LibrePCB Documentation

2023-08-03

Table of Contents

Welcome to the documentation of LibrePCB 0.1.7!

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

Offline Documentation

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

Chapters:

- Installation
- Quickstart Tutorial
- User Manual
- CLI Reference
- Library Conventions
- Development

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 know from the following packages:

- Arch Linux AUR Package (builds from source)
- NixOS Package
- Gentoo Package
- Homebrew Cask Package
- OpenPandora Package

> 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

Online Installer

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

- Installs a maintenance tool to easily download and install updates.
- Creates start menu entries for LibrePCB and the maintenance tool.

- Optionally registers `*.lpp` files, so LibrePCB projects can be opened with a double-click in the file manager.

Just download and run [librepcb-installer-0.1.7-windows-x86.exe](#)**.** Afterwards you'll find LibrePCB in your start menu.

Portable Package

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

Download and extract [librepcb-0.1.7-windows-x86.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](#) (or [Online Installer](#))

Online Installer (x86_64)

Although an installer is a rather unusual way to install applications on Linux, it provides the advantage that `*.lpp` project files can (optionally) be registered with the LibrePCB application, so you can open LibrePCB projects with a double-click in the file manager.

Download [librepcb-installer-0.1.7-linux-x86_64.run](#)**, make it executable and run it:**

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

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.

Portable AppImage (x86_64)

The AppImage is a single-file portable package which runs on most Linux distributions. It does not install anything on your system and `*.lpp` project files will not be registered with the LibrePCB application.

Download [librepcb-0.1.7-linux-x86_64.AppImage](#), make it executable and run it:

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

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.

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
```

> By default, Snap applications do not have access to the file system outside your home folder. Depending on where you want to store LibrePCB files, you might need to grant this permission manually after installing LibrePCB.
>
> Within *Ubuntu Software*, this is as simple as clicking on **[Permissions]** and enabling *Read/write files on removable storage devices*.
>
> From the terminal, run this command:
>
> ```
> sudo snap connect librepcb:removable-media :removable-media
> ```

> 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 [FlatPak setup](#), you can configure FlatHub and install LibrePCB as follows:

> **!** After installing FlatPak, 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
```

On macOS

Portable Package

The recommended way to install LibrePCB is to download the portable `*.dmg` file:

Download [librepcb-0.1.7-mac-x86_64.dmg](#) and double-click it in Finder. Then drag and drop the app onto the "Applications" folder in Finder. Afterwards you'll find LibrePCB in the Launchpad.

Online Installer

Alternatively you could install LibrePCB with the online installer. This also installs a maintenance tool which allows to update LibrePCB once a new release is available.

Download [librepcb-installer-0.1.7-mac-x86_64.dmg](#). Right-click on the downloaded file and choose **[Open]** from the context menu (because the application is considered as *unverified*). Then double-click on the contained file to run the installer.

Build From Sources

Requirements

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

- g++ >= 4.8, MinGW >= 4.8, or Clang >= 3.3 (C++11 support is required)
- [Qt](#) >= 5.5
- [zlib](#)
- [OpenSSL](#)
- [CMake](#) 3.5 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: `0.1.7`) 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-0.1.7-source.zip

```
wget "https://download.librepcb.org/releases/0.1.7/librepcb-0.1.7-source.zip"
unzip ./librepcb-0.1.7-source.zip
cd ./librepcb-0.1.7
```

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 in future it will also be usable with different LibrePCB versions.

You can just accept the default workspace location (you could still move it to another location afterwards, if desired):
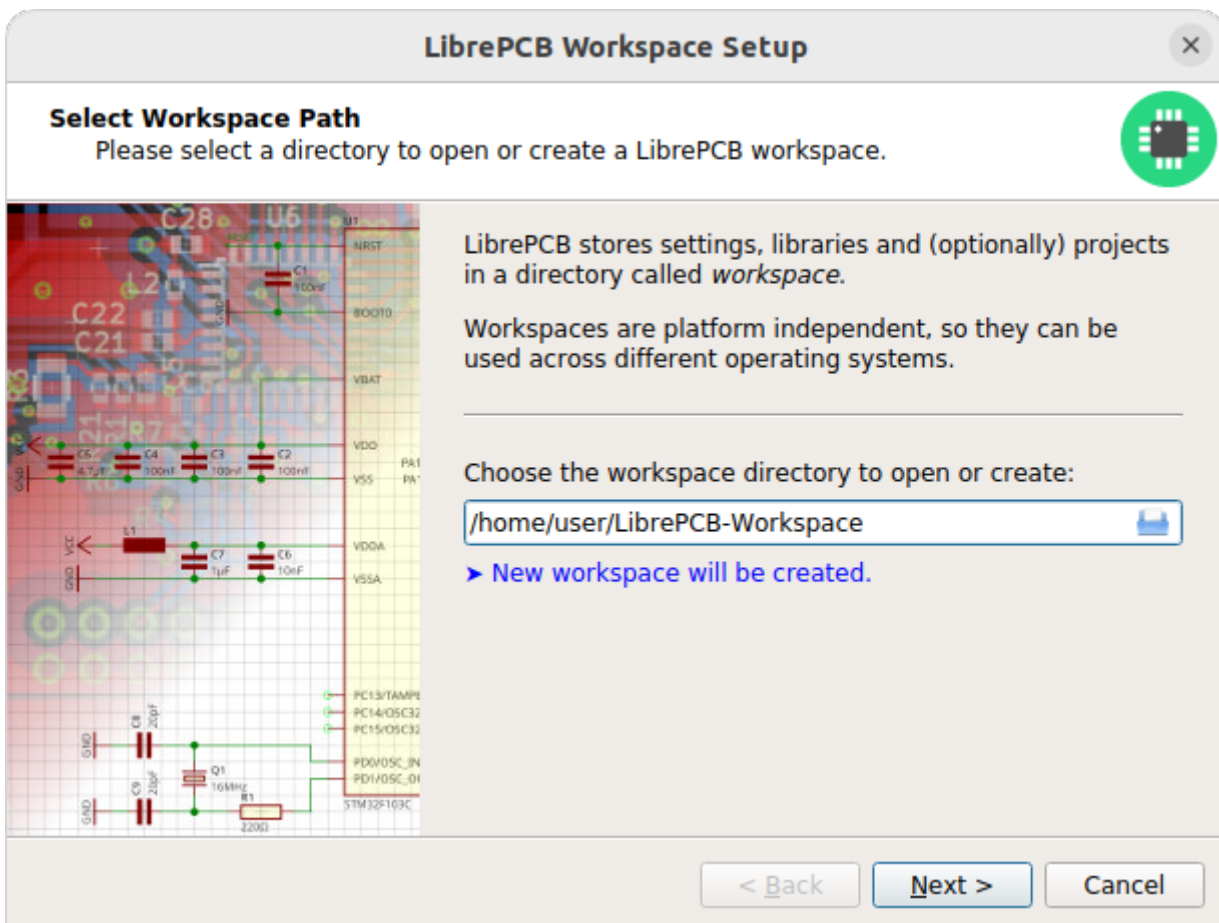
If the selected path does not contain a workspace yet, clicking on **[Next]** will show a page to choose the most important settings:
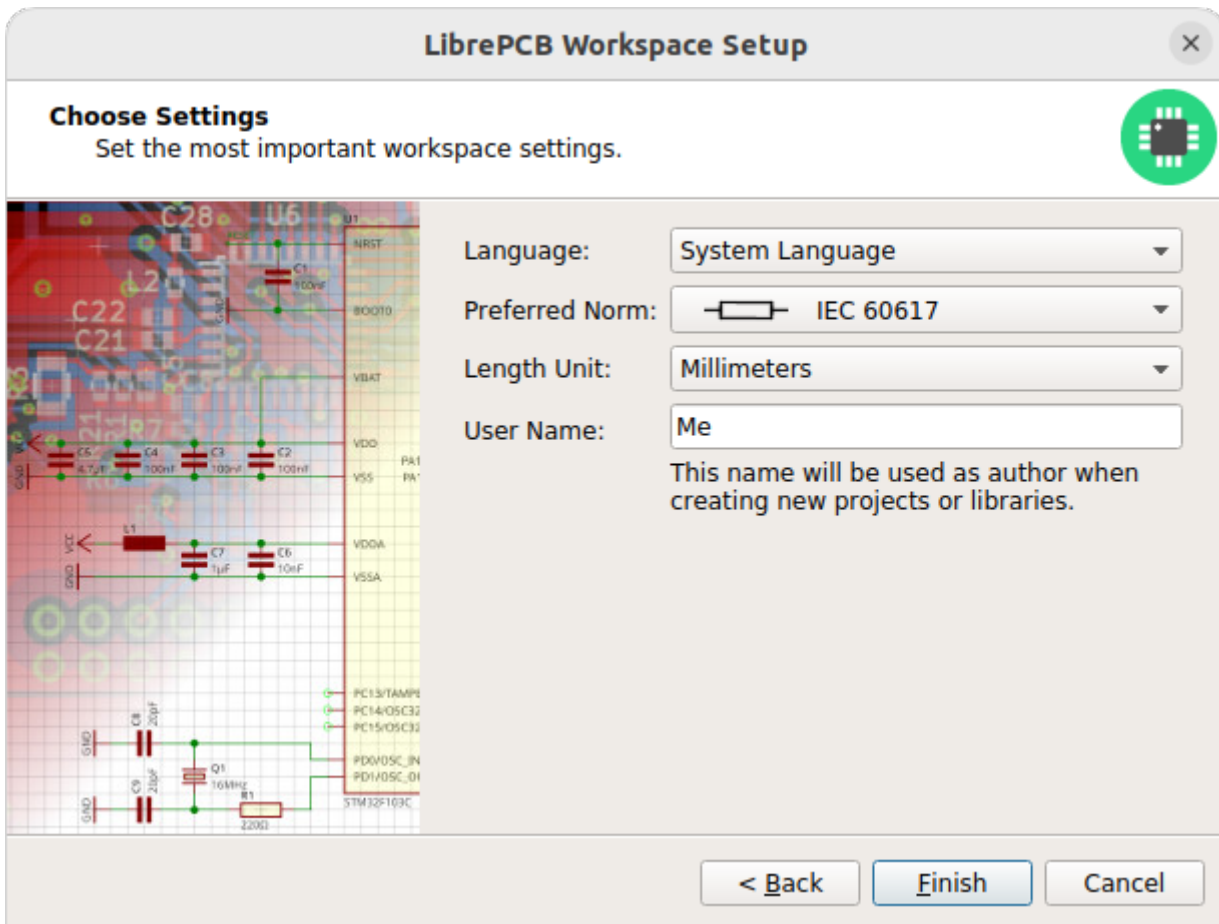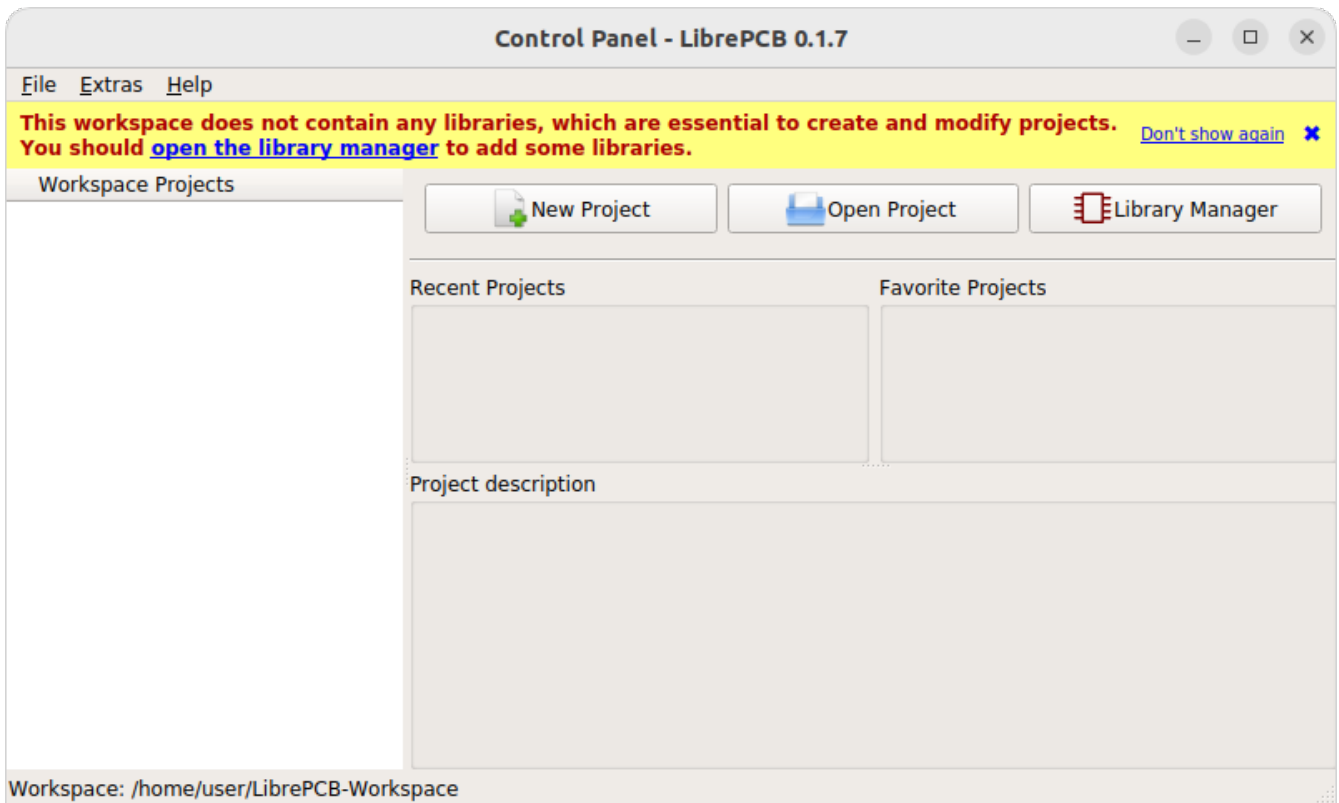
It is recommended to select at least your **preferred norm** and **length unit** since these usually depend on where you're living.

> You can change these settings at any time later in the control panel under **Extras › Workspace Settings**.

After clicking **[Finish]**, the control panel 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 **[Library Manager]** in the control panel:
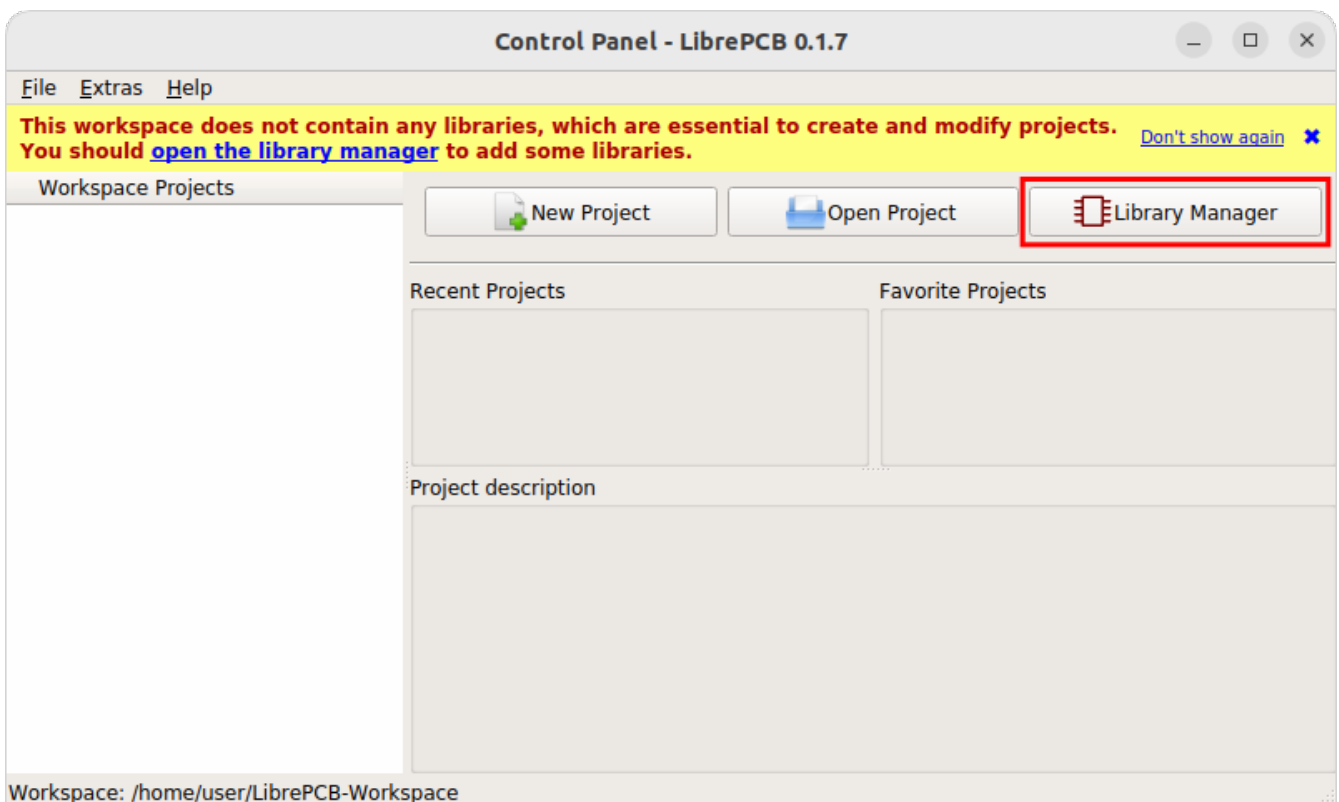
The library manager immediately fetches the list of available libraries from the Internet. Most of these libraries are hosted at github.com/LibrePCB-Libraries.

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:
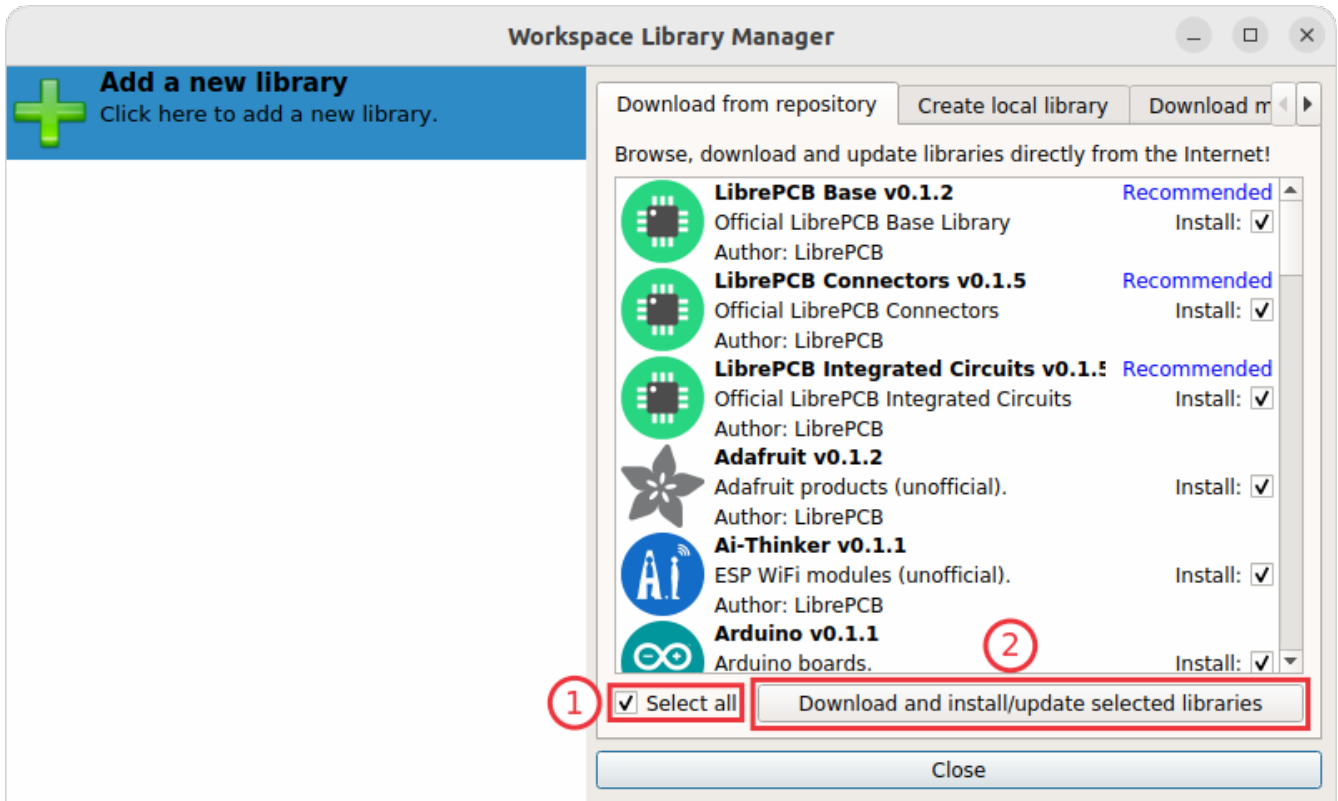
Later you can keep the installed libraries up to date exactly the same way. Just open the library manager from time to time to see which libraries can be updated to a new version.

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 `v0.1/libraries/local/` in your workspace.

After the selected libraries have been downloaded, they will appear in the list of installed libraries on the left side of the library manager:
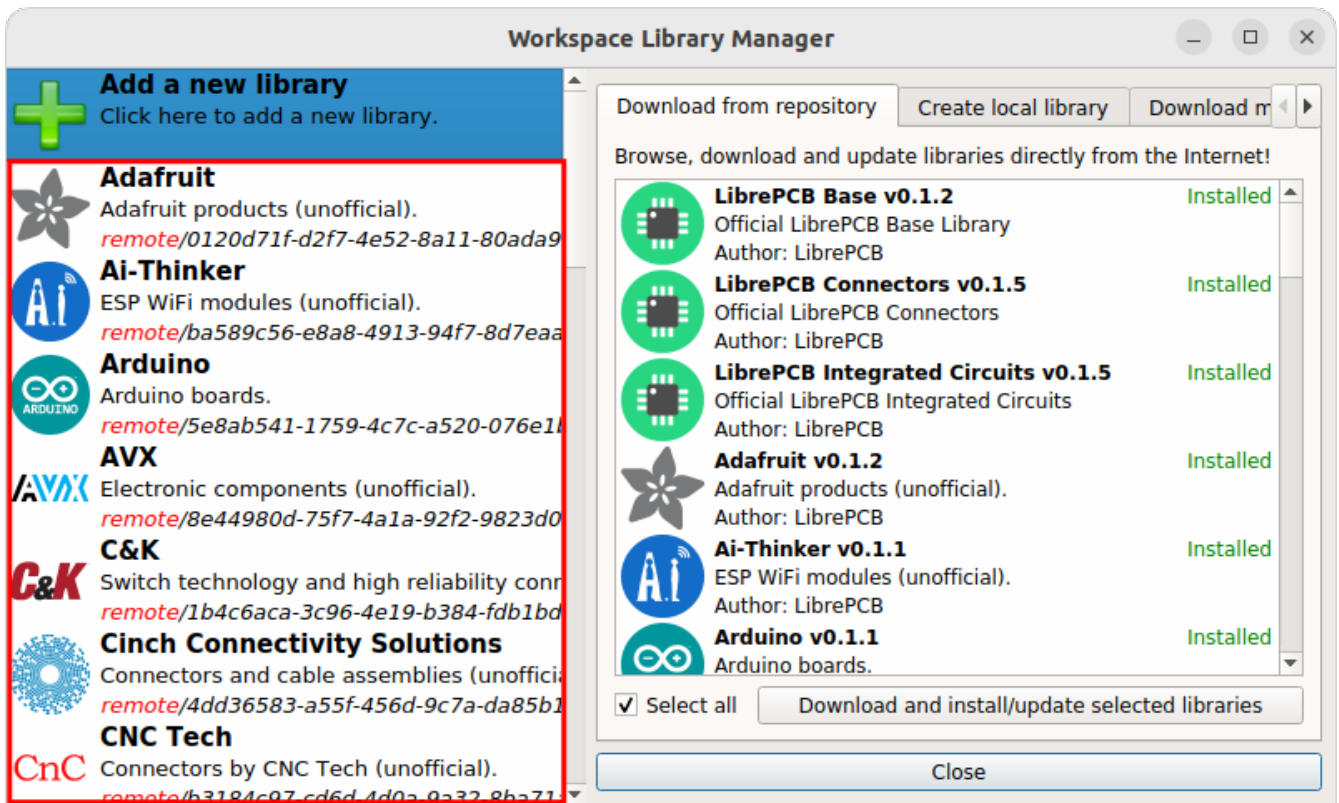
Note that after the libraries were installed, it takes a moment to create an index of all the contained elements. This process automatically runs in background and is indicated with a progress bar at the bottom right of all main windows. The installed libraries are ready to use once the progress bar 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, go to tab *Create local library* tab, optionally enter some metadata (default values are good enough) and click on **[Create Library]**:
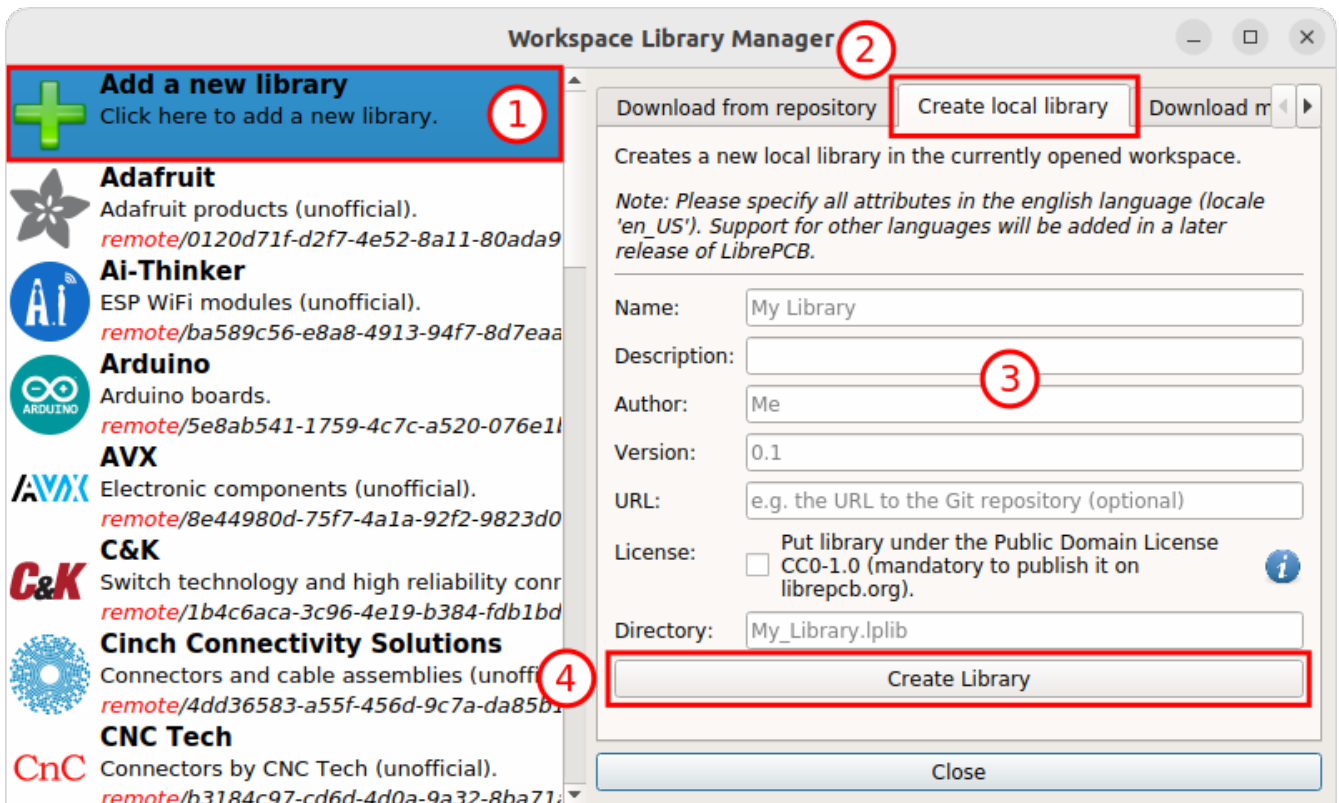
If you're curious how the library looks like, select your library on the left and then click on **[Open Library Editor]** (or just double-click on your library):
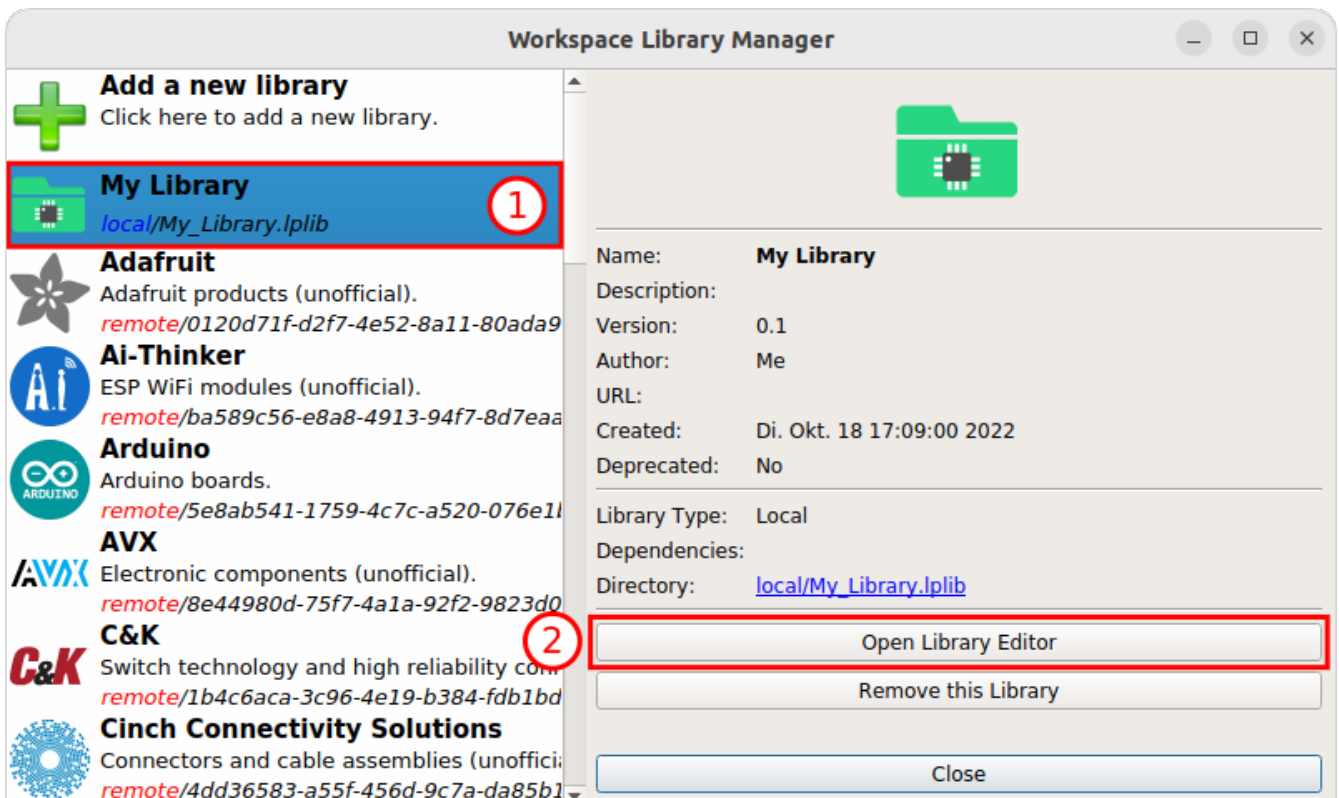

You'll see an empty library editor since the library doesn't contain any elements yet.

Your workspace setup is now complete and ready to start creating your first PCB project! You can close both the library editor and the library manager for now. 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 control panel:
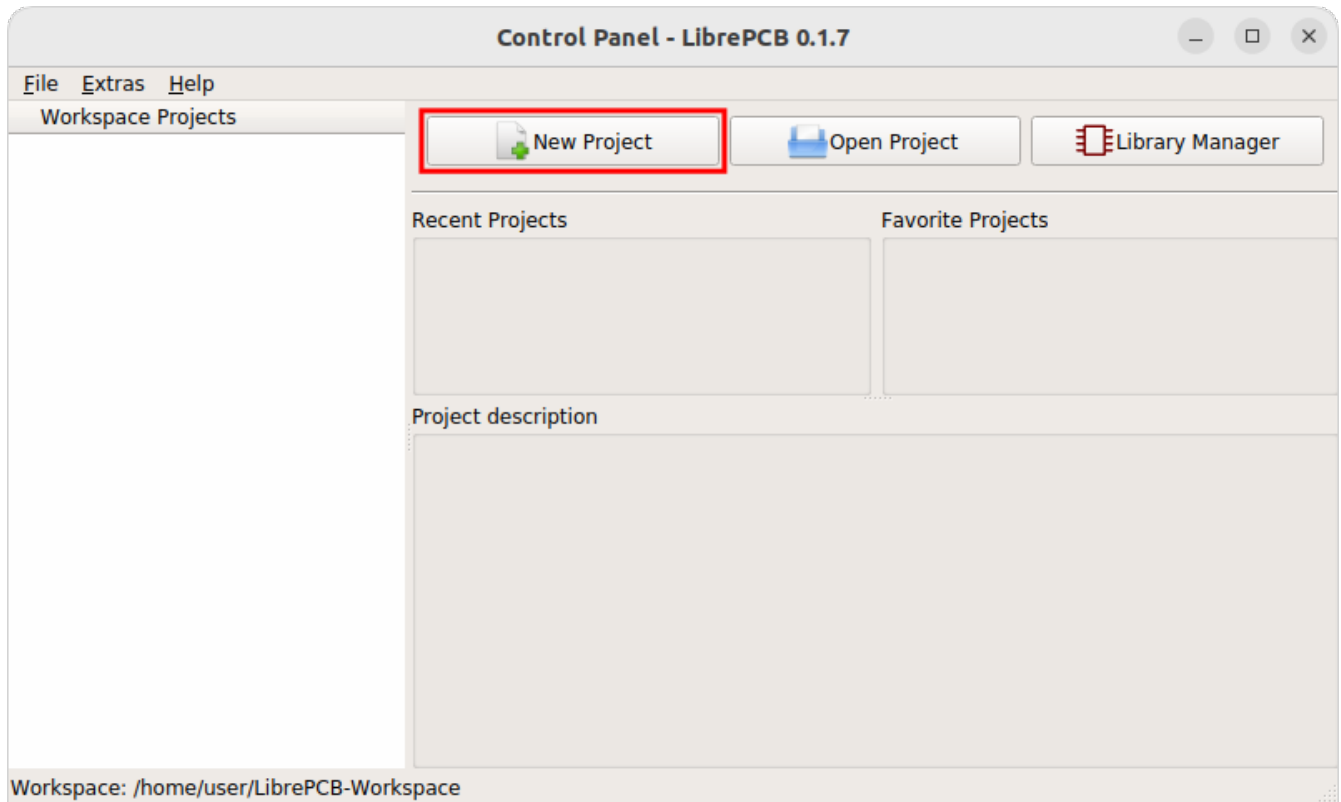


Then specify some project metadata:
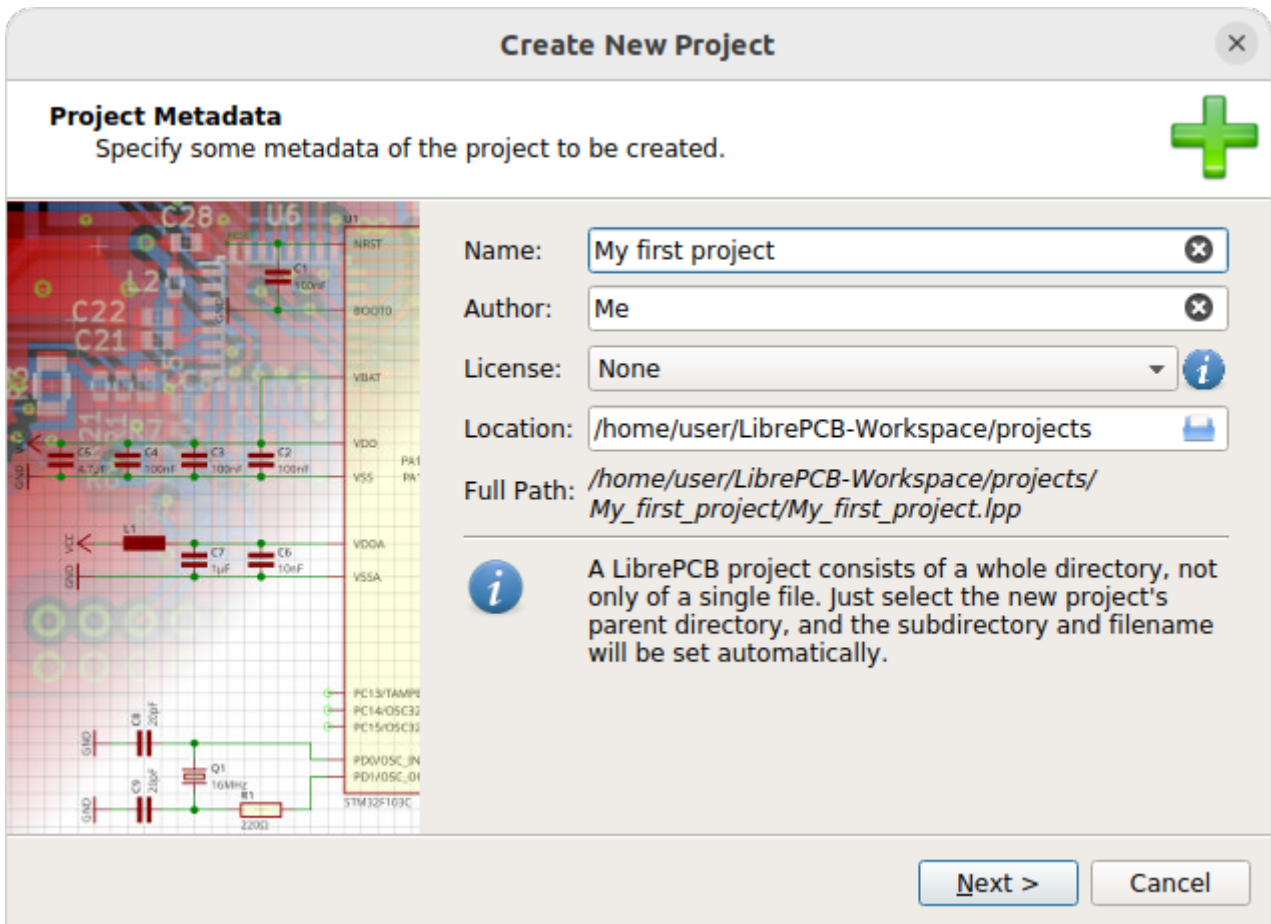
It's recommended to store projects within the workspace subdirectory named `projects` (the default location suggested by the wizard) because these projects are then shown in the control panel file explorer, making them easy to locate and use. But of course projects can be created at any other location 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:
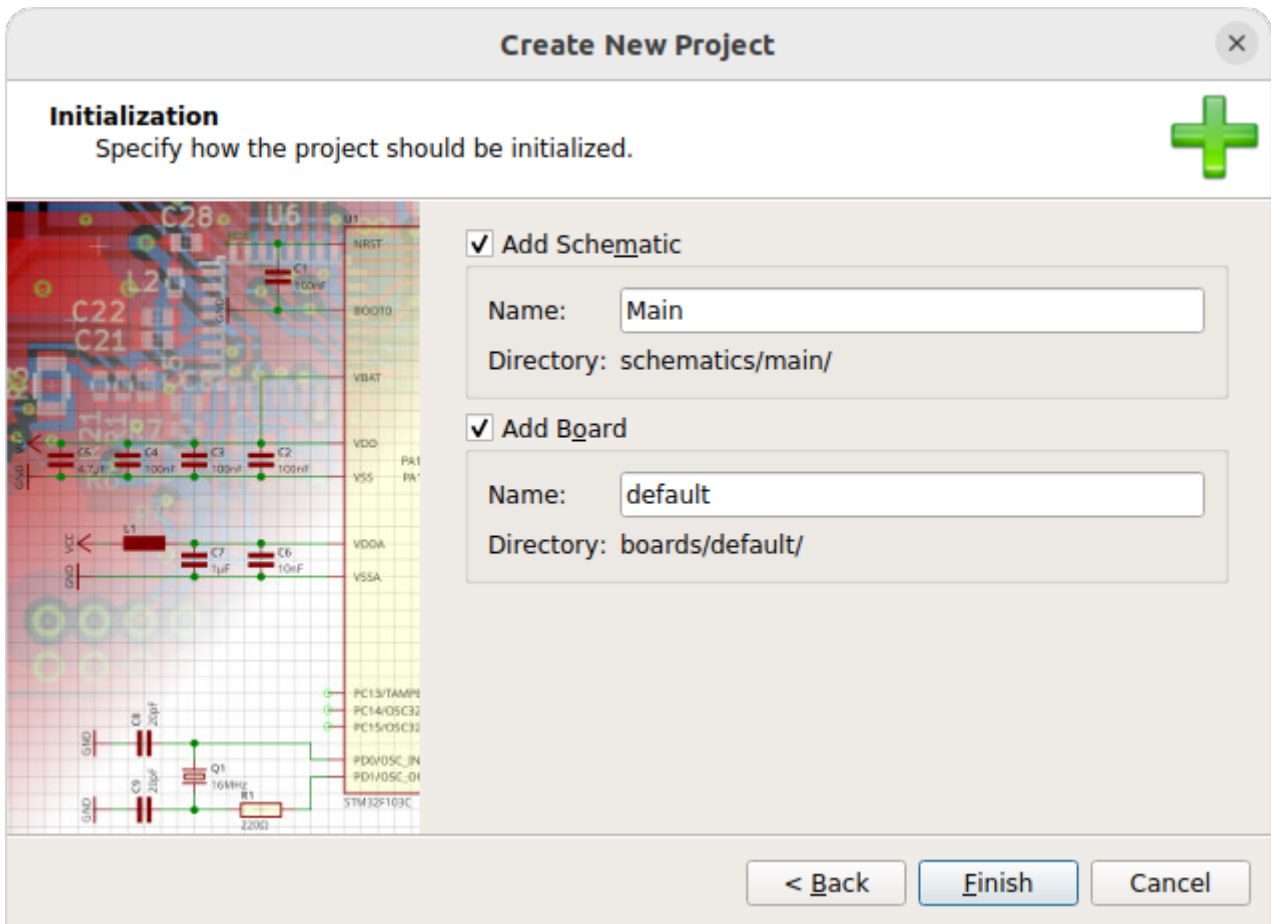
After clicking on **[Finish]**, the schematic- and board editors show up:
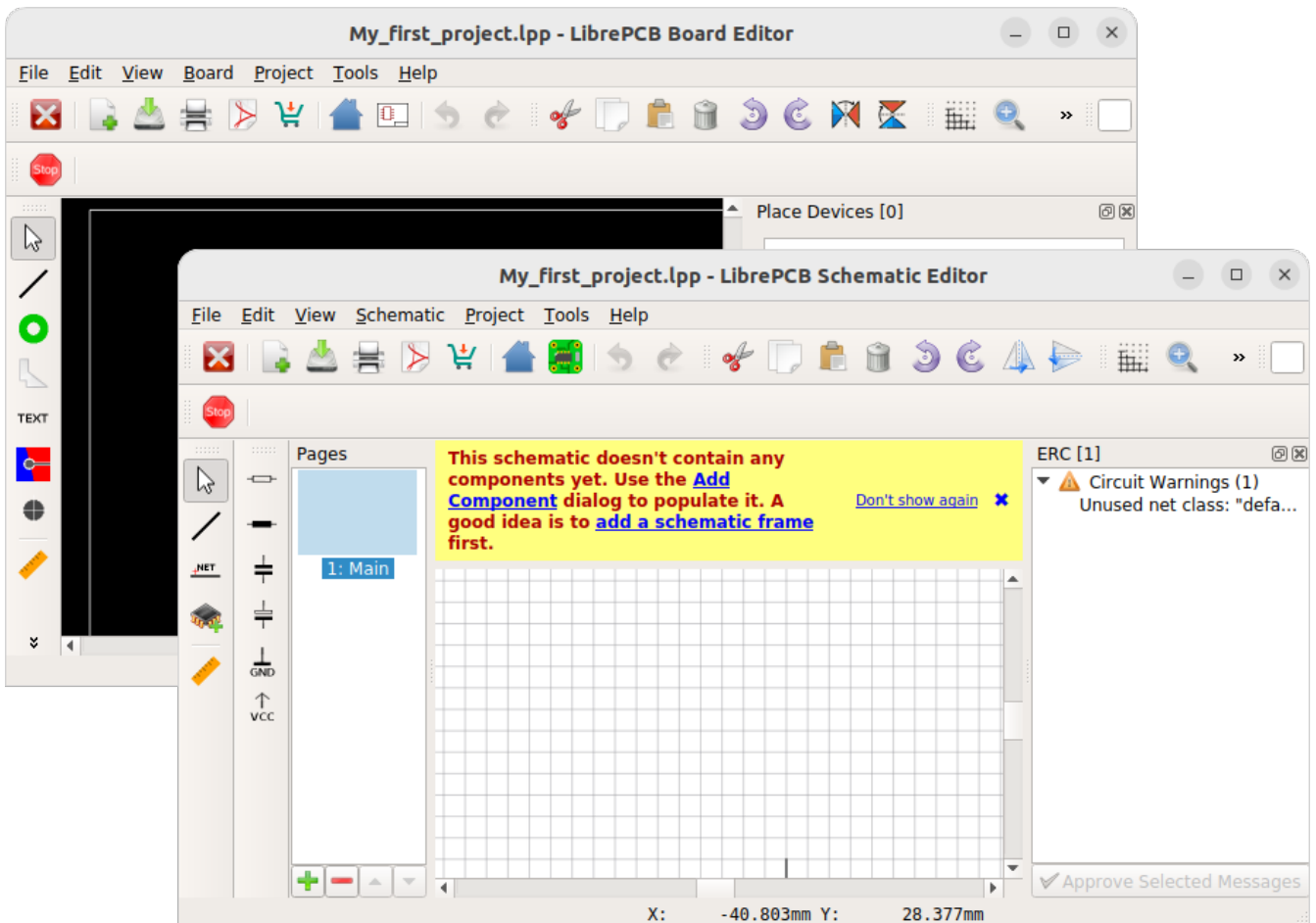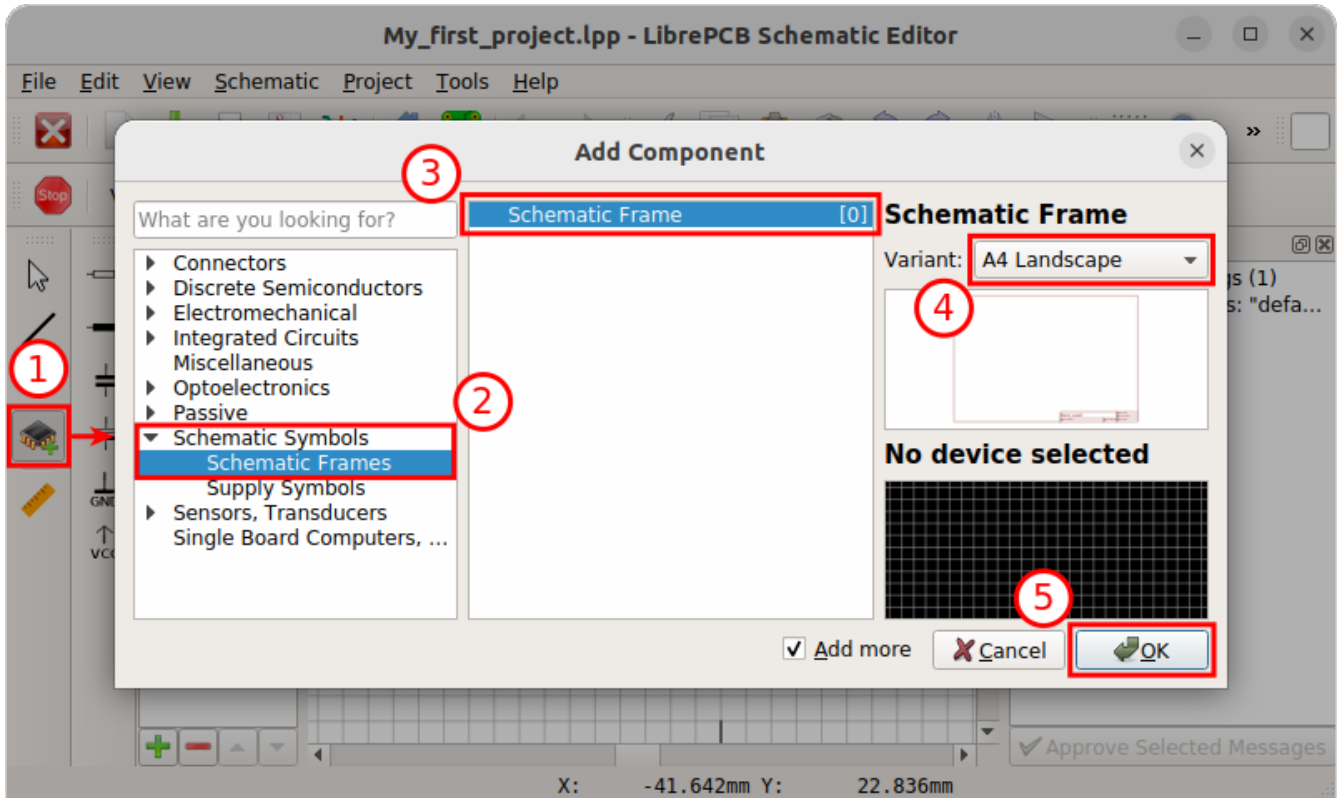
Create Schematics

Before starting with the board layout, a schematic will be needed. So let's see how to draw a schematic.

Add Frame

First, you may want to add a frame to the schematic. Click on **[Add Component]** in the toolbar and select a schematic frame:


After clicking on **[OK]**, the selected component is attached to the cursor. Click on the origin of your schematics to place the frame at coordinate (0, 0).

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.

Add Components/Devices

Now add all the resistors, capacitors, ICs etc. the same way to your schematic. However, for real parts (in contrast to the schematic frame) the dialog lets you select a concrete device. Here an explanation about the displayed information:
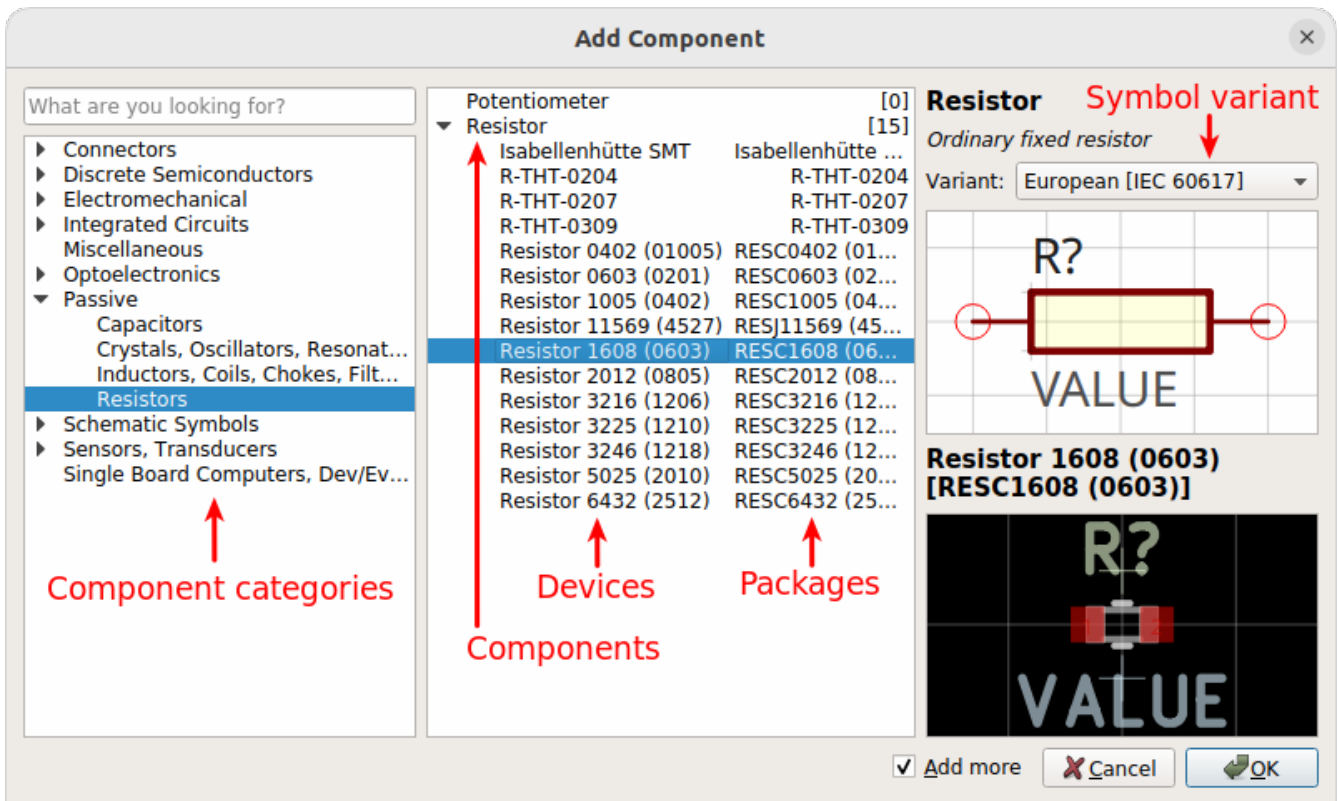
You can choose between adding a **component** or a **device**:

- **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**: Represents a real, orderable part (with MPN[1]). Basically it's the combination of a component and a package with a particular pinout.

To add a part to a board, you need to choose a **device**. 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.

> 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 and devices 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. Just click on a pin to start a new wire:
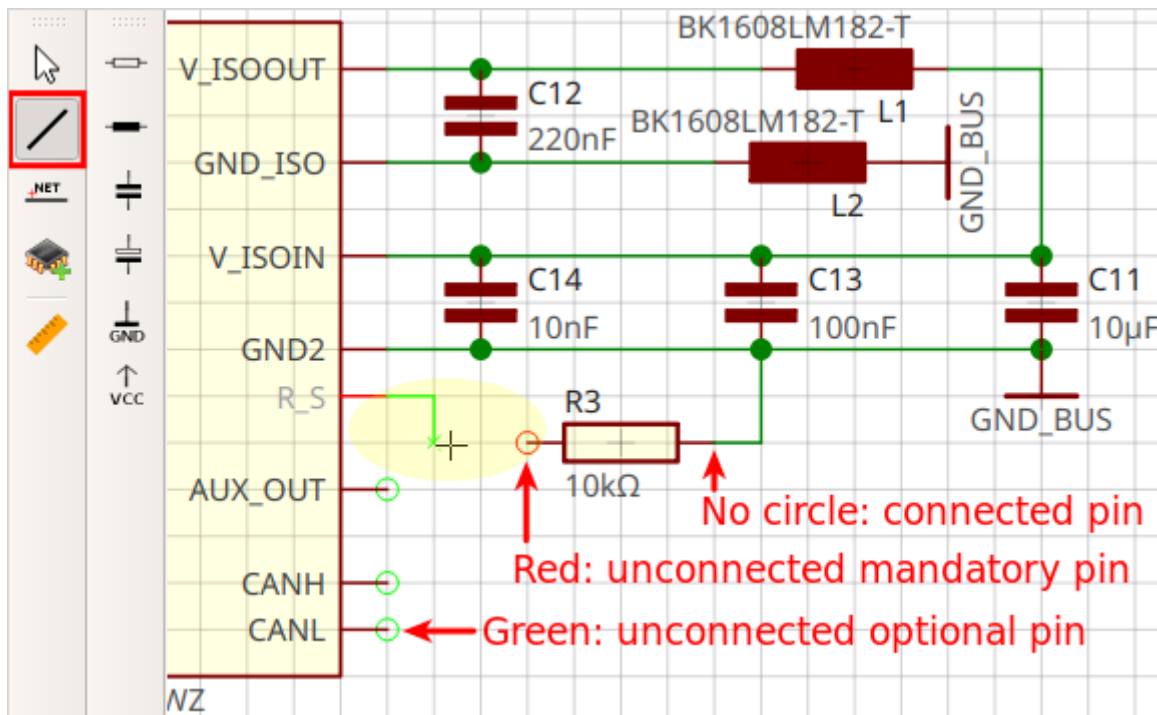
Pay attention to the circles around the pins. If a wire appears to be starting at a pin, but the circle is visible, it is **not** connected.

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[2] 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.
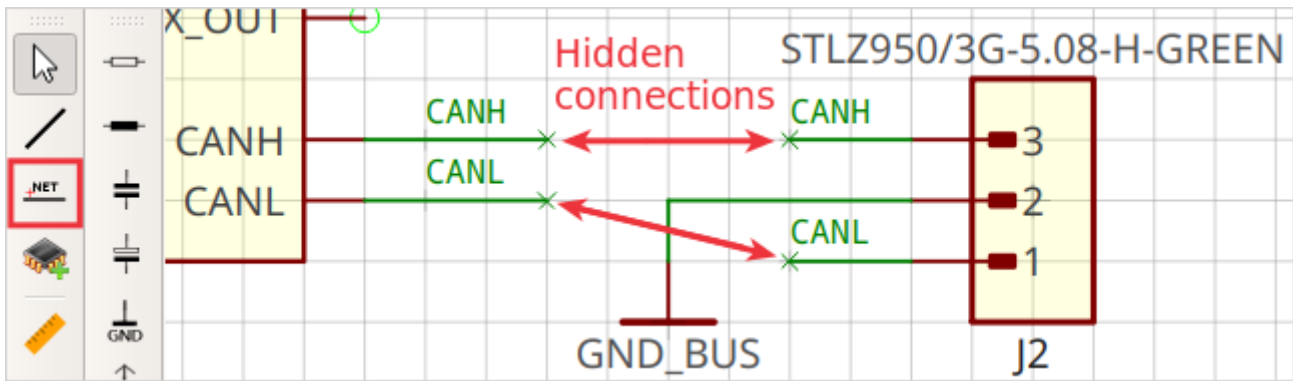
Add Net Labels

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 **[Add Net Label]** tool.

2. click on the wire where to attach the label.

3. Click to specify the label position.

While placing labels, press R to rotate.

> **i** 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 **Schematic › 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 dock with **View › Go to Dock › Electrical Rule Check (ERC)**:
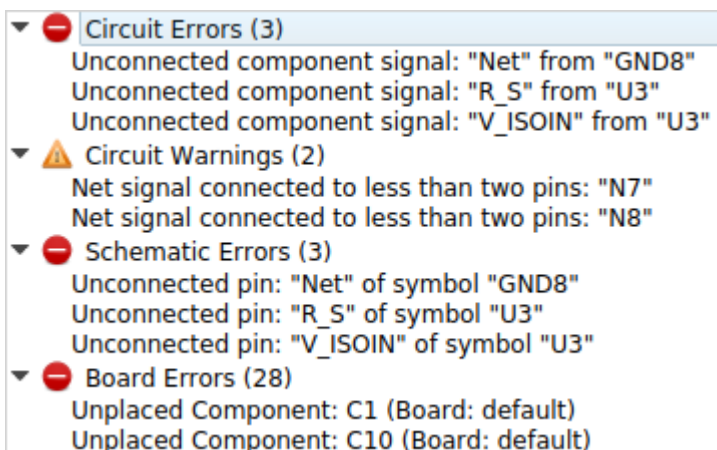
Note that the *Board Errors* can be ignored for now since we didn't create the board yet. Just fix the *Schematic Errors*, then *Schematic Warnings*. Usually the *Circuit Errors* and *Circuit Warnings* will then disappear automatically.

Create Board

Once the schematic is (more or less) complete, you can start designing the PCB in the board editor. If the board editor window is not opened already, click on the **[Board Editor]** toolbutton to open it.

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 with the **[Grid Properties]** toolbutton (or with F4):
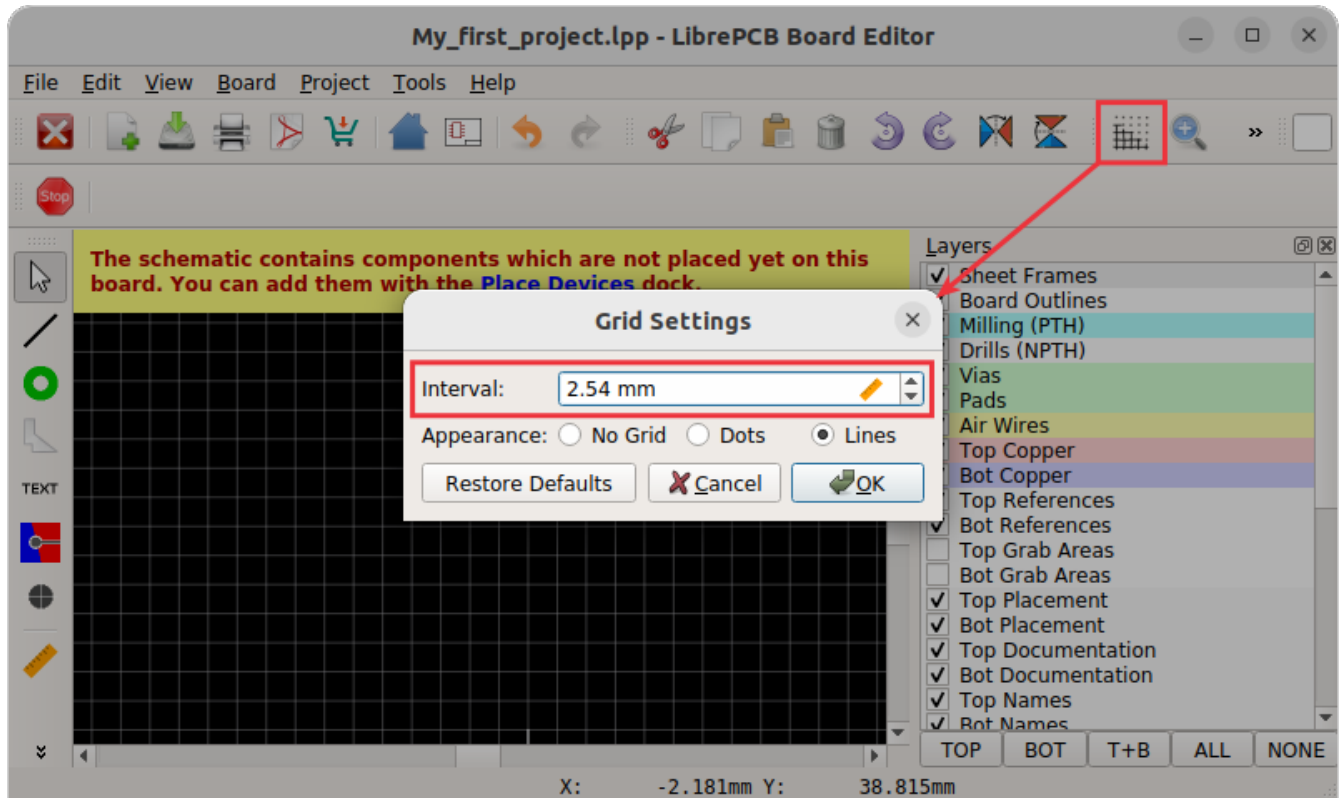
Draw Outlines

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

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

> **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* layer shall represent the actual board outlines (i.e. the edges), **NOT** the paths for the milling cutter! The PCB manufacturer will automatically offset the outline polygons to calculate the actual paths for the cutter.
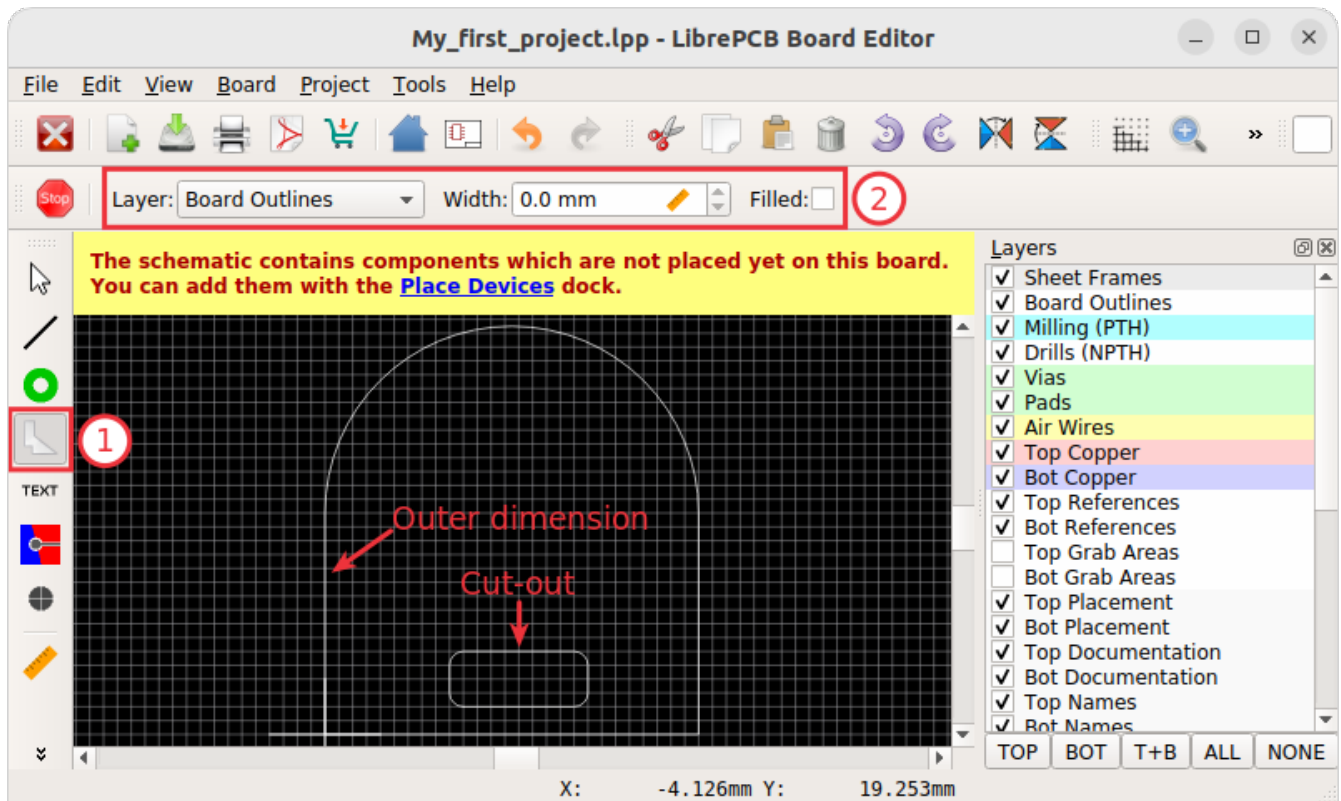
Keep in mind that inner edges can only be produced with a specific minimum radius (corresponding to the milling cutter diameter of the PCB manufacturer). Although PCB manufacturers may produce your PCB anyway even if it contains inner edges with no or too small radius, it's highly recommended to draw all inner edges with a proper radius. Often a radius of 1.2mm or more works fine, while a smaller radius might lead to additional cost.

To draw polygons with arcs, open **[Properties]** from the polygon's context menu (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 to define the outer board outline.
- Cut-out polygons (if there are any) are located fully inside the outer board outline.
- There are no tangent or intersecting *Board Outlines* polygons.
- Polygon layer is *Board Outlines* and line width is 0.0mm (the latter is optional, but recommended).
- Polygons are closed (start and end coordinates are exactly identical) and consisting of a single polygon object (**NOT** multiple joined lines!).
- There are no other objects on the *Board Outlines* layer.

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* dock (**View › Go to Dock › Place Devices**).

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 **[Add]** 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.
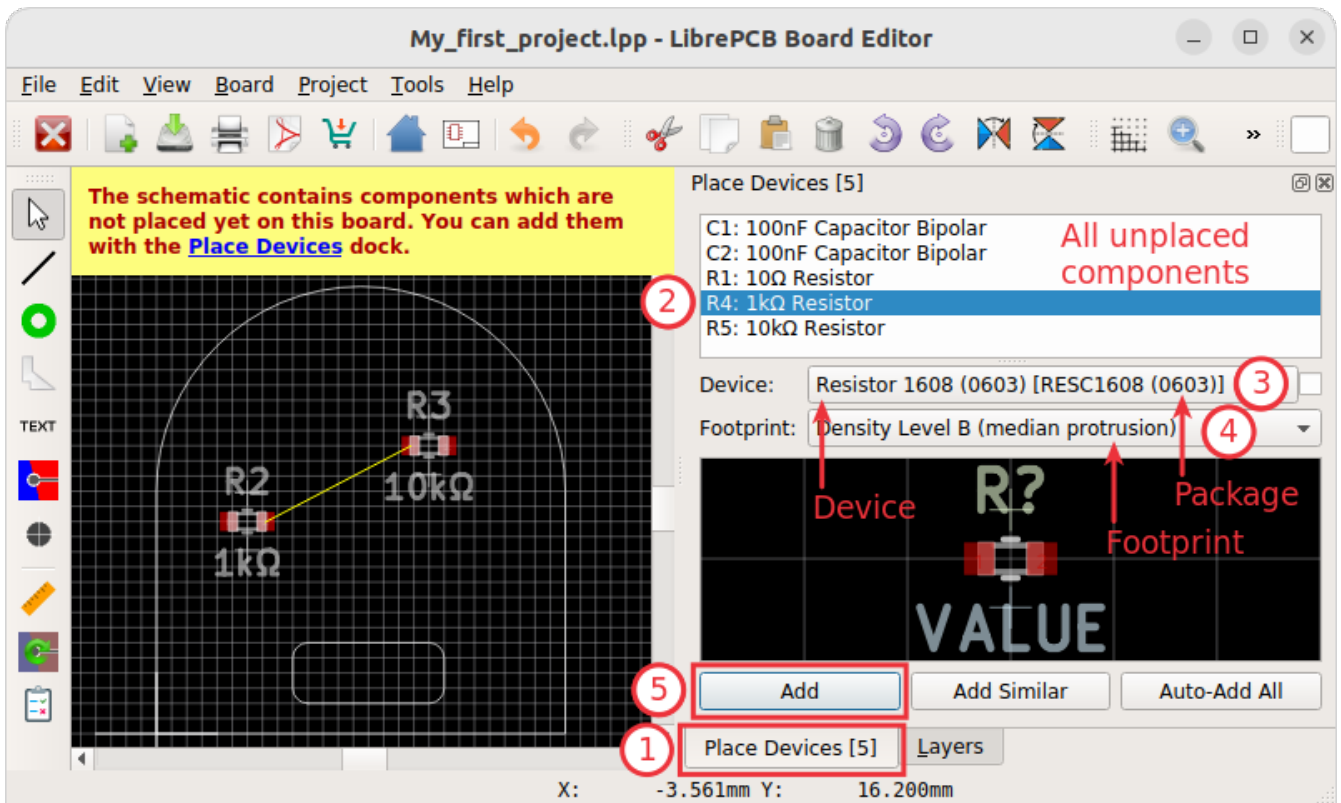
Repeat these steps until there are no more unplaced components.

> 💡 If you want to use the same device and footprint for all instances of a particular component, use the **[Add 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):
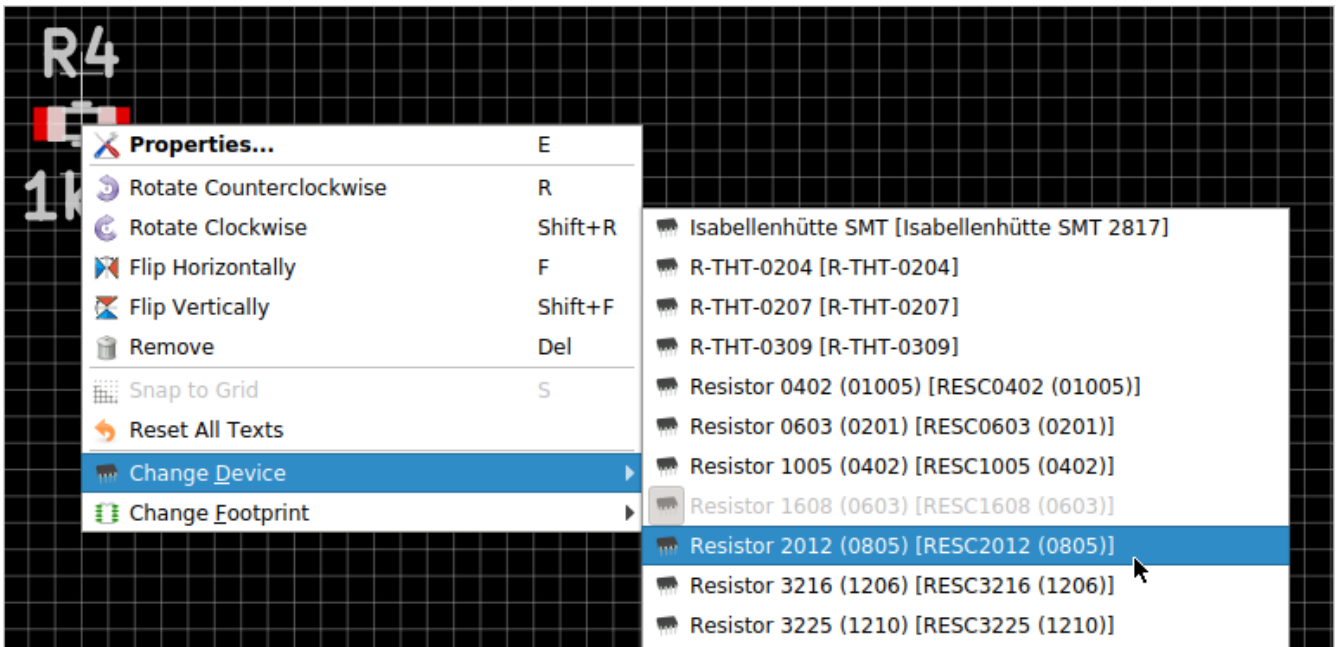
Exactly the same way you can switch to a different footprint, just use the **[Change Footprint]** context menu item instead.

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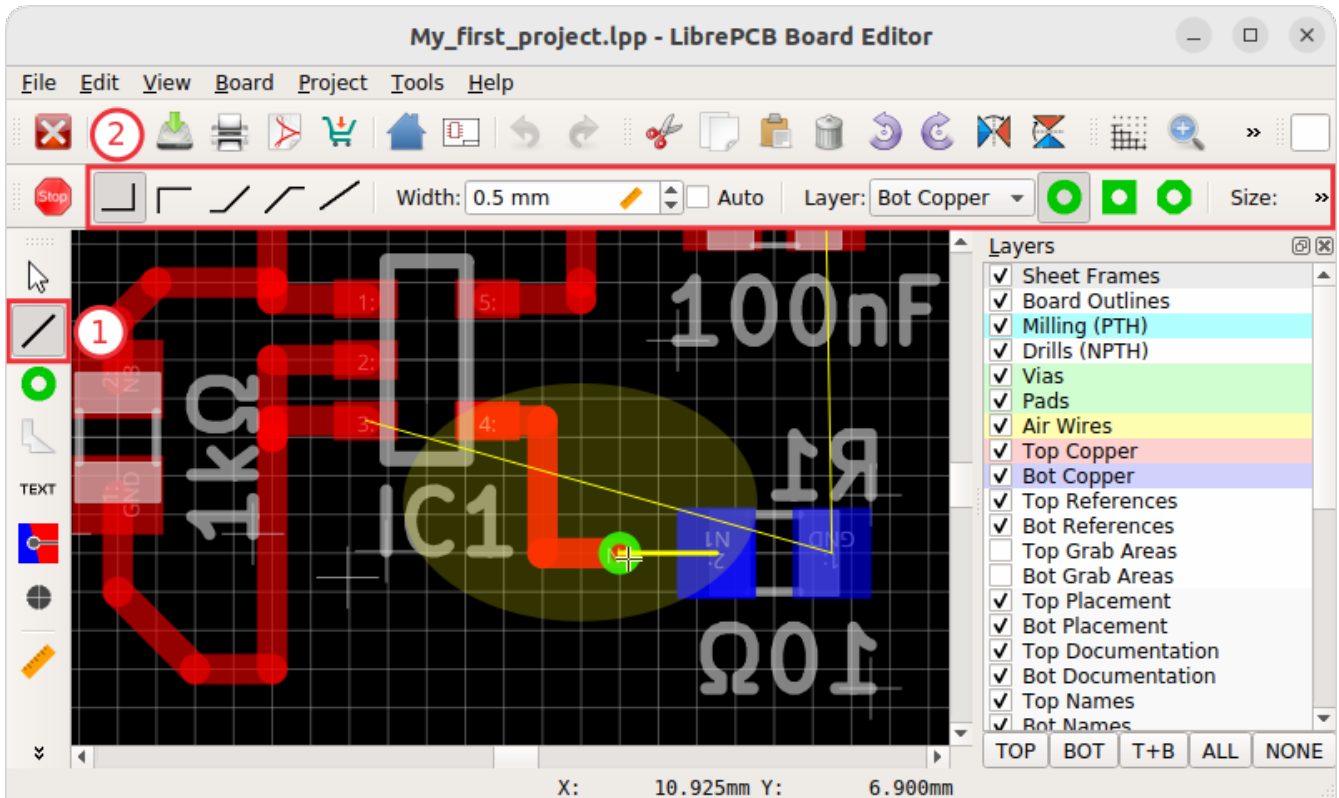
> ℹ️ The cursor automatically snaps on objects of the same net. If this is not desired, hold `Shift` while drawing.

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 electrical net and copper layer 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.

4. Once the planes are added, you need to trigger a rebuild manually with the **[Rebuild All Planes]** toolbutton (or with `Ctrl` + `Shift` + `R`).
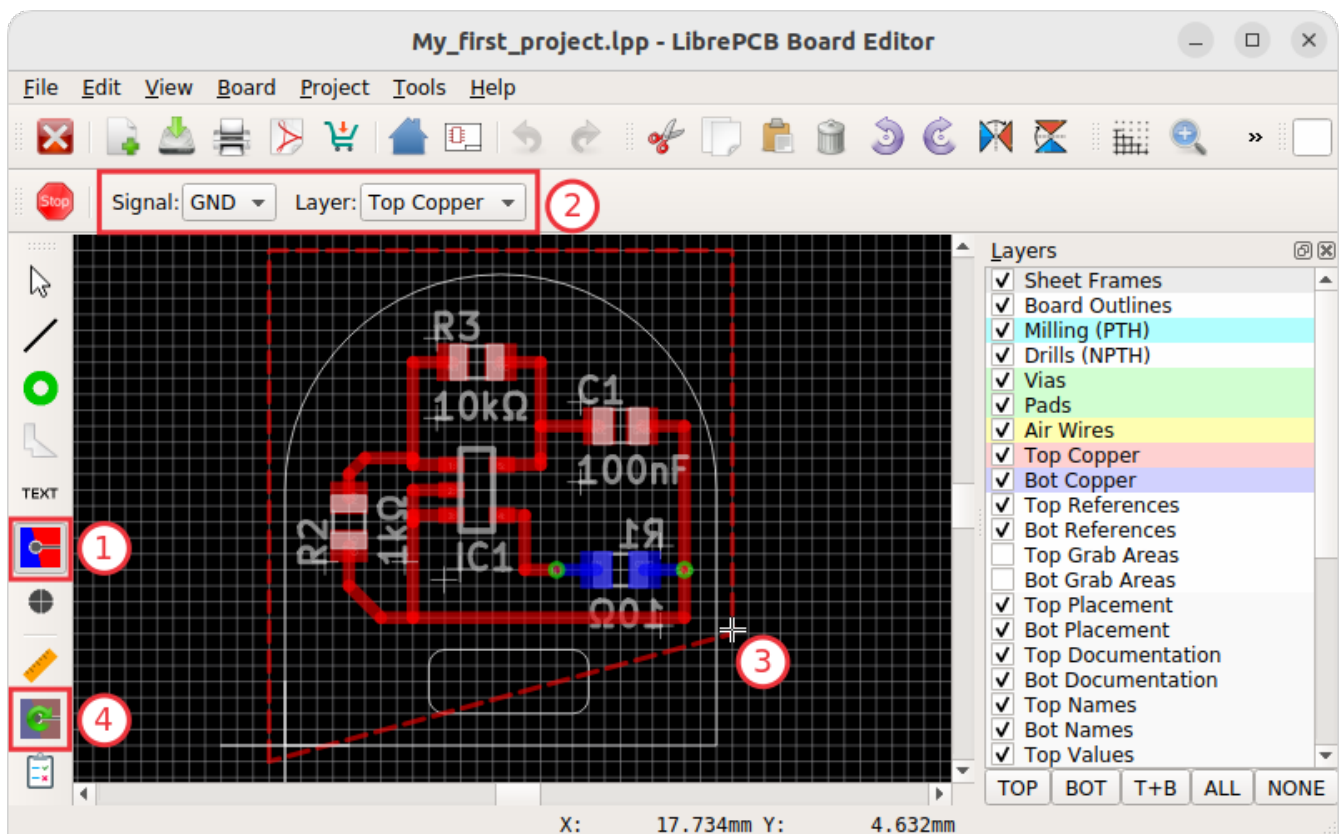
After rebuilding the planes, they appear with a filled area. As you can see, the area is automatically clipped to the board outline:
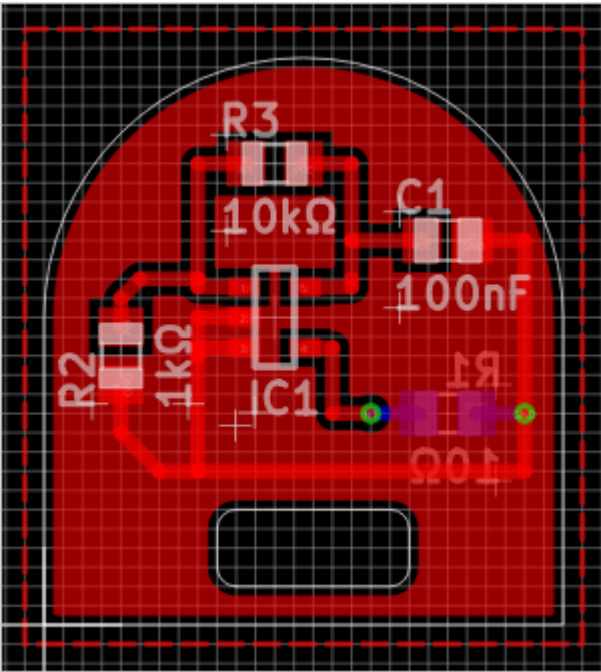
Planes are not rebuilt automatically. Whenever you make a change which affects the planes (e.g. moving devices, drawing traces, …) the plane areas might be outdated and you need to trigger a rebuild manually.

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 Orphans* 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 Orphans* 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.

But first you should check or adjust the design rules which are used to calculate via/pad restrings and cream/stop mask clearances. For that, open **Board › Design Rules** or press F7 :


>
>
> Actually it's better to set the design rules *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 if you're unsure about these values, just keep the defaults.

Once the design rules are set, open **Board › Design Rule Check** or press F8 to configure the checks and to run the DRC:
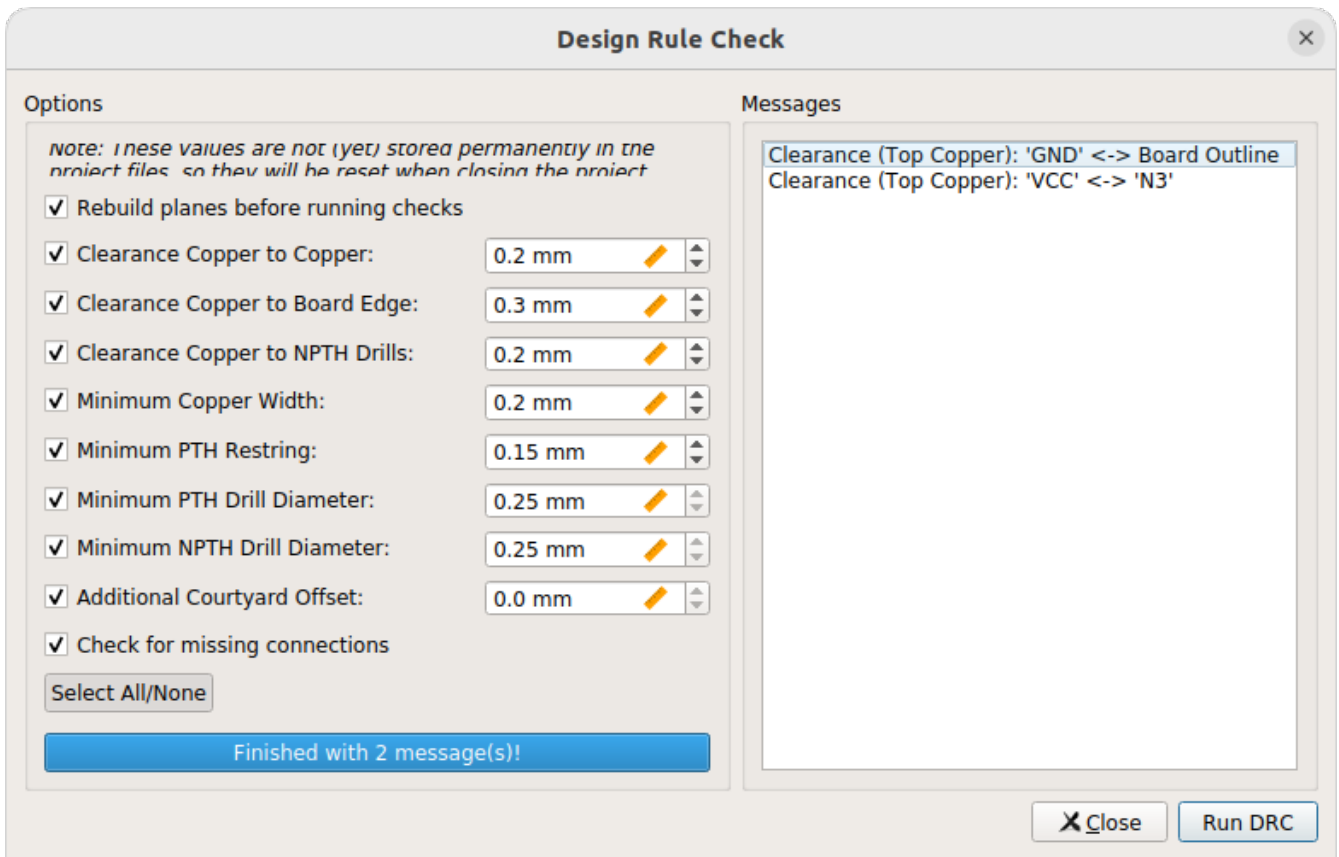
> ⚠️ These settings are not yet stored persistently in your project, therefore you have to set them each time again. This will be improved in the next major release, LibrePCB 0.2.x.

If there are any DRC messages, the DRC dock will pop up automatically after closing the *Design Rule Check* dialog. Then just click on a message to highlight the issue in the board editor. There are also buttons to quickly run the DRC again or open the settings dialog:
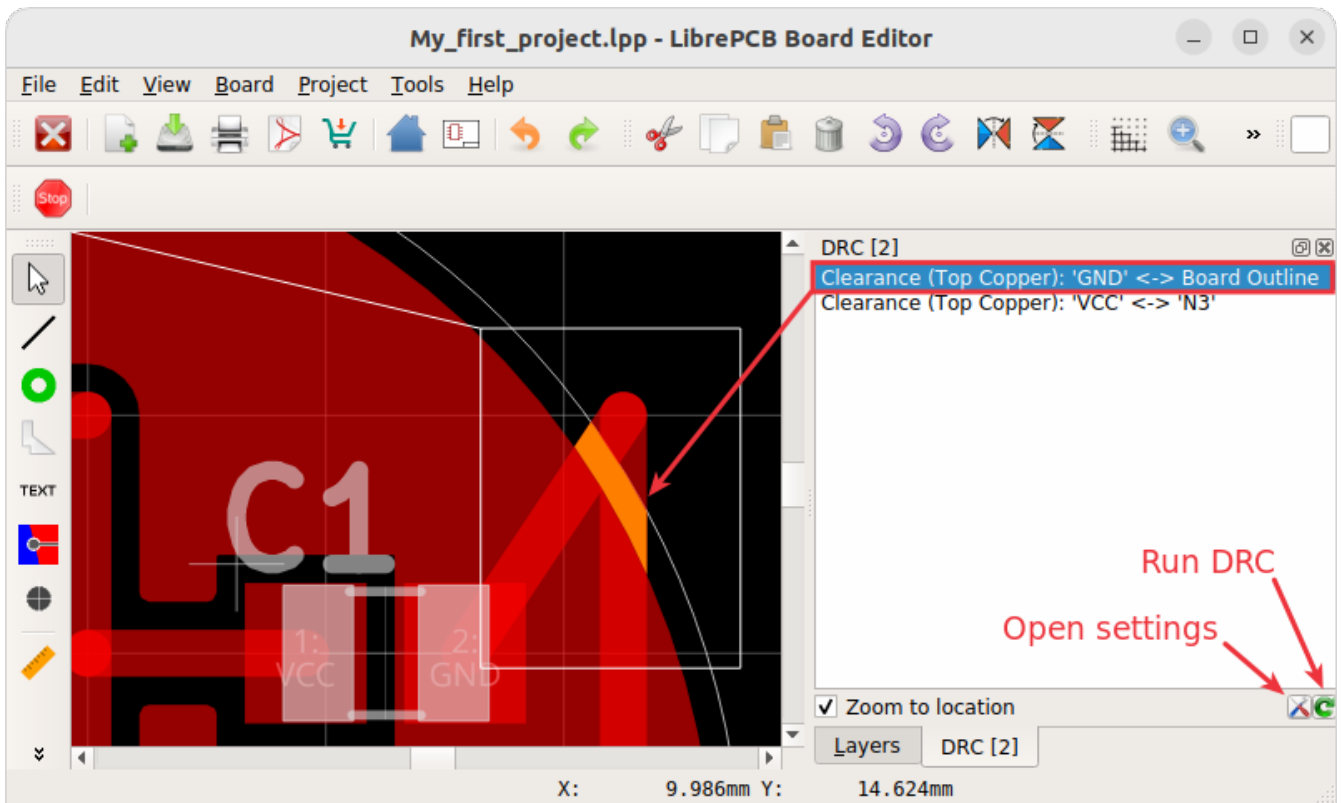
> 💡 Now it would be a good time to also check for ERC issues again to be sure all electrical issues are fixed as well.

Once you fixed all issues, 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, click the **[Order PCB]** toolbutton in either the schematic- or board editor:
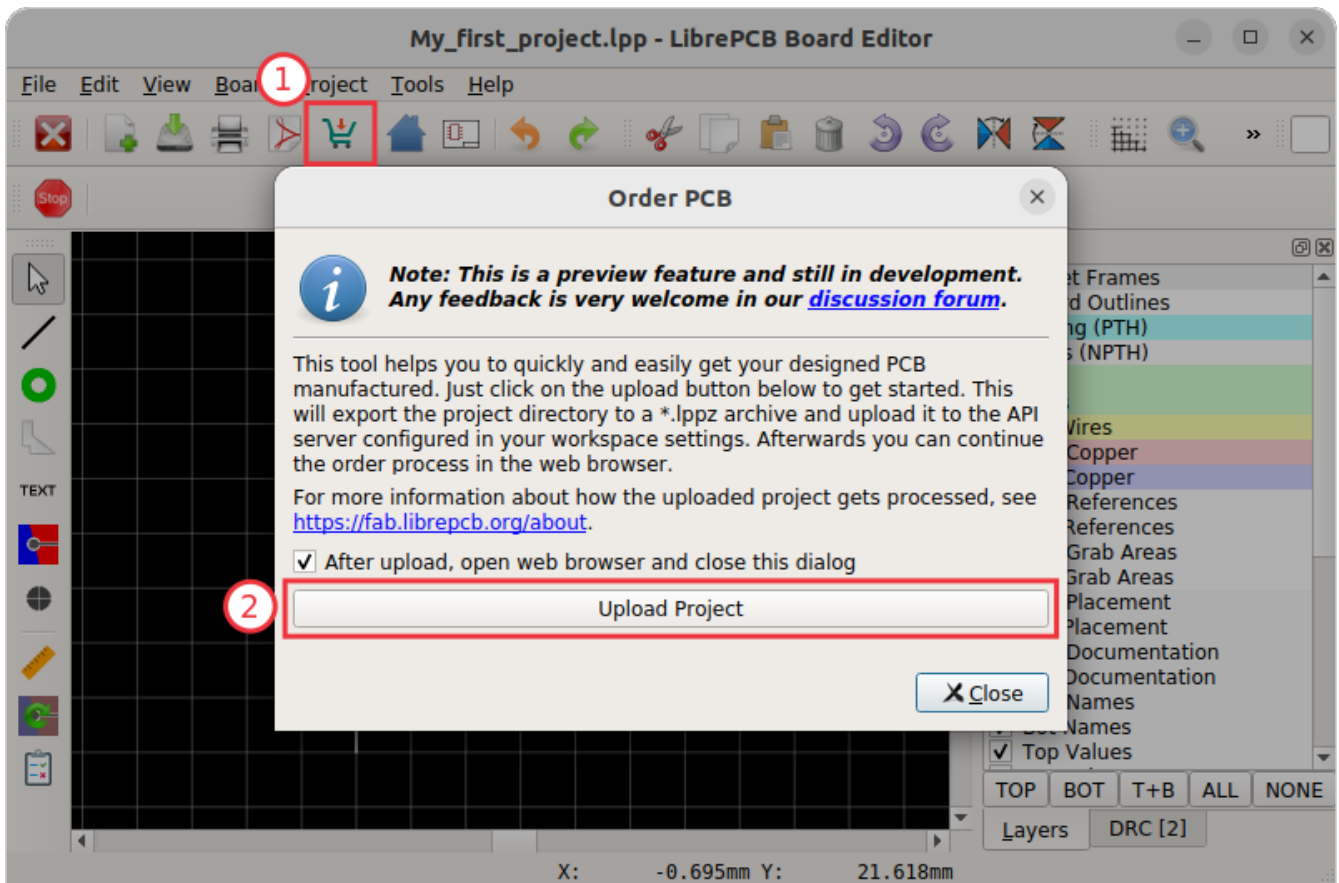
With **[Upload Project]**, the project is uploaded to our order service fab.librepcb.org. Then your web browser should open a website where you can review and continue the order.

> 💡 Alternatively you could also export your LibrePCB project as a `*.lppz` archive (**File › Export › 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.

Gerber/Excellon

Open **File › Production Data › Generate Fabrication Data** or press `F10`. Then choose the settings of the Gerber/Excellon export. Generally you should determine what format your PCB manufacturer accepts. Many manufacturers accept Protel-style settings, so if you're unsure, click on **[Protel naming]**:
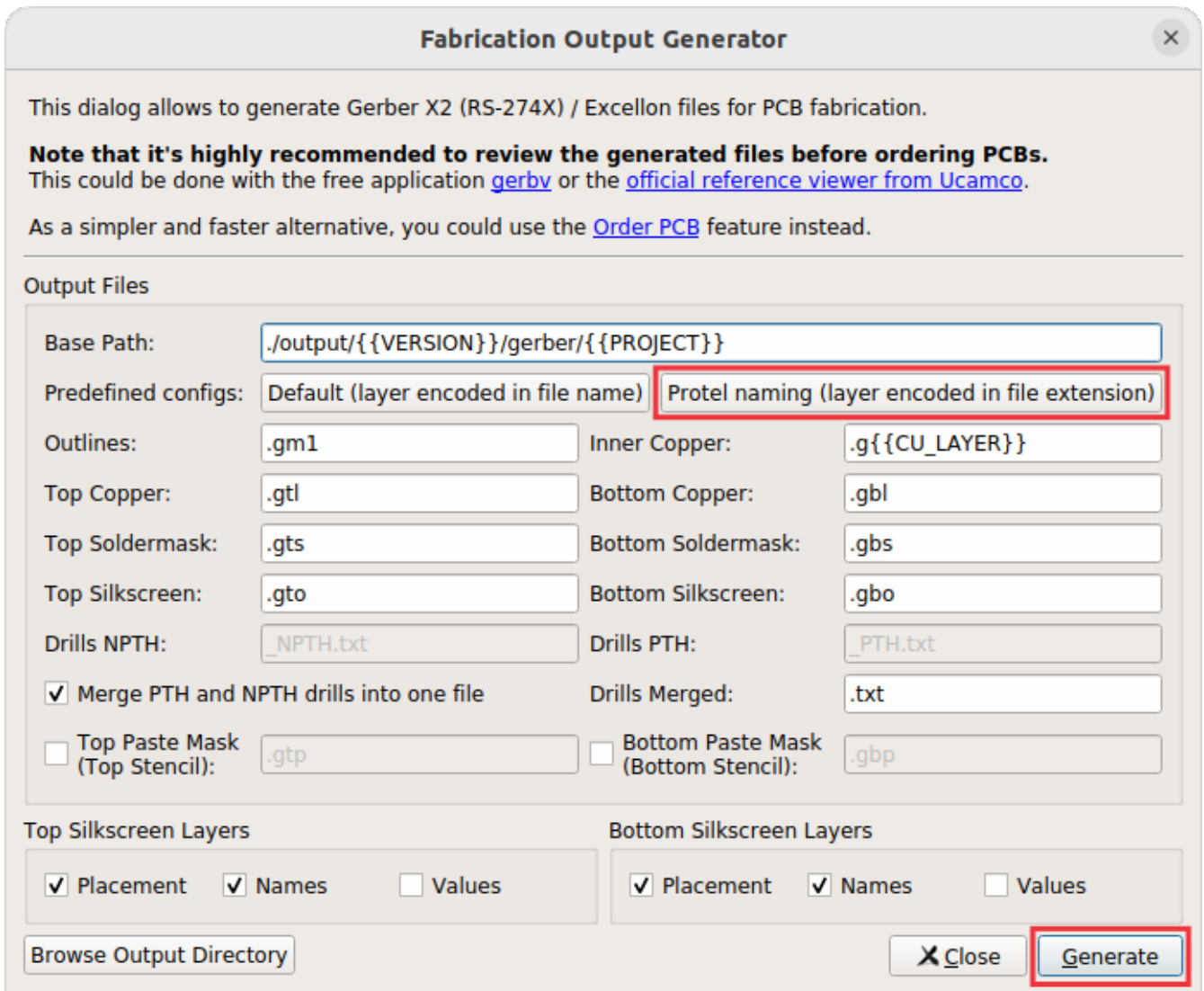
Press **[Generate]** and LibrePCB writes the files into the selected directory within the project. These files can then be sent to your PCB manufacturer for production.

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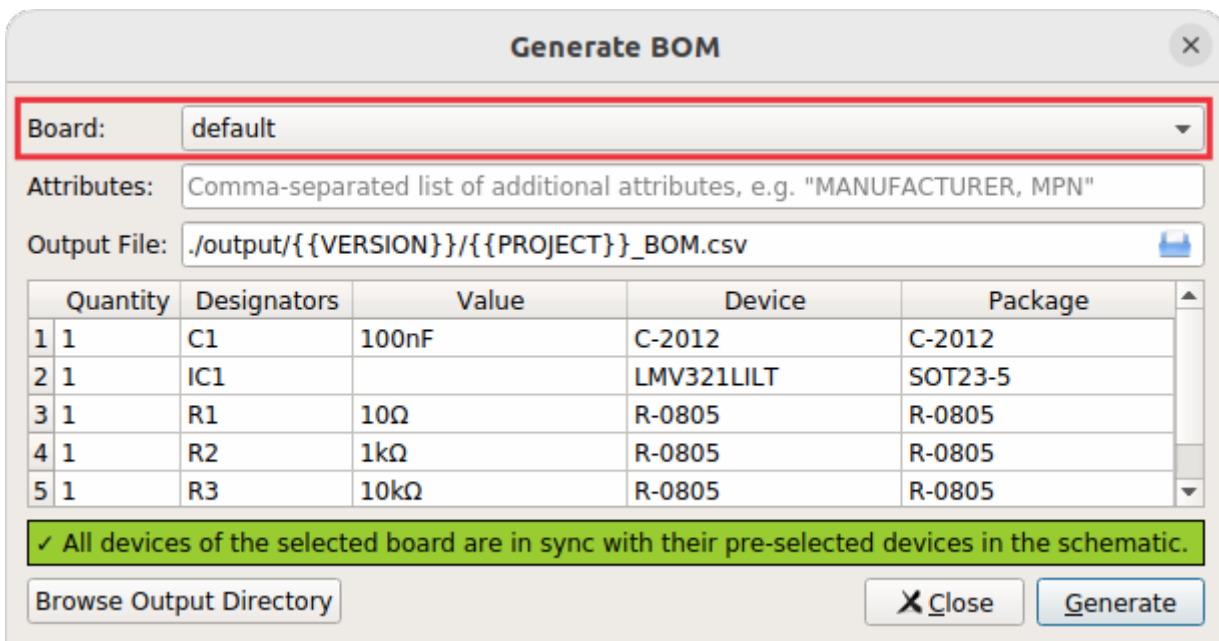
Pick&Place Data

If you also need pick&place files for automated assembly, open **File › Production Data › Generate Pick&Place Files** or press `F11`:
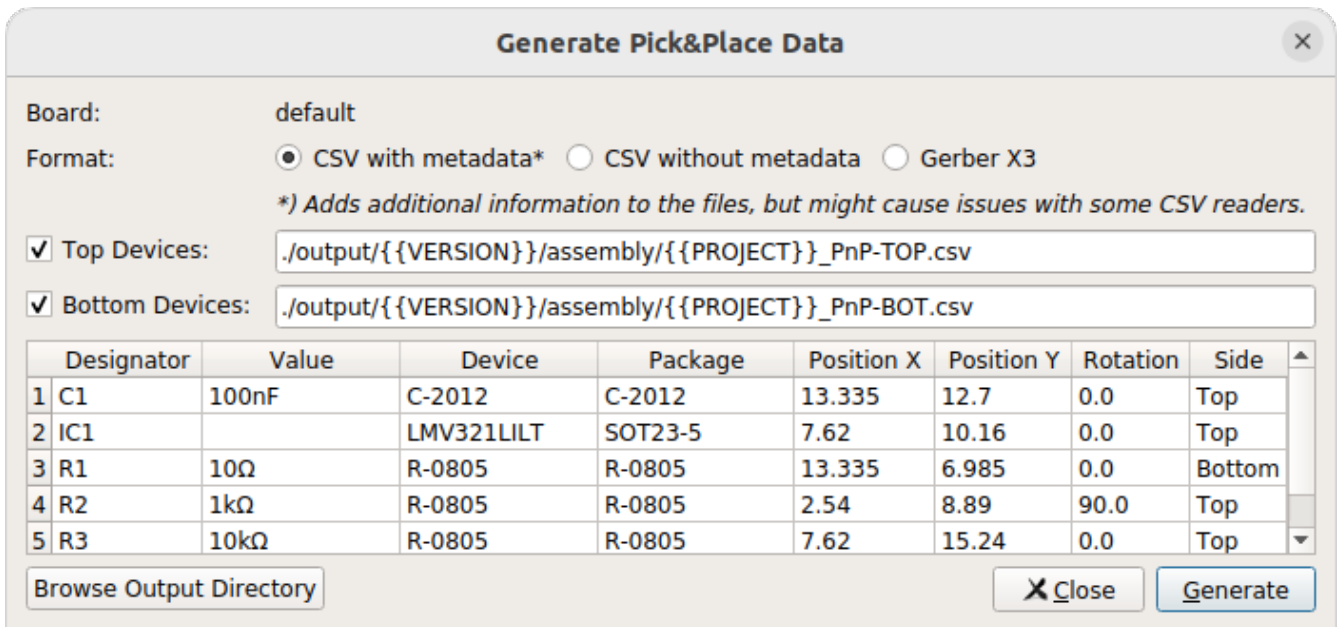
Export BOM

To get a list of all parts to purchase, export the bill of materials (BOM) from either the schematic editor or board editor with **File › Production Data › Generate Bill Of Materials** or with `F9`:

> ⛔ The BOM depends on the board since the concrete devices are specified in the board, not in schematics. Therefore, make sure the desired board is selected in the dropdown. However, if you start the BOM export in the board editor (instead of the schematic editor), the currently opened board will be pre-selected automatically.

> ℹ️ LibrePCB does not yet provide a built-in mechanism to specify manufacturer part numbers (MPN). This will be added in a future release. However, you can already add arbitrary attributes to each component which can then be included in the BOM. Just open **[Properties]** from the context menu of components (right-click)

and add a string attribute named `MPN`.

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 in the library manager:
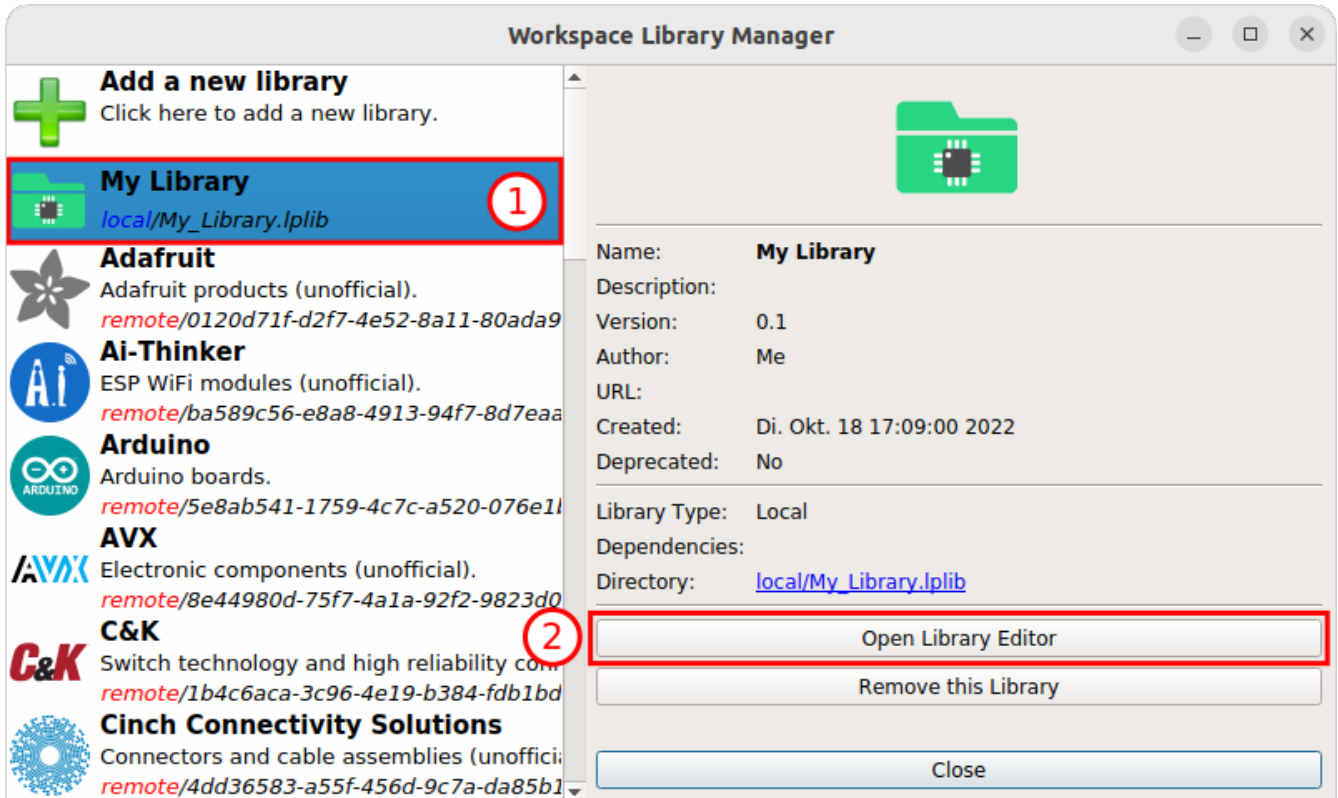
The **[New Library Element]** toolbutton (or `Ctrl` + `N`) in the library editor is the entry point for every new library element. There you can choose what kind of library element you want to create:
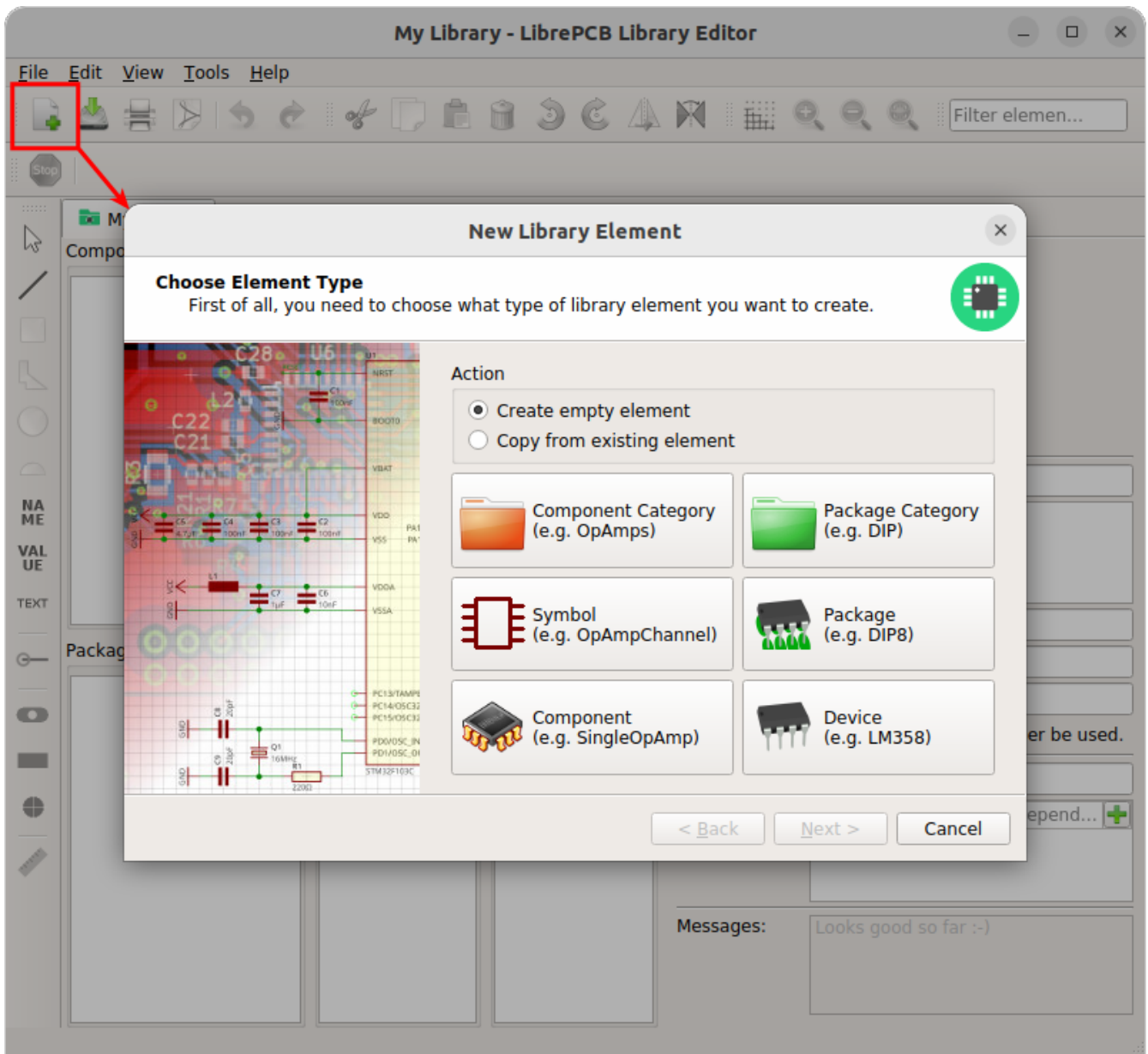
Concept Overview

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). Open **New Library Element › Component Category**, choose a suitable (generic!) name and select a parent category. **You may first need to create the required parent categories.**

> ℹ️ 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 **[Finish]**, 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.

▼ *Component categories available in the LibrePCB Base library*

Symbol

Now we need to create a symbol for the OpAmp. Open **New Library Element › Symbol**, choose a name and the component category we just created and click **[Finish]**:

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.
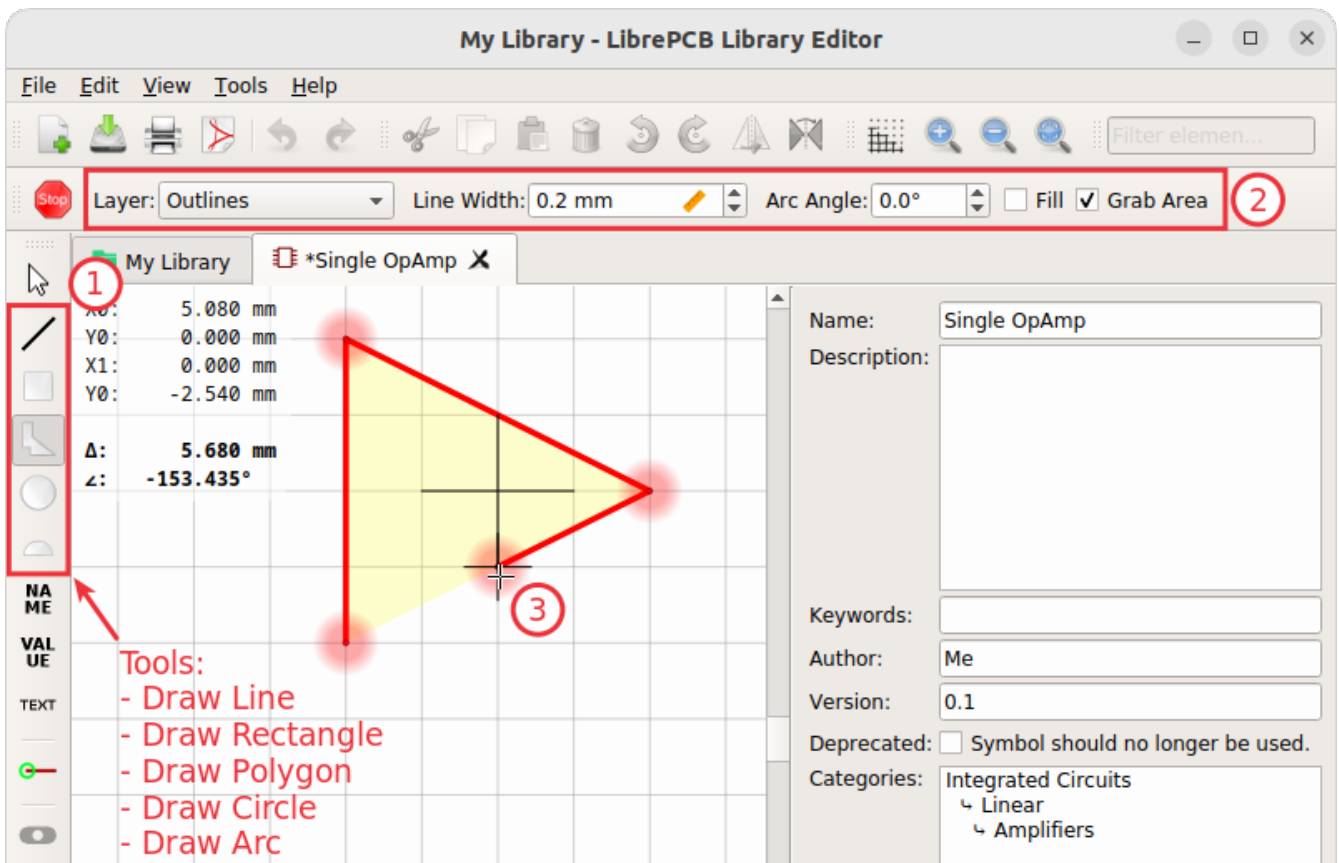
Add Texts

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.
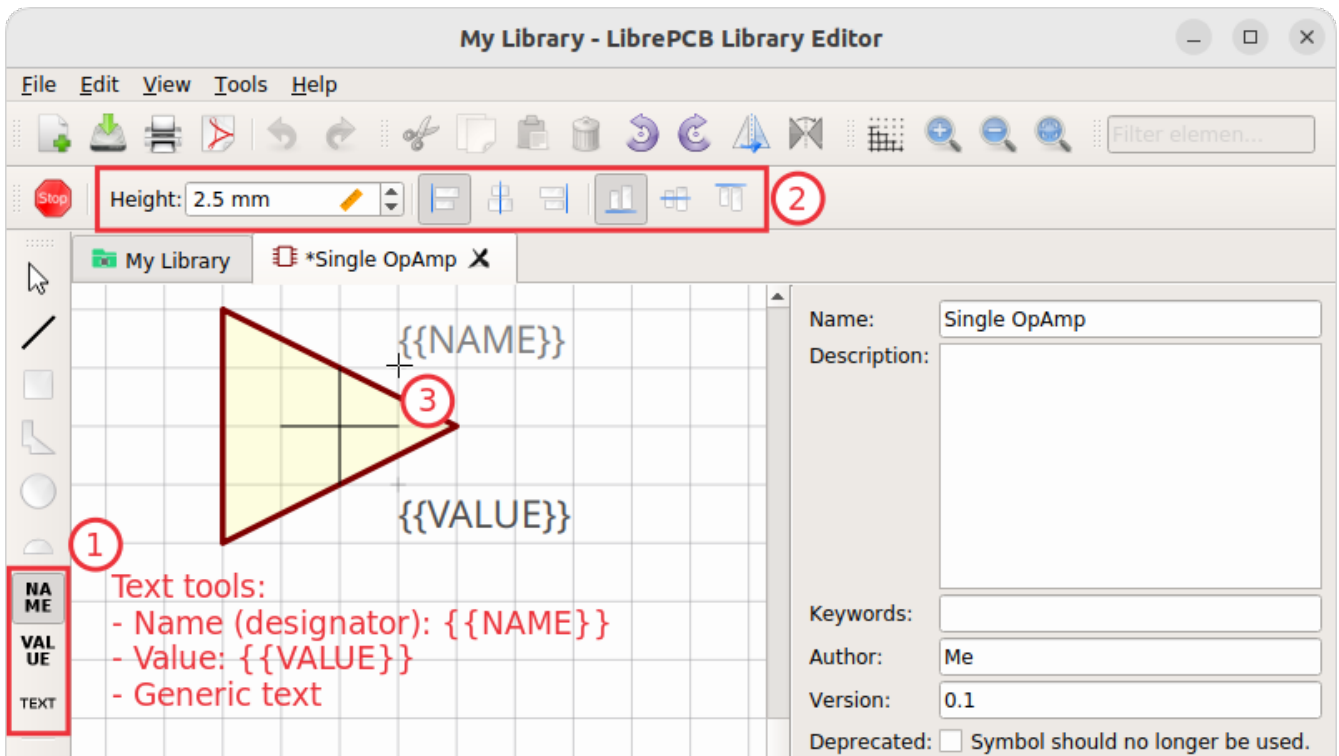
Add Pins

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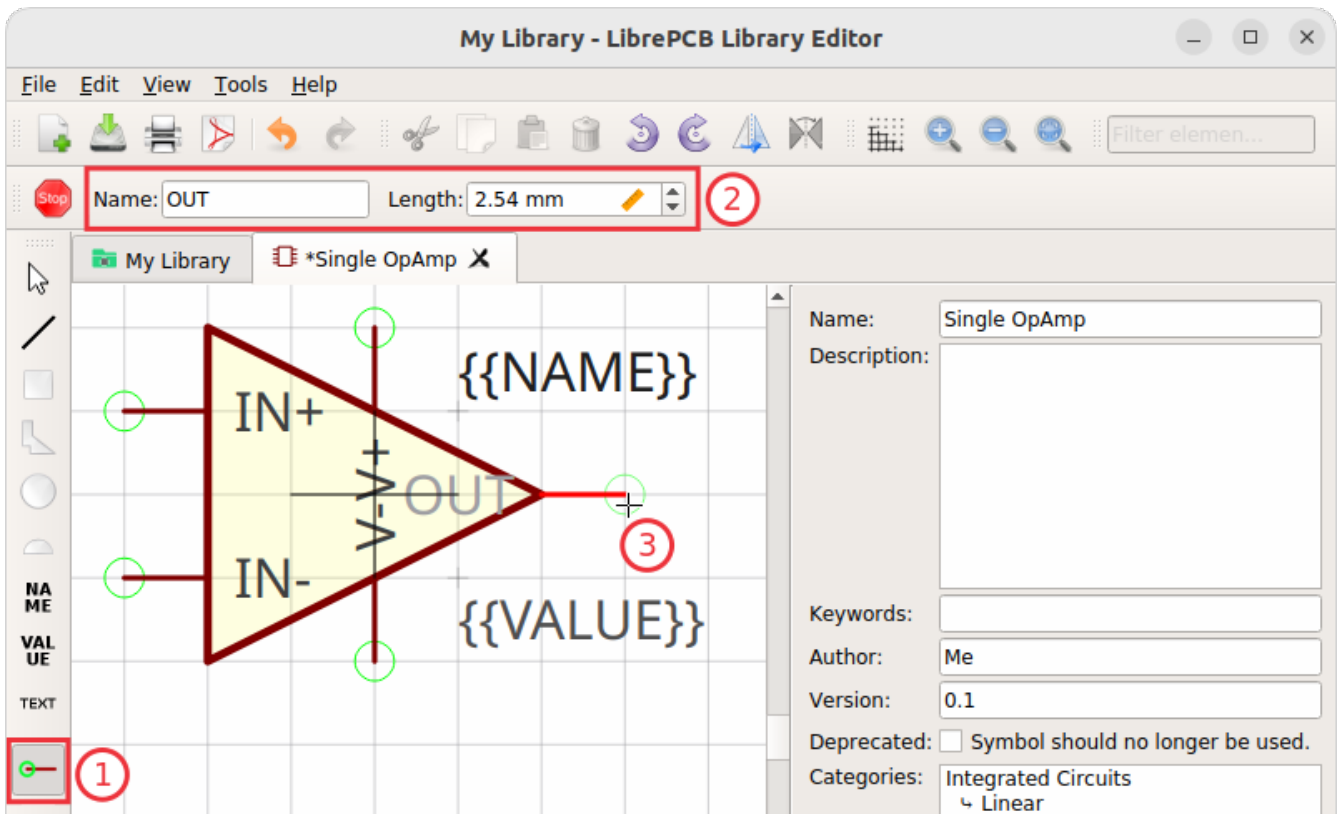
The overlapping pin texts look a bit ugly, but let's ignore that for the moment.

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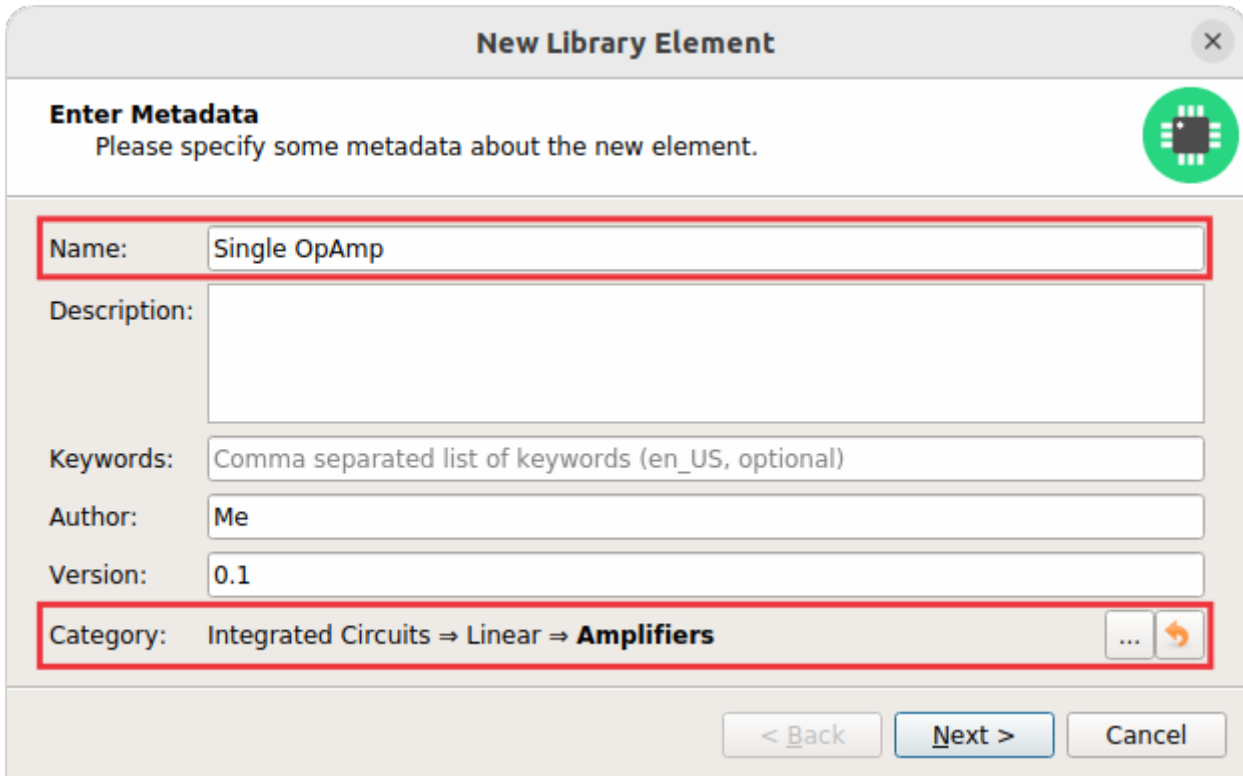
For details about how symbols should be designed, please take a look at our symbol conventions. The most important rules are:

- 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*.

Open **New Library Element › Component**, enter the name and assign the component category we created previously:

Set Properties

After clicking on **[Next]** you're asked to specify some properties of the component:

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 {{PARTNUMBER}}, {{DEVICE}} or {{CAPACITANCE}}. If you are unsure, just leave it empty, the component editor will help you to

assign a value later.

New Library Element

Component Properties
Set the component and the package of the new component.

Schematic-Only: ☐ Component cannot be used in devices and boards.

Prefix: IC

Default Value: {{PARTNUMBER or DEVICE}}

< Back   Next >   Cancel

Add Symbols (aka *Gates*)

Now you need to choose the symbols which represent the component in schematics (also called *gates*). Most components have only one symbol, but you can also add more than one, for example an OpAmp could have separate symbols for power and amplifier. In our case, select the *Single OpAmp* symbol we created previously:
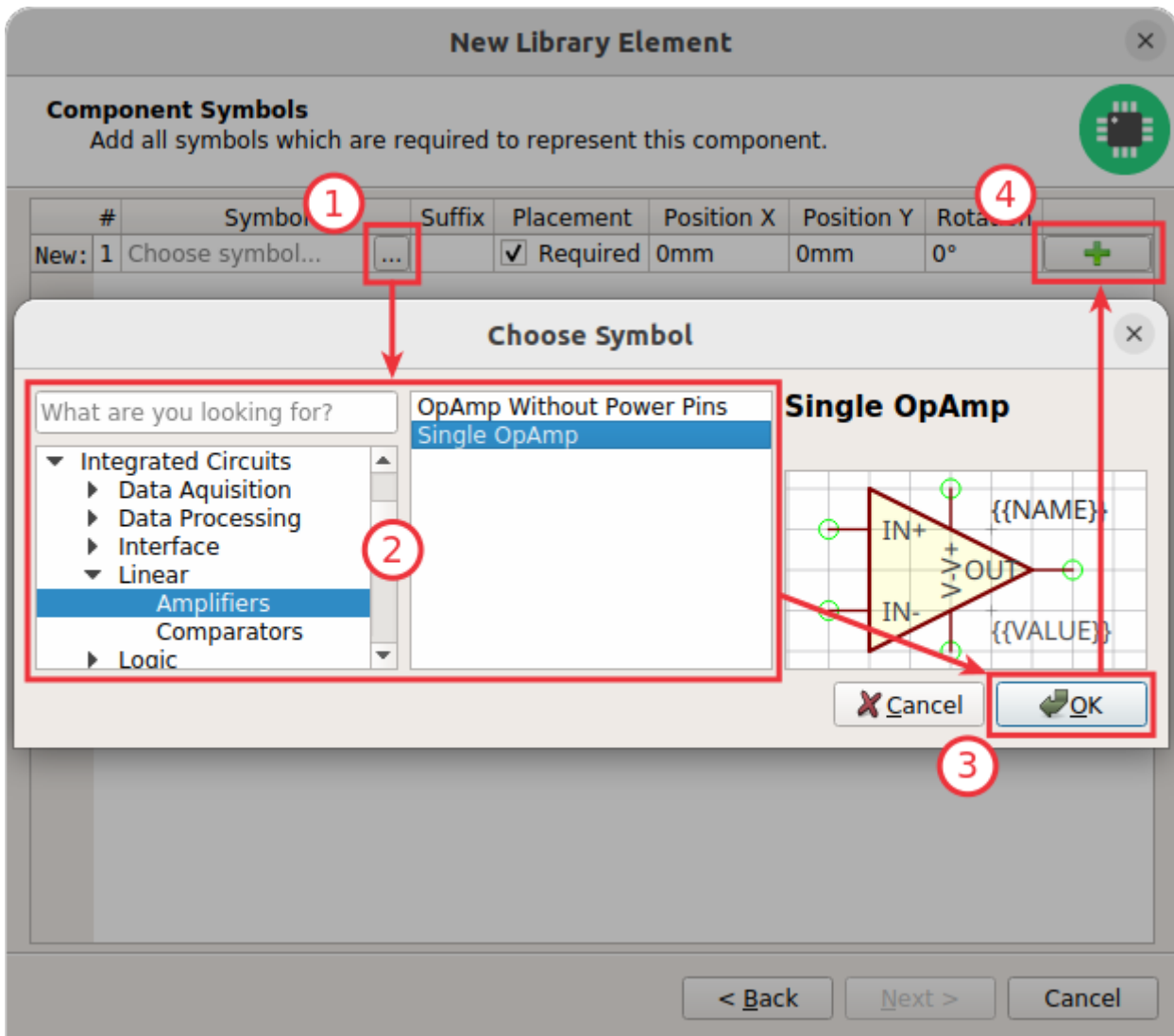
Don't forget to click on the **[+]** button after closing the symbol chooser dialog. Then click on **[Next]**.

Add 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 pins of the specified symbols, so often you don't have to do this by hand. But sometimes you still should adjust the names or properties of these 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:
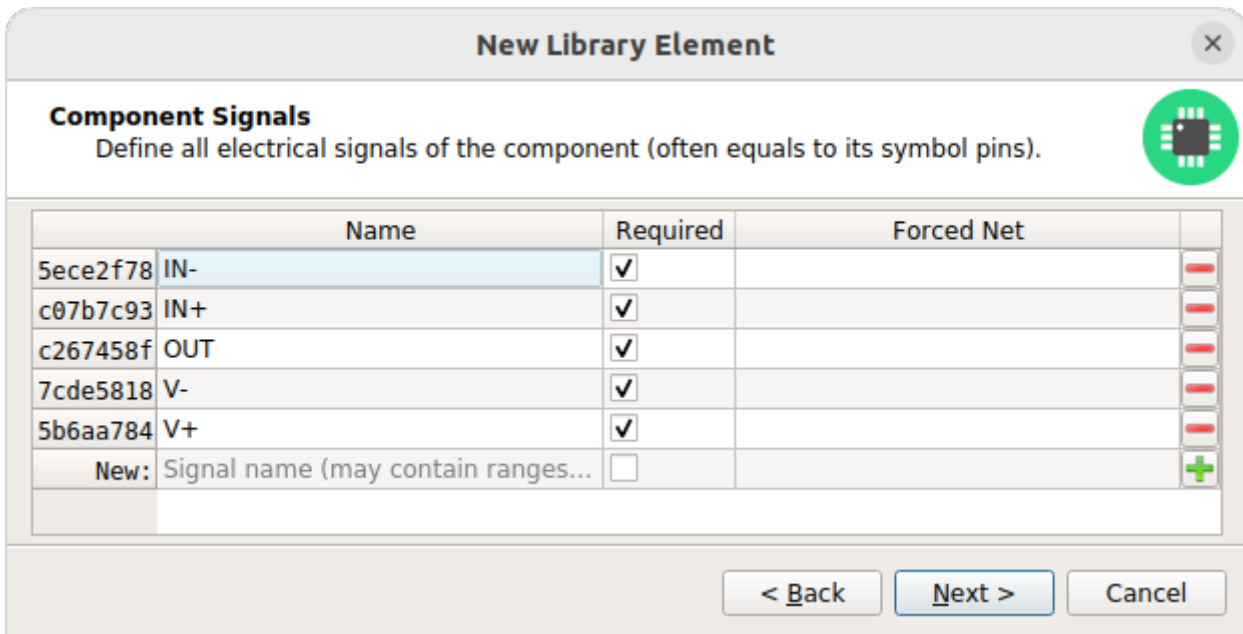
Connect Pins To Signals

These signals now need to be assigned to the corresponding symbol pins to create the connections. But since they were automatically generated from the pins, you can just click on **[Automatically assign all signals by name]**:
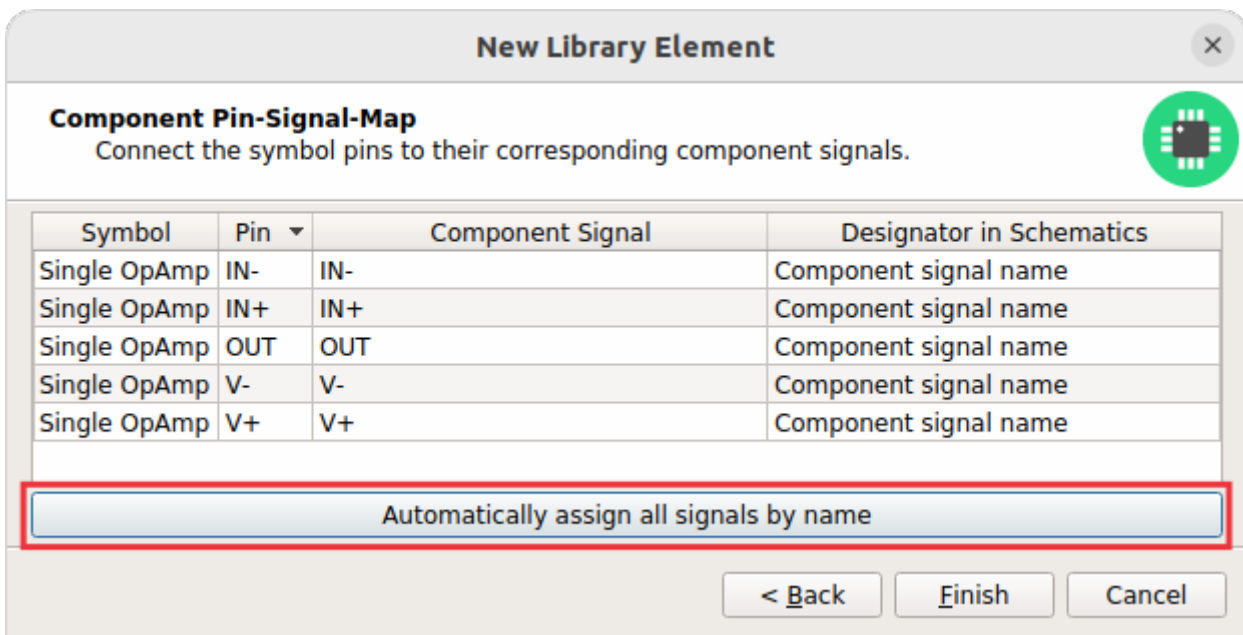
Component Editor

After clicking on **[Finish]**, the component is complete:
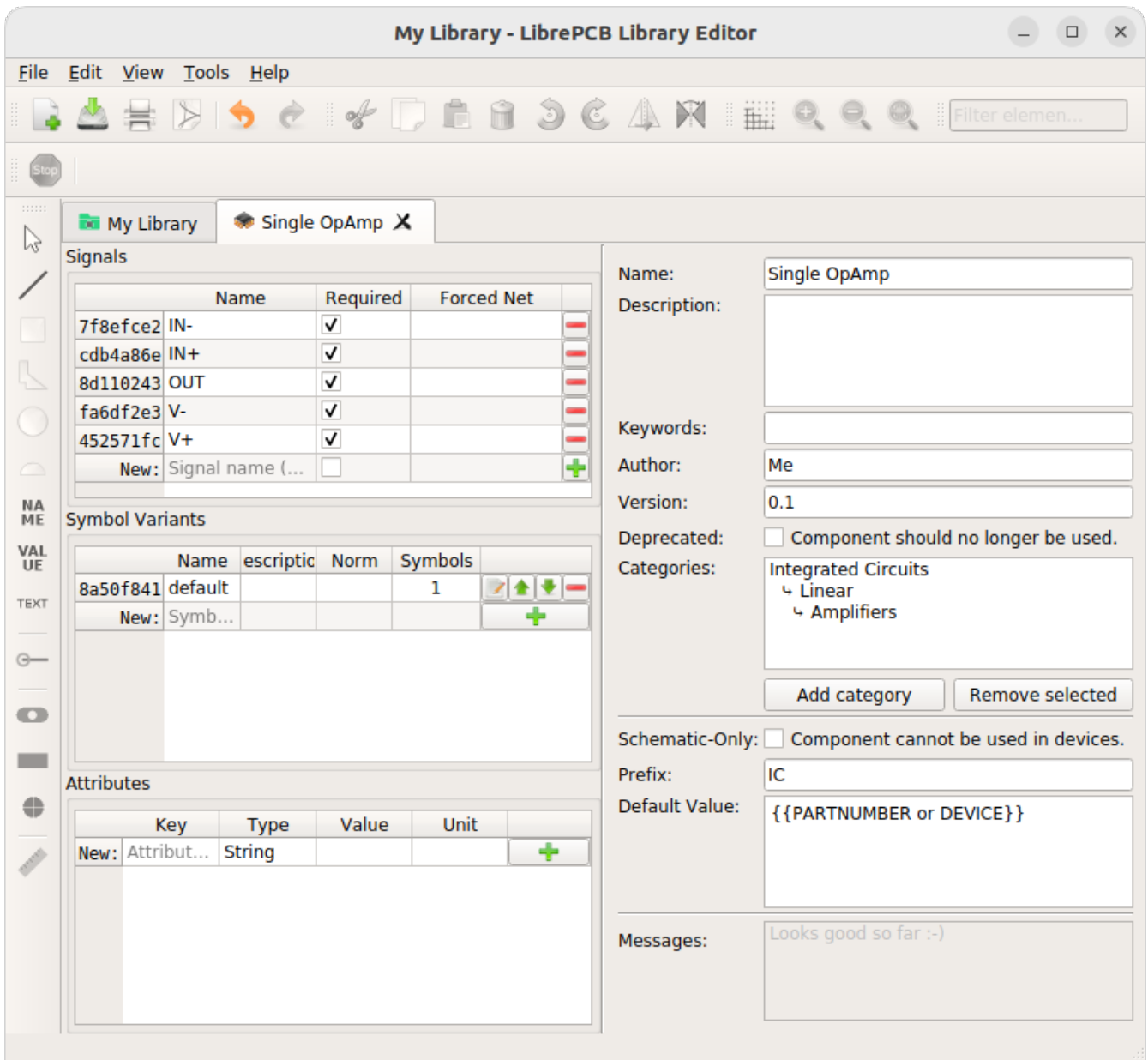
For our simple example this procedure might feel a bit complicated. This is due to the broad flexibility of the LibrePCB library approach which will save time in the long term due to high reusability of library elements.

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 **[Finish]** the package category is complete and after a moment the new category is ready to use.

▼ *Package categories available in the LibrePCB Base library*
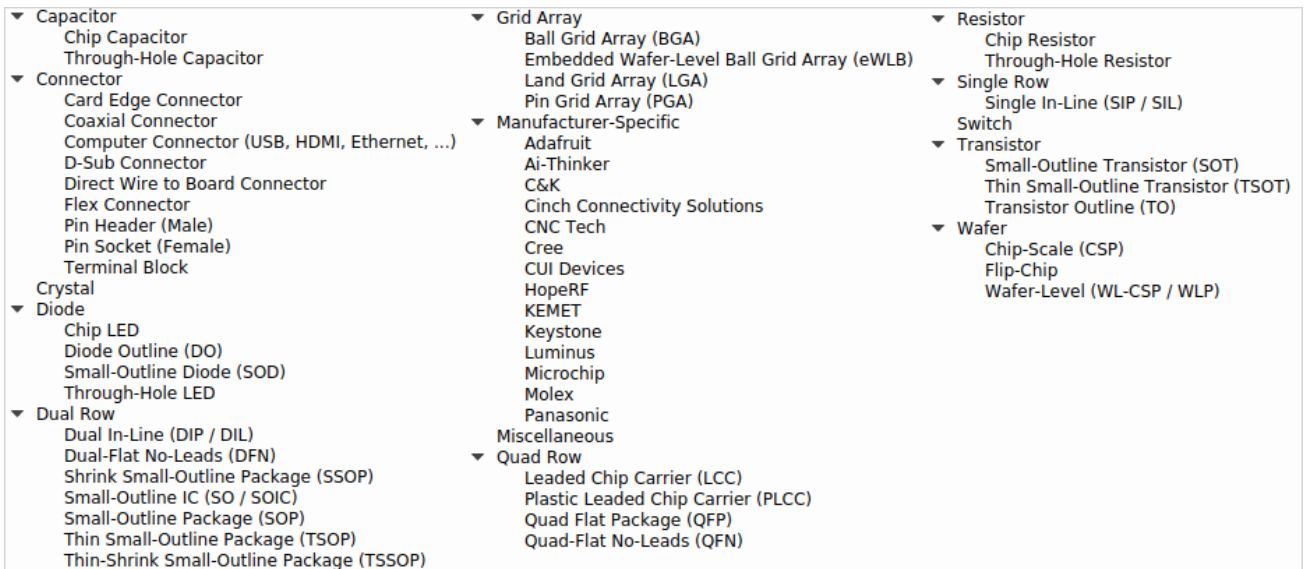
Package

Then you need to create the package for the LMV321LILT, which is called *SOT23-5*. As usual, open
New Library Element › Package and specify the name and category:

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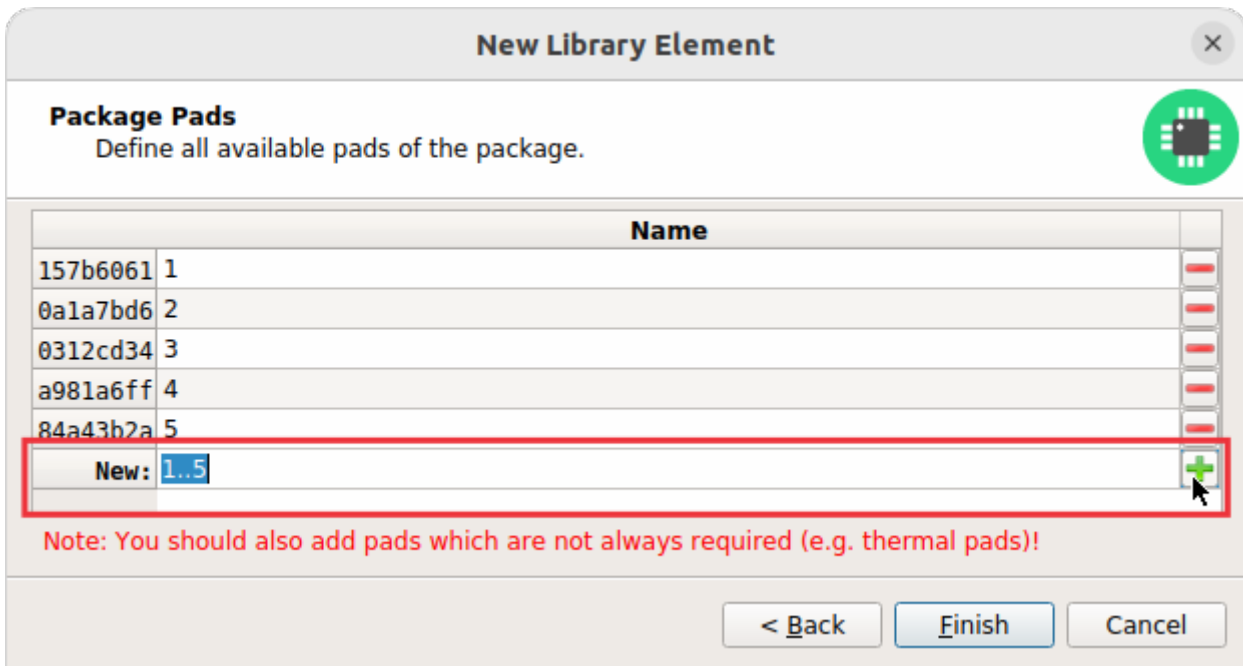
When adding the pads, don't consider their electrical functions or internal connections. For example if a transistor with three electrical signals has three pads plus a thermal pad connected to one of the other signals, the package has **four** pads in total. It's not relevant whether some of them are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

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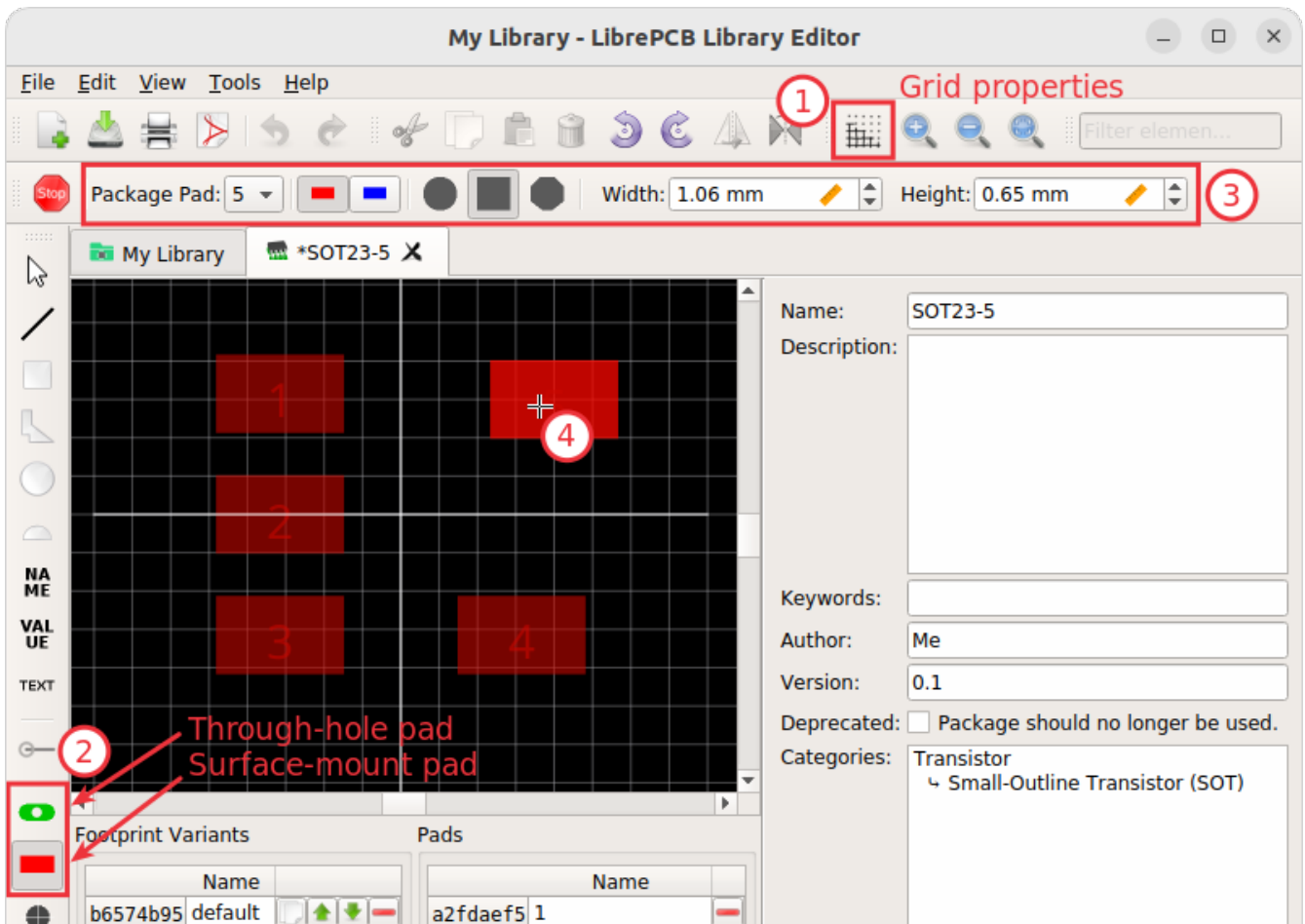
> ℹ️ The tool only allows to place pads on the grid. To specify exact coordinates, just place the pads rougly and open **[Properties]** from the pad's context menu (right-click) afterwards to enter exact values.

Draw Polygons

Then add graphical object just as done in the symbol editor:
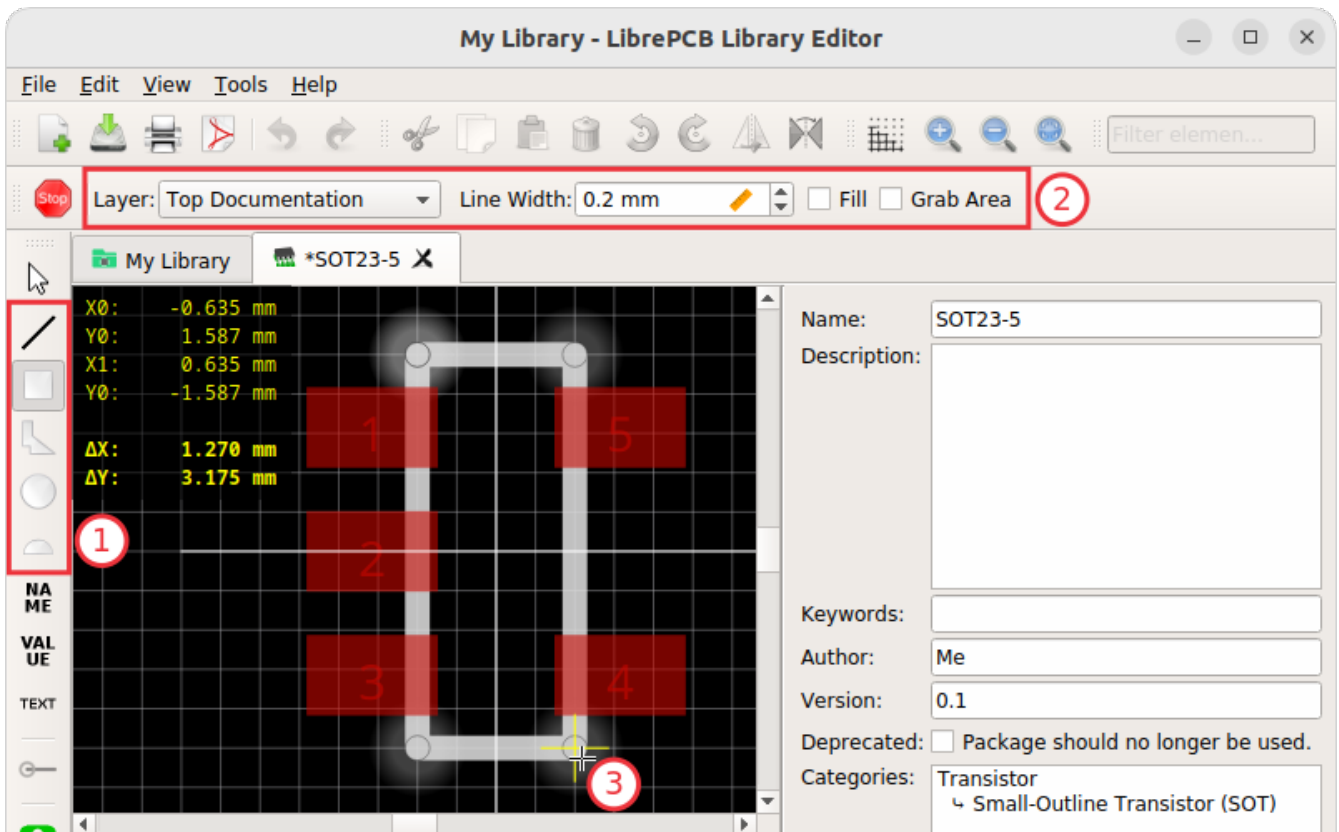
It's recommended to add at least two polygons:

- 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 Placement* 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:
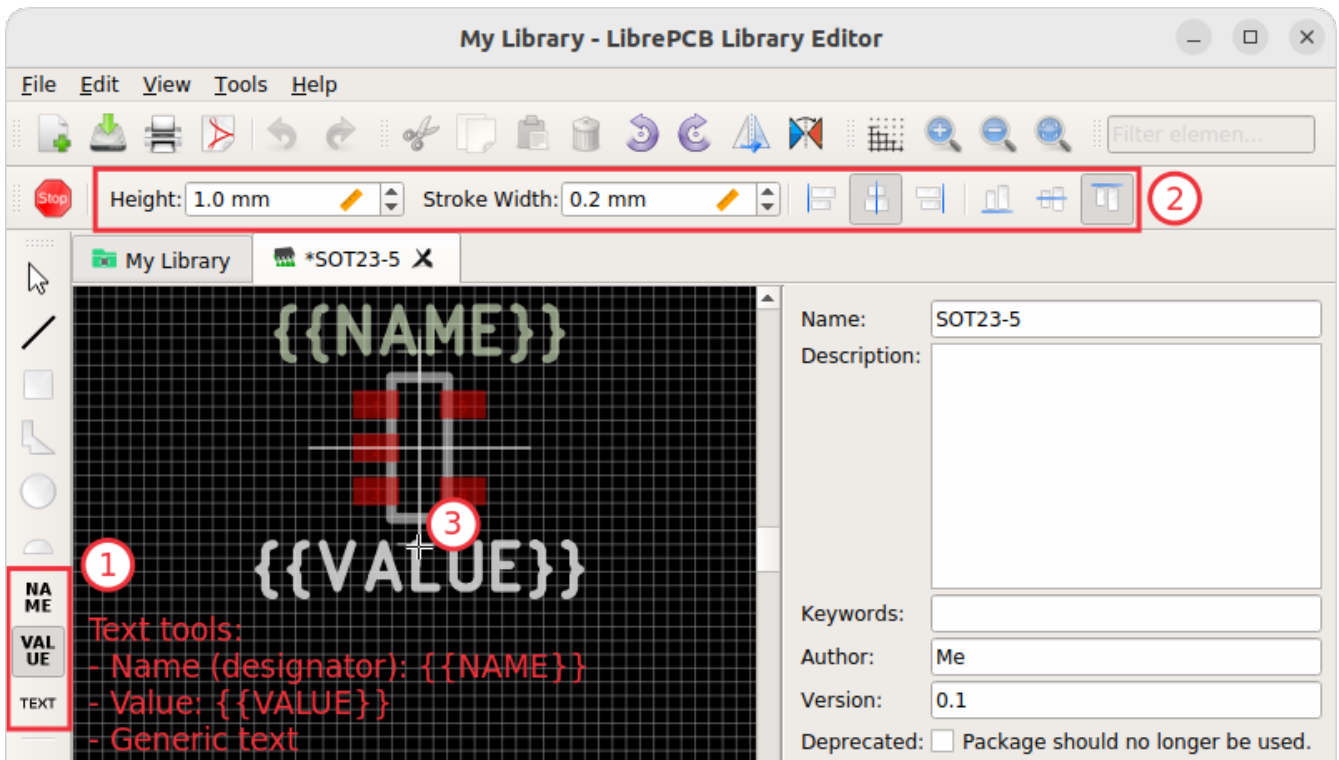
Add Non-Plated Holes

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.

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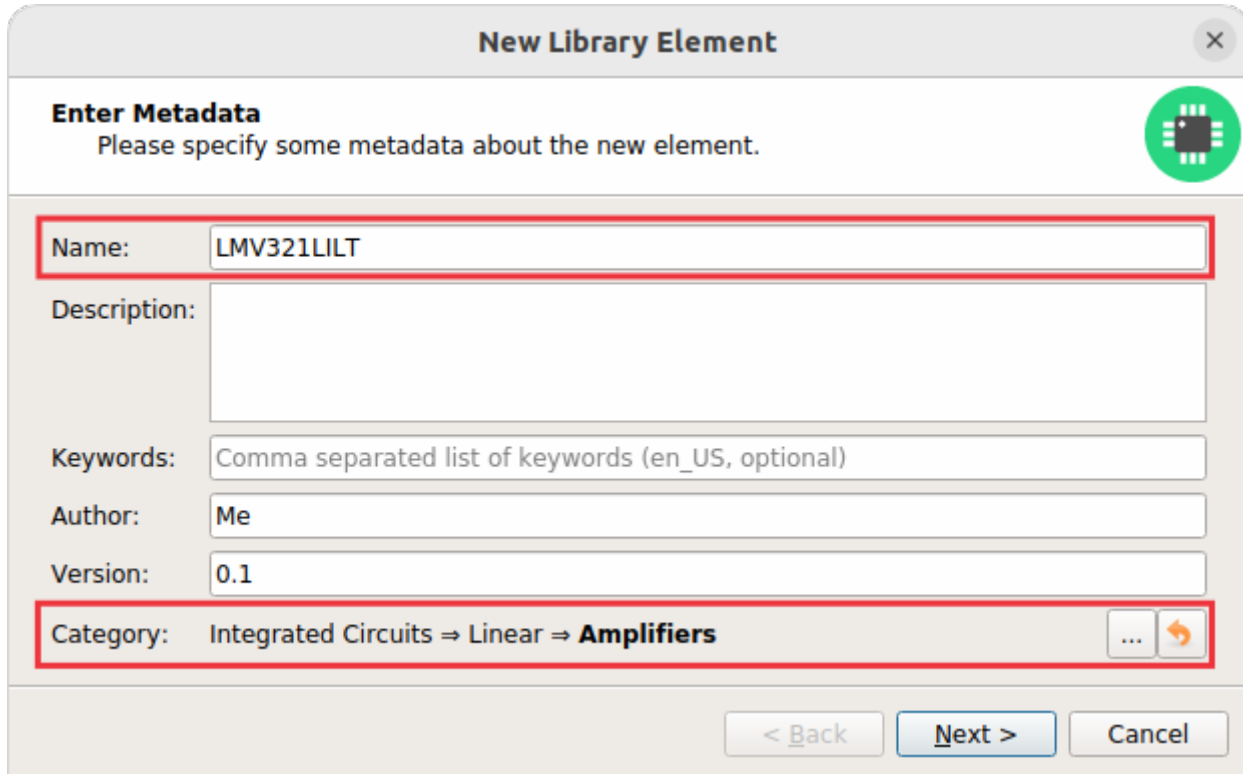
For details about how packages should be designed, please take a look at our package conventions. The most important rules are:

- 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!
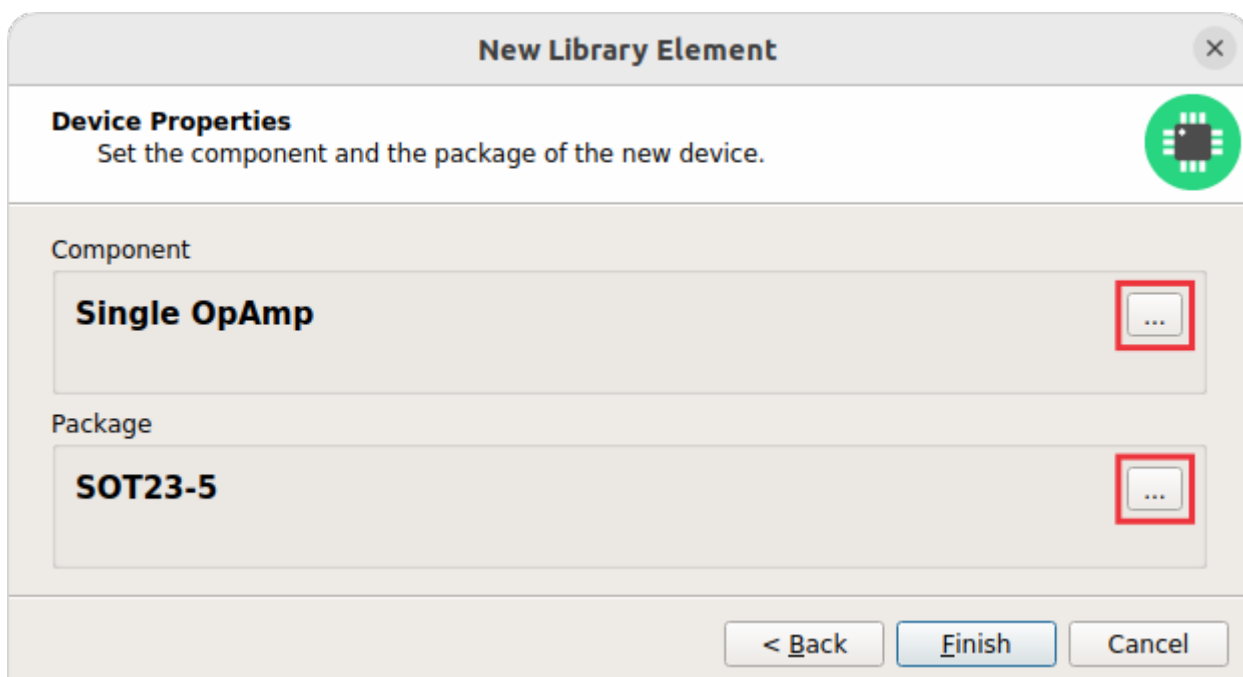
Again, open **New Library Element › Device** and specify the name and category for the new device:

Choose Component & Package

After clicking **[Next]**, you need to choose the component and package we created for this device:

Then click on **[Finish]**.

Connect Pads To Signals

Now you have to connect the package pads to component signals according to the pinout in the datasheet of LMV321LILT:

Then **save the device** to finish it and quickly wait until the background library scan completes before adding the new device to a project.

And that's it! The LMV321LILT is now ready to be added to schematics and boards. 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] Electrical rule check

User Manual

Sorry, the user manual is not available yet :-(

Help us creating it on GitHub!

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.

> ℹ️ *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. But there are two exceptions: The AppImage and the macOS bundle.

Linux AppImage

Download librepcb-cli-0.1.7-linux-x86_64.AppImage, make it executable and run it:

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

macOS Bundle

Download librepcb-cli-0.1.7-mac-x86_64.dmg and double-click it in Finder. Then drag and drop the app onto the "Applications" folder in Finder.

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:0.1.7 --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.
  --verbose      Verbose output.

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

Commands:
  open-library   Open a library to execute library-related tasks.
  open-project   Open a project to execute project-related tasks.

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

Command "open-library"

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

Command

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

Output

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

Options:
  -h, --help     Print this message.
  -v, --version  Displays version information.
```

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


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

Examples

Check and Upgrade File Format of Library Elements

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

Command

```
./librepcb-cli open-library \
    --all \
    --strict \
    MyLibrary.lplib
```

Output

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

Command "open-project"

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

Command

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

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

Options:
  -h, --help                     Print this message.
  -v, --version                  Displays version information.
  --verbose                      Verbose output.
  --erc                          Run the electrical rule check, print all
                                 non-approved warnings/errors and report
                                 failure (exit code = 1) if there are
                                 non-approved messages.
  --export-schematics <file>     Export schematics to given file(s).
                                 Existing files will be overwritten.
                                 Supported file extensions: pdf, svg, png
  --export-bom <file>            Export generic BOM to given file(s).
                                 Existing files will be overwritten.
                                 Supported file extensions: csv
  --export-board-bom <file>      Export board-specific BOM to given
                                 file(s). Existing files will be
                                 overwritten. Supported file extensions: csv
  --bom-attributes <attributes>  Comma-separated list of additional
                                 attributes to be exported to the BOM.
                                 Example: "MANUFACTURER, MPN"
  --export-pcb-fabrication-data  Export PCB fabrication data
                                 (Gerber/Excellon) according the fabrication
                                 output settings of boards. Existing files
                                 will be overwritten.
  --pcb-fabrication-settings <file> 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>        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>     Export pick&place file for automated
                                 assembly of the bottom board side. Existing
                                 files will be overwritten. Supported file
                                 extensions: csv, gbr
  --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.
  --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

Check ERC Messages and Export Schematics & Boards

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

Command

```
./librepcb-cli open-project \
    --erc \
    --export-schematics="output/{{VERSION}}/{{PROJECT}}_Schematics.pdf" \
    --export-pcb-fabrication-data \
    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"
Export schematics to 'output/{{VERSION}}/{{PROJECT}}_Schematics.pdf'...
  => 'output/v1/MyProject_Schematics.pdf'
Export PCB fabrication data...
  Board 'default':
    => 'output/v1/gerber/MyProject_DRILLS-PTH.drl'
    => 'output/v1/gerber/MyProject_OUTLINES.gbr'
    => 'output/v1/gerber/MyProject_COPPER-TOP.gbr'
    => 'output/v1/gerber/MyProject_COPPER-BOTTOM.gbr'
    => 'output/v1/gerber/MyProject_SOLDERMASK-TOP.gbr'
    => 'output/v1/gerber/MyProject_SOLDERMASK-BOTTOM.gbr'
    => 'output/v1/gerber/MyProject_SILKSCREEN-TOP.gbr'
    => 'output/v1/gerber/MyProject_SILKSCREEN-BOTTOM.gbr'
```

```
Finished with errors!
```

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

Library Conventions

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

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

Symbol Conventions

> ℹ️ 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)
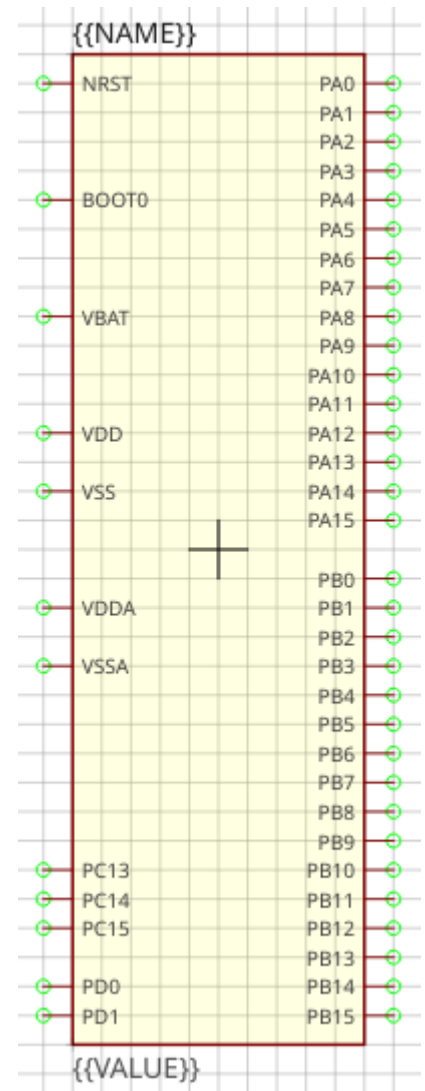
Origin

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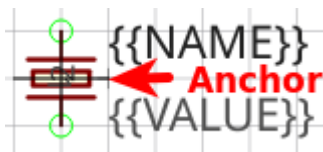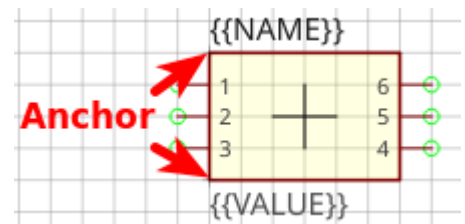
Irregularly shaped symbols may have text elements placed differently, see for example the crystal at the left. Keep in mind that the value of a component can consist of several lines, so there should always be enough space available for it.

Typical text element properties

Property	Name text element	Value text element
Layer	*Names*	*Values*
Text	*{{NAME}}*	*{{VALUE}}*
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:
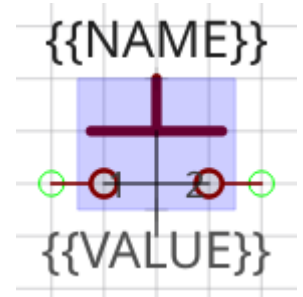
- **Layer**: *Hidden Grab Areas* (will not be visible in the schematic editor)
- **Line Width**: *0.0 mm*
- **Filled**: *yes*
- **Grab Area**: *yes*

> 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 1. Color (e.g. 0805 LED)

> Even if a 0805 LED with a transparent lens has exactly the same footprint as a 0805 LED with a

red lens, they should have **separate packages because of the different color**. This way a device can link to the package with the proper color, and thus it will appear with the proper color in the 3D PCB preview (once LibrePCB supports 3D models).

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

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

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

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

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

Naming

The following conventions apply to package names:

- We generally **follow IPC-7351 when naming packages** (e.g. "SOT23-5P95_280X145L60" instead of "SOT23-5"). Alternative names (like "SOT23-5") should be added to the comma-separated keywords list and maybe to the description.

- For packages not covered by IPC-7351, use following naming conventions:

  - **Language must be American English** (en_US), if applicable (many packages have language-neutral names anyway).

  - **Size information must use metric units**, not imperial units.

  - For packages which are available with different pin counts, **append the pin count with a hyphen as separator and omit leading zeros** (e.g. "DIP-8" instead of "DIP08").

- For packages which are well known by their size in imperial units (e.g. "0805" which is "2012" in metric), it's recommended to write the well known name in parentheses. For example, a chip resistor could be named "RESC2012X70 (0805)".

- The name of manufacturer-specific packages should start with the manufacturers name (e.g. "Molex 53261-06"). *Note: Libraries do not act as namespaces for package names, so you should start the package name with the manufacturers name even if the package is located in a manufacturer-specific library.*

Pads

- **Always add all pads of packages**, even those which are not always connected. For example, the

package "TO220" has a metal tab, so you should define it as a pad, no matter if it's often not connected (and even not connectable when mounted vertically).

- **Use pad names according IPC-7351** (if applicable). For packages which are not covered by IPC-7351:

  - If the function of a pad is absolutely clear, choose its abbreviated functionality as name (e.g. "A" for anode and "C" for cathode).

  - Otherwise just use numbers starting with "1" at top left and increment them counterclockwise.

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:
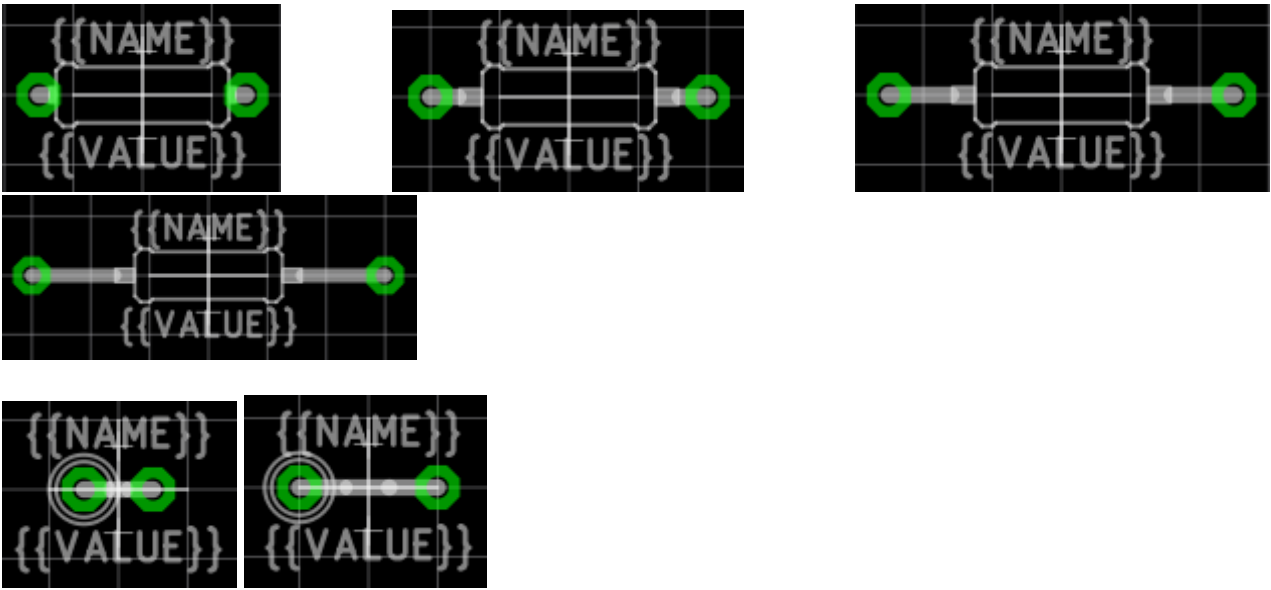
- Generic packages: Designed according to IPC density level B (if applicable)

- 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 4. 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.
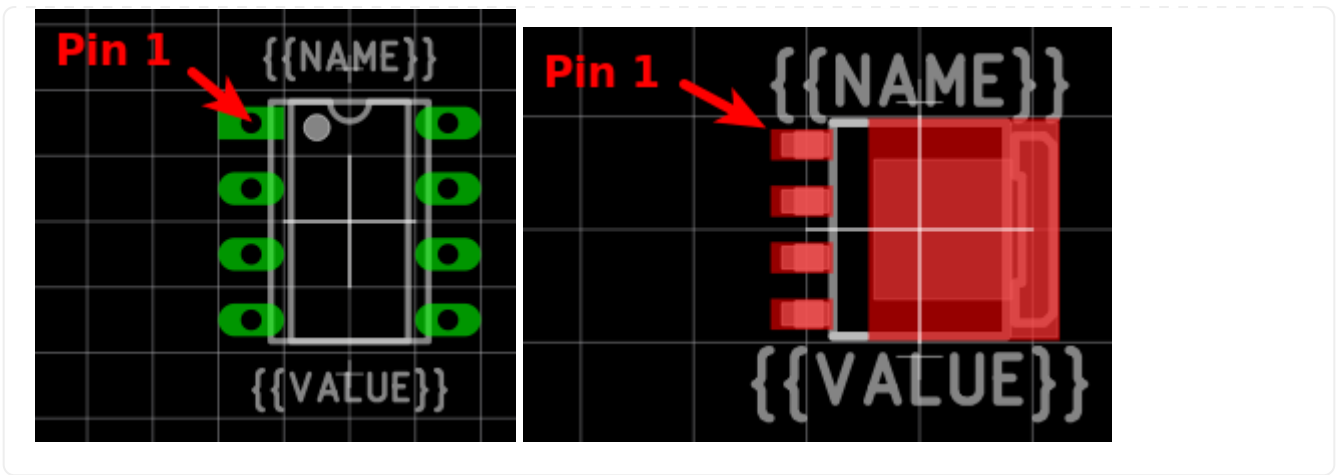
Orientation

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 5. Footprint orientation examples*
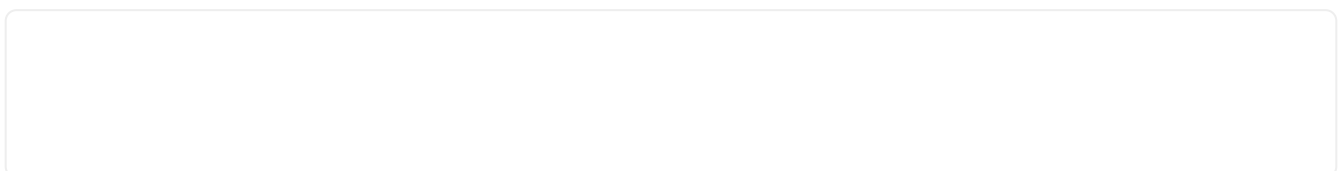
Placement Layer

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

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

The placement should be drawn according to IPC-7351C. The most important rules are the following:

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

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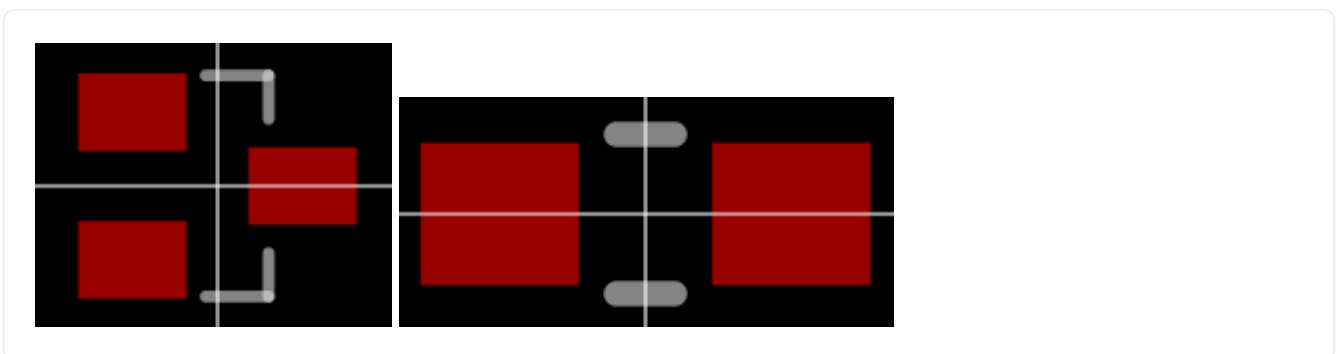

Documentation Layer

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 7. Documentation layer examples (only documentation and copper layers shown)
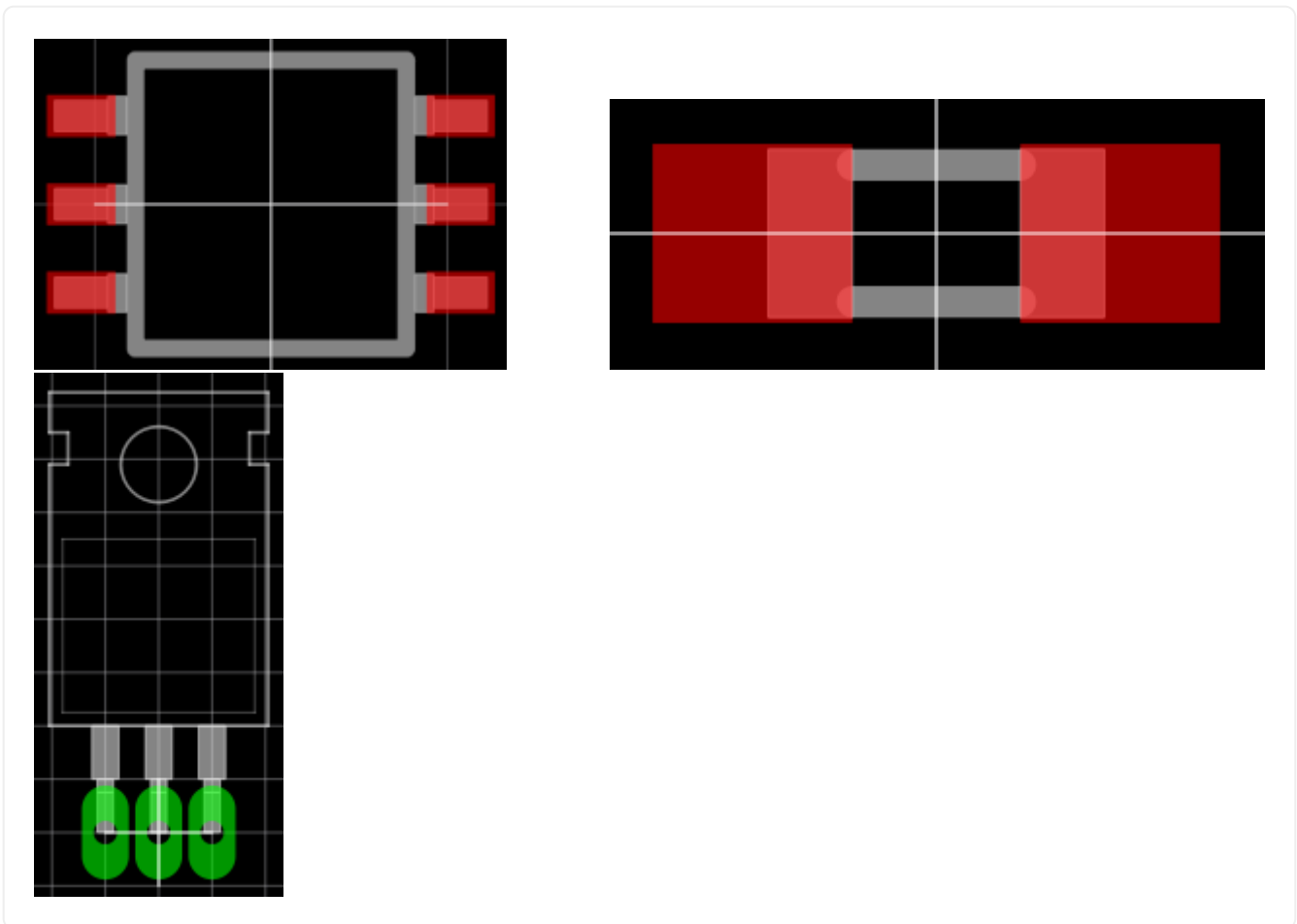
Text Elements

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.
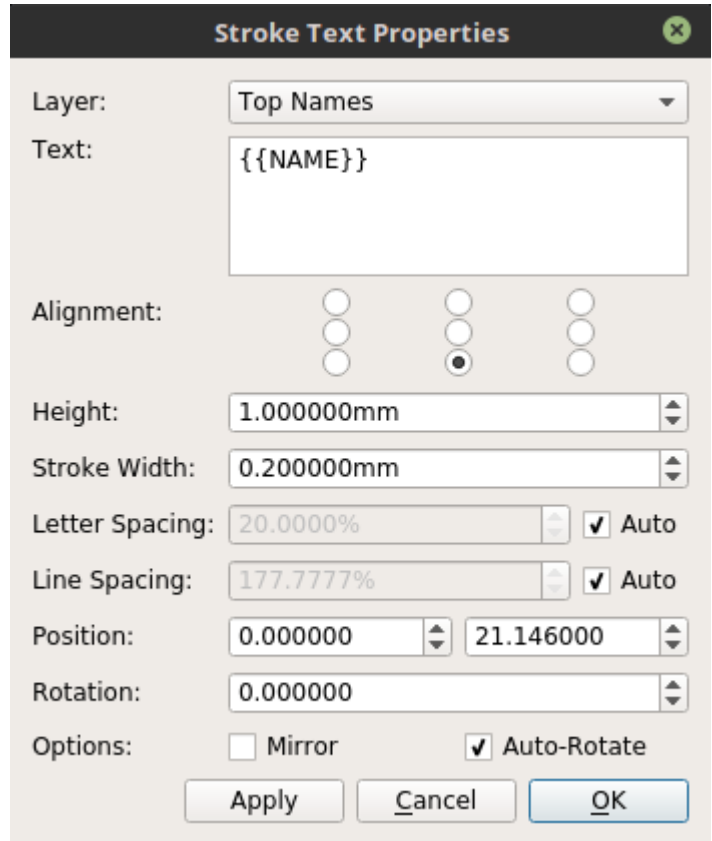

Figure 1. Typical footprint name properties

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 8. Footprint text element examples
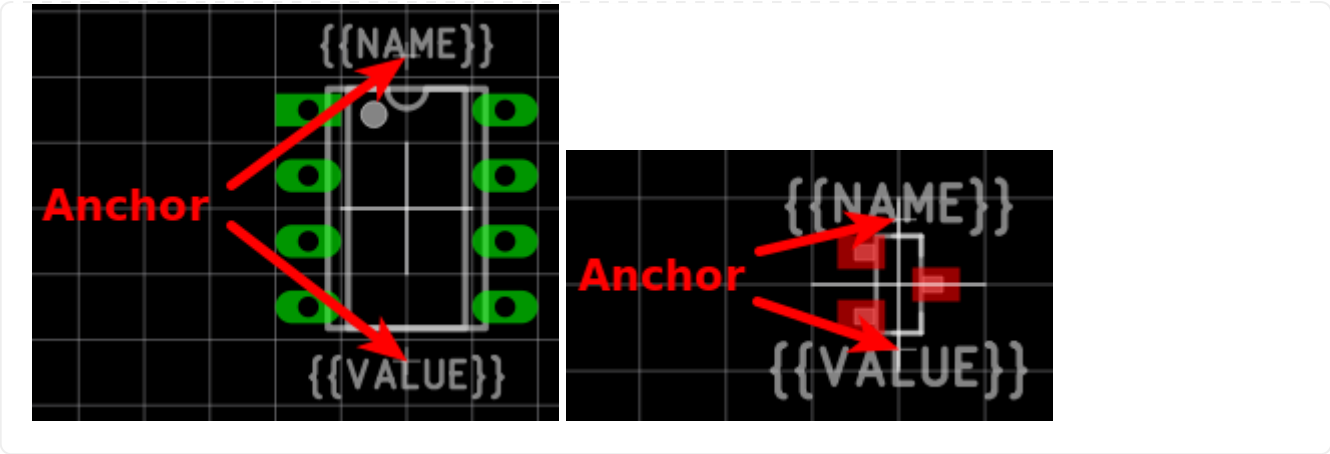
Development

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