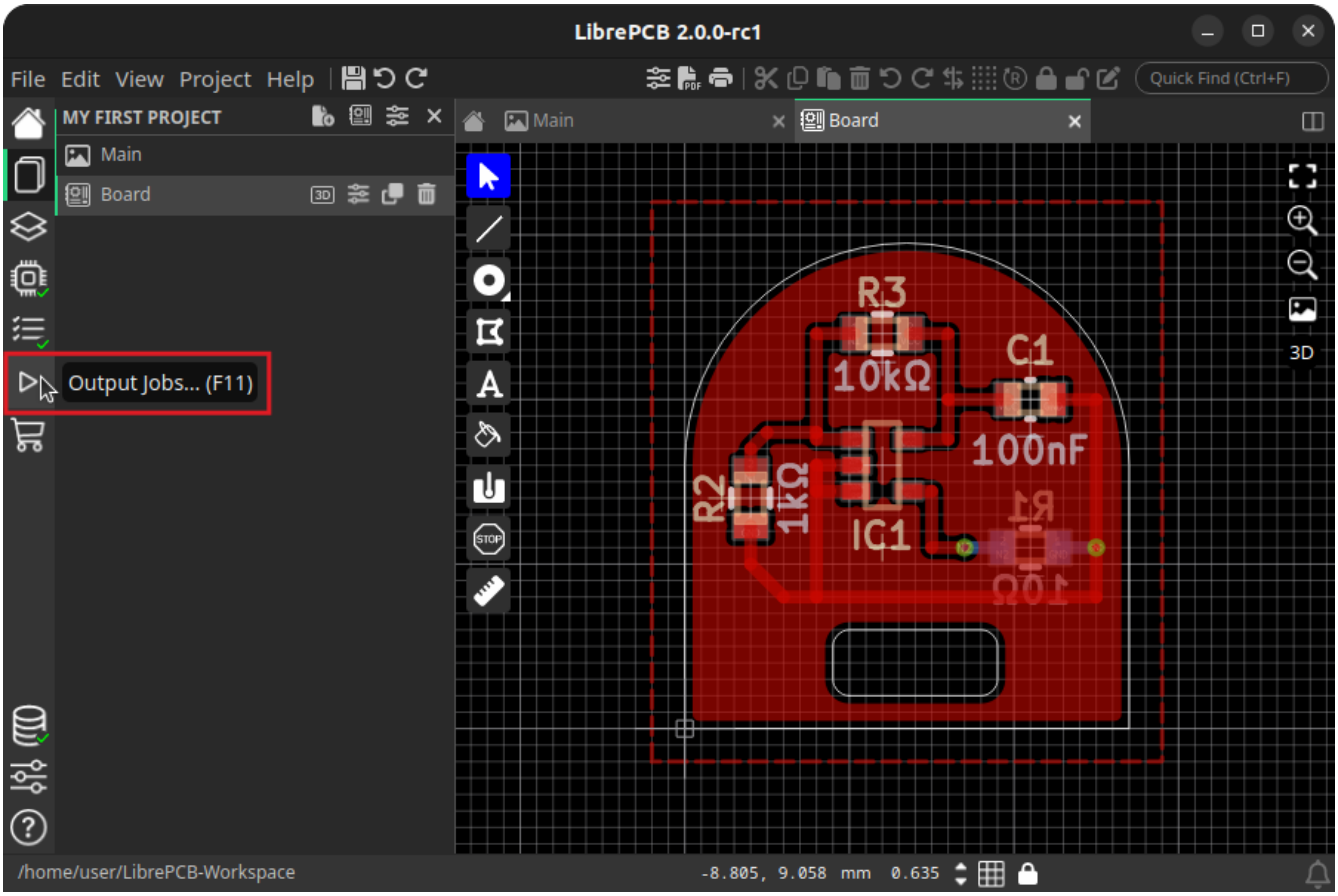


With **[Upload Project]**, the project is uploaded to our order service fab.librepcb.org. Then your web browser should open a website where you can review and continue the order.

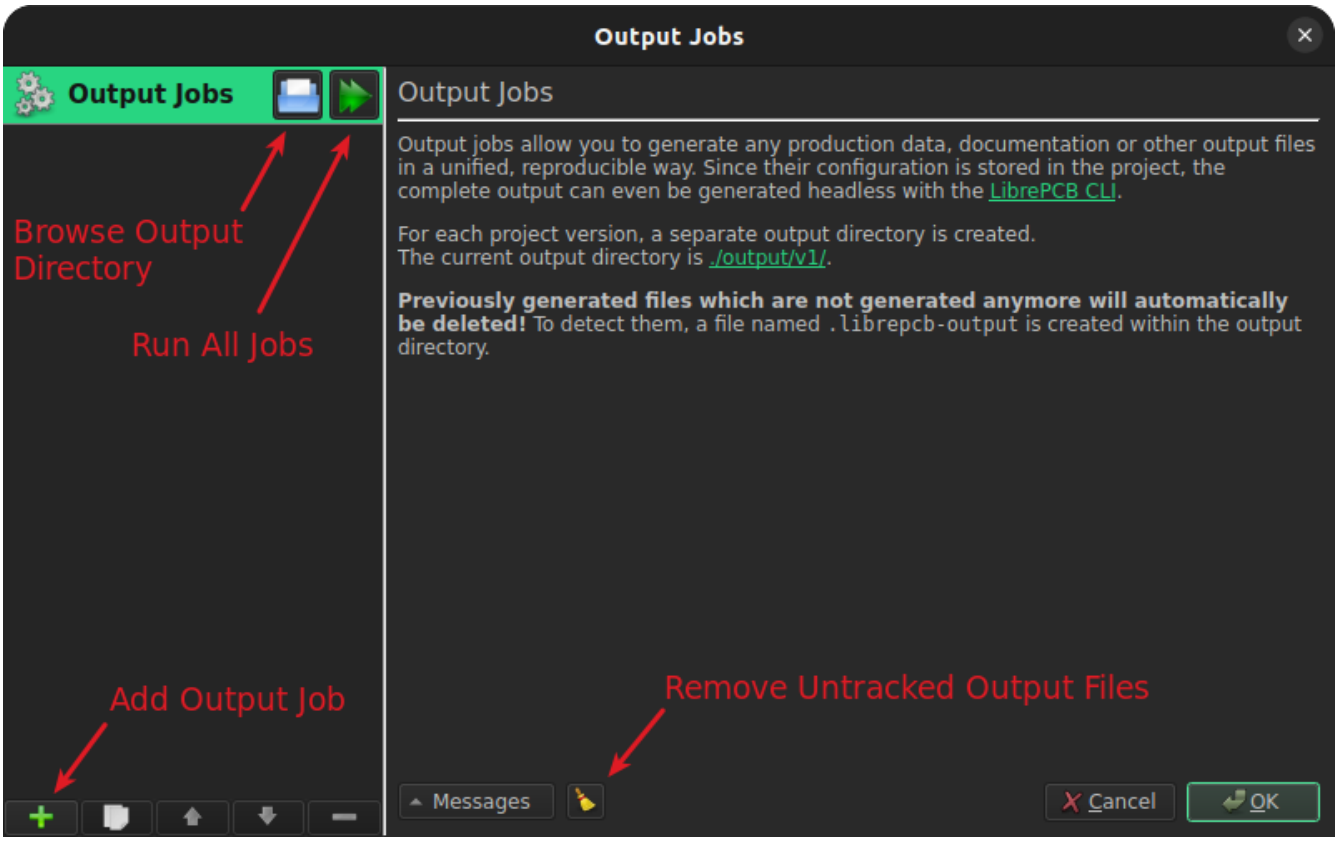
Alternatively you could also export your LibrePCB project as a `*.lppz` archive (**Project › Export *.lppz Archive**) and then upload this file with the web browser on fab.librepcb.org. This procedure might be useful if for some reason the direct upload is not desired or doesn't work (e.g. due to a corporate firewall).

Generate Production Data

Instead of using LibrePCB Fab, of course you can also generate the production data manually and forward these files to any PCB manufacturer you like. For this, open the **Output Jobs** dialog from the sidebar (`F11`):



This opens all output jobs of the project in a new window:



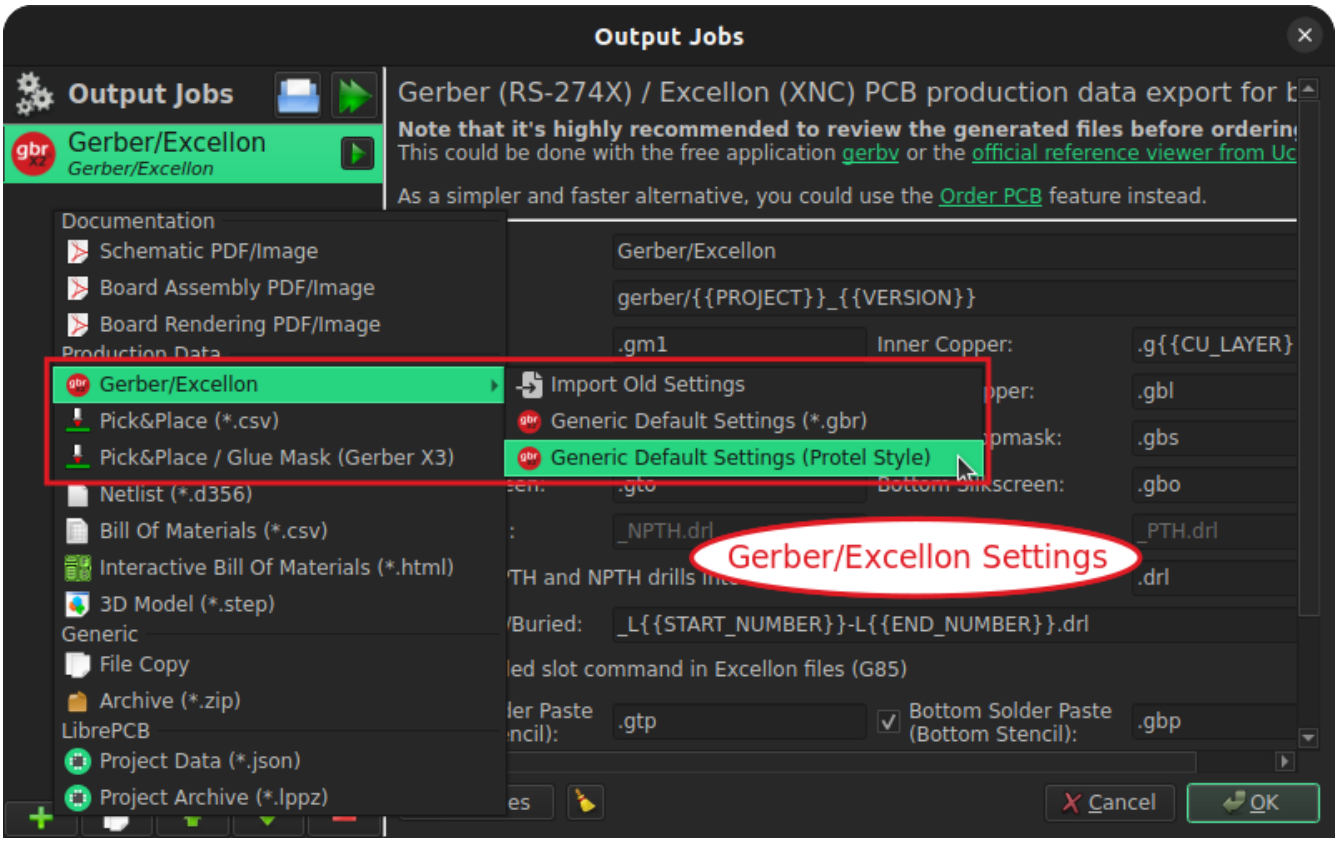
Then for any output you like to generate, click on the **[+]** button at the bottom left. See the following sections for details on the available jobs.

Any files generated through output jobs will be written to the path `./output/<VERSION>/` within the project directory, where `<VERSION>` is the project's version number as defined in the **Project Setup** dialog. So make sure the version number is set as desired to avoid overwriting e.g. the output files of a previous PCB version.

Once you set up all output jobs, just click on the "Run all jobs" button and all files will be written to the output directory. Then click on **[OK]** and save the project to store the output jobs configuration.

Gerber/Excellon

For the Gerber/Excellon production data you need to choose the settings of the Gerber/Excellon export. There are two different presets built-in, a default style and a Protel style. Generally you should determine what format your PCB manufacturer accepts. Many manufacturers accept Protel-style settings, so if you're unsure, choose **Gerber/Excellon (Protel Style)**.

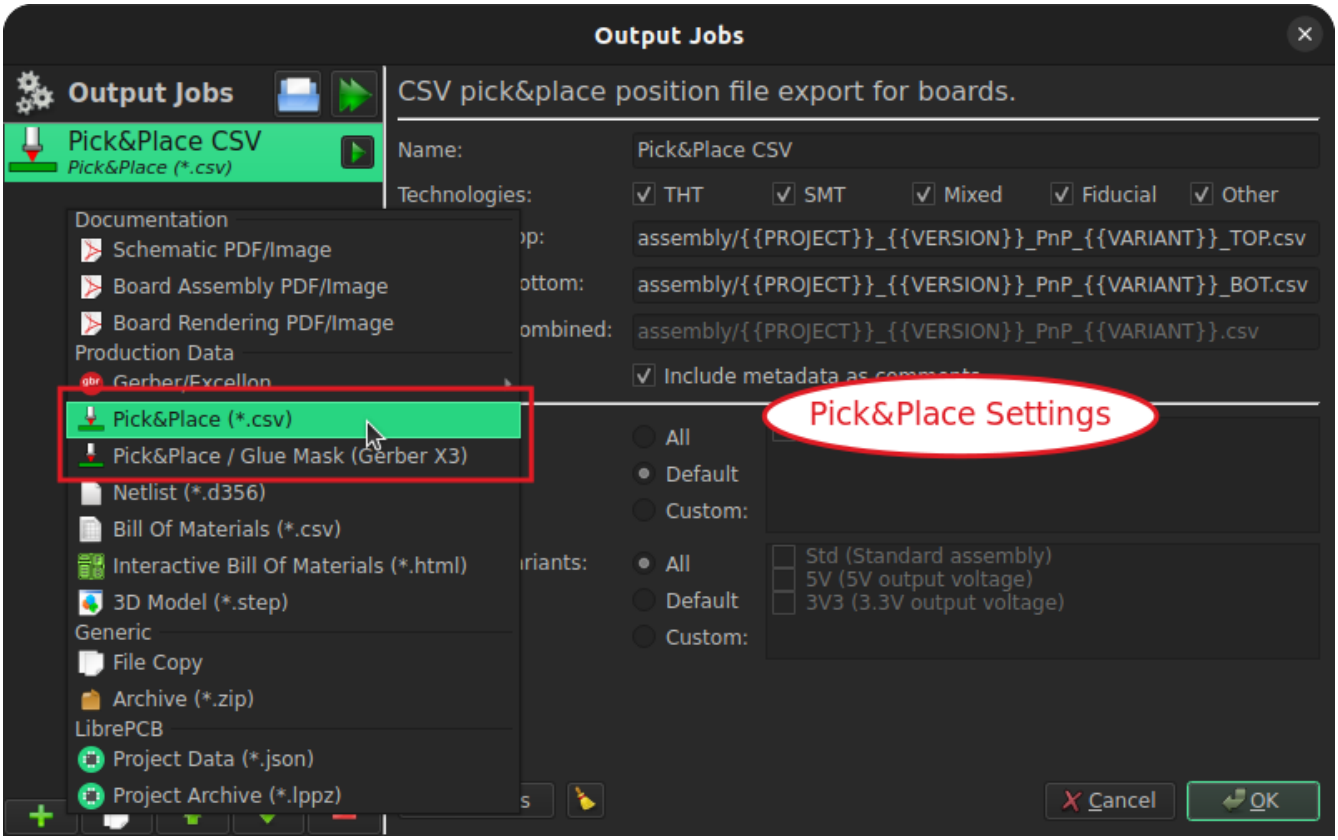


If required, the settings can now be adjusted manually.

It's highly recommended to cross-check the generated files with third-party tools like the reference Gerber viewer (preferred) or gerbv. LibrePCB developers are not responsible for any implications caused by wrong production data.

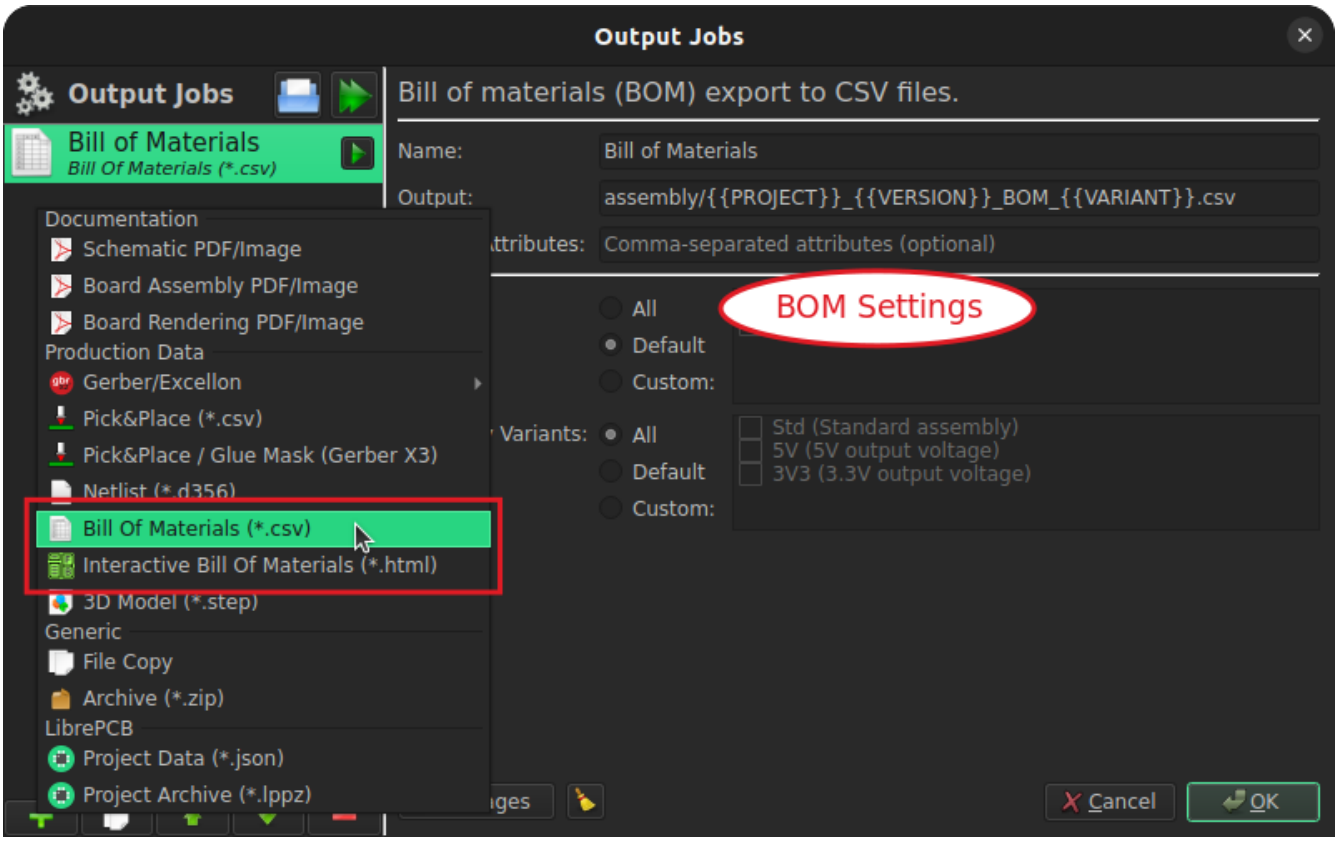
Pick&Place Data

If you also need pick&place files for automated assembly, just choose **Pick&Place (*.csv)** (or alternatively ther Gerber X3 variant):



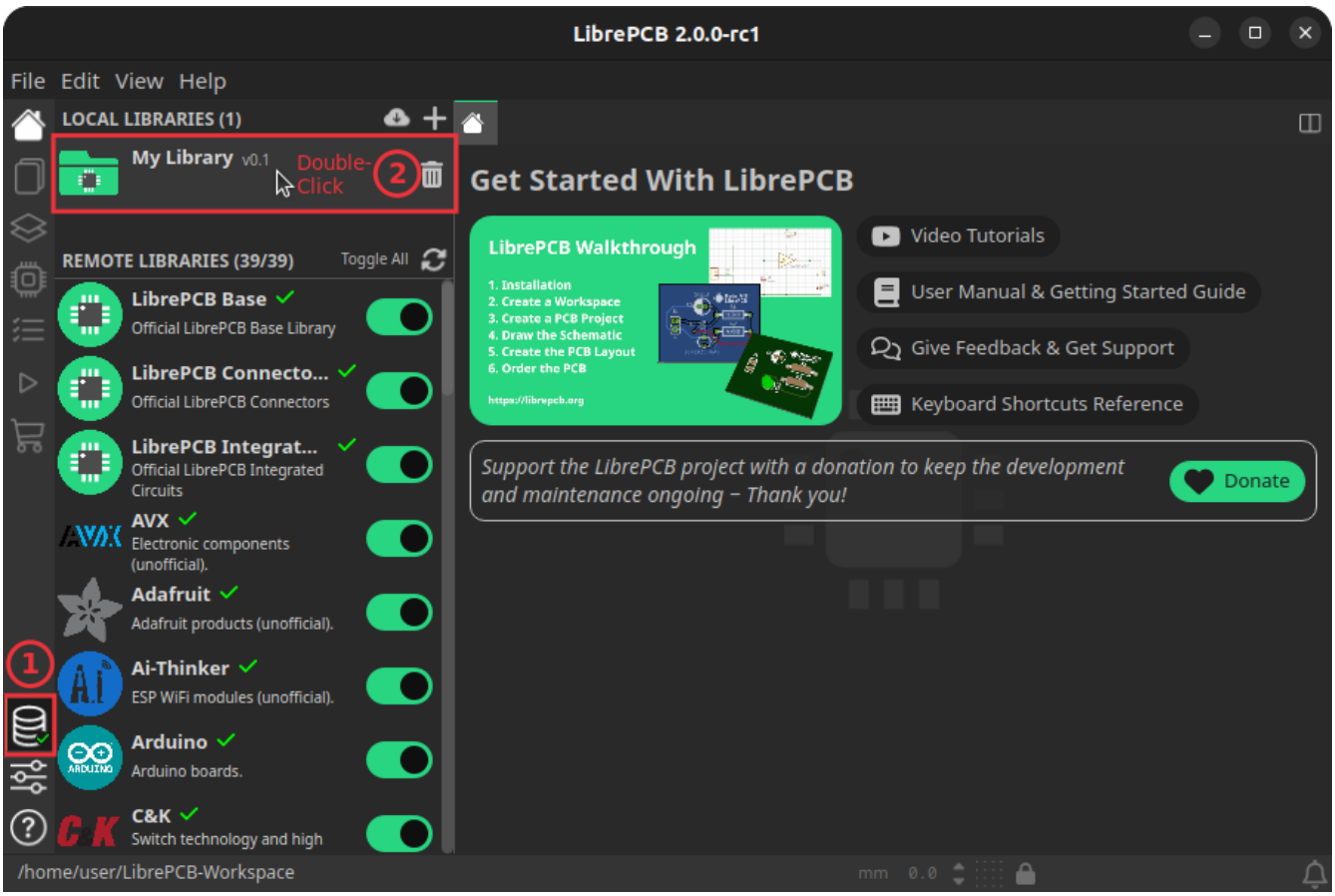
Bill of Materials

To get a bill of materials (BOM), add the output job **Bill Of Materials (*.csv)**, and/or the interactive HTML BOM if you plan to assembly your board by hand:

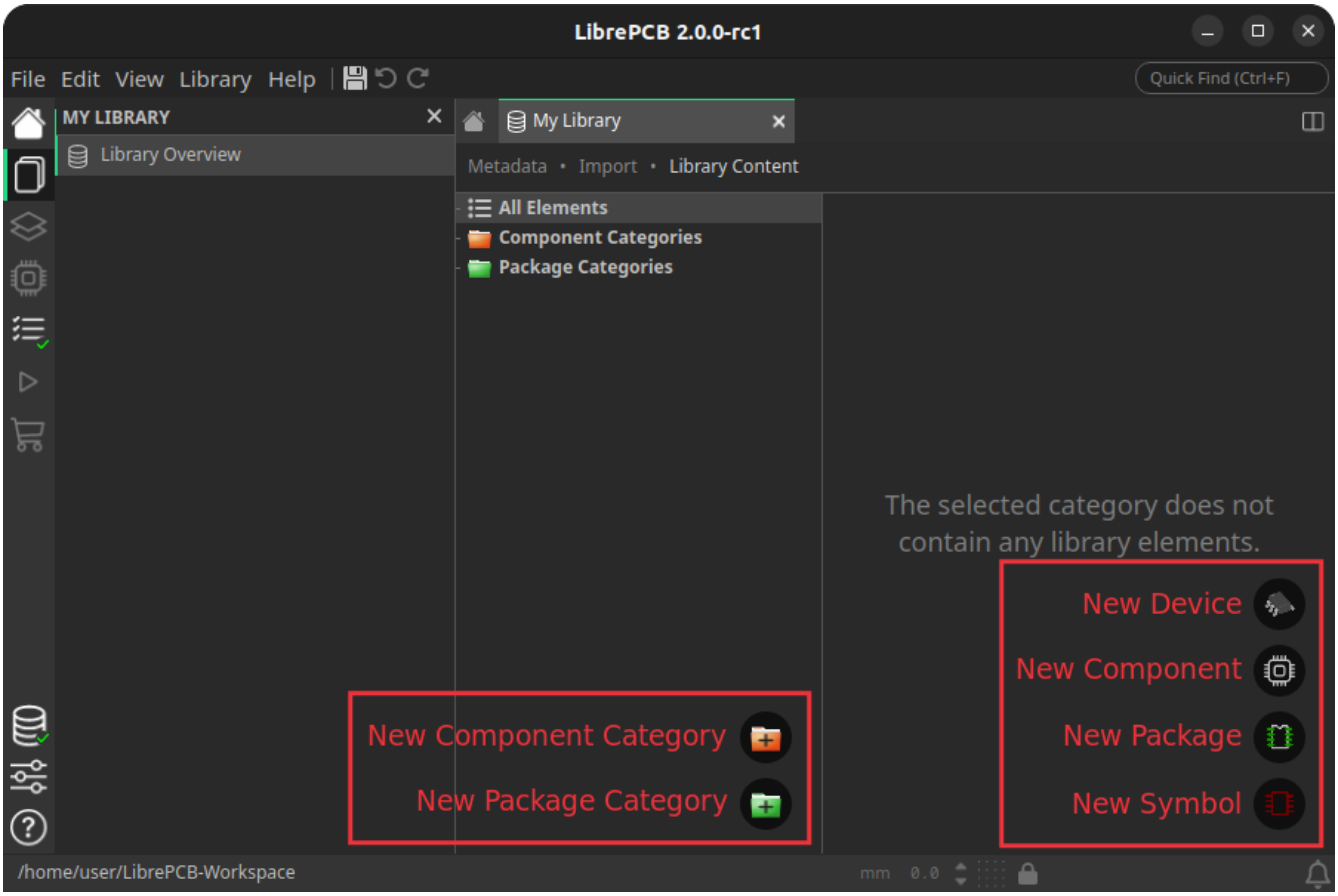


Create Library Elements

Sooner or later you'll need to create your own library elements in your local library you have created previously. Open that library now in the libraries panel:



In the library editor, there are six buttons to create the six different kinds of library elements which LibrePCB is based on (no worries, important are only four of them). So every new library element you will create during this tutorial, starts with one of those buttons:



Concept Overview

But first 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 to make them browsable in the category tree you used in the schematic editor for adding components/devices. 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 ore more symbols). But it does not define how the part looks physically on a board. Examples: *Resistor*, *Bipolar Capacitor*, *4-channel OpAmp*

Package Category

Exactly the same as the component category, but for packages instead of components. This allows to browse packages in a category tree. 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 and defines the pinout to connect component signals with package pads. Examples: *0805 Resistor*, *LM358D*, *STM32F103C*

The order of this list is 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.

No worries if this is a bit too much theory for now. The rest of the tutorial is more practical, which will help you to understand the concept step by step.

Our Example: LMV321LILT

Let's say you want to create the part **LMV321LILT** (OpAmp, see datasheet) from A to Z. We will now create all the necessary library elements for the LMV321LILT, though in practice you only need to create the elements which do not exist already. You can even use elements from other libraries, for example the symbol from library *X*, the component from library *Y* and the package from library *Z*.

It's really important to understand how to re-use already existing components and packages. In many cases, your desired component (e.g. *Single OpAmp*) and package (e.g. *SOT23-5*) already exist in our libraries. **Then the only element you have to create is the device, which just takes a minute.**

If you want to learn the whole concept, follow the tutorial (recommended). If you only want to create a device, skip the basics and go directly to the device tutorial.

Here an overview which library elements we'll create for the LMV321LILT:

- **Component category**: *Integrated Circuits › Linear › Amplifiers*
- **Symbol**: *Single OpAmp*
- **Component**: *Single OpAmp*
- **Package category**: *SOT*
- **Package**: *SOT23-5*
- **Device**: *LMV321LILT*

Component Category

First you should create a component category for the LMV321LILT (if it doesn't exist already). Click

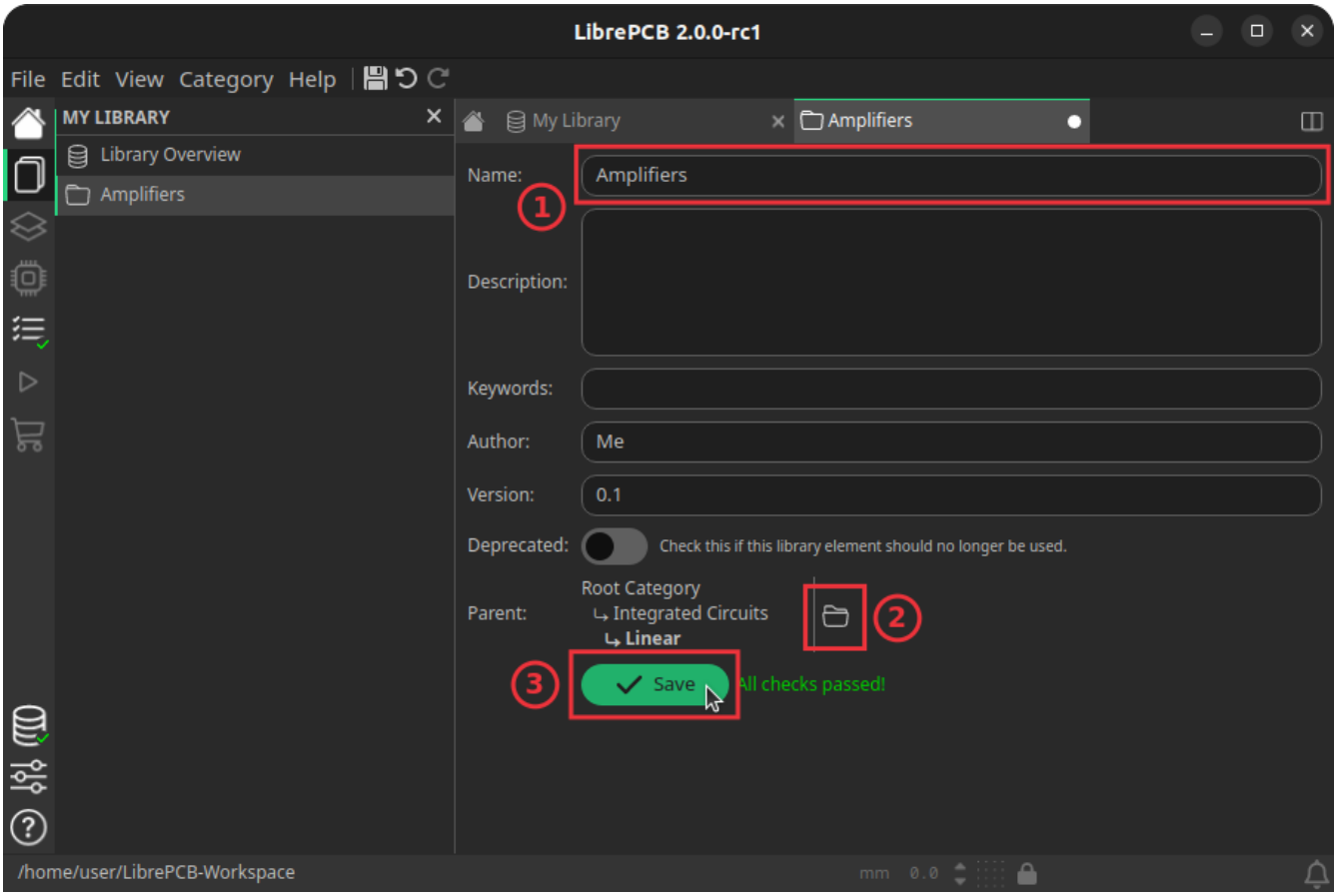
on **[New Component Category]** in the library editor, choose a suitable (generic!) name and select a parent category. **You may first need to create the required parent categories.**

Creating component categories is optional. Everything works even without creating such categories so if you're in a hurry, just skip this step. However, categories help to keep your libraries organized and to quickly find components in the schematic editor.

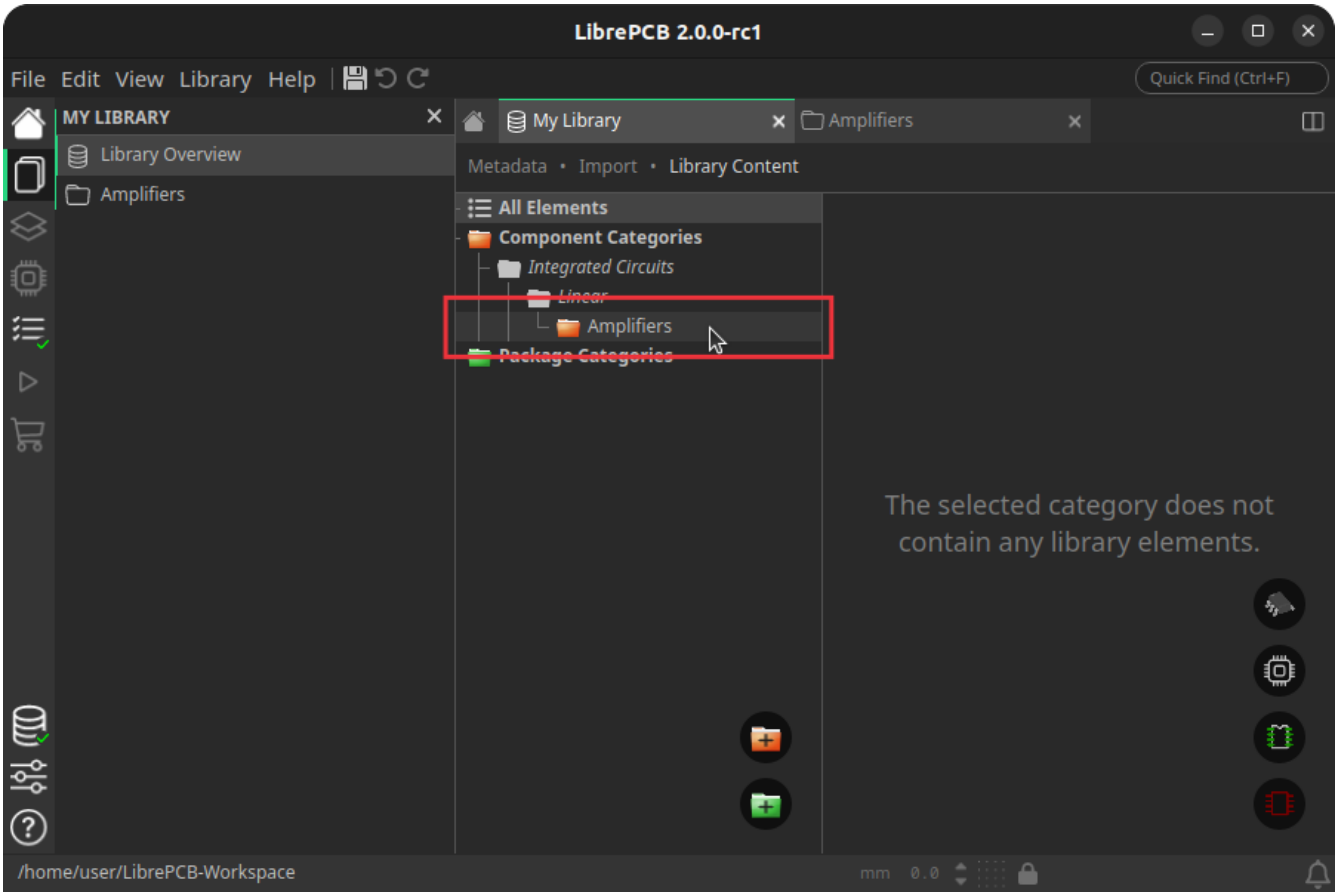
In our example, we choose the following properties (any other metadata is optional):

- **Name**: *Amplifiers* (since the LMV321LILT is an amplifier)
- **Parent**: *Integrated Circuits › Linear* (let's assume these categories exist already)

If you're unsure about the category name, take a look at the navigation trees of digikey.com or mouser.com for inspiration. But don't use a nesting level higher than 3 levels (usually 2 levels are enough).



After clicking on **[Save]**, your first component category is already complete! It may just take a moment for the background library scan until the new component category appears in the category trees. Then you will see the new category in your library:

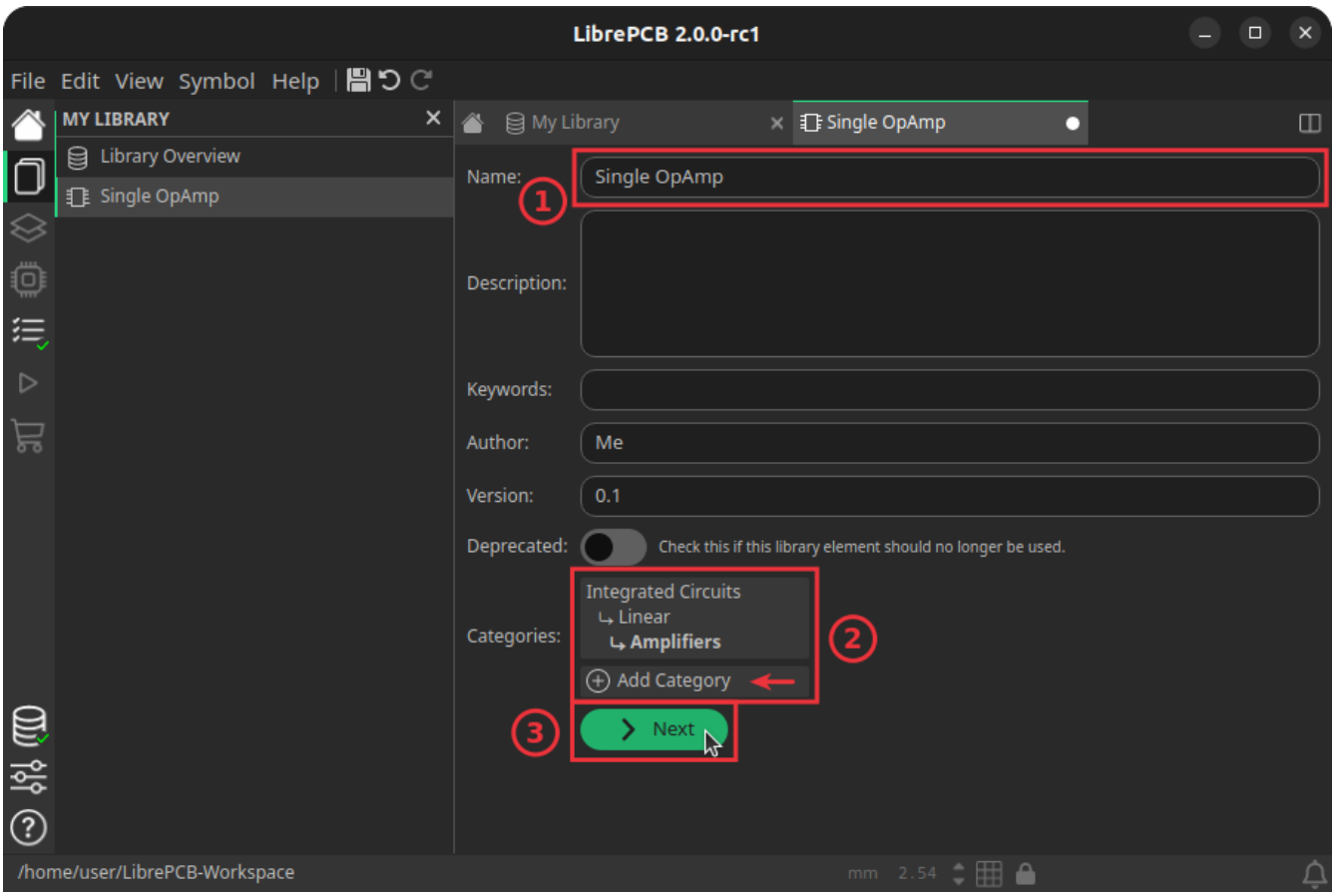


▼ *Component categories available in the LibrePCB Base library*

- Connectors
  - Card Edge Connectors
  - Coaxial Connectors
  - Computer Connectors (USB, HDMI, Ethernet, ...)
  - D-Sub Connectors
  - Direct Wire to Board Connectors
  - Flex Connectors
  - Pin Headers (Male)
  - Pin Sockets (Female)
  - Terminal Blocks
- Discrete Semiconductors
  - Diodes, Rectifiers
  - Transistors, Thyristors
- Electromechanical
  - Battery Holders
  - Board Spacers
  - Card Slots
  - Fuse Holders
  - IC Sockets
  - Motors, Actuators
  - Relays
  - Switches
- Integrated Circuits
  - Clock, Timing
    - Clock Generators, PLLs, Frequency Synthesizers
    - Real Time Clocks (RTC)
    - Timers, Oscillators
  - Data Aquisition
    - Analog Front End (AFE)
    - Analog to Digital Converters (ADC)
    - Digital Potentiometers
    - Digital to Analog Converters (DAC)
  - Data Processing
    - Complex Programmable Logic Devices (CPLD)
    - Digital Signal Processors (DSP)
    - Field Programmable Gate Arrays (FPGA)
    - Microcontrollers, Microprocessors
    - Programmable Logic Devices (PLD)
    - Systems On Chip (SoC)
  - Interface
    - Analog Switches, Multiplexers, Demultiplexers
    - Drivers, Receivers, Transmitters
    - Signal Buffers, Repeaters, Splitters
    - USB Interfaces
  - Isolators
  - Linear
    - Amplifiers
    - Comparators
  - Logic
    - Buffers, Drivers
    - Counters, Dividers
    - Flip Flops, Latches
    - Gates, Inverters
    - Shift Registers
    - Signal Switches, Multiplexers, Decoders
    - Translators, Level Shifters
  - Memory
  - Power Management (PMIC)
    - AC/DC Converters
    - Battery Chargers
    - Battery Management
    - Full/Half Bridge Drivers
    - Gate Drivers
    - LED Drivers
    - Motor Drivers/Controllers
    - Supervisors
    - Voltage References
    - Voltage Regulators
  - Miscellaneous
- Optoelectronics
  - Displays
  - LEDs
  - Nixie Tubes
  - Optocouplers
- Passive
  - Capacitors
  - Crystals, Oscillators, Resonators
  - Inductors, Coils, Chokes, Filters
  - Resistors
- Schematic Symbols
  - Schematic Frames
  - Supply Symbols
- Sensors, Transducers
  - Current Transducers
  - Encoders
  - Gas Sensors
  - Humidity Sensors
  - Magnetic Sensors
  - Motion Sensors
  - Optical Sensors
  - Particle and Dust Sensors
  - Temperature Sensors
- Single Board Computers, Dev/Eval Boards

Symbol

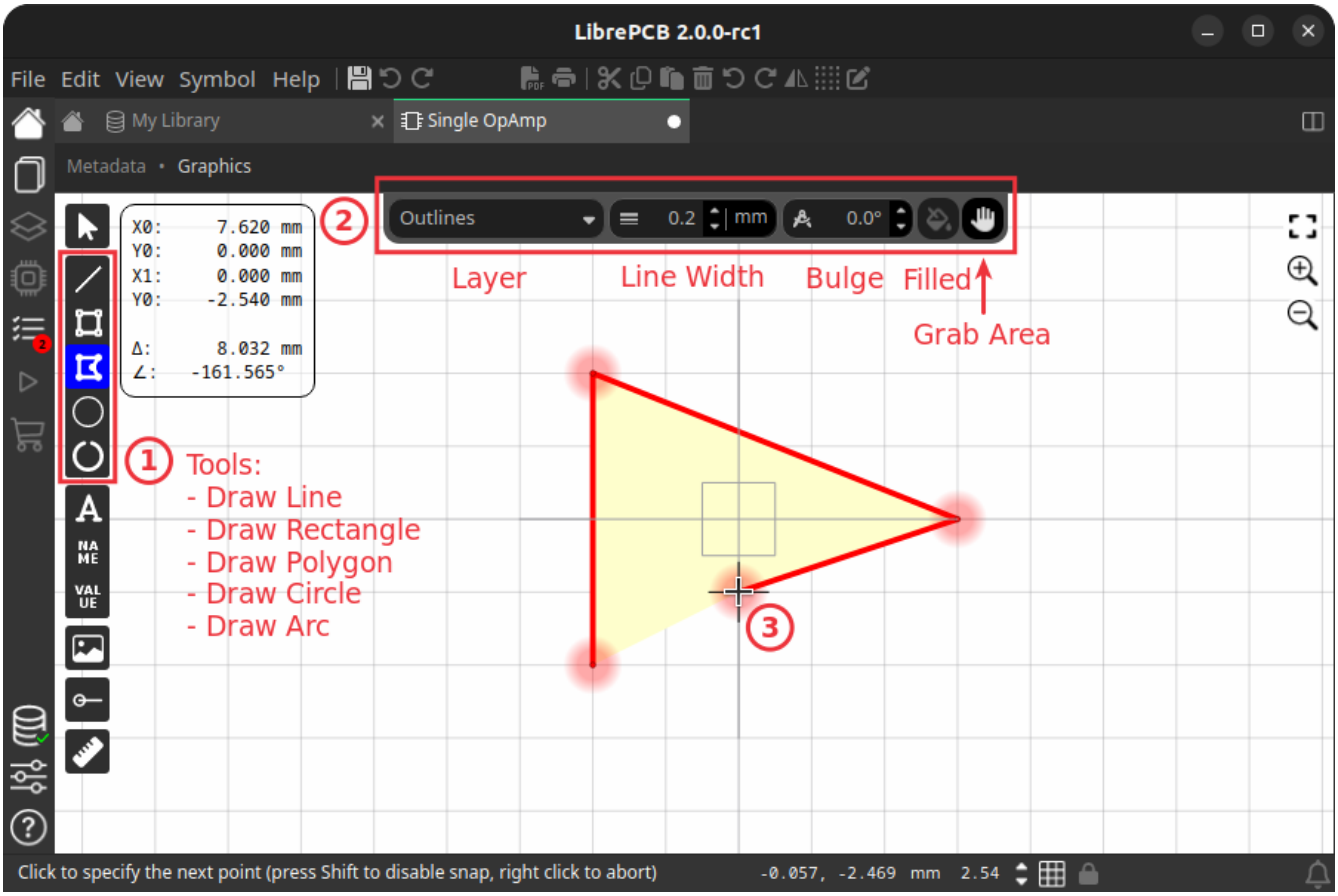
Now we need to create a symbol for the OpAmp. Click on **[New Symbol]** in the library editor, choose a name and the component category we just created and click **[Next]**:



Draw Polygons

Now let's draw the graphical objects of the symbol:

1. Choose a tool. There are several similar tools available, but often you need only the **[Draw Rectangle]** or the **[Draw Polygon]** tool.
2. Specify the polygon properties. **For the symbol's "body", choose the *Outlines* layer.** When checking *Grab Area*, you'll be able to drag the symbol in the schematic editor by clicking on the polygon's area.
3. Draw the polygon with the cursor.



Note that to get a bigger working area, we collapsed the side panel by clicking on the currently active panel button in the sidebar.

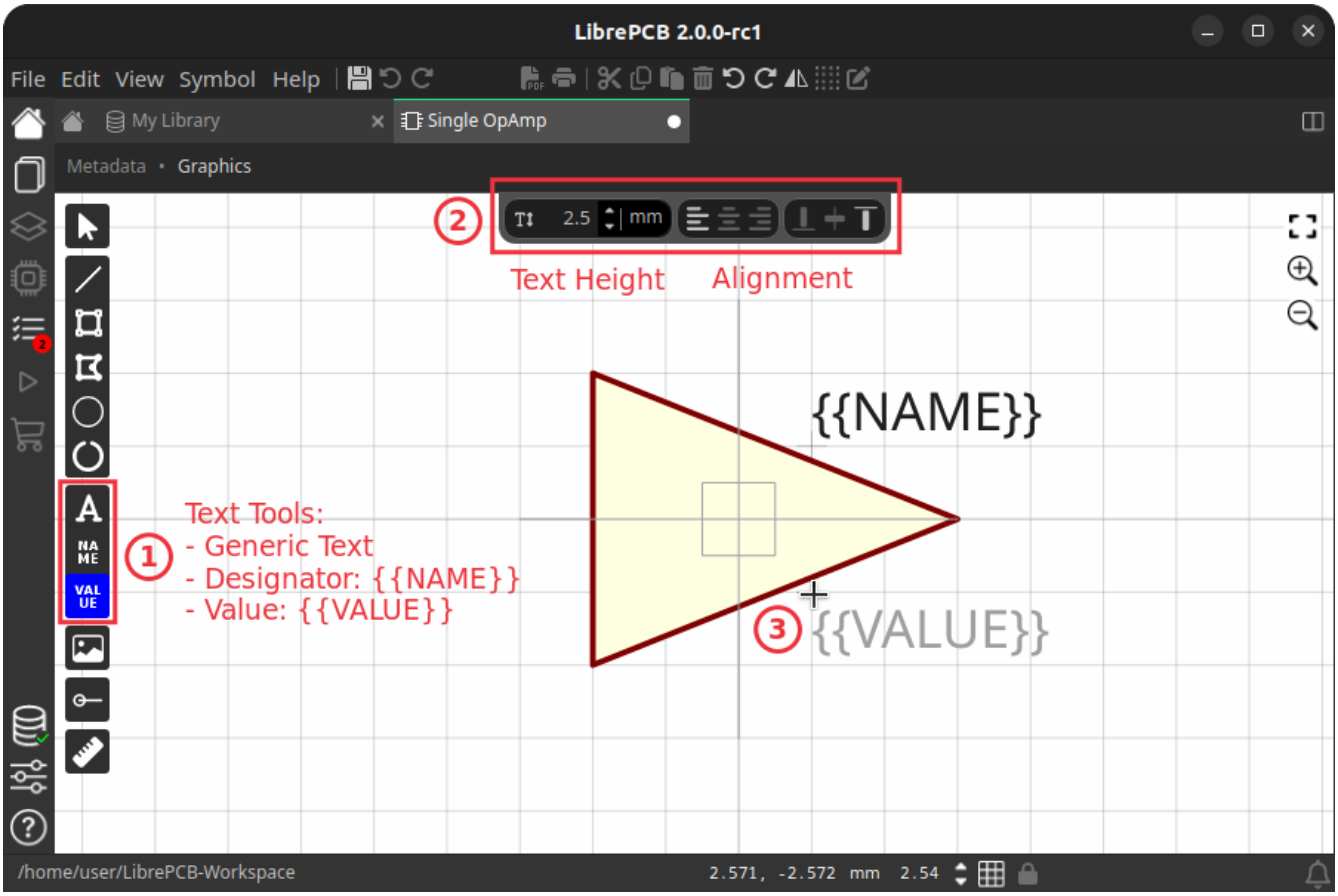
Add Texts

Then you should **add at least two text objects**:

- **Name**: Using the placeholder `{{NAME}}` which will be substituted by the component's designator (e.g. "R5") in the schematics.
- **Value**: Using the placeholder `{{VALUE}}` which will be substituted by the component's value (e.g. "100nF") in the schematics.

For convenience, there are dedicated tools for these two text objects. Use them as follows:

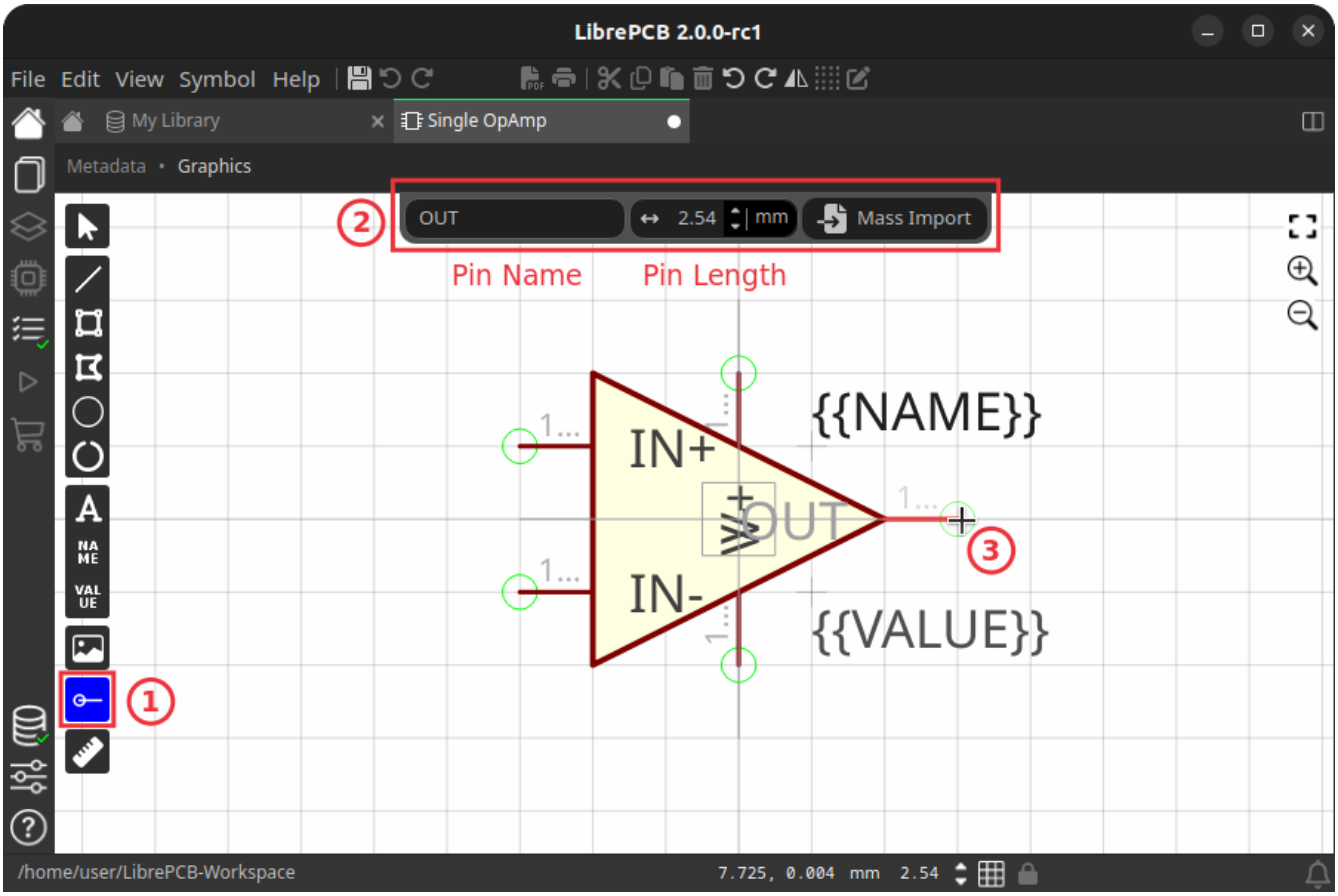
1. Start one of the text tools.
2. If needed, adjust the text properties in the toolbar.
3. Place the text object with a mouse click. Press `R` or `Right Click` to rotate or `M` to mirror the alignment while moving.



Add Pins

Then, the most important thing is to add pins since these are required later in the schematics to attach wires to the symbol.

1. Start the **[Add Pin]** tool.
2. Choose a reasonable (unique!) pin name and length. Press `Tab` to move the focus into the name input field.
3. Place the pin with a mouse click. Press `R` or `Right Click` to rotate while moving.



The overlapping pin texts look a bit ugly, but let's ignore that for the moment.

It's not possible to add multiple pins with the same name. If your device for example has multiple *GND* pads which are all connected together (i.e. you don't need to distinguish between them), add **only one** *GND* pin to the symbol. If you need to distinguish between the different pins, assign unique names (e.g. *GND_1*, *GND_2* etc.).

Now **save the symbol** to let the background scan picking up the new symbol (this takes a moment) before you can use this symbol in a component.

Recommendations

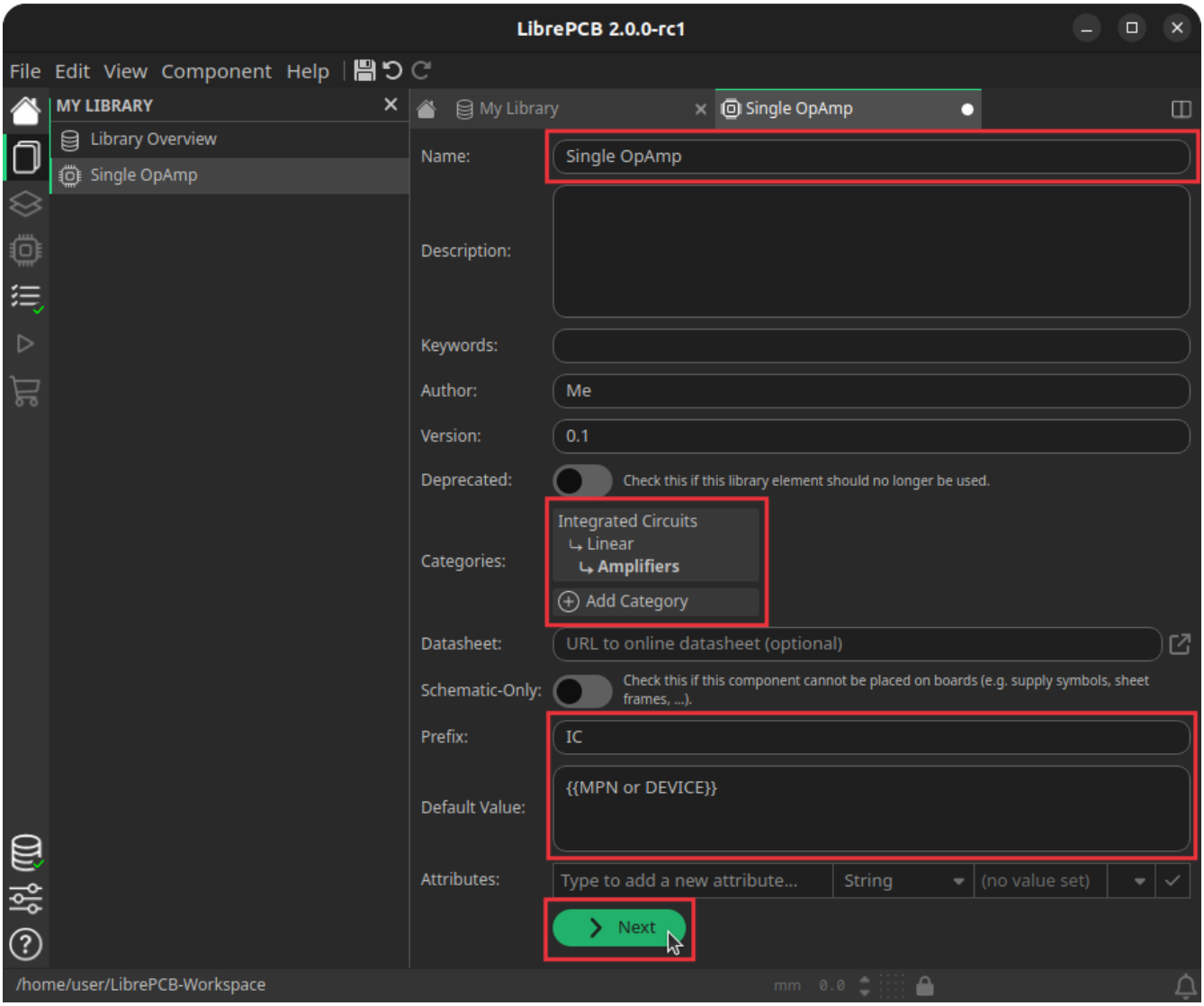
For details about how symbols should be designed, please take a look at our symbol conventions. The most important rules are:

- 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 placed on the 2.54mm grid.
- There should be text elements for `{{NAME}}` and `{{VALUE}}`.

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), you should enter a generic name like *Single OpAmp*.

Choose **[New Component]**, enter the name, assign the component category we created previously, and specify its prefix and default value (see explanations below):



Of course you may also set more properties, but it's not strictly required. The most important properties beside name and categories are:

Schematic-Only

Check this if the component must not appear on a board, but only in the schematics. This is typically used for schematic frames.

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.

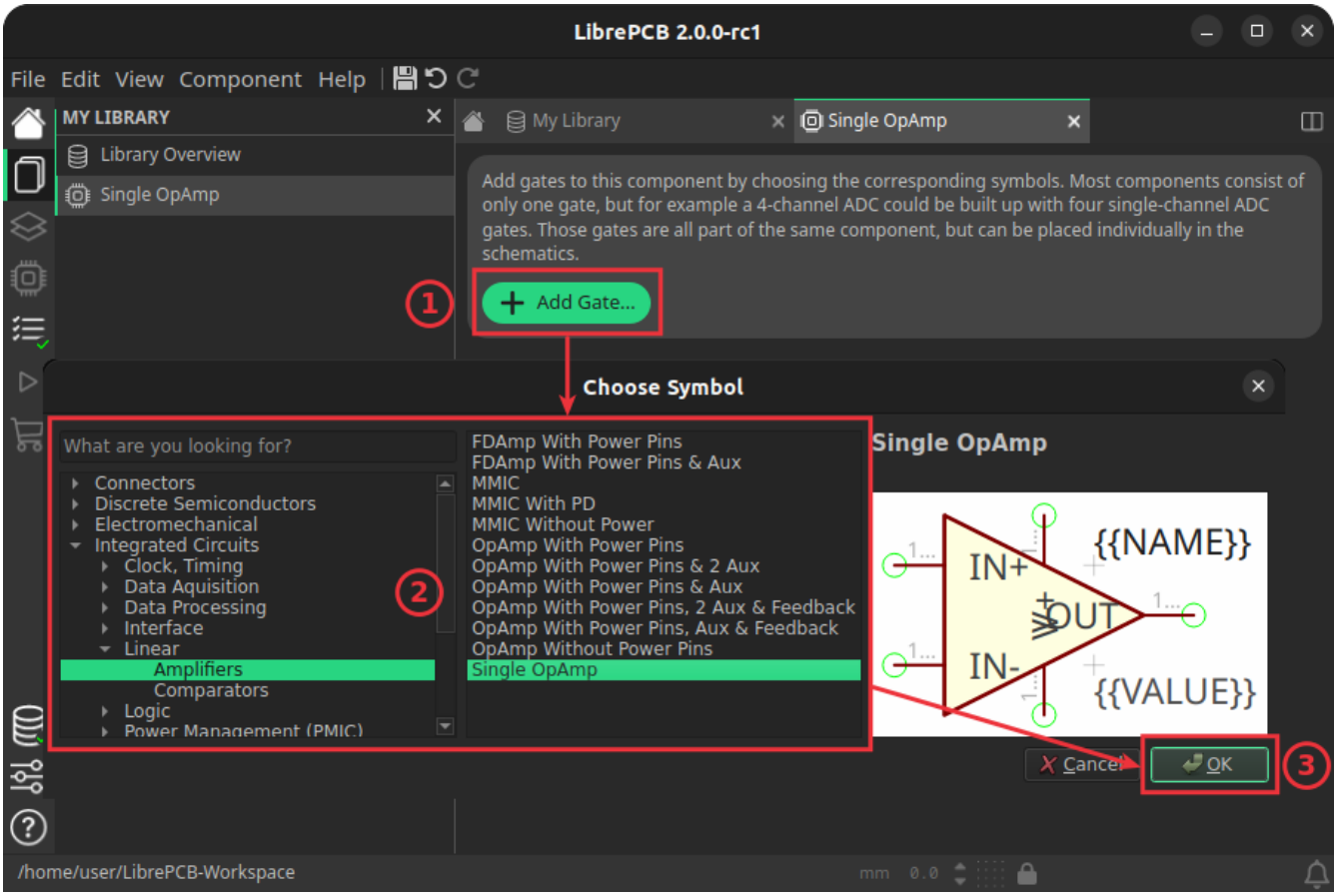
Default Value

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 as 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 `{{MPN}}`, `{{DEVICE}}` or `{{CAPACITANCE}}`. If you are unsure, just leave it empty, the component editor will help you to assign a value later.

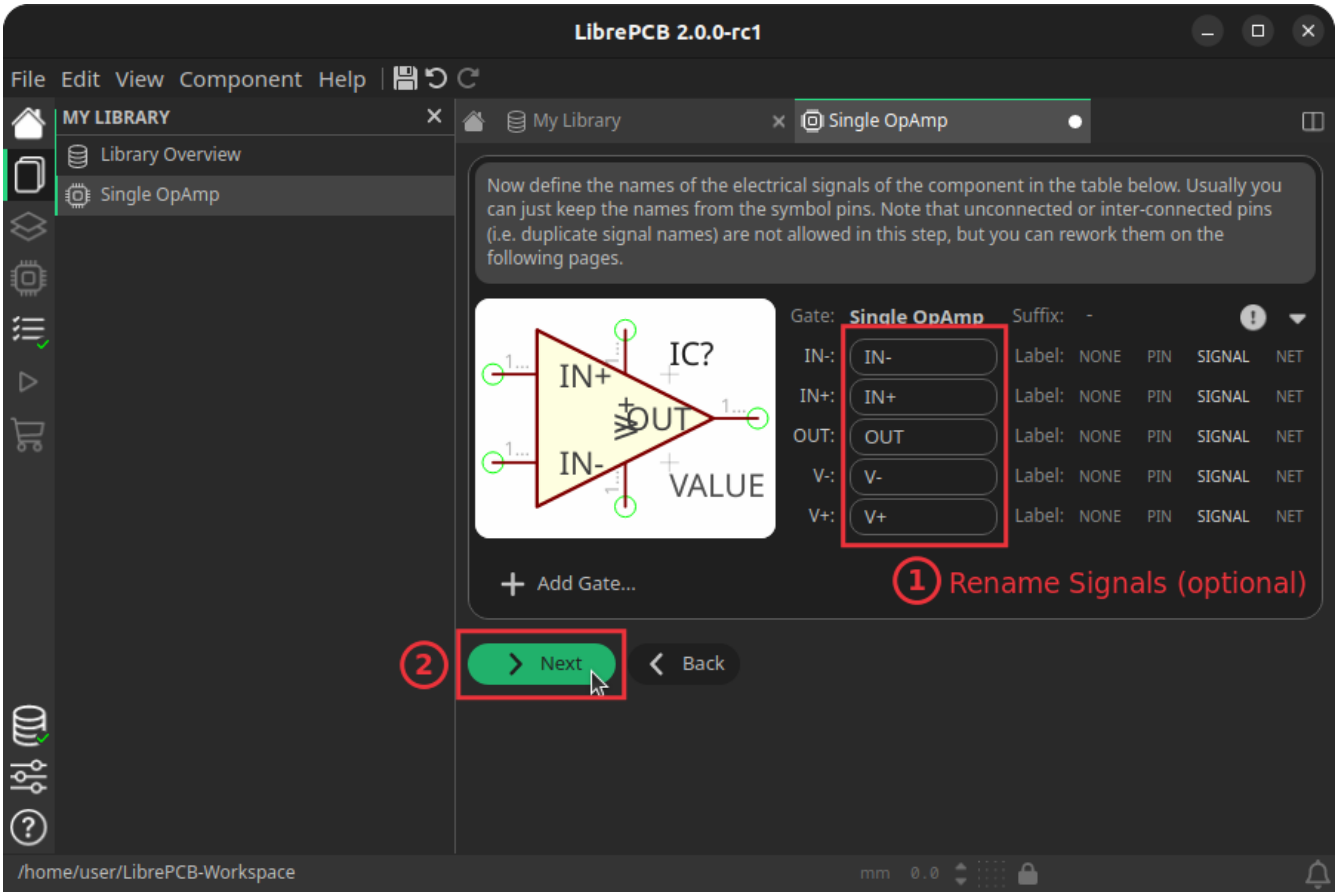
After specifying the properties, click on **[Next]**.

Add Gates

Now you need to choose the symbols which represent the component in schematics, called *gates* in this context. Most components have only one gate, but you can also add more than one. For example a Quad OpAmp could consist of a power gate and four amplifier gates. In our case, select the *Single OpAmp* symbol we created previously:



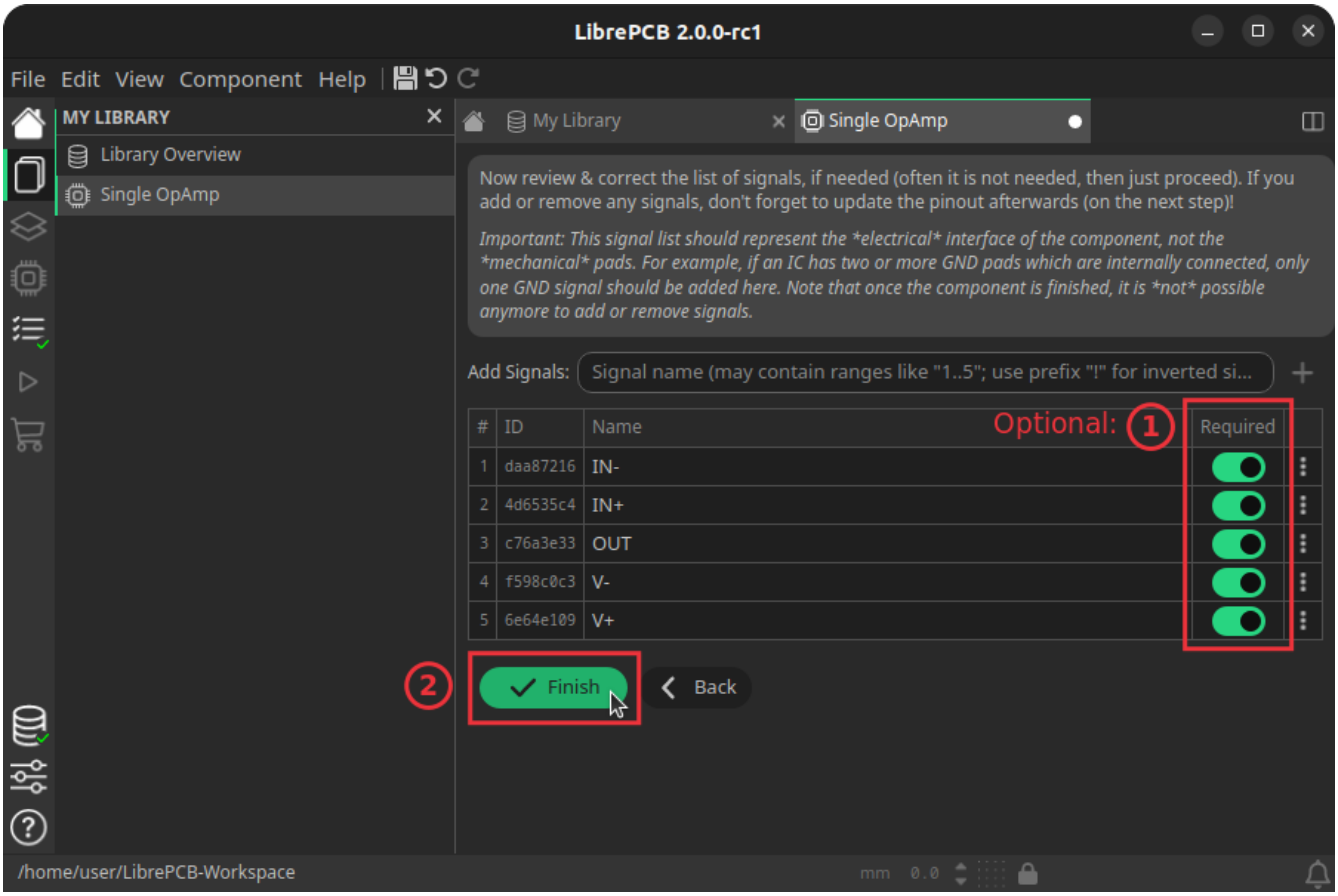
After you added all the required gates (usually only one), it is possible to rename the pins, which are called *signals* in this context. Renaming is required for example to ensure unique signal names if you add multiple identical gates, or to simply choose more expressive names if the symbol pin names are not suitable for a particular component. Often you can just keep the symbol pin names as-is:



Define Signals

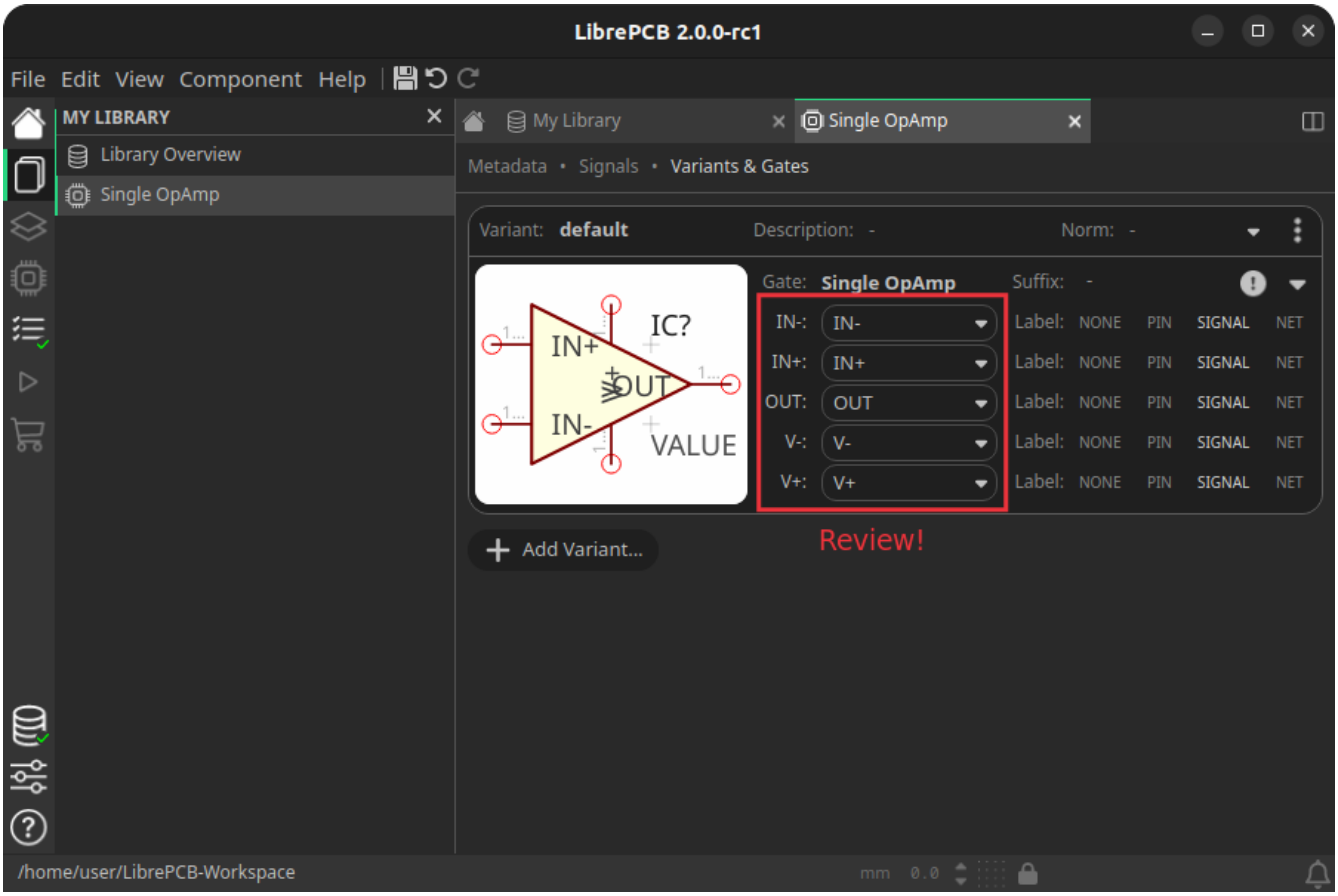
The next step is to define all so-called signals of a component. Signals represent the "electrical interface" of a component. For example a transistor consists of the signals *Base*, *Collector* and *Emitter*. For a component it's irrelevant whether the "real" transistor has multiple emitter pads, or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

LibrePCB automatically extracts the signals from the gates you added in the previous step, so usually you don't have to do this by hand. But sometimes you still need to make some adjustments, like deleting unused signals or changing the properties of some signals. For our OpAmp, we check the *Required* checkbox of all signals to ensure the ERC will raise a warning if these signals are not connected to a net in the schematics:



Review Pin-Signal-Map

After clicking on **[Finish]**, the component is complete. It is highly recommended to review the mapping from symbol pins to their component signals, especially for more complicated cases like multi-gate components of if you made any manual changes to the component signals in the previous step. If there is any mistake, you can just correct the pin-signal-map with the dropdowns:



In this component editor you can also make more changes, like hiding specific symbol pin labels or specifying suffixes for gates, but that is not important for now.

For our simple example this procedure might feel a bit complicated. This is due to the broad flexibility of the LibrePCB library approach which will save time in the long term due to high reusability of library elements.

The component which we created uses only very basic library features, but as soon as you understand the library concept in more detail, you will be able to easily create much more complicated library elements. We're sure you will learn to love the flexibility of the library concept step by step.

Recommendations

Following are the most important rules to create reusable components:

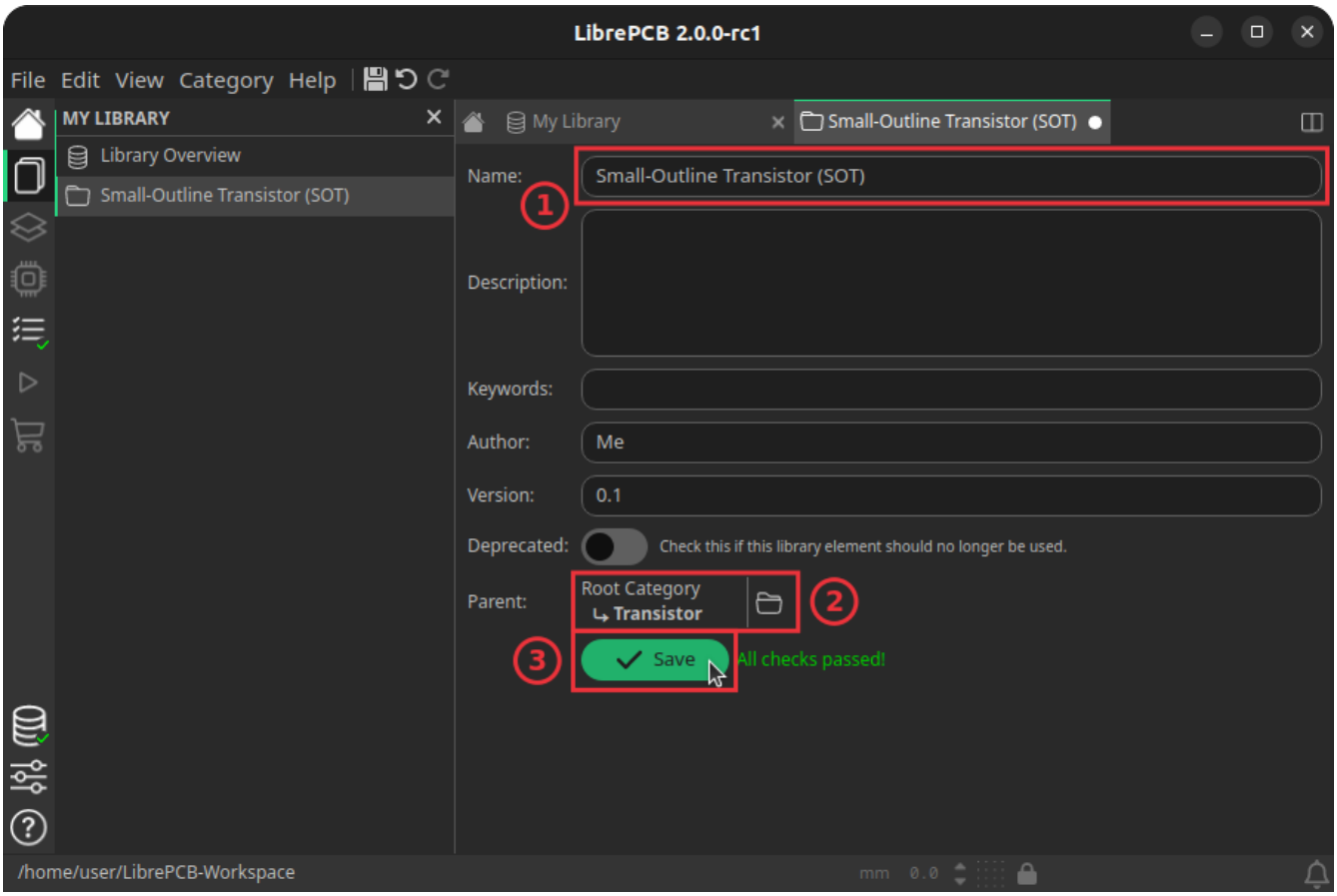
- Create generic components whenever possible. Only create specific components for manufacturer-specific parts (like microcontrollers).
- Generally name signals according their electrical purpose (e.g. *Source*, *Drain*, *Gate*).
- Don't add multiple signals which are considered as connected. Even for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

Package Category

Before creating a package for the LMV321LILT, you should (optionally) create a category for it. This is done exactly the same way as you already created the component category.

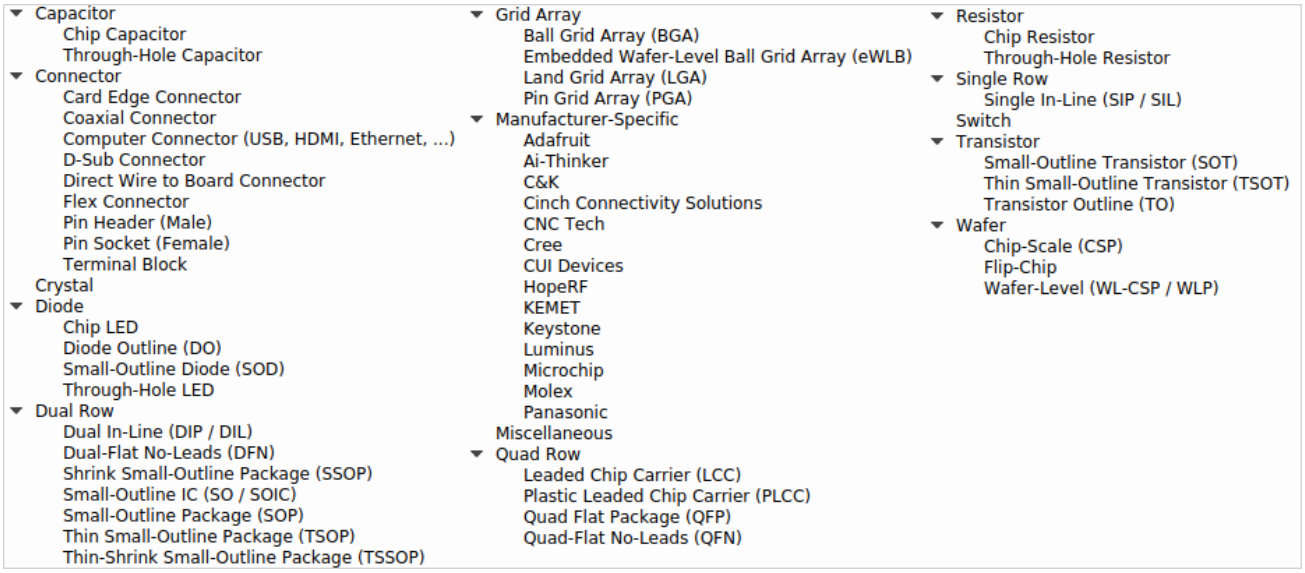
Since we need to create a *SOT23-5* package, let's choose the following properties for its category:

- **Name**: *Small-Outline Transistor (SOT)*
- **Parent**: *Transistor* (let's assume this category exists already)



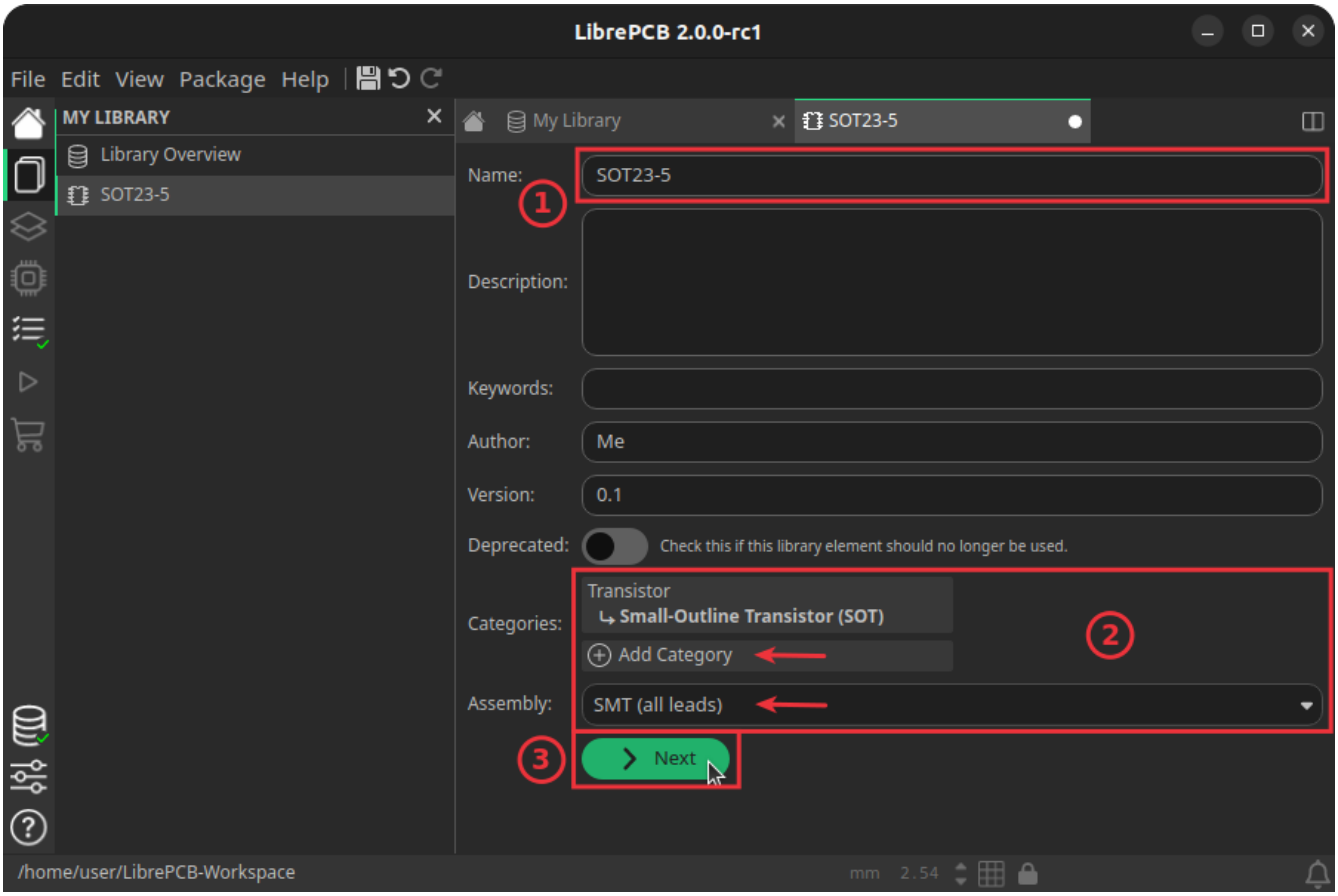
With a click on **[Save]** the package category is complete and after a moment the new category is ready to use.

▼ *Package categories available in the LibrePCB Base library*



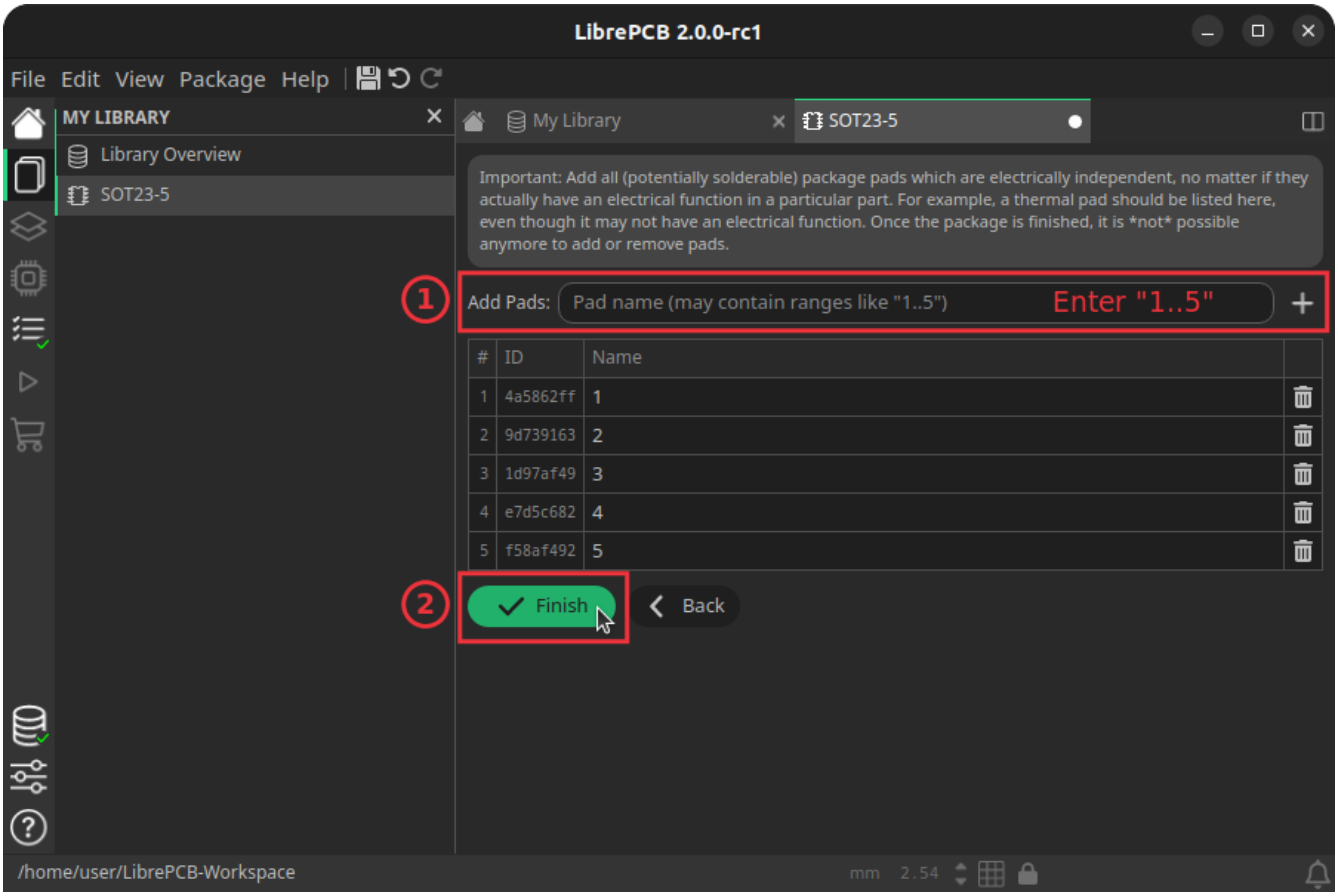
Package

Then you need to create the package for the LMV321LILT, which is called *SOT23-5*. As usual, click **[New Package]** and specify the name and category of the new package. In addition, it's recommended to also specify the assembly type, i.e. whether this is a through-hole (THT) or surface-mount (SMT) device:



Add Pads

Now you need to specify all pads of the package. The *SOT23-5* has 5 pads named from *1* to *5*, so you can just enter the term *1..5* and click on the **[+]** button (or press `Return`):



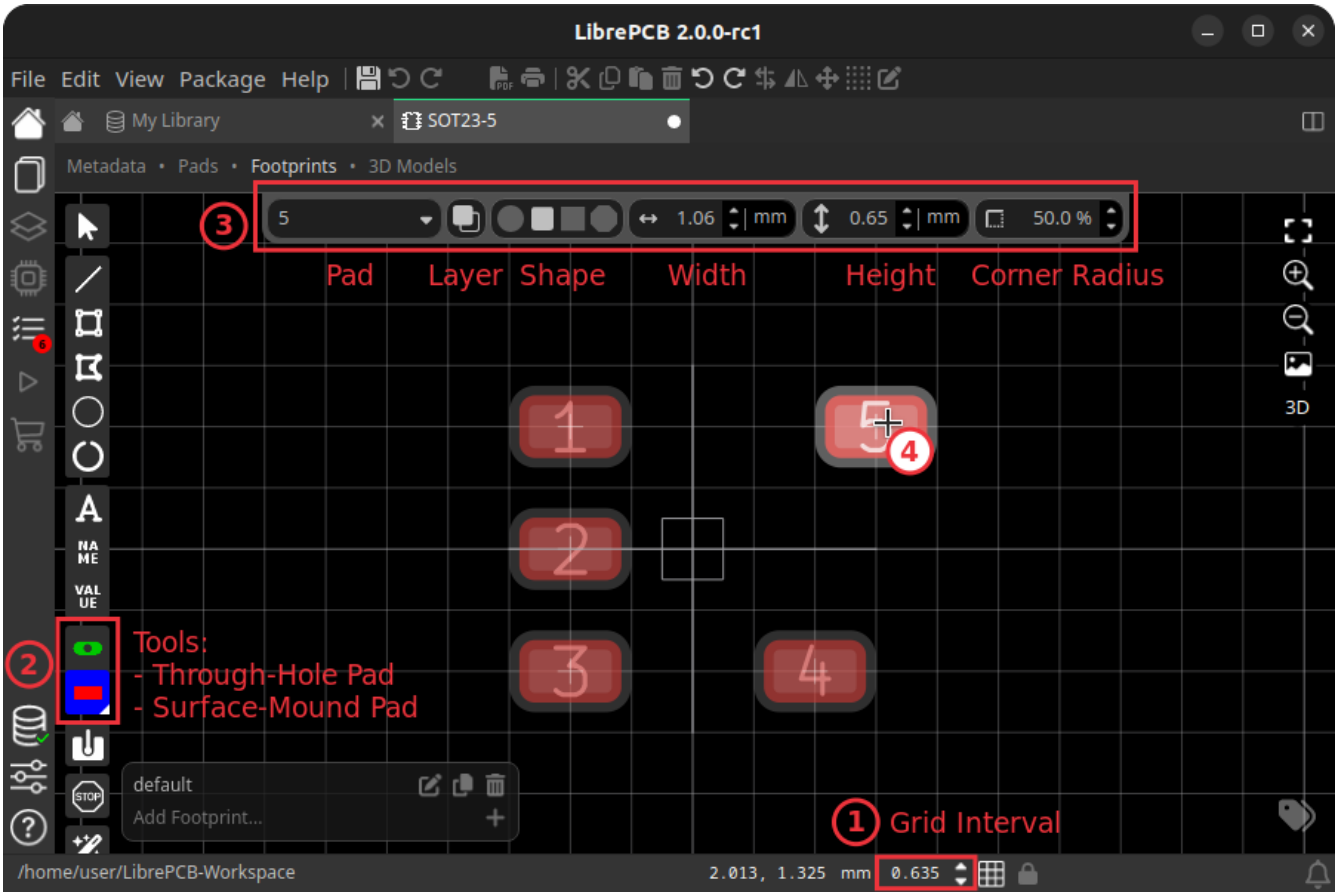
When adding the pads, don't consider their electrical functions or internal connections. For example if a transistor with three electrical signals has three pads plus a thermal pad connected to one of the other signals, the package has **four** pads in total. It's not relevant whether some of them are connected to each other *within* the package.

General rule of thumb: If in doubt, better specify too many pads than too few ;-)

Place Pads

After clicking on **[Finish]**, you can draw the footprint. It's recommended to start with placing the pads:

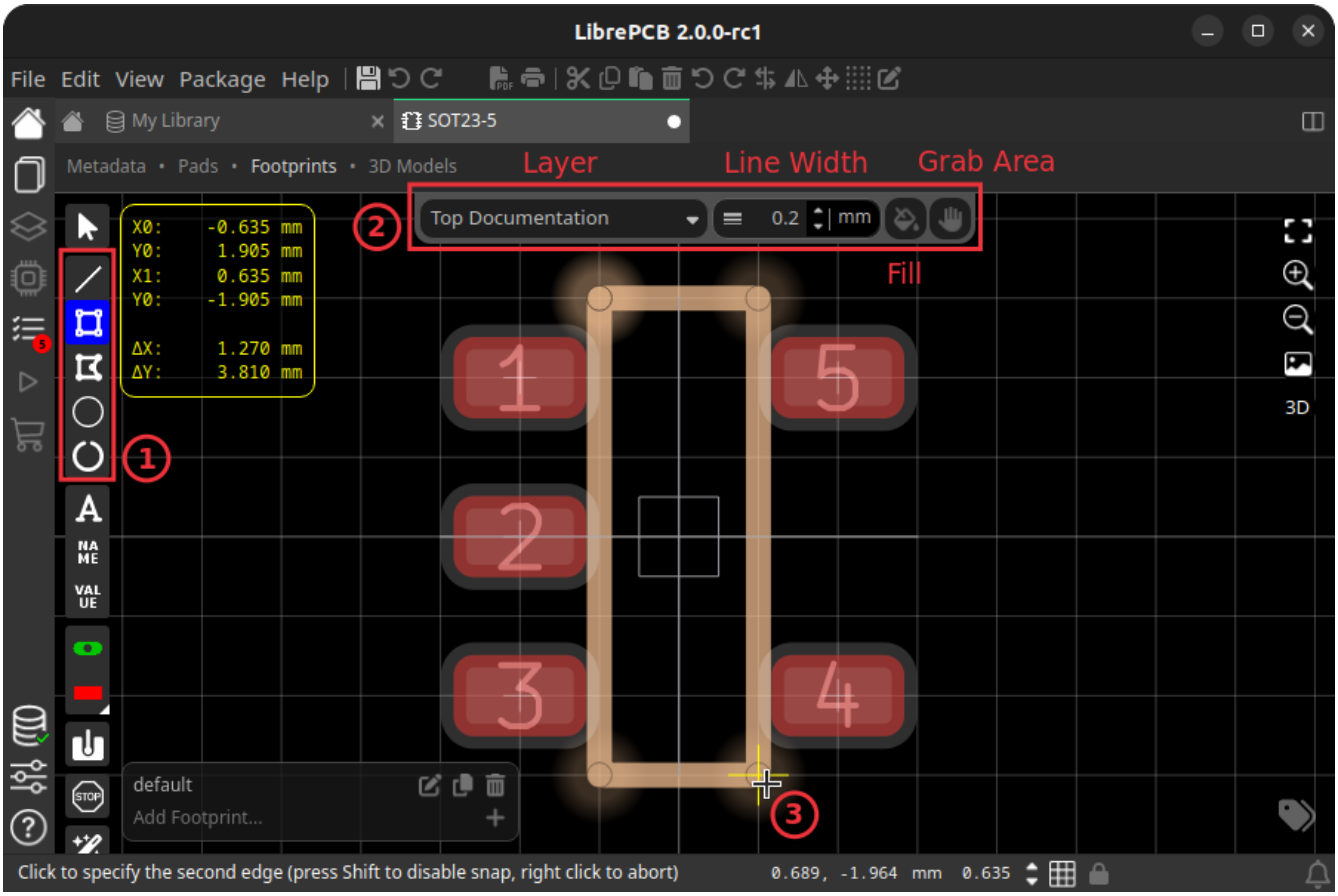
1. Set a reasonable grid interval in the status bar, if desired. You can either press `F4` and type in the new interval, or hover with the mouse on the number and scroll up or down to change the value.
2. Start either the **[Add THT Pad]** or **[Add SMT Pad]** tool.
3. Choose the package pad to place and specify its properties, most notably the shape and size.
4. Place the pad with a click. Press `R` to rotate it while moving.



The tool only allows to place pads on the grid. To specify exact coordinates, just place the pads rougly and open **[Properties]** from the pad's context menu (right-click) afterwards to enter exact values.

Draw Polygons

Then add graphical object just as done in the symbol editor:



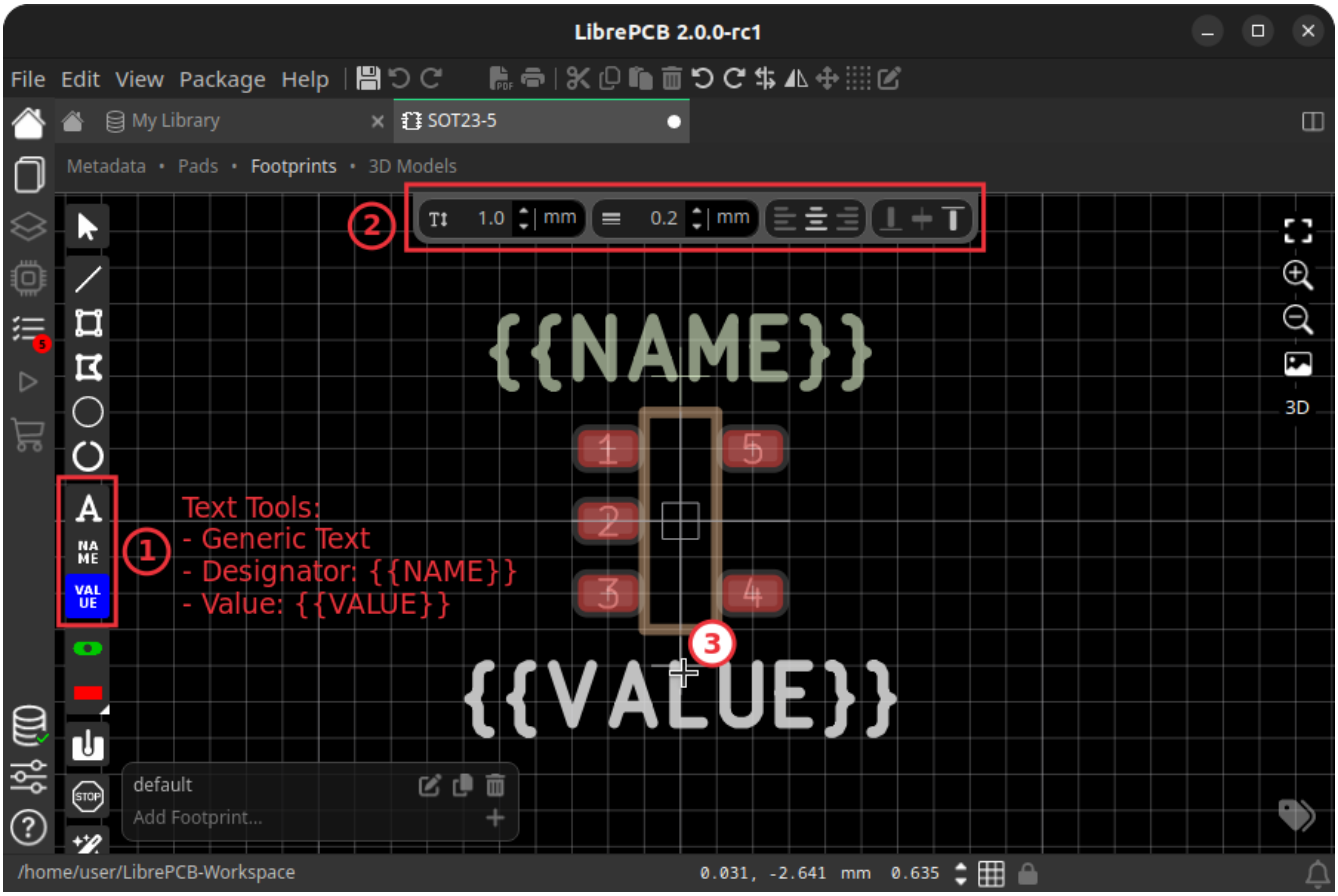
It's recommended to add at least two polygons:

- One **on the *Top Documentation* layer** to represent the body outline of the package. This layer will appear on assembly drawings, but not on the PCB silkscreen.
- One **on the *Top Legend* layer** to include a placement help which will be visible on the PCB silkscreen — **most notably pad-1 markings**.

To create highly functional, beautiful looking footprints, check out our package conventions.

Add Texts

Just like in the symbol, you should add `{{NAME}}` and a `{{VALUE}}` text objects:



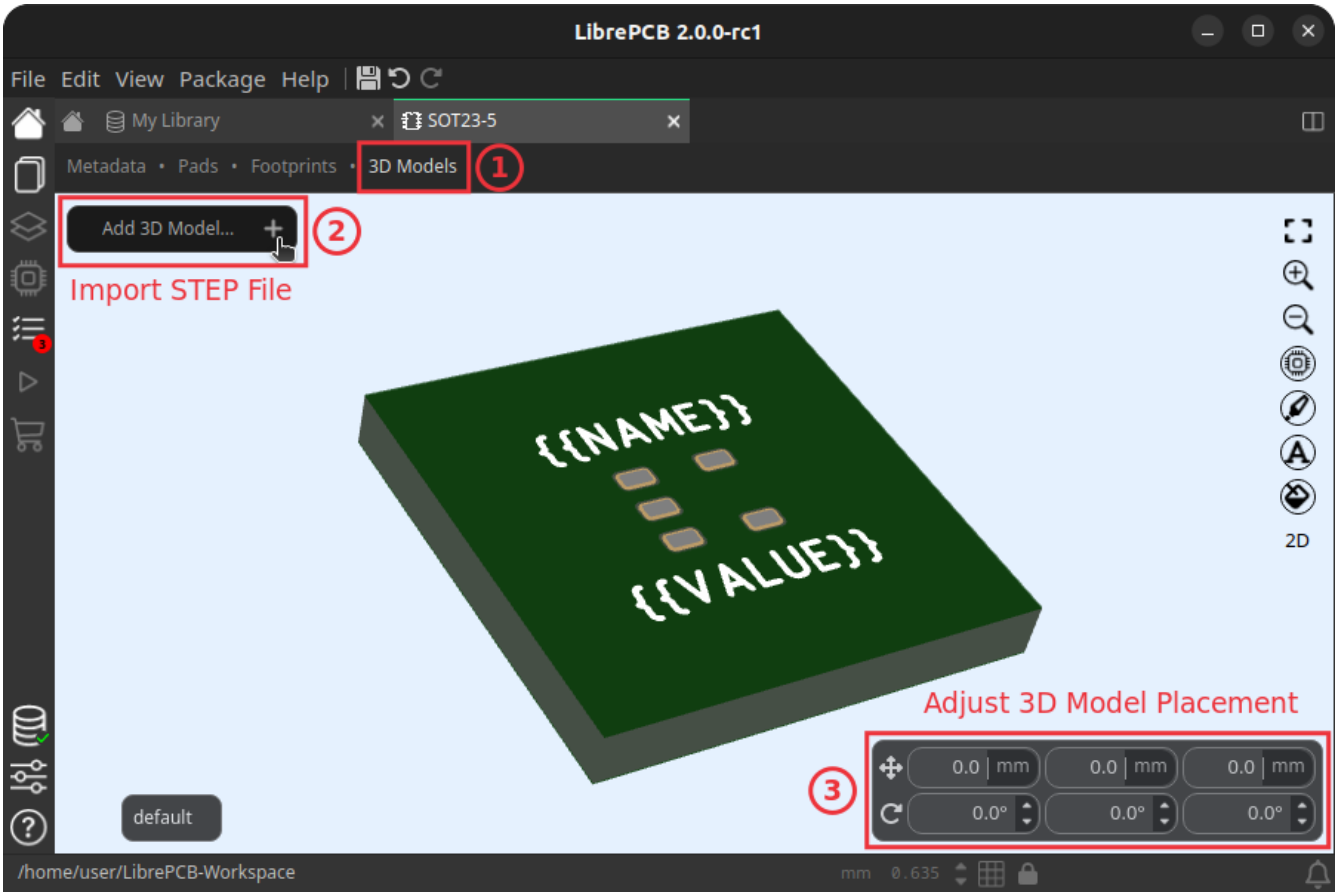
Add Non-Plated Holes

In case your package requires to drill non-plated holes into the PCB (for example to insert a screw), use the **[Add Hole]** tool and specify its diameter. However, for our *SOT23-5* package we don't need a hole.

That's all you need for a simple package! Now **save the package** to ensure the background library scan picks up the new package.

Add 3D Model

If you have a STEP file of the package, you can add it as a 3D model to the package. Switch to the tab **[3D Models]** (`Ctrl` + `3`) and click on the **[+]** button to import the STEP file (this may take a while):



If required, the position and rotation can be adjusted at the bottom right (note that `Return` needs to be pressed to accept new values).

Recommendations

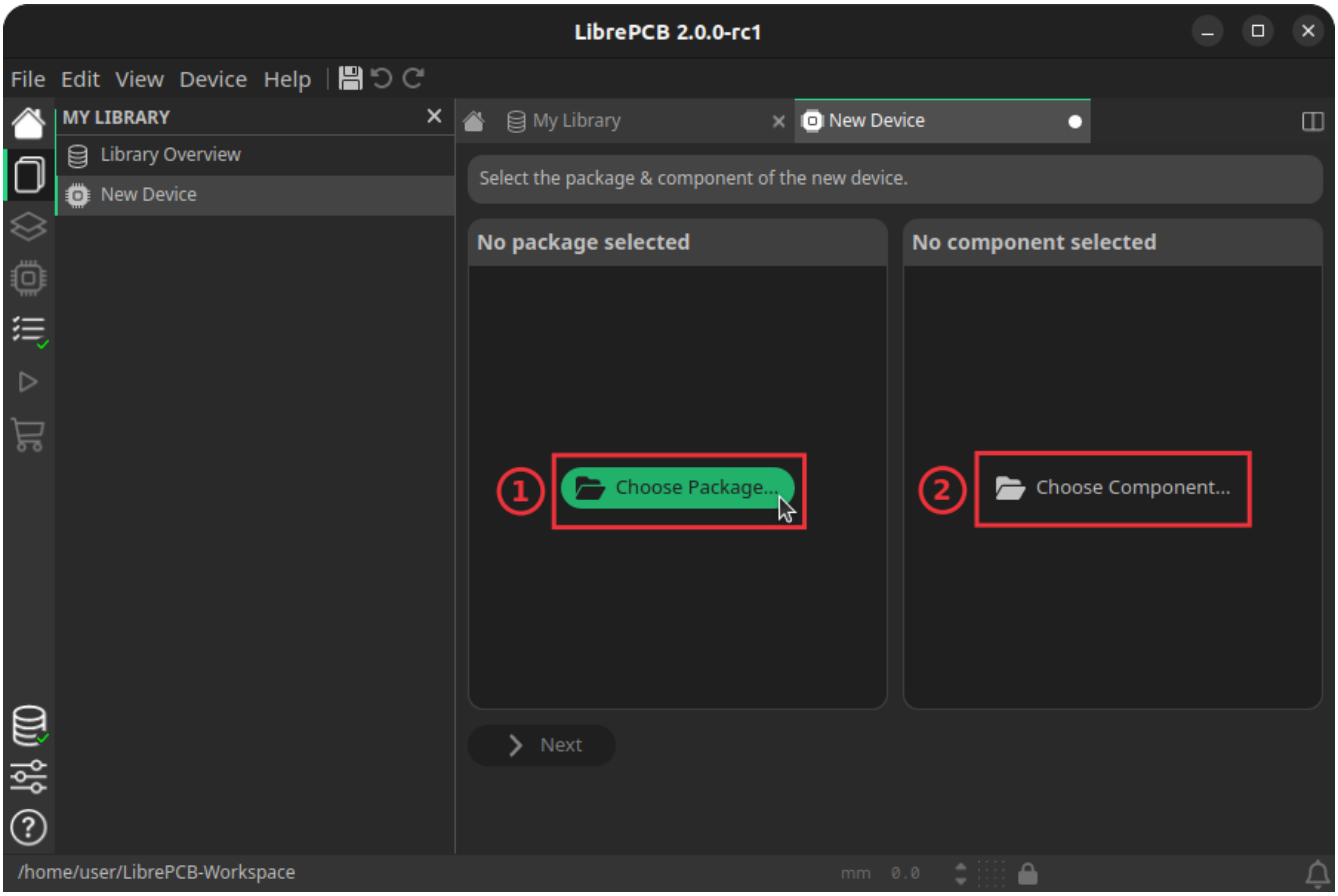
For details about how packages should be designed, please take a look at our package conventions. The most important rules are:

- 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 (or near to) the center of the package body.
- Footprints must always 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 flipping 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 package conventions for more information), typically just *1*, *2*, *3* etc. Only name pads according their electrical purpose (e.g. *Anode*) if the package is very specific for a particular purpose (like an LED).
- Pad 1 should always be at the top left.
- There should be text elements for `{{NAME}}` and `{{VALUE}}`.

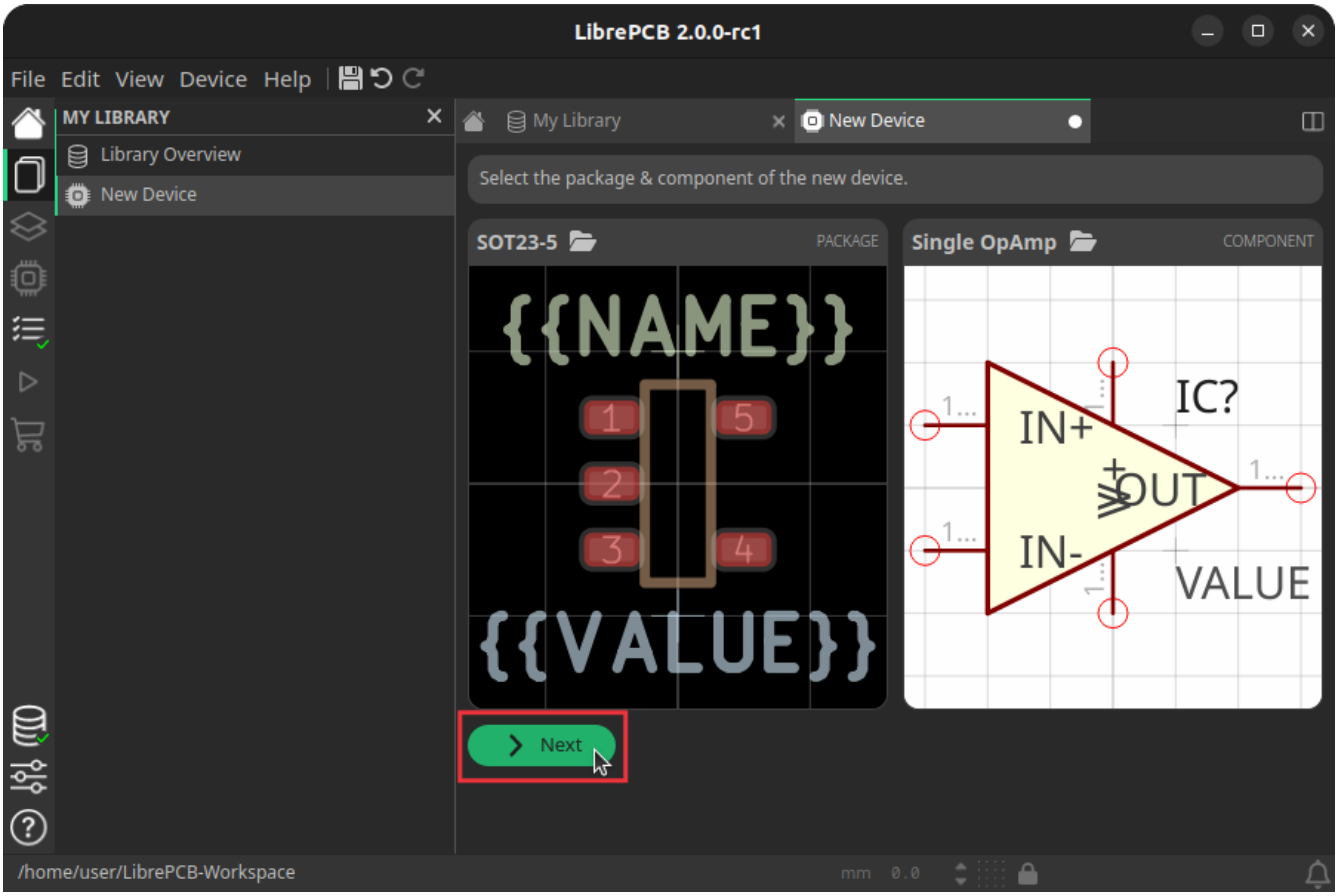
Device

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 and reusable for other OpAmps!

Again, click **[New Device]** in the library editor. Then you first have to choose the package and the component of the device to create:

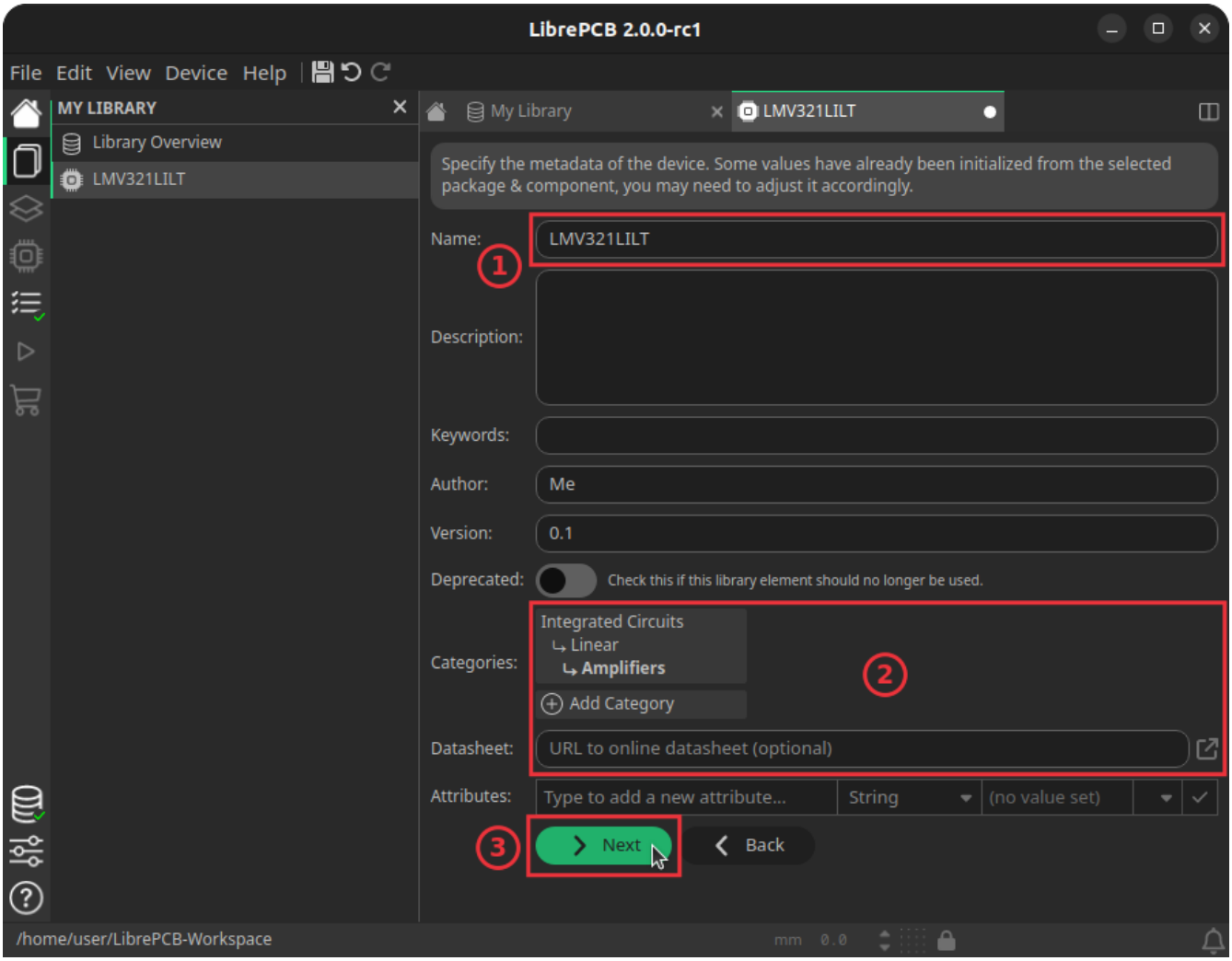


Choose the package and the component we just created in the previous steps. The result should look like this:



Specify Metadata

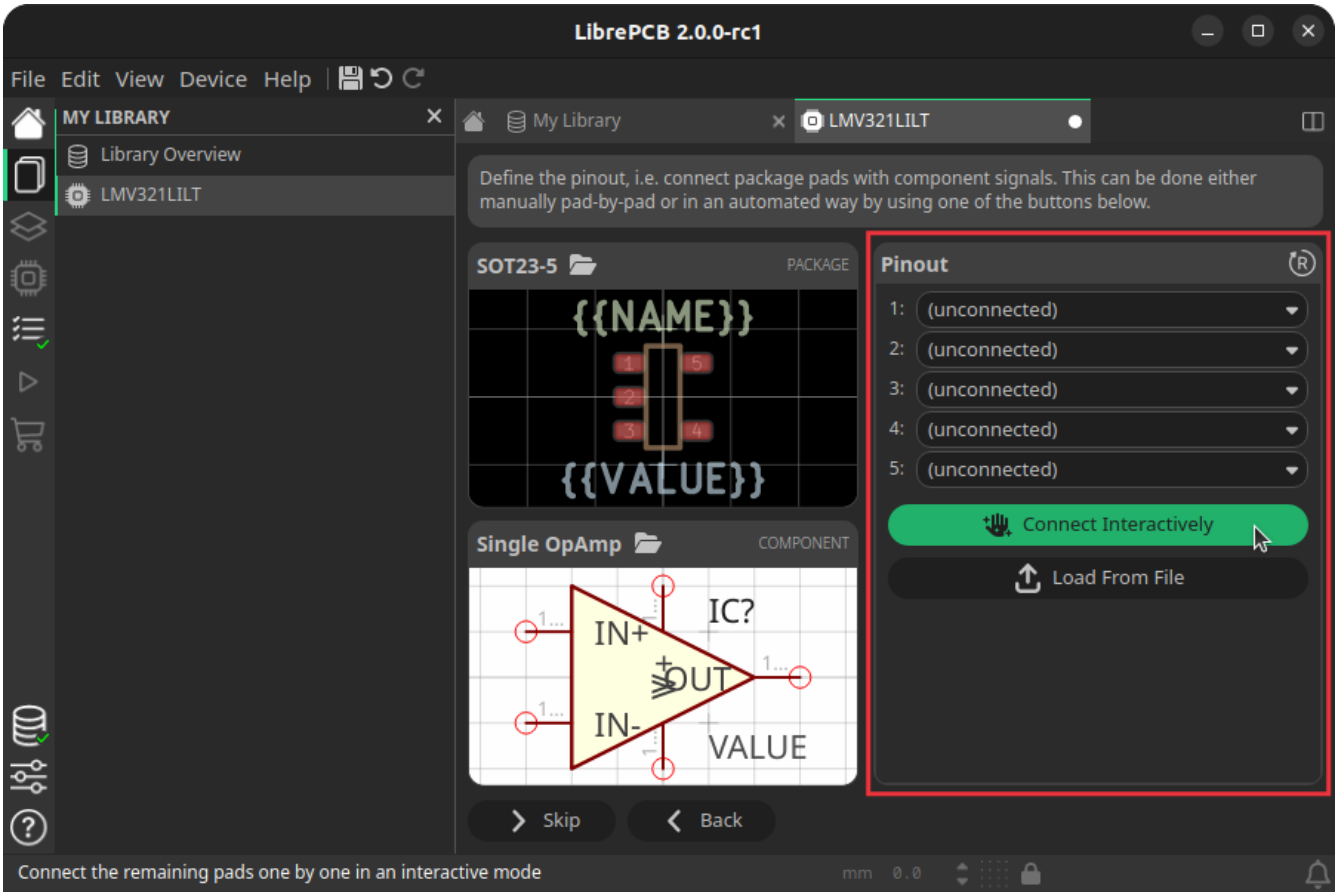
After clicking **[Next]**, specify the name of the new device. The category is now already taken from the component, so usually this is fine. If you have an URL to the datasheet, you may enter it also (optional):



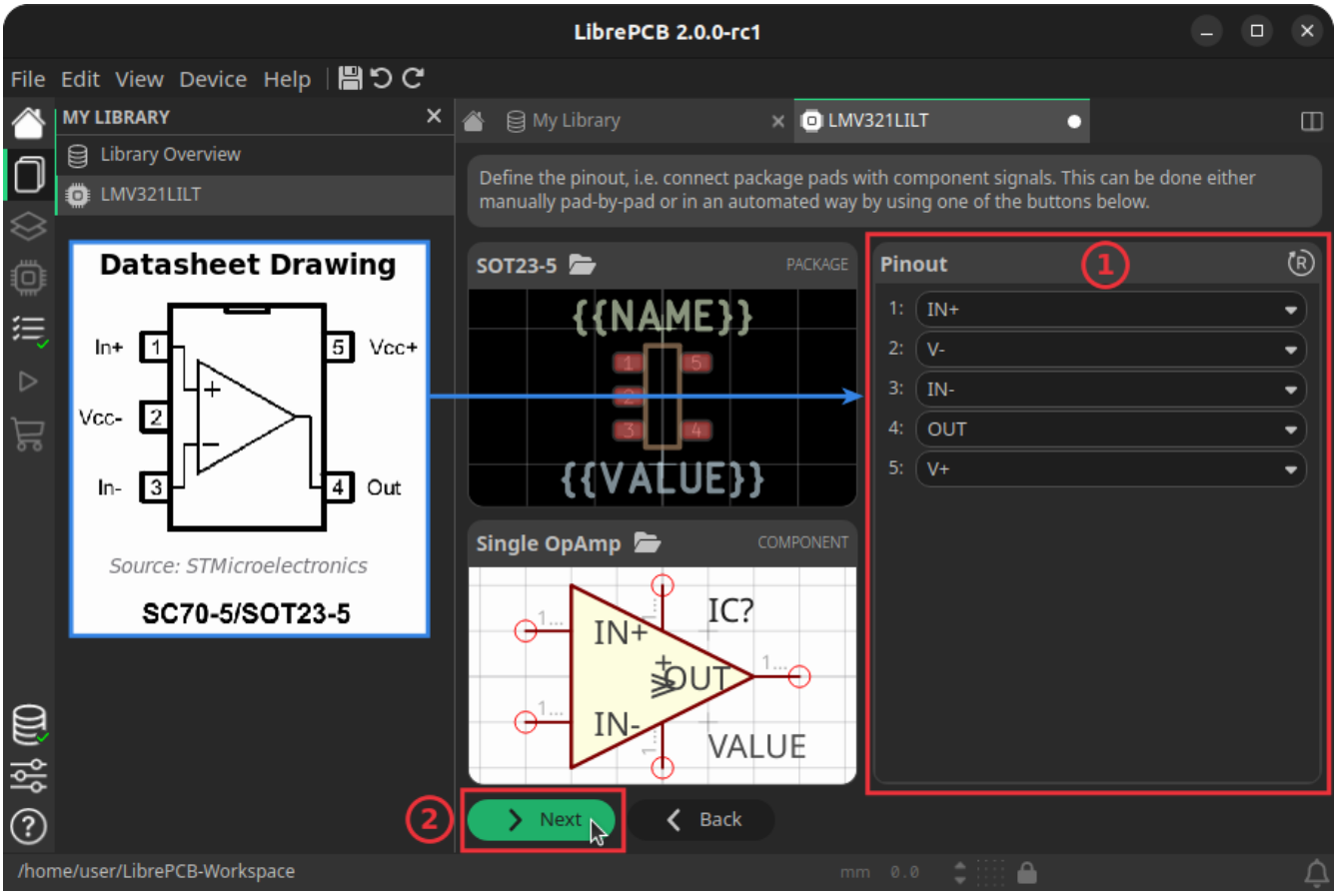
Then click on **[Next]**.

Define Pinout

Now you need to define the pinout, i.e. the connection between the package pads and the component signals. This needs to be don according to the pinout in the datasheet of LMV321LILT. There are various ways of defining the pinout, for example you can just use the dropdowns:



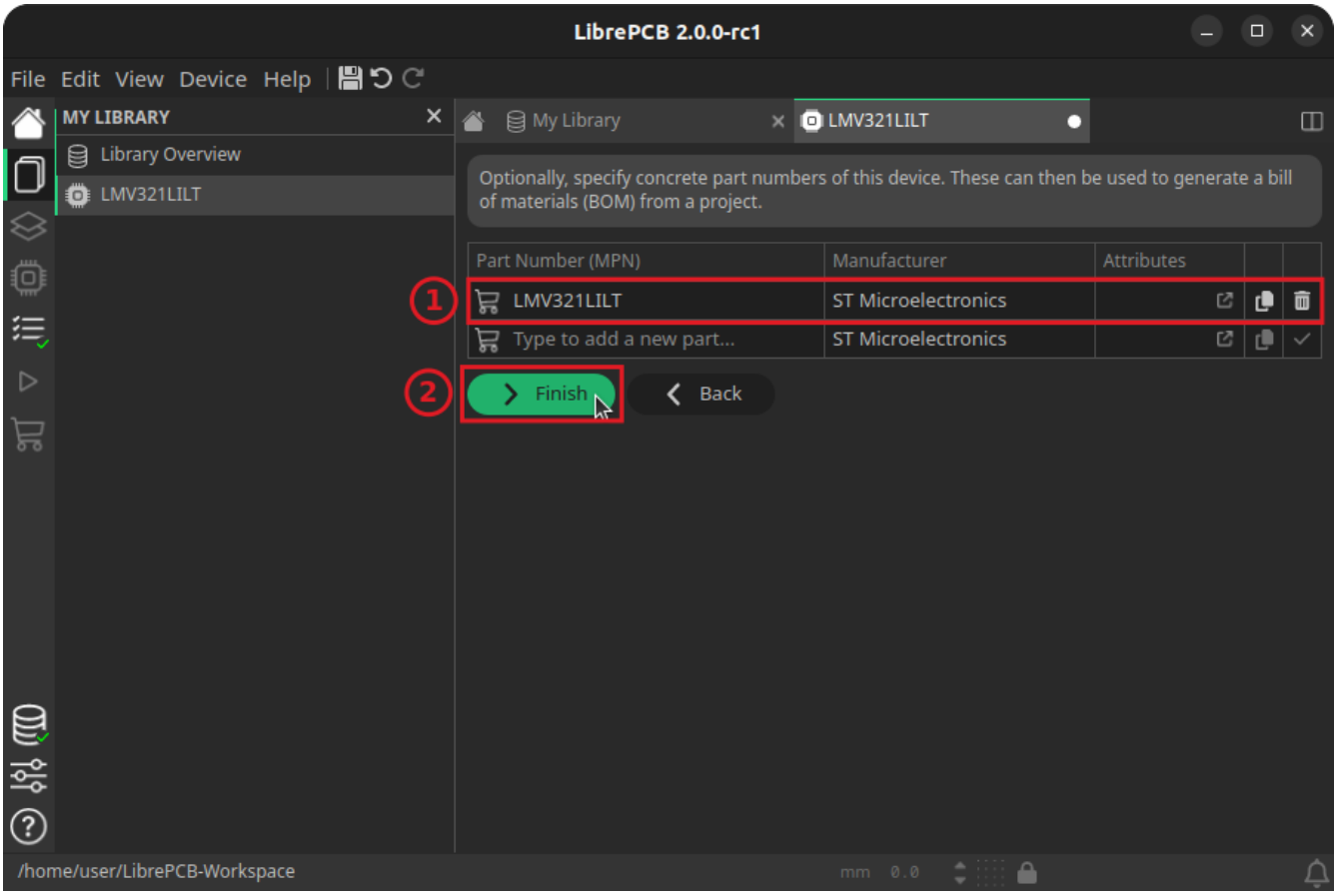
LibrePCB may also provide the options to define the pinout either automatically or interactively (which is very handy!). Once you have done that, the pinout should look like in the datasheet:



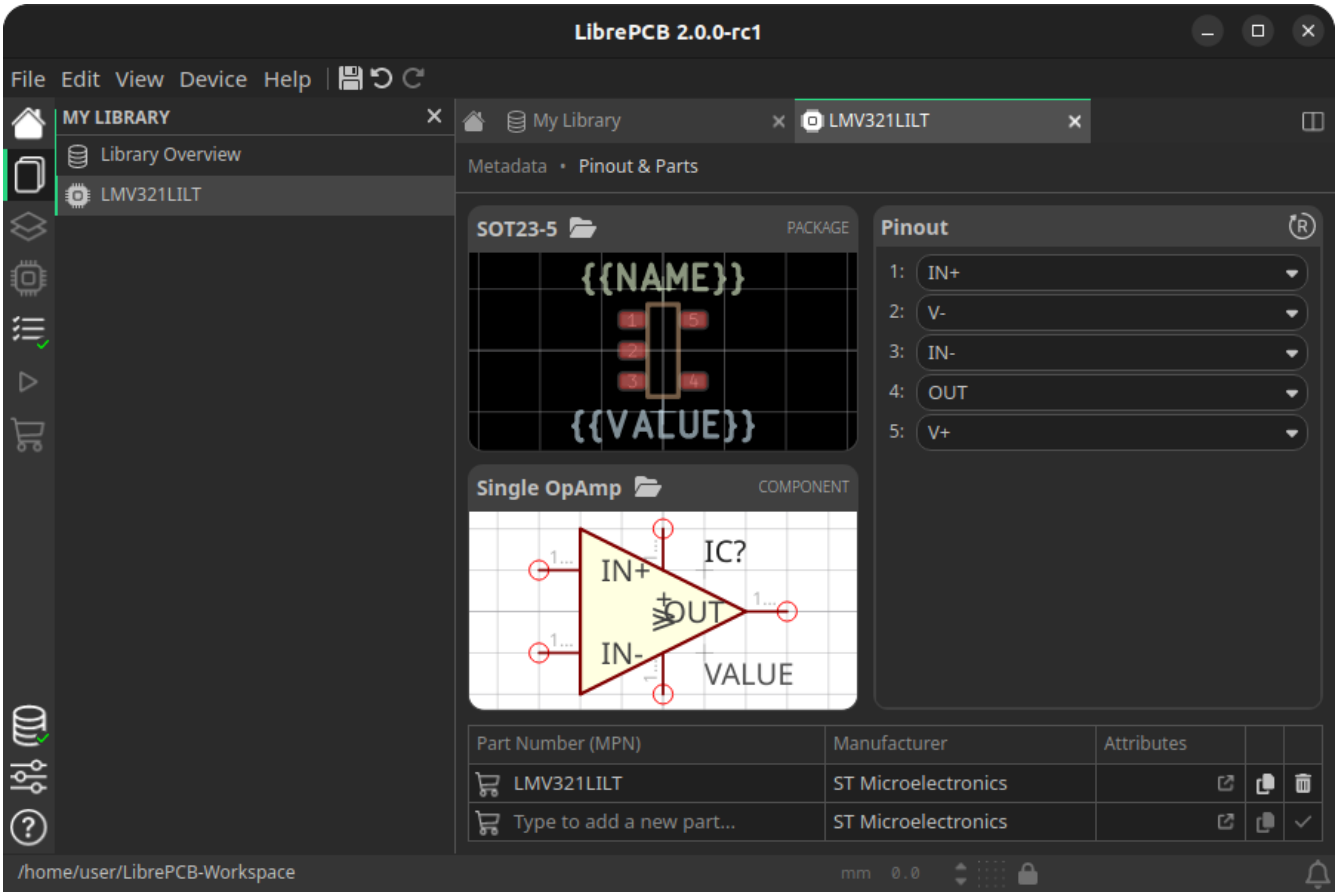
It’s very important that the pinout is correct. When you are sure, click **[Next]**.

Add Parts

In the last step, you can add the exact part numbers (MPNs) as listed on distributors. This step is optional, but it helps to get accurate bill of materials (BOMs) out of projects so it improves the ordering workflow. To add so, add as many part numbers which are valid for this specific device (often only one, but can be multiple) and also specify their manufacturer name:



Then click **[Finish]** and **save the device**. You can now review the created device and make any adjustments if needed:



And that's it! The LMV321LILT is now ready to be added to schematics and boards (after the library rescan has completed). And because the categories, symbol, component and package are very generic, you created not only one single device, but the basement for many more devices in the future! For any additional single-channel OpAmp (with an already available package), you need to create only a device which is now a matter of a minute.

[1] Manufacturer part number

[2] Bill of materials

[3] Electrical rule check

User Manual

This chapter provides detailed documentation about concepts and features of LibrePCB. In contrast to the Quickstart Tutorial, it does not need to be read in a particular order — just read the sections you're interested in to learn more about LibrePCB.

There are only very few things documented yet. Help us extending it on GitHub!

Layers

Layers are an integral part of every EDA software, thus it's important to know what they're intended for and how to use them. Note that all existing layers are described here, but the availability of layers within LibrePCB depends on the context. For example the layers *Top/Bottom Courtyards* are only available in the package editor, but not in the board editor since they make only sense within packages.

Schematic Layers

Sheet Frames

Intended to be used within symbols to draw schematic sheet frames (graphics and texts). Usually not used directly within schematics, but mainly in symbols. No special treatment by LibrePCB.

Documentation

General purpose layer for graphics and texts, e.g. to add some additional information to a schematic (like a formula, or a dawing of an external device). Not recommended (but possible) to be used within symbols. No special treatment by LibrePCB.

Comments

General purpose layer intended for comment annotations within schematics. Not recommended (but possible) to be used within symbols. No special treatment (yet) by LibrePCB.

Example: Text "Place C1 close to U1" next to the symbols of C1/U1

Guide

General purpose layer for graphics and texts, e.g. to draw boxes around parts of a schematic and give them a title. Not recommended (but possible) to be used within symbols. No special treatment by LibrePCB.

Outlines

Intended to draw outlines/content of symbols (e.g. a rectangle for an IC, or a thick line for a GND symbol). No special treatment by LibrePCB.

Example: Outline

Hidden Grab Areas

Used to define custom grab areas of symbols by drawing closed polygons or circles. These areas are not visible in the schematic editor, but the editor allows to grab symbols within these areas.

Only available in the symbol editor, not in the schematic editor.

Example: Grab Area

Names

Intended for the `{{NAME}}` text of symbols. Not used directly within schematics, only in symbols. No special treatment by LibrePCB.

Example: Text Elements

Values

Intended for the `{{VALUE}}` text of symbols. Not used directly within schematics, only in symbols. No special treatment by LibrePCB.

Example: Text Elements

Board Layers

Sheet Frames

Intended to be used within footprints to draw board sheet frames (graphics and texts). Usually not used directly within boards, but mainly in footprints. Ignored by the Gerber export, but usually contained when printing.

Board Outlines

Mandatory layer where a single, closed board outline polygon (or circle) must be drawn. The line represents the board edge after manufacturing. It's highly recommended to set its width to zero since it has no meaning. This layer will be exported to Gerber files and is usually contained when printing the board.

Example: Draw Board Outline

In the board editor layers dock, this layer is named *Outlines* and is coupled with the *Board Cutouts* layer.

Board Cutouts

Any non-plated board cut-outs (millings) more complex than slotted holes must be drawn on this layer as polygons or circles. The line represents the board edge after manufacturing. It's highly recommended to set its width to zero since it has no meaning. This layer will be exported to Gerber files (into the same file as the *Board Outlines* layer) and is usually contained when printing the board.

Example: Draw Board Outline

In the board editor layers dock, this layer is named *Outlines* and is coupled with the *Board Outlines* layer.

Plated Board Cutouts

Any plated board cut-outs (millings) more complex than slotted pads must be drawn on this

layer as polygons or circles. The line represents the board edge after manufacturing. It's highly recommended to set its width to zero since it has no meaning. This layer will be exported to Gerber files (into the same file as the *Board Outlines* layer) and is usually contained when printing the board.

Measures

General purpose layer for graphics and texts, e.g. to add manual measurements (e.g. the PCB size) to the board. Not recommended (but possible) to be used within footprints. Ignored by the Gerber export, but might be contained when printing.

Alignment

Intended to add alignment helper drawings to footprints, mainly just straight lines with zero width. For example in an edge-mounted device, the exact location of the board edge could be drawn on this layer as a help/reference for the board designer to correctly place the device in relation to the board edge. Ignored by the Gerber export, and usually not contained when printing.

Documentation

General purpose layer for graphics and texts, e.g. to add some additional information or drawings to a board. Not recommended (but possible) to be used within footprints. Ignored by the Gerber export, but might be contained when printing.

Comments

General purpose layer intended for comment annotations within boards. Not recommended (but possible) to be used within footprints. Ignored by the Gerber export, but might be contained when printing.

Guide

General purpose layer for graphics and texts, e.g. to draw boxes around parts of a board and give them a title. Not recommended (but possible) to be used within footprints. Ignored by the Gerber export, but might be contained when printing.

Top/Bottom Names

Intended for the `{{NAME}}` text of footprints. Not used directly within boards, only in footprints. Usually printed on silkscreen (by the Gerber export), depending on the board setup configuration.

Example: Text Elements

Top/Bottom Values

Intended for the `{{VALUE}}` text of footprints. Not used directly within boards, only in footprints. Might be printed on silkscreen (by the Gerber export), depending on the board setup configuration.

Example: Text Elements

Top/Bottom Legend

Intended for any drawings to appear on silkscreen, e.g. device placement/orientation lines, pin-1

dots and custom text. To be used within footprints or directly within boards. Typical line width is 0.2 mm, recommended minimum is 0.1 mm. Usually printed on silkscreen (by the Gerber export), depending on the board setup configuration.

Example: Legend Layer

Top/Bottom Documentation

Intended for drawings to represent devices in a nice way, including body outlines, leads and polarity/pin-1 markings. Basically as a 2D projection of the 3D model to somehow see the packages in 2D views, for example to export a nice looking assembly plan. Ignored by the Gerber export, but usually contained when printing.

Example: Documentation Layer

Top/Bottom Package Outlines

Intended for footprints to draw the exact mechanical outlines of the device to be assembled. To be drawn with a zero-width polygon or circle. Used by the DRC to detect and warn about overlapping devices, or devices placed within the courtyard of another device. This DRC check doesn't work if no package outline is drawn.

Top/Bottom Courtyard

Intended for footprints to draw the courtyard (clearance) of the device to be assembled, typically just the package outlines with a small offset (e.g. 0.2 mm). To be drawn with a zero-width polygon or circle. Used by the DRC to detect and warn about devices placed within the courtyard of another device. This DRC check doesn't work if no courtyard is drawn.

Top/Bottom Hidden Grab Areas

Used to define custom grab areas of footprints by drawing closed polygons or circles. These areas are not visible in the board editor, but the editor allows to grab devices within these areas. Only available in the footprint editor, not in the board editor.

Top/Bottom Stop Mask

Used to add solder resist openings, i.e. areas on the PCB where no solder resist shall be applied. So in contrast to most other layers, this layer has inverted polarity. Note that LibrePCB adds content on this layer automatically where necessary (pads, holes etc.), so manual usage of this layer is generally not needed. But it still allows to add custom solder resist openings. Any content on this layer is exported to the corresponding Gerber files.

Top/Bottom Solder Paste

Used to add stencil openings, i.e. areas on the PCB where solder paste is applied for reflow soldering of THT devices. Note that LibrePCB adds content on this layer automatically for SMT pads, so manual usage of this layer is generally not needed. But it still allows to add custom solder paste areas. Any content on this layer is exported to the corresponding Gerber files.

Top/Bottom Finish

Not supported yet (ignored in the Gerber export)!

Top/Bottom Glue

Used to draw areas as polygons or circles where glue should be applied on the board to hold components during overhead soldering. This requires to add a dedicated output job for exporting, i.e. by default the glue mask is not exported. But in almost every case, glue masks are not needed anyway.

Top/Inner/Bottom Copper

Should be pretty clear Inner layers are numbered from top to bottom, i.e. *Inner Copper 1* is just below *Top Copper* and on a 6-layer PCB *Inner Copper 4* is just above *Bottom Copper*.

Custom Layers

To allow sharing symbols and footprints between users, it's crucial that the purpose of every layer is identical for each user. Therefore LibrePCB does not allow to add custom, user-defined layers.

If the built-in layers are not sufficient for you, please let us know.

Licenses

When creating a new project, LibrePCB allows to specify a license for it. This chapter gives an overview about the available licenses to help you deciding which makes most sense for your project.

First of all, choosing a license is not mandatory. Especially if you don't intend to make the project public, it's totally fine to skip the license selection. However, for projects made public it's highly recommended to specify a license to let other people know what they are allowed or not allowed to do with your project. Theoretically you could write your own license text, but it's recommended to choose one of the already existing, well-known licenses.

Please always read the full, original license text instead of relying on the information on this page. First, this page presents licenses in a very simplified form without all their details. Second, this documentation is not approved by a lawyer so it may not be correct. We're not responsible for any implications caused by incomplete or wrong information on this page.

Available Licenses

The following licenses are provided by LibrePCB (depending on the installed version):

License	Permissions	Conditions	Limitations
CC0-1.0 Summary	Distribution Modification Commercial use		No liability No warranty
CC-BY-4.0 Summary	Distribution Modification Commercial use	License and Copyright Notice State Changes	No liability No warranty

License	Permissions	Conditions	Limitations
CC-BY-SA-4.0 Summary	✓ Distribution ✓ Modification ✓ Commercial use	✓ License and Copyright Notice ✓ State Changes ✓ Same License	✗ No liability ✗ No warranty
CC-BY-NC-4.0	✓ Distribution ✓ Modification	✓ License and Copyright Notice ✓ State Changes	✗ No liability ✗ No warranty ✗ Non-commercial only
CC-BY-NC-SA-4.0	✓ Distribution ✓ Modification	✓ License and Copyright Notice ✓ State Changes ✓ Same License	✗ No liability ✗ No warranty ✗ Non-commercial only
CC-BY-NC-ND-4.0	✓ Distribution ✓ (Modification)	✓ License and Copyright Notice ✓ State Changes	✗ No liability ✗ No warranty ✗ No derivatives ✗ Non-commercial only
CC-BY-ND-4.0	✓ Distribution ✓ (Modification) ✓ Commercial use	✓ License and Copyright Notice ✓ State Changes	✗ No liability ✗ No warranty ✗ No derivatives
TAPR-OHL-1.0	✓ Distribution ✓ Modification ✓ Commercial use	✓ License and Copyright Notice ✓ State Changes ✓ Same License ✓ Notify upstream developers	✗ No liability ✗ No warranty
CERN-OHL-P-2.0 Summary FAQ	✓ Distribution ✓ Modification ✓ Commercial use	✓ License and Copyright Notice ✓ State Changes	✗ No liability ✗ No warranty
CERN-OHL-W-2.0 Summary FAQ	✓ Distribution ✓ Modification ✓ Commercial use	✓ License and Copyright Notice ✓ State Changes ✓ Same License (relaxed) ✓ Disclose source	✗ No liability ✗ No warranty
CERN-OHL-S-2.0 Summary FAQ	✓ Distribution ✓ Modification ✓ Commercial use	✓ License and Copyright Notice ✓ State Changes ✓ Same License (strong) ✓ Disclose source	✗ No liability ✗ No warranty

License and Copyright Notice

A copy of the license and copyright notice must be included with the licensed material.

State Changes

Modifications made to the licensed material must be documented.

Same License

If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

Other Licenses

If you like to use a license not available in LibrePCB, or if you're not sure yet and want to decide later, just skip the license selection when creating the project. Then add it manually afterwards as follows:

1. Close the project in LibrePCB.
2. In the root directory of the project, add a file named `LICENSE.txt` containing the license terms.
3. Open `README.md` in a text editor and replace this line:

   ```
   No license set.
   ```

   by this one:

   ```
   See [LICENSE.txt](LICENSE.txt).
   ```

Or if you want to switch to a different license after you created a new project, just replace the file `LICENSE.txt` manually. LibrePCB itself does not provide a tool to change the license of an existing project.

Additional Actions

Note that some licenses may require to perform additional actions to correctly apply those licenses to a project. This applies mainly to the OHL licenses, but to be sure please check the license texts.

Please help us documenting the additional actions on GitHub!

Recommendation

Of course choosing a license is a personal decision and any recommendation from our side will be subjective. But generally, if you simply don't care what people will do with your project, or you want to give them maximum freedom (without any liability or warranty from your side), we think **CC0-1.0** is a great choice as this will allow *everyone* to use your project for *any* purpose. If you want to be quite restrictive and don't want your project to be used for closed-source products, **CERN-OHL-S-2.0** shouldn't be a bad choice.

License of Libraries

Not only projects, but also libraries can be released under a certain license. However, we think public libraries should be released exclusively under the **CC0-1.0** license. Any other license would make it too complicated for other people to create projects using these libraries, fulfilling all license terms of any used library.

Thus when creating a new library, CC0-1.0 is the only available option and all official libraries are published under this license so you can do with these libraries whatever you want, whether for commercial or non-commercial use.

UUID References

This is a somewhat advanced, technical section. It is intended to give power users more knowledge about the internal function of LibrePCB, as this knowledge can be useful to increase productivity and to reduce the likelihood of running into troubles caused by broken UUID references.

More technical details about references between library elements are available here.

Purpose of References

In every EDA tool, libraries and projects are full of cross-references between different objects. For example, when attaching a wire to a pin in the schematic, the wire has to store a reference to that particular pin to successfully restore that connection when you open the schematic the next time.

Many EDA tools use the object's name as reference. For example when you connect a wire of the *GND* net to the pin named *IN+* of the component with the designator *U3*, the schematic editor stores a reference like `U3:IN+` for the attached wire in the schematic file. When opening the schematic the next time, the editor searches for a component with the designator *U3* in the project library, and then for a pin named *IN+* within this component to restore the wire connection.

This system works, but it comes with a big disadvantage. Whenever you rename any referenced object (e.g. renaming pin *IN+* to *+*), all the references to this object will break, causing broken library elements or projects. So in fact it is not allowed to ever rename or move things, or you will run into troubles. A typical symptom is that the editor throws errors like "Component XYZ not found", for example after updating libraries and then opening an older project.

References in LibrePCB

LibrePCB follows a different approach. Instead of referencing objects by their name, it assigns a random identifier to every newly created object, and uses this for all the references. In particular, it uses universally unique identifiers (UUIDs) which consist of 36 characters and look like *3da6ef2c-03ff-4ed1-a8b8-c8acf2ed0291*.

For example, when you add a pin to a symbol, the editor automatically generates and assigns a random UUID to the pin. You may give the pin a human-readable name like *IN+*, but the editor gives it a UUID like *cc3af4a4-cc26-4548-a026-468e1d689ab3*. This UUID is typically not user-facing, so usually you won't ever see it. But behind the scenes, LibrePCB uses this UUID for any references. So a wire in the schematic which is attached to a pin won't reference the pin like `U3:IN+` in other tools, but like `7b8fef14-64f9-4502-8094-7e2c1c0b6e9a:cc3af4a4-cc26-4548-a026-468e1d689ab3` where the first UUID refers to the component (*U3*) and the second UUID refers to a pin of it (*IN+*).

Now whenever you rename objects like a symbol pin, it will only affect the human-readable name, but not its UUID. The UUID of an existing object will never change, so all the references to it will stay intact.

Drawbacks of UUIDs

This system comes with the advantage of keeping references intact even when renaming objects, thus avoiding broken library elements or projects after renaming or moving things. However, it also comes with some drawbacks you should be aware of.

Names Are Non-Functional

As LibrePCB itself only considers the UUIDs for making references, in fact it does not care about the name of objects. Names are only for *you*, not for LibrePCB.

Now when you make substantial renames to existing objects, like renaming pin *IN+* to *IN-* because you want to swap those two pins of an OpAmp, this will change the meaning of the symbol pins. The pin which was *IN+* before is now *IN-* and vice versa. However, since the underlying UUIDs were not changed, for LibrePCB the pin function has not changed either. So if a wire of the net *VBAT* was attached to pin *IN+* before the rename, that wire will remain attached to the same pin after the rename, even though it's now called *IN-*.

So, keep this in mind and be careful when changing the meaning of existing pins. In this particular example, it would be better to swap the positions of the two OpAmp pins instead of swapping their names, since swapping the positions will keep the relation between UUID and pin function.

Copy != Duplicate

The LibrePCB library editor allows you to copy existing library elements in two different ways, and it is crucial to know the difference of those operations to understand in which situation you need which of those.

Copy

The copy operation is provided in the library overview tab, called *Copy to Other Library*. This operation will **copy an existing library element 1:1 into another library, keeping all its UUIDs**. In fact, this operation just copies the files from one library into another. As the UUIDs are preserved in this operation, in fact **the copy still represents the same object** — it is *not* a new, independent symbol.

For example when you copy a symbol named *OpAmp* into another library, and you rename the new copy to *Comparator*, in fact you won't have two symbols afterwards! Since both symbols still have the same UUID, LibrePCB will pick only one of them (the one with the higher version number) and considers the other as an (outdated) duplicate. That duplicate won't be listed anywhere, so it feels like it is lost.

The copy operation is only intended to **override** library elements from read-only remote libraries with your own modifications. If you're not happy with a library element from our official libraries, you can copy it into your local library, modify it, give it a higher version number and save it. LibrePCB will then use *your* library element in place of the original library element.

Except from this particular use-case, the copy operation is usually not the right operation. If used wrongly, it can lead to surprising behavior.

Save To

This operation is provided in the *Project Library Manager* and is called *Copy elements to local library*. It is exactly the same as the *Copy* operation, just for copying library elements from a project into a library rather than between two libraries. It allows to add library elements from a project to your local library (keeping their UUIDs), so you can modify the elements or reuse them in other projects.

Duplicate

This operation is provided in the library overview tab to duplicate a library element within the opened library. **In contrast to the *Copy* operation, the *Duplicate* operation generates new UUIDs for the duplicated library element and all its contained objects.** It is *not* just a file copy operation.

This is the right operation to create a new library element which is similar to an existing element, so you don't have to start from scratch. For example when you want to create a symbol called *Comparator*, you can duplicate the existing *OpAmp* symbol, rename it to *Comparator*, adjust its metadata, graphics and pins as desired, and save it. The result will be a new, independent *Comparator* symbol which has no relation to *OpAmp* anymore. All UUIDs are re-generated, so it is like a new symbol created manually from scratch.

Usually, this is the operation you are looking for.

Breaking Changes

As you may understand by now, the data structures of libraries and projects are full of references. A project consists of hundreds or thousands of wires, traces, nets, pins and pads which are all linked together by referencing each other. This is true for every EDA tool, no matter if references are made by names, UUIDs or anything else. References are everywhere.

Given this fact, it is also clear that there is always the risk of *broken* references. Imagine you add a symbol to a schematic and connect wires to its pins. Now you realize the symbol is not exactly correct as it has a pin which the real part doesn't have. So you open the symbol in the library editor, delete that pin, and save the symbol. Then you update the project's library (with the *Project Library Manager* in LibrePCB) to update the symbol in your schematic with the new one. But you have already attached a wire to the pin which no longer exists. This means that the wire holds a reference to a pin that doesn't exist (anymore), a case which we call *broken reference*.

Some tools may silently ignore such broken references and try to revert to a valid state. In this particular case, they might silently disconnect the wire from the pin, leaving behind a "dead end" wire.

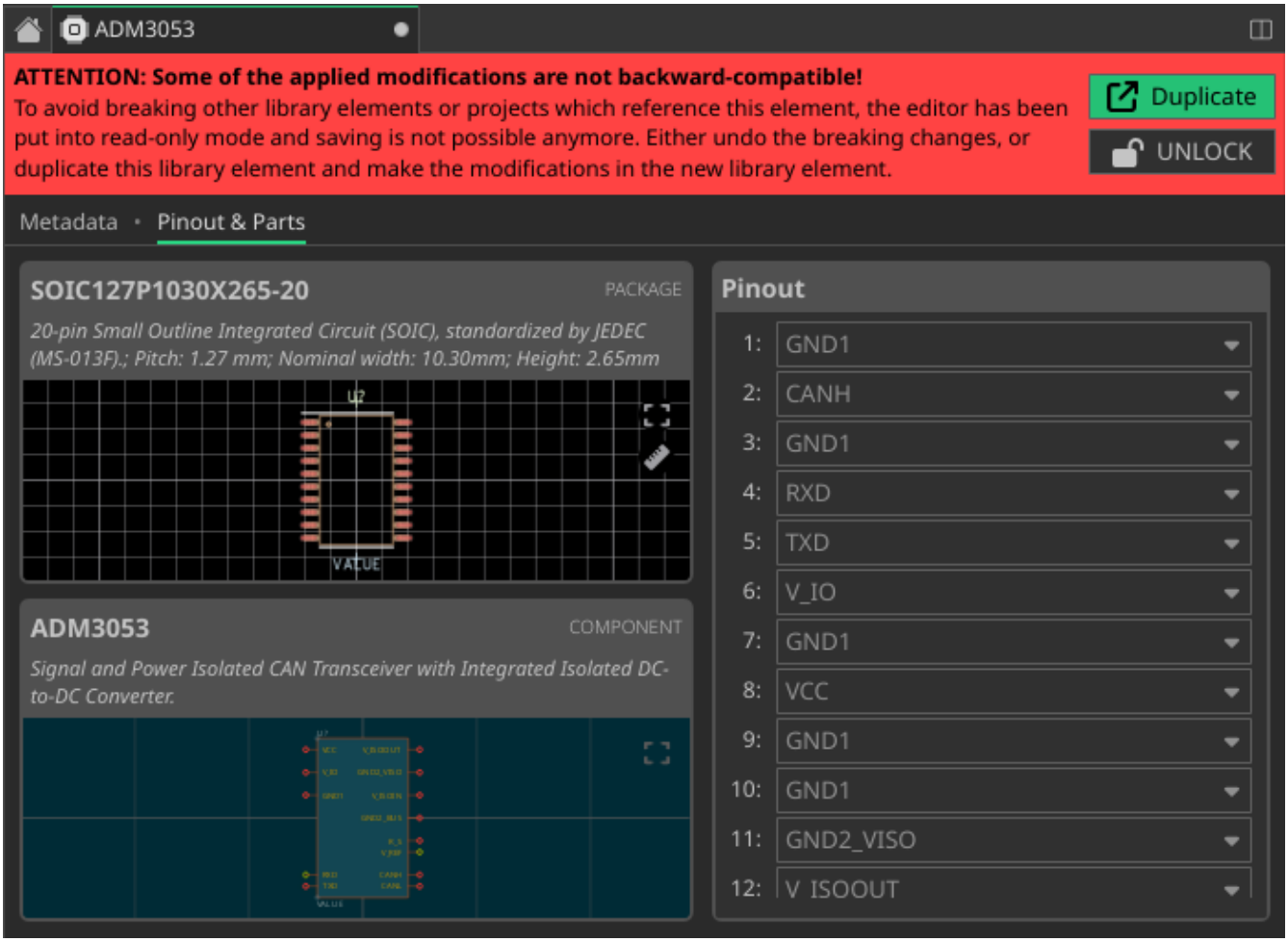
Other tools, like LibrePCB, are more strict about broken references. Instead of silently ignoring such (potentially unintended, dangerous) invalid situations, the tool throws an error to inform you about the problem. This may lead to a situation where you cannot open the project anymore until the problem is fixed (e.g. the deletion of the pin is reverted).

To avoid headaches and wasted time, such situation should be avoided to happen at all. In other words, it is highly recommended to never make so-called *breaking changes* to existing library

elements. If you have a situation like the described superfluous symbol pin, it is better to create a new library element (using the *Duplicate* operation) and make the breaking changes there. Then just replace the wrong symbol in the schematics with the new one. Using this workflow, there will never be any broken references, causing no errors or other troubles.

Making Breaking Changes

LibrePCB tries to prevent you from (accidentally) making breaking changes. Generally you don't need to worry about breaking changes in library elements — whenever you made a breaking change to an existing library element, LibrePCB will warn you with this banner:



As soon as this red banner shows up, **the editor is put into read-only mode and saving is not possible anymore**. Now you have three options:

Undo

If the breaking change was not intended, you can just undo the change (`Ctrl` + `Z`) and the editor will leave the read-only mode automatically.

Duplicate

If you instead want to make the change in a new library element rather than breaking the opened element (as described/recommended in the Duplicate section above), just click the **Duplicate** button in the banner and make the changes in the newly opened tab. LibrePCB will then automatically undo the breaking changes in the existing library element to leave the read-only mode.

Unlock

The third option is the only dangerous one — by pressing & holding the **Unlock** button you can leave the read-only mode, so it will be possible to actually save the breaking changes to disk.

This operation is regularly the reason why inexperienced LibrePCB users end up in troubles and ask us for help, complain about errors, or even blaming us about bugs in the libraries or the tool — bugs which don't exist!

So: **Only use the unlock feature if you have fully understood how our referencing system works, what breaking changes are, and what consequences they have.**

Simply put, there is only one case where it is safe to unlock and make breaking changes to an existing library element: **If that particular library element is not referenced yet from anywhere else.** For example a symbol is not referenced from anywhere else if it has never been loaded as a gate into a component. If you create a component using that particular symbol, the symbol is considered as referenced — even though you never added it to a project.

A typical case for using the unlock feature is to continue work on an unfinished library element. For example you started to create a new symbol today, but you have to shut down the computer before it is finished. Tomorrow you open the symbol again to continue working on it. In this case, LibrePCB will prevent you from making breaking changes, and you have to unlock it to continue work. It wouldn't be reasonable in this case to duplicate the unfinished library element — just unlock it.

For unlocking, you need to press and hold the button for a moment. The required holding time depends on the age of the library element to unlock. For the typical use-case of continuing work on a library element you just created the day before, only a short hold time is needed. But to unlock a library element that has existed for a year, you need to hold the unlock button for several seconds because the older the library element is, the higher the probability that it has already been referenced from other objects, and therefore the higher the risk of troubles.

So if a library element needs a long time to unlock, you should think twice about if you are really sure to make breaking changes to it.

Please note that there is also a case where the "breaking changes"-banner doesn't show up, even if you made a breaking change: When you create a new library element, the banner will be suppressed as long as the tab of the new element stays open. This is the case because for new library elements it is normal that you make breaking changes all the time (e.g. every single pin you add to a new symbol is a breaking change).

To not annoy you with the banner during the normal workflow of creating a new library element, it will be suppressed. However, this can lead to a dangerous situation: Imagine you create a new library element, save it but keep the tab open,

then use it in other library elements or projects, and then make further (possibly breaking) changes in the still opened tab of the new library element. In this case, you may have just created a hell of a mess but LibrePCB didn't warn about it.

Recommendations

Here are some simple recommendations which help to avoid running into troubles due to broken references:

- **Fully finish library elements before starting to use them (*right first time*)!** Do not just create a quick&dirty package & corresponding device, add them to a project, and then iterate those library elements many times to get them finished & right. It is no problem to correct smaller things later (silkscreen drawings, pad sizes, MPNs, datasheet URLs etc.), but **the fundamental things (number of pads, package & component of a device, pinout etc.) need to be done right from the beginning on.** You can save yourself a lot of headaches (and time) if you seriously finish library elements **one after the other** instead of just hacking around with trial & error. LibrePCB is not the tool to hack around.
- After you created a new library element, **close the editor tab before using the new library element anywhere else.** This will enable the detection of breaking changes for the new library element — without closing the tab, the "breaking changes"-banner will be suppressed, and you won't notice when you are about to create troubles.
- When the "breaking changes"-banner shows up in the library editor, **do not click on Unlock unless you are 100% sure that the library element has not been used by another library element or a project yet.** If you are not 100% sure about this, use the **Duplicate** operation.
- When creating new library elements by copying an existing library element, **use the *duplicate* operation, not the *copy* operation**. Generally, the *copy* operation is almost never what you are looking for.
- **Never copy/move/modify files on the file system directly.** Always perform operations in LibrePCB, as LibrePCB will take care of UUID handling then. As soon as you edit files on the file system directly, there is no safety net anymore and you can run easily into a hell of a mess.

Project Editor

Assembly Data

This chapter explains how to specify assembly data like MPNs[1] in your project to get an accurate BOM[2] ready to order the correct parts. LibrePCB supports many different use-cases which will be explained below.

Assembly data is a rather advanced topic, and often not that relevant for simple hobby projects which are assembled by hand. So for your first steps with LibrePCB, you could simply ignore this whole topic. LibrePCB will let you generate a BOM for your project anyway, it might just be less accurate (e.g. missing exact MPNs).

Assembly Variants

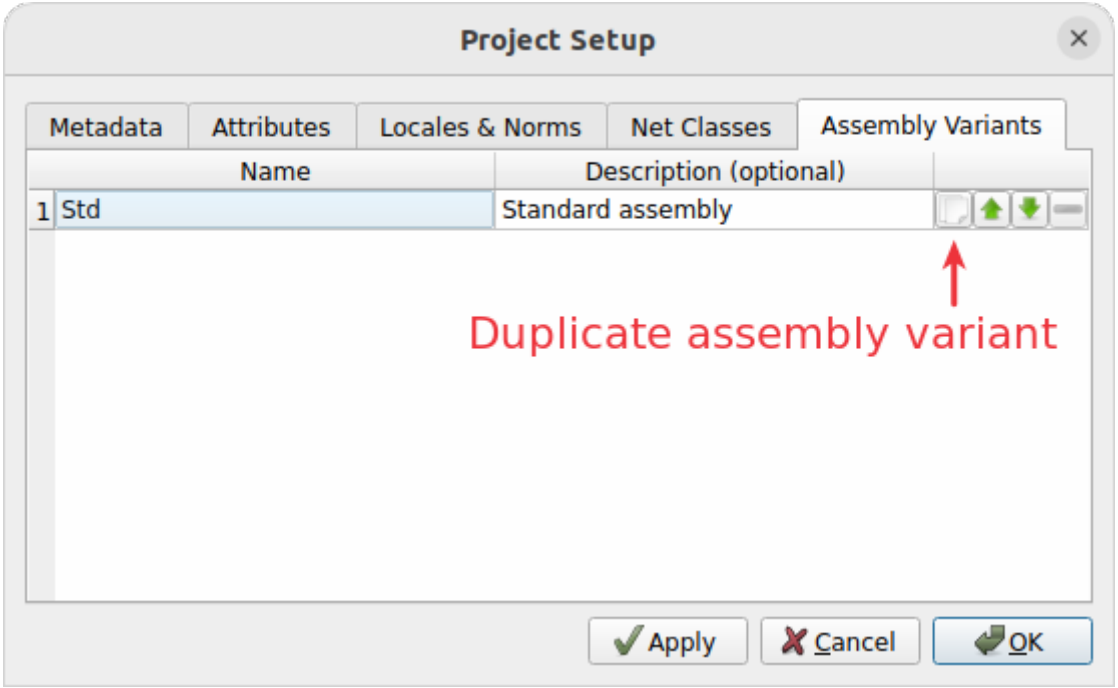
First of all, you should understand the concept of assembly variants because the assembly data of each part is related to an assembly variant.

Basically an assembly variant simply represents a variation of the BOM. For each assembly variant you'll get a separate BOM which may differ in which parts are contained, or what MPNs are specified for the parts. So if you project contains 5 assembly variants, the BOM export will create 5 BOM files.

Every project requires to contain at least one assembly variant. LibrePCB therefore automatically adds a default variant called *Std* when you create a new project. In many cases, this is sufficient and you don't need to care about assembly variants at all.

However, there are use-cases where you want to generate different BOMs for a single PCB project. Usually this is desired if multiple different products are created from the same PCB, differing only on the assembled parts. For example the same PCB could be assembled with either an RS485 interface, or with an Ethernet interface (or both). Or a voltage regulator module can either output 3.3V or 5V, depending on the value of an assembled resistor.

To define the assembly variants, open **Project › Project Setup** or press `F6` and navigate to the **Assembly Variants** tab:



A new assembly variant can only be added by duplicating an existing one. The new variant will then initially contain exactly the same parts as the existing one.

It's highly recommended to keep the variant name as short as possible (e.g. just *5V* instead of *OutputVoltage5V*) because it will be part of the file name of the exported BOMs. It must therefore also not contain any special characters. A more descriptive name may by specified in the *Description* field.

If one of the variants is considered as the default or most used variant, it's recommended to move it

to the top of the list. Some features in LibrePCB will respect only the first assembly variant, no matter how many are defined.

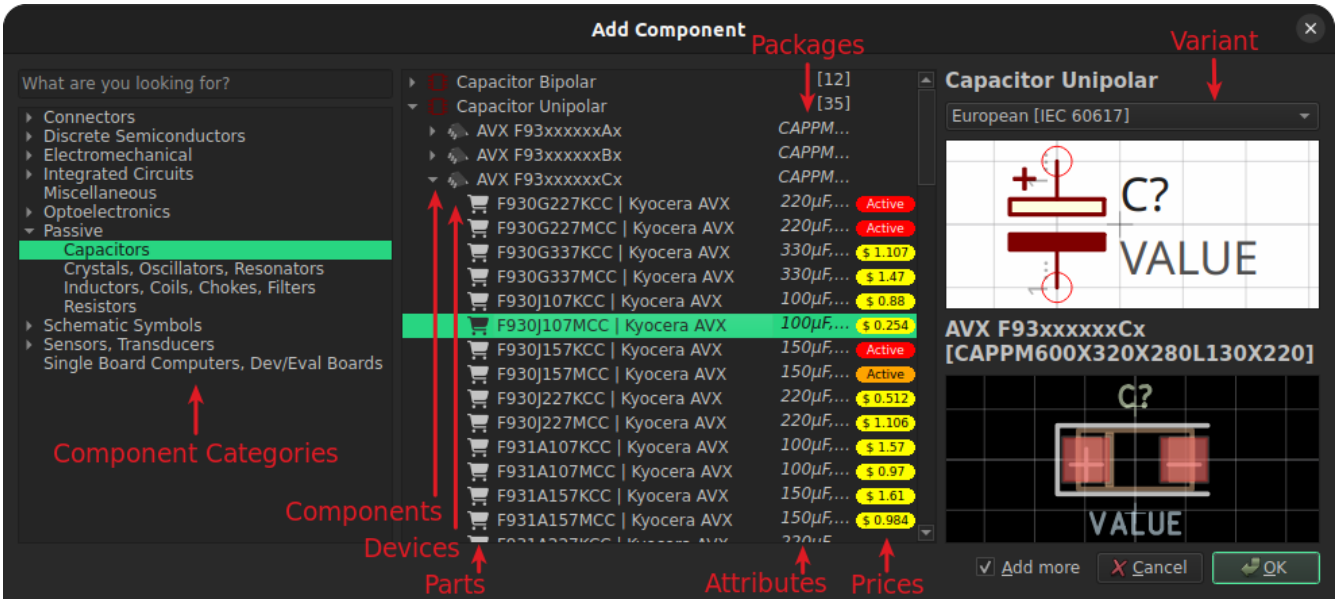
Component Assembly Data

The assembly variants explained above are defined at project scope, i.e. they are valid for a whole project. This section now describes what assembly data is defined on *component scope*.

Add Component By MPN

Often, assembly data for a component consists of a manufacturer name and an MPN (the part number). Those two values are sufficient to know which part needs to be ordered from your supplier.

The easiest way to add this data to a component is at the time when you add a new component to the schematic. Maybe you remember this screenshot from the Quickstart Tutorial:

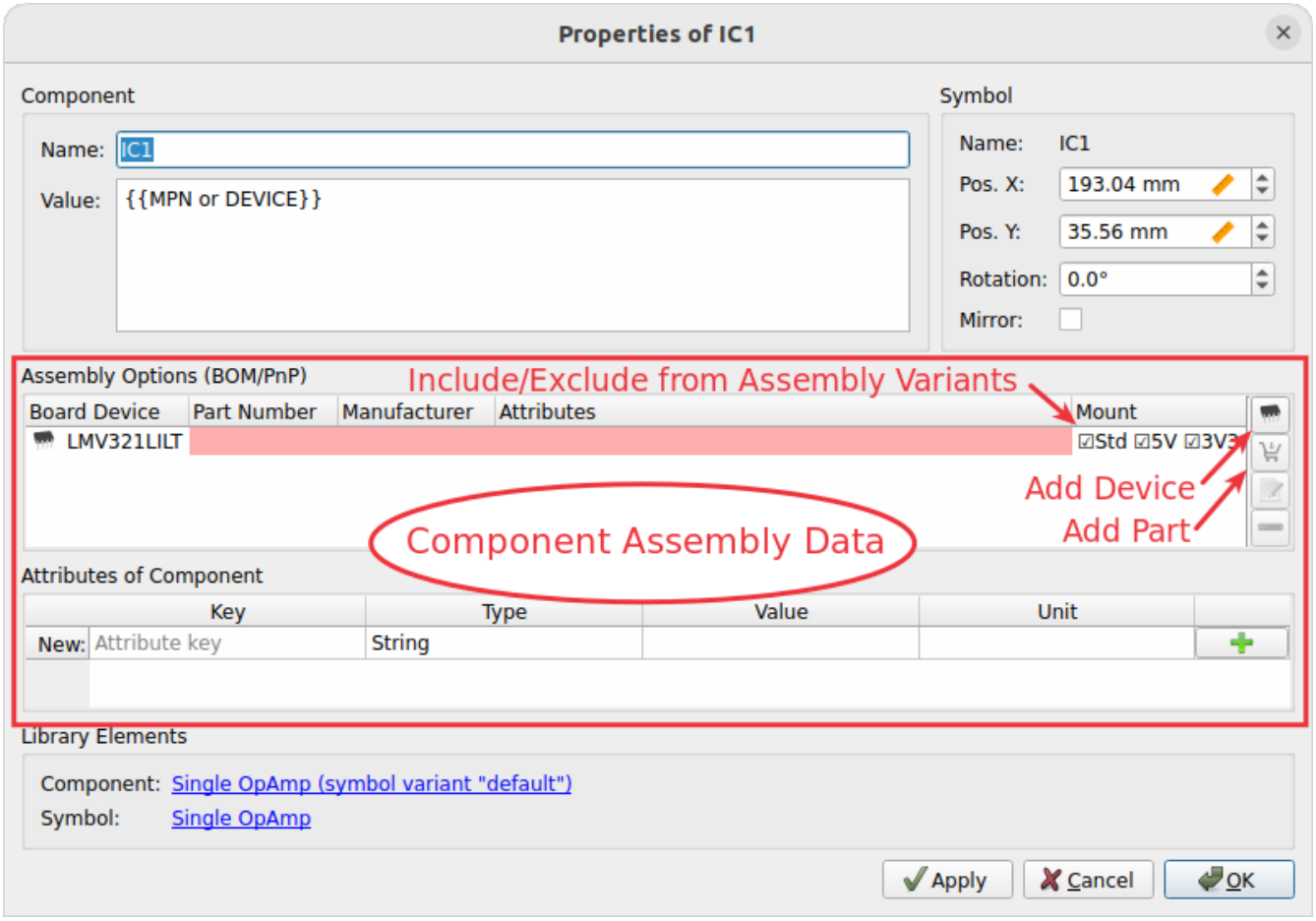


You can add either a *component*, a *device* or a *part* to the schematic. Components and devices however do not contain any assembly data. But if you add a part, the component in the schematic will automatically contain the MPN and (usually) the manufacturer as assembly data. So in that case, no manual action is needed anymore.

Assembly Data Editor

No matter if you added components by MPN to the schematic or not, the assembly data of a component can always be manually edited in its properties dialog. This is also required for more complex scenarios, e.g. if you like to exclude a component from a particular assembly variant or if you like to specify second source parts.

Click on **Properties** from the components context menu or select the component in the schematic and press `E`:



Each row in this table represents one assembly data set (or a *part*). Note that each part is valid only for a particular device — for example an OpAmp might be provided in a SOT23-5 package by one device, and in a DIP8 package by another device. So it's clear their order information will be different. Thus the first column specifies for which device a part is valid for.

In the last column, you can specify for which assembly variants a part is valid for, i.e. in which variants the part shall be assembled.

The three columns in the middle contain the actual assembly data which is accessed by the BOM export. This might be an MPN and manufacturer name, but it might also be just a single attribute like `CAPACITANCE=100nF`. Empty fields do not mean the part will be excluded from the BOM — the corresponding BOM fields will just be empty as well.

Unfortunately the UI is be a bit confusing at the moment, we'll try to improve it in upcoming releases. The most important thing to know is that **the attributes editor at the bottom is related to the selected row in the assembly options table**. So to specify an attribute for a particular assembly option, select the corresponding row and add the attribute to the table afterwards.

If no assembly option is selected, the attributes table displays the attributes set **directly on the component** (i.e. attributes not depending on assembly variants).

No worries if you don't fully understand the concept yet. The sections below will show you concrete examples for each of the possible use-cases.

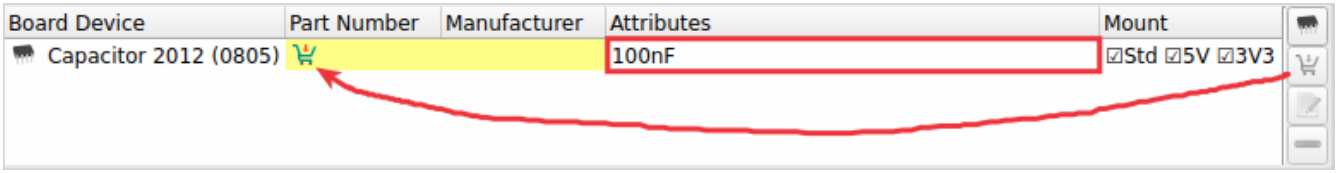
Example: Specify MPN

As mentioned above, each part could either be represented by an MPN plus manufacturer name, or just as simple attributes. If you want to have an MPN contained in the BOM, click on the **[Add Part By MPN]** button and choose a part. If the desired part doesn't exist in the library, just choose a device instead and type in the MPN and manufacturer manually. The result should then look like this:

- A row in the table needs to be selected to enable the mentioned button (not shown in the screenshot for readability reasons).
- If there's no row in the table at all yet, use the upper button **[Add Device]** instead. It works the same way, but adds a new assembly option (i.e. a row) instead of modifying an existing one.

Example: Specify Attributes

If you don't care about an exact MPN but just want to specify some attributes (like the capacitance of a capacitor), select the corresponding row and add the attributes to the attributes table. The result should look like that:



- If there's no shopping cart symbol in the desired row, you first need to add initial part information with the **[Add Part By MPN]** button as explained in the previous section. Just select a device instead of a part.
- If there's no row in the table at all yet, use the upper button **[Add Device]** first and select the desired device. Afterwards proceed as explained above.

Unfortunately this procedure is a bit cumbersome, we'll try to improve it in upcoming releases.

Note that these attributes are not automatically included in the BOM. Usually, only the `{{VALUE}}` of components is included, so it's recommended to include any relevant attributes in the component's value.

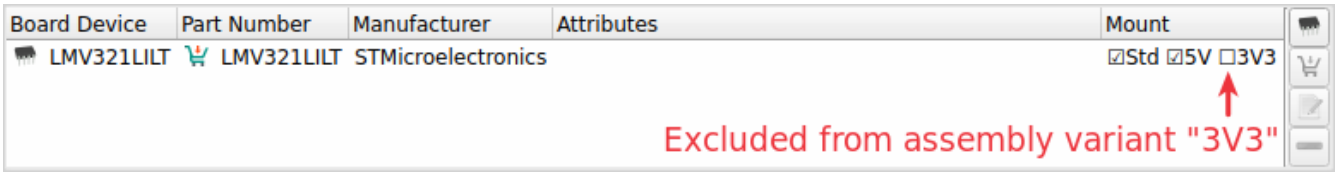
For example if you want to have the attributes `{{CAPACITANCE}}` and `{{VOLTAGE}}` of a (capacitor) component included in the BOM, set the component value to this:

```
{{CAPACITANCE}}
```

```
{{VOLTAGE}}
```

Example: Do Not Mount

A typical use-case of assembly variants is to simply exclude some components from particular assembly variants. This is done by unchecking the corresponding variants in the last column:



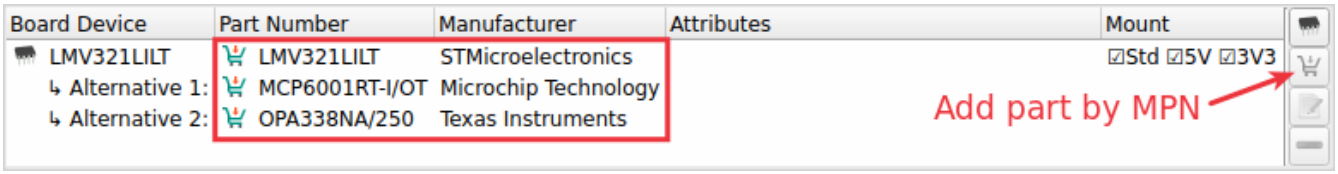
So this component won't be contained in the BOM for *3V3*, but it will be contained in the BOM for the other two assembly variants.

If your project contains only one assembly variant, the last column won't display its name. In that case there will be just a plain checkbox, but this procedure works exactly the same way.

Example: Alternative Parts (2nd Source)

Another possible use-case is to specify alternative part numbers (or attributes) for a component, to make the assembly house know which other parts they can choose if the primary part is out of stock.

This is done by selecting the corresponding assembly option row and clicking on the **[Add Part By MPN]** button multiple times, once for each alternative part. The first one is automatically considered as the *primary* part, while any further parts are *alternative* parts. The result should look like that:



This will cause the BOM to include three additional columns (those ending with `[#]` where `#` is an incrementing number) for each alternative part (common columns omitted and text abbreviated for readability):

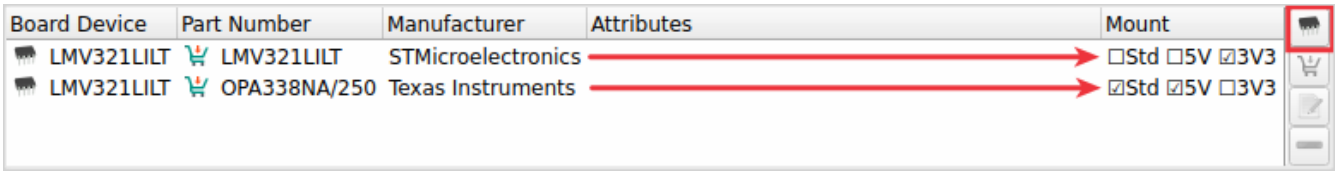
Value	MPN	Manufacturer	Value[2]	MPN[2]	Manufacturer[2]	Value[3]	MPN[3]	Manufacturer[3]
	LMV321	ST		MCP6001	Microchip		OPA338	TI

Instead of specifying alternative parts by MPN, you could also specify alternative parts by attributes exactly the same way — just specify attributes instead of MPNs

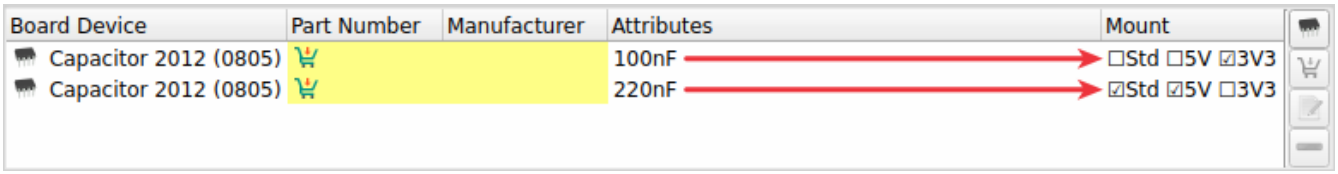
Example: Different Assembly Variants

Last but not least, it is also possible to include a component in multiple assembly variants, but using different parts. This can be done by adding multiple rows with the **[Add Device]** button and selecting the checkboxes accordingly.

For example to assemble the part *LMV321LILT* in the *3V3* assembly variant, but *OPA338NA/250* in the other two assembly variants, it would look like that:



Once again, exactly the same procedure works for specifying different attributes instead of MPNs:



In this example, the BOM for the *3V3* assembly variant specifies 100nF for the capacitor, while the BOMs for the other variants specify 220nF.

Usage Of Assembly Data

As explained earlier, the assembly data is mainly used for the BOM export. However, there are also some other things depending on assembly data.

Schematics

In schematics, typically you see the attributes and/or MPNs next to each component (usually in the `{{VALUE}}` label). But as you should now understand, this data could depend on the assembly variant or there could also be multiple MPNs added to a component. So the question arises, which of them is displayed in schematics?

The rules for substituting the placeholder `{{VALUE}}` in schematics are as follows:

1. If there exists at least one board, and the first board (which is considered as the *primary* board) contains a device for the component in question, and there is at least one part assigned to that device, any attributes from that first part are displayed in schematics.
2. If there is no such board, device or part, but the component has at least one part assigned, any attributes from that first part are displayed instead.
3. If there is no part assigned to the component at all, only the direct component attributes are taken into account (which are not depending on the assembly variant, and usually don't contain an MPN). Note that component attributes are always taken into account anyway in case a part doesn't specify a particular attribute (fallback mechanism).

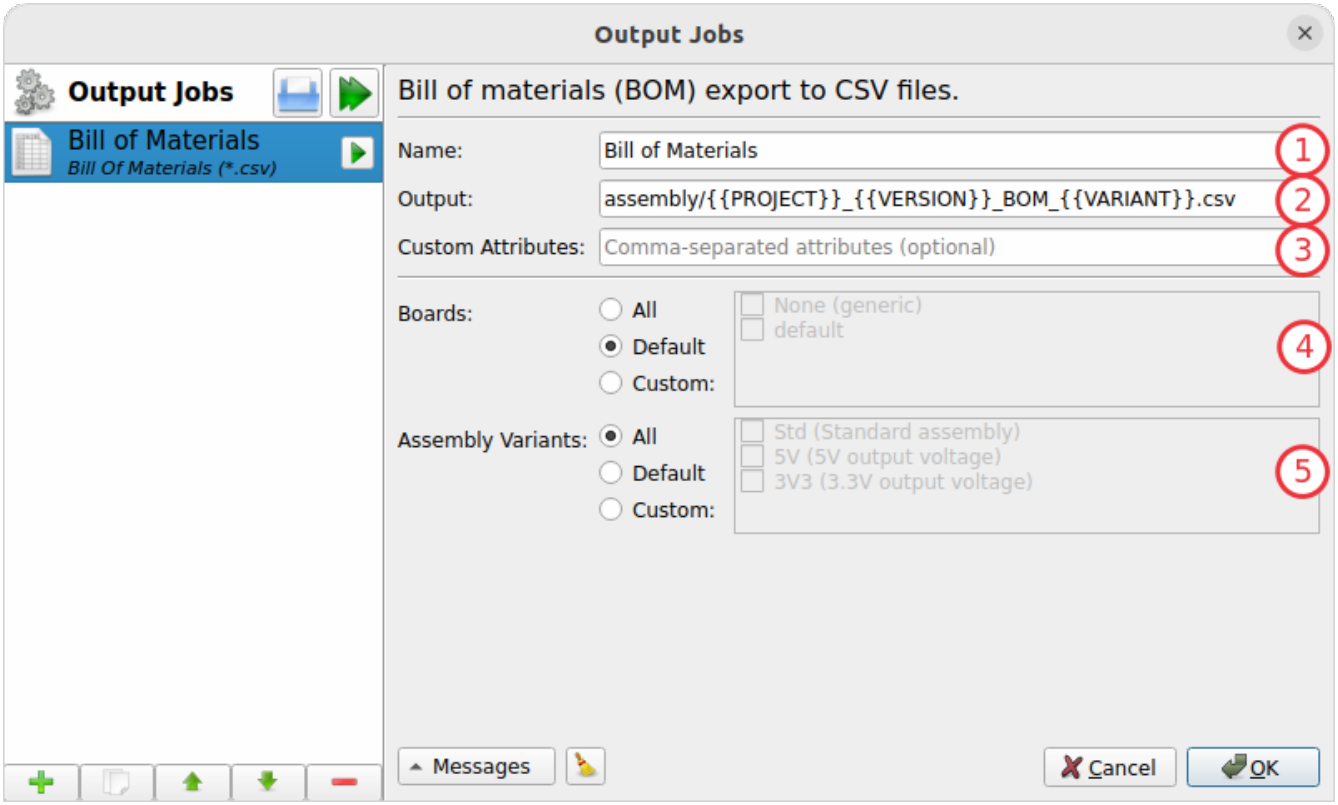
3D Board Viewer

The 3D board viewer displays only devices which are contained in the first defined assembly variant (i.e. the *default* variant). So if a device is excluded from that assembly variant, its 3D model won't show up in the 3D view.

BOM Output Job

The biggest effect of assembly data is for sure the BOM export. As explained above, for each assembly variant a separate BOM file will be created. Strictly speaking, this is not always true since it is configurable — but at least it's the default behavior.

Let's take a look at the options of the BOM output job:



1) Name

Name of the output job as shown in the list on the left (no impact on the exported files).

2) Output

File path of the BOM files to generate. Since multiple files might be generated, placeholders are required to avoid conflicting file paths. The most important placeholder here is `{{VARIANT}}` which will be substituted by the name of the assembly variant (e.g. *Std*).

3) Custom Attributes

Comma-separated list of additional columns to be included in the BOM CSV files. For example the value `SUPPLIER,DATASHEET` adds two more columns to the CSV with the component attributes `SUPPLIER` and `DATASHEET`. Usually this field can just be left empty.

4) Boards

Selection of boards to export BOMs for. If your project contains multiple boards and you want to

get a BOM for each of them (since they usually differ!), you may choose *All* or *Custom* here. **Attention**: If multiple boards are selected, you have to add the placeholder `{{BOARD}}` (or any other board-specific attribute) to the output file path to avoid generating conflicting files!

5) Assembly Variants

This option does exactly the same as the *Boards* option, just for assembly variants. Here you'll see that *All* is selected by default to get a separate BOM file for each assembly variant. And that's why the `{{VARIANT}}` placeholder is contained in the output file path by default.

For the latter two options, the value *Default* just means that the *first* object is used, i.e. the first board or the first assembly variant.

Note that **every generated BOM will contain only those components which are contained in the corresponding assembly variant**. Components excluded from a particular assembly variant won't appear in the BOM at all (no "do-not-mount" mark or something like that).

Similarly, only components actually added to the selected board will be contained in the BOM. Even if a component is contained in an assembly variant, but not added to the board selected for the export, it won't be contained in the BOM.

If a component specifies multiple parts (e.g. for second source reasons), the BOM will contain additional columns to include all those parts. So the number of columns depends on the maximum number of parts added to components. Of course only those parts valid for the selected board will be contained in the BOM.

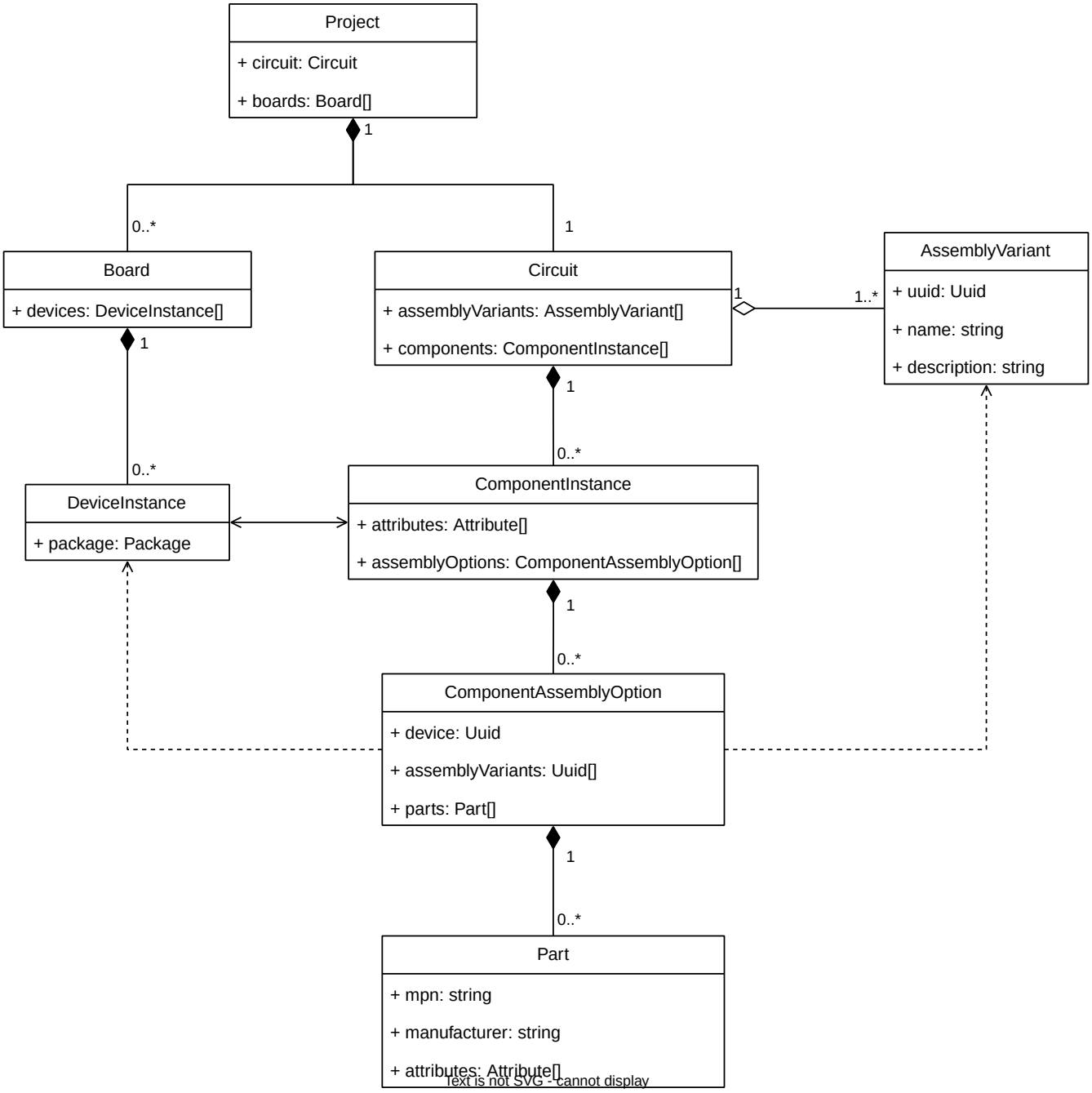
Other Output Jobs

Although the BOM export is the primary use for assembly data, the same concept also applies to these output jobs:

- Pick&Place CSV / Gerber X3
- Board STEP Model

Software Architecture

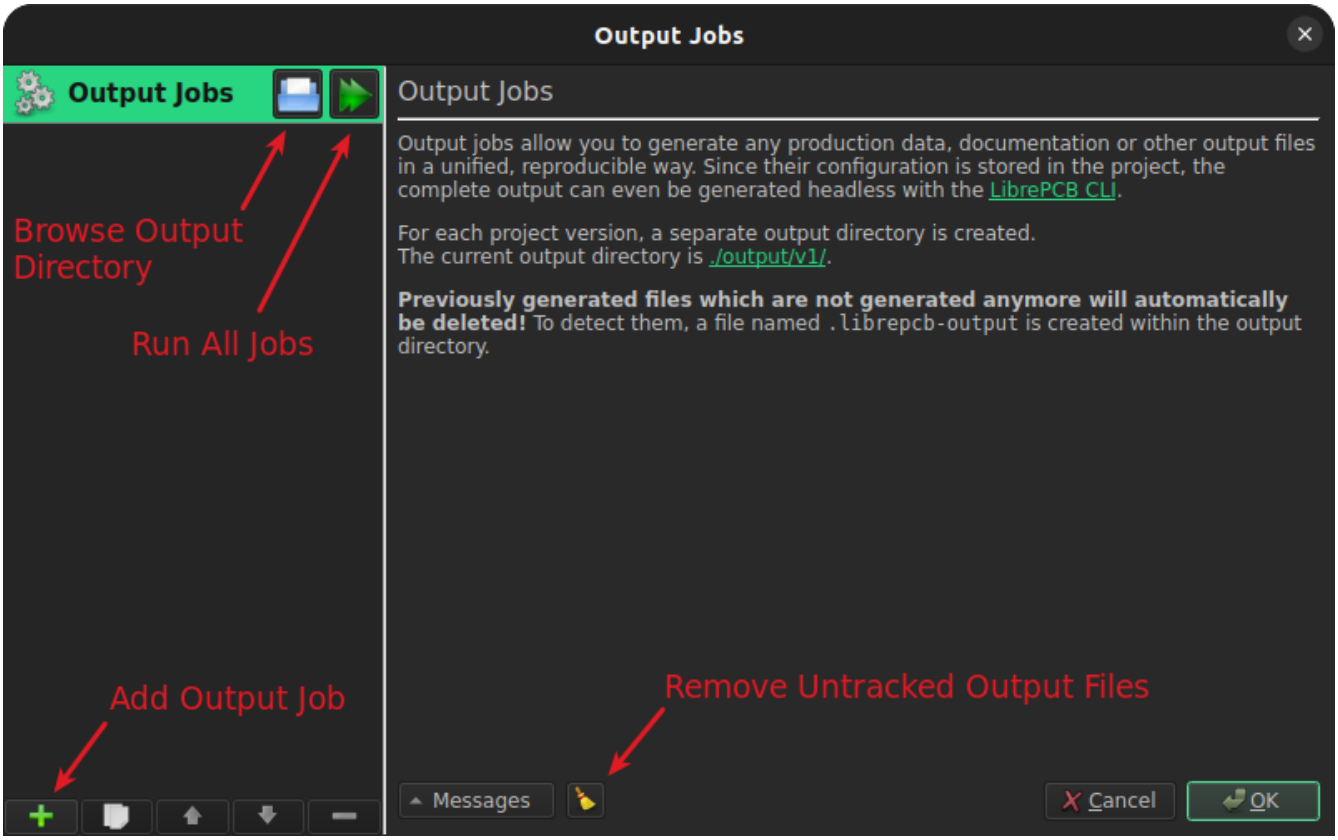
For the nerds among us, this diagram about the underlying software architecture might help to understand this feature:



Output Jobs

At the end of every PCB project, we need to generate various production data files, e.g. for PCB manufacturing, PCB assembly, or simply for documentation purposes. LibrePCB provides a unified interface to generate any kind of production data for a project. It is called *Output Jobs*.

From either the schematic- or board editor, open the *Output Jobs* dialog with the menu item **Project › Output Jobs...** or by pressing `F11`:



Output Directory

Any output files are always generated into the project directory `output/<VERSION>/` where `<VERSION>` represents the project's version number as specified in the project setup dialog (`F6`). So for the default version number `v1`, the output data goes to `output/v1/` within the project directory.

Since existing output files are overwritten without prompt when running jobs, always make sure the version number has the desired value (e.g. bump it after releasing a PCB version).

Note that LibrePCB automatically creates a file named `.librepcb-output` within this output directory. It is used internally to detect untracked output files (see details below).

Add Jobs

The list of output jobs is initially empty for every new project. So for any output data you like to generate, you need to add a corresponding job with the plus button at the bottom left. This will add a new job to the list on the left side of the dialog. Select the new job to modify its properties.

This way you can add as many jobs as you need, even multiple jobs of the same type are allowed (e.g. if different settings are needed). The available job types and their configuration options are listed below.

Run Jobs

Once you added all desired jobs, you can run them either individually or all at once with the corresponding buttons in the job list. Note that every run overwrites the output files of previous runs. In addition, a run also removes previously generated files of the same job, for example after

modifying the output file path configuration. This ensures that no outdated files are left over in the output directory.

Untracked Output Files

After running any output job, LibrePCB automatically scans the output directory for any untracked files, i.e. files not generated by any output job. For example when deleting an output job, its previously generated output files are still left over in the output directory. As this might not be intended, LibrePCB reports them in the output messages and provides a button to delete them. It is your choice to either keep or delete these files.

Common Configuration Options

Although the configuration options differ from job to job, there are some options available in several jobs so these are documented here:

Name

Just a user-defined, unique name to identify each job in the list of all jobs.

Output

File path of the file(s) to generate, relative to the Output Directory. May use placeholders like `{{PROJECT}}` or `{{VERSION}}`. For example the output path `{{PROJECT}}_{{VERSION}}_BOM.csv` might result in the full output file path `output/v1/MyProject_v1_BOM.csv` relative to the project's root directory.

Boards

Selection of boards to run the job for. For example if a project contains multiple boards and you want to generate Gerber files for each of them, this option allows to do that with a single output job instead of creating separate output jobs for each board. The available values are:

- **Default**: Run the job only for the default board (which is always the *first* board in a project). So the job is run exactly once, except if the project contains no board at all; then the job is not run.
- **All**: Run the job for each existing board. If no board exists, the job is not run.
- **Custom**: Manually select the boards the job shall run for (0..n). Some jobs also provide the special value "None" which means a job is run outside the context of a board.

If multiple boards are selected, you have to add the placeholder `{{BOARD}}` (or any other board-specific attribute) to the output file path to avoid generating conflicting files!

Assembly Variants

Similar to the *Boards* option, this option allows to specify the assembly variants a job shall run for. For example if a project contains multiple assembly variants and you want to generate a separate BOM for each of them, this option allows to do that with a single output job instead of creating separate output jobs for each assembly variant. The available values are:

- **Default**: Run the job only for the default assembly variant (which is always the *first* one). So

the job is run exactly once.

- **All**: Run the job for each existing assembly variant.
- **Custom**: Manually select the assembly variants the job shall run for (0..n).

If multiple assembly variants are selected, you have to add the placeholder `{{VARIANT}}` to the output file path to avoid generating conflicting files!

Example 1. Boards & Assembly Variants

To help understanding the *Boards* and *Assembly Variants* options, consider a project with the boards "Board1" and "Board2", and the assembly variants "AV1" and "AV2". If you set both configuration options to "All", a job is run for each combination of them, i.e. four times:

- Run 1: `{{BOARD}}=Board1`, `{{VARIANT}}=AV1`
- Run 2: `{{BOARD}}=Board1`, `{{VARIANT}}=AV2`
- Run 3: `{{BOARD}}=Board2`, `{{VARIANT}}=AV1`
- Run 4: `{{BOARD}}=Board2`, `{{VARIANT}}=AV2`

Job Types

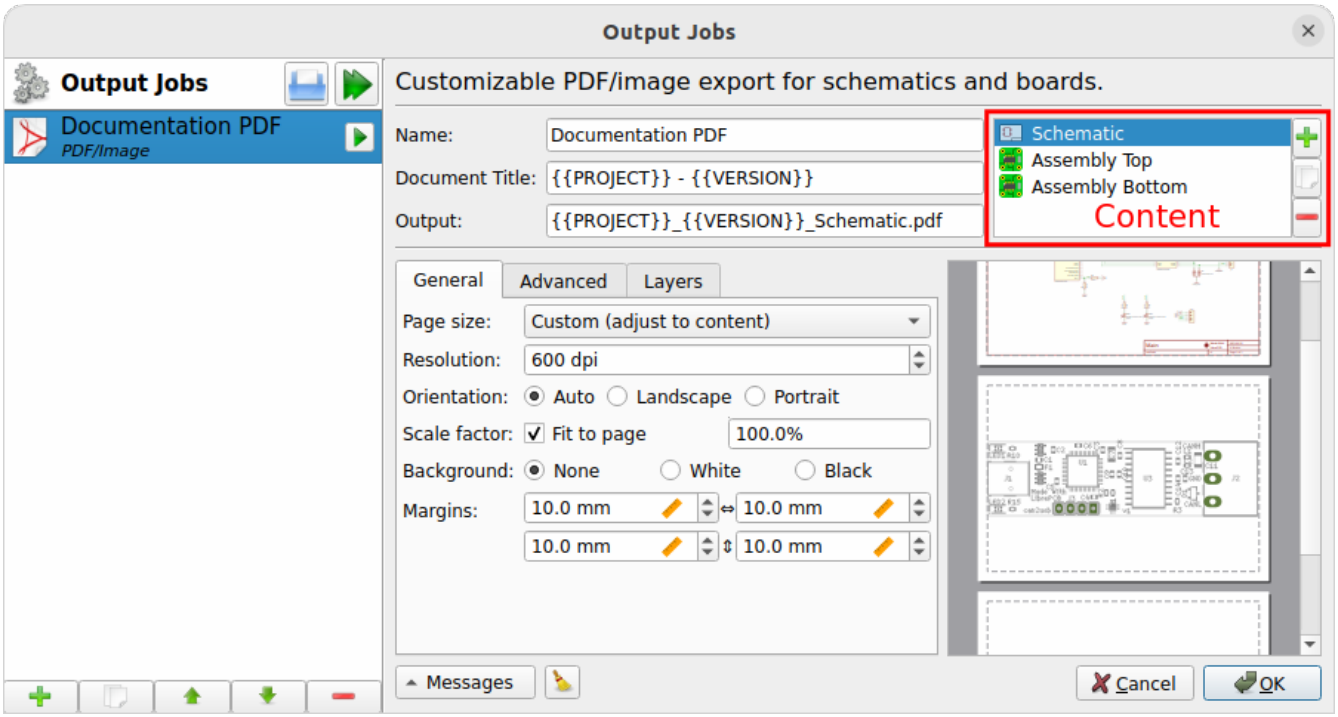
This section descibes all the available job types and their configuration options.

PDF/Image

Generates a PDF or image(s) containing either the schematics, the board(s) or both. For convenience, there are two built-in presets to quickly generate frequently needed documents:

- **Schematic PDF/Image**: Adds a job to generate a PDF with the schematics.
- **Board Assembly PDF/Image**: Adds a job to generate a PDF with top/bottom assembly plans for the board(s).

Both presets add the same type of output job, just with different initial configuration options.



Name

See Common Configuration Options.

Document Title

Title of the generated document, to be set in the metadata of the output file. This option only has an effect if the output file type is either PDF or SVG. Placeholders like `{{PROJECT}}` may be used in the value.

Output

See Common Configuration Options. The specified file extension determines what output file format is used. The extension `.pdf` generates a single PDF containing all pages. The extension `.svg` generates a separate SVG for each page. Pixmap extensions like `.png` generate a separate image file for each page. Note that the supported pixmap extensions depend on the platform, but `.png` should always be available.

If multiple image files are generated, a page number is automatically appended to the file name, for example the output path `image.png` may generate the files `image1.png` and `image2.png`.

Content (list view on the right side)

The actual content of the output document is specified in the list view on the right side. A content item could either be the schematic or the board, while for the board there exist two different presets *Board Image* and *Assembly Top/Bottom* for convenience. The schematic type adds 0..n pages to the output (depending on how many sheets your project has), while a board type adds one page per board to the output document. So with this output job you can freely choose whether the output document represents a schematic, or a board, or even contains both. The pages in the output document are added in the same order as the specified content items.

General/Advanced

These options affect the layout of the output document and should be mostly self-explaining.

Note that these options refer to the currently selected content item (the list at the top right), so they are independent for each content item.

Layers

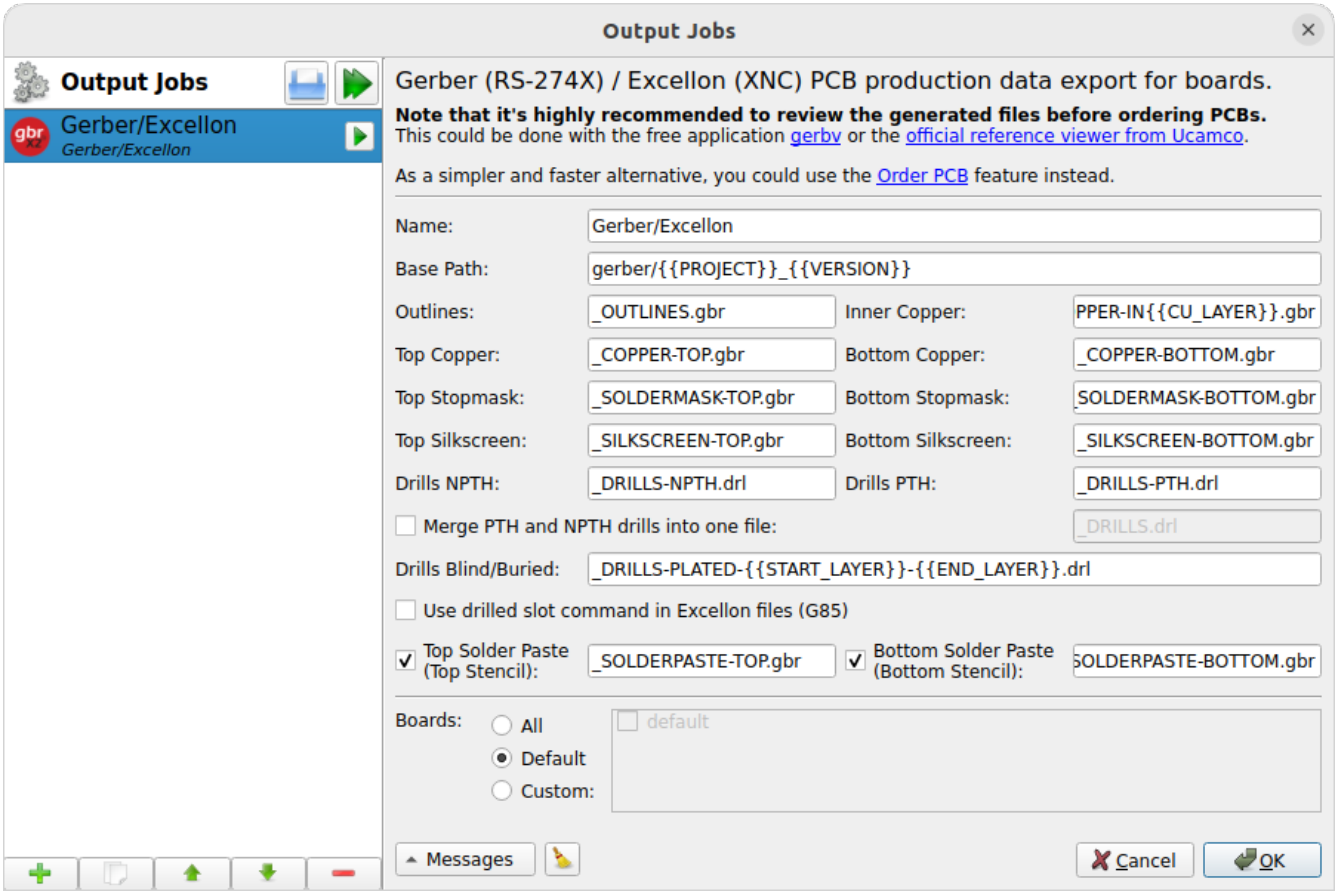
Selection of layers to be included in the output document, and their color. The color of each layer can be changed by double-clicking on a layer list item. The colors are not takes from your workspace settings to make this output job independent of user settings. Note that these layer settings refer to the currently selected content item (the list at the top right), so they are independent for each content item.

Gerber/Excellon

Generates RS-274X (Gerber X2) and IPC-NC-349/XNC (Excellon) files for PCB production. For convenience, there are two built-in presets available:

- **Gerber/Excellon**: Uses default options with `.gbr` file extension according recommendation by the Gerber standard.
- **Gerber/Excellon (Protel style)**: Configures Protel file extensions and sets some options for compatibility with cheap PCB manufacturers.

Both presets add the same type of output job, just with different initial configuration options. Please check the documentation of your desired PCB manufacturer which options are supported. If you intend to order the PCB through LibrePCB Fab, you don't need to add a Gerber/Excellon output job at all.



Name

See Common Configuration Options.

Base Path

Specifies the common output path prefix to be used for all the output files. So the actual output file paths consist of this path appended by the corresponding output file suffix as explained below. See also the *Output* option documented in Common Configuration Options.

Outlines

Output file suffix for the board outlines. Will contain all objects on the *Board Outlines* and *Board Cutouts* layers.

Top/Bottom Copper

Output file suffix for the *Top Copper* resp. *Bottom Copper* layers.

Inner Copper

Output file suffix for the *Inner Copper* layers. For each used inner layer, a separate Gerber file is created. Therefore the placeholder `{{CU_Layer}}` needs to be used, which is substituted by the inner layer number ("1" for the first inner layer, just below *Top Copper*).

Top/Bottom Stopmask

Output file suffix for the *Top Stop Mask* resp. *Bottom Stop Mask* layers.

Top/Bottom Silkscreen

Output file suffix for the top resp. bottom silkscreen layers as configured in the board setup dialog. Note that these files are only generated when enabled in the board setup dialog.

Drills NPTH

Output file suffix for the non-plated through-hole Excellon drill file, i.e. all drills which are called *Hole* in LibrePCB (including slotted holes).

Drills PTH

Output file suffix for the plated through-hole Excellon drill file, i.e. all through-hole pads and through-hole vias (including slotted pads).

Merge PTH and NPTH drills into one file

If this option is enabled, all through-hole drills are exported into a single Excellon drill file (with the suffix provided next to this option) instead of generating separate files. So the *Drills NPTH* and *Drills PTH* files won't be created if this option is checked. Note that generally this option is not recommended, but some PCB manufacturers (especially cheap ones) are not able to handle separate files for PTH and NPTH. In that case, this option needs to be enabled.

Drills Blind/Buried

If blind/buried vias are used in the board, a separate Excellon drill file will be created for each different drill layer pair. This option specifies the file name suffix for these files. Since multiple files might be created, the placeholders `{{START_LAYER}}` and `{{END_LAYER}}` need to be used, which will be substituted by either "TOP", "BOTTOM" or "INx" where "x" is the inner layer number starting at 1. Or alternatively, the placeholders `{{START_NUMBER}}` and `{{END_NUMBER}}` are

also available which are substituted by just a number (1 = top layer, 2 = first inner layer etc.).

Use drilled slot command in Excellon files (G85)

If your board contains slots (plated or non-plated), they are exported to Gerber files with G00..G03 commands by default. By checking this option, the G85 slot command will be used instead. This is generally not recommended, but some PCB manufacturers may not support the G00..G03 commands. In that case, the G85 command might need to be used instead.

Top/Bottom Solder Paste

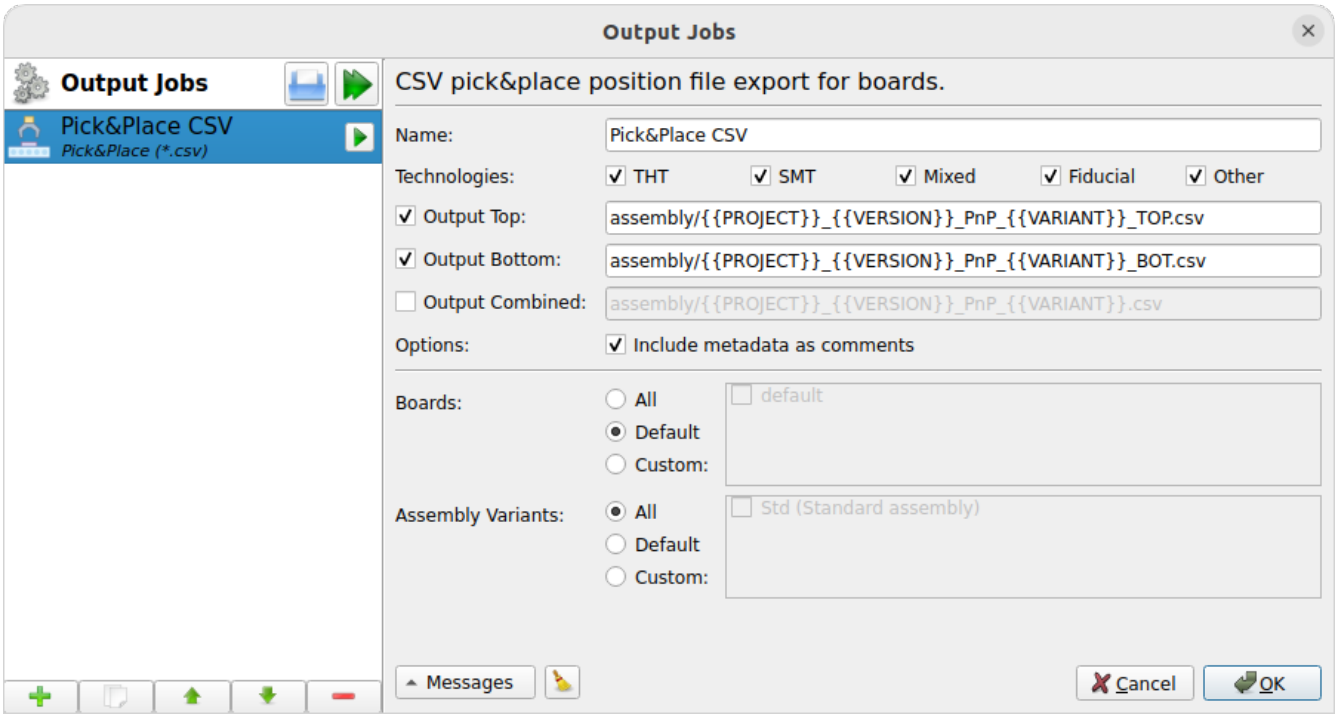
Output file suffix for the *Top Solder Paste* resp. *Bottom Solder Paste* layers. These files are not directly used for the PCB production, but for the SMD stencil to apply solder paste on the PCB. If you don't need a stencil, the generation of these files should be disabled by unchecking the corresponding checkboxes.

Boards

See Common Configuration Options.

Pick&Place CSV

Generates a pick&place position file containing the coordinates of each device on the PCB as comma-separated values (CSV). This file is needed for automatic PCB assembly by pick&place machines. Alternatively, the Gerber X3 format might be used instead, which is provided by the output job type Pick&Place Gerber X3.



Name

See Common Configuration Options.

Technologies

Selection of device types to be included in the output file. For example if only "THT" is selected, you'll get a CSV file containing only THT devices. The available technologies are:

- **THT**: Pure through-hole devices, i.e. all leads are THT.
- **SMT**: Pure surface-mount devices, i.e. all leads are SMT.
- **Mixed**: Devices containing both through-hole and surface-mount loads (for example SMT connectors wich THT pads for mechanical stability).
- **Fiducial**: Whether fiducial coordinates (for PCB alignment) should be contained in the pick&place file or not.
- **Other**: Any other special device types, for example pure mechanical devices to be mounted with screws instead of soldering.

Output Top/Bottom/Combined

See Common Configuration Options. Three different output paths can be configured to get either separate files for top/bottom devices, or a single file for all devices. Must have file extension `.csv`.

Use the checkboxes to select the files to generate.

Include metadata as comments

If checked, the output CSV files will contain a header comment with some metadata like project name, generation date etc. This is helpful for traceability/documentation purposes, but some CSV readers fail to ignore this comment. If you're unsure, just uncheck this option as this is always safe.

Boards

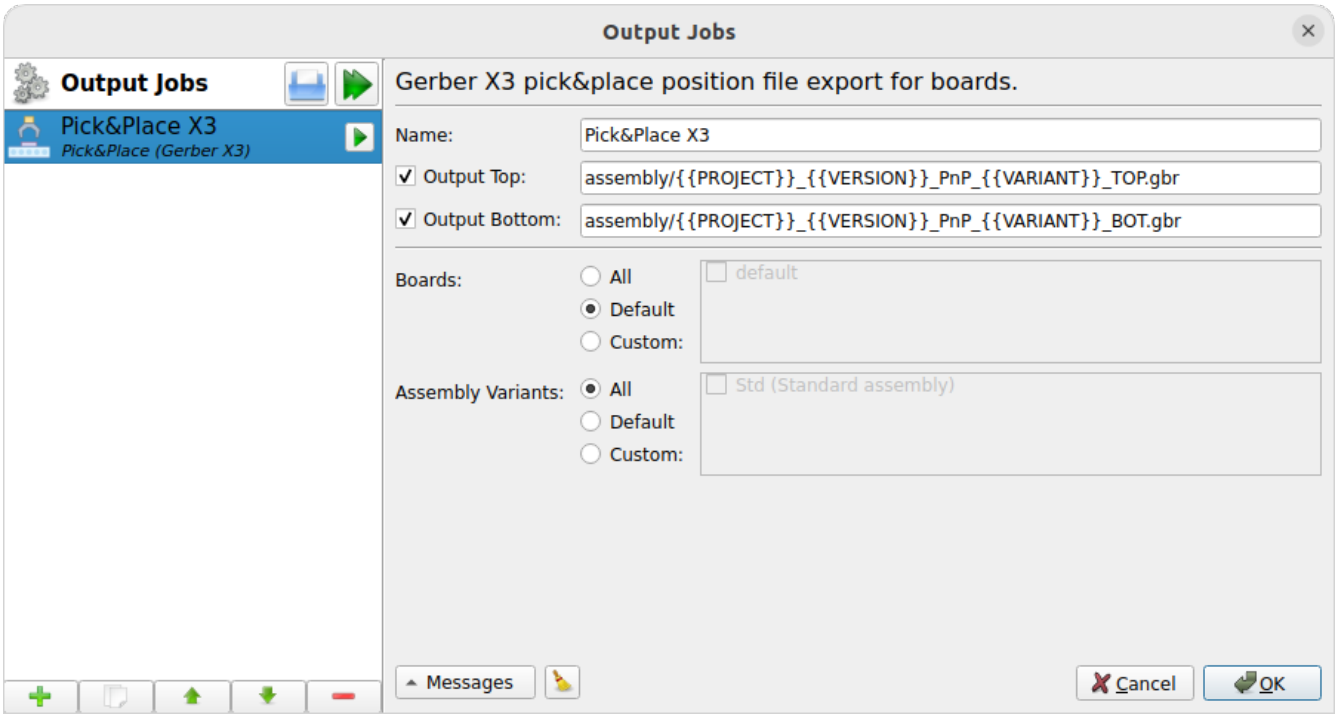
See Common Configuration Options.

Assembly Variants

See Common Configuration Options.

Pick&Place Gerber X3

Same as Pick&Place CSV, but generating Gerber X3 pick&place files instead of CSV files. The advantage of this format is that it's standardized, while there's no standard for CSV pick&place files so CSV might cause issues or at least involves manual effort during pick&place machine setup. However, Gerber X3 is not as widely supported by assembly houses as CSV files.



Name

See Common Configuration Options.

Output Top/Bottom

See Common Configuration Options. Two different output paths can be configured to get separate files for top/bottom devices. Should have file extension `.gbr`.

Use the checkboxes to select the files to generate.

Boards

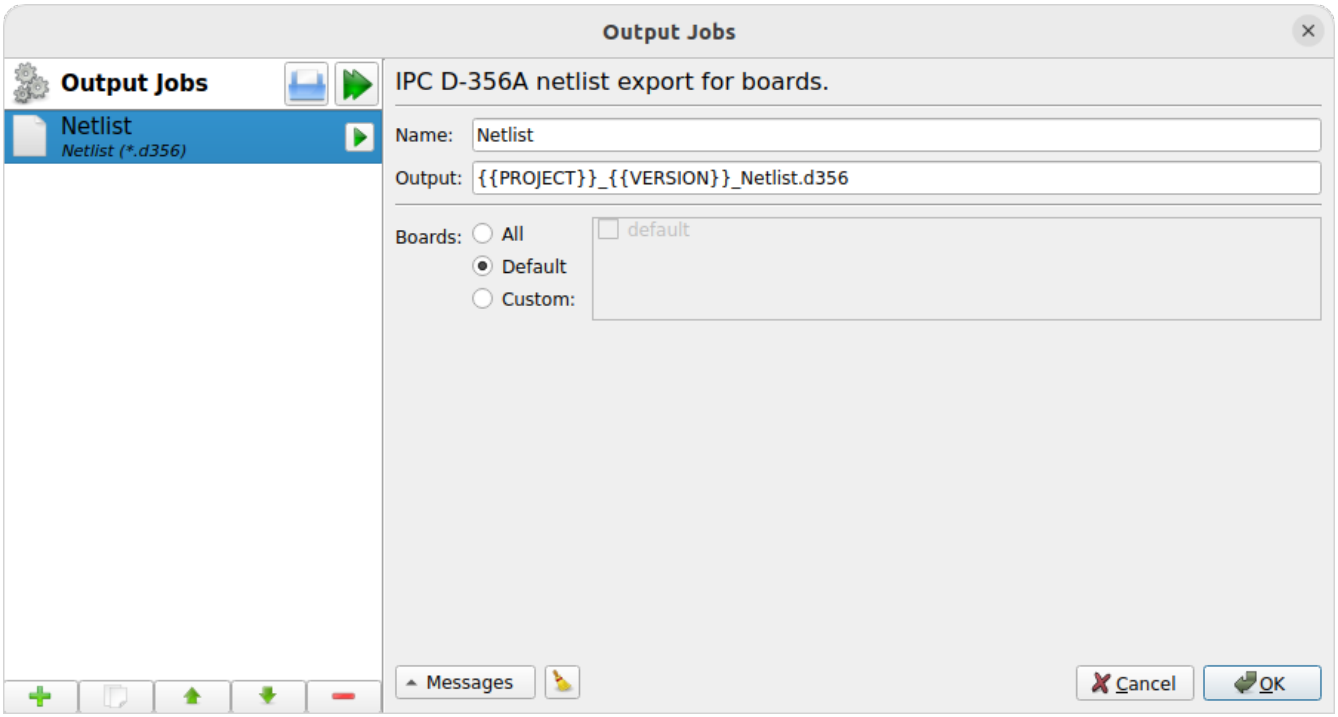
See Common Configuration Options.

Assembly Variants

See Common Configuration Options.

Netlist

Generates an IPC D-356A netlist used for automatic electrical testing of the PCB.



Name

See Common Configuration Options.

Output

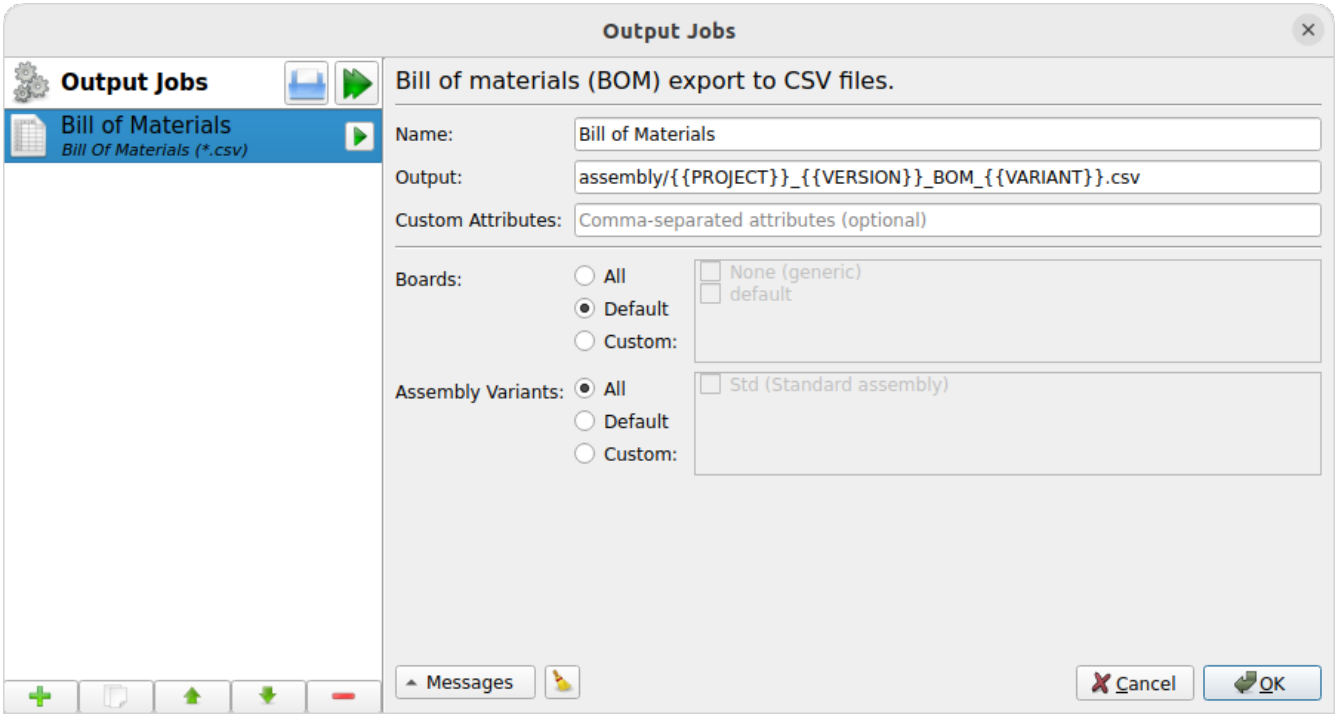
See Common Configuration Options. Must have file extension `.d356`.

Boards

See Common Configuration Options.

Bill Of Materials

Generates a bill of materials (BOM) in CSV format, containing all devices to be ordered for a particular assembly variant.



Name

See Common Configuration Options.

Output

See Common Configuration Options. Must have file extension `.csv`.

Custom Attributes

This option allows to add custom additional columns to the output CSV file. For this purpose, attributes are used. For example the value "DIGIKEY,MOUSER" adds the two columns "DIGIKEY" and "MOUSER" to the CSV, with the corresponding values of these attributes on devices. The value will be empty for devices not providing a particular attribute.

When adding the suffix `[]` to a custom attribute, it is considered as a **per-part** attribute instead of a **global** attribute. This means the attribute is not exported to the BOM only once, but once per part of a component. This might be desired if some components have alternative (i.e. multiple) part numbers specified. See Assembly Data for details.

Boards

See Common Configuration Options.

Assembly Variants

See Common Configuration Options.

The actual number of columns in the output file depends on whether (and how many) alternative part numbers are specified on components. The component with the most part numbers defines how many columns the BOM CSV will have. See Assembly Data for details.

The order and name of columns is as follows:

- Quantity
- Designators
- Package
- Custom global attribute columns (optional, see above)
- *For each part (minimum 1):*
  - Value
  - MPN
  - Manufacturer
  - Custom per-part attribute columns (optional, see above)

For the typical use-case (without custom attributes and no alternative part numbers specified in schematics), the CSV header looks as following:

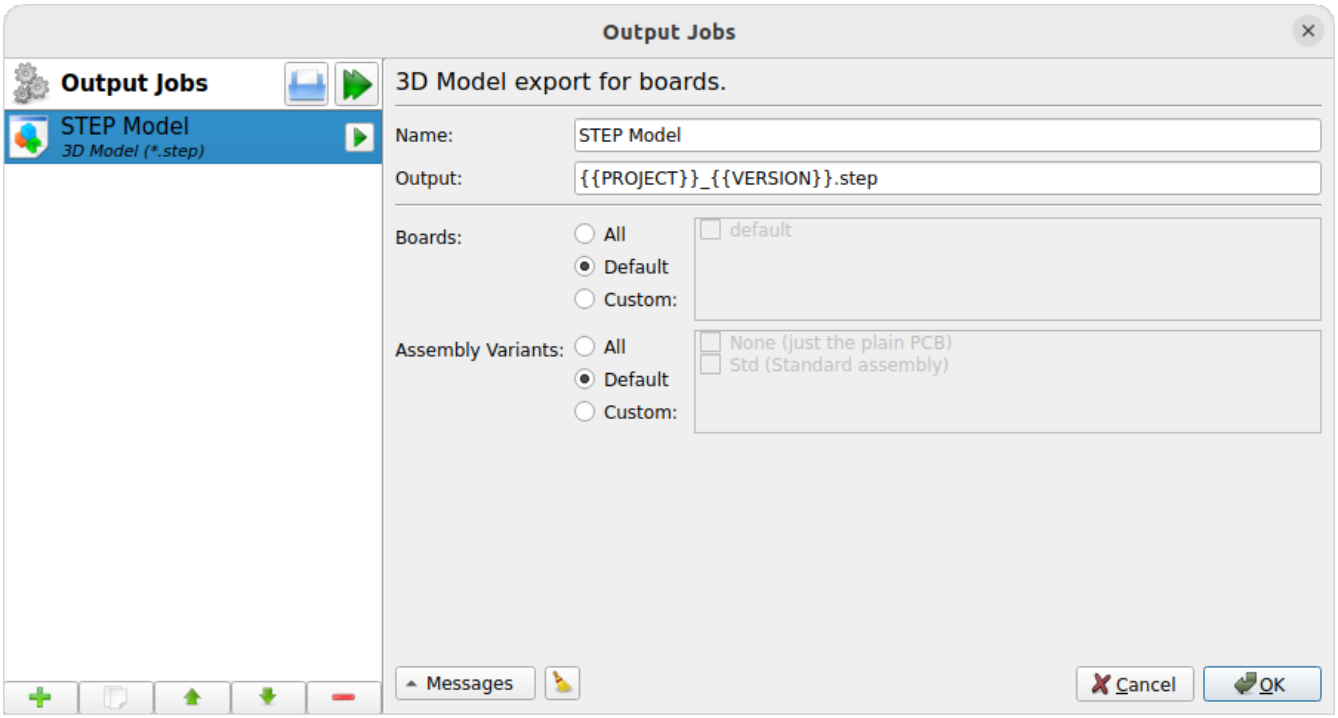
```
Quantity,Designators,Package,Value,MPN,Manufacturer
```

If at least one component has specified one alternative part number, three additional columns will appear:

```
Quantity,Designators,Package,Value,MPN,Manufacturer,Value[2],MPN[2],Man
ufacturer[2]
```

3D Model

Exports a board as a 3D STEP file for importing it in a mechanical CAD (MCAD). Note that in contrast to the built-in 3D viewer, the exported STEP model won't contain details like copper traces, solder resist or silkscreen.



Name

See Common Configuration Options.

Output

See Common Configuration Options. Must have file extension `.step` or `.stp`.

Boards

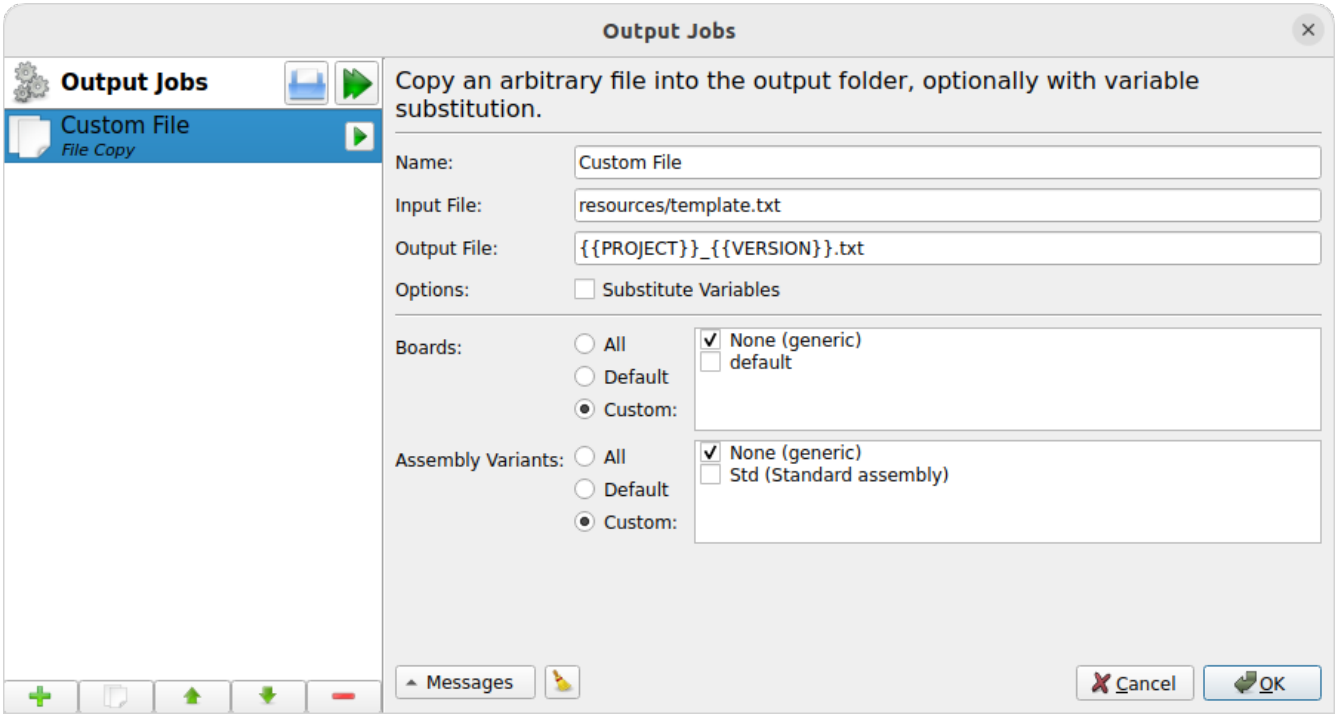
See Common Configuration Options.

Assembly Variants

See Common Configuration Options. Only devices contained in the corresponding assembly variant will be exported. The special value "None" means that only the plain PCB is exported, without any devices on it.

File Copy

Special job which actually doesn't *generate* anything, but *copies* an existing file into the output directory. This is intended for example to include custom files like instruction notes in the data to be sent to the PCB manufacturer or assembly house.



Name

See Common Configuration Options.

Input File

Path to the (existing) input file to copy, relative to the project's root directory. It's highly recommended to place this file within the `resources/` directory (at least do not place it in the `output/` directory!).

Output File

See *Output* in Common Configuration Options. Should have the same file extension as the input file.

Substitute Variables

If checked, the input file is read by LibrePCB and any occurrences of attribute placeholders like `{{PROJECT}}`, `{{VERSION}}` or `{{DATE}}` will be substituted by their value before writing that content to the output destination. Project attributes are always available, while board attributes and assembly variant attributes are only available if the job is run in the context of a board resp. assembly variant (see options below).

That this option shall only be used on text files, not on binary input files.

Boards

See Common Configuration Options. The special value "None" means that this job does not run in the context of a board and will be run exactly once, no matter how many boards the project

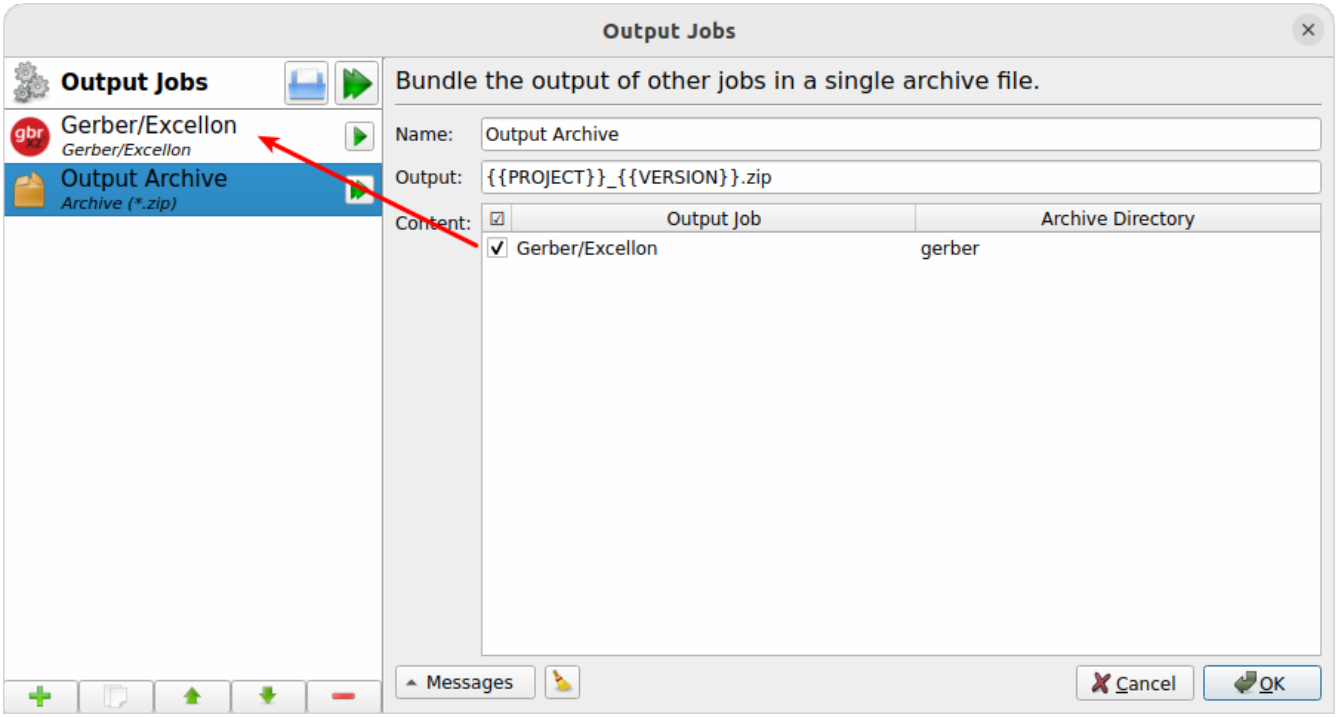
contains.

Assembly Variants

See Common Configuration Options. The special value "None" means that this job does not run in the context of an assembly variant and will be run exactly once, no matter how many assembly variants the project contains.

Archive

Special output job which combines the output of other jobs in a single archive file (e.g. ZIP).



Name

See Common Configuration Options.

Output

See Common Configuration Options. The file extension specifies the type of archive to create. Currently only `.zip` is supported.

Content

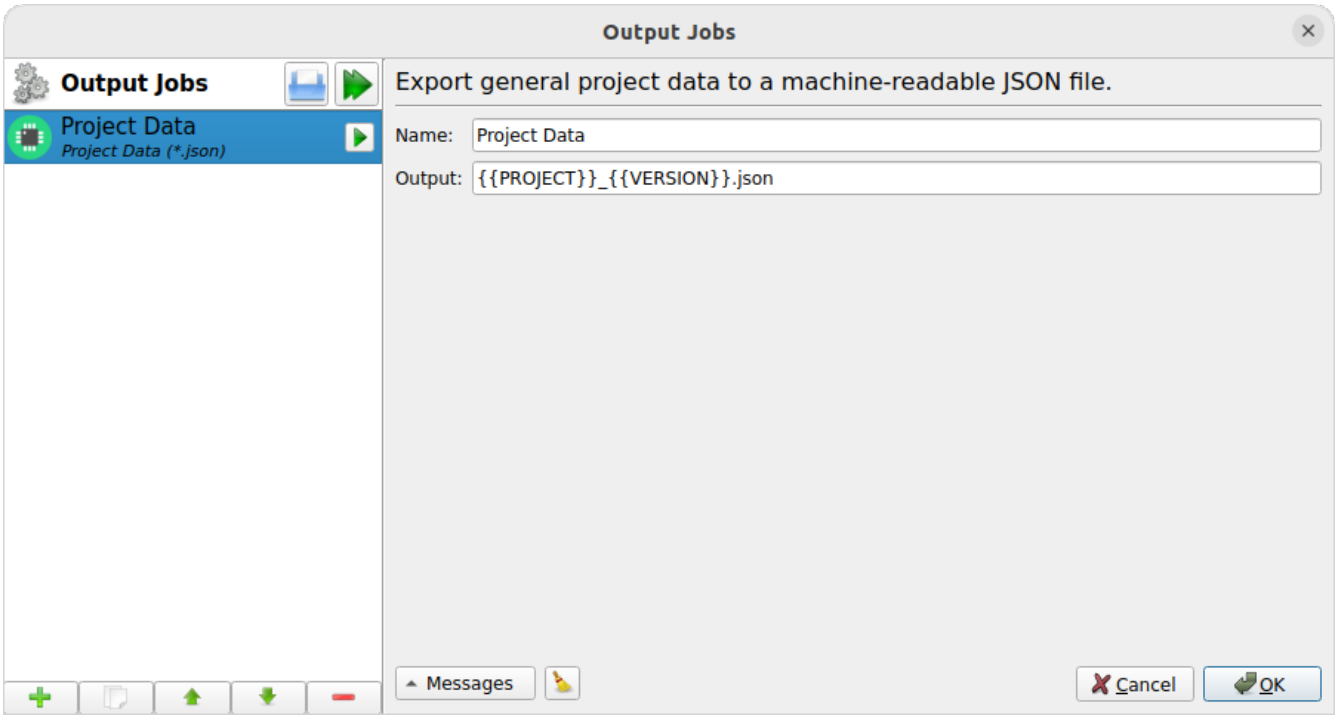
Selection of jobs which output files shall be added to the archive. Note that only jobs listed **prior** to the archive job may be selected to ensure no cyclic dependencies can be created (an error will be raised when violating this rule).

All output files of the selected jobs are added to the root directory of the archive, with their original file name but with any subdirectory stripped. It's not possible to specify different file names just for the archive. However, the *Archive Directory* column allows to move all files of a particular job into a custom subdirectory (for example to move all Gerber files into a `gerber/` directory and all pick&place files into `assembly/`).

When running an archive job, LibrePCB will automatically run all its dependent jobs first to generate their output.

Project Data

Generates a custom JSON file containing some metadata about the project. This is not intended for end users but it's still listed publicly for completeness.



Name

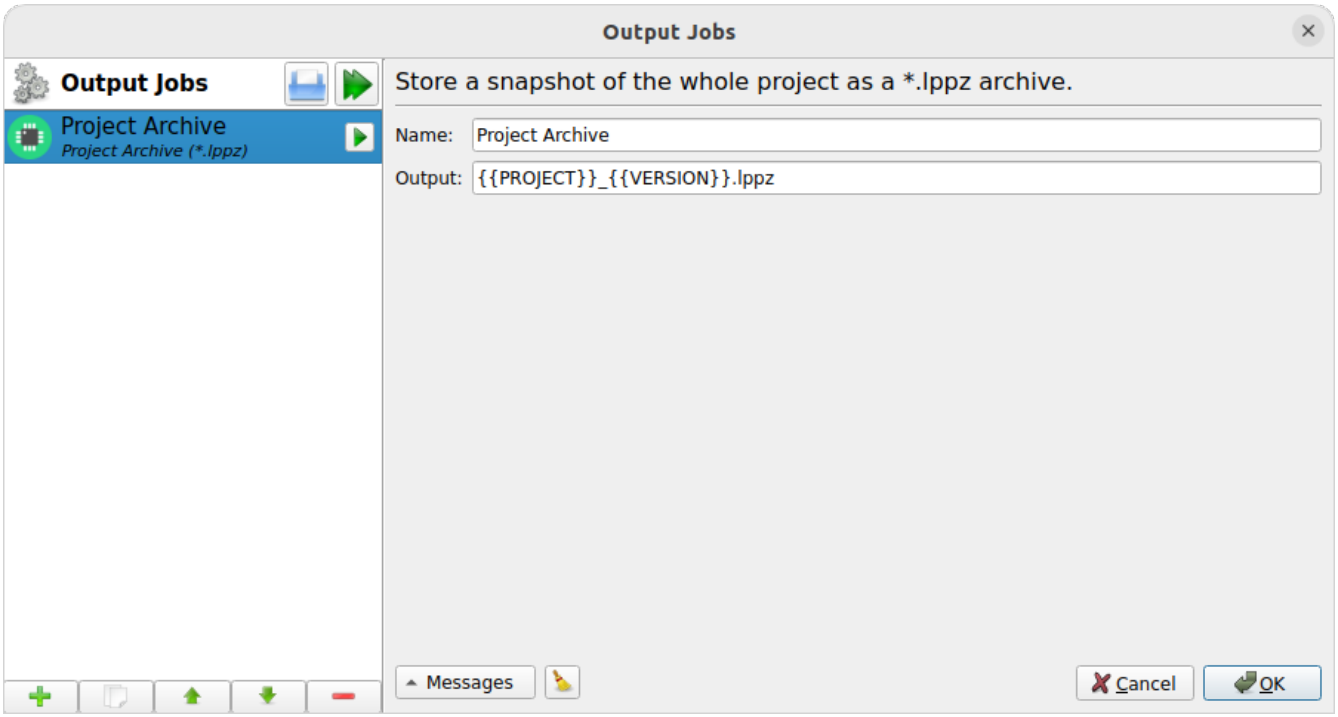
See Common Configuration Options.

Output

See Common Configuration Options. Should have file extension `.json`.

Project Archive

Exports the whole project to a single `*.lppz` file (which is simply a ZIP). Intended to keep a snapshot of a particular project version which can directly be opened with LibrePCB from the desktop file manager. In addition, this file could be uploaded to fab.librepcb.org to (re-)order the PCB.



Name

See Common Configuration Options.

Output

See Common Configuration Options. Must have file extension `.lppz`.

[1] Manufacturer part numbers

[2] Bill of materials

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.

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
```

Installation

Binary Releases

Our official LibrePCB binary releases contain the `librepcb-cli` executable next to the GUI application, so usually no separate installation is needed.

MacOS

You need to invoke the CLI with the full path to the binary:

```
/Applications/LibrePCB.app/Contents/MacOS/librepcb-cli --help
```

Linux AppImage

The LibrePCB AppImage also contains the CLI, but since it's a single binary you can't run `librepcb-cli` explicitly. Instead, you have to rename the AppImage to `librepcb-cli` to make it acting as the CLI (or create a symlink):

```
wget "https://download.librepcb.org/releases/2.1.1/librepcb-2.1.1-linux-
x86_64.AppImage"
chmod +x ./librepcb-2.1.1-linux-x86_64.AppImage
mv ./librepcb-2.1.1-linux-x86_64.AppImage ./librepcb-cli
./librepcb-cli --help
```

Docker Image

The easiest way to get the LibrePCB CLI on Linux (especially for usage on CI) is to pull our official Docker image librepcb/librepcb-cli:

```
docker run -it --rm -v `pwd`:/work -u `id -u`:`id -g` \
    librepcb/librepcb-cli:2.1.1 --help
```

Show Help Text

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.
  -v, --verbose  Verbose output.

Arguments:
  command        The command to execute (see list below).

Commands:
  open-library   Open a library to execute library-related tasks.
  open-package   Open a package to execute package-related tasks.
  open-project   Open a project to execute project-related tasks.
  open-step      Open a STEP model to execute STEP-related tasks outside of a library.
  open-symbol    Open a symbol to execute symbol-related tasks.

List command-specific options:
  ./librepcb-cli <command> --help
```

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.
  -v, --verbose  Verbose output.
  --all          Perform the selected action(s) on all elements contained in
                 the opened library.
  --check        Run the library element check, print all non-approved messages
                 and report failure (exit code = 1) if there are non-approved
                 messages.
  --minify-step  Minify the STEP models of all packages. Only works in
                 conjunction with '--all'. Pass '--save' to write the minified
                 files to disk.
  --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).
```

Examples

Check 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 or STEP models. In addition, the library check is run (`--check`) and reports failure if there are any non-approved messages.

Command

```
./librepcb-cli open-library --all --check --minify-step --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
```

Command "open-symbol"

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

Command

```
./librepcb-cli open-symbol --help
```

Output

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

Options:
  -h, --help       Print this message.
  -V, --version    Displays version information.
  -v, --verbose    Verbose output.
  --check          Run the symbol check, print all non-approved messages and
                   report failure (exit code = 1) if there are non-approved
                   messages.
  --export <file>  Export the symbol to a graphical file. Supported file
                   extensions: pdf, svg, bmp, cur, icns, ico, jfif, jpeg, jpg,
                   pbm, pgm, png, ppm, tif, tiff, wbmp, webp, xbm, xpm

Arguments:
  open-symbol      Open a symbol to execute symbol-related tasks.
  symbol           Path to symbol directory (containing *.lp).
```

Command "open-package"

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

Command

```
./librepcb-cli open-package --help
```

Output

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

Options:
  -h, --help       Print this message.
  -V, --version    Displays version information.
  -v, --verbose    Verbose output.
```

```
  --check          Run the package check, print all non-approved messages and
                   report failure (exit code = 1) if there are non-approved
                   messages.
  --export <file>  Export the contained footprint(s) to a graphical file.
                   Supported file extensions: pdf, svg, bmp, cur, icns, ico,
                   jfif, jpeg, jpg, pbm, pgm, png, ppm, tif, tiff, wbmp, webp,
                   xbm, xpm

Arguments:
  open-package     Open a package to execute package-related tasks.
  package          Path to package directory (containing *.lp).
```

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.
  -v, --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.
  --drc                             Run the design rule check, print all
                                    non-approved warnings/errors and report
                                    failure (exit code = 1) if there are
                                    non-approved messages.
  --drc-settings <file>             Override DRC settings by providing a *.lp
                                    file containing custom settings. If not
                                    set, the settings from the boards will be
                                    used instead.
  --run-job <name>                  Run a particular output job. Can be given
                                    multiple times to run multiple jobs.
  --run-jobs                        Run all existing output jobs.
  --jobs <file>                     Override output jobs with a *.lp file
                                    containing custom jobs. If not set, the
                                    jobs from the project will be used instead.
  --outdir <path>                   Override the output base directory of
                                    jobs. If not set, the standard output
```

```
                                  directory from the project is used.
  --export-schematics <file>      [DEPRECATED, REPLACED BY: --run-jobs]
                                  Export schematics to given file(s).
                                  Existing files will be overwritten.
                                  Supported file extensions: pdf, svg, bmp,
                                  cur, icns, ico, jfif, jpeg, jpg, pbm, pgm,
                                  png, ppm, tif, tiff, wbmp, webp, xbm, xpm
  --export-bom <file>             [DEPRECATED, REPLACED BY: --run-jobs]
                                  Export generic BOM to given file(s).
                                  Existing files will be overwritten.
                                  Supported file extensions: csv
  --export-board-bom <file>       [DEPRECATED, REPLACED BY: --run-jobs]
                                  Export board-specific BOM to given file(s).
                                  Existing files will be overwritten.
                                  Supported file extensions: csv
  --bom-attributes <attributes>   [DEPRECATED, REPLACED BY: --run-jobs]
                                  Comma-separated list of additional
                                  attributes to be exported to the BOM.
                                  Example: "SUPPLIER, SKU"
  --export-pcb-fabrication-data   [DEPRECATED, REPLACED BY: --run-jobs]
                                  Export PCB fabrication data
                                  (Gerber/Excellon) according the fabrication
                                  output settings of boards. Existing files
                                  will be overwritten.
  --pcb-fabrication-settings <file>  [DEPRECATED, REPLACED BY: --jobs] 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.
  --export-pnp-top <file>         [DEPRECATED, REPLACED BY: --run-jobs]
                                  Export pick&place file for automated
                                  assembly of the top board side. Existing
                                  files will be overwritten. Supported file
                                  extensions: csv, gbr
  --export-pnp-bottom <file>      [DEPRECATED, REPLACED BY: --run-jobs]
                                  Export pick&place file for automated
                                  assembly of the bottom board side. Existing
                                  files will be overwritten. Supported file
                                  extensions: csv, gbr
  --export-netlist <file>         [DEPRECATED, REPLACED BY: --run-jobs]
                                  Export netlist file for automated PCB
                                  testing. Existing files will be
                                  overwritten. Supported file extensions:
                                  d356
  --board <name>                  The name of the board(s) to export. Can be
                                  given multiple times. If not set, all
                                  boards are exported.
  --board-index <index>           Same as '--board', but allows to specify
                                  boards by index instead of by name.
  --remove-other-boards           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.
  --variant <name>                  The name of the assembly variant(s) to
                                    export. Can be given multiple times. If not
                                    set, all assembly variants are exported.
  --variant-index <index>           Same as '--variant', but allows to specify
                                    assembly variants by index instead of by
                                    name.
  --set-default-variant <name>      Move the specified assembly variant to the
                                    top before executing all the other actions.
                                    Pass '--save' to save the modified project
                                    to disk.
  --save                            Save project before closing it (useful to
                                    upgrade file format).
  --strict                          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.

Arguments:
  open-project                      Open a project to execute project-related
                                    tasks.
  project                           Path to project file (*.lpp[z]).
```

Examples

Run ERC, DRC and Output Jobs

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

Command

```
./librepcb-cli open-project --erc --drc --run-jobs MyProject.lpp
```

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"
Run DRC...
  Board 'default':
    Approved messages: 0
```

```
    Non-approved messages: 5
      - [ERROR] Clearance copper □ hole < 0.25 mm
      - [ERROR] Clearance copper □ hole < 0.25 mm
      - [ERROR] Clearance drill □ drill < 0.35 mm
      - [ERROR] Clearance plane □ board outline < 0.3 mm
      - [ERROR] Clearance plane □ board outline < 0.3 mm
Run output job 'Schematic PDF'...
  => 'output/v1/MyProject_v1_Schematic.pdf'
Run output job 'Gerber/Excellon'...
  => 'output/v1/gerber/MyProject_v1_DRILLS-NPTH.drl'
  => 'output/v1/gerber/MyProject_v1_DRILLS-PTH.drl'
  => 'output/v1/gerber/MyProject_v1_OUTLINES.gbr'
  => 'output/v1/gerber/MyProject_v1_COPPER-TOP.gbr'
  => 'output/v1/gerber/MyProject_v1_COPPER-BOTTOM.gbr'
  => 'output/v1/gerber/MyProject_v1_SOLDERMASK-TOP.gbr'
  => 'output/v1/gerber/MyProject_v1_SOLDERMASK-BOTTOM.gbr'
  => 'output/v1/gerber/MyProject_v1_SILKSCREEN-TOP.gbr'
  => 'output/v1/gerber/MyProject_v1_SILKSCREEN-BOTTOM.gbr'
  => 'output/v1/gerber/MyProject_v1_SOLDERPASTE-TOP.gbr'
  => 'output/v1/gerber/MyProject_v1_SOLDERPASTE-BOTTOM.gbr'
Finished with errors!
```

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

Command "open-step"

This command opens a STEP file and lets you execute some tasks with it.

Command

```
./librepcb-cli open-step --help
```

Output

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

Options:
  -h, --help        Print this message.
  -V, --version     Displays version information.
  -v, --verbose     Verbose output.
  --minify          Minify the STEP model before validating it. Use in
                    conjunction with '--save-to' to save the output of the
                    operation.
  --tesselate       Tesselate the loaded STEP model to check if LibrePCB is
                    able to render it. Reports failure (exit code = 1) if no
                    content is detected.
  --save-to <file>  Write the (modified) STEP file to this output location (may
```

```
                    be equal to the opened file path). Only makes sense in
                    conjunction with '--minify'.

Arguments:
  open-step         Open a STEP model to execute STEP-related tasks outside of
                    a library.
  file              Path to the STEP file (*.step).
```

Examples

Minify & Validate STEP File

Command

```
./librepcb-cli open-step --minify --tesselate --save-to minified.step model.step
```

Output

```
Open STEP file 'model.step'...
Perform minify...
 - Minified from 512,464 bytes to 313,374 bytes (-39%)
Save to 'out.step'...
Load model...
Tesselate model...
 - Built 17616 vertices with 2 different colors
SUCCESS
```

Library Conventions

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

These guidelines are not yet complete. Help us create sensible conventions on GitHub!

Common Conventions

These guidelines are not yet complete. Help us create sensible conventions on GitHub!

The conventions described on this page apply to all kinds of library elements (symbols, packages, categories, ...) unless explicitly specified differently in the conventions of the corresponding library element type.

Name

Generally, the capitalization of library element names should be Title Case. If this is not suitable in certain cases (e.g. *Schottky Diodes Dual in Series*), the corresponding warning from the library check can be approved.

Special Rules:

- **Component categories:** Shall be in **plural form**, not singular (e.g. *Diodes*, not *Diode*).
- **Package categories:** Shall be in **singular form**, not plural (e.g. *Chip Resistor*, not *Chip Resistors*).
- **Packages:** Follow IPC-7351 conventions (see details in Package Naming).

Description

If the description of a library element is very obvious from its name (e.g. a symbol named *Diode*), the description should be kept empty. If a description is needed, its first line (if multiple) should be a short, expressive summary about the library element (e.g. *General-purpose 1A/400V rectifier diode in DO-41 package*). Any sentence or summary shall end with a period.

Keywords

- Keywords are **all lowercase, comma-separated, without spaces after the comma**. Example: *"soic8,schmitt trigger,inverted"*
- Do not add keywords which are already contained in the library elements name. For example a symbol called *Schmitt Trigger* shall not have the keyword *"schmitt trigger"*.
- To decide about keywords, just ask yourself which terms a user might type to find that library element, and are not already contained in its name.

Author

Every library element should have an author. It is allowed to use nicknames instead of real names (ideally, it should correspond to your GitHub username then).

When making significant changes to an existing library element created by someone else, it is allowed/recommended to add your name to the author field too, separated by a comma and a space. Example: *"Author 1, Author 2"*

Version

- New elements should have the version number 0.1.
- Every change to a library element shall increase its version number.
  - Metadata-only changes (i.e. information in the *Metadata* tab/panel):
    Bump the third digit (e.g. 0.1 → 0.1.1, or 1.1 → 1.1.1).
  - Functional changes (graphics, pin names, MPNs, ...):
    Bump the second digit (e.g. 0.1 → 0.2, or 1.1 → 1.2)
  - Huge refactorings / complete rewrites (but still no breaking changes):
    Bump the first digit (e.g. 0.1 → 1.0, or 1.2 → 2.0). This is almost never done.

Breaking changes on existing library elements are never allowed! Whenever the library editor shows a warning about breaking changes, create a new library element instead of breaking an existing one. If the existing element shall not be used anymore then, mark it as deprecated.

Only bump the version number of library elements you have modified. The version number of the library which contains that element shall not be bumped, unless you made changes to the library itself. Generally, only library maintainers bump the verison number of libraries.

Symbol Conventions

These guidelines are not yet complete. Help us create sensible conventions on GitHub!

Generic vs. Specific

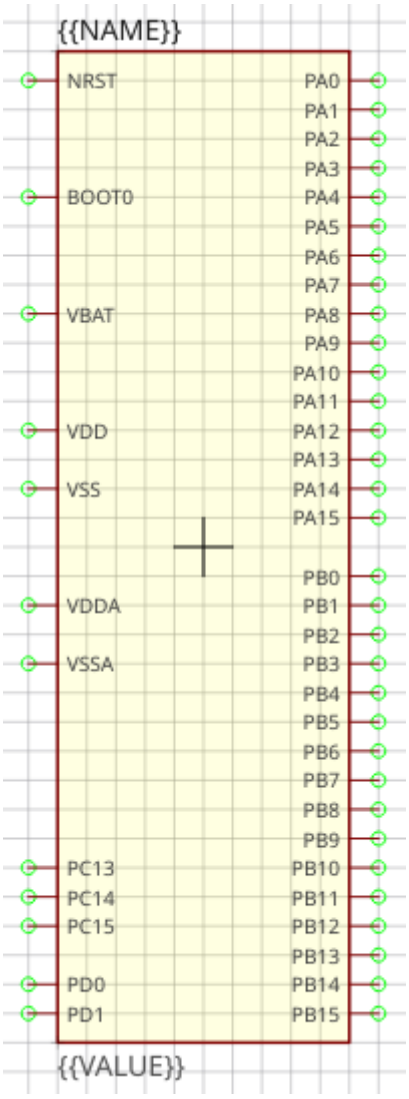
Generic components should have generic symbols. For example a diode (let's say *1N4007*) doesn't need its own symbol, a generic diode symbol is fine. So you should name it something like "Diode" and use the same symbol also for all other standard diodes. Of course every kind of diode (e.g. Zener) should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

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)



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.

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.

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.

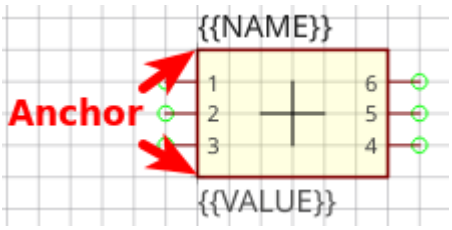
Pin Naming

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.

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.

Typical text element properties

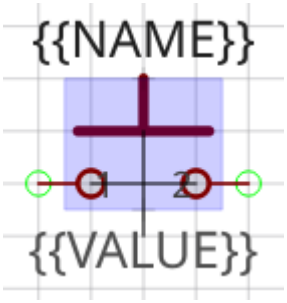
Property	Name text element	Value text element
Layer	*Names*	*Values*
Text	*{{NAME}}*	*{{VALUE}}*

Property	Name text element	Value text element
Alignment	*Bottom Left*	*Top Left*
Height	*2.5mm*	*2.5mm*
Rotation	*0°*	*0°*

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.

Package Conventions

These guidelines are not yet complete. Help us create sensible conventions on GitHub!

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 2. 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 3. 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 4. 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.

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 in capital letters, followed by an underscore (e.g. "MOLEX_53261-06"). The Library Expert naming

conventions contain concrete recommendations for many manufacturers, please follow them. *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.*

Pads

- **Each lead of the package should be represented by a separate package pad**, even if there are internal connections (e.g. multiple GND leads) or unconnected leads (e.g. the often unused metal tab of a TO220). Exceptions:
  - Multiple mechanical leads which have a package-internal connection (e.g. tabs of the metal housing of an USB connector) should be represented by a single package pad (and all footprint pads connected to it).
  - Leads for pure mechanical purpose without any internal connection at all (e.g. split solder tabs of a plastic connector) shouldn't be added as package pads (the corresponding footprint pads can be left unconnected). If in doubt or if connected to a metal cover of significant size (possibly having a shielding function), treat them like normal, electrically relevant leads.
- **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.

Footprints

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

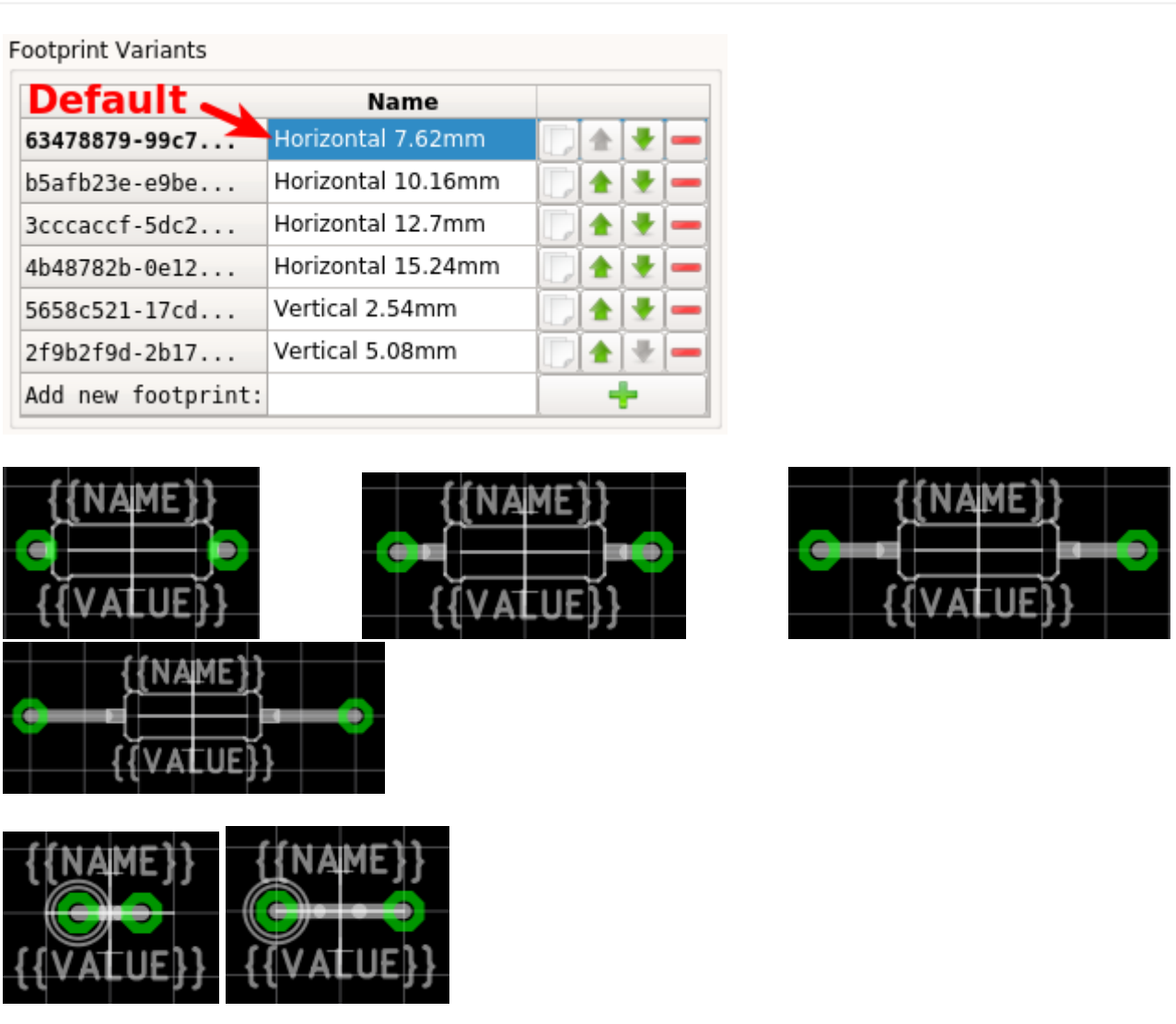
combinations of them should be added. For example a package which should have all three density levels as defined in IPC-7351 and can be mounted either vertically or horizontally would need six footprint variants to support all possible use-cases.

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 5. THT resistor 0207 footprint variants



Origin

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

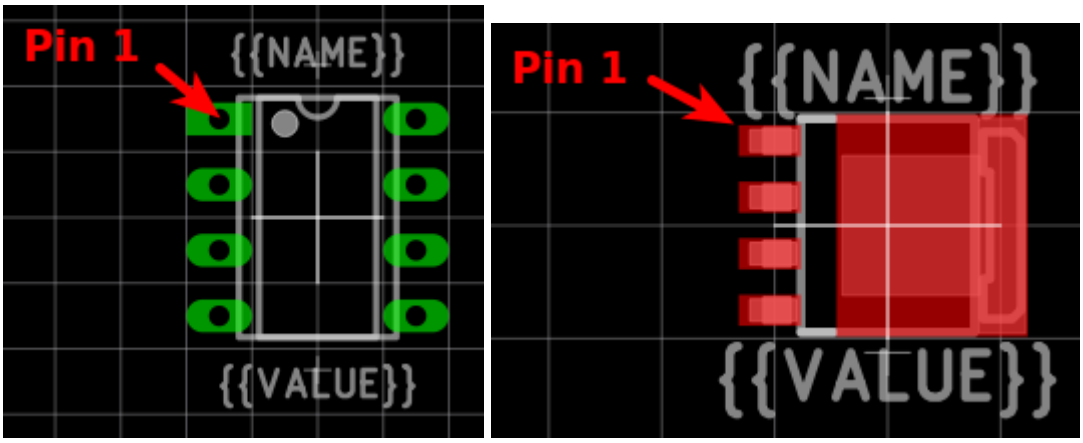
Some packages (especially those with non-symmetrical body) have the origin explicitly specified in the datasheet. In that case, use the origin from the datasheet.

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 6. Footprint orientation examples



Legend Layer

In LibrePCB 0.1.x, these layers were called *Top/Bottom Placement*. Starting with LibrePCB 1.0, they are now called *Top/Bottom Legend*.

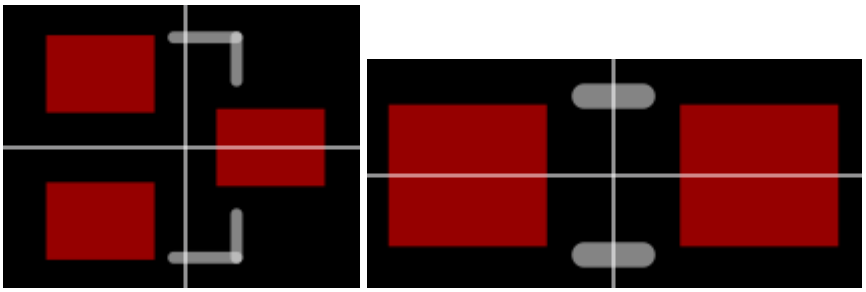
The *Top Legend* layer is intended to be printed on silkscreen and thus should contain information required for assembling the PCB. But don't put too many things on that layer as it would waste space on the PCB!

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 legend 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 legend 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 7. Legend layer examples (only legend and copper layers shown)



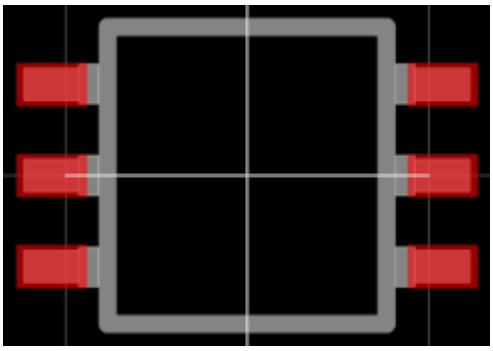
Documentation Layer

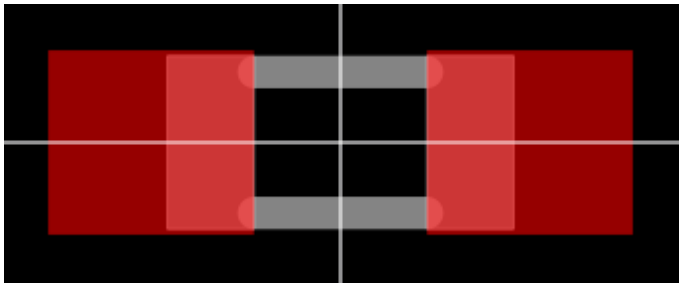
The layer *Top Documentation* should be used to draw the most important details of the package's appearance. It could be considered as an alternative to the 3D model of a package. But in contrast to the 3D model, the documentation layer is visible in the board editor while layouting the PCB.

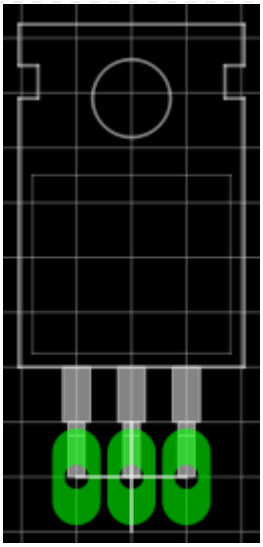
Following things should be placed on the documentation layer:

- **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 8. Documentation layer examples (only documentation and copper layers shown)







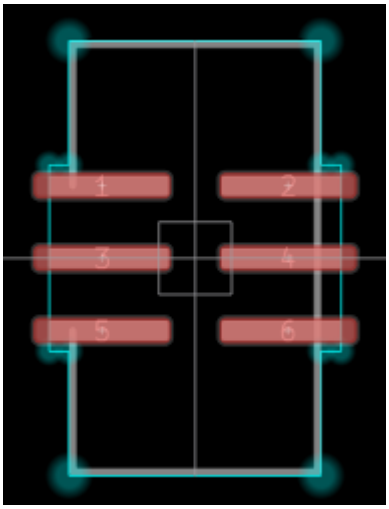
Package Outlines Layer

Every typical footprint should contain a single polygon on the *Top Package Outlines* layer to specify the outer dimension of the package. It is used by the DRC to check the clearance between devices.

General rules:

- **Any leads shall be included**, but pads not.
- **Line width:** 0.0mm

Example 9. Package outlines layer examples (the line in cyan)



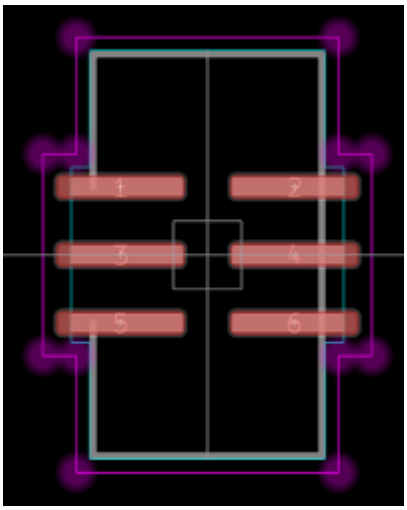
Courtyard Layer

Every typical footprint should contain a single polygon on the *Top Courtyard* layer to specify the area where no other device shall be placed. It is used by the DRC to check this requirement. Usually this is equal to the Package Outlines Layer, just with an offset of several 0.1mm.

General rules:

- **Line width:** 0.0mm
- **Offset to outlines:** According to IPC 7351 if applicable. A typical value for SMT devices is 0.2mm. For THT devices, a larger value (e.g. 0.4mm) is recommended.

Example 10. Courtyard layer examples (the line in magenta)



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.

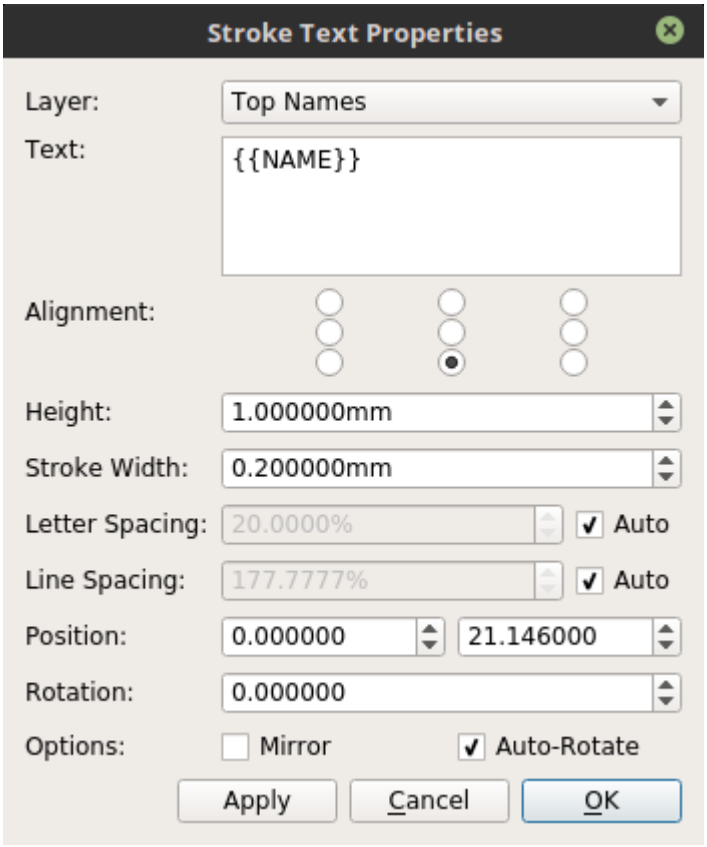


Figure 1. Typical footprint name properties

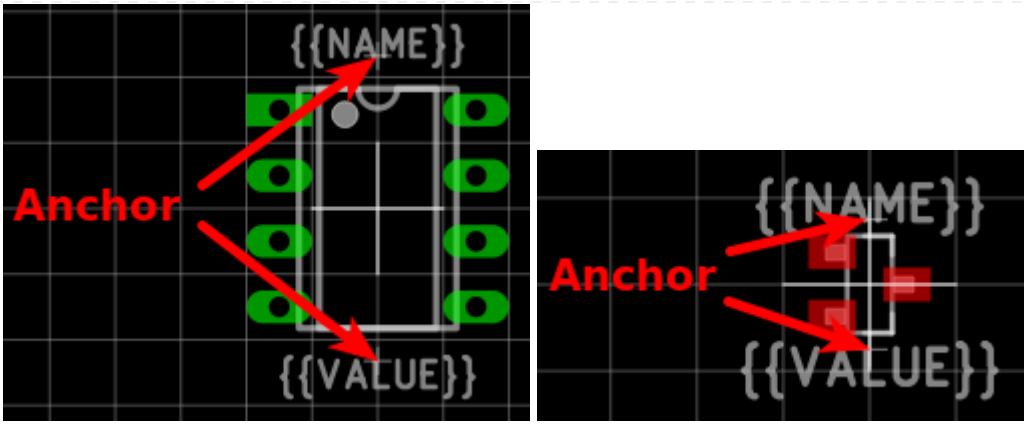
Typical text element properties

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 11. Footprint text element examples



3D Models

Packages might be populated with 3D models from STEP files. However, there are several things to consider carefully.

Some general notes:

- **File size:** Try to keep STEP models as small as possible to avoid unnecessary long download- and loading times. Usually it is fine to keep STEP models rather simple (i.e. not adding too much details).
- **License:** Keep in mind that all libraries provided by LibrePCB are released under the CC0 Public Domain license. This also applies to STEP models.

Almost every STEP model available in the Internet (whether from a manufacturer or some other website) are not published under the CC0 license and sometimes are also very bloated (way too detailed). Such models must not be contributed to our official libraries (we won't accept them). We may change this requirement some day, but at the moment this needs to be respected.

In addition, we prefer STEP models to be generated with CadQuery to allow making modifications in future. Contributions of STEP models created in any other way may not be accepted.

Troubleshooting

In case you encounter any problems with LibrePCB, this chapter gives you some tips to get them solved or to get help from the community.

Error: There is no element of type 'X' with the UUID 'Y'

Whenever an error message pops up with a message like *There is no element of type 'X' with the UUID 'Y'* or something like *Element with UUID 'X' not found*, it is very very very likely that you made some breaking change in a library element. Those errors then occur either in the component- or device editor, or after updating the project library with the *Project Library Updater*.

Since these are complex issues, the correct solution depends on a case-by-case basis. All we can give here are some general recommendations:

1. Read UUID References until you understand it.
2. From now on, follow the recommendations on that page to avoid running into this problem again in future.
3. If you have any idea in *which* of the library elements you (recently) made breaking changes, delete them completely. It may hurt, but it's often the only way out of the mess. You may want to back up the library elements before deletion — use the **Device › Show in File Manager** menu item to locate the directory to back up.
4. If the project library update still fails after deleting the suspicious library elements, you may need to remove the suspicious elements also from the schematics & boards, and then re-add them the normal way with the *Add Component* dialog.
5. In important/urgent cases, you may ask LibrePCB developers for help. However, we kindly ask for a donation as a compensation for our invested time.

Project Library Update Fails

See Error: There is no element of type 'X' with the UUID 'Y'.

Workspace Sync (Dropbox, Cloud, Git, ...)

If you sync your LibrePCB workspace with a cloud or similar, it's important to follow some rules to avoid problems:

- Exclude all files with pattern `cache_*` from the synchronization. These files are stored in the workspace subdirectory `data/libraries/`. If LibrePCB does not work correctly and you had these files synced, delete those files manually (while LibrePCB is closed) and try again. These files are automatically recreated after deletion.
- Consider excluding files named `.lock` from the synchronization too if you experience problems with file locks. However, never open a project or library at the same time from multiple computers!

- Hidden files ("dotfiles") must be synchronized. If hidden files are ignored by the sync, LibrePCB will not work correctly.

Note that even when following these rules, it's still not guaranteed that everything works correctly. Especially with clouds the problem is that they are not operating "atomically", which can cause very serious troubles. Therefore we do not recommend to store any LibrePCB files in a cloud.

Working with a version control system like Git however is fine, since it works atomically and even allows to roll back a change in the very unlikely case something is messed up. We just recommend to use version control per-project and per-library, not for the whole workspace.

Wayland

There are some known issues when using LibrePCB natively on Wayland. If you experience any problems, please try XWayland or X11 (both should work fine).

Slow/Laggy UI

On some systems, especially with large projects, the UI could get a bit laggy. We are aware of this and try to improve it. In the mean time, try the following things:

- Reduce grid density or disable grid completely
- Avoid huge schematics — split them into multiple sheets (e.g. DIN A4 format)
- In the schematic editor, hide pin numbers (toggle **View › Show Pin Numbers**)
- In the board editor, reduce the number of visible layers
- Enable or disable OpenGL in workspace settings (test both modes)
- If using a high-resolution display, try to reduce resolution
- On a laptop, plug in the charger :-)

Logging Output

For various kinds of problems, it helps to see what LibrePCB internally does and what low-level errors occurred but aren't displayed (in every detail) in the graphical user interface.

Those messages are always written to `stderr` — To see them, just run LibrePCB from a terminal. Note that on Windows you have to redirect `stderr` to a file and open the file in Notepad afterwards.

Windows cmd.exe (PowerShell doesn't work!)

```
"C:\Program Files\LibrePCB\bin\librepcb.exe" > log.txt 2>&1
```

After closing LibrePCB, open `C:\Users\%USERNAME%\log.txt` in Notepad.

MacOS Terminal

```
/Applications/LibrePCB.app/Contents/MacOS/librepcb
```

Linux/UNIX Shell

```
/path/to/librepcb
```

Currently this is the only way to get logging messages. Logging to a file is not implemented yet, and the verbosity cannot be configured.

Example Output

```
./librepcb-1.1.0-linux-x86_64-qt6.AppImage
[  INFO   ] LibrePCB 1.1.0 (18a3d4589)
[  INFO   ] Qt version: 6.6.2 (compiled against 6.6.2)
[  INFO   ] Resources directory: "/tmp/.mount_librepHOglKc/opt/share/librepcb"
[  INFO   ] Application settings: "/home/user/.config/LibrePCB/LibrePCB.ini"
[  INFO   ] Cache directory: "/home/user/.cache/LibrePCB/LibrePCB"
[DEBUG-MSG] Network access manager thread started.
[DEBUG-MSG] Recently used workspace: "/home/user/LibrePCB-Workspace"
[DEBUG-MSG] Detected light theme based on window background color #efefef.
[DEBUG-MSG] Open workspace data directory "/home/user/LibrePCB-Workspace/data"...
[DEBUG-MSG] Load workspace settings...
[DEBUG-MSG] Successfully loaded workspace settings.
[DEBUG-MSG] Load workspace library database...
[DEBUG-MSG] Successfully loaded workspace library database.
[DEBUG-MSG] Successfully opened workspace.
[DEBUG-MSG] Workspace library scanner thread started.
[  INFO   ] Loaded parts information cache from
"/home/user/.cache/LibrePCB/LibrePCB/parts.lp".
[DEBUG-MSG] Cleaned outdated live information about 0 parts.
[DEBUG-MSG] Start workspace library scan in worker thread...
[DEBUG-MSG] Workspace libraries indexed: 47 libraries in 25 ms.
[DEBUG-MSG] Workspace library scan succeeded: 5515 elements in 12876 ms.
[DEBUG-MSG] Network access manager thread stopped.
[DEBUG-MSG] Workspace library scanner thread stopped.
[DEBUG-MSG] Exit application with code 0.
```

Reporting Problems

If you like to report a problem or you want to ask for help, choose one of the ways listed here.

Generally the discussion forum is always a good place to ask. A GitHub issue is preferred for bug reports, but only if it's clearly a bug — otherwise ask in the forum first.

For any problem report, please include as much details as possible! Many problems are platform-specific, deployment-specific, usecase-specific etc. and

we're all not clairvoyants so please let us know these details.

System Information

If you are able to run LibrePCB, open the *About LibrePCB* dialog and **copy the whole text** from the *Details* tab into your report. This is very important information.

If you are not able to run LibrePCB, please let us know the following information:

- Operating system & version
- CPU Architecture (x86, x86_64, ARM, Apple Silicon, …)
- LibrePCB version (MAJOR.MINOR.PATCH)
- On Linux: X11 or Wayland?

Installation Method

How did you install LibrePCB? Installer, Portable, Snap, Self-built, …?

Steps to Reproduce

What did you do before the problem occurred? Describe **exactly**, step-by-step, what you did before the problem occurred.

Problem Description

Describe **exactly** what happened. Which error messages occurred? How did LibrePCB behave?

Logging Messages

For technical problems, it can be helpful to also include all logging messages.

Development

For developers of LibrePCB, or of you're interested in technical details of LibrePCB, check out the developers documentation at developers.librepcb.org.