

- Switches
- Voltage and current sources
- Diodes
- Transistors (BJTs, MOSFETs, MESFETs, and JFETs)
- XSPICE code models
- Raw SPICE elements

To add a built-in model to a symbol, open the Simulation Model Editor dialog (**Symbol Properties** → **Simulation Model...**) and select **Built-in SPICE model**. You can then select the kind of device from the **device** dropdown and the device subtype from the **device type** dropdown.

Refer to the ngspice documentation [<https://ngspice.sourceforge.io/docs.html>] for more details about these models and their parameters.

Simulation Model Editor

Model Pin Assignments

☐ SPICE model from file (*.lib, *.sub or *.ibs)

File:

Model:

☒ Built-in SPICE model

Device:

Device type:

Parameters Code

Parameter	Value	Unit	Default	Type
Resistance (r)	400	Ω		Float

☐ Save parameter 'Resistance (r)' in Value field

Cancel OK

Device sets the type of device to simulate: a resistor, BJT, voltage source, etc. This value is stored in the symbol's `Sim.Device` field.

Device type selects the type of model to use for the device. Most devices have several types of models to choose from. Models may vary in their degree of accuracy, which characteristics they are optimized for, what parameters they have available, and how many pins they have. For example, the **ideal** resistor type models a simple resistor with two terminals and a single resistance parameter, while

the **potentiometer** resistor type models an adjustable resistor with three terminals and an additional parameter for wiper position. Some devices have an especially large number of types to choose from: N-channel MOSFETs, for example, have 17 available types, each of which uses a different mathematical model to simulate the transistor behavior. One model may be more or less appropriate than another for simulating a specific device or circuit or for performing a particular analysis. Refer to the ngspice documentation [<https://ngspice.sourceforge.io/docs.html>] for detailed information about models and their parameters. The **device type** value is stored in the symbol's `Sim.Type` field.

The **parameters** tab displays the parameters of the model and lets you edit them. For example, a resistor's resistance, a voltage source's waveform, a MOSFET's width and length, etc. Any parameters that differ from the model's defaults are stored in the symbol's `Sim.Params` field.

The **code** tab displays the generated SPICE model as it will be written to the SPICE netlist for simulation.

The **Save parameter <parameter name> in Value field** checkbox uses the symbol's `Value` field for storing parameters instead of the `Sim.Params` field. This may make it easier to edit simple models from the schematic, without opening the Simulation Model Editor. This option is only available for ideal passive models (R, L, C) and DC sources. If the field `Sim.Params` exists in the symbol, it will take priority over the `Value` field.

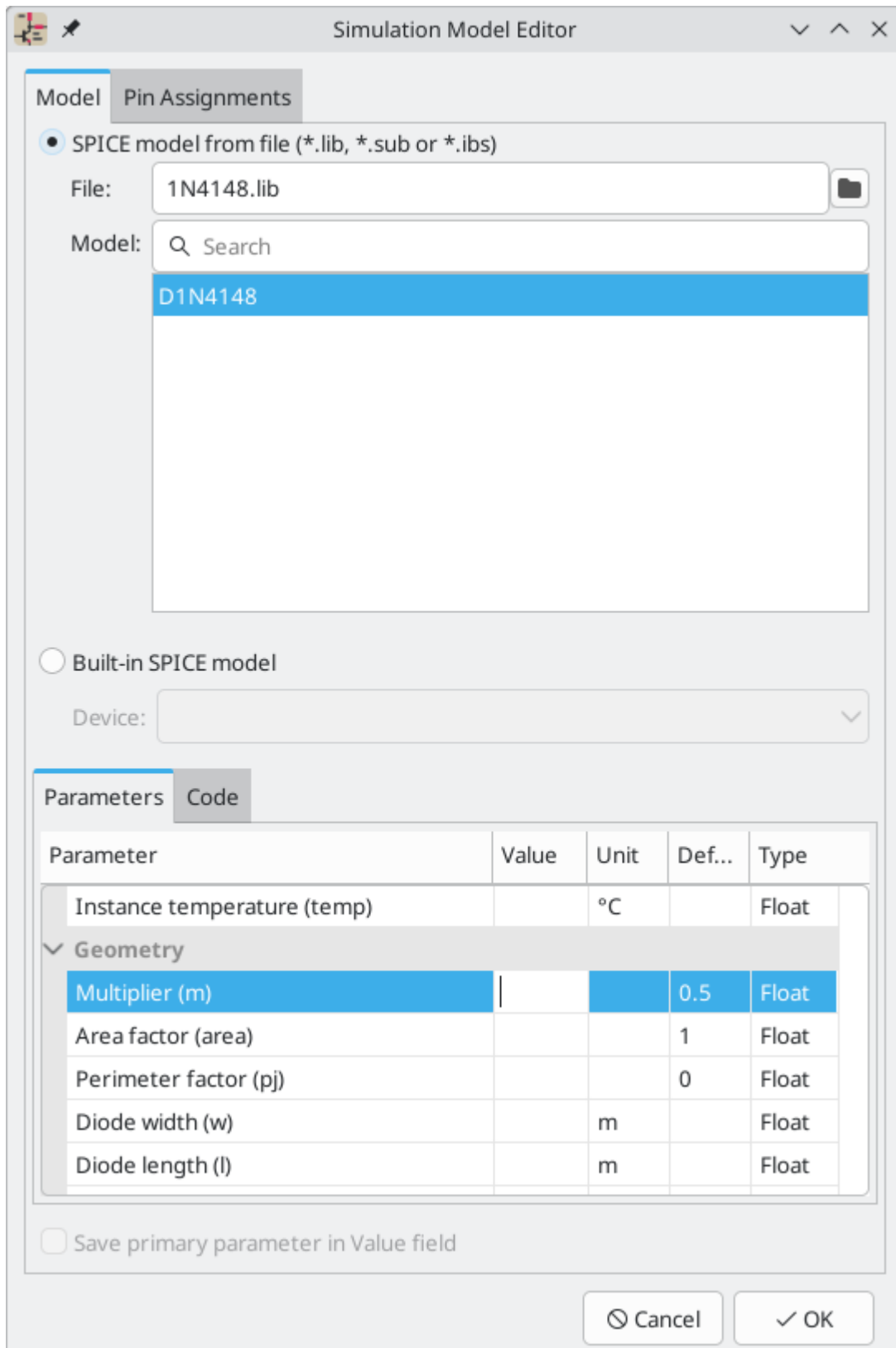
10.1.3. External models

KiCad can also load SPICE models from external files. This is typically how you will add a SPICE model of a specific commercially-available part (for example, a 555 timer or a TL071 operational amplifier) to your simulation. Models such as these are readily available from numerous sources, including manufacturers' web sites. These models must be in a standard SPICE format and must not be encrypted.

An external model can be one of the following types:

- A device model (`.model`). This is an intrinsic device (a passive, diode, transistor, etc.) with a set of parameter values defining its behavior. The parameters for a device model are editable in the Simulation Model Editor GUI.
- A subcircuit model (`.subckt`). This is a model that uses a collection of other ngspice circuit elements to define its behavior. If a subcircuit model contains parameters (in a `params:` sequence in its definition), the parameters are editable in the Simulation Model Editor GUI.

To load a model from an external file, open the Simulation Model Editor dialog (**Symbol Properties** → **Simulation Model...**) and select **SPICE model from file**.



File is the path to the model file to use. Unencrypted model files are plain, human-readable text files and often have extensions such as .lib, .sub etc., although KiCad will accept a valid model with any extension.

The path to the file can be absolute, or relative to the project folder. The path can also be relative to the value of SPICE_LIB_DIR if you have defined that path variable. If you enable the **Embed File** checkbox in the file browser, the library file will be embedded in the schematic (or symbol library).

This makes the schematic (or schematic) more portable as it doesn't rely on an external library file. The library filename is saved in the symbol's `Sim.Library` field.

Model is the name of the desired model in the model file. A model file may contain multiple models, and if so they will all be shown in the list. You can filter the list of models using the search box. The selected model is listed in the symbol's `Sim.Name` field.

Parameters can be overridden (or additional parameters specified) using the **Parameters** tab. For device models, all parameters for that type of device are editable. For subcircuit models, any parameters included in the subcircuit definition are editable. Any parameters that are overridden in the **Parameters** tab are stored in the symbol's `Sim.Params` field.

The **Code** tab displays the generated SPICE model as it will be written to the SPICE netlist for simulation.

Uwaga

KiCad is not distributed with SPICE models for specific commercial devices. These models are usually available from device manufacturers or other internet sources.

10.1.4. IBIS models

IBIS (I/O Buffer Information Specification) files are an alternative to SPICE models for modeling the behavior of input/output buffers on digital parts. In order to load an IBIS file, users should follow the procedure for SPICE library models, but provide a `.ibs` file.


Simulation Model Editor
⌵ ⌶ ✕

Model

Pin Assignments

☒ SPICE model from file (*.lib, *.sub or *.ibs)

File:

ibis_v1_1.ibs



Component:

Virtual

⌵

Pin:

4 - Y

⌵

Model:

Output

⌵

☐ Built-in SPICE model

Device:

IBIS Model

⌵

Type:

Rectangular wave driver

⌵

Parameters

Code

Parameter	Value	Unit	Def...	Type
Power supply (vcc)	typ		typ	String
Parasitic pin resistance (rpin)	typ		typ	String
Parasitic pin inductance (lpin)	typ		typ	String
Parasitic pin capacitance (cpin)	typ		typ	String
⌵ Waveform				
ON time (ton)	100n	s		Float
OFF time (toff)	100n	s		Float
Delay (td)	0	s	0	Float
Number of cycles (n)	50		1	Int

☐ Save primary parameter in Value field

⊗ Cancel

✓ OK

File is the path to the model file to use. The path can be absolute or relative to the project folder. The path can also be relative to the value of `SPICE_LIB_DIR` if you have defined that path variable. The library filename is saved in the symbol's `Sim.Library` field. If an IBIS model file is loaded, the remaining fields in the dialog will relate to the IBIS model.

Component selects which component from the IBIS file to use, as IBIS files can contain multiple components. The component name is saved in the symbol's `Sim.Name` field.

Pin selects which pin in the IBIS model to simulate. The selected pin must be mapped to a symbol pin in the **Pin Assignments** tab. The chosen pin's number is saved in the symbol's `Sim.Ibis.Pin` field.

Model is the list of models available for the selected pin, for example an input or an output. The chosen model name is saved in the symbol's `Sim.Ibis.Model` field.

Type selects what the pin should do in the simulation. A pin can be a passive **device** that doesn't drive any value; it can be a **DC driver** that drives high, low, or high-impedance; or it can be a **rectangular wave** or **PRBS driver**. This value is stored in the symbol's `Sim.Type` field.

The **Parameters** tab lets you see and edit the parameters of the model. For each parameter, you can switch between a minimum, typical, or maximum value, as defined in the IBIS file. You can also choose the parameters of the driven waveform, depending on the pin's chosen **type**. Any parameters that differ from the defaults are stored in the symbol's `Sim.Params` field.

Uwaga

KiCad does not ship with IBIS models for symbols. IBIS models are usually available from device manufacturers.

Uwaga

KiCad's `Simulation_SPICE` symbol library provides several symbols that may be useful for IBIS simulations. `IBIS_DEVICE` can be used for device (input) pins, while `IBIS_DRIVER` can be used for simulating driver pins. There are also variants of each for differential pins.

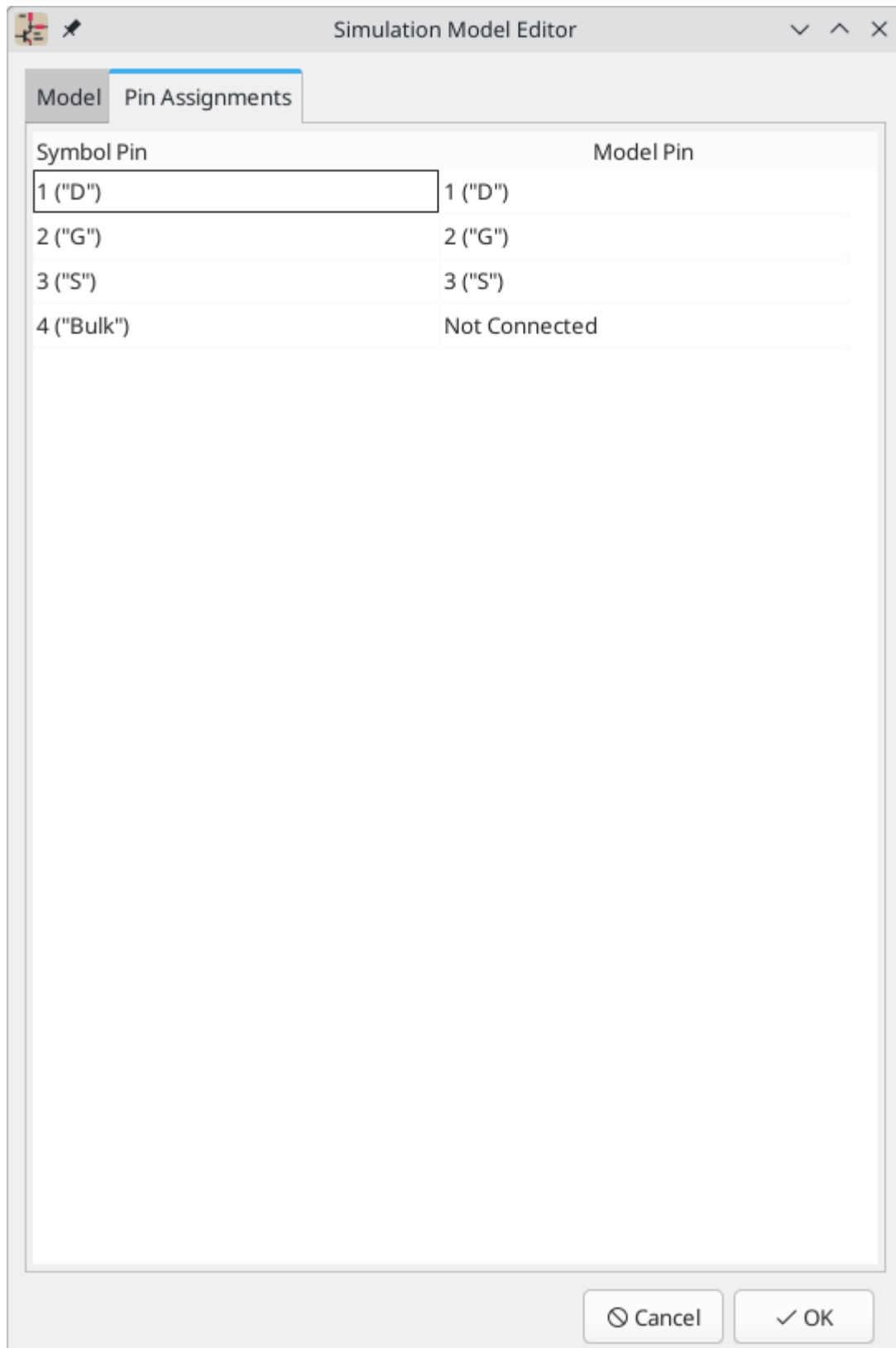
10.1.5. Pin Assignment

Simulation models may have their pins numbered differently than the corresponding symbol. For example, SPICE models for diodes usually consider pin 1 to be the anode, while schematic symbols are usually drawn with pin 1 as the cathode. Operational amplifier models are also very likely to have model pin assignments that do not match package or schematic pin numbers.

You can use the Simulation Model Editor's **Pin Assignments** tab to map the symbol's pins to the simulation model pins.

Uwaga

Always make sure symbol pins are correctly mapped to simulation model pins. Mistakes here can lead to erroneous or confusing simulation results, or a failure to simulate at all.



The left column displays the name and number of each symbol pin, i.e. the pin numbers and names that appear on the schematic part in KiCad. The right column displays the corresponding pin as defined in the model file in use. For each symbol pin, you can select the corresponding pin from the simulation model in the dropdown in the right column. In the cases where a schematic part has pins that are not in the model, as in the case of an operational amplifier with *nulling* pins that are not modeled, the schematic part pin may be assigned to the *Not Connected* option in the Pin

Assignments dropdown. Unlike other pin assignments, *Not Connected* may be assigned to multiple pins if necessary.

When you use a subcircuit model, the dialog displays the model's code under the pin assignments for use as a reference while assigning pins. A well-written model will often include a helpful reference section (as a set of comments) to inform the user how the model pins are mapped.

10.2. Value notation

The simulator supports several notations for writing numerical values in simulation model parameters, simulation analysis setup options, and SPICE directives:

- Plain notation : 10100, 0.003,
- Scientific notation: 1.01e4, 3e-3,
- Prefix notation: 10.1k, 3m.
- RKM notation: 4k7, 10R.

You can mix prefix and scientific notations. As such, 3e-4k is a valid input and is equivalent to 0.3. The list of valid prefixes is shown below. They are case sensitive.

Prefix	Name	Multiplier
a	atto	10^{-18}
f	femto	10^{-15}
p	pico	10^{-12}
n	nano	10^{-9}
u	micro	10^{-6}
m	milli	10^{-3}
k	kilo	10^3
M	mega	10^6
G	giga	10^9
T	tera	10^{12}
P	peta	10^{15}
E	exa	10^{15}

Uwaga

Raw SPICE Element models and directives are passed to ngspice directly, without KiCad reformatting the values for ngspice to consume. ngspice uses a different, case-insensitive notation: 1 mega (10^6) is denoted there as 1Meg, while 1M is 1 milli (10^{-3}). Depending on the compatibility mode selected, ngspice may not support the same value notations as KiCad, so care should be taken when using raw SPICE elements and simulation directives.

10.3. SPICE directives

It is possible to add SPICE directives by placing them in text fields on a schematic sheet. This approach is convenient for defining the default simulation type. The list of supported directives in text fields is:

- Directives starting with a dot (e.g. `.tran 10n 1m`)
- Coupling coefficients for inductors (e.g. `K1 L1 L2 0.89`)

It is not possible to place additional components using text fields.

If a simulation command is included in schematic text, the simulator will use it as the simulation command when you open the simulator. However, you can override it in the Simulation Command dialog.

Refer to the ngspice documentation [<https://ngspice.sourceforge.io/docs.html>] for more details about SPICE directives.

10.4. Running simulations

Circuits for simulation are drawn in the Schematic Editor, but simulations are run in the Simulator window.

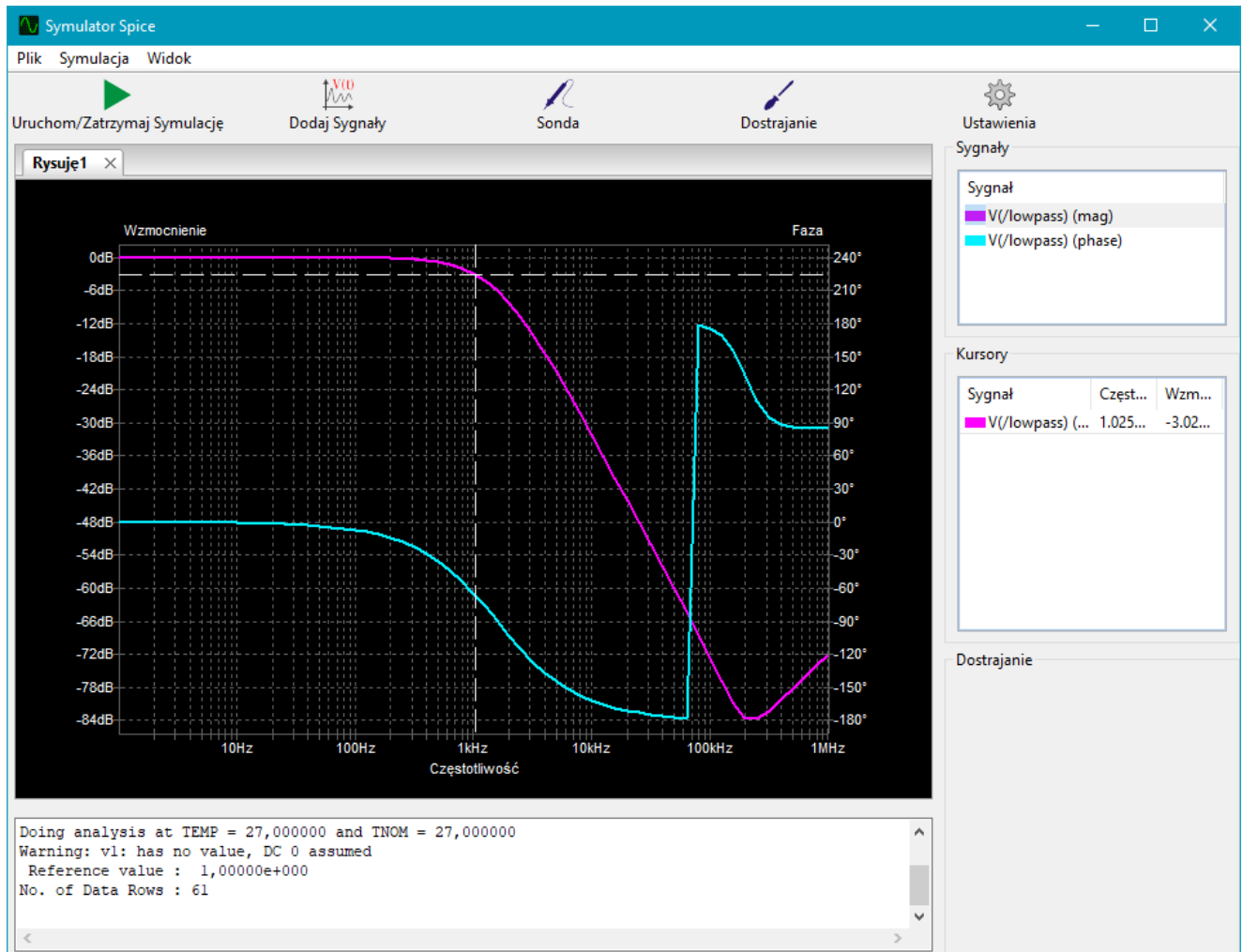
After creating a schematic for simulation, the following steps are needed to run a simulation:

1. Open the Simulator window by clicking **Inspect** → **Simulator** in the Schematic Editor or using the  button in the top toolbar.
2. Create at least one analysis by clicking **Simulation** → **New Analysis Tab...** (kbd:[Ctrl+N]) or using the  button. This lets you choose a type of simulation and configure its options. These options are explained below. Each analysis has its own tab, and multiple analyses can be created. The options for an analysis can be adjusted after it is created with **Simulation** → **Edit Analysis Tab...** or by clicking on the  button.
3. Run the simulation by clicking **Simulation** → **Run Simulation** (kbd:[R]) or by clicking the  button. This only runs the simulation in the active analysis tab. You can stop a running simulation at its current point by clicking the  button.
4. Examine the simulation results. Most analyses result in plots; for these you will need to select the signals to be plotted. Other analyses print their results in the output log window.

Uwaga

Once a simulation has been set up, the configuration can be saved in a workbook.

10.4.1. Simulator window



The simulator window is divided into several sections:

- The top of the window has a toolbar with buttons for commonly used actions.
- The main part of the window graphically shows the simulation results. The simulation needs to run and signals need to be selected from the list of available signals or probed before they are displayed in the plot.
- Below the plot panel, the output log window shows logs from the ngspice simulation engine. Some types of analyses print their results here.
- The right side of the window displays a list of signals, a list of active cursors, measurements, and a tuning tool for adjusting component values based on simulation results.

10.4.2. Simulation types

Each simulation is a specific type of analysis. The following analysis types are available:

- **OP** — DC Operating Point
- **DC** — DC Sweep Analysis
- **AC** — AC Small-signal Analysis
- **TRAN** — Transient Analysis
- **PZ** — Pole-zero Analysis
- **NOISE** — Noise Analysis
- **SP** — S-parameter Analysis
- **FFT** — Frequency-content Analysis

Each analysis type and its options are explained below. You can configure an analysis when you create it (



button) or by editing an existing analysis (

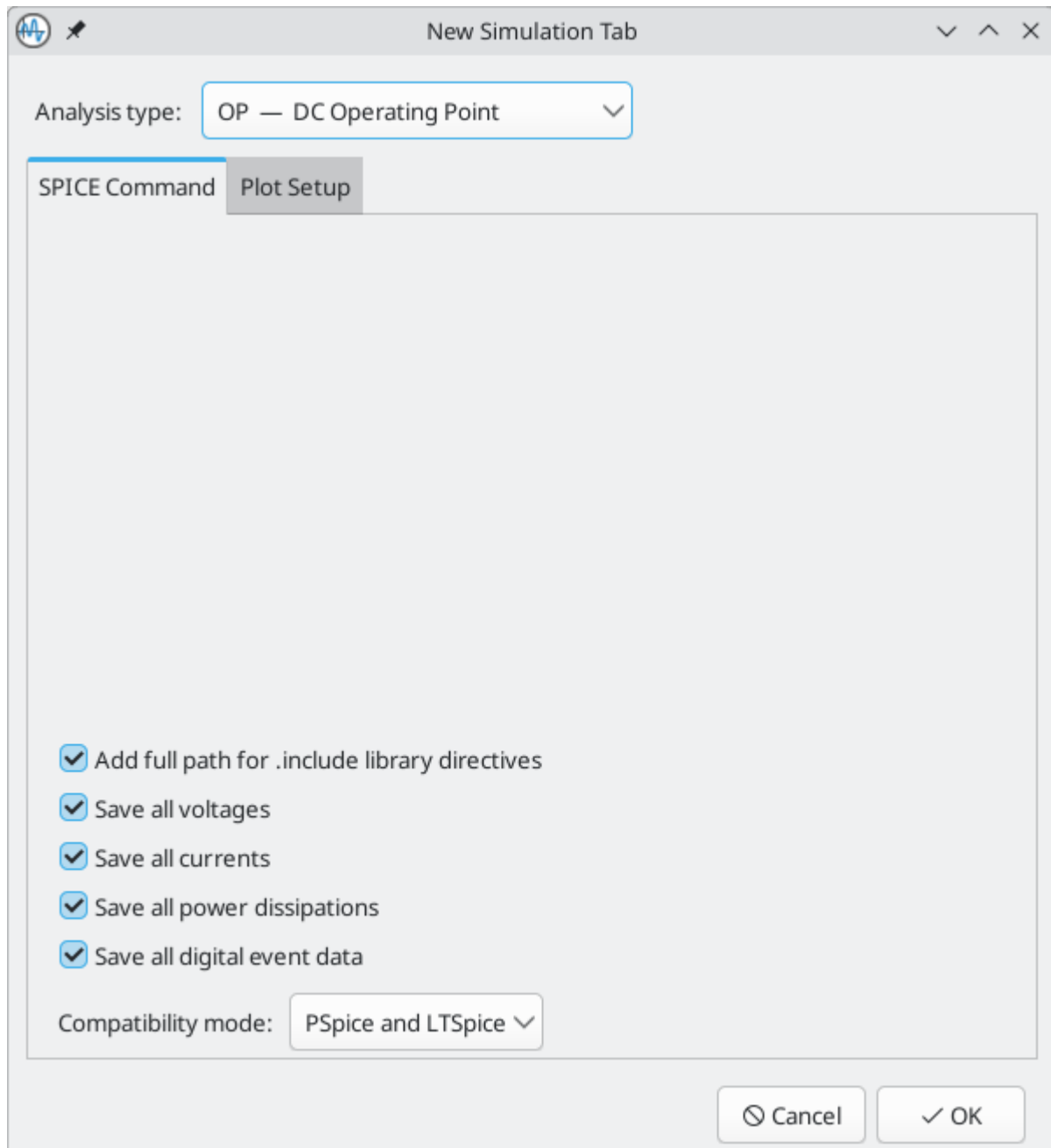


button). These analysis types are explained in more detail in the ngspice documentation [<https://ngspice.sourceforge.io/docs.html>].

Uwaga

Another way to configure a simulation is to type SPICE directives into text fields on schematics. Any text field directives related to a simulation command are overridden by the settings selected in the dialog. This means that once a simulation has run, the dialog overrides the schematic directives until the simulator is reopened.

Operating point analysis (OP)



Calculates the DC operating point of the circuit. This analysis has no options.

Operating point analyses do not have any plotted results. Results are printed in the output log window. You can also display the node voltages and device currents calculated by this analysis as annotations in the schematic.

DC sweep analysis (DC)

New Simulation Tab

Analysis type: DC — DC Sweep Analysis

SPICE Command Plot Setup

Source 1 ☐ **Source 2**

Sweep type: V V

Source: V V

Starting value: V V

Final value: V V

Increment step: V V

Swap sources

☒ Add full path for .include library directives

☒ Save all voltages

☒ Save all currents

☒ Save all power dissipations

☒ Save all digital event data

Compatibility mode: PSpice and LTSpice

Cancel OK

Calculates the DC behavior of the circuit while sweeping one or two parameters. The following parameters can be swept:

- value of an independent voltage source
- value of an independent current source
- value of a resistor
- simulation temperature

DC analyses have the following options, which are listed here with the corresponding ngspice parameter name:

- **Sweep type:** the type of variable to sweep. This can be a voltage source, a current source, a resistor, or the simulation temperature.
- **Source:** the particular voltage source, current source, or resistor to sweep (srcnam). The list is populated with each item in the schematic of the relevant type. It is disabled for temperature sweeps.
- **Starting value:** the starting value for the sweep (vstart).
- **Final value:** the ending value for the sweep (vstop).
- **Increment step:** the amount to increase the value at each step of the sweep (vincr). Smaller increments result in more output points.

If **Source 2** is enabled, the same options are available to simultaneously sweep a second source. Clicking the **Swap sources** buttons swaps Source 1 with Source 2.

The output is displayed as a plot.

AC small-signal analysis (AC)

New Simulation Tab

Analysis type: AC — Small-Signal Analysis

SPICE Command Plot Setup

Number of points per decade:

Start frequency: Hz

Stop frequency: Hz

☒ Add full path for .include library directives

☒ Save all voltages

☒ Save all currents

☒ Save all power dissipations

☒ Save all digital event data

Compatibility mode: PSpice and LTSpice

Cancel OK

Calculates the small-signal AC behavior of the circuit in response to a stimulus. Performs a decade sweep of stimulus frequency.

To run an AC analysis you must choose a number of points to measure per decade and the start and end frequencies for the decade sweep.

AC analyses have the following options, which are listed here with the corresponding ngspice parameter name:

- **Number of points per decade:** the number of points to calculate per decade (nd).
- **Start frequency:** the lower bound of the frequency range to analyze (fstart).
- **Stop frequency:** the upper bound of the frequency range to analyze (fstop).

The output is displayed as a Bode plot (output magnitude and phase vs. frequency).

Transient analysis (TRAN)

New Simulation Tab

Analysis type: TRAN — Transient Analysis

SPICE Command **Plot Setup**

Time step: seconds

Final time: seconds

Initial time: seconds (optional; default 0)

Max time step: seconds (optional; default $\min\{tstep, (tstop-tstart)/50\}$)

☐ Use initial conditions

☒ Add full path for .include library directives

☒ Save all voltages

☒ Save all currents

☒ Save all power dissipations

☒ Save all digital event data

Compatibility mode: PSpice and LTSpice

Cancel OK

Calculates the time-varying behavior of the circuit.

Transient analyses have the following options, which are listed here with the corresponding ngspice parameter name:

- **Time step:** a suggested time step ($tstep$).
- **Final time:** the time at which the simulation will end ($tstop$).
- **Initial time:** the time at which the simulation will start ($tstart$). If not specified, this defaults to 0.
- **Max time step:** the maximum time step ($tmax$). If not specified, this defaults to the suggested time step or the total simulation duration divided by 50, whichever is smaller.

- **Use initial conditions:** if enabled, the simulator will not calculate the quiescent operating point before starting the transient simulation. Instead, the simulation will use the initial conditions specified in a `.ic` directive and element IC parameters. This corresponds to ngspice's `uic` option.

The output is displayed as a plot.

Pole-zero analysis (PZ)

New Simulation Tab

Analysis type: PZ — Pole-Zero Analysis

SPICE Command Plot Setup

Transfer function: (output voltage) / (input voltage)

Input: Ref:

Output: Ref:

Find: Poles and Zeros

☒ Add full path for .include library directives

☒ Save all voltages

☒ Save all currents

☒ Save all power dissipations

☒ Save all digital event data

Compatibility mode: PSpice and LTSpice

Cancel OK

Calculates the poles and zeroes of the small-signal (AC) transfer function of the circuit.

Pole-zero analyses have the following options, which are listed here with the corresponding ngspice parameter name:

- **Transfer function:** selects between calculating the transfer function as output voltage divided by input voltage or output voltage divided by input current. These correspond to ngspice's `vol` and `cur` options, respectively.

- **Input:** the input node (node1) and the reference node for the input (node2).
- **Output:** the output node (node3) and the reference node for the output (node4).
- **Find:** selects between calculating poles and zeroes, only poles, or only zeroes. These correspond to ngspice's pz, pol, and zer options, respectively.

Operating point analyses do not have any plotted results. Results are printed in the output log window.

Noise analysis (NOISE)

New Simulation Tab

Analysis type: **NOISE — Noise Analysis**

SPICE Command **Plot Setup**

Measured node:

Reference node: (optional; default GND)

Noise source:

Number of points per decade:

Start frequency: Hz

Stop frequency: Hz

☒ Save contributions from all noise generators

☒ Add full path for .include library directives

☒ Save all voltages

☒ Save all currents

☒ Save all power dissipations

☒ Save all digital event data

Compatibility mode: **PSpice and LTSpice**

Calculates the noise generated by the devices in the circuit, both as output noise and input-referred noise. The noise spectrum (V/#Hz or A/#Hz) of the circuit can be plotted, and the total noise over the specified frequency range is reported. Optionally, the noise contributions of each device are reported individually.

Noise analyses have the following options, which are listed here with the corresponding ngspice parameter name:

- **Measured node:** the node at which output noise is measured (`output`).
- **Reference node:** the reference node for measuring the output noise (`ref`). The output noise is calculated as the noise at the measured node minus the noise at the reference node. If it is not specified, the default is the ground node.
- **Noise source:** a source that is considered the circuit's input for the purposes of calculating input-referred noise (`src`). The source must specify an AC magnitude, i.e. the source must have an `ac` parameter.
- **Number of points per decade:** the number of points to calculate per decade (`pts`).
- **Start frequency:** the lower bound of the frequency range to analyze (`fstart`).
- **Stop frequency:** the upper bound of the frequency range to analyze (`fstop`).
- **Save contributions from all noise generators:** if enabled, noise magnitudes of individual noise generators are saved and reported. If disabled, only the overall output and input-referred noise over the specified frequency range are reported.

The output and input-referred noise spectra can be plotted. Total noise values for the specified frequency range are printed in the output log window.

S-parameter analysis (SP)

New Simulation Tab

Analysis type: SP — S-Parameter Analysis

SPICE Command Plot Setup

Number of points per decade:

Start frequency: Hz

Stop frequency: Hz

☐ Compute noise current correlation matrix

☒ Add full path for .include library directives

☒ Save all voltages

☒ Save all currents

☒ Save all power dissipations

☒ Save all digital event data

Compatibility mode: PSpice and LTSpice

Cancel OK

Calculates the scattering parameters, admittance matrix, and impedance matrix for the circuit. Optionally calculates the noise current correlation matrix. Note that this analysis requires at least two voltage sources configured as RF ports as described in the ngspice manual.

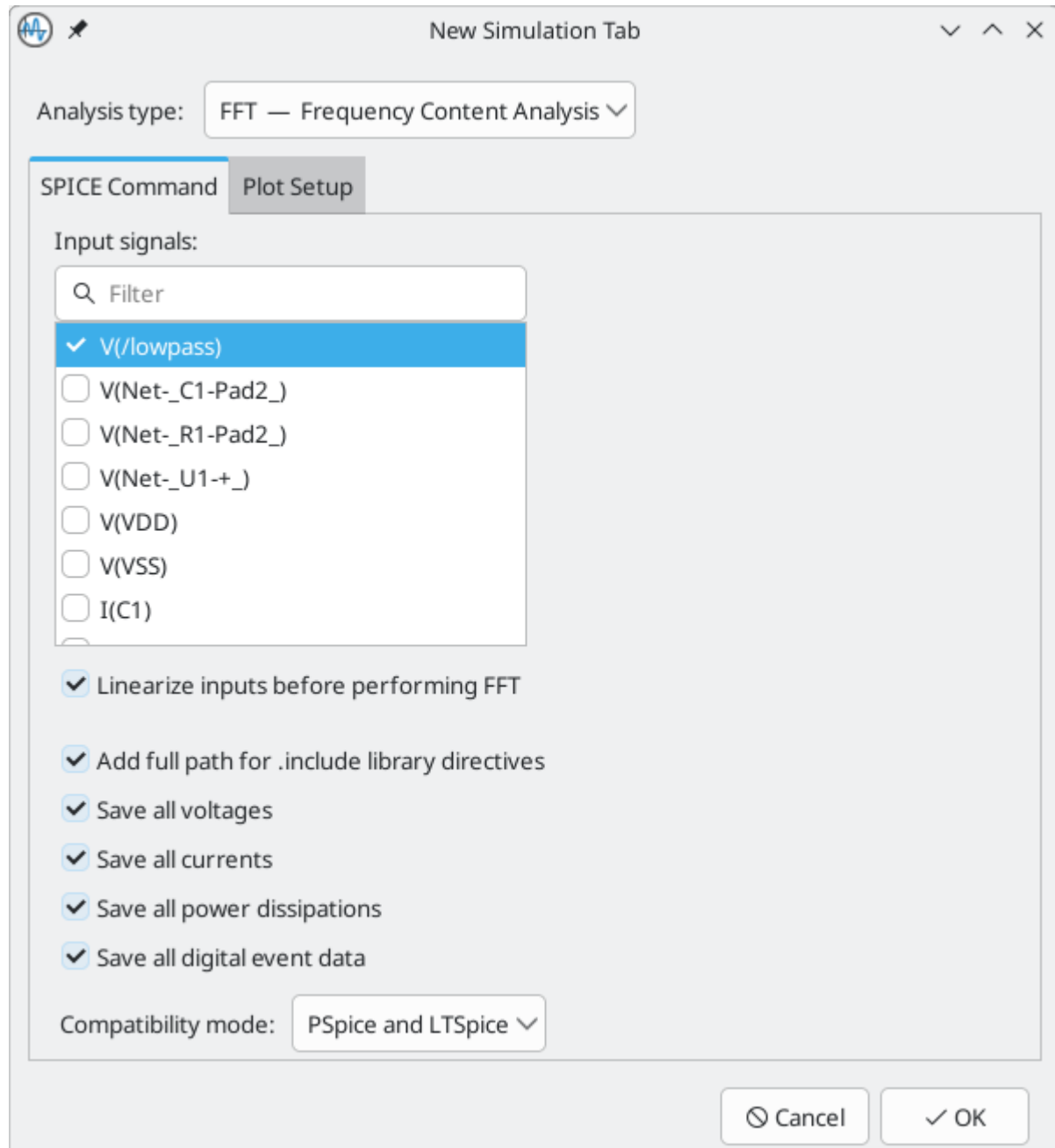
S-parameter analyses have the following options, which are listed here with the corresponding ngspice parameter name:

- **Number of points per decade:** the number of points to calculate per decade (nd).
- **Start frequency:** the lower bound of the frequency range to analyze (fstart).
- **Stop frequency:** the upper bound of the frequency range to analyze (fstop).

- **Compute noise current correlation matrix:** if enabled, the noise current correlation matrix is also calculated. This corresponds to ngspice's donoise option.

The output is displayed as a plot.

Frequency content analysis (FFT)



Calculates an FFT based on an existing analysis tab. Any of the signals from an existing analysis can be used as inputs to the FFT. The signals are windowed using a Hanning window.

FFT analyses have the following options, which are listed here with the corresponding ngspice parameter name:

- **Input signals:** the signal(s) to perform an FFT on. An FFT is independently applied to each selected signal.

- **Linearize inputs before performing FFT:** When enabled, the input vector is converted to have equidistant time points prior to performing the FFT. Corresponds to ngspice's `linearize` command.

The output is displayed as a plot.

Additional simulation settings

There are several simulation options that apply to all types of simulations. These are located at the bottom of the dialog.

- **Add full path for .include library directives:** if enabled, relative paths to SPICE models will be converted to absolute paths in the SPICE netlist.
- **Save all voltages:** if enabled, the simulator will save voltages for each node in the simulation results so that they can be plotted. This corresponds to ngspice's `.save all` command in the SPICE netlist. If disabled, voltages will not be saved in the results, and therefore cannot be plotted, unless a SPICE directive is used to manually probe a node voltage.
- **Save all currents:** if enabled, the simulator will save currents through each device pin in the simulation results so that they can be plotted. This corresponds to ngspice's `.probe all i` command in the SPICE netlist. If disabled, currents will not be saved in the results, and therefore cannot be plotted, unless a SPICE directive is used to manually probe a current.
- **Save all power dissipations:** if enabled, the simulator will save power dissipations for each device in the simulation results so that they can be plotted. This corresponds to ngspice's `.probe P(<device>)` command in the SPICE netlist for each device in the schematic. If disabled, power dissipations will not be saved in the results, and therefore cannot be plotted, unless a SPICE directive is used to manually probe a power dissipation.
- **Save all digital event data:** if enabled, the simulator will save digital event data for digital (event-driven) simulation models. This corresponds to ngspice's `.esave all` command. If disabled, digital event data will be discarded.
- The **Compatibility mode** dropdown selects the compatibility mode that the simulator uses to load models. The **User configuration** option refers to the user's `.spiceinit` ngspice configuration file. Compatibility modes are described in the ngspice documentation [<https://ngspice.sourceforge.io/docs.html>].

10.5. Viewing simulation results

Each analysis has its own tab, containing its own separate plot, signal list, and output log window. Only the active tab is updated when a simulation is run. In this way it is possible to compare simulation results between different runs.

Simulation results from most analysis types are visualized as plots. However, DC Operating Point (OP) and Pole-zero (PZ) analyses do not generate plots. Instead, they print their results in the output log at the bottom of the simulator window. OP analysis results can additionally be displayed as annotations on the schematic canvas.

10.5.1. Plotted results

Most types of analyses display their results in a plot. The type of plot depends on the analysis: transient simulations display signal values over time, for example, while AC simulations display results in a Bode plot.

You can zoom and move a plot using the following gestures:

- Scroll mouse wheel to zoom in/out. kbd:[Shift], kbd:[Ctrl], and kbd:[Alt] can modify the scroll action depending on the configuration in the Simulator panel in the Schematic Editor Preferences.
- Right click to open a context menu to adjust the view.
- Draw a selection rectangle to zoom in the selected area.
- Drag a cursor marker to move the cursor.

The list of signals that can be plotted in the active plot is shown in the Signals pane on the right side of the simulator window. The following types of signals can be plotted:

- Node voltages: the voltage of each net in the schematic, displayed as $V(<net>)$.
- Device node currents: the current for each device in the schematic, displayed as $I(<device>)$ or $I(<device:terminal>)$. For two-terminal devices, the device's current is listed as a single signal corresponding to the current into the device's pin 1. For devices with more than two terminals, the current into each terminal is a separate signal.
- Device power dissipations: the power dissipated by each component, displayed as $P(<device>)$.
- User-defined signals: custom signals defined as an expression based on other signals. User-defined signals can be arbitrary mathematical expressions. One common use for user-defined signals is to plot a voltage differential, i.e. the voltage between two points.

To plot a signal, check the box in the plot column next to the signal of interest. To remove a signal from the plot, clear its checkbox.

You can also interactively select signals to plot by using the Probe Schematic tool. To activate the tool, use **Simulation** → **Probe Schematic...** (kbd:[P]) or click the (



) button. When activated, the tool lets you click elements in the schematic to plot the corresponding signal. Different types of signals are probed depending on what you click:

- Clicking on a wire plots the voltage of that net. When you hover over a wire, the net is highlighted to indicate that its voltage can be probed.
- Clicking on a symbol pin plots the current going into that pin. When you hover over a pin, the cursor changes to a current clamp to indicate that clicking will probe the current.

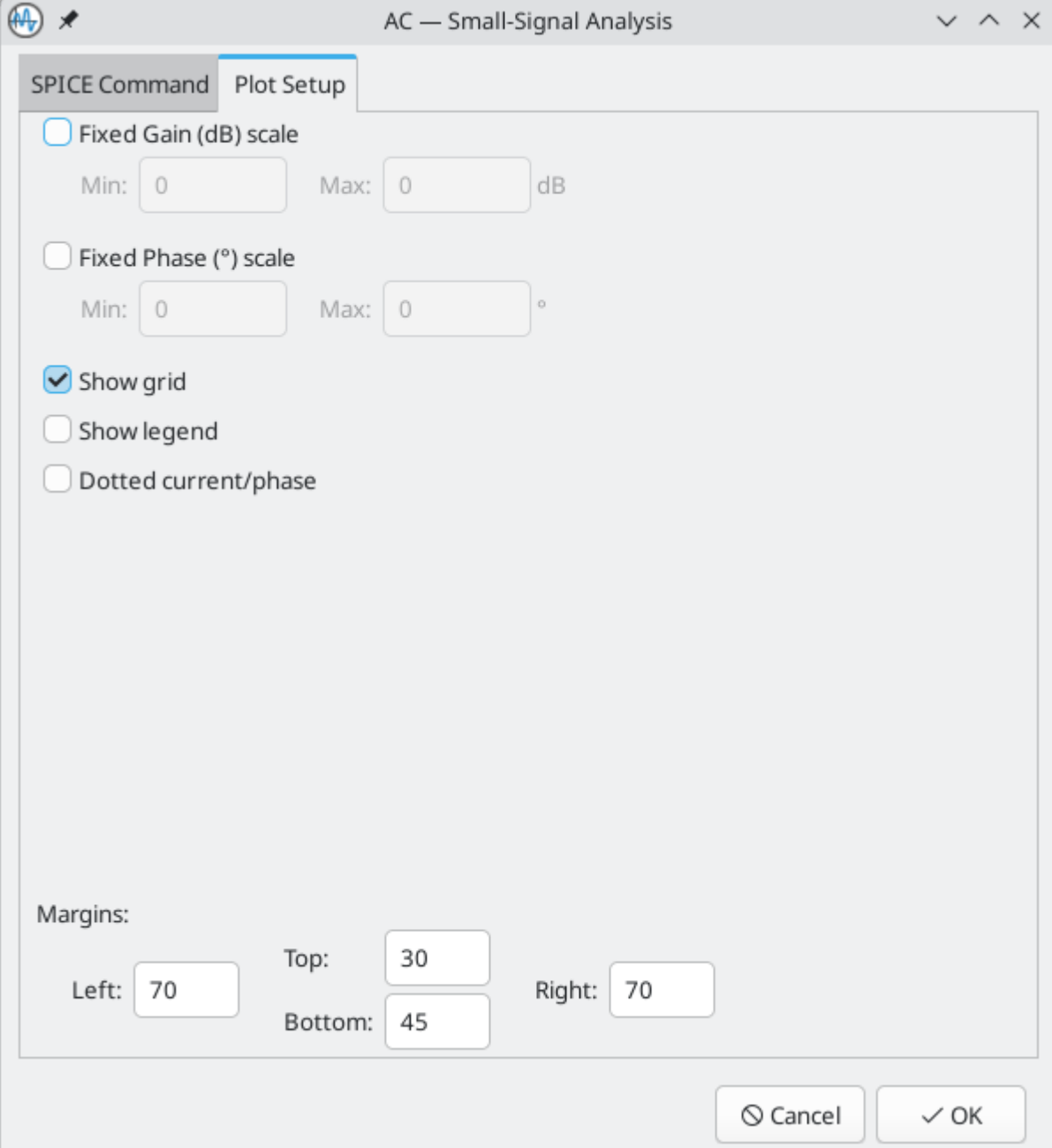
Uwaga

KiCad does not support ngspice's `.plot` directive. This directive has no effect when a simulation is run using KiCad.

Plot settings and appearance

Colors of individual signals may be set by clicking the color field associated with each signal in the Signals grid. You may choose from a predefined palette (**Defined Colors**), or select a custom color (**Color Picker**). You can toggle the color of the plot background from black to white with **View** → **Dark Mode Plots** (this affects all analysis tabs, not just the active tab).

Many plot settings can be configured in the **Plot Setup** tab of the Analysis Setup window ( button).



AC — Small-Signal Analysis

SPICE Command Plot Setup

☐ Fixed Gain (dB) scale
Min: 0 Max: 0 dB

☐ Fixed Phase (°) scale
Min: 0 Max: 0 °

☒ Show grid

☐ Show legend

☐ Dotted current/phase

Margins:
Left: 70 Top: 30 Right: 70
Bottom: 45

Cancel OK

- **Fixed scale:** when enabled, this fixes the vertical and/or horizontal plot scales to the specified ranges. Manually zooming will not affect an axis if its scale is fixed.
- **Show grid:** when enabled, a grid will be shown behind the plotted signals. You can also turn the grid on or off with **View** → **Show Grid**.

- **Show legend:** when enabled, a legend will be shown on the plot for the enabled signals. You can reposition the legend by dragging it.
- **Dotted current/phase:** when enabled, plotted signals representing current (transient simulations) or phase (AC simulations) are displayed using dotted lines instead of solid lines. You can also enable this setting with **View** → **Dotted Current/Phase**.
- **Margins:** this controls the padding on each side of the plot.

Cursors

For precise measurement, cursors are available in the plot window. You can add a cursor to a signal by checking the **Cursor 1** or **Cursor 2** checkbox for the signal in the signal grid on the right.

Once cursors have been added, the horizontal position of each cursor is shown by a number in a triangular marker at the top of the plot display. You can reposition each cursor by clicking and dragging its marker. The vertical position of each cursor tracks its assigned signal. The horizontal and vertical value for each cursor are shown in the cursor grid on the right of the simulator window. If cursors 1 and 2 are both enabled, the difference between them is also shown.

To precisely position a cursor, you can directly edit its horizontal position in the cursors grid. You can also modify the display format (unit range and number of significant digits) by right clicking a value in the cursors grid and clicking **Format** for the relevant quantity.

Measurements

You can add automatically calculated measurements for any plotted signal, such as a minimum, average, or peak-to-peak measurement.

Uwaga

Measurements are made over all the data resulting from the simulation, not just the data that is visible in the plot window at the current zoom setting.

Uwaga

It is not necessary to have selected a signal for plotting (by selecting its Plot checkbox) in order to make a measurement on it. Even unplotted signals can be measured.

To add a predefined measurement to a signal, right click on a signal in the signals grid and select a measurement. The following measurements are available:

- **Measure Min:** measures the minimum value of the entire signal
- **Measure Max:** measures the maximum value of the entire signal
- **Measure RMS:** measures the root-mean-square value of the entire signal
- **Measure Peak-to-peak:** measures the peak-to-peak value of the entire signal

- **Measure Time of Min:** measures the time at which the minimum value of the entire signal occurs
- **Measure Time of Max:** measures the time at which the maximum value of the entire signal occurs
- **Measure Integral:** computes the time integral value of the entire signal
- **Perform Fourier Analysis:** performs a Fourier analysis of the selected signal based on a specified fundamental frequency. Calculates the amplitude of the harmonics of the fundamental as well as the total harmonic distortion. This measurement is only available for transient analyses, and its results are printed in the simulation log window instead of in the measurement panel.

Measurement results are displayed in the Measurement pane at the lower right of the Simulator window. Multiple measurements will display as multiple rows in this area. You can delete a measurement by right clicking it and clicking **Delete Measurement**. To modify the display format of a measurement result (unit range and number of significant digits), right click the value and click **Format Value....**

The above context menu options are a shortcut for directly specifying measurements in the the Measurement pane. To manually create a new measurement, click in an empty row in the Measurement pane and type in an ngspice measurement function. You can also edit existing measurements by clicking on them. For more information about measurements and their syntax, refer to the ngspice manual.

10.5.2. Numerical results

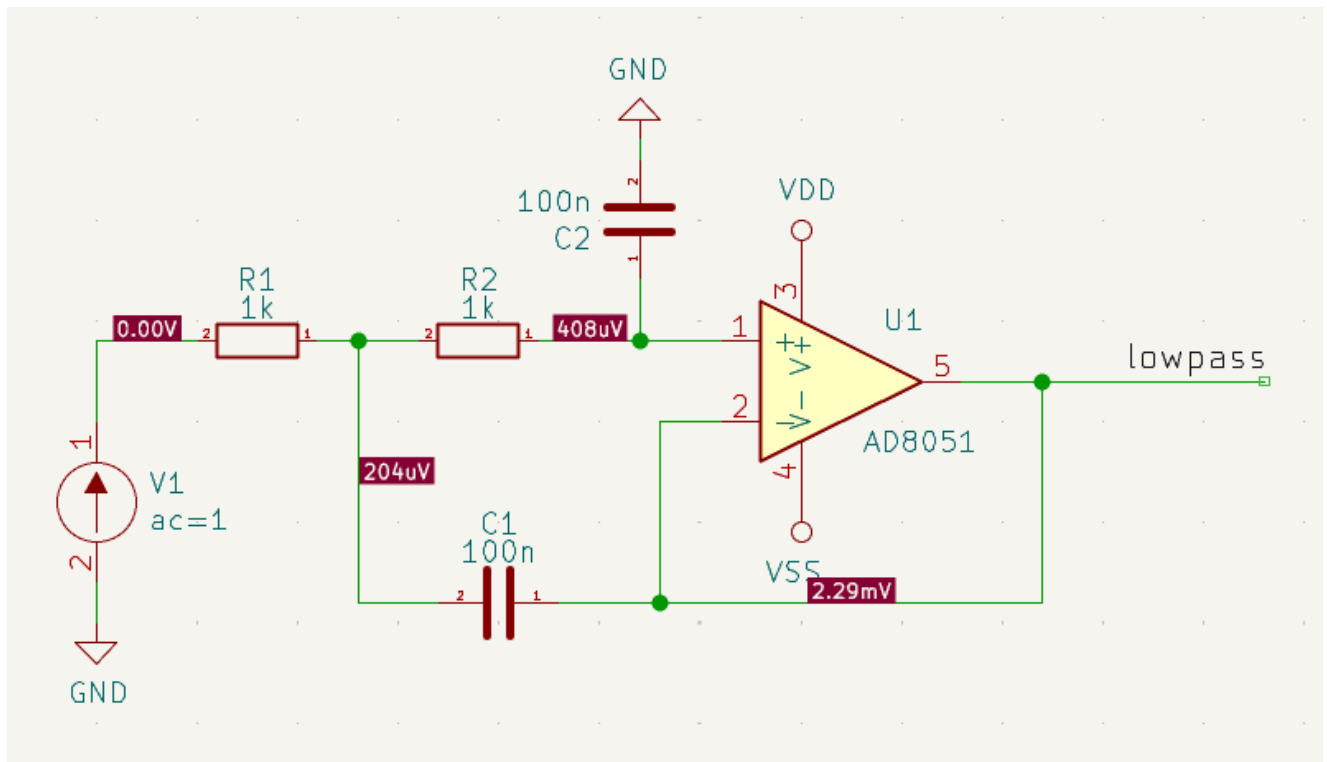
While most analyses display their results as plots, DC Operating Point (OP) and Pole-zero (PZ) analyses do not result in plots. Instead, these analyses print their results as text in the simulation log window.

```
Simulation results:

Background thread stopped with timeout = 0
pole(1) = -1.47250e+03,0.000000e+00
pole(2) = -1.88233e-01,0.000000e+00
pole(3) = 0.000000e+00,0.000000e+00
zero(1) = -1.00000e+04,0.000000e+00
zero(2) = -1.88233e-01,0.000000e+00
zero(3) = 0.000000e+00,0.000000e+00
```

Operating point annotations

For the OP analysis, annotations can also be added to the schematic indicating the operating point voltage and current values at the nodes. Operating point voltage annotations can be shown or hidden for every node with **View** → **Show OP Voltages**. Operating point current annotations can be shown or hidden for every symbol pin with **View** → **Show OP Currents**. These annotations are globally shown or hidden for every node or every pin. You can control the formatting of these annotations in the Formatting panel of Schematic Setup.



You can also display operating point results using text variables. Using text variables requires more work to set up but provides more control over how the data is displayed. For example, you can use text variables to display only certain voltages or currents, or to control the display formatting of particular values. You can also use text variables to display power dissipation measurements.

To use text variables to display an operating point voltage for a net, add a label to the net, then add a field to that label. The label can contain any text as long as it contains the `${OP}` text variable. The text variable can contain formatting specifiers in the form `${OP.<precision><unit>}`, but both the precision and unit are optional. Precision is the number of significant digits to display, which is 3 by default. Unit is the unit to use, including a prefix, such as mV for millivolts.

As an example, a label field containing the text `Vout: ${OP}`, attached to a net with an operating point voltage of 5.123V, displays as "Vout: 5.12V". The text `${OP}.4mV` displays as "5123mV", `${OP}.2` displays as "5.1V", and `${OP}.mV` displays as "512mV".

Using text variables to display an operating point current into a device pin is similar, but uses symbol fields instead of label fields. The field can contain any text as long as it contains the `${OP:<pin>}` text variable. The pin can be specified as a pin name or a pin number. For two-terminal devices, the pin can be omitted, and the current into the device's pin 1 will be reported. The same optional formatting specifiers are supported as for voltages, in the form `${OP:<pin>.<precision><unit>}`.

For example, in a symbol with an operating point current of 5.123mA through pin 1 (pin name C), a symbol field containing the text `Ic: ${OP:C}` displays as "Ic: 5.12mA". The text `${OP:C}.2mA` displays as "5.1mA", `${OP:1}.4` displays as "5.123mA", and `${OP:C.A}` displays as "0.00512A".

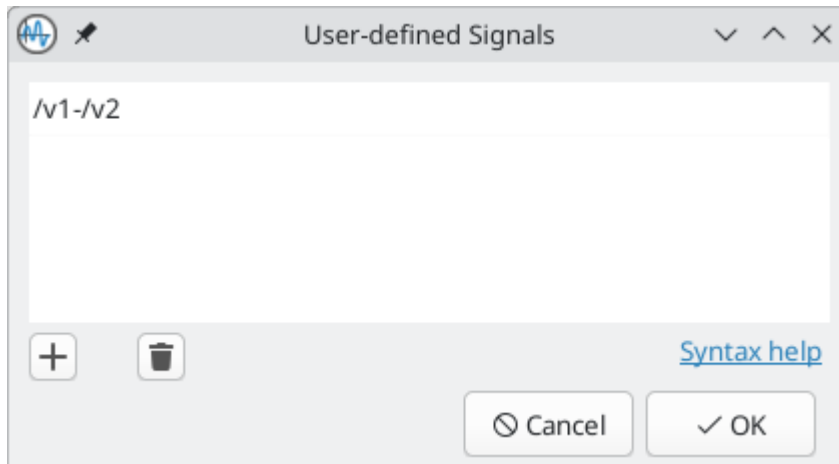
You can also use text variables to display a device's power dissipation. This works identically to current, but with power instead of a pin name or number. For example, in a symbol with an operating point power dissipation of 5.123mW, a symbol field containing `${OP:power}.2mW` displays as "5.1mW".

Uwaga

Don't forget to set the label field or symbol field to visible, or it will not be displayed.

10.5.3. User-defined signals

In addition to the list of signals that come from the nets in the schematic, you can define your own signals which behave like normal signals in most respects. User defined signals are defined as mathematical operations on one or more basic signals.



To add a user-defined signal, open the User-defined Signal dialog with **Simulation** → **User-defined Signals...** (



button) and add a signal. The new signal will appear in the Signal grid, where it can be plotted and used like any other signal. User-defined signals are also included in OP results.

Uwaga

Unlike normal signals, which are plotted incrementally as a simulation progresses, user-defined signals are not plotted until the simulation completes.

One use for user-defined signals is to plot a differential voltage, i.e. the voltage between two arbitrary nodes. For example, to plot the voltage difference between the nets /v1 and /v2, you can create a user-defined signal with the expression /v1 - /v2 and then plot it. This expression could also be written in a number of different ways, for example V(/v1) - V(/v2) or V(/v1, /v2), which are both equivalent.

A number of mathematical functions are available for use in user-defined signals. To see a list, click the **Syntax help** link in the User-defined Signals dialog. Refer to the ngspice manual for details on each of these functions.

Uwaga

Net names may contain special characters that have particular meaning to the ngspice interpreter. In particular, ngspice interprets the dash character (-) as a subtraction operation. To ensure that net names are interpreted correctly, surround the net name with double quote (") characters when referring to it in a user-defined signal.

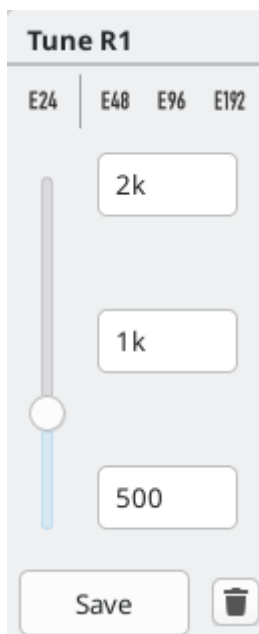
10.6. Tuning components

It is possible to adjust the value of basic components in the schematic from within the simulation interface, which lets you conveniently adjust component values based on simulation results. Each time the component value is tuned, the simulation is re-run with the new component value. You can tune the value of passive resistors, inductors, and capacitors, or the voltage or current of DC voltage and current sources.

Uwaga

You can only tune components when running analyses that result in a plot. In other words, you cannot tune components when running an operating point analysis.

To tune a component, use **Simulation** → **Add Tuned Value...**, the keyboard shortcut kbd:[T], or the  button in the toolbar, and then click on the component to tune in the schematic. This adds a tuning control for that component in the bottom right corner of the simulator window. Multiple components can be tuned at once, with a separate tuning control per component.



- The top text field sets the top end of the tuning range.
- The bottom text field sets the bottom end of the tuning range.
- The middle text field sets the actual component value that is used in the simulation. This value can also be adjusted using the slider.
- The **Save** button updates the schematic symbol with the tuned value. Until you press the **Save** button, the schematic symbol will keep its original value.
- The  button removes the component from the Tune panel and restores its original value.

In addition, it is possible to restrict the component values to those from a particular series of Preferred Values — either of the E24, E48, E96 or E192 series. This is particularly useful when it is necessary to restrict component values to commercially available parts.

10.7. Saving simulation setups

You can save a simulation setup in a workbook. Workbooks are files that store information about a simulation setup, including which analyses are configured and what their settings are, which signals are plotted for each analysis, user-defined signals, measurements, cursors, and display settings. A workbook can be saved and reloaded later to restore the previously configured settings in a new session. Workbooks can include more than one simulation: for example, it may contain separate tabs for transient, operating point, and AC analyses which you added as you worked.

You can save a workbook using **File** → **Save Workbook** and load one using **File** → **Open Workbook**. The most recently used workbook is automatically loaded when the simulator is opened.

Uwaga

Workbooks store simulation setup information, but they do not store simulation results. You can export simulation results to PNG (graphics) or CSV (simulation data values) with **File** → **Export Current Plot as PNG...** and **File** → **Export Current Plot as CSV....** To recreate the results of simulations stored in a workbook, select the appropriate analysis tab in the Simulator window and run the simulation again (kbd:[R] or **Simulation** → **Run Simulation**).

10.8. Exporting simulation results

KiCad's simulator offers two ways to export results:

- as a PNG (Portable Network Graphics format) image of a simulation data plot,
- as a plain text file of simulation data values, in CSV (Comma-Separated Value) format.

Only simulations that produce plots (see above) can be exported as an image or a CSV file. The results of OP and PZ analyses cannot be exported in this way.

10.8.1. Exporting results as graphics

The currently-visible Simulator plot may be exported as a PNG file using the command **File** → **Export Current Plot as PNG...**

The size and aspect-ratio of the saved image will match that of the displayed plot in the Simulator.

You can also copy an image of the active plot to the clipboard using **File** → **Export Current Plot to Clipboard**, or insert an image of the active plot into the schematic using **File** → **Export Current Plot to Schematic**. Exporting a plot to the schematic is equivalent to exporting the plot to the clipboard and then pasting it into the schematic.

10.8.2. Exporting results as numerical data

The currently-visible Simulator plot may be exported as a CSV file using the command **File** → **Export Current Plot as CSV...**

The data in the simulation plot is exported as multiple columns. The precise format is dependent upon the analysis type. In general, there will be multiple columns of data, one corresponding to each variable selected for plotting. The first row of the file is a header row, containing the name of the variable in the column (i.e. *time*, *V(Vout)1* or similar).

Uwaga

When exporting to CSV, only variables selected using their Plot checkbox will be included in the exported file.

Uwaga

The data in the exported file contains all the data that would be plotted if the entire plot were displayed. If the plot is zoomed in to show a particular region this will still be the case, resulting in the output file containing more data than the user might expect.

Uwaga

This function exports the data with data separated by semicolons (;) rather than commas. Programs reading this data may need to be configured to expect a semicolon as a delimiter.

10.9. Troubleshooting simulations

Sometimes a simulation will fail, either with or without errors being reported. Paying attention to the error messages reported, taking care in the development and entry into KiCad of the circuit to be simulated, and making use of the KiCad, ngspice and general SPICE documentation and information from fellow users in forums is very worthwhile and can often point the way to a solution.

It's worth noting, for users unfamiliar with SPICE, ngspice or circuit simulation in general that some *common* and interesting circuits can sometimes be tricky to simulate accurately or reliably. These include apparently simple circuits such as oscillators, which in some cases may fail to oscillate at all! It is nearly always possible to build a working simulation but sometimes this can more require SPICE experience than might be initially apparent, or the guidance of someone who already has it. The ngspice documentation, once again, is worth reading for insights into good and effective simulation practices if you are encountering difficulty.

10.9.1. Incorrect netlist

It is possible to inspect the SPICE netlist with **Simulation** → **Show SPICE netlist...** (



button). This method of troubleshooting requires some SPICE knowledge, but spotting errors in the netlist can help determine the cause of simulation problems, as well as providing confirmation of what input ngspice is actually acting on.

10.9.2. Simulation error messages

The output console displays messages from the simulator. It is advisable to check the console output to verify there are no errors or warnings. Messages appearing in the console may be conveniently selected, copied, and pasted if you wish to share them.

A common error message is "timestep too small". This message means that the simulation engine is unable to calculate the next point in the simulation, even when using the minimum possible time increment. This error can have many causes, including numerical convergence issues with a simulation model used in the circuit or with the circuit itself. It can also be caused by mistakes in drawing the circuit, such as incorrectly assigning pins to the simulation model or forgetting to provide a voltage supply.

10.9.3. Convergence problems

In case the simulation does not converge in a reasonable amount of time (or not at all), it is possible to add the following SPICE directives. More information is available in the ngspice manual.

Ostrze#enie

Changing convergence options can lead to erroneous results. These options should be used with caution.

```
.options gmin=1e-10
.options abstol=1e-10
.options reltol=0.003
.options cshunt=1e-15
```

- `gmin` is the minimum conductance allowed by the program. The default value is $1e-12$ (1 pS).
- `abstol` is the absolute current error tolerance of the program. The default value is $1e-12$ (1 pA).
- `reltol` is the relative error tolerance of the program. The default value is 0.001 (0.1%).
- `cshunt` adds a capacitor of the specified value from each voltage node in the circuit to ground.

10.9.4. Simulation pin assignments

If unexpected results are generated by a simulation despite it running without obvious errors, it is worth double-checking that the assignments between the pins of the part instance in the KiCad schematic and that of the associated model are correct. These have to be correct for each instance of the model in the schematic.

10.10. Helpful hints

Some helpful tips, hints and advice to help you get the most from using ngspice in KiCad for simulation.

10.10.1. The ngspice manual is your friend!

Fundamentally the KiCad Simulator is a user-friendly front-end to the powerful ngspice circuit simulator. Therefore problems with the details of simulation, the correct use of SPICE elements, models, etc. is beyond the scope of the KiCad documentation but is very likely to be fully and completely addressed in the ngspice documentation itself. It's therefore recommended not to overlook this valuable resource [<https://ngspice.sourceforge.io/docs.html>].

Uwaga

KiCad updates may result in changes to the ngspice version used by the simulator. The current version of ngspice used in a particular version of KiCad is shown on the **Help** → **About KiCad** dialog, under the **Version** tab. Referring to the online ngspice documentation will ensure you always have access to the latest information for reference.

10.10.2. Organizing third-party models

Although it is certainly possible to keep simulation models (i.e. .lib, .sub files) in the directories associated with individual KiCad projects, this will likely result in unnecessary copies of model files proliferating as you create simulations. Consider creating a dedicated storage location (directory, folder) for models, perhaps organized by manufacturer or device type, to hold these files. Then they may simply be referenced in simulations at a common location.

10.10.3. Simulation symbols and models in KiCad's libraries

The KiCad libraries provide a number of symbols for simulation in the `Simulation_SPICE` symbol library. Most of these symbols have SPICE models already assigned as appropriate, although you may need to adjust the model parameters in order to obtain the desired simulation behavior.

The `Simulation_SPICE` symbol library contains the following symbols:

Symbol Name	Description
0	A ground (node 0) symbol. Note that a standard GND symbol from another library can also be used.
BSOURCE	A symbol for a SPICE B-source (nonlinear dependent voltage or current source). Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Non_linear_Dependent_Sources] for information on B-sources.
D	A diode symbol. Note that the pin assignment is appropriate for both simulation and PCB layout.
ESOURCE	A symbol for a SPICE E-source (linear voltage-controlled voltage source). By default it is set up with a gain of 1 V/V. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#subsec_Exxxx_Linear_Voltage_Controlled] for more information on E-sources.
GSOURCE	A symbol for a SPICE G-source (linear voltage-controlled current source). By default it is set up with a gain of 1 A/V. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#subsec_Gxxxx_Linear_Voltage_Controlled] for more information on G-sources.
IAM	A symbol for a SPICE amplitude-modulated independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/

	manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
IBIS_DEVICE	An IBIS device symbol representing a digital input pin. This symbol is not preconfigured with a simulation model. It is intended to be used with an external IBIS model for a device.
IBIS_DEVICE_DIFF	An IBIS device symbol representing a digital differential input pin. This symbol is not preconfigured with a simulation model. It is intended to be used with an external IBIS model for a differential device.
IBIS_DRIVER	An IBIS driver symbol representing a digital output pin. This symbol is not preconfigured with a simulation model. It is intended to be used with an external IBIS model for a driver.
IBIS_DRIVER_DIFF	An IBIS driver symbol representing a pair of digital differential output pins. This symbol is not preconfigured with a simulation model. It is intended to be used with an external IBIS model for a differential driver.
IDC	A symbol for a SPICE DC independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
IEXP	A symbol for a SPICE exponential independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
IPULSE	A symbol for a SPICE pulsed independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
IPWL	A symbol for a SPICE piecewise linear independent current source. The waveform is specified as space-separated time-current pairs. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
ISFFM	A symbol for a SPICE single-frequency frequency-modulated independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
ISIN	A symbol for a SPICE sinusoidal independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
ITRNOISE	A symbol for a SPICE transient noise independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/

	ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
ITRRANDOM	A symbol for a SPICE transient random independent current source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent current sources.
NJFET	A symbol for an N-channel JFET with drain, gate, and source terminals, suitable for use with 3-terminal N-JFET models.
NMOS	A symbol for an N-channel MOSFET with drain, gate, and source terminals, suitable for use with 3-terminal N-MOSFET models.
NMOS_Substrate	A symbol for an N-channel MOSFET with drain, gate, source, and substrate (bulk) terminals, suitable for use with 4-terminal N-MOSFET models.
NPN	A symbol for an NPN transistor with collector, base, and emitter terminals, suitable for use with 3-terminal NPN models.
NPN_Substrate	A symbol for an NPN transistor with collector, base, emitter, and substrate terminals, suitable for use with 4-terminal NPN models.
OPAMP	A generic single-pole operational amplifier symbol. There are parameters for pole frequency, open loop gain, offset voltage, and output resistance. These parameters can be edited in the Parameters grid of the SPICE Model Editor dialog.
PJFET	A symbol for a P-channel JFET with drain, gate, and source terminals, suitable for use with 3-terminal P-JFET models.
PMOS	A symbol for a P-channel MOSFET with drain, gate, and source terminals, suitable for use with 3-terminal P-MOSFET models.
PMOS_Substrate	A symbol for a P-channel MOSFET with drain, gate, source, and substrate (bulk) terminals, suitable for use with 4-terminal P-MOSFET models.
PNP	A symbol for a PNP transistor with collector, base, and emitter terminals, suitable for use with 3-terminal PNP models.
PNP_Substrate	A symbol for a PNP transistor with collector, base, emitter, and substrate terminals, suitable for use with 4-terminal PNP models.
SWITCH	A symbol for a voltage-controlled switch. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#subsec_Switches] for more information on switches.
TLINE	A symbol for a transmission line. By default it is set up as a lossless transmission line but it can also be used for a lossy transmission line. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Lossless_Transmission_Lines] for more information on transmission lines.
VAM	A symbol for a SPICE amplitude-modulated independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/

	manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VDC	A symbol for a SPICE DC independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VEXP	A symbol for a SPICE exponential independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VOLTMETER_DIFF	A differential voltmeter symbol that can be used to measure the voltage between two arbitrary nodes. The voltage difference between the + and - pins is output as a ground-referenced voltage on the out pin.
VPULSE	A symbol for a SPICE pulsed independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VPWL	A symbol for a SPICE piecewise linear independent voltage source. The waveform is specified as space-separated time-voltage pairs. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VSFFM	A symbol for a SPICE single-frequency frequency-modulated independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VSIN	A symbol for a SPICE sinusoidal independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VTRNOISE	A symbol for a SPICE transient noise independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.
VTRRANDOM	A symbol for a SPICE transient random independent voltage source. Refer to the ngspice manual [https://ngspice.sourceforge.io/docs/ngspice-html-manual/manual.xhtml#sec_Independent_Sources_for] for more information on independent voltage sources.

Most of these symbols in the `Simulation_SPICE` library use built-in models, but several symbols use external models from the `Simulation_SPICE.sp` simulation model library. The simulation model library is a separate file in the same folder as the symbol library. This simulation model library also contains SPICE models that may be useful for use with other symbols.

The `Simulation_SPICE.sp` simulation model library contains the following models:

Model Name	Description
kicad_builtin_opamp	A generic single-pole operational amplifier model. This model is used by the OPAMP symbol in the Simulation_SPICE symbol library, and the pins are ordered appropriately for use with any standard single-unit opamp symbol. There are parameters for pole frequency, open loop gain, offset voltage, and output resistance. These parameters can be edited in the Parameters grid of the SPICE Model Editor dialog.
kicad_builtin_opamp_dual	A dual version of the kicad_builtin_opamp model, with the same parameters. This model is not assigned to any symbols in the Simulation_SPICE symbol library, but the pins are ordered appropriately for use with standard dual opamp symbols.
kicad_builtin_opamp_quad	A quad version of the kicad_builtin_opamp model, with the same parameters. This model is not assigned to any symbols in the Simulation_SPICE symbol library, but the pins are ordered appropriately for use with standard quad opamp symbols.
kicad_builtin_varistor	A generic varistor model. This model is not assigned to any symbols in the Simulation_SPICE symbol library. There are parameters for threshold voltage, dynamic resistance, and leakage resistance. These parameters can be edited in the Parameters grid of the SPICE Model Editor dialog.
kicad_builtin_vdiff	A differential voltmeter model. The voltage difference between the first two terminals is output as a ground-referenced voltage on the third terminal. This model is used by the VOLTMETER_DIFF symbol in the Simulation_SPICE symbol library.

Rozdział# 11. Advanced Topics

11.1. Configuration and Customization

Uwaga

This section of the KiCad documentation has not yet been written. We appreciate your patience as our small team of volunteer documentation writers work to update and expand the documentation.

11.2. Text variables

KiCad supports text variables, which allow you to substitute the variable name with a defined text string. This substitution happens anywhere the variable name is used inside the variable replacement syntax of `${VARIABLENAME}`.

You can define project text variables in the schematic or board setup dialogs. Project text variables are defined for the whole project, so a project text variable defined in the Schematic Editor can also be used in the Board Editor.

There are also a number of built-in system text variables. System text variables may be available in some contexts and not others. The following system text variables can be used in schematic text, label names, label fields, hierarchical sheet fields, symbol text, symbol fields, and drawing sheet fields. There are also a number of variables that can be used in the PCB Editor.

Variables used in hierarchical sheet fields refer to the properties of the hierarchical sheet, not the parent, unless otherwise noted. For example, `${#}` returns the subsheet's page number when used in a hierarchical sheet field, but the parent sheet's page number when used in graphic text in the parent sheet.

Variables can also be used for field names. A field with a variable as its name will automatically have its value set to the same variable. For example, in a project with a project variable `MY_VAR` set to `MY_VALUE`, a user-created symbol field named `${MY_VAR}` will automatically have its value set to `${MY_VAR}`, which will then resolve to `MY_VALUE`. If the field's **Show Name** property is set, the variable's name will be displayed as the field name, for example `MY_VAR: MY_VALUE`.

Variable name	Description
#	Sheet number.
##	Total number of schematic sheets.
COMMENT1 - COMMENT9	Contents of drawing sheet's Comment<n> field.
COMPANY	Contents of drawing sheet's Company field.
CURRENT_DATE	Today's date, in ISO format.
FILENAME	Filename of the root schematic sheet , with a file extension.
FILEPATH	Full file path of the root schematic sheet , with a file extension.
ISSUE_DATE	Contents of drawing sheet's Issue Date field.

Variable name	Description
KICAD_VERSION	Current version of KiCad. This variable is only available in drawing sheet fields.
PAPER	Current sheet's paper size. This variable is only available in drawing sheet fields.
PROJECTNAME	Project name, without a file extension.
REVISION	Contents of drawing sheet's Revision field.
SHEETFILE	Filename of the current sheet , with a file extension.
SHEETNAME	Sheet name of the current sheet.
SHEETPATH	Sheet path of the current sheet.
TITLE	Contents of drawing sheet's Title field.
<variablename>	Contents of project text variable <variablename>.
<fieldname>	Contents of symbol field, symbol attribute, hierarchical sheet field, or label field <fieldname>. Fields can only be accessed from within their parent object, so symbol fields can be accessed from other text or fields within the symbol, and hierarchical sheet fields can be accessed within the sheet or in other sheet fields of the sheet. Both built-in and user-defined fields are available. Built-in fields use all uppercase letters: for example, to access a symbol's value, use \${VALUE}. Built-in symbol fields are DATASHEET, DESCRIPTION, FOOTPRINT, FOOTPRINT_LIBRARY, FOOTPRINT_NAME, NET_CLASS(<pin_number>), NET_NAME(<pin_number>), OP, PIN_NAME(<pin_number>), REFERENCE, SHORT_NET_NAME(<pin_number>), SHORT_REFERENCE, SYMBOL_DESCRIPTION, SYMBOL_KEYWORDS, SYMBOL_LIBRARY, SYMBOL_NAME, UNIT, VALUE. Built-in symbol attributes are DNP, EXCLUDE_FROM_BOARD, EXCLUDE_FROM_BOM, and EXCLUDE_FROM_SIM. These attributes expand to the friendly name of the attribute if the attribute is set (e.g. Excluded from board for EXCLUDE_FROM_BOARD and DNP for DNP), or to an empty string if the attribute is not set. Built-in sheet fields are SHEETFILE, SHEETNAME, and SHEETPATH. These refer to the child sheet's filename, sheet name, and sheet path, respectively, rather than the parent sheet's. Built-in label fields are CONNECTION_TYPE, NET_CLASS, NET_NAME, OP, SHORT_NET_NAME, and INTERSHEETREFS (global labels only).
<refdes>:<fieldname>	Contents of field or attribute <fieldname> in symbol <refdes>. Both built-in and user-defined fields are available. Built-in fields use all uppercase letters: for example, to access the value of U1, use \${U1:VALUE}. Built-in symbol fields are DATASHEET, DESCRIPTION, FOOTPRINT, FOOTPRINT_LIBRARY, FOOTPRINT_NAME, NET_CLASS(<pin_number>), NET_NAME(<pin_number>), OP, PIN_NAME(<pin_number>), REFERENCE,

Variable name	Description
	SHORT_NET_NAME(<pin_number>), SYMBOL_DESCRIPTION, SYMBOL_KEYWORDS, SYMBOL_LIBRARY, SYMBOL_NAME, UNIT, VALUE. Built-in symbol attributes are DNP, EXCLUDE_FROM_BOARD, EXCLUDE_FROM_BOM, and EXCLUDE_FROM_SIM. These attributes expand to the friendly name of the attribute if the attribute is set (e.g. Excluded from board for EXCLUDE_FROM_BOARD and DNP for DNP), or to an empty string if the attribute is not set.
ERC_ERROR <errorname>	Generates an ERC error named <errorname>. Everything inside the braces resolves to an empty string, while everything after the braces is included in the descriptive text for the ERC violation. The text variable must be at the beginning of the text item. For example, a text item containing <code>\${ERC_ERROR TODO}Calculate resistor value</code> would display as the text "Calculate resistor value" and generate a ERC error named "TODO" with the description "Calculate resistor value".
ERC_WARNING <warningname>	Generates an ERC warning named <warningname>. This behaves the same as ERC_ERROR, except a warning is generated rather than an error.

11.3. Database Libraries

A database library is a type of KiCad symbol library that holds data about parts in an external SQL database. Database libraries do not contain any symbol or footprint definitions by themselves. Instead, they **reference** symbols and footprints found in other KiCad libraries. Each database library entry maps a KiCad symbol (from another library) to a set of properties (fields) and usually a KiCad footprint (from a footprint library).

Using database libraries allows you to create fully-defined parts (sometimes called **atomic parts**) out of KiCad symbols and footprints without needing to store all the part properties in a symbol library. The external database can be linked to third-party tools for managing part data and lifecycles. Database library workflows are generally more complex than the standard KiCad library workflows, and so this type of library is typically only used in situations where it makes managing a large library of fully-defined parts more efficient (such as in organization or team settings).

KiCad does not provide a GUI for editing a SQL database or defining a database library. It is up to the user to find the most appropriate workflow and toolchain for creating and updating the database itself. Some users may want to directly edit the database through a third-party database client, and some may use other third-party software such as a part lifecycle management (PLM) tool to create and edit data.

In a database library, there are one or more **tables** that generally represent a single type of part (such as Resistors or Capacitors). Each table can have an independent schema, meaning that different types of parts can have different properties that are translated into symbol fields in KiCad. Each table must have a unique ID column which is used as the identifier for a symbol placed from that table. This unique ID will typically be a part number (either a manufacturer's part number, or an internal organization part number). Each table must also have a column that contains a mapping to a KiCad symbol, in the form `LibraryNickname:SymbolName`. The `LibraryNickname` must match a symbol library that is present in the KiCad library tables. Tables may also contain a column containing a

KiCad footprint, in the form `LibraryNickname:FootprintName`. If this column is present, symbols placed from the table will include a footprint mapping.

Tables may also contain arbitrary additional columns that may optionally be mapped to symbol fields in KiCad. The KiCad database library configuration file controls how these fields should be named, whether or not to make the fields visible, and whether or not to include the fields in the data displayed in the Symbol Chooser.

11.3.1. Database Library Configuration Files

To create a database library, you must create a configuration file that contains the necessary information for KiCad to connect to your database and retrieve data from tables. Copy the template below into a new file and save it with a `kicad_dbl` extension. You can then add this file to your global symbol library table using the Configure Symbol Libraries dialog.

```
{
  "meta": {
    "version": 0
  },
  "name": "My Database Library",
  "description": "A database of components",
  "source": {
    "type": "odbc",
    "dsn": "",
    "username": "",
    "password": "",
    "timeout_seconds": 2,
    "connection_string": ""
  },
  "libraries": [
    {
      "name": "Resistors",
      "table": "Resistors",
      "key": "Part ID",
      "symbols": "Symbols",
      "footprints": "Footprints",
      "fields": [
        {
          "column": "MPN",
          "name": "MPN",
          "visible_on_add": false,
          "visible_in_chooser": true,
          "show_name": true,
          "inherit_properties": true
        },
        {
          "column": "Value",
          "name": "Value",
          "visible_on_add": true,
          "visible_in_chooser": true,
          "show_name": false
        }
      ]
    },
    {
      "name": "Capacitors",
      "table": "Capacitors",
      "key": "Part ID",
      "symbols": "Symbols",
      "footprints": "Footprints",
      "fields": [
        {
          "column": "MPN",
          "name": "MPN",
          "visible_on_add": false,
          "visible_in_chooser": true,
          "show_name": true,
          "inherit_properties": true
        },
        {
          "column": "Value",
          "name": "Value",
          "visible_on_add": true,
          "visible_in_chooser": true,
          "show_name": false
        }
      ]
    }
  ],
  "properties": {
    "description": "Description",
    "footprint_filters": "Footprint Filters",
    "keywords": "Keywords",
    "manufacturer": "Manufacturer",
    "part": "Part",
    "value": "Value"
  }
}
```

```
        "exclude_from_bom": "No BOM",  
        "exclude_from_board": "Schematic Only"  
    }  
}  
]  
}
```

Uwaga

Database library files are in JSON format. Standard JSON syntax rules apply. To check if your file contains syntax errors, you may use a JSON validator or linter (available online).

Configuring the source

KiCad currently only supports ODBC connections to SQL databases. You can either connect with a DSN or a connection string. If a DSN name is supplied, the optional username and password fields will be used to connect to the DSN. If a connection string is supplied, the `dsn`, `username`, and `password` fields are ignored. The connection string will be passed directly to the ODBC driver, so you can include any parameters your ODBC driver supports.

When using a DSN connection, leave the `connection_string` property blank or omit it from the file. When using a connection string, leave the `dsn`, `username`, and `password` fields blank or omit them from the file. Connection strings must start with a `Driver` key indicating to the ODBC manager which driver should be used, and may include other keys that depend on the specific driver. Check the documentation for your ODBC driver for details. You may also find a reference site like [connectionstrings.com](https://www.connectionstrings.com/) [https://www.connectionstrings.com/] useful when configuring a database connection.

KiCad does not recommend or endorse any particular ODBC driver or database server, but has been tested to work with SQLite, MySQL, MariaDB, and PostgreSQL.

Uwaga

Windows users: the backslash character (`\\`) must be escaped with a second backslash when included in a JSON quoted string. If including a file path in your connection string, make sure to use double backslashes (`\\`).

Uwaga

Flatpak users: You need to install the corresponding ODBC drivers as Flatpak extensions. You can do this via the "Add-ons" section for KiCad in your software manager (i.e. GNOME Software), or via the command line: Run `flatpak install org.kicad.KiCad.ODBCDriver.sqliteodbc` for SQLite, `flatpak install org.kicad.KiCad.ODBCDriver.mariadb-connector-odbc` for MariaDB or MySQL, or `flatpak install org.kicad.KiCad.ODBCDriver.psycopg2` for PostgreSQL.

Uwaga

Flatpak users: Due to Flatpak sandboxing, a possible way to connect to database servers running on your local machine is via TCP/IP. Make sure that your database server

allows TCP/IP connections, then add the required `Port` parameter to your connection string. For example, add `Port=3306`; for the default TCP port of MySQL/MariaDB, or `Server=localhost;Port=5432`; to force PostgreSQL to use a TCP connection to the local server. Using the default UNIX domain socket connections for MySQL, MariaDB, or PostgreSQL is only possible when overriding host file system permissions via `flatpak override`.

Configuring libraries

Each database library can contain "sub-libraries" mapped to a single database table. The `libraries` entry in the configuration file contains a list of objects that each define a single library. The following settings must exist for each library:

name: The name of the sub-library (table) that will be shown in the KiCad UI and included as a prefix in each symbol name placed from this sub-library. This name can include any valid characters for a symbol name except for a forward slash (/) because the slash character is used as a separator between the sub-library name and the symbol name. If this field is left blank, no prefix will be added to symbols in this sub-library.

table: The name of the table in the database.

key: The column name containing a unique key that will be used to identify parts from the table.

symbols: The column name containing KiCad symbol references.

footprints: The column name containing KiCad footprint references.

fields: A list of field definitions. Each field defined here will be added to the symbol when it is placed on the schematic. If a field with a matching name is already defined in the source symbol, the value from the database table will override whatever value was defined in the source symbol. Each field definition may contain:

column: The name of the database table column that should be mapped to a field.

name: The name of the KiCad field to populate from the database.

visible_on_add: If `true`, this field will be visible in the schematic when a symbol is added. If this setting is not specified, it will default to `false`.

visible_in_chooser: If `true`, this field will be shown in the Symbol Chooser as a column. If this setting is not specified, it will default to `false`.

show_name: If `true`, the field's name will be shown in addition to its value in the schematic. If this setting is not specified, it will default to `false`.

inherit_properties: If `true`, and a field with the given name already exists on the source symbol, only the field contents will be updated from the database, and the other properties (`visible_on_add`, `show_name`, etc) will be kept as they were set in the source symbol. If the given field name does not exist in the source symbol, this setting is ignored. If this setting is not specified, it will default to `false`.

properties: A map of symbol properties to database columns. All properties are optional; any that are not specified in the database library configuration will be inherited from the values set for the source symbol. The following properties are supported:

description: The symbol's Description property.

footprint_filters: Reserved for future expansion.

keywords: The symbol's Keywords property.

exclude_from_bom: The symbol's "Exclude from Bill of Materials" setting. The column named here must be a numeric type, and will be taken as a boolean (0 for false, 1 for true).

exclude_from_board: The symbol's "Exclude from PCB" setting. The column named here must be a numeric type, and will be taken as a boolean (0 for false, 1 for true).

exclude_from_sim: The symbol's "Exclude from simulation" setting. The column named here must be a numeric type, and will be taken as a boolean (0 for false, 1 for true).

Database columns may be mapped to custom (user-defined) fields, or to certain built-in KiCad fields, including Value and Datasheet.

Uwaga

KiCad only supports text (string) fields. If you map a database column containing a numeric SQL data type, it will be converted to a string using a general-purpose conversion algorithm that will switch to scientific notation for very large or very small numbers. This format conversion cannot be fine-tuned by the user, so if explicit control over number-to-string conversion is needed, a new column or view should be used to do the conversion in the database.

11.3.2. Using database libraries

After creating your configuration file and adding it to your symbol library table, you can place parts from the database tables using the Symbol Chooser. Parts placed from a database library can be updated using the Update Symbols from Library function, which will update any fields that were changed in the database as well as updating the underlying symbol if it was changed in the source library.

Note that any source library referenced by a database table must also be present in the symbol library table for the database library to function. If you want to use a library only as a source of symbols for a database library, you can hide it from the Symbol Chooser by clearing the "Visible" checkbox in the Manage Symbol Libraries dialog.

11.4. HTTP Libraries

HTTP libraries are a type of KiCad symbol library that sources data about parts for an external source such as an ERP system. They do not contain any symbol or footprint definitions as standard KiCad libraries do. Instead, they **reference** symbols and footprints found in other KiCad libraries.

HTTP libraries are read only and support REST or REST-like APIs.

11.4.1. HTTP Library Configuration Files

To create an HTTP library, you must create a configuration file that contains the necessary information for KiCad to connect to the providing library (API) and to retrieve data from it.

Copy the template below into a new file and save it using the `.kicad_httplib` file extension. You should then edit this file and replace `root_url` and `token` values with your own. Once saved, add this file to your global symbol library table using the Configure Symbol Libraries dialog which can be found under **Preferences#Manage Symbol Libraries....**

Users have the option to configure two timeout settings. The `timeout_parts_seconds` setting dictates the validity duration of a part's information, while the `timeout_categories_seconds` setting determines how long categories remain valid. The default values are set to 60 seconds and 600 seconds respectively, but if the data for either setting is anticipated to remain unchanged, users can opt for higher values. This will significantly speed up the opening of the symbol chooser. It's important to note that KiCad will re-cache the data on the initial startup regardless of these timeout settings.

```
``` { "meta": { "version": 1.0 }, "name": "KiCad HTTP Library", "description": "A KiCad library sourced from a REST API", "source": { "type": "REST_API", "api_version": "v1", "root_url": "http://localhost:8000/kicad-api", "token": "usertokendatastring", "timeout_parts_seconds": 60, "timeout_categories_seconds": 600 } } ```
```

### 11.4.2. Authentication

Authentication is done via an **Access Token** only. Users need to ask their administrators to get a valid token issued if the HTTP library is maintained externally.

### 11.4.3. Caching Behaviour for Categories

KiCad caches all available Categories once when opening the Symbol Chooser Dialog. Subsequently, any alterations made to the categories on the server side will remain undetected by KiCad until the user performs a program restart. This implementation is intentionally designed to conserve bandwidth resources, as it prevents KiCad from attempting to retrieve data from the API every time the user opens the Symbol Chooser Dialog. Such continuous data fetching, especially under constrained bandwidth conditions, would severely impede KiCad's performance.

### 11.4.4. Server Response Codes

If KiCad receives an API error, it will display an error message to the user. For more information about API errors and server responses, see the APIs and Bindings section at [dev-docs.kicad.org](http://dev-docs.kicad.org).

## 11.5. Custom Netlist and BOM Formats

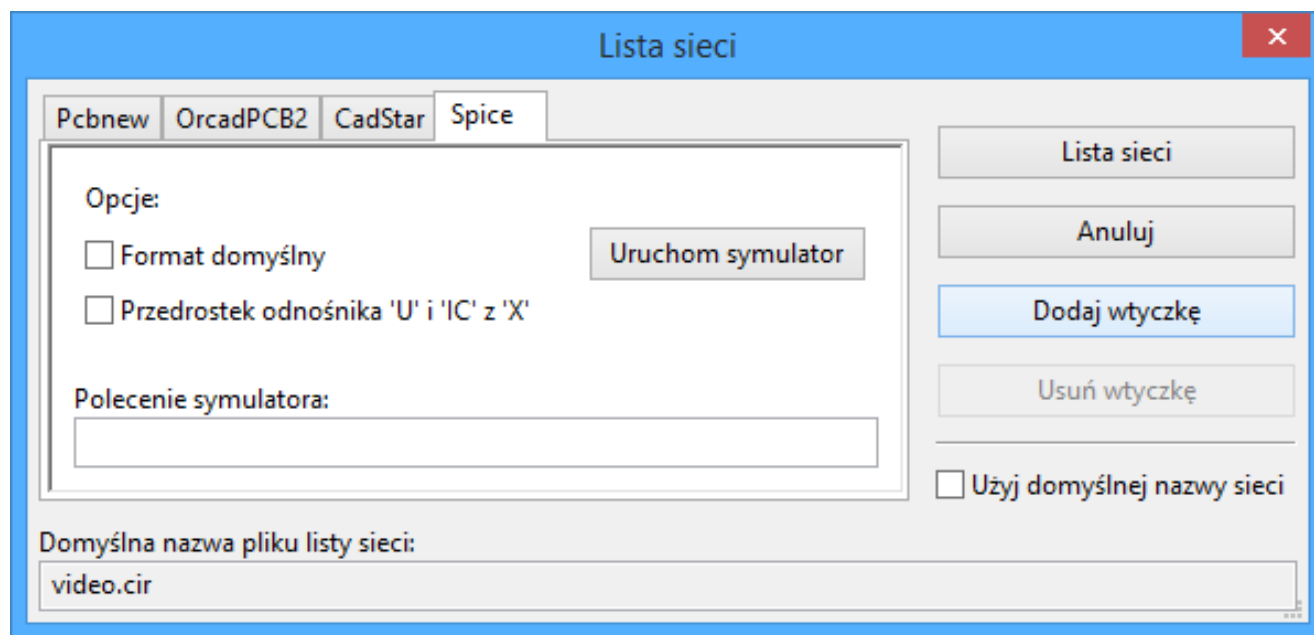
KiCad can output netlists and BOMs in various formats, and users can define new formats if desired.

The process of exporting a netlist is described in the netlist export section. BOM output is described in the BOM export section.

The following section describes how to create an exporter for a new output format.

## 11.5.1. Adding new netlist generators

New netlist generators are added to the **Export Netlist** dialog by clicking the **Add Generator...** button.



New generators require a name and a command. The name is shown in the tab label, and the command is run whenever the **Export Netlist** button is clicked.

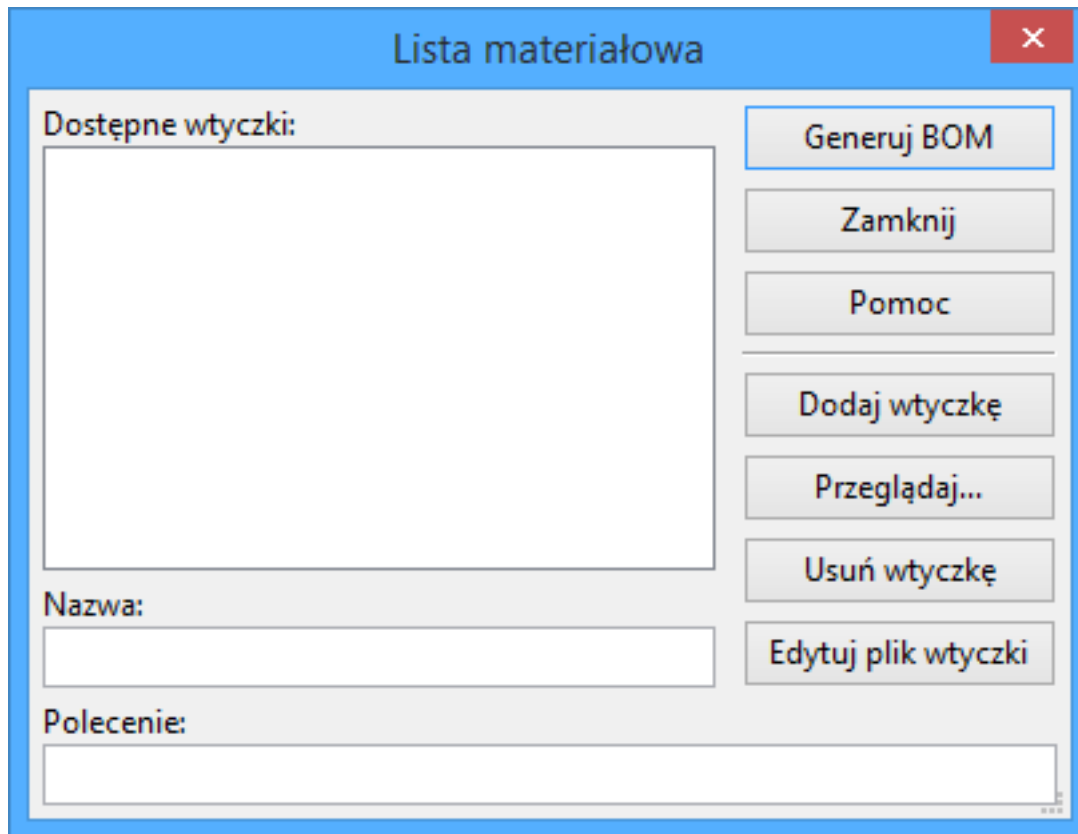
When the netlist is generated, KiCad creates an intermediate XML file which contains all of the netlist information from the schematic. The generator command is then run in order to transform the intermediate netlist into the desired netlist format.

The netlist command must be set up properly so that the netlist generator script takes the intermediate netlist file as input and outputs the desired netlist file. The exact netlist command will depend on the generator script used. The command format is described below.

Python and XSLT are commonly used tools to create custom netlist generators.

## 11.5.2. Adding a new BOM generator

KiCad also uses the intermediate netlist file to generate BOMs with the Generate BOM tool.



Additional scripts can be added to the list of BOM generator scripts by clicking the  button. Scripts can be removed by clicking the  button. The  button opens the selected script in a text editor.

Generator scripts written in Python and XSLT can contain a header comment that describes the generator's functionality and usage. This header comment is displayed in the BOM dialog as the description for each generator. The header comment must contain the string `@package`. Everything following that string until the end of the comment is used as the description for the generator.

KiCad automatically fills the command line field when a new generator script is added, but the command line might need to be adjusted by hand depending on the generator script. KiCad attempts to automatically determine the output file extension from the example command line in the generator script's header.

### 11.5.3. Generator command line format

The command line for a netlist or BOM exporter defines the command that KiCad will run to generate the selected output file.

For a netlist exporter using `xsltproc`, an example is:

```
xsltproc -o %0.net /usr/share/kicad/plugins/netlist_form_pads-pcb.asc.xsl %I
```

For a BOM exporter using Python, an example is:

```
/usr/bin/python3 /usr/share/kicad/plugins/bom_csv_grouped_by_value.py "%I"
"%O.csv"
```

### Uwaga

It is recommended to surround arguments in the command line with quotes (") in case they contain spaces or other special characters.

Some character sequences like %I and %O have a special meaning in the command line, because KiCad replaces them with a filename or path before executing the command.

Parameter	Replaced with...	Description
%I	<project path>/<project name>.xml	Absolute path and filename of the intermediate netlist file, which is the input to the BOM or netlist generator plugin
%O	<project path>/<project name>	Absolute path and filename of the output BOM or netlist file (without file extension). An appropriate file extension may need to be specified after the %O sequence.
%B	<project name>	Base filename of the output BOM or netlist file (without path or file extension). An appropriate file extension may need to be specified after the %B sequence.
%P	<project path>	Absolute path of the project directory, without trailing slash.

## 11.5.4. Plik pośredniej listy sieci

When exporting BOM files and netlists, KiCad creates an intermediate netlist file and then runs a separate tool which post-processes the intermediate netlist into the desired netlist or BOM format.

The intermediate netlist uses XML syntax. It contains a large amount of data about the design. Depending on the output (BOM or netlist), different subsets of the complete intermediate netlist file will be included in the final output file.

The structure of the intermediate netlist file is described in detail below.

Because the conversion from intermediate netlist file to output netlist or BOM is a text-to-text transformation, the post-processing filter can be written using Python, XSLT, or any other tool capable of taking XML as input.

### Uwaga

XSLT is not recommended for new netlist or BOM exporters; Python or another tool should be used instead. Beginning with KiCad 7, xsltproc is no longer installed with KiCad, although it can be installed separately. Nevertheless, several examples of netlist exporters using XSLT are included below.

## 11.5.5. Plik pośredni listy sieci

Poniższy przykład ukazuje ideę samego pliku pośredniego.

```
<?xml version="1.0" encoding="utf-8"?>
<export version="D">
```

```
<design>
 <source>F:\kicad_aux\netlist_test\netlist_test.sch</source>
 <date>29/08/2010 21:07:51</date>
 <tool>eeschema (2010-08-28 BZR 2458)-unstable</tool>
</design>
<components>
 <comp ref="P1">
 <value>CONN_4</value>
 <libsource lib="conn" part="CONN_4"/>
 <sheetpath names="/" tstamps="/" />
 <tstamps>4C6E2141</tstamps>
 </comp>
 <comp ref="U2">
 <value>74LS74</value>
 <libsource lib="74xx" part="74LS74"/>
 <sheetpath names="/" tstamps="/" />
 <tstamps>4C6E20BA</tstamps>
 </comp>
 <comp ref="U1">
 <value>74LS04</value>
 <libsource lib="74xx" part="74LS04"/>
 <sheetpath names="/" tstamps="/" />
 <tstamps>4C6E20A6</tstamps>
 </comp>
 <comp ref="C1">
 <value>CP</value>
 <libsource lib="device" part="CP"/>
 <sheetpath names="/" tstamps="/" />
 <tstamps>4C6E2094</tstamps>
 <comp ref="R1">
 <value>R</value>
 <libsource lib="device" part="R"/>
 <sheetpath names="/" tstamps="/" />
 <tstamps>4C6E208A</tstamps>
 </comp>
</components>
<libparts/>
<libraries/>
<nets>
 <net code="1" name="GND">
 <node ref="U1" pin="7"/>
 <node ref="C1" pin="2"/>
 <node ref="U2" pin="7"/>
 <node ref="P1" pin="4"/>
 </net>
 <net code="2" name="VCC">
 <node ref="R1" pin="1"/>
 <node ref="U1" pin="14"/>
 <node ref="U2" pin="4"/>
 <node ref="U2" pin="1"/>
 <node ref="U2" pin="14"/>
 <node ref="P1" pin="1"/>
 </net>
 <net code="3" name="">
 <node ref="U2" pin="6"/>
 </net>
 <net code="4" name="">
 <node ref="U1" pin="2"/>
 <node ref="U2" pin="3"/>
 </net>
</nets>
```

```

</net>
<net code="5" name="/SIG_OUT">
 <node ref="P1" pin="2"/>
 <node ref="U2" pin="5"/>
 <node ref="U2" pin="2"/>
</net>
<net code="6" name="/CLOCK_IN">
 <node ref="R1" pin="2"/>
 <node ref="C1" pin="1"/>
 <node ref="U1" pin="1"/>
 <node ref="P1" pin="3"/>
</net>
</nets>
</export>

```

## Struktura ogólna

Plik pośredni listy sieci posiada 5 sekcji:

- Sekcja nagłówka.
- Sekcja komponentów.
- Sekcja elementów bibliotecznych.
- Sekcja bibliotek.
- Sekcja sieci po##cze#.

The file content has the delimiter <export>

```

<export version="D">
...
</export>

```

## Sekcja nagłówka

The header has the delimiter <design>

```

<design>
<source>F:\kicad_aux\netlist_test\netlist_test.sch</source>
<date>21/08/2010 08:12:08</date>
<tool>eeschema (2010-08-09 BZR 2439)-unstable</tool>
</design>

```

Sekcja ta może być widoczna jako komentarze.

## Sekcja komponentów

The component section has the delimiter <components>

```

<components>
<comp ref="P1">
<value>CONN_4</value>
<libsource lib="conn" part="CONN_4"/>
<sheetpath names="/" tstamps="/">
<tstamps>4C6E2141</tstamps>

```

```
</comp>
</components>
```

Jest to lista na której znajdują się poszczególne komponenty schematu. Każdy komponent jest opisany w następujący sposób:

```
<comp ref="P1">
<value>CONN_4</value>
<libsource lib="conn" part="CONN_4"/>
<sheetpath names="/" tstamps="/">
<tstamps>4C6E2141</tstamps>
</comp>
```

Element name	Element description
libsource	name of the lib where this component was found.
part	component name inside this library.
sheetpath	path of the sheet inside the hierarchy: identify the sheet within the full schematic hierarchy.
tstamps	timestamp of the component.

===== Note about time stamps for components

To identify a component in a netlist and therefore on a board, the timestamp reference is used as unique for each component. However KiCad provides an auxiliary way to identify a component which is the corresponding footprint on the board. This allows the re-annotation of components in a schematic project and does not lose the link between the component and its footprint.

Znacznik czasowy jest unikalnym identyfikatorem dla każdego skadnika lub arkusza schematu w projekcie. Jednak w złożonych hierarchiach, w tym samym arkuszu skadnik może być używany więcej niż raz, a zatem arkusz ten zawiera elementy o tym samym znaczniku czasowym.

A given sheet inside a complex hierarchy has an unique identifier: its sheetpath. A given component (inside a complex hierarchy) has a unique identifier: the sheetpath and its timestamp.

## Sekcja elementów bibliotecznych

The libparts section has the delimiter `<libparts>`, and the content of this section is defined in the schematic libraries.

```
<libparts>
<libpart lib="device" part="CP">
 <description>Condensateur polarise</description>
 <footprints>
 <fp>CP*</fp>
 <fp>SM*</fp>
 </footprints>
 <fields>
 <field name="Reference">C</field>
 <field name="Valeur">CP</field>
 </fields>
 <pins>
 <pin num="1" name="1" type="passive"/>
 <pin num="2" name="2" type="passive"/>
 </pins>
</libpart>
```

```

</pins>
</libpart>
</libparts>

```

Element name	Element description
<footprints>	The symbol's footprint filters. Each footprint filter is in a separate <fp> tag.
<fields>	The symbol's fields. Each field's name and value is given in a separate <field name="fieldname">...</field> tag.
<pins>	The symbol's pins. Each pin is given in a separate <pin num="pinnum" type="pintype"/> tag. Possible pintypes are described below.

Possible electrical pin types are:

Pintype	Description
Input	Usual input pin
Output	Usual output
Bidirectional	Input or Output
Tri-state	Bus input/output
Passive	Usual ends of passive components
Unspecified	Unknown electrical type
Power input	Power input of a component
Power output	Power output like a regulator output
Open collector	Open collector often found in analog comparators
Open emitter	Open emitter sometimes found in logic
Not connected	Must be left open in schematic

## Sekcja bibliotek

The libraries section has the delimiter <libraries>. This section contains the list of schematic libraries used in the project.

```

<libraries>
 <library logical="device">
 <uri>F:\kicad\share\library\device.lib</uri>
 </library>
 <library logical="conn">
 <uri>F:\kicad\share\library\conn.lib</uri>
 </library>
</libraries>

```

## Sekcja sieci

The nets section has the delimiter <nets>. This section describes the connectivity of the schematic by listing all nets and the pins connected to each net.

```

<nets>
 <net code="1" name="GND">
 <node ref="U1" pin="7"/>
 <node ref="C1" pin="2"/>
 <node ref="U2" pin="7"/>
 <node ref="P1" pin="4"/>
 </net>
 <net code="2" name="VCC">
 <node ref="R1" pin="1"/>
 <node ref="U1" pin="14"/>
 <node ref="U2" pin="4"/>
 <node ref="U2" pin="1"/>
 <node ref="U2" pin="14"/>
 <node ref="P1" pin="1"/>
 </net>
</nets>

```

Poszczególne sieci *s#* pogrupowane wewnątrz tagu `<net>`:

```

<net code="1" name="GND">
 <node ref="U1" pin="7"/>
 <node ref="C1" pin="2"/>
 <node ref="U2" pin="7"/>
 <node ref="P1" pin="4"/>
</net>

```

Element name	Element Description
net code	an internal identifier for this net
name	the net name
node	the pin (identified by pin) of a symbol (identified by ref) which is connected to the net

## 11.5.6. Example netlist exporters

Some example netlist exporters using XSLT are included below.

XSLT itself is an XML language very suitable for XML transformations. The `xsltproc` program [<http://xmlsoft.org/XSLT/xsltproc.html>] can be used to read the Intermediate XML netlist input file, apply a style-sheet to transform the input, and save the results in an output file. Use of `xsltproc` requires a style-sheet file using XSLT conventions. The full conversion process is handled by KiCad, after it is configured once to run `xsltproc` in a specific way.

The document that describes XSL Transformations (XSLT) is available here: <http://www.w3.org/TR/xslt>

### Uwaga

When writing a new netlist exporter, consider using Python or another tool rather than XSLT.

## PADS netlist example using XSLT

The following example shows how to create an exporter for the PADS netlist format using `x1stproc`.

The PADS netlist format is comprised of two sections:

- A list of footprints
- A list of nets, together with the pads connected to each net.

Below is an XSL style-sheet which converts the intermediate netlist file to the PADS netlist format.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--XSL style sheet to Eeschema Generic Netlist Format to PADS netlist format
 Copyright (C) 2010, SoftPLC Corporation.
 GPL v2.

 How to use:
 https://lists.launchpad.net/kicad-developers/msg05157.html
-->

<!DOCTYPE xsl:stylesheet [
 <!ENTITY nl "
"> <!--new line CR, LF -->
]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" omit-xml-declaration="yes" indent="no"/>

<xsl:template match="/export">
 <xsl:text>*PADS-PCB*&nl;*PART*&nl;</xsl:text>
 <xsl:apply-templates select="components/comp"/>
 <xsl:text>&nl;*NET*&nl;</xsl:text>
 <xsl:apply-templates select="nets/net"/>
 <xsl:text>*END*&nl;</xsl:text>
</xsl:template>

<!-- for each component -->
<xsl:template match="comp">
 <xsl:text> </xsl:text>
 <xsl:value-of select="@ref"/>
 <xsl:text> </xsl:text>
 <xsl:choose>
 <xsl:when test = "footprint != '' ">
 <xsl:apply-templates select="footprint"/>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text>unknown</xsl:text>
 </xsl:otherwise>
 </xsl:choose>
 <xsl:text>&nl;</xsl:text>
</xsl:template>

<!-- for each net -->
<xsl:template match="net">
 <!-- nets are output only if there is more than one pin in net -->
 <xsl:if test="count(node)>1">
 <xsl:text>*SIGNAL* </xsl:text>
 <xsl:choose>
 <xsl:when test = "@name != '' ">
 <xsl:value-of select="@name"/>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text>N-</xsl:text>
 <xsl:value-of select="@code"/>
 </xsl:otherwise>
 </xsl:choose>
 </xsl:if>
</xsl:template>
```

```

 </xsl:choose>
 <xsl:text>&nl;</xsl:text>
 <xsl:apply-templates select="node"/>
 </xsl:if>
</xsl:template>

<!-- for each node -->
<xsl:template match="node">
 <xsl:text> </xsl:text>
 <xsl:value-of select="@ref"/>
 <xsl:text>.</xsl:text>
 <xsl:value-of select="@pin"/>
 <xsl:text>&nl;</xsl:text>
</xsl:template>

</xsl:stylesheet>

```

And here is the PADS netlist output file after running xsltproc:

```

PADS-PCB
PART
P1 unknown
U2 unknown
U1 unknown
C1 unknown
R1 unknown
NET
SIGNAL GND
U1.7
C1.2
U2.7
P1.4
SIGNAL VCC
R1.1
U1.14
U2.4
U2.1
U2.14
P1.1
SIGNAL N-4
U1.2
U2.3
SIGNAL /SIG_OUT
P1.2
U2.5
U2.2
SIGNAL /CLOCK_IN
R1.2
C1.1
U1.1
P1.3
END

```

Polecenie które dokona#o takiej konwersji wygl#da nast#puj#co:

```

kicad\\bin\\xsltproc.exe -o test.net kicad\\bin\\plugins\\netlist_form_pads-
pcb.xsl test.tmp

```

## Cadstar netlist example using XSLT

The following example shows how to create an exporter for the Cadstar netlist format using `xlstproc`.

The Cadstar format is comprised of two sections:

- The footprint list
- The Nets list: grouping pads references by nets

Below is an XSL style-sheet which converts the intermediate netlist file to the Cadstar netlist format.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--XSL style sheet to Eeschema Generic Netlist Format to CADSTAR netlist format
 Copyright (C) 2010, Jean-Pierre Charras.
 Copyright (C) 2010, SoftPLC Corporation.
 GPL v2. -->

<!DOCTYPE xsl:stylesheet [
 <!ENTITY nl "
"> <!--new line CR, LF -->
]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" omit-xml-declaration="yes" indent="no"/>

<!-- Netlist header -->
<xsl:template match="/export">
 <xsl:text>.HEA&nl;</xsl:text>
 <xsl:apply-templates select="design/date"/> <!-- Generate line .TIM <time> -->
 <xsl:apply-templates select="design/tool"/> <!-- Generate line .APP <eeschema version> -->
 <xsl:apply-templates select="components/comp"/> <!-- Generate list of components -->
 <xsl:text>&nl;&nl;</xsl:text>
 <xsl:apply-templates select="nets/net"/> <!-- Generate list of nets and components -->
 <xsl:text>&nl;.END&nl;</xsl:text>
</xsl:template>

 <!-- Generate line .TIM 20/08/2010 10:45:33 -->
<xsl:template match="tool">
 <xsl:text>.APP "</xsl:text>
 <xsl:apply-templates/>
 <xsl:text>"&nl;</xsl:text>
</xsl:template>

 <!-- Generate line .APP "eeschema (2010-08-17 BZR 2450)-unstable" -->
<xsl:template match="date">
 <xsl:text>.TIM </xsl:text>
 <xsl:apply-templates/>
 <xsl:text>&nl;</xsl:text>
</xsl:template>

<!-- for each component -->
<xsl:template match="comp">
 <xsl:text>.ADD_COM </xsl:text>
 <xsl:value-of select="@ref"/>
 <xsl:text> </xsl:text>
 <xsl:choose>
 <xsl:when test = "value != '' ">
 <xsl:text>"</xsl:text> <xsl:apply-templates select="value"/> <xsl:text>"</xsl:

```

```

 </xsl:when>
 <xsl:otherwise>
 <xsl:text>""</xsl:text>
 </xsl:otherwise>
 </xsl:choose>
 <xsl:text>&nl;</xsl:text>
</xsl:template>

<!-- for each net -->
<xsl:template match="net">
 <!-- nets are output only if there is more than one pin in net -->
 <xsl:if test="count(node)>1">
 <xsl:variable name="netname">
 <xsl:text>"</xsl:text>
 <xsl:choose>
 <xsl:when test = "@name != ' ' ">
 <xsl:value-of select="@name"/>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text>N-</xsl:text>
 <xsl:value-of select="@code"/>
 </xsl:otherwise>
 </xsl:choose>
 <xsl:text>"&nl;</xsl:text>
 </xsl:variable>
 <xsl:apply-templates select="node" mode="first"/>
 <xsl:value-of select="$netname"/>
 <xsl:apply-templates select="node" mode="others"/>
 </xsl:if>
</xsl:template>

<!-- for each node -->
<xsl:template match="node" mode="first">
 <xsl:if test="position()=1">
 <xsl:text>.ADD_TER </xsl:text>
 <xsl:value-of select="@ref"/>
 <xsl:text>.</xsl:text>
 <xsl:value-of select="@pin"/>
 <xsl:text> </xsl:text>
 </xsl:if>
</xsl:template>

<xsl:template match="node" mode="others">
 <xsl:choose>
 <xsl:when test='position()=1'>
 </xsl:when>
 <xsl:when test='position()=2'>
 <xsl:text>.TER </xsl:text>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text> </xsl:text>
 </xsl:otherwise>
 </xsl:choose>
 <xsl:if test="position()>1">
 <xsl:value-of select="@ref"/>
 <xsl:text>.</xsl:text>
 <xsl:value-of select="@pin"/>
 <xsl:text>&nl;</xsl:text>
 </xsl:if>

```

```
</xsl:template>
</xsl:stylesheet>
```

Poniżej znajduje się plik wyjściowy dla programu Cadstar.

```
.HEA
.TIM 21/08/2010 08:12:08
.APP "eeschema (2010-08-09 BZR 2439)-unstable"
.ADD_COM P1 "CONN_4"
.ADD_COM U2 "74LS74"
.ADD_COM U1 "74LS04"
.ADD_COM C1 "CP"
.ADD_COM R1 "R"

.ADD_TER U1.7 "GND"
. TER C1.2
 U2.7
 P1.4
.ADD_TER R1.1 "VCC"
. TER U1.14
 U2.4
 U2.1
 U2.14
 P1.1
.ADD_TER U1.2 "N-4"
. TER U2.3
.ADD_TER P1.2 "/SIG_OUT"
. TER U2.5
 U2.2
.ADD_TER R1.2 "/CLOCK_IN"
. TER C1.1
 U1.1
 P1.3

.END
```

## OrcadPCB2 netlist example using XSLT

This format has only one section which is the footprint list. Each footprint includes a list of its pads with reference to a net.

Below is an XSL style-sheet which converts the intermediate netlist file to the Orcad netlist format.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--XSL style sheet to Eeschema Generic Netlist Format to CADSTAR netlist format
Copyright (C) 2010, SoftPLC Corporation.
GPL v2.

How to use:
https://lists.launchpad.net/kicad-developers/msg05157.html
-->

<!DOCTYPE xsl:stylesheet [
 <!ENTITY nl "
"> <!--new line CR, LF -->
]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```

<xsl:output method="text" omit-xml-declaration="yes" indent="no"/>

<!--
 Netlist header
 Creates the entire netlist
 (can be seen as equivalent to main function in C
-->
<xsl:template match="/export">
 <xsl:text>({ Eeschema Netlist Version 1.1 </xsl:text>
 <!-- Generate line .TIM <time> -->
<xsl:apply-templates select="design/date"/>
<!-- Generate line eeschema version ... -->
<xsl:apply-templates select="design/tool"/>
<xsl:text>}&nl;</xsl:text>

<!-- Generate the list of components -->
<xsl:apply-templates select="components/comp"/> <!-- Generate list of components -->

<!-- end of file -->
<xsl:text>)&nl;*&nl;</xsl:text>
</xsl:template>

<!--
 Generate id in header like "eeschema (2010-08-17 BZR 2450)-unstable"
-->
<xsl:template match="tool">
 <xsl:apply-templates/>
</xsl:template>

<!--
 Generate date in header like "20/08/2010 10:45:33"
-->
<xsl:template match="date">
 <xsl:apply-templates/>
 <xsl:text>&nl;</xsl:text>
</xsl:template>

<!--
 This template read each component
 (path = /export/components/comp)
 creates lines:
 (3EBF7DBD $noname U1 74LS125
 ... pin list ...
)
 and calls "create_pin_list" template to build the pin list
-->
<xsl:template match="comp">
 <xsl:text> (</xsl:text>
 <xsl:choose>
 <xsl:when test = "tstamp != ' ' ">
 <xsl:apply-templates select="tstamp"/>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text>00000000</xsl:text>
 </xsl:otherwise>
 </xsl:choose>
 <xsl:text> </xsl:text>
 <xsl:choose>
 <xsl:when test = "footprint != ' ' ">

```

```

 <xsl:apply-templates select="footprint"/>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text>$noname</xsl:text>
 </xsl:otherwise>
</xsl:choose>
<xsl:text> </xsl:text>
<xsl:value-of select="@ref"/>
<xsl:text> </xsl:text>
<xsl:choose>
 <xsl:when test = "value != ' ' ">
 <xsl:apply-templates select="value"/>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text>"~"</xsl:text>
 </xsl:otherwise>
</xsl:choose>
<xsl:text>&nl;</xsl:text>
<xsl:call-template name="Search_pin_list" >
 <xsl:with-param name="cmplib_id" select="libsource/@part"/>
 <xsl:with-param name="cmp_ref" select="@ref"/>
</xsl:call-template>
<xsl:text>)&nl;</xsl:text>
</xsl:template>

<!--
 This template search for a given lib component description in list
 lib component descriptions are in /export/libparts,
 and each description start at ./libpart
 We search here for the list of pins of the given component
 This template has 2 parameters:
 "cmplib_id" (reference in libparts)
 "cmp_ref" (schematic reference of the given component)
-->
<xsl:template name="Search_pin_list" >
 <xsl:param name="cmplib_id" select="0" />
 <xsl:param name="cmp_ref" select="0" />
 <xsl:for-each select="/export/libparts/libpart">
 <xsl:if test = "@part = $cmplib_id ">
 <xsl:apply-templates name="build_pin_list" select="pins/pin">
 <xsl:with-param name="cmp_ref" select="$cmp_ref"/>
 </xsl:apply-templates>
 </xsl:if>
 </xsl:for-each>
</xsl:template>

<!--
 This template writes the pin list of a component
 from the pin list of the library description
 The pin list from library description is something like
 <pins>
 <pin num="1" type="passive"/>
 <pin num="2" type="passive"/>
 </pins>
 Output pin list is (<pin num> <net name>)
 something like
 (1 VCC)
 (2 GND)

```

```

-->
<xsl:template name="build_pin_list" match="pin">
 <xsl:param name="cmp_ref" select="0" />

 <!-- write pin number and separator -->
 <xsl:text> (</xsl:text>
 <xsl:value-of select="@num"/>
 <xsl:text> </xsl:text>

 <!-- search net name in nets section and write it: -->
 <xsl:variable name="pinNum" select="@num" />
 <xsl:for-each select="/export/nets/net">
 <!-- net name is output only if there is more than one pin in net
 else use "?" as net name, so count items in this net
 -->
 <xsl:variable name="pinCnt" select="count(node)" />
 <xsl:apply-templates name="Search_pin_netname" select="node">
 <xsl:with-param name="cmp_ref" select="$cmp_ref"/>
 <xsl:with-param name="pin_cnt_in_net" select="$pinCnt"/>
 <xsl:with-param name="pin_num"> <xsl:value-of select="$pinNum"/>
 </xsl:with-param>
 </xsl:apply-templates>
 </xsl:for-each>

 <!-- close line -->
 <xsl:text>)&nl;</xsl:text>
</xsl:template>

<!--
 This template writes the pin netname of a given pin of a given component
 from the nets list
 The nets list description is something like
 <nets>
 <net code="1" name="GND">
 <node ref="J1" pin="20"/>
 <node ref="C2" pin="2"/>
 </net>
 <net code="2" name="">
 <node ref="U2" pin="11"/>
 </net>
 </nets>
 This template has 2 parameters:
 "cmp_ref" (schematic reference of the given component)
 "pin_num" (pin number)
-->

<xsl:template name="Search_pin_netname" match="node">
 <xsl:param name="cmp_ref" select="0" />
 <xsl:param name="pin_num" select="0" />
 <xsl:param name="pin_cnt_in_net" select="0" />

 <xsl:if test="@ref = $cmp_ref">
 <xsl:if test="@pin = $pin_num">
 <!-- net name is output only if there is more than one pin in net
 else use "?" as net name
 -->
 <xsl:if test="$pin_cnt_in_net>1">
 <xsl:choose>
 <!-- if a net has a name, use it,

```

```

 else build a name from its net code
 -->
 <xsl:when test = "../@name != '' ">
 <xsl:value-of select="../@name"/>
 </xsl:when>
 <xsl:otherwise>
 <xsl:text>$N-0</xsl:text><xsl:value-of select="../@code"/>
 </xsl:otherwise>
 </xsl:choose>
 </xsl:if>
 <xsl:if test = "$pin_cnt_in_net < 2">
 <xsl:text>?</xsl:text>
 </xsl:if>
</xsl:if>
</xsl:template>
</xsl:stylesheet>

```

Poniżej znajduje się plik wyjściowy programu OrcadPCB2.

```

({ Eeschema Netlist Version 1.1 29/08/2010 21:07:51
eeschema (2010-08-28 BZR 2458)-unstable}
(4C6E2141 $noname P1 CONN_4
(1 VCC)
(2 /SIG_OUT)
(3 /CLOCK_IN)
(4 GND)
)
(4C6E20BA $noname U2 74LS74
(1 VCC)
(2 /SIG_OUT)
(3 N-04)
(4 VCC)
(5 /SIG_OUT)
(6 ?)
(7 GND)
(14 VCC)
)
(4C6E20A6 $noname U1 74LS04
(1 /CLOCK_IN)
(2 N-04)
(7 GND)
(14 VCC)
)
(4C6E2094 $noname C1 CP
(1 /CLOCK_IN)
(2 GND)
)
(4C6E208A $noname R1 R
(1 VCC)
(2 /CLOCK_IN)
)
)
*)

```

---

## Rozdzia# 12. Actions reference

Below is a list of every available **action** in the KiCad Schematic Editor: a command that can be assigned to a hotkey.

### 12.1. Schematic Editor

The actions below are available in the Schematic Editor. Hotkeys can be assigned to any of these actions in the **Hotkeys** section of the preferences.

Action	Default Hotkey	Description
Align Items to Grid		
Annotate Schematic...		Fill in schematic symbol reference designators
Annotate Automatically		Toggle automatic annotation of new symbols
Assign Footprints...		Run footprint assignment tool
Clear Net Highlighting	kbd:[~]	Clear any existing net highlighting
Export Drawing to Clipboard		Export drawing of current sheet to clipboard
Edit Library Symbol...	kbd:[Ctrl+Shift+E]	Open the library symbol in the Symbol Editor
Edit Sheet Page Number...		Edit the page number of the current or selected sheet
Edit Symbol Fields...		Bulk-edit fields of all symbols in schematic
Edit Symbol Library Links...		Edit links between schematic and library symbols
Edit with Symbol Editor	kbd:[Ctrl+E]	Open the selected symbol in the Symbol Editor
Export Netlist...		Export file containing netlist in one of several formats
Export Symbols to Library...		Add symbols used in schematic to an existing symbol library (does not remove other symbols from this library)
Export Symbols to New Library...		Create a new symbol library using the symbols used in the schematic (if the library already exists it will be replaced)
Generate Bill of Materials...		Generate a bill of materials for the current schematic

Action	Default Hotkey	Description
Generate Bill of Materials (External)...		Generate a bill of materials for the current schematic using external generator
Generate Legacy Bill of Materials...		Generate a bill of materials for the current schematic (Legacy Generator)
Highlight Net	kbd:[`] ]	Highlight net under cursor
Highlight Nets		Highlight wires and pins of a net
Import Footprint Assignments...		Import symbol footprint assignments from .cmp file created by board editor
Import Graphics...	kbd:[Ctrl +Shift+F]	Import 2D drawing file
Increment Annotations From...		Increment a subset of reference designators starting at a particular symbol
Line Mode for Wires and Buses		Constrain drawing and dragging to horizontal, vertical, or 45-degree angle motions
Line Mode for Wires and Buses		Draw and drag at any angle
Line Mode for Wires and Buses	kbd:[Shift +Space]	Switch to next line mode
Line Mode for Wires and Buses		Constrain drawing and dragging to horizontal or vertical motions
Mark items excluded from simulation		Draw 'X's over items which have been excluded from simulation
Next Symbol Unit		Open the next unit of the symbol
Previous Symbol Unit		Open the previous unit of the symbol
Remap Legacy Library Symbols...		Remap library symbol references in legacy schematics to the symbol library table
Repair Schematic		Run various diagnostics and attempt to repair schematic
Rescue Symbols...		Find old symbols in project and rename/rescue them
Save Current Sheet Copy As...		Save a copy of the current sheet to another location or name
Schematic Setup...		Edit schematic setup including annotation styles and electrical rules
Select on PCB		Select corresponding items in PCB editor
Do not Populate		Set the do not populate attribute

Action	Default Hotkey	Description
Exclude from Bill of Materials		Set the exclude from bill of materials attribute
Exclude from Board		Set the exclude from board attribute
Exclude from Simulation		Set the exclude from simulation attribute
Show Directive Labels		
Show ERC Errors		Show markers for electrical rules checker errors
Show ERC Exclusions		Show markers for excluded electrical rules checker violations
Show ERC Warnings		Show markers for electrical rules checker warnings
Show Hidden Fields		
Show Hidden Pins		
Net Navigator		Show/hide the net navigator
Show OP Currents		Show operating point current data from simulation
Show OP Voltages		Show operating point voltage data from simulation
Switch to PCB Editor		Open PCB in board editor
Simulator		Show simulation window for running SPICE or IBIS simulations.
Show Pin Alternate Icons		Show indicator icons for pins with alternate modes
Hierarchy Navigator	kbd:[Ctrl+H]	Show/hide the schematic sheet hierarchy navigator
Symbol Checker		Show the symbol checker window
Compare Symbol with Library		Show differences between schematic symbol and its library equivalent
Electrical Rules Checker		Show the electrical rules checker window
Show Bus Syntax Help		
Decrement Primary		Decrement the primary field of the selected item(s)
Decrement Secondary		Decrement the secondary field of the selected item(s)
Increment		Increment the selected item(s)

Action	Default Hotkey	Description
Increment Primary		Increment the primary field of the selected item(s)
Increment Secondary		Increment the secondary field of the selected item(s)
Close Outline		Close the in-progress outline
Delete Last Point		Delete the last point added to the current item
Draw Arcs		
Draw Bezier Curve		
Draw Circles		
Draw Rectangles		
Draw Rule Areas		
Draw Hierarchical Sheets	kbd:[S]	
Draw Sheet from Design Block		Copy design block into project as a sheet on current sheet
Draw Sheet from File		Copy sheet into project and draw on current sheet
Draw Tables		
Draw Text Boxes		
Import Sheet		Import sheet into project
Place Wire to Bus Entries	kbd:[Z]	
Place Directive Labels		
Place Design Block	kbd:[Shift+B]	Add selected design block to current sheet
Place Global Labels	kbd:[Ctrl+L]	
Place Hierarchical Labels	kbd:[H]	
Place Images		
Place Junctions	kbd:[J]	
Place Net Labels	kbd:[L]	
Place Next Symbol Unit		Place the next unit of the current symbol that is missing from the schematic
Place No Connect Flags	kbd:[Q]	

Action	Default Hotkey	Description
Place Power Symbols	kbd:[P]	
Draw Text	kbd:[T]	
Place Sheet Pins		
Place Symbols	kbd:[A]	
Sync Sheet Pins		Synchronize sheet pins and hierarchical labels
Sync Sheet Pins		Synchronize sheet pins and hierarchical labels
Draw Buses	kbd:[B]	
Draw Lines	kbd:[I]	
Draw Wires	kbd:[W]	
Switch Segment Posture	kbd:[/]	Switches posture of the current segment.
Undo Last Segment	kbd:[Back]	Walks the current line back one segment.
Unfold from Bus	kbd:[C]	Break a wire out of a bus
Assign Netclass...		Assign a netclass to nets matching a pattern
Autoplace Fields	kbd:[O]	Runs the automatic placement algorithm on the symbol's (or sheet's) fields
Break		Divide into connected segments
Change Symbol...		Assign a different symbol from the library
Change Symbols...		Assign different symbols from the library
Cleanup Sheet Pins		Delete unreferenced sheet pins
Edit Footprint...	kbd:[F]	
Edit Reference Designator...	kbd:[U]	
Edit Text & Graphics Properties...		Edit text and graphics properties globally across schematic
Edit Value...	kbd:[V]	
Mirror Horizontally	kbd:[X]	Flips selected item(s) from left to right
Mirror Vertically	kbd:[Y]	Flips selected item(s) from top to bottom
Pin Table...		Displays pin table for bulk editing of pins
Properties...	kbd:[E]	
Repeat Last Item	kbd:[Ins]	Duplicates the last drawn item

Action	Default Hotkey	Description
Rotate Counterclockwise	kbd:[R]	
Rotate Clockwise		
De Morgan Alternate		Switch to alternate De Morgan representation
De Morgan Standard		Switch to standard De Morgan representation
Slice		Divide into unconnected segments
Swap	kbd:[Alt+S]	Swap positions of selected items
Symbol Properties...		
Change to Directive Label		Change existing item to a directive label
Change to Global Label		Change existing item to a global label
Change to Hierarchical Label		Change existing item to a hierarchical label
Change to Label		Change existing item to a label
Change to Text		Change existing item to a text comment
Change to Text Box		Change existing item to a text box
De Morgan Conversion		Switch between De Morgan representations
Update Symbol...		Update symbol to include any changes from the library
Update Symbols from Library...		Update symbols to include any changes from the library
Drag	kbd:[G]	Move items while keeping their connections
Move	kbd:[M]	
Select Connection	kbd:[Ctrl+4]	Select a complete connection
Select Node	kbd:[Alt+3]	Select a connection item under the cursor
Navigate Back	kbd:[Alt+Left]	Move backward in sheet navigation history
Change Sheet		Change to provided sheet's contents in the schematic editor
Enter Sheet		Display the selected sheet's contents in the schematic editor
Navigate Forward	kbd:[Alt+Right]	Move forward in sheet navigation history
Leave Sheet	kbd:[Alt+Back]	Display the parent sheet in the schematic editor

Action	Default Hotkey	Description
Next Sheet	kbd:[PgDn]	Move to next sheet by number
Previous Sheet	kbd:[PgUp]	Move to previous sheet by number
Navigate Up	kbd:[Alt+Up]	Navigate up one sheet in the hierarchy
Push Pin Length		Copy pin length to other pins in symbol
Push Pin Name Size		Copy pin name size to other pins in symbol
Push Pin Number Size		Copy pin number size to other pins in symbol
Create Corner		
Remove Corner		
Properties...		Edit properties of design block
Delete Design Block		Remove the selected design block from its library
Save Selection as Design Block...		Create a new design block from the current selection
Save Current Sheet as Design Block...		Create a new design block from the current sheet
Design Blocks		Show/hide design blocks library
User-defined Signals...		Add, edit or delete user-defined simulation signals
New Analysis Tab...	kbd:[Ctrl+N]	Create a new tab containing a simulation analysis
Open Workbook...	kbd:[Ctrl+O]	Open a saved set of analysis tabs and settings
Probe Schematic...	kbd:[P]	Add a simulator probe
Run Simulation	kbd:[R]	
Save Workbook	kbd:[Ctrl+S]	Save the current set of analysis tabs and settings
Save Workbook As...	kbd:[Ctrl+Shift+S]	Save the current set of analysis tabs and settings to another location
Show SPICE Netlist		
Edit Analysis Tab...		Edit the current analysis tab's SPICE command and plot setup
Stop Simulation		
Add Tuned Value...	kbd:[T]	Select a value to be tuned
Export Current Plot as CSV...		

Action	Default Hotkey	Description
Export Current Plot as PNG...		
Export Current Plot to Schematic		
Export Current Plot to Clipboard		
Dark Mode Plots		Draw plots with a black background
Dotted Current/Phase		Draw secondary signal trace (current or phase) with a dotted line
Show Legend		
Draw Lines		Draw connected graphic lines
Draw Polygons		
Draw Text Boxes		
Move Symbol Anchor		
Draw Pins	kbd:[P]	
Draw Text		
Add Symbol to Schematic		Add the current symbol to the schematic
Copy		
Cut		
Delete Symbol		Remove the selected symbol from its library
Derive from Existing Symbol...		Create a new symbol, derived from an existing symbol
Duplicate Symbol		
Edit Symbol		Show selected symbol on editor canvas
Export Symbol as SVG...		Create SVG file from the current symbol
Export View as PNG...		Create PNG file from the current view
Import Symbol...		Import a symbol to the current library
New Symbol...	kbd:[Ctrl+N]	Create a new symbol in an existing library
Paste Symbol		
Rename Symbol...		
Save Library As...	kbd:[Ctrl+Shift+S]	Save the current library to a new file
Save As...		Save the current symbol to a different library or name

Action	Default Hotkey	Description
Save Copy As...		Save a copy of the current symbol to a different library or name
Set Unit Display Name...		Set the display name for a particular unit in a multi-unit symbol
Show Pin Electrical Types		Annotate pins with their electrical types
Show Hidden Fields		
Show Hidden Pins		
Show Pin Numbers		Annotate pins with their numbers
Synchronized Pins Mode		Synchronized Pins Mode When enabled propagates all changes (except pin numbers) to other units. Enabled by default for multiunit parts with interchangeable units.
Update Symbol Fields...		Update symbol to match changes made in parent symbol

## 12.2. Common

The actions below are available across KiCad, including in the Schematic Editor. Hotkeys can be assigned to any of these actions in the **Hotkeys** section of the preferences.

Action	Default Hotkey	Description
Refresh Plugins		Reload all python plugins and refresh plugin menus
Exclude Marker		Mark current violation in Checker window as an exclusion
Next Marker		
Previous Marker		
Add Library...		Add an existing library folder
Center Justify		Center-justify fields and text items
Pan to Center Selected Objects		
Collapse All		
Click	kbd:[Return]	Performs left mouse button click
Double-click	kbd:[End]	Performs left mouse button double-click
Cursor Down	kbd:[Down]	
Cursor Down Fast	kbd:[Ctrl +Down]	
Cursor Left	kbd:[Left]	

Action	Default Hotkey	Description
Cursor Left Fast	kbd:[Ctrl+Left]	
Cursor Right	kbd:[Right]	
Cursor Right Fast	kbd:[Ctrl+Right]	
Cursor Up	kbd:[Up]	
Cursor Up Fast	kbd:[Ctrl+Up]	
Grid Origin...		Set the grid origin point
Edit Grids...		Edit grid definitions
Expand All		
Switch to Fast Grid 1	kbd:[Alt+1]	
Switch to Fast Grid 2	kbd:[Alt+2]	
Cycle Fast Grid	kbd:[Alt+4]	
Switch to Next Grid	kbd:[N]	
Switch to Previous Grid	kbd:[Shift+N]	
Reset Grid Origin		
Grid Origin		Place the grid origin point
Hide Library Tree		
Inactive Layer View Mode		Toggle inactive layers between normal and dimmed
Inactive Layer View Mode (3-state)	kbd:[H]	Cycle inactive layers between normal, dimmed, and hidden
Inches		
Left Justify		Left-justify fields and text items
Focus Library Tree Search Field	kbd:[Ctrl+L]	
Snap to Objects on the Active Layer Only		Enables snapping to objects on the active layer only
Snap to Objects on All Layers		Enables snapping to objects on all visible layers

Action	Default Hotkey	Description
Toggle Snapping Between Active and All Layers	kbd:[Shift+S]	Toggles between snapping on all visible layers and only the active area
Millimeters		
Mils		
New...	kbd:[Ctrl+N]	Create a new document in the editor
New Library...		Create a new library folder
Open...	kbd:[Ctrl+O]	Open existing document
Open in file explorer...		Open a library file with system file explorer
Edit in a Text Editor...		Open a library file with a text editor
Page Settings...		Settings for paper size and title block info
Pan Down	kbd:[Shift+Down]	
Pan Left	kbd:[Shift+Left]	
Pan Right	kbd:[Shift+Right]	
Pan Up	kbd:[Shift+Up]	
Pin Library		Keep the library at the top of the list
Plot...		
Print...	kbd:[Ctrl+P]	
Quit		Close the current editor
Redo Last Zoom		Return zoom to level prior to last zoom undo
Reset Local Coordinates	kbd:[Space]	
Revert		Throw away changes
Right Justify		Right-justify fields and text items
Save	kbd:[Ctrl+S]	Save changes
Save All		Save all changes
Save As...	kbd:[Ctrl+Shift+S]	Save current document to another location
Save a Copy...		Save a copy of the current document to another location
Select Columns...		
3D Viewer	kbd:[Alt+3]	Show 3D viewer window

Action	Default Hotkey	Description
Show Context Menu		Perform the right-mouse-button action
Show Datasheet	kbd:[D]	Open the datasheet in a browser
Footprint Library Browser		
Footprint Editor		Create, delete and edit board footprints
Library Tree		
Switch to Project Manager		Show project window
Properties		Show/hide the properties manager
Symbol Library Browser		
Symbol Editor		Create, delete and edit schematic symbols
Draw Bounding Boxes		
Always Show Crosshairs	kbd:[Ctrl+Shift+X]	Display crosshairs even when not drawing objects
Full-Window Crosshairs		Switch display of full-window crosshairs
Show Grid		Display background grid in the edit window
Grid Overrides	kbd:[Ctrl+Shift+G]	Enables item-specific grids that override the current grid
Polar Coordinates		Switch between polar and cartesian coordinate systems
Switch units	kbd:[Ctrl+U]	Switch between imperial and metric units
Undo Last Zoom		Return zoom to level prior to last zoom action
Unpin Library		No longer keep the library at the top of the list
Update PCB from Schematic...	kbd:[F8]	Update PCB with changes made to schematic
Update Schematic from PCB...		Update schematic with changes made to PCB
Center on Cursor	kbd:[F4]	
Zoom to Objects	kbd:[Ctrl+Home]	
Zoom to Fit	kbd:[Home]	
Zoom to Selected Objects		
Zoom In at Cursor	kbd:[F1]	
Zoom In		

Action	Default Hotkey	Description
Zoom In Horizontally		Zoom in horizontally the plot area
Zoom In Vertically		Zoom in vertically the plot area
Zoom Out at Cursor	kbd:[F2]	
Zoom Out		
Zoom Out Horizontally		Zoom out horizontally the plot area
Zoom Out Vertically		Zoom out vertically the plot area
Refresh	kbd:[F5]	
Zoom to Selection	kbd:[Ctrl+F5]	
Embedded Files		Manage embedded files
Extract File		Extract an embedded file
Remove File		Remove an embedded file
Cancel		Cancel current tool
Copy	kbd:[Ctrl+C]	Copy selected item(s) to clipboard
Copy as Text	kbd:[Ctrl+Shift+C]	Copy selected item(s) to clipboard as text
Cut	kbd:[Ctrl+X]	Cut selected item(s) to clipboard
Cycle Arc Editing Mode	kbd:[Ctrl+Space]	Switch to a different method of editing arcs
Delete	kbd:[Del]	Delete selected item(s)
Interactive Delete Tool		Delete clicked items
Duplicate	kbd:[Ctrl+D]	Duplicates the selected item(s)
Find	kbd:[Ctrl+F]	
Find and Replace	kbd:[Ctrl+Alt+F]	
Find Next	kbd:[F3]	
Find Next Marker	kbd:[Ctrl+Shift+F3]	
Find Previous	kbd:[Shift+F3]	
Finish	kbd:[End]	Finish current tool
Measure Tool	kbd:[Ctrl+Shift+M]	Interactively measure distance between points

Action	Default Hotkey	Description
Paste	kbd:[Ctrl+V]	Paste item(s) from clipboard
Paste Special...		Paste item(s) from clipboard with options
Redo	kbd:[Ctrl+Y]	
Replace All		
Replace and Find Next		
Search	kbd:[Ctrl+G]	Show/hide the search panel
Select All	kbd:[Ctrl+A]	Select all items on screen
Undo	kbd:[Ctrl+Z]	
Unselect All	kbd:[Ctrl+Shift+A]	Unselect all items on screen
Select Row(s)		Select complete row(s) containing the current selected cell(s)
Select Column(s)		Select complete column(s) containing the current selected cell(s)
Select Table		Select parent table of selected cell(s)
Select item(s)		
About KiCad		
Configure Paths...		Edit path configuration environment variables
Donate		Open "Donate to KiCad" in a web browser
Get Involved		Open "Contribute to KiCad" in a web browser
Getting Started with KiCad		Open "Getting Started in KiCad" guide for beginners
Help		Open product documentation in a web browser
List Hotkeys...	kbd:[Ctrl+F1]	Displays current hotkeys table and corresponding commands
Preferences...	kbd:[Ctrl+,]	Show preferences for all open tools
Report Bug		Report a problem with KiCad
Manage Design Block Libraries...		Edit the global and project design block library lists
Manage Footprint Libraries...		Edit the global and project footprint library lists
Manage Symbol Libraries...		Edit the global and project symbol library lists
Add Column After		Insert a new table column after the selected cell(s)
Add Column Before		Insert a new table column before the selected cell(s)
Add Row Above		Insert a new table row above the selected cell(s)
Add Row Below		Insert a new table row below the selected cell(s)

Action	Default Hotkey	Description
Delete Column(s)		Delete columns containing the currently selected cell(s)
Delete Row(s)		Delete rows containing the currently selected cell(s)
Merge Cells		Turn selected table cells into a single cell
Unmerge Cells		Turn merged table cells back into separate cells.