QUCS-S Documentation

The QUCS-S Contributors

Dec 06, 2025

WELCOME

Qucs-S is a circuit simulation program, forked from the original Qucs circuit simulator.

Qucs-S aims to provide a convenient, unified GUI around various free circuit simulation kernels (such as ngspice, Xyce, Qucsator, etc). It merges the power of SPICE with the simplicity of the Qucs GUI.

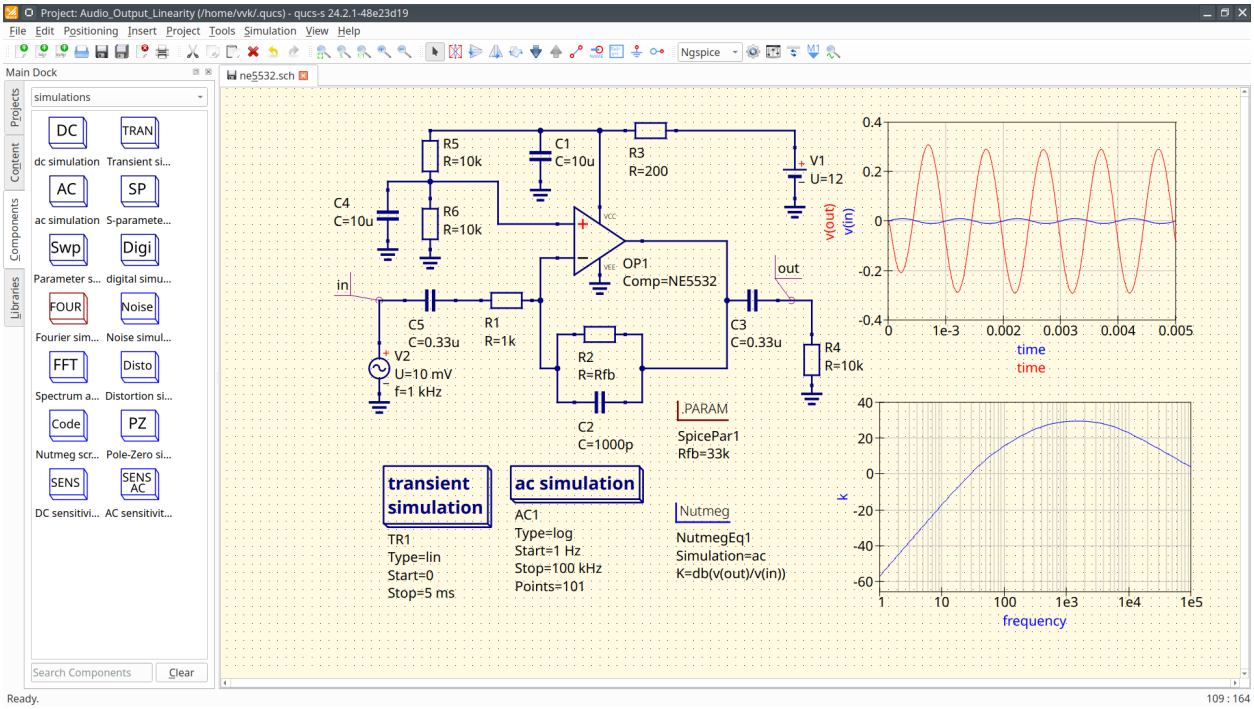


Fig. 1: An example of an NE5532 amplifier simulated in Qucs-S, using the ngspice simulation backend.

You may wish to:

- Download Qucs-S
- View Qucs-S Source on GitHub
- Discuss in the Project Forum
- Donate to the project via Boosty
- View Legacy Documentation (largely obsolete, but maintained for reference)

Please be advised that this documentation is a work-in-progress. However, if you find an error in this documentation, please submit an issue on the docs repository on GitHub. Pull requests are also welcomed!

Otherwise, continue on through this documentation to learn more about QUCS-S.

CHAPTER

ONE

WHAT IS QUCS-S?

QUCS-S is a fork of the popular Qucs Circuit Simulator. The fork began in 2017.

The "S" in QUCS-S is for SPICE - the primary purpose of QUCS-S is to merge the user-friendly Qucs GUI with the power of the SPICE simulation ecosystem.

1.1 QUCS-S vs Qucs

The original Qucs project uses its own SPICE-incompatible simulator, called Qucsator. Qucsator has advanced RF and AC-domain simulation features, but is incompatible with most SPICE models used across industry.

1.2 QUCS-S vs QucsStudio

QucsStudio is another simulator package which is also based on the original Qucs project. QucsStudio is freeware, but not open-source. Apart from being forked from the same original Qucs project, QucsStudio has no relation to QUCS-S.

1.3 Simulation Backends

In contrast to Qucs and QucsStudio, QUCS-S does not include its own simulation backend at all. Rather, it serves as a frontend for several different simulation backends:

1.3.1 Analog/Digital/RF Simulators

- **ngspice (recommended)**: A powerful mixed-level/mixed-signal circuit simulator. Most SPICE models distributed across industry are compatible with it. It has excellent performance for time-domain simulation of switching circuits, and a powerful postprocessor. If you are unsure which simulation backend to use with QUCS-S, ngspice is recommended.
- **Xyce**: A new SPICE-compatible circuit simulator, written from scratch by Sandia National Laboratory. Xyce has the notable advantage of supporting large-scale parallel computing platforms, making it a good fit for solving very large circuits (although it can run on an ordinary desktop platform as well).
- **SpiceOpus**: A free general purpose circuit simulator, based on the Berkeley SPICE-3f5 codebas, specially suited for optimization loops.
- **QucsatorRF**: A fork of Qucsator, the built-in simulation engine from the original Qucs project. QucsatorRF shares the original Qucsator netlist syntax, and all RF features. It's primarily intended for RF simulation with microwave devices and microstrip lines. It is not generally recommended for general-purpose circuit simulation, since ngspice typically has better performance.

1.3.2 Digital-Only Simulators

- **Icarus Verilog**: A digital-only simulation backend for simulating Verilog devices.
- **GHDL**: A digital-only simulation backend for simulating VHDL devices. Fully supports the 1987, 1993, 2002 versions of the IEEE 1076 VHDL standard, and partially the latest 2008 revision (well enough to support fixed_generic_pkg or float_generic_pkg).

1.4 Notable Features

A few features of particular note are:

- Basic and advanced simulation types: AC, DC, transient, S-parameter, FFT, distortion, pole-zero, parametric sweep.
- Advanced RF simulation with Qucsator RF backend;
- Quick switching of the simulation kernel without application restart;
- Tuner simulation mode
- Direct support of SPICE models from components datasheets;
- Basic SPICE components: RCL, BJT, MOSFET, JFET, MESFET, switches;
- Advanced SPICE components: Equation-defined sources and RCLs, transmission lines;
- Parametric circuits (.PARAM) and SPICE postprocessor (Nutmeg)
- Basic SPICE simulations: DC, AC, TRAN;
- Advanced SPICE simulation: DISTO, NOISE, SENS (added in 0.0.20), Spectrum analysis;
- Harmonic balance analysis with XYCE and Qucsator RF backends;
- Nutmeg script simulation: direct access to the SPICE code and construct your own simulation;
- XYCE script simulation type;
- XYCE digital devices library;

1.5 Supported Platforms

QUCS-S is available for Windows and Mac OSX, as well as numerous Linux distributions.

Keep in mind that some simulation backends may need to be installed separately from QUCS-S, depending on your installation platform and method.

See *Installing QUCS-S* for more information.

CHAPTER

TWO

INSTALLING QUCS-S

2.1 Windows

For each release, an executable installer (not a `.msi` package) is available, as well as a `.zip` file containing a portable installation. Visit the Releases page on GitHub.

Both the installer and the portable version include ngspice and QucsatorRF simulation backends.

2.2 Mac OSX

OSX installation `.dmg`'s are available with each Release on GitHub.

There are also homebrew packages available in a separate GitHub repository.

Only the QucsatorRF simulation backend is included with these binaries. Other simulation backends will need to be installed separately.

2.3 Linux

For Debian, Ubuntu, Fedora, and OpenSUSE, `.deb` and `.rpm` packages are available on the OpenSUSE Build Service. Simply follow the installation instructions on the build service page.

Arch and Manjaro users may install QUCS-S from AUR.

AltLinux packages are available from Sysyphus repo, here.

> **Note: If you install on Ubuntu or Debian using the packages above, the `ngspice` simulator and the QucsatorRF simulator are both installed automatically.** Other platforms only come with QucsatorRF installed. If you wish to use any of the other supported simulation backends, you will need to install them separately._

For all other distributions, an AppImage file is available. Get the latest release from GitHub.

2.4 FreeBSD

QUCS-S can be installed on FreeBSD using either ports or a binary package.

- To install ports: `cd /usr/ports/cad/qucs-s/ && make install clean`
- To install binary package: `pkg install qucs-s`

Only the QucsatorRF simulation backend is included with this install method. Other simulation backends will need to be installed separately.

2.5 Compiling from Source

Of course, Qucs-S can be compiled from source if desired. See the main Qucs-S GitHub repository's README for the latest information on how to compile from source.

Once you have installed QUCS-S, you may need to install simulation backends separately. This is covered in the next chapter.

CHAPTER

THREE

INSTALLING SIMULATION BACKENDS

Qucs-S is able to serve as an interface around 6 different simulation backends. However, these 6 backends are not all bundled with a Qucs-S install. See the sections below for information on installing each backend.

> **Tip**
>
> For tips on which backend is appropriate for your use case, see *Choosing a Simulation Backend.*

3.1 Executable Paths and Simulator Settings

Qucs-S must be able to access the executables for each simulation backend. The Mixed-Signal Simulators (ngspice, Xyce, SpiceOpus, and QucsatorRF) are configured in a special dialog within Qucs-S, but the Digital-Only Simulators (IVerilog and GHDL) are pulled from your system's environment variables. See the sections below for details.

3.1.1 Mixed-Signal Simulators

If you install a simulation backend, but Qucs-S is not showing it in the dropdown menu at the top of the window, you may need to manually point Qucs-S to the simulation executable.

You can do this using the `Simulate > Simulators Settings` menu and manually specifying the path, as shown in the image below.

3.1.2 Digital-Only Simulators

> **Tip**
>
> Unfamiliar with the digital simulation capabilities of Qucs-S? Visit *Digital Simulation* for an overview.

The executables for the digital backends, Icarus Verilog and GHDL, are automatically located using your system's `PATH` environment variable. For IVerilog, you must ensure that the `iverilog` executable is set up in your system's environment variable. For GHDL, you must ensure that the `ghdl` executable is set up in your system's environment variable.

3.2 Installing Mixed-Signal Backends

3.2.1 ngspice

3.2.1.1 Windows

ngspice is included with the Qucs-S Windows Installer, no additional installation steps are needed.

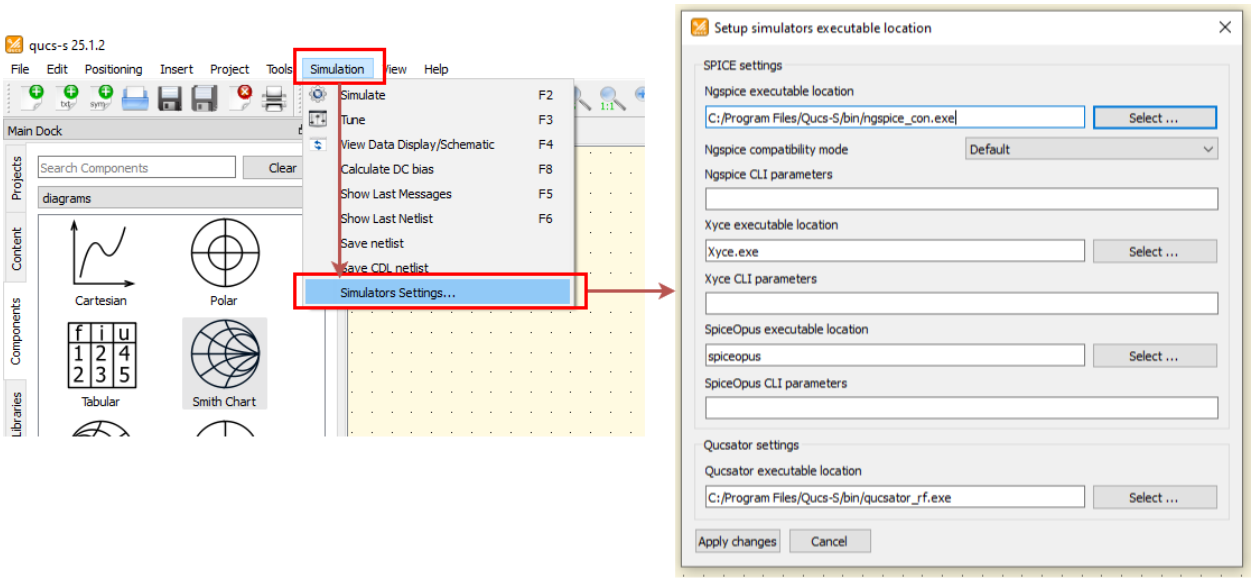


Fig. 1: Navigating to the *Simulator Settings* dialog, which allows you to configure the paths to your simulation backend executables.

3.2.1.2 Linux

If you install Qucs-S using the provided packages with your distribution's package manager, ngspice is typically installed as a dependency.

Linux distributions which do not receive official packages from Qucs-S, or users who opt to compile from source, may need to install ngspice manually.

3.2.1.3 MacOS

ngspice is NOT included with the `.dmg`'s provided on the Qucs-S GitHub, or with the homebrew packages. ngspice must be installed manually, alongside Qucs-S.

3.2.1.4 Other Platforms

Visit the ngspice project website for installation instructions.

3.2.2 Xyce

Xyce is not bundled with Qucs-S on any platform, it must be installed separately if desired. It is compatible with Windows (including Windows 11), numerous Linux distributions, and MacOS. For installation instructions, visit the Xyce project website.

3.2.3 SpiceOpus

SpiceOpus is also not bundled with Qucs-S on any platform, it must be installed separately if desired. It is currently compatible with Windows and Linux. Visit the SpiceOpus website for installation instructions.

3.2.4 QucsatorRF

The latest stable QucsatorRF is included with every Qucs-S release package, for all platforms. Unless you are building from source, no additional steps should be necessary to use QucsatorRF as a simulation backend.

3.3 Installing Digital-Only Backends

3.3.1 IVerilog

Icarus Verilog is not included with any Qucs-S package, it must be installed separately. There are a few options:

- For Linux, depending on your distribution, a version of IVerilog may be available through the package manager.
- For Windows, Pablo Bleyer Kocik provides Windows installers on his website. This is not officially affiliated with the Icarus Verilog project or the Qucs-S project, so if you have security concerns, it may be best to compile from the official source instead.
- For other platforms, see the Icarus Verilog project website for instructions on compiling from source.

3.3.2 GHDL

The GHDL project provides official binaries for numerous platforms, including Windows, MacOS, and various Linux distributions, on their GitHub Releases page.

CHAPTER

FOUR

INTERFACE OVERVIEW

This page will introduce you to the basic controls in the Qucs-S user interface.

4.1 Main Window Summary

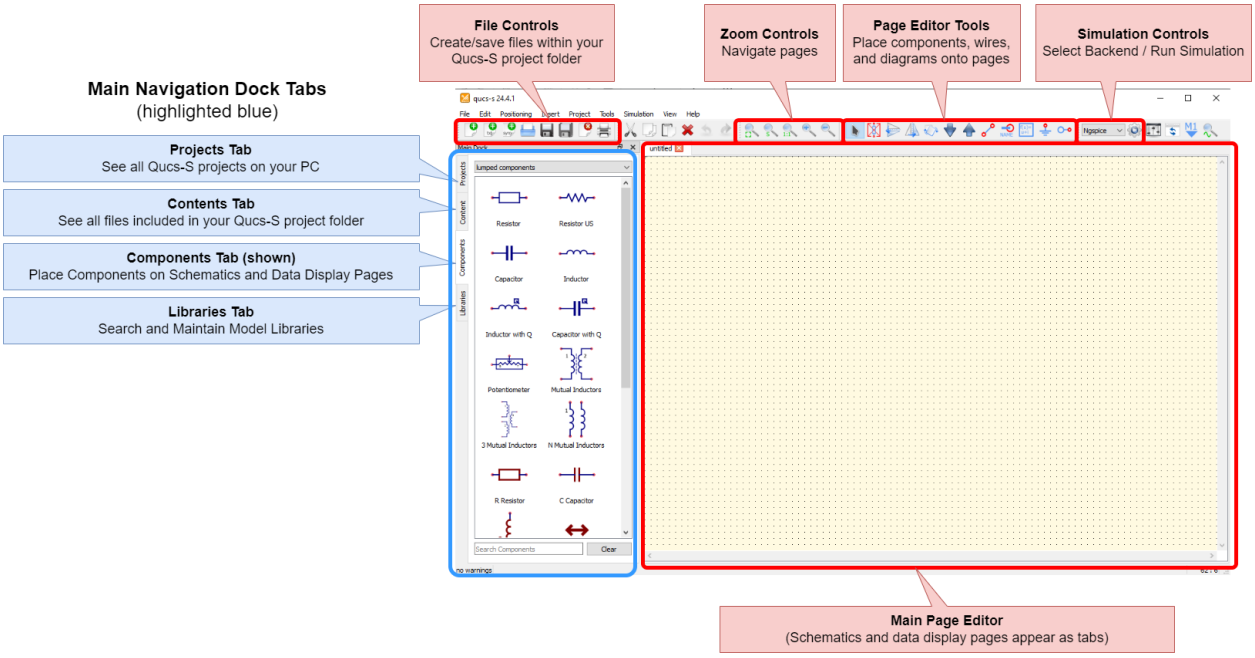


Fig. 1: The main Qucs-S window, with the most important controls annotated.

Here are the major controls to be aware of in the main Qucs-S window:

1. **Main Navigation Dock Tabs**: These tabs are used to manage your project's files, and add components and diagrams to your project's pages. See section below for details.
2. **File Controls**: Use these buttons to create and save files (such as schematic pages) within your Qucs-S project.
3. **Zoom Controls**: Use these controls to navigate the main page editor.
4. **Page Editor Tools**: Use these buttons to place components and wires on your pages, and modify their position/orientation.
5. **Simulation Controls**: Select your *simulation backend* using the drop-down menu. Then push the gear icon to run your simulation.

6. **Main Page Editor**: A tabbed navigation area in the center of the window. Each schematic page, data display page, and other resource in your Qucs-S project will appear as a tab here.

4.2 Main Navigation Dock Tabs

The left portion of the Qucs-S window contains the main navigation dock. It can be repositioned at will, but it defaults to the left side of the window.

It contains 4 tabs, described below (from top to bottom).

4.2.1 Projects Tab

This tab is used to choose a Qucs-S project to work in. It lists all the Qucs-S projects stored in the default directory (Typically, `C:\Users\John Doe\QucsWorkspace` or `/home/johndoe/QucsWorkspace`).

See the *Understanding File Structure* page for more information on how Qucs-S projects are stored.

> **Warning**
>
> **Qucs-S can only have one active project at a time.** In other words, you *cannot* have a schematic open from project A, and also open a schematic from project B. If you try to do this, Qucs-S will simply close the first project's files.

This tab allows a few basic tasks:

- To activate a project, either double-click it, or single-click it and choose "Open". Upon activating the project, you'll be taken to the Contents tab, summarizing that project.
- To create a new project, click the "New" button.
- To delete a project, click the "Delete" button.
- If your Qucs-S Home Directory contains subfolders to organize your projects, you can navigate the subfolders within the Projects tab by double-clicking (much like a typical file explorer view).

4.2.2 Contents Tab

The Contents tab serves as an organized summary of all the files in your active project. It is akin to a file explorer within your Qucs-S project folder.

To open a document within your project, double-click on it within the Contents tab. For instance, double-clicking the schematic `test1_bogatin.sch` in the screenshot below will open that schematic page for editing, as a tab in the main page editor.

For more information on the types of files which make up a Qucs-S project, and how they are stored, see *Understanding File Structure*.

4.2.3 Components Tab

This tab can be used to place various components onto a schematic page or display page. Here, "components" is a broad term, which includes:

- Ideal passive components
- Simulation commands
- References to external SPICE files
- Diagrams, for graphing/displaying simulation results

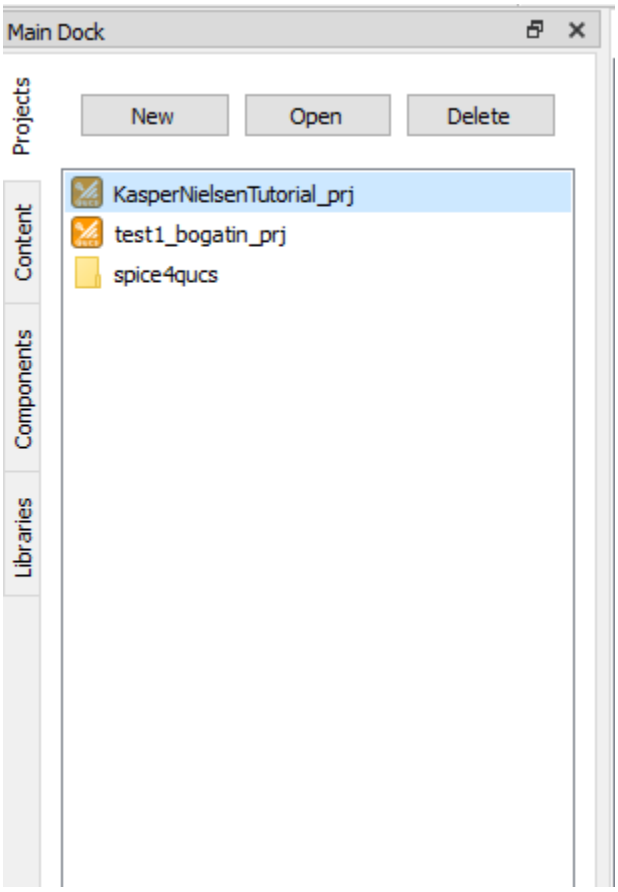


Fig. 2: The Qucs-S Projects tab, in the main navigation dock.

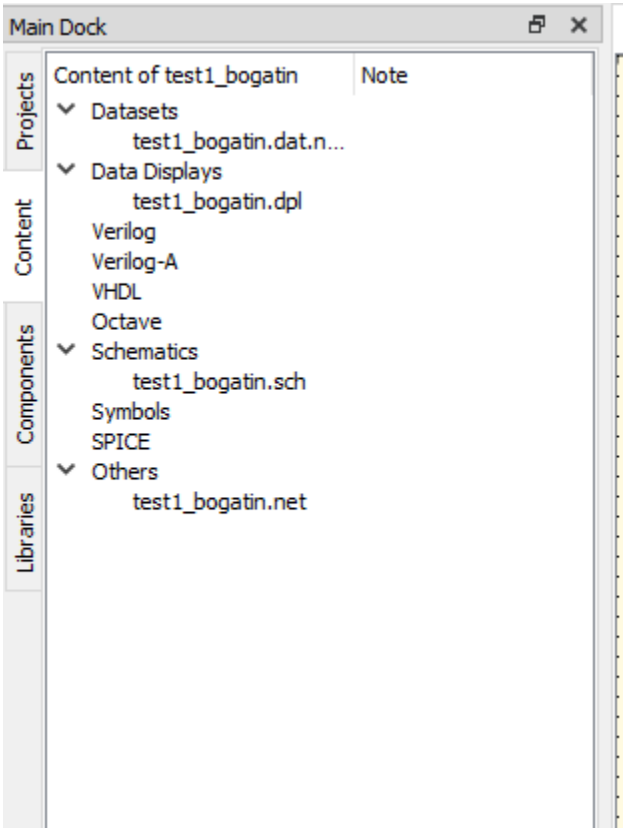


Fig. 3: An example project, with its constituent files shown in the Contents tab.

- Equations
- Paintings (non-intelligent, graphics-only elements such as text, lines, rectangles, etc)

> **Note**
>
> All those things, and more, are components. However, specific real-world parts (e.g. "LM334 Current Source") are *NOT* part of the Components tab. Rather, these models of real-world parts come from the separate Libraries tab.

> **Warning**
>
> The contents of the components tab may change if you *select a different simulation backend.* Not all components are compatible with all simulation backends.

4.2.3.1 Component Color Coding

All Qucs-S components can be classified as one of two types:

1. **Universal (Colored Blue)**: These components are compatible with both Qucsator and ngspice simulation backends. This allows easy simulation of the same circuit across multiple backends, however, the downside is that the full set of special features in the SPICE models may not be leveraged.
2. **SPICE-Only (Colored Red)**: These components are ONLY compatible with the SPICE-based backends (ngspice recommended). However, the full SPICE model specification is accessible, allowing more advanced customization of device models. This is useful when modeling real-world semiconductor devices, based on manufacturer-provided model parameters. These components will only be visible when a SPICE-compatible backend is selected (such as ngspice).

See the figure below for examples of each component type.

The same colors remain when these components are placed on a schematic page.

4.2.3.2 Placing a Component

1. Select a component category, using the drop-down at the top of the tab. For instance, "Nonlinear Components."
2. Find the component you want, either by scrolling or by using the "Search Components" feature at the bottom of the tab.
3. Single-click the component to select it for placement.
4. Move your mouse into the main page editor, and single-click to place the component. You can place multiple copies of the component by continuing to click.
5. When you are done placing the component, push `ESC`.

4.2.4 Libraries Tab

This tab contains all the models of real-world parts that are configured with Qucs-S.

For example: where the Components tab might contain an configurable, general model for PMOSFET, this tab contains numerous models for real-world, purchaseable PMOSFETs. This example is shown in the figure below.

Similar to the Components tab, single-clicking on one of the Library items will allow you to place instances of it onto the page.

Within this tab, component models come from 3 places:

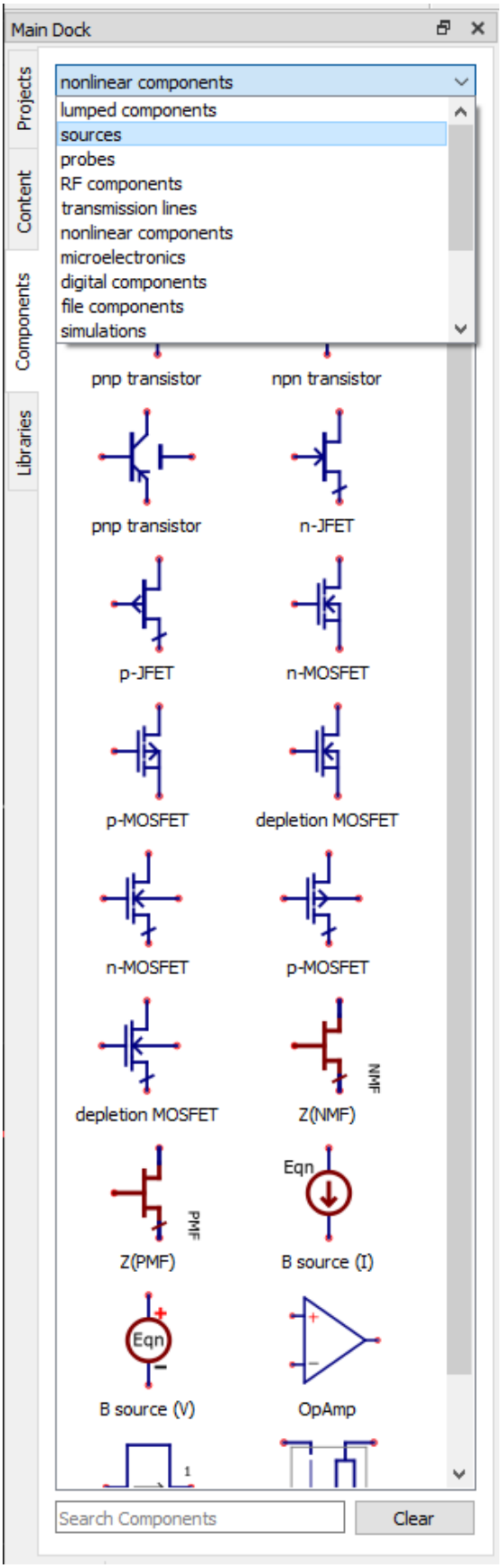


Fig. 4: An example of the Components tab.

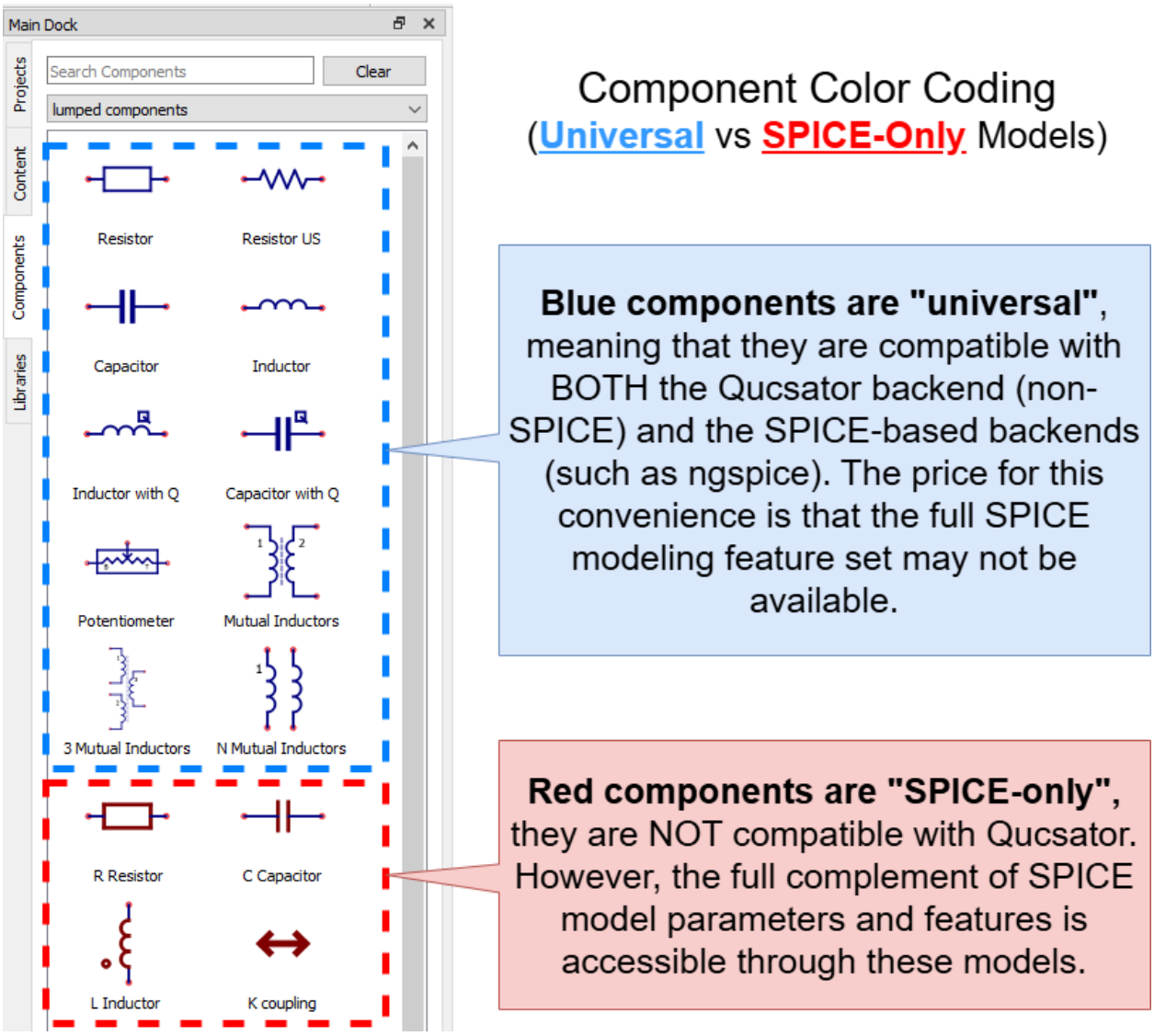


Fig. 5: Comparison of “Universal” components (blue color) and “SPICE-only” components (red color).

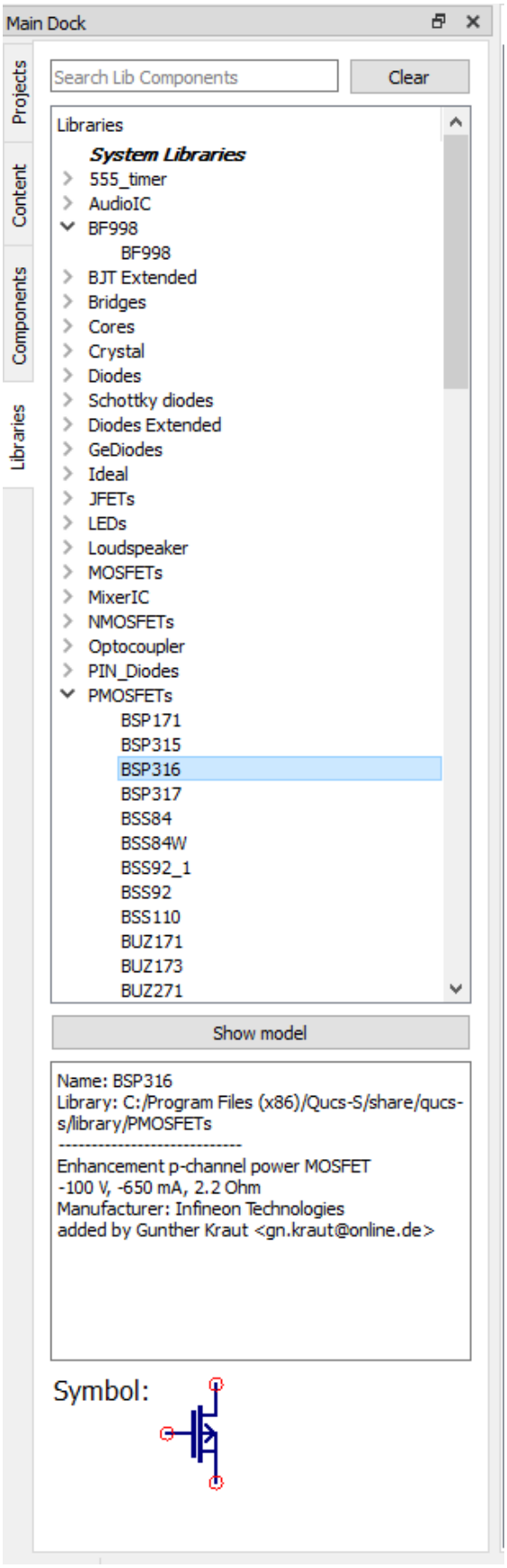


Fig. 6: An example of the Libraries tab, with the PMOSFETS category in the standard System Libraries expanded, and the BSS92 part selected. Note that the selected component's symbol, as well as basic metadata, is shown in the bottom portion of the tab.

1. **System Libraries**: These libraries come with QUCS-S upon installation.
2. **User Libraries**: These libraries are configurable by the user, and persist across all projects on a given PC.
3. **Project Libraries**: These libraries only exist within the active Qucs-S project.

For more information, see the section on *Managing Libraries in QUCS-S*.

4.3 Selecting a Simulation Backend

To select a simulation backend, use the drop-down menu at the top right of the Qucs-S window. Be advised that:

- **The drop-down menu will only show simulators for which an executable has been found.** In other words, it will only show simulators which are *both* installed on your system *and* configured properly within Qucs-S (via the Simulation > Simulators Settings dialog).
- **The drop-down menu will never show the digital-only simulation backends (GHDL and IVerilog).** These backends are called via a special object on the schematic page.

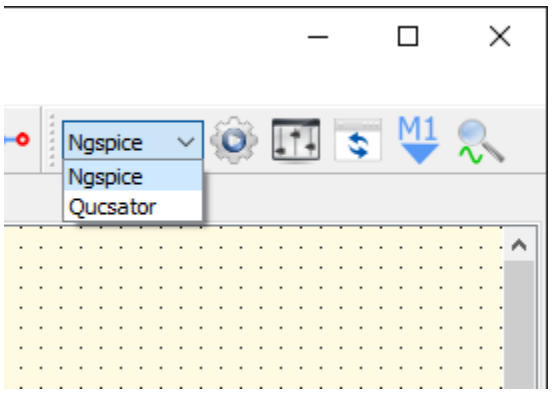


Fig. 7: The drop-down menu for choosing your simulation backend. Note that only simulators which are installed and properly configured with Qucs-S are shown in this menu, and the digital-only backends (GHDL and IVerilog) are never shown in this menu.

For more information on which simulation backend is appropriate for your situation, see *Choosing a Simulation Backend*.

4.3.1 Incompatible Items on Page

Normally, Qucs-S will not allow you to place components on the page which are incompatible with your *currently selected* simulation backend. For example, if you have ngspice selected in the drop-down menu, the Components tab will only contain items compatible with the ngspice simulator.

However, if you are changing between simulators frequently for your project, you may end up with incompatible components on your schematic. For instance, if you set the project up with ngspice selected as your backend, but you later change to Qucsator or Xyce, some components may not be compatible with the new backend.

In such cases, Qucs-S will warn you of the incompatibility by greying out the component. You can still run a simulation, but **these incompatible components will be treated as open-circuits and completely excluded from the simulation.** An example is shown below.

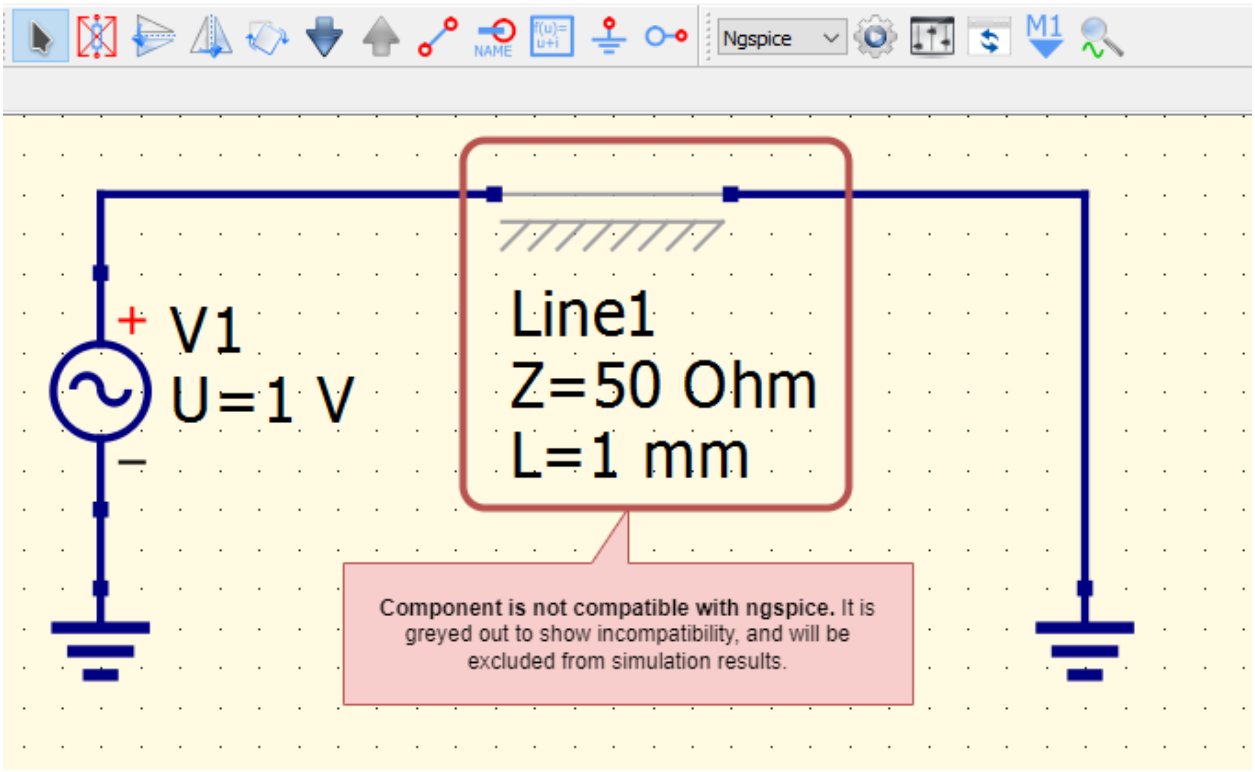


Fig. 8: Qucs-S is indicating that the transmission line component is incompatible with the selected simulation backend (ngspice). In this example, this component was placed while Qucsator was selected as the backend, and will not be included in any simulation results produced with ngspice.

4.4 Navigating Pages

To move around within schematic/display pages, use the following controls:

- **Pan by Dragging**: Middle mouse button
- **Zoom In/Out**: CTRL + Scroll Wheel
- **Pan by Scrolling**: Scroll for Vertical Pan, SHIFT + Scroll for Horizontal Pan
- **Select Multiple Items**: Hold CTRL and single-click

CHAPTER

FIVE

UNDERSTANDING FILE STRUCTURE

To understand how Qucs-S stores files, you must understand at a basic level how the simulation process works.

1. **The schematic is the primary input to a simulation.** It can reference external models (SPICE subcircuits, netlists, Verilog models, etc) if needed, or in simple cases it may only use models built-in to your selected circuit simulator.
2. **When the Simulate button is clicked, the schematic data is fed into the simulator and produces a Dataset.** Datasets are the raw data from a single run of a simulation in a given simulator.
3. **Datasets are used to generate graphical diagrams, so you can interpret your results.** Diagrams include Cartesian plots, Smith charts, polar plots, truth tables, etc. Diagrams are most commonly placed on the same schematic page as your circuit, although they can be placed on separate Data Display Pages if you desire.

The figure below describes this process, and the files it references.

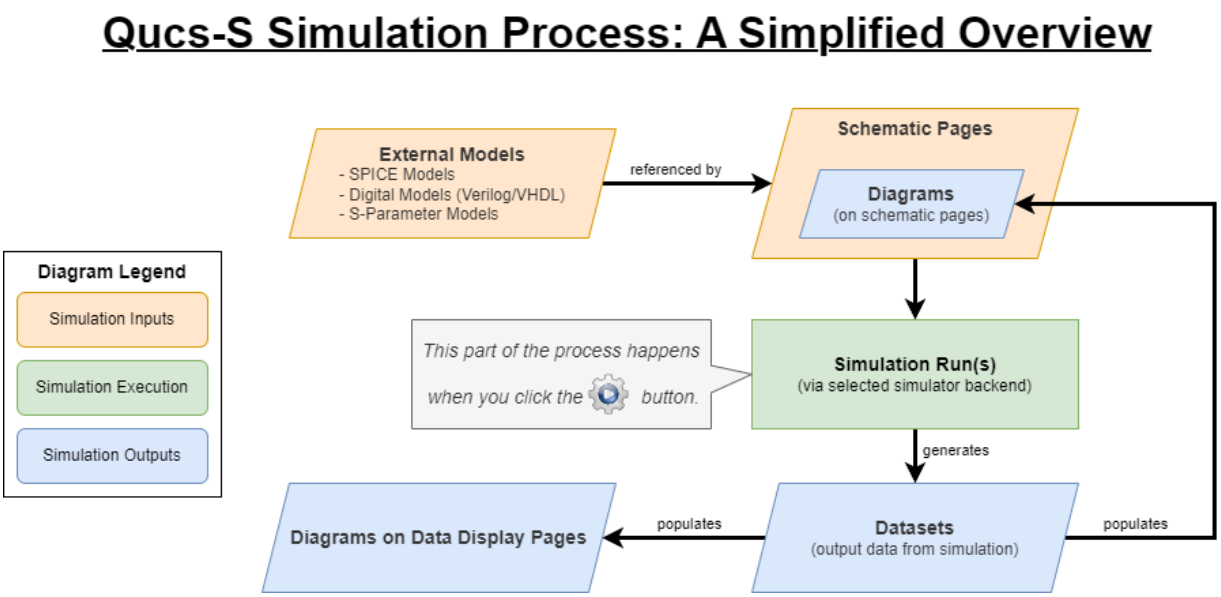


Fig. 1: A simplified diagram of the Qucs-S simulation process, showing which files it takes as inputs, and which files are produced as outputs.

5.1 Projects

A Qucs-S Project is a folder in the Qucs-S Home Directory, containing multiple files related to Qucs-S simulations (such as schematics, datasets, SPICE models, etc).

To be considered a Project, the folder's name *MUST* end in `_prj`. For example, `myCircuit_prj` is a valid Project folder name, while `myCircuit` is not.

5.1.1 Qucs-S Home Directory

The Qucs-S home directory will typically default to `$HOME/QucsWorkspace`. If an older installation of Qucs-S or original Qucs is found, it may default to `$HOME/.qucs` instead.

It can be set to an arbitrary location by visiting `File > Application Settings > Locations > Qucs Home`, as shown in the figure below.

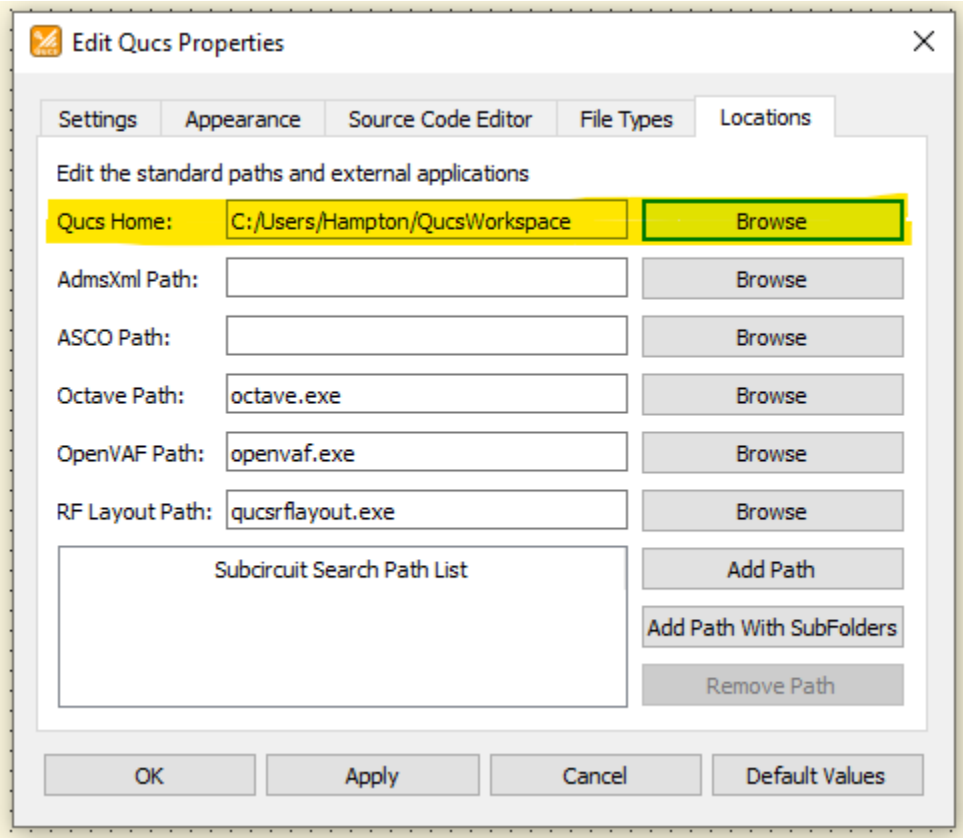


Fig. 2: Where to change the Qucs-S home directory, in the File > Application Settings menu.

5.1.2 File Types in a Project

A Project may contain many different types of files, listed below:

Input Files:

- **Schematics** (`.sch`): These are the schematic pages within Qucs-S. One project may contain several schematics, and the schematics also support hierarchy (similar to the Hierarchical Sheets feature in KiCAD and other EDA software).
- **SPICE** (`.cir`, `.ckt`, or `.sp`): External SPICE models. You may reference these in your Qucs-S schematic.

- **Symbols** (`.sym`): Files created by Qucs-S when adding a *Spice Library Device* (a method of creating circuit symbols around imported SPICE models). These will reference SPICE files, the SPICE files are *not* embedded in the `.sym` file.
- **Verilog** and **VHDL**: Verilog or VHDL source files, used only for pure-digital simulations.
- **Verilog-A** (`.va`): Analog Verilog-A source files. Further documentation is needed for this feature.

Output Files:

- **Datasets** (`.dat`, `.dat.ngspice`, and `.dat.xyce`): Raw output data from a simulation run. Extension and format will vary depending on the simulation backend used.
- **Data Displays** (`.dpl`): Pages used only for displaying Diagrams showing simulation results. These are not created automatically, since you can simply place Diagrams directly on your schematics. You may still use them if you prefer.

You can use the Contents tab in the Main Navigation Dock (see *Interface Overview*) to inspect all the files in your current open Qucs-S project. They will be displayed in groups based on the file types described above. An example of how this may look is shown below.

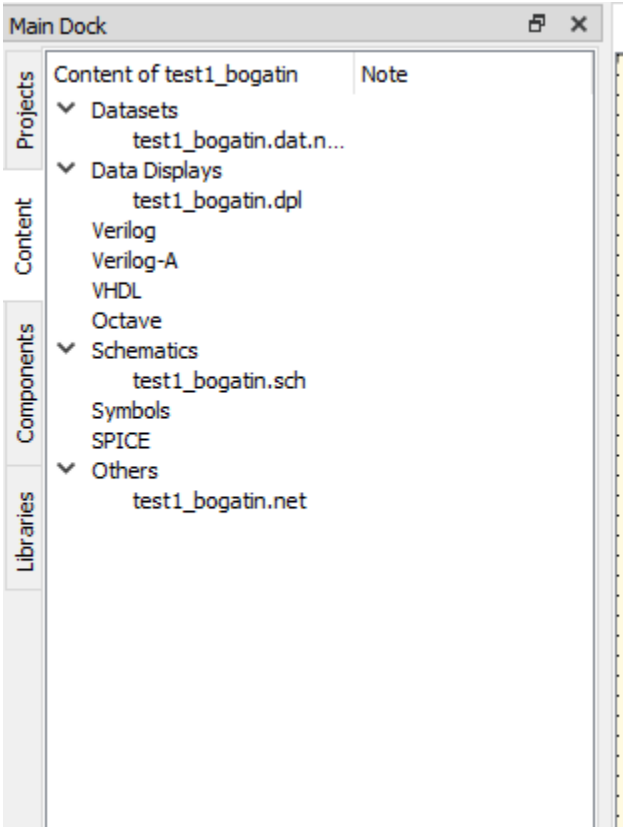


Fig. 3: An example of the Contents tab, showing the files in the currently-opened project, grouped into their types.

The example above corresponds to the following structure on the filesystem:

Qucs-S Home Directory (`$HOME/.qucs` in this case):

- `test1_bogatin_prj` : The Project folder
  - `test1_bogatin.dat.ngspice` : Raw Dataset output from an ngspice run.
  - `test1_bogatin.dpl` : Data Display page.

- `test1_bogatin.net` : Netlist file used by ngspice.
- `test1_bogatin.sch` : Schematic page defining the simulation.

5.1.3 Sharing Projects (Best Practices)

You may want to share Qucs-S projects between multiple computers. To do this, it's recommended to ZIP up the entire Project folder into an archive file, then unzip the archive into the Qucs-S Home Directory on the second computer. By doing this, Qucs-S will have access to all the needed files, and you will not have any broken references.

> **Warning**
>
> Remember that Qucs-S will only consider a folder a *Project* if its name ends in `_prj`! If you are unzipping a shared project on your system, be sure your unzipped folder still follows the `_prj` naming rule.

5.2 Working Outside a Project

It is possible to work outside a Project folder, storing files outside the Qucs-S home directory. You can open any type of Qucs-S supported file (schematic, data display, symbol, etc) using the *Open* button in the top right of the main window.

However, this may make files difficult to find on your system, since Qucs-S will place the simulation datasets (and any other additional files) next to your Schematic (`.sch`) files on the filesystem.

If you'd like to open a `.sch` file and save it into a Project for continued work, simply open it with the Open command, then use the Save As button to save a copy in your Project folder. You can then open your Project and continue your work.

> **Warning**
>
> **If you open a `.sch` file in a directory where you do not have write access, you will not be able to display any simulation results!** When you attempt to run a simulation, Qucs-S will be unable to save the simulation *Dataset* in the same directory as the *Schematic*, and therefore unable to render any *Diagrams* or otherwise display the output of the simulation.
>
> **This usually applies to the built-in example schematics!** (accessible via `File > Examples`) These files are stored in the Qucs-S program folder, which is not writeable by normal users on most operating systems. To use these files, you'll need to open them and perform a `File > Save As` into a folder where you have write access.

CHAPTER

SIX

CHOOSING A SIMULATION BACKEND

6.1 Quick Start

Support for multiple simulation backends is one of Qucs-S's key features. Each backend has its own advantages, disadvantages, and differentiating features. This makes Qucs-S an extremely capable and well-rounded simulation platform.

But for new users, this myriad of options can be overwhelming. **To that end, here are some simple rules of thumb to help you select a backend to get going with Qucs-S.**

6.1.1 Rules of Thumb

1. If your project involves mostly microwave/RF models (such as transmission lines), rather than traditional circuit components, use QucsatorRF.
2. If you know that you need the special additional features of Xyce or SpiceOpus, use one of those engines.
3. Otherwise, use ngspice.

Tip

Remember, a simulation backend is not a long-term commitment. You can always change which simulation backend you're using for a given project. See *Interface Overview : Selecting a Simulation Backend*.

Caution

As your simulations get more sophisticated, or if you begin to run into performance problems, you may need to go beyond the rules of thumb above. Ask for help on the Qucs-S Discussion Forum. More experienced users may be able to speak to the best simulation backend for your particular project.

6.2 Complete List of Backends

Qucs-S supports numerous different simulation backends:

6.2.1 Analog/Digital/RF Simulators

- **ngspice (recommended)**: A powerful mixed-level/mixed-signal circuit simulator. Most SPICE models distributed across industry are compatible with it. It has excellent performance for time-domain simulation of switching circuits, and a powerful postprocessor. If you are unsure which simulation backend to use with QUCS-S, ngspice is recommended.

- **Xyce**: A new SPICE-compatible circuit simulator, written from scratch by Sandia National Laboratory. Xyce has the notable advantage of supporting large-scale parallel computing platforms, making it a good fit for solving very large circuits (although it can run on an ordinary desktop platform as well).
- **SpiceOpus**: A free general purpose circuit simulator, based on the Berkeley SPICE-3f5 codebase, specially suited for optimization loops.
- **QucsatorRF**: A fork of Qucsator, the built-in simulation engine from the original Qucs project. QucsatorRF shares the original Qucsator netlist syntax, and all RF features. It's primarily intended for RF simulation with microwave devices and microstrip lines. It is not generally recommended for general-purpose circuit simulation, since ngspice typically has better performance.

6.2.2 Digital-Only Simulators

- **Icarus Verilog**: A digital-only simulation backend for simulating Verilog devices.
- **GHDL**: A digital-only simulation backend for simulating VHDL devices. Fully supports the 1987, 1993, 2002 versions of the IEEE 1076 VHDL standard, and partially the latest 2008 revision (well enough to support fixed_generic_pkg or float_generic_pkg).

> **Note**
>
> A full feature comparison table is on the to-do list for these docs. If you have knowledge on simulation backends and would like to contribute, please see issue #21 on the docs repository.

CHAPTER

SEVEN

SIMULATION TYPES

Qucs-S is capable of many different simulation types. It can simulate traditional analog circuits, in the time domain or the frequency domain. It can sweep various parameters of the circuit to quickly compare multiple simulations. Depending on your *selected simulation backend*, it is also capable of Fourier analysis, and many other specialized simulations.

7.1 Simulation Components

Calling a specific simulation in Qucs-S starts with placing a Simulation Component on your schematic page. Once the component is placed, you can click on it and configure its various properties, just like a standard circuit component. Multiple Simulation Components can be placed on a single schematic page, and Qucs-S will process all the simulations when you click the "Run Simulation" button. An example of a Transient Simulation Component, one of the most common types of analog simulations, is shown in the figure below.

7.2 Learn More

To learn more about Qucs-S's simulation capabilities, please see the specific pages below for common simulation types. Note that this is not exhaustive documentation - it only aims to cover the most frequently used Simulations.

7.2.1 Analog Simulation

Qucs-S can perform a number of different common analog circuit simulation tasks. This page does *not* seek to be a step-by-step tutorial on each simulation type, but rather a brief overview of Qucs-S's capabilities, with links to more in-depth resources.

7.2.1.1 DC Operating Point Simulation

A DC Operating Point simulation is the simplest type of circuit simulation. It does not look at transient/alternating current signals at all, but only considers DC circuit elements. A DC operating point simulation will simply output DC bias voltages for each electrical net in your schematic, and output currents for DC sources in your schematic. This is the same type of simulation as the `.op` command in other SPICE-based simulators, such as LTSpice.

There are two ways to make Qucs-S perform a DC operating point simulation:

1. Place any other type of simulation component on your schematic. A DC operating point simulation will be included automatically.
2. Place the dedicated "DC Simulation" component on your schematic. Use this if a DC simulation is the *only* type of simulation you wish to perform.

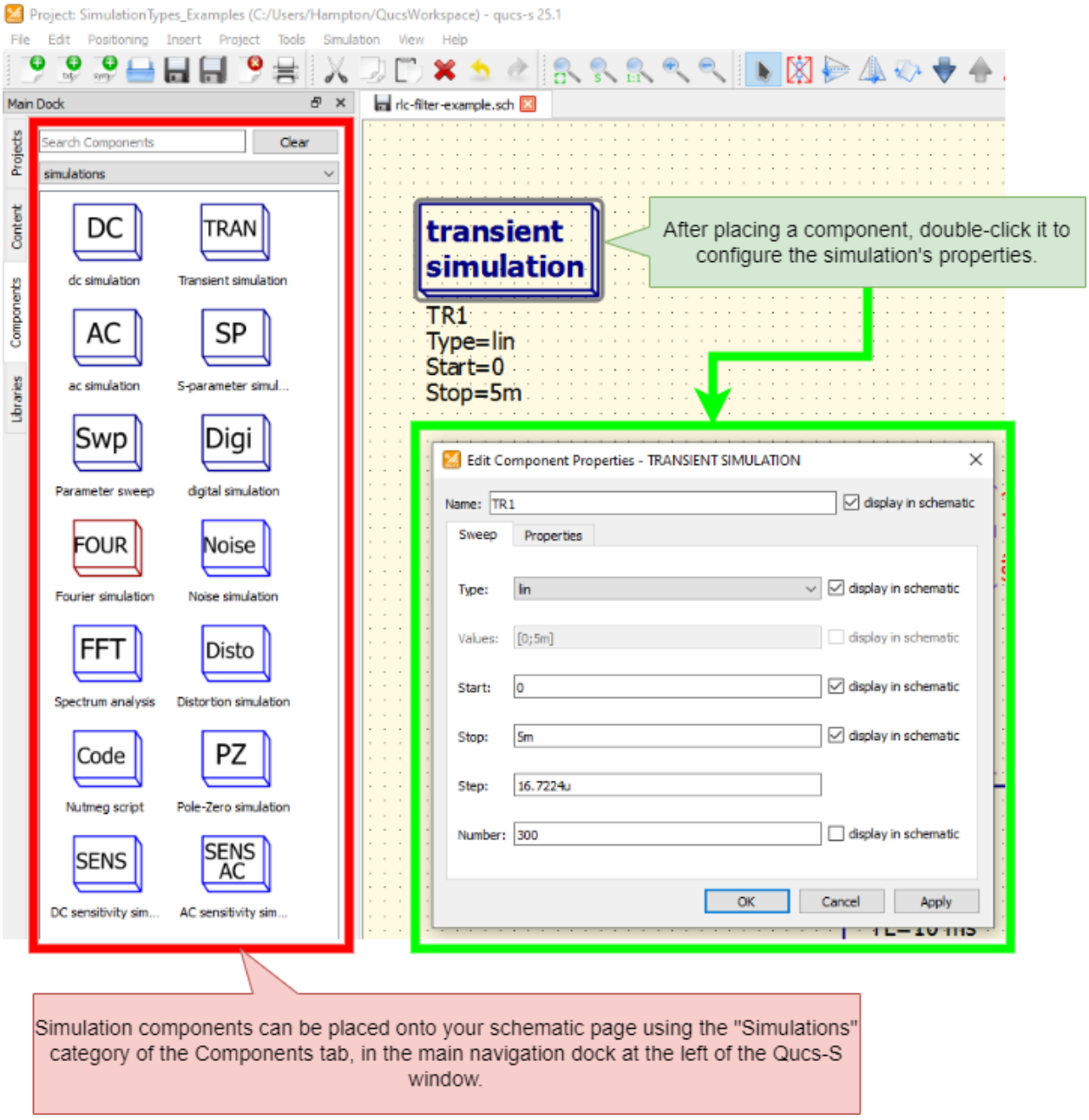


Fig. 1: An example of how to place a Simulation Component in a Qucs-S schematic page, and how to configure the Simulation's parameters.

Method 1: DC Simulation in Addition to Other Simulation

When any other simulation command (such as Transient, AC, etc) is called on your circuit, Qucs-S will *also* perform a DC analysis automatically. To view this DC analysis data, either use the `Simulation > Calculate DC Bias` menu button, or push the `F8` key. This will cause the DC bias voltages and currents to display directly on your schematic page, as in the figure below.

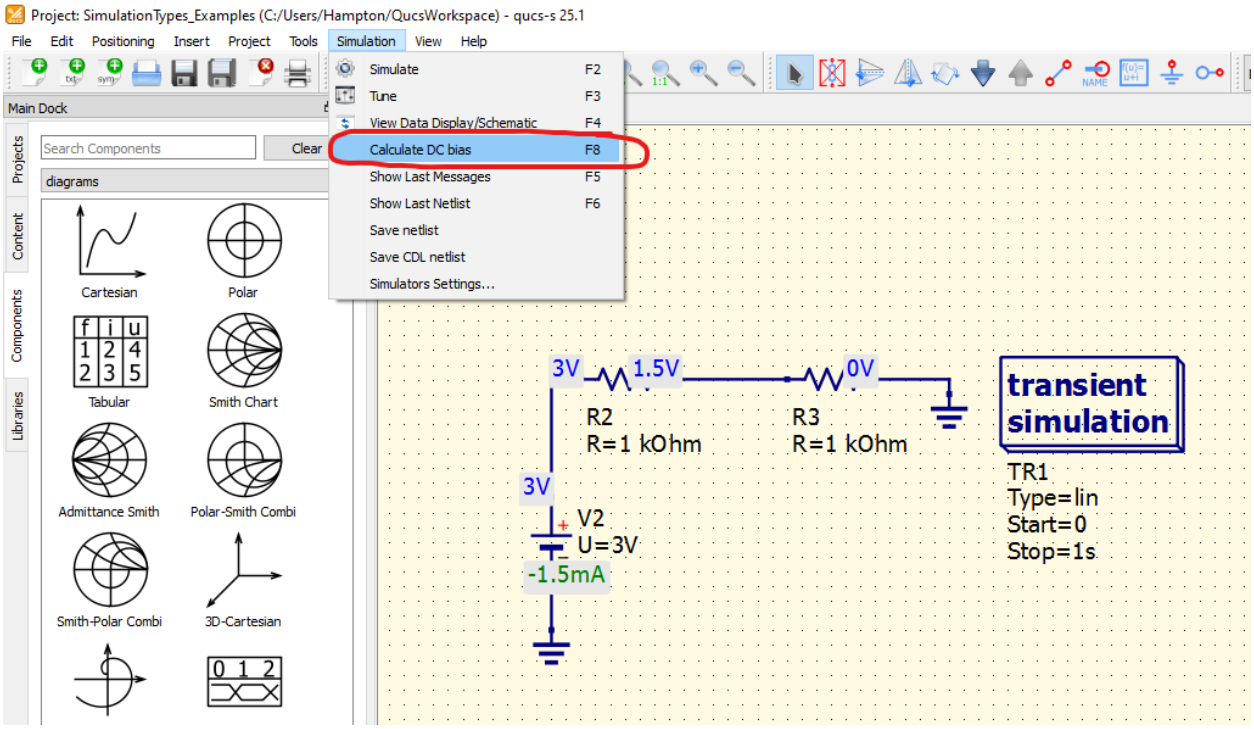


Fig. 2: A simple resistive voltage divider, with a transient analysis being performed, and the DC bias data displayed on the schematic. Note the menu button to turn on the DC bias display on the schematic page.

Method 2: DC Simulation Only

If you *only* wish to perform DC simulation on your circuit, you can place the "DC Simulation" component on your schematic. This will ensure DC simulation occurs when you run the simulation.

You will still need to use the `Simulation > Calculate DC Bias` button, or push `F8`, to display the DC simulation results on your schematic page.

An example of a DC-only simulation is shown below.

7.2.1.2 Transient Simulation

Another very common type of simulation is the Transient simulation. This is a simulation of a dynamic circuit's behavior across a specified amount of time. DC, AC, and totally nonperiodic signals can be modeled in a Transient Simulation.

The example below shows a single square pulse exciting an LC filter, using a 5 millisecond transient simulation.

Transient simulations have 4 major parameters:

- **Sweep Type**: This determines whether the time in the simulation proceeds linearly (`lin`), logarithmically (`log`), or via a user-defined list of discrete time points (`list`).

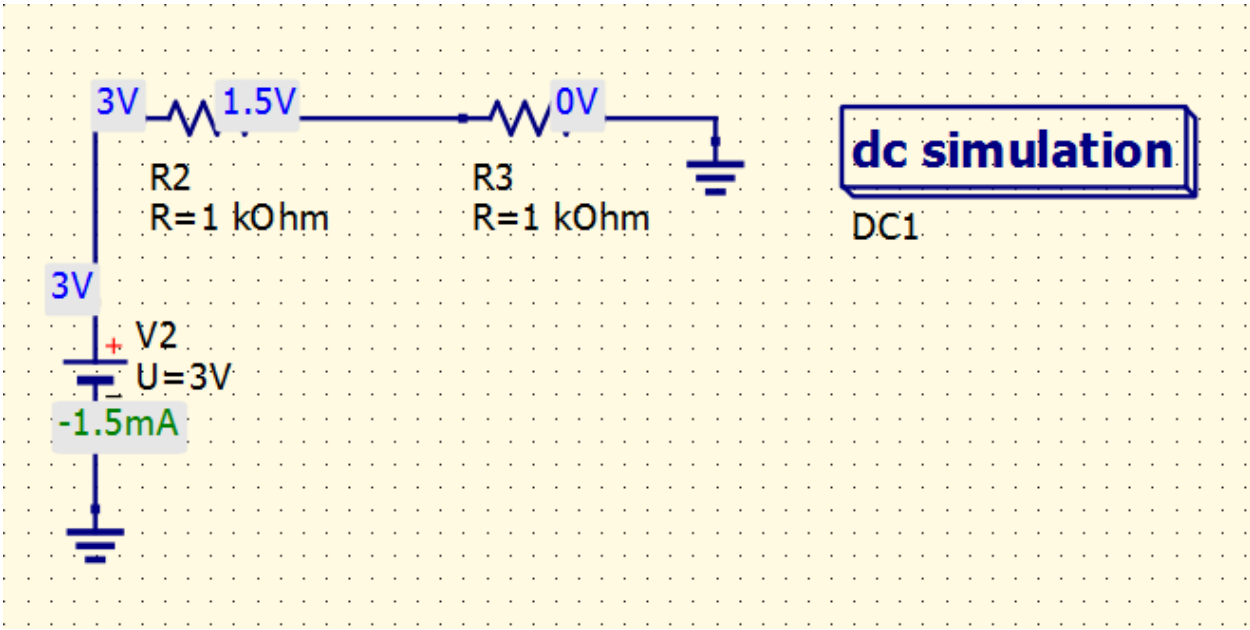


Fig. 3: A simple resistive voltage divider, with a DC analysis being performed, and the DC bias data displayed on the schematic. Recall that you must click `Simulation` > `Calculate DC Bias` or push the `F8` key to display the DC bias data on the schematic.

- **Start Time**: Typically this is left at `0`. You may wish to set this to some offset value to control the initial conditions of your circuit.
- **Stop Time**: Your transient simulation will end at this absolute time. If your Start Time is set to `0`, this represents the duration of your simulation.
- **Step** and **Number**: These control the time-steps used during the simulation. The *Step* parameter tells the length of one simulation step, and the *Number* parameter tells the total number of simulation steps. The two parameters are linked, so changing one will cause the other to recalculate.

The figure below shows the most important parameters for a Transient Simulation. Additional parameters can be configured on the Properties tab, but that is currently outside the scope of this documentation.

7.2.1.3 AC Analysis Simulation

Qucs-S can also perform small-signal AC analysis to characterize a circuit's performance in the frequency domain. This is analagous to the `.ac` command in LTSpice, and other common SPICE circuit simulators.

This simulation sweeps fixed-magnitude sinusoidal source(s) (either voltage or current sources can be used) across a set frequency range, to test your circuit's frequency response.

To do this, place an "AC Simulation" component on your schematic page, and configure the following parameters:

- **Type**: This determines whether the frequency in the simulation proceeds linearly (`lin`), logarithmically (`log`), or via a user-defined list of discrete frequencies (`list`).
- **Start**: Defines the start frequency.
- **Stop**: Defines the stop frequency.
- **Points**: Defines the number of discrete frequency points to simulate.

In addition to placing the AC Simulation component, you must also make sure you have at least one AC source in your schematic. These are the only sources which will be frequency-swept by the AC Simulation - other source types will

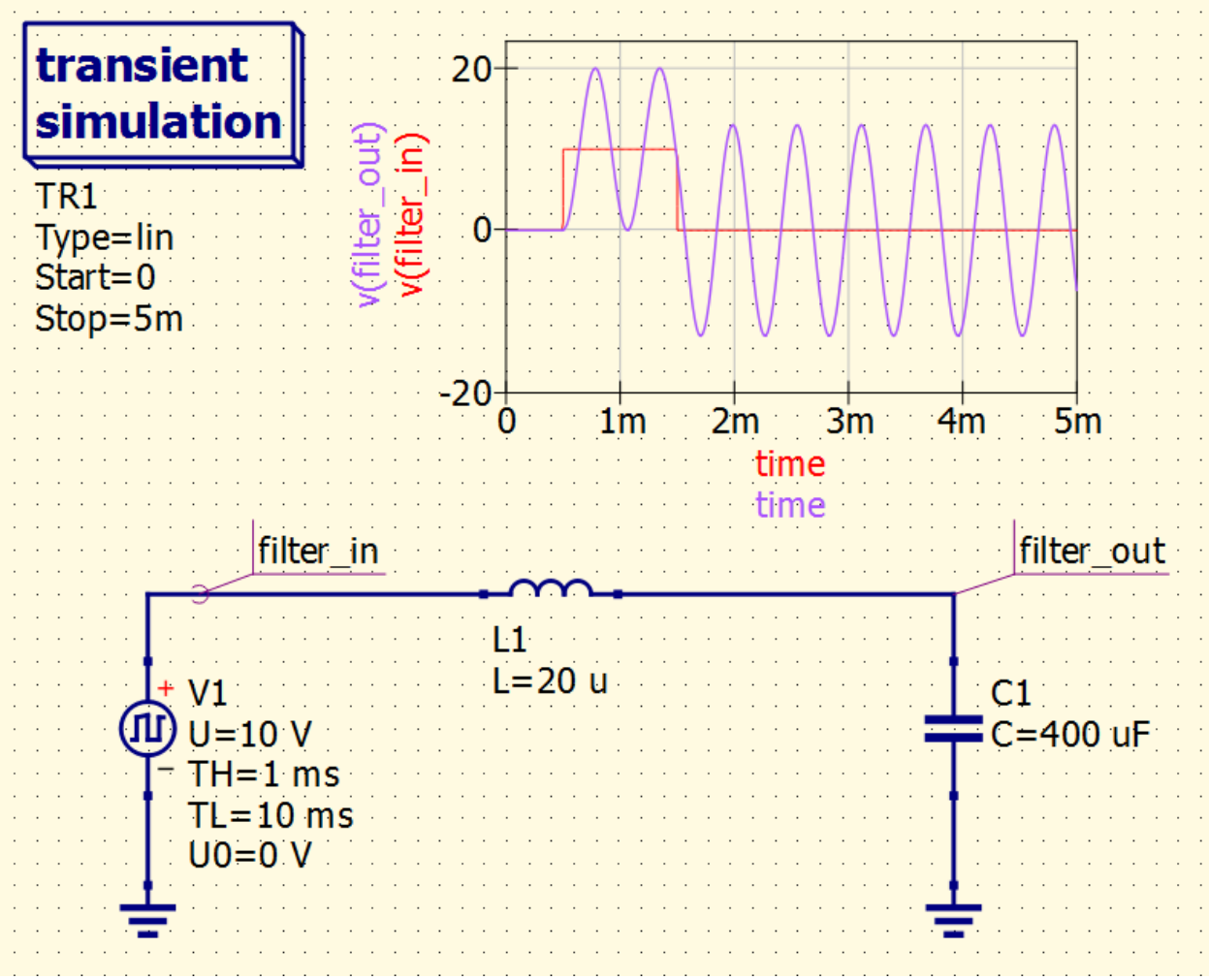


Fig. 4: A simple LC filter, being excited by a single 10V pulse with a duration of 1ms. The input voltage is graphed in red, while the output voltage is graphed in violet. These results came from the Transient Simulation command shown in the figure, which has a duration of 5 milliseconds.

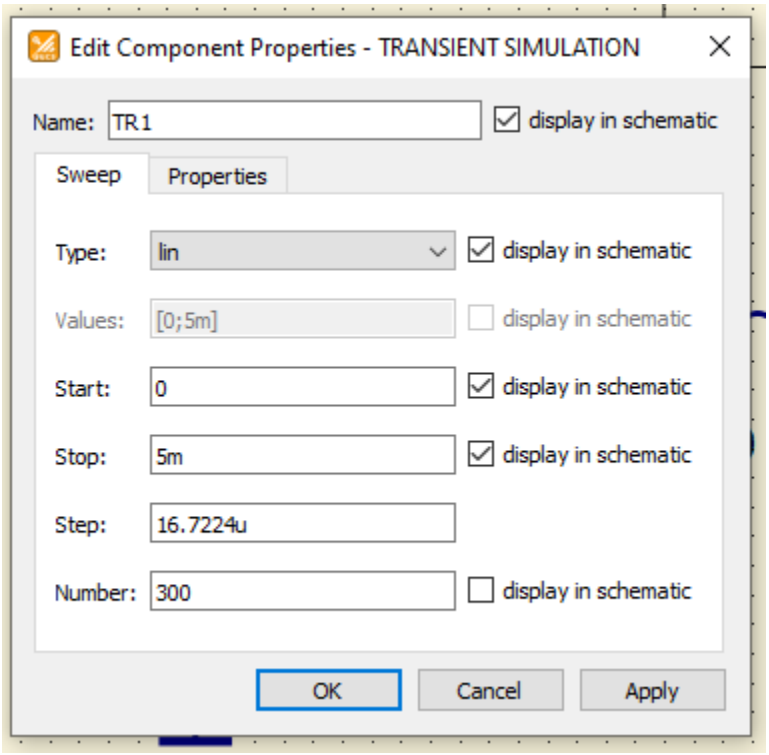


Fig. 5: An example of the major properties in a Transient Simulation.

be unaffected. AC Voltage or AC Current sources are available in Qucs-S, as shown in the figure below.

An example of an AC Simulation is shown in the figure below.

Warning

In the example above, the absolute voltage at the circuit's output is being graphed directly. However, it is more common in AC analysis to graph the ratio of the output to the input, usually in dB. In this example, that would be `dB( v(filter_out)/v(filter_in) )`.

Qucs-S *can* do this, but it requires an additional step of utilizing the Equations feature. Exactly how to do this depends on which simulation backend you are using.

Equations also make it possible to graph in dB, graph the signal's phase in addition to the magnitude, and many other additional features. *See the documentation on Equations for more information.*

7.2.2 Parameter Sweep Simulations

Often in circuit simulation, it is useful to repeat a similar simulation while varying one or more parameters. This makes it easy to see how specific parameters affect your circuit's behavior.

Tip

You may wish to interactively modify circuit parameters using sliders, rather than defining a range ahead of time and performing a traditional sweep. This is possible using the *Tuning Mode* feature!

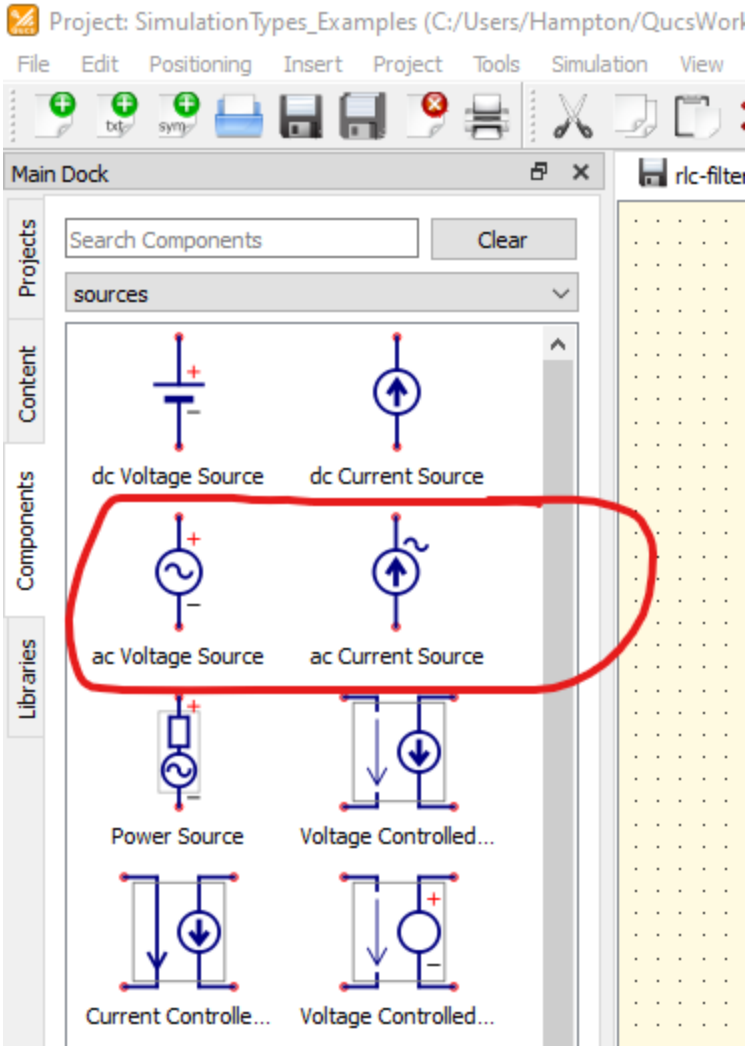


Fig. 6: AC Current and AC Voltage sources, available via the "Sources" category in the "Components" tab, in the Main Navigation Dock. You must have at least one AC Source in your circuit to perform an AC Simulation, since these are the sources that will have their frequencies swept by the AC Simulation.

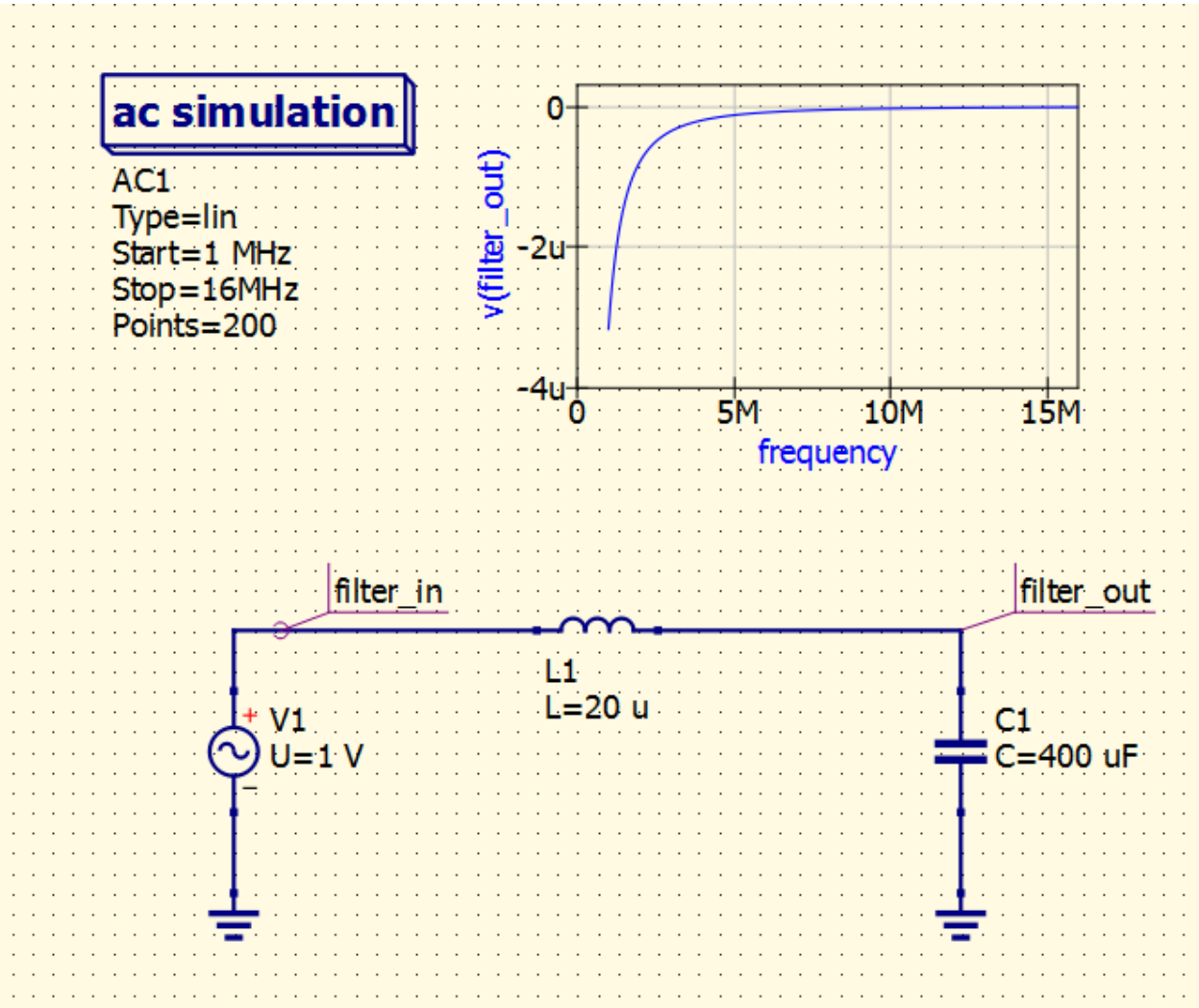


Fig. 7: An example of a simple AC simulation. Note that it is possible to graph magnitude in dB, graph phase, and other more advanced features by utilizing the *Equations feature*.

Qucs-S can support this type of simulation using the *Parameter Sweep* Simulation. This component can sweep the value of a passive component (resistor, inductor, capacitor, or source), or with some additional configuration, can sweep a custom parameter. It accepts an existing Simulation Component as an input, and the Parameter Sweep will repeat that simulation for all the values of the swept parameter.

> **Warning**
>
> **The standard Parameter Sweep Simulation can *only* sweep the values of passive components!** (This includes resistors, capacitors, inductors, and voltage sources.) **On its own, it *cannot* define and sweep an arbitrary variable in your schematic.**
>
> Defining and sweeping arbitrary variables *is possible* for some simulation backends, but requires additional steps. See the section below on *Sweeping Arbitrary Parameters* to learn more.

An example of this Simulation Component is shown in the figure below. It's set up to sweep a resistor `R1`'s value, and repeat the Transient Simulation `TR1` for each iteration of `R1`.

The *Parameter Sweep* Simulation accepts 6 major inputs:

- **Sim**: This is the ID of the simulation which should be run inside the sweep "loop." For example, in the figure above, `TR1` is used, meaning the Parameter Sweep will run a new instance of the `TR1` Transient Simulation for each swept value of Resistor `R1`.
- **Type**: The mathematical distribution of the swept parameter. This can be linear (`lin`), logarithmic (`log`), or a discrete list of values defined by the user (`list`).
- **Param**: The device whose value is to be swept. Typically, this must be a passive component (resistor/capacitor/inductor/source). With additional configuration, some simulation backends may be able to sweep arbitrary variables, rather than being limited to only passive component values. See the section on *Custom Parameter Sweeps* to learn more.
- **Start**: Defines the start point for the swept device or parameter.
- **Stop**: Defines the end point for the swept device or parameter.
- **Points**: Defines the number of points in the sweep (between the Start and Stop values).

7.2.2.1 Sweeping Passive Component Values (Device Sweeps)

The most basic way to utilize the *Parameter Sweep* Simulation Component is to sweep the value of a passive device (resistor, capacitor, inductor, or source). This is possible using just the Simulation Component itself, no additional SPICE declarations or extra steps needed.

Example: Capacitor Charging (Before Sweep)

Before adding a sweep, let's consider an example circuit *without* a sweep. In this circuit, a voltage source is being used to charge a capacitor. At the start of the simulation, the capacitor's voltage is zero. Then, the voltage source turns on and begins charging the capacitor. We can see the capacitor's charge voltage increase on the Cartesian diagram as it charges.

In the next section, we'll look at how a Device Sweep could be performed to give more insight into this circuit.

Example: Capacitor Charging (With Device Sweep Added)

In the previous example, we might wish to compare how the capacitor charges as we vary resistor `R1` (allowing more or less current to flow into the capacitor). This can be done by adding a Parameter Sweep simulation, as shown in the figure below.

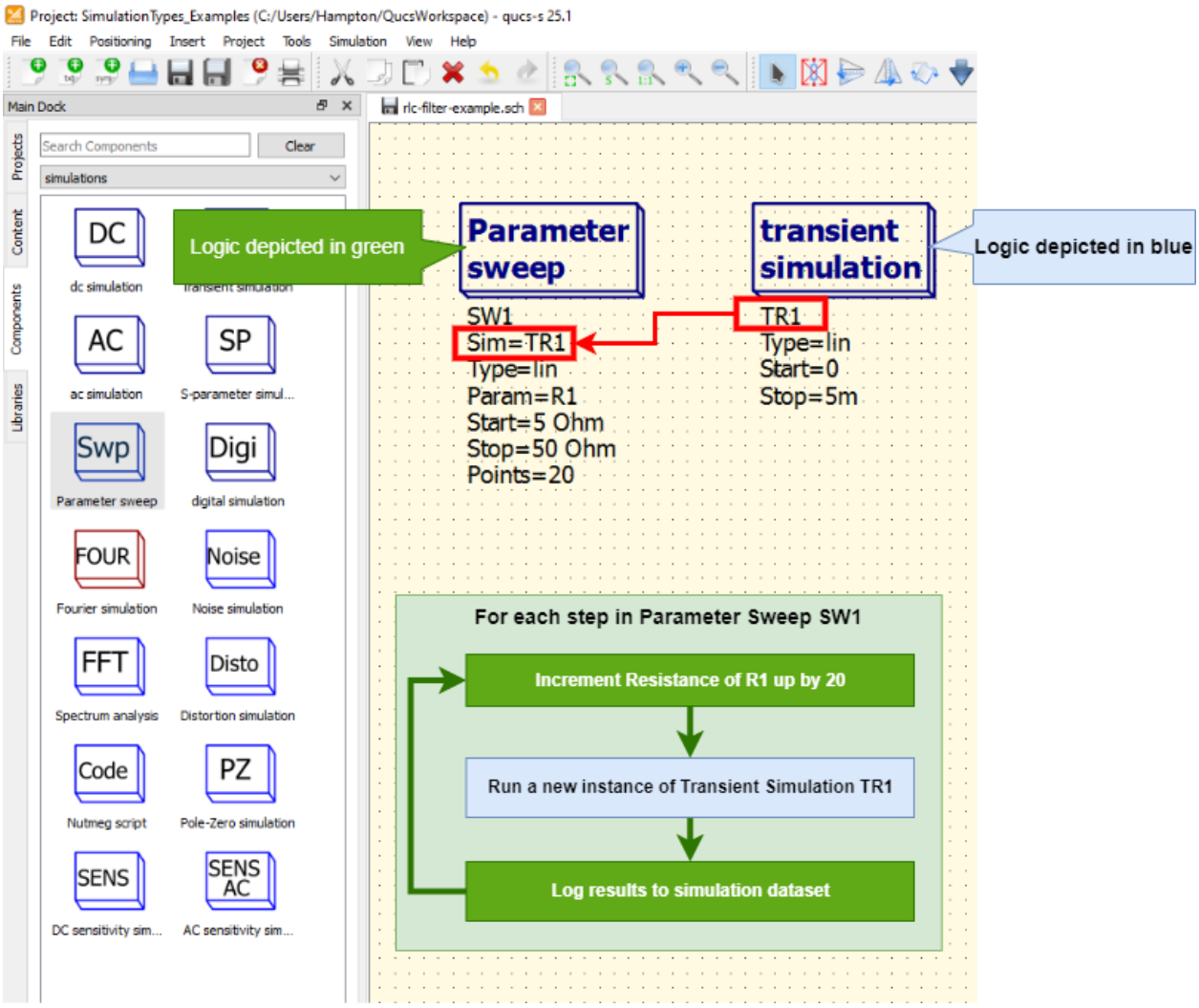


Fig. 8: An example of a Parameter Sweep Simulation Component placed on a schematic page. In this case, it is sweeping the value of a resistor (off-screen) being used in a Transient Simulation (TR1). Note the flowchart showing the Parameter Sweep's logic.

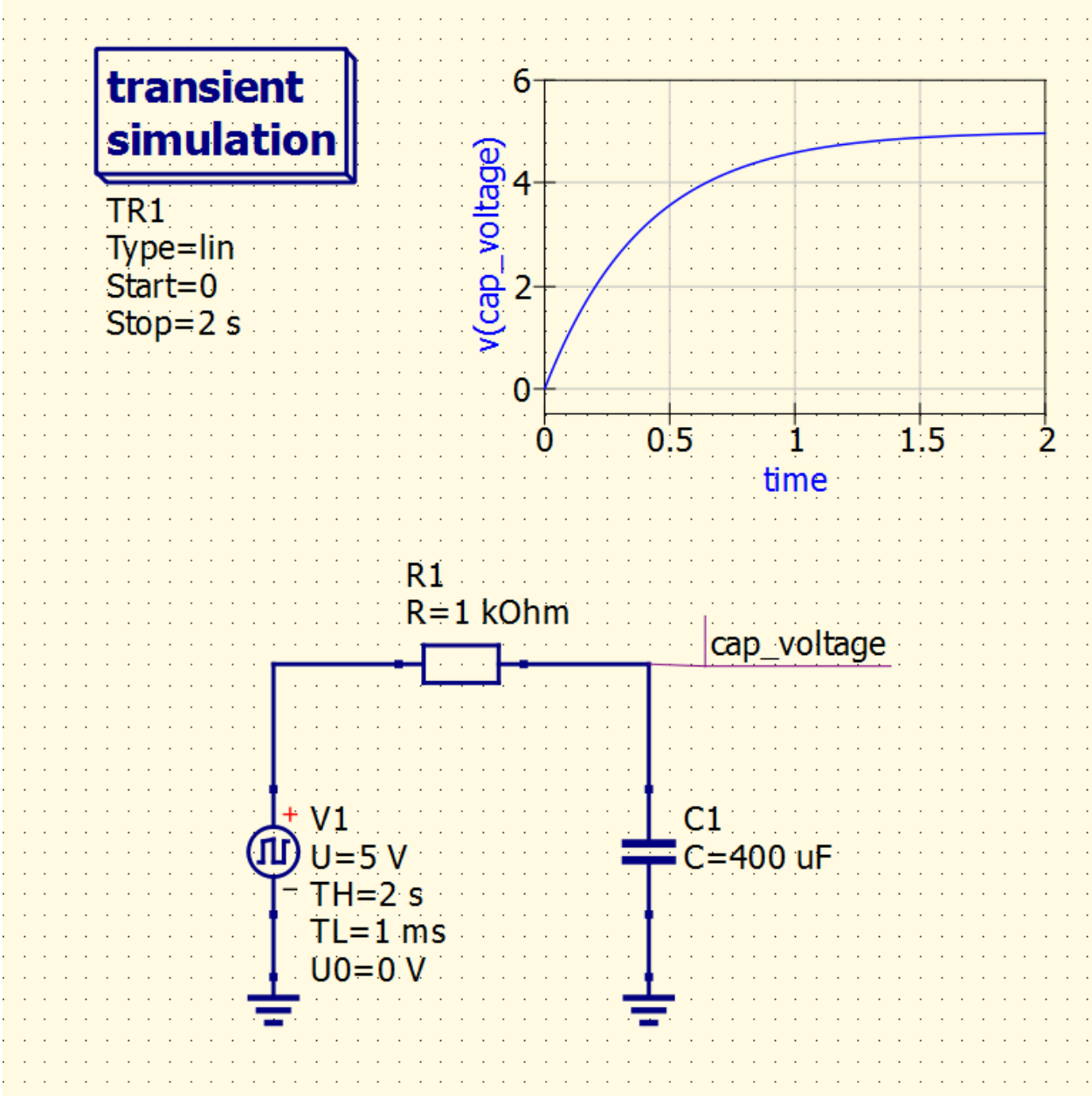


Fig. 9: A simple capacitor, which is being charged through a resistor. Using a Transient simulation, we can see the capacitor's voltage increase over time, as it is charged by the voltage source.

Warning

Note that the Parameter Sweep should be added *in addition to* the existing simulation. Do not delete or deactivate your original simulation component when you add a Sweep. You must retain your original "unswept" simulation (regardless of what type of simulation it may be), since the Parameter Sweep will reference it.

In this new example, 4 Transient Simulations are run, with 4 different values of resistor R1 (200 Ohm, 800 Ohm, 1.4 kOhm, and 2 kOhm). As a result, the 4 distinct capacitor charging curves can be seen on the Cartesian diagram. No modification was required to the diagram's properties compared to the previous example - the complete set of curves was graphed automatically.

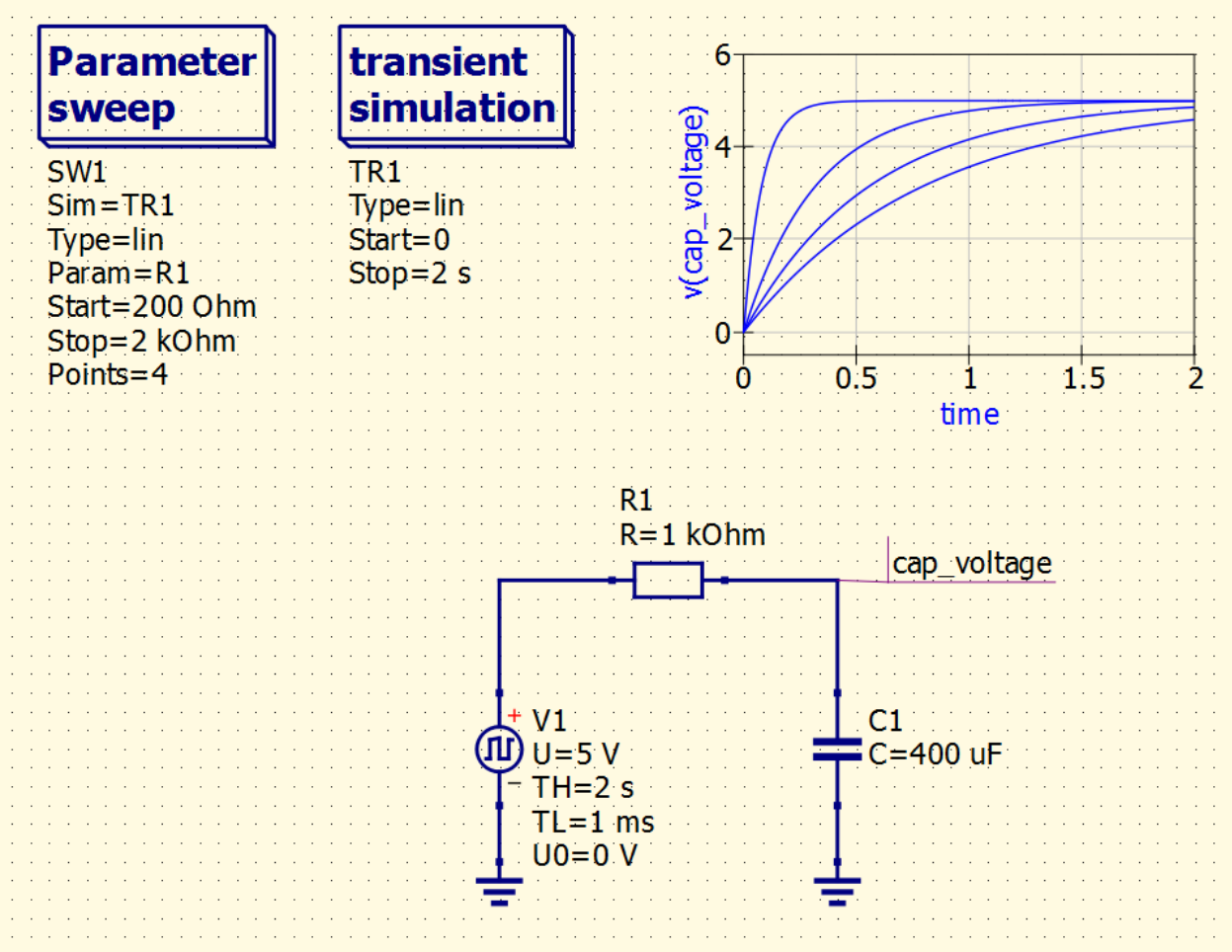


Fig. 10: By adding a Parameter Sweep simulation component to the previous example, we can now observe the charging behavior of the capacitor as resistor R1 is varied from 200 Ohms to 2 kOhms, in 600 Ohm increments.

Tip

The value of a voltage source, current source, inductor, or capacitor could also be swept, in the same way as the resistor in the above example.

DC Sweeps

In Qucs-S, a DC Sweep refers to a Parameter Sweep combined with a DC Simulation. This is often useful for determining how a resistor or power supply voltage affects DC bias points.

For example, the circuit below is a simple resistive voltage divider connected to a DC voltage source. A standard DC Simulation can be used to calculate the output voltage of the divider. However, in this example, a Parameter Sweep has been added, sweeping the resistor `R1` from 1 kOhm to 4 kOhm.

The resulting output voltage for each value of `R1` is displayed in a Table Diagram (although it could just as easily be graphed).

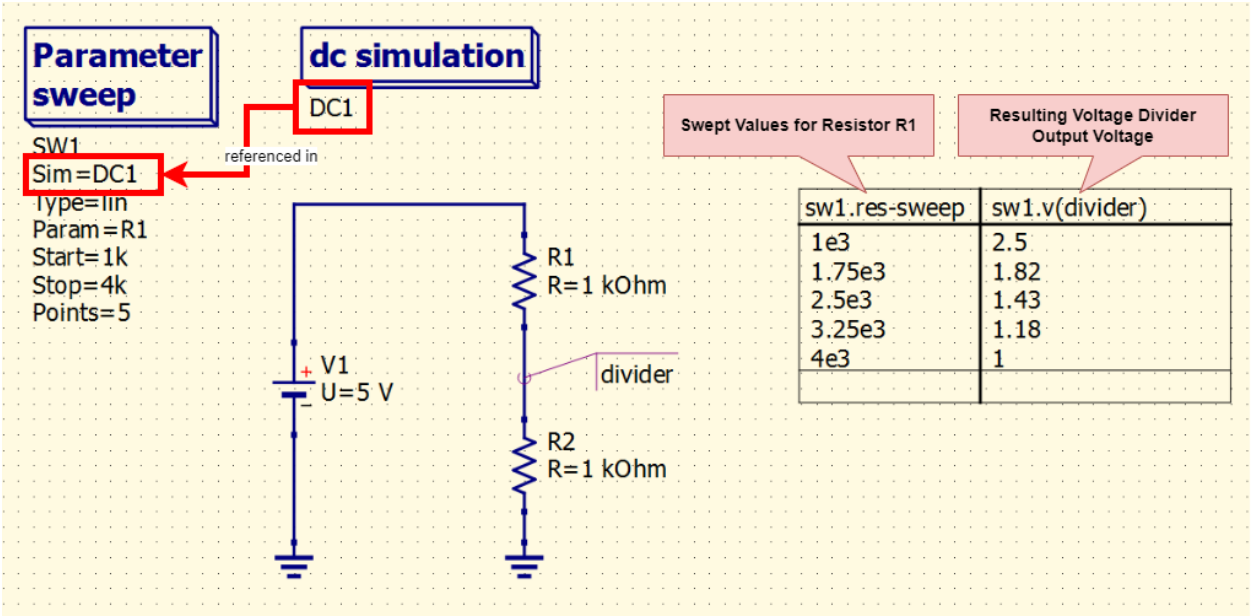


Fig. 11: An example of a "DC Sweep", which combines the Parameter Sweep Simulation with a DC Simulation. In this case, a resistor in a simple voltage divider is swept, and the resulting output voltages are shown in a Table.

> **Warning**
>
> **Only resistors and sources can be swept as part of a DC Sweep.**
>
> Inductors and capacitors are not considered in DC sweeps (inductors become short-circuits while capacitors become open-circuits). Therefore, these components cannot be swept.
>
> **Sweeping custom parameters in a DC Simulation is *NOT* possible, regardless of simulation backend!** Even the `.PARAM` feature in ngspice (see next section) does not apply to this case - it is simply not possible to sweep anything except a resistor or a source when performing a DC Simulation.

7.2.2.2 Sweeping Arbitrary Parameters

In the sections above, only passive components and sources were swept. However, when using the *ngspice Simulation Backend*, it is possible to use a SPICE `.PARAM` command to define and sweep *any* arbitrary parameter within a circuit. When using this method, you are no longer limited to just passive components and sources.

> **Warning**
>
> **This section only applies to recent versions of ngspice! ngspice prior to version 30 (released in 2018) may not support this feature.** This tactic also does not apply to other available simulation backends.

To do this, a `.PARAM` directive must be added to the SPICE netlist. To do this in Qucs-S, visit the "SPICE netlist sections" category of the Components tab. Add a *.PARAM Section* component to your schematic page (circled in the figure below).

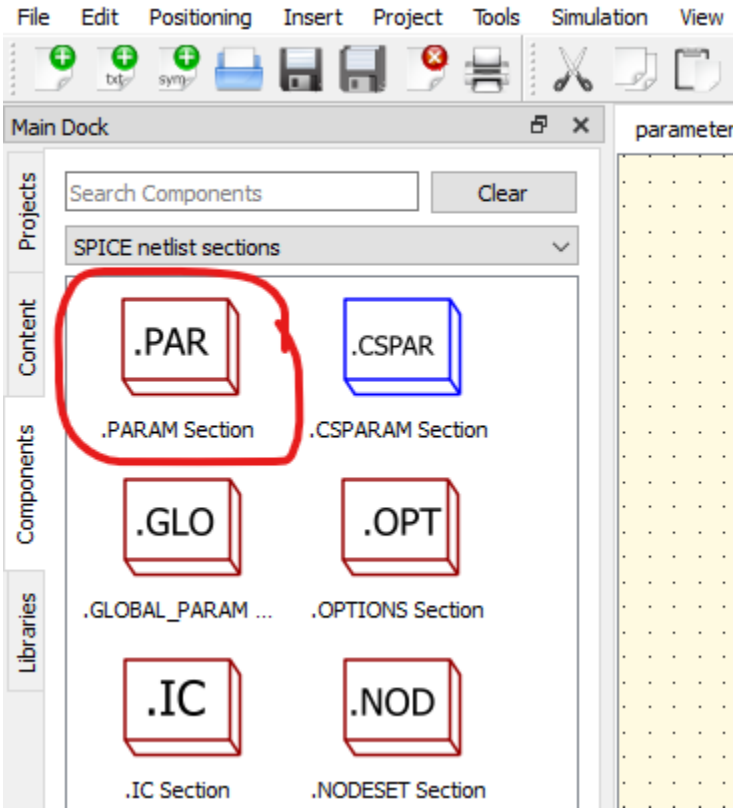


Fig. 12: To add a `.PARAM` directive to your SPICE netlist, place a *.PARAM Section* component on your schematic page.

Example: Sweeping Filter Cutoff Frequency

For example, the circuit below is a simple RC filter, being fed with an AC voltage source. It has an AC Simulation, to characterize the frequency response of the filter.

Recall that for an RC filter, the "-3dB cutoff frequency" can be calculated with $f_{cutoff} = \frac{1}{2\pi RC}$.

It's possible to calculate the necessary capacitor for a given cutoff frequency by rearranging the equation, yielding $C = \frac{1}{2\pi R f_{cutoff}}$.

For the sake of example, let's utilize the `.PARAM` feature to define `C1`'s capacitance in terms of the filter's cutoff frequency (f_{cutoff}), and then sweep f_{cutoff} with a Parameter Sweep simulation. This will show the frequency response of RC filters across a number of cutoff frequencies.

To do this, we'll perform the following modifications to the circuit:

1. **Add a *.PARAM Section* to the schematic, and define relevant variables.** You can define as many variables

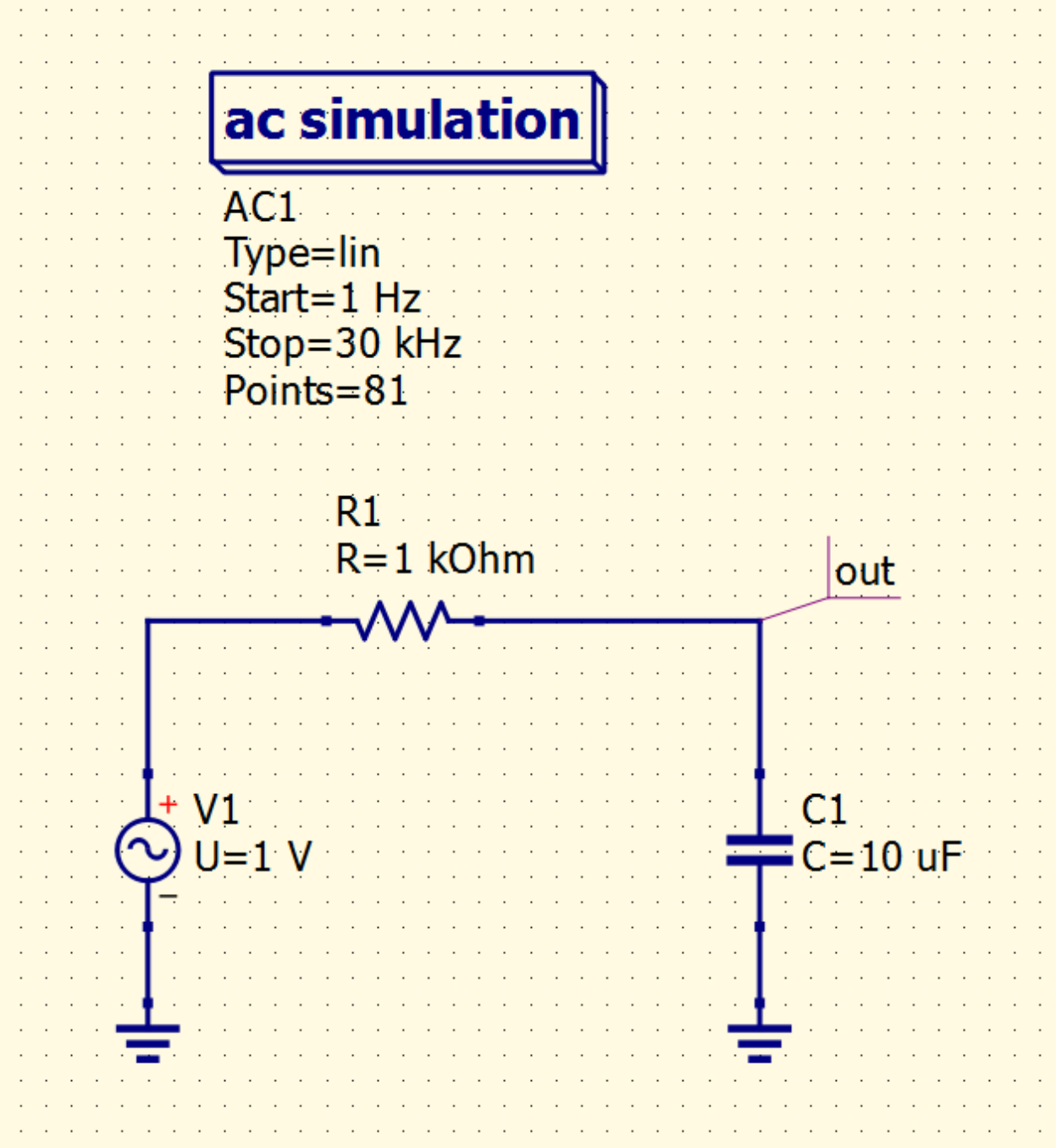


Fig. 13: Simple RC filter example, with an AC Simulation command.

as you like in a single *.PARAM Section* component - you do not need to add multiple components to the page to define multiple variables.

2. **Modify the circuit to reference our new parameters.** In this case, we will modify `R1` and `C1`. Be sure to encapsulate with `{` and `}` so the parameters and math is parsed properly by the simulator!
3. **Add a Parameter Sweep component, and select the variable defined in the *.PARAM Section* as the sweep parameter.**

> **Tip**
>
> There is no built-in Pi constant in ngspice. To easily utilize Pi (π) in your schematics, put the following into a *.PARAM Section*: `pi=4*atan(1)`
>
> This allows you to use `pi` in your schematics.

> **Warning**
>
> You must enclose parameters and math operations in curly braces, or ngspice will not parse them correctly!

After the modifications, and the addition of a Cartesian diagram to graph the results, the circuit looks like this:

7.2.2.3 Double Sweeps (aka Nested Sweeps)

You might wish to sweep *two* parameters instead of just one. This is possible by nesting multiple Parameter Sweeps. Set the first Parameter Sweep up with the `Sim` property pointing to a normal simulation (for example, a transient simulation). Then, set the second Parameter Sweep up with the `Sim` property pointing to *the first parameter sweep*. This allows you to sweep multiple parameters, and see how the circuit's performance varies.

For example, if you want to graph the V_{CE} vs I_c curves of a transistor, you'll need to sweep *two* parameters. First, you need to sweep the collector-emitter voltage (V_{CE}), and then repeat that sweep for a different value of base current - rinse and repeat. This is a great use case for "nested sweeps" in Qucs-S.

An example of how to do this is shown in the figure below. Note that while Parameter Sweep `SW1` is sweeping the DC Simulation, Parameter Sweep `SW2` is sweeping `SW1`!

7.2.3 Simulating in Tuning Mode

Sometimes, you may wish to get an intuitive sense of a circuit's behavior by varying one component's value and seeing results in real-time, rather than performing a litany of *Sweeps* or another more traditional circuit simulation method. That is what Qucs-S *Tuning Mode* is for!

Tuning mode allows you to vary the value of a component property using sliders, and see updated simulation results instantly. This is particularly useful if you are trying to gauge the effects of a filtering component, tune the length of a transmission line, or some other intuitive, optimization-related task.

7.2.3.1 Using Tuning Mode

To use Tuning Mode:

1. **Ensure your schematic is set up with at least one Simulation Component (DC, Transient, AC, etc).** Tuning Mode does not run any of its own simulation logic, it simply calls the Simulation Components already on your schematic page when you vary the tuned parameter.
2. **Click the "Tuning Mode" button in the simulation toolbar.** It's immediately to the right of the Simulation Backend Selection dropdown and the Run Simulation button.

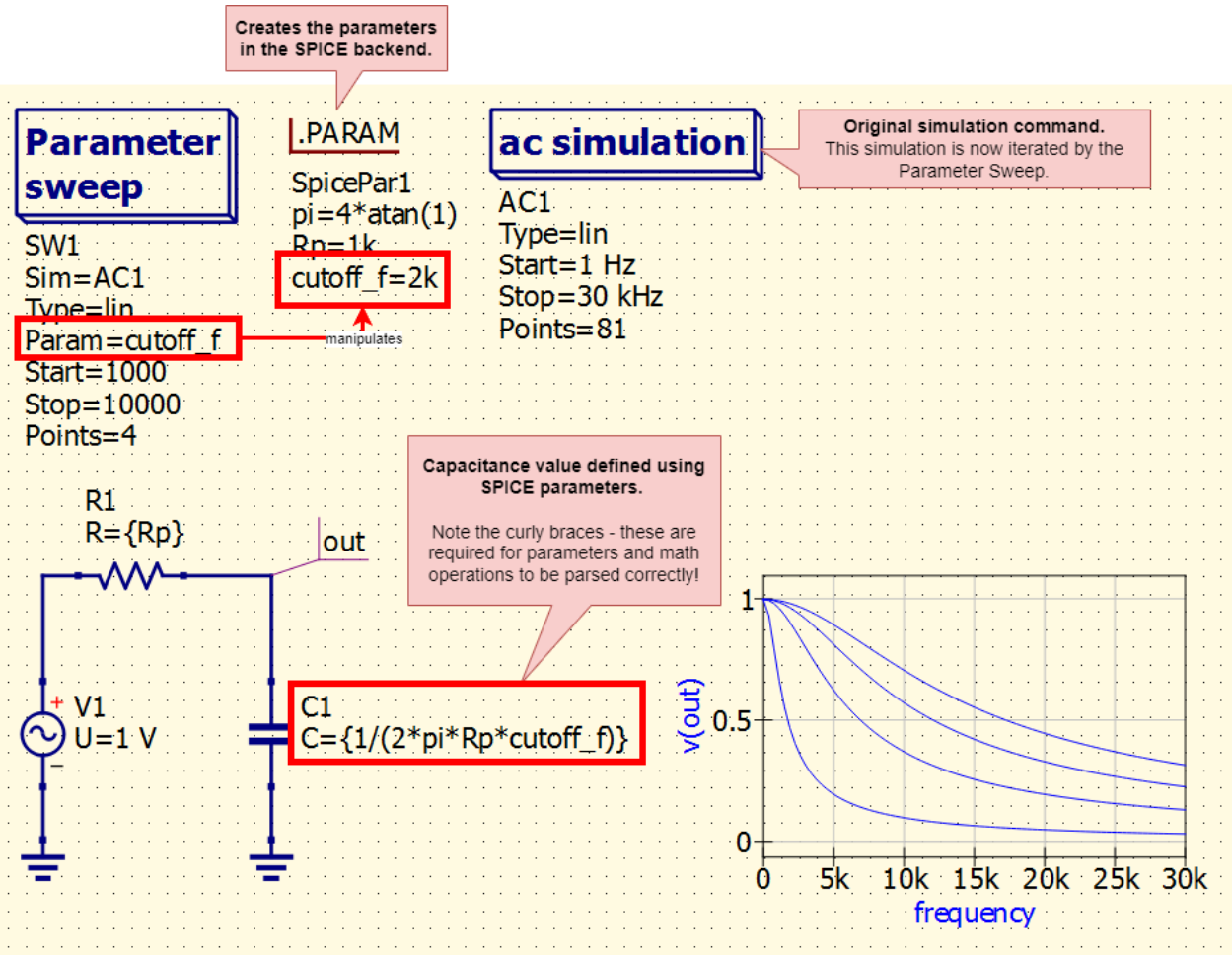


Fig. 14: RC filter example, with capacitance defined in terms of cutoff frequency. The cutoff frequency is parameterized (`f_cutoff`), and swept using a Parameter Sweep. This allows simulation of filters designed for varying cutoff frequencies.

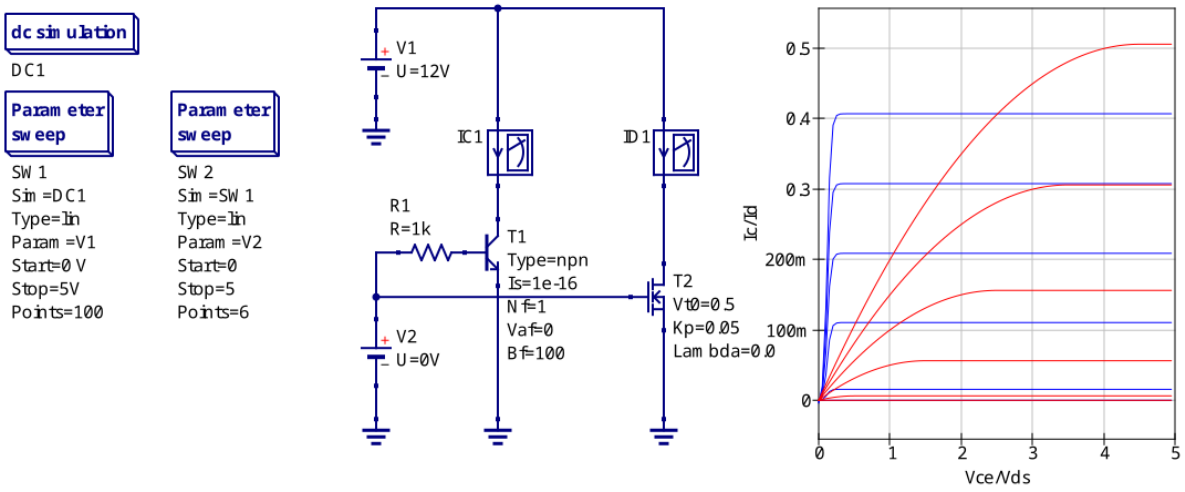


Fig. 15: Example of a double/nested sweep, plotting the V_{CE} vs I_c curves of a transistor.

3. **Select properties in your schematic to tune.** You can select as many properties as you like. You can also define the range/scale of the tuning sliders.
4. **Adjust the slider(s), and watch your simulation results recompute in (near) real-time.** Note that Tuning Mode will recompute *every* Simulation Component on your schematic page. If you are experiencing performance issues, try deleting or deactivating some of the Simulation Components to reduce the compute time.

> **Tip**
>
> For a demonstration of Tuning Mode, see this YouTube video:
>
> `https://youtu.be/mJrAN2WxNoM`

7.2.4 Digital Simulation

> **Warning**
>
> This section of the documentation would benefit from additional detail. Contributions are welcome! Please see this GitHub issue if you wish to contribute.

Qucs-S supports a number of digital devices, which can be found in the *Digital Components* component group. This includes devices such as:

- Logic gates
- Inverters
- Buffers
- Flip Flops (D-, JK-, and T-)
- Digital Sources

These devices can be used in two ways:

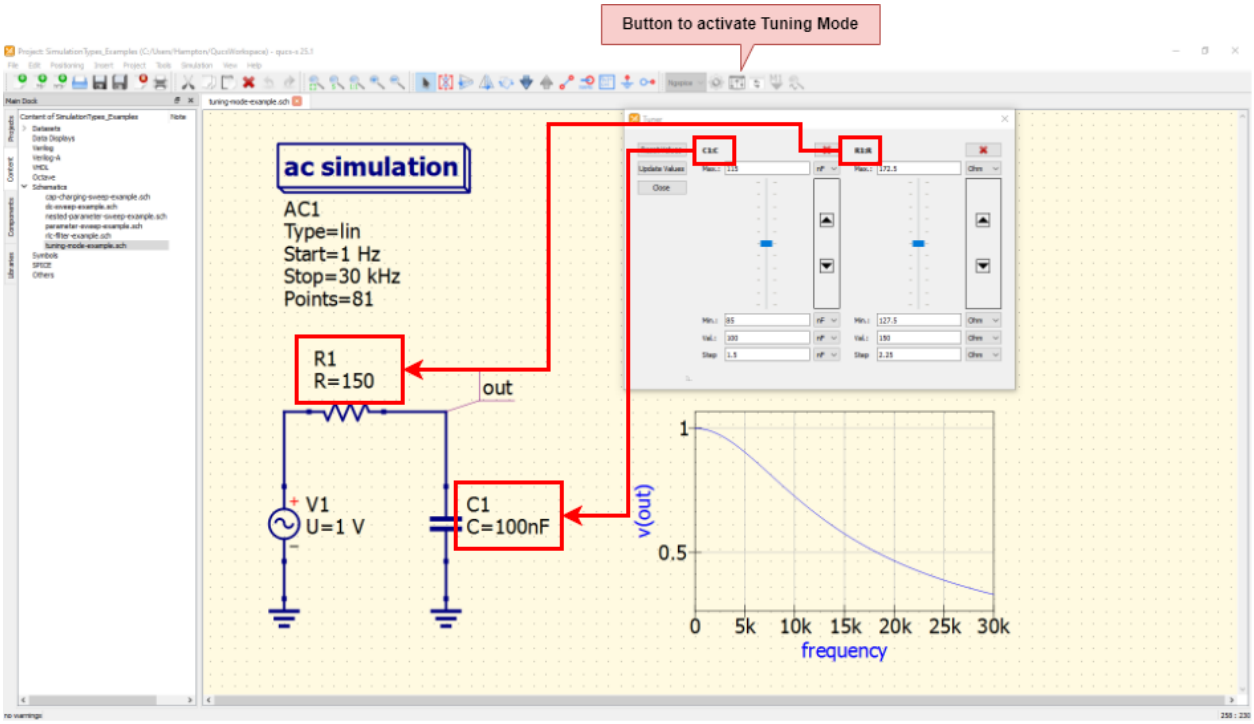


Fig. 16: Example of how to use Tuning Mode. In this case, a simple RC filter is being tuned.

1. As simplified, ideal digital devices in standard analog circuits. We'll call this "Mixed-Signal Digital Simulation."
2. As digital devices in a pure-digital simulation environment (using Verilog or VHDL backends). We'll call this "Pure-Digital Simulation."

7.2.4.1 Mixed-Signal Digital Simulation

It's possible to add devices from the *Digital Components* group to standard analog circuit. In the case of the ngspice simulation backend, these devices operate using XSPICE models. This is a convenient way to simulate ideal digital devices, without having to obtain SPICE models for a specific real-world logic gate IC.

Warning

In addition to placing digital components on your schematic, you must also set the logic voltage for the whole circuit, using the SPICE parameter `vcc`. (Use the .PARAM block from the *Spice Netlist Sections* component group to do this.)

This *also* means that all digital devices in your circuit *must* share the same logic voltage! You cannot utilize this simulation technique with multiple logic power busses, or in a "level shifting" situation.

7.2.4.2 Pure-Digital Simulation

Qucs-S also has the ability to utilize entirely-digital simulation backends (VHDL or Verilog). This feature does NOT facilitate mixing analog and digital components into a single circuit.

Warning

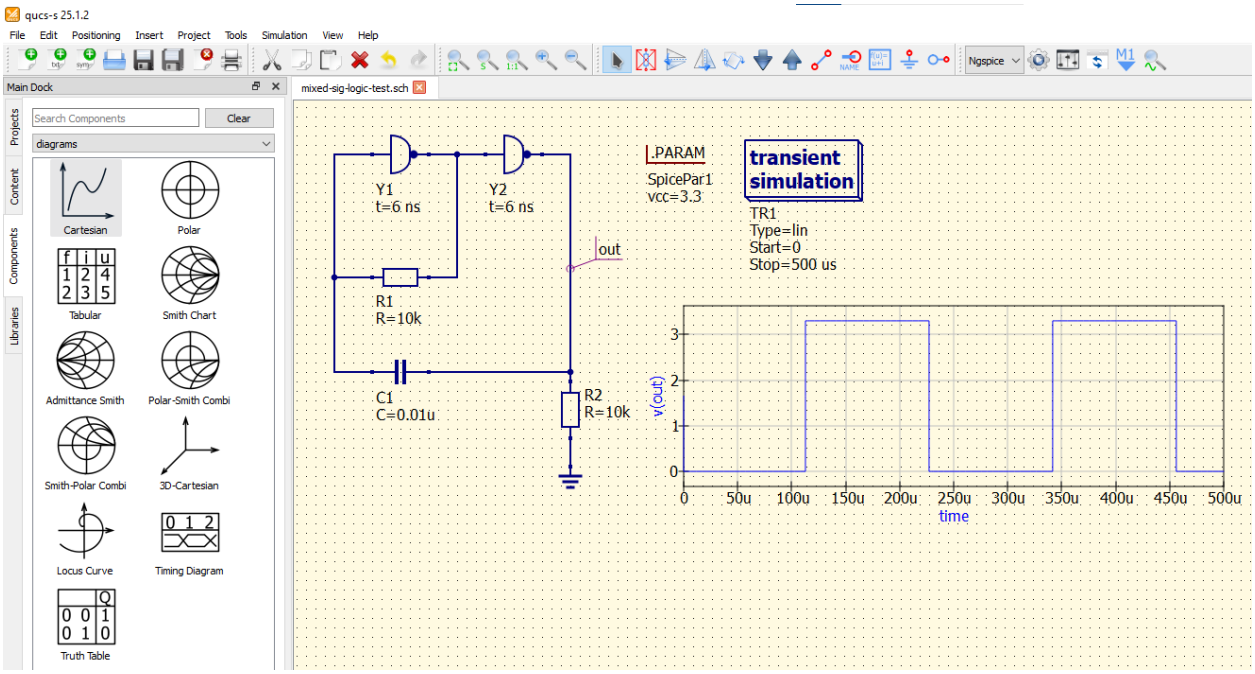


Fig. 17: In this example, an oscillator is being simulated using inverters. The inverters have their delay time set at 6ns, which contributes to the oscillator behavior. This simulation is done in Transient mode, with the ngspice simulation backend (although it may work in other backends as well).

To perform these simulations, you must install the `iverilog` or `ghdl` backend. See *Installing Simulation Backends* for more information.

To start a digital simulation, place the *Digital Simulation* component on your schematic page. This block has 3 parameters:

- **Simulation Type**: The type of simulation to run. Choose Truth Table or TimeList (timing diagram).
- **Time**: The duration of the simulation.
- **Model**: The simulation backend/netlist format to use (Verilog or VHDL).

Note

If the *Digital Simulation* block is placed on your schematic page, Qucs-S will run the digital simulation regardless of which of the simulation backends is selected at the top of the window. In other words, you do NOT need to select GHDL or IVerilog in the same place where you would select ngspice, Xyce, etc. Simply place the *Digital Simulation* block on your schematic page.

The example below shows a Digital Simulation in TimeList (Timing Diagram) mode, simulating a 2-bit decoder/demultiplexer.

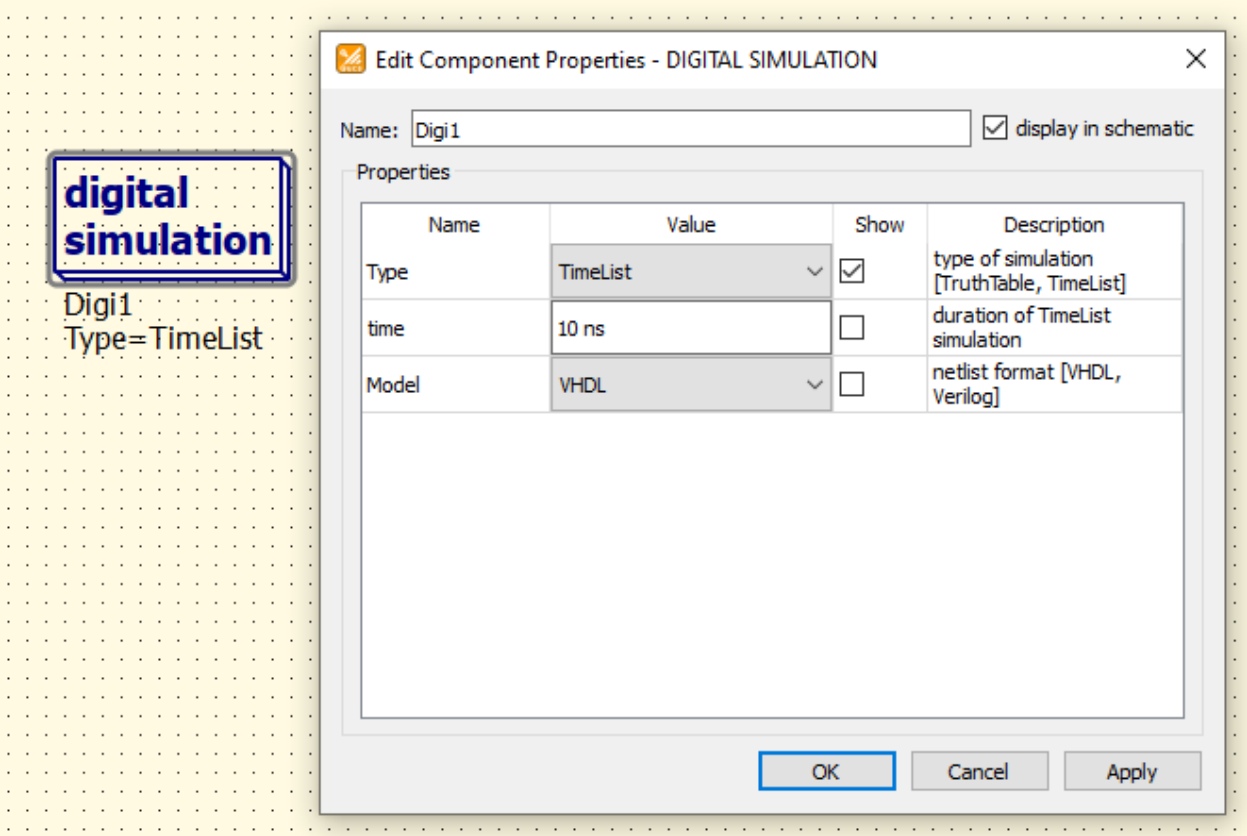


Fig. 18: The *Digital Simulation* block, and the configurable parameters in its Properties.

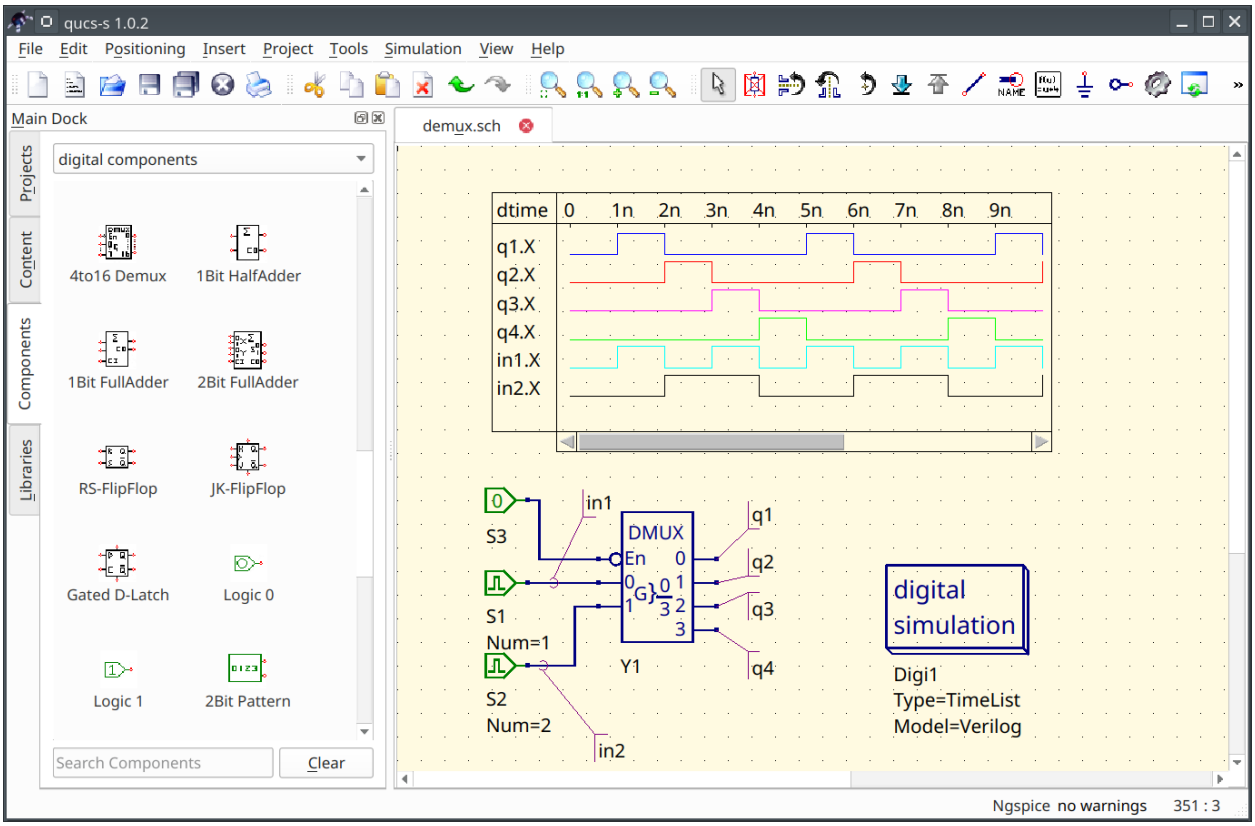


Fig. 19: Simulation of a 2-bit decoder/demultiplexer, in the TimeList (timing diagram) mode, using the Verilog backend.

7.2.5 RF Simulation

> **Warning**
>
> This page could still use more content! Contributions are welcome.
>
> If you are willing to contribute documentation, please see GitHub issue #23.

Qucs-S also supports a number of simulation devices and techniques that are tailored to RF and microwave (high-frequency) simulation. The specific features available depend on which simulation backend you have selected - see the sections below for more details.

Qucs-S also provides a number of diagrams which are especially useful for microwave and RF circuits. These include:

- Smith Chart
- Admittance Smith Chart
- Polar Complex Plane
- Combined Smith Chart and Complex Plane

7.2.5.1 S-Parameter Simulation with ngspice

> **Warning**
>
> ngspice introduced S-Parameter Simulation in Version 37 (released mid-2022). Make sure your ngspice backend is Version 37 or later, or you will not be able to take advantage of these features!

S-Parameters (Scattering Parameters) are a commonly used tool in high-frequency circuit analysis. For a brief introduction to S-Parameters, see this video (no affiliation to the Qucs-S project)..

Setting up the circuit

To use S-Parameter simulation, you must place two special components on your schematic:

- **The "S-Parameter Simulation" simulation component.** This tells Qucs-S that you wish to perform an S-Parameter simulation. Note that this is the same component used in S-Parameter simulations with Qucsator, although that is where the similarities end.
- **The "Power Source" component.** This is the special RF source that will be used to test and obtain the S-Parameters of your circuit.

Examples of each component, and its properties, are shown in the figures below.

As an example, we'll simulate the frequency response of a bandpass filter for the 20-meter band (part of the amateur radio spectrum). The figure below shows the circuit, and the resulting output graphs.

The filter itself is made up of LC devices, and the schematic also contains two *Power Source* components to stimulate the filter. One is connected to the input node (labeled `in`), and the other is connected to the output node (labeled `out`).

Finally, the *S-Parameter Simulation* component defines the frequency sweep range: 6MHz to 20MHz.

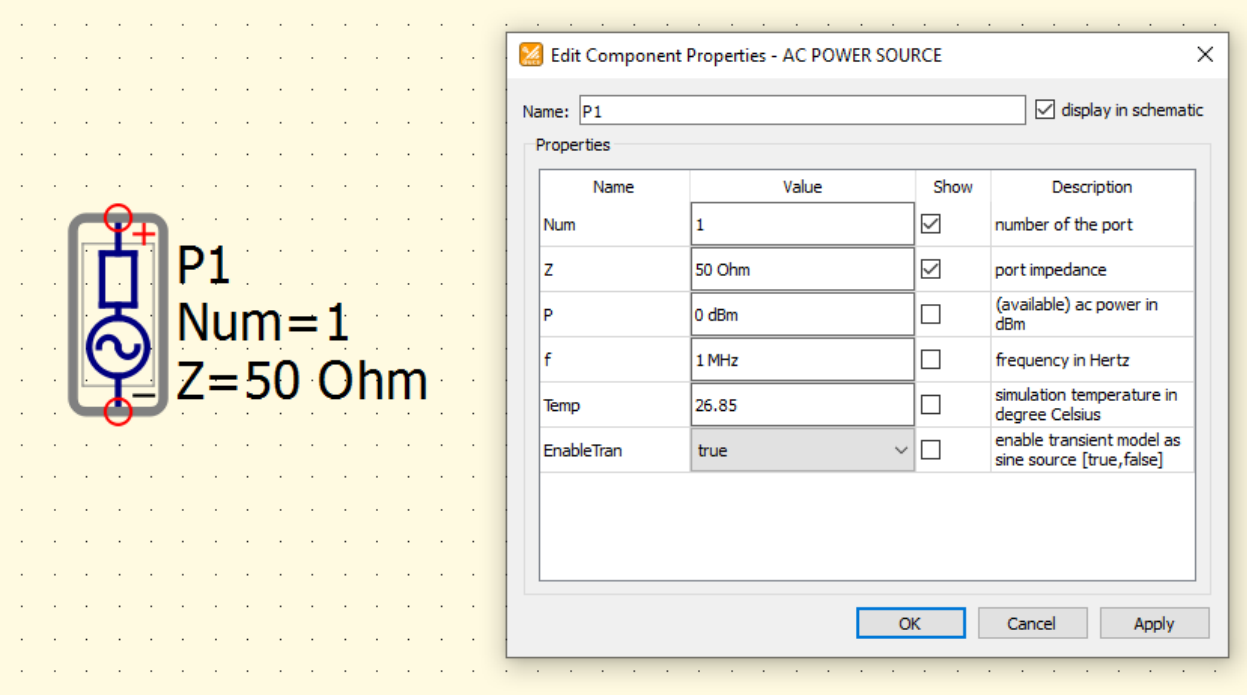


Fig. 20: Example of the *Power Source* component, for use with ngspice when performing S-Parameter Simulations.

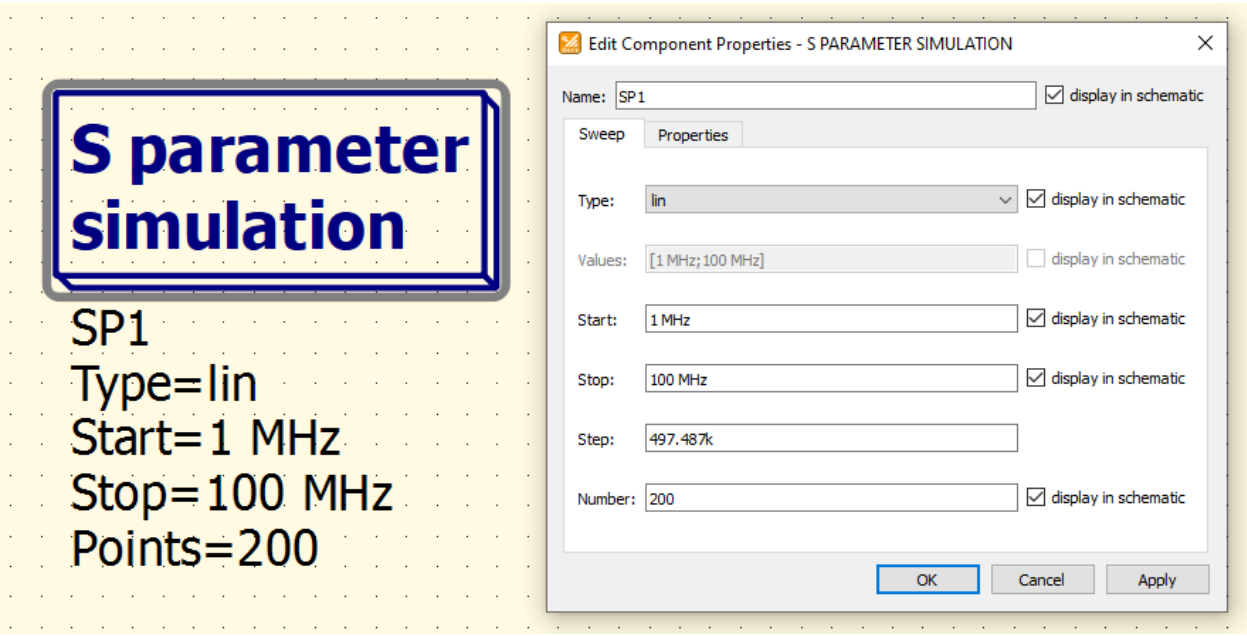


Fig. 21: Example of the *S-Parameter Simulation* component.

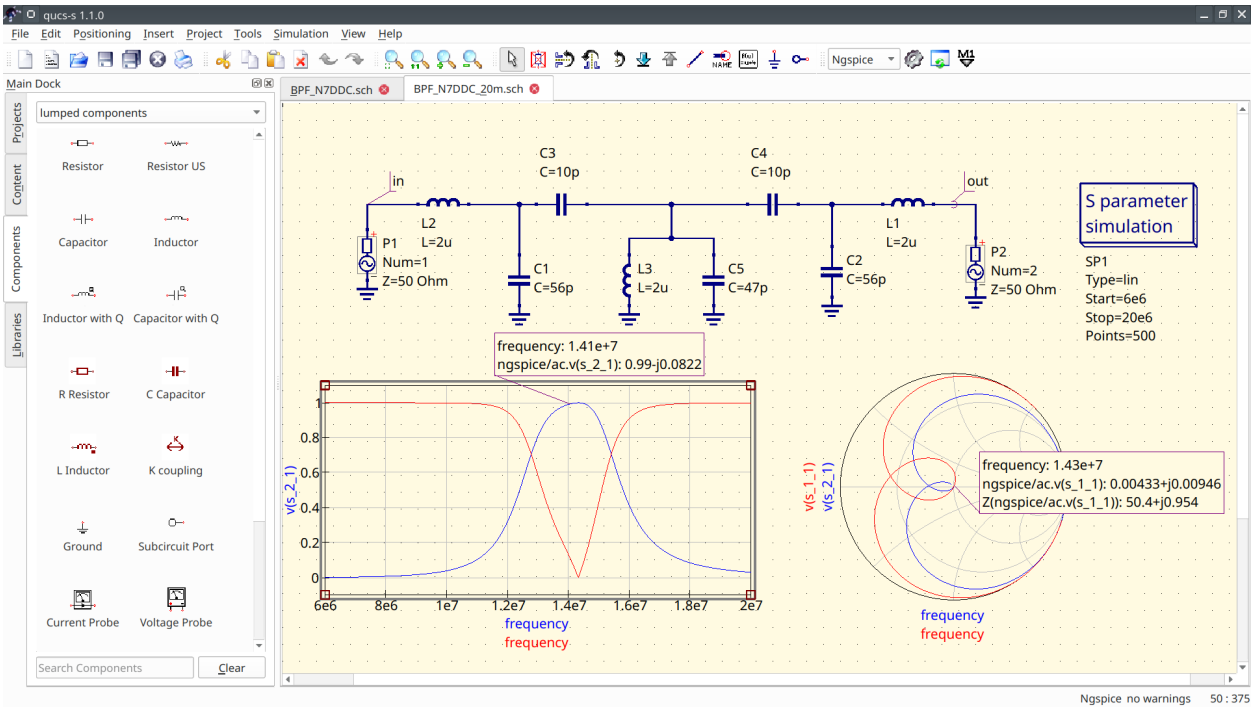


Fig. 22: Example of a bandpass filter (designed for the 20-meter amateur radio band), utilizing the ngspice S-Parameter Simulation feature.

Accessing S-Parameter Outputs from ngspice

After running the simulation, the S-Parameter data can be plotted by referencing special "voltages" that ngspice provides in the simulation output. For example, to plot the S_{21} parameter, you'd need to plot the `v(s_2_1)` variable.

ngspice also converts the S-Parameters to Y and Z parameters automatically. The Y and Z parameters are also available via special "voltage" variables, in the same syntax as the S parameters. For example, to plot Y_{11}, you should select `v(y_1_1)`.

7.2.5.2 RF/Transmission Line Simulation with QucsatorRF

While the QucsatorRF simulation backend is very poorly suited for general-purpose time-domain circuit simulation, it is very well suited for RF simulation, particularly in the frequency domain. It has a number of special features that make it a great choice for simulating RF and microwave systems, particularly those involving transmission lines.

> **Tip**
>
> **QucsatorRF has been shipped with the main Qucs-S application since version `v24.2.0`.** No manual installation of QucsatorRF is necessary.
>
> For more information on QucsatorRF and the other simulation backends, see *the page on simulation backends*.

QucsatorRF supports advanced RF simulation techniques, such as multi-port S-Parameter simulation (including 1-port), and harmonic balance (HB) analysis.

QucsatorRF also includes a number of transmission line models, allowing you to model particular geometries on particular substrates. While these models are not full 3D field solvers, they are significantly more detailed and customizable than the transmission line models included with other backends, such as ngspice.

The most commonly used QucsatorRF transmission line models are:

- **4-Terminal Transmission Line**
- **Twisted Pair**
- **Coaxial Cable**
- **Microstrip Lines** (including special components for modeling tees and corners)
- **Coupled Microstrips**
- **Coplanar Lines**
- **Waveguides**
- **Generic RLCG Line** (distributed-element lines, the model used by the telegrapher's equations)

Using Transmission Line Models with QucsatorRF

> **Warning**
>
> If you are having trouble finding the components referenced in this section, be sure that QucsatorRF is selected as your simulation backend. Qucs-S will only show components that are compatible with the currently-selected simulation backend. See the *Interface Overview* page for details on how to adjust this setting.

To use any of the transmission line models with QucsatorRF, you must first place a *Substrate* component on the schematic page. This defines the material properties. Each substrate is assigned an identifier (for example, `Substr1`), so you can use multiple substrates on a single schematic page, to model more complex multi-material systems.

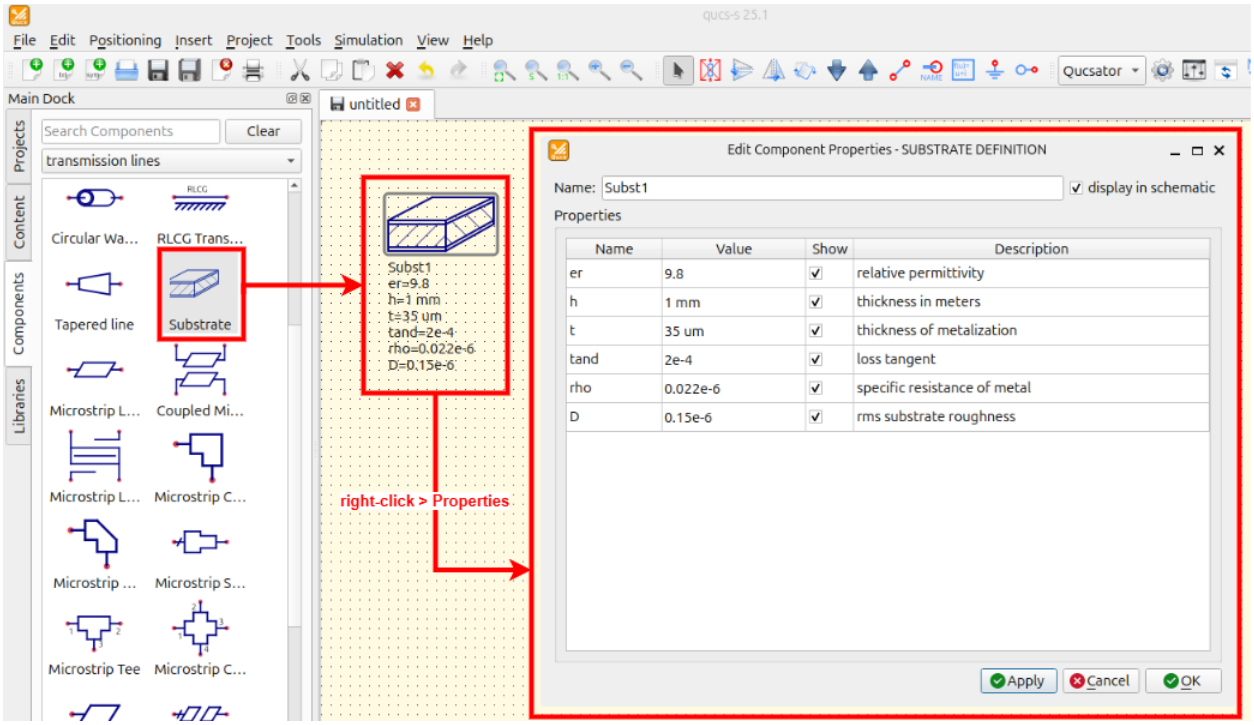


Fig. 23: Example of the *Substrate* component, used to specify material properties of a transmission line component.

After placing at least one *Substrate* component, transmission line components can be placed on the schematic page, referencing the substrate for their material properties. Simply type in the ID of the substrate (for example, `Substr1`) as the value for the `Substr` parameter on your transmission line.

Example

As an example, the figure below shows a 1.5GHz coupled microstrip band-pass filter. It has two ports (*Power Source* devices `P1` and `P2`) connected to input and output, and two microstrip devices (`MS1` and `MS2`), bound to a substrate (`Substr1`). Finally, the *S-Parameter Simulation* component defines the simulation type, and frequency range.

> **Tip**
>
> Although QucsatorRF supports different transmission line models than ngspice, the *S-Parameter Simulation* component is the same across both backends.

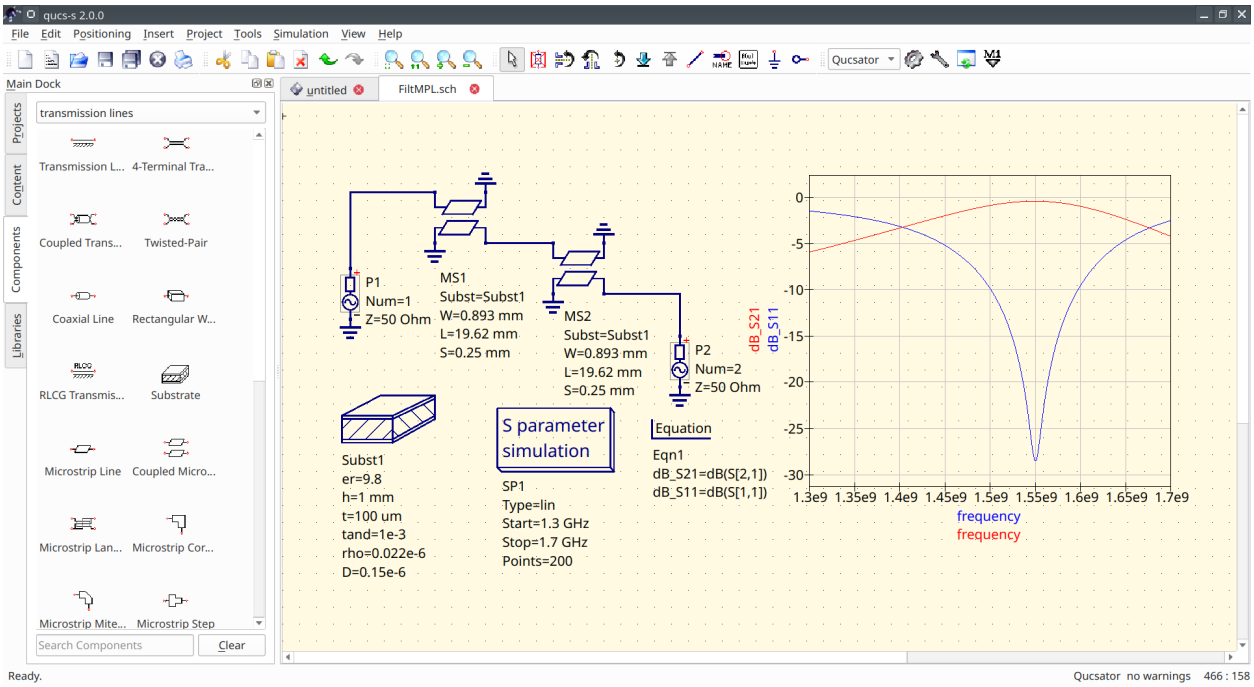


Fig. 24: Example of a 1.5GHz coupled microstrip band-pass filter, being simulated with QucsatorRF. It has two ports (*Power Source* devices `P1` and `P2`) connected to input and output, and two microstrip devices (`MS1` and `MS2`), bound to a substrate (`Substr1`). Finally, the *S-Parameter Simulation* component defines the simulation type, and frequency range.

Accessing Results

There are a few key differences in how results are accessed in QucsatorRF, compared to other backends like ngspice:

- The equation syntax for QucsatorRF is different, and not cross-compatible with ngspice.
- In QucsatorRF, the S-parameters are represented in `S[i,j]` form. For example, `S[2,1]` for S_{21}.
- In QucsatorRF, Y and Z matrices are *NOT* calculated automatically (unlike ngspice). If you want these matrices, you'll need to use the `stoy()` and `stoz()` functions.

CHAPTER

EIGHT

USING EQUATIONS & PARAMETERS

There are many cases in circuit simulation where you may wish to parameterize device properties in your circuit for easy changes, or graph mathematical functions of your simulation data, rather than raw voltages and currents. Qucs-S has various parameter and equation features to support these use cases, depending on which *simulation backend* you are using.

See the sub-pages below for information on using your particular backend's features.

8.1 Equations & Parameters with ngspice

8.1.1 Equations vs Parameters

ngspice has two complementary features:

- **Parameters** are computed *BEFORE* your simulation executes. They can be used to manipulate component values or other circuit properties. This uses the SPICE `.PARAM` feature.
- **Equations** are computed *AFTER* your simulation is complete. They can be used to manipulate the results of your simulation, for graphing or further analysis.

The diagram below shows where *Parameters* and *Equations* each fit into the Simulation process.

8.1.2 Parameters in ngspice with `.PARAM`

The *Parameters* feature can be used much like the Parametric Design features in many mechanical CAD packages. It allows you to describe your circuit elements using parameters, instead of being limited to "hardcoded" constant values. When you run your simulation, these Parameters will be computed *BEFORE* the simulation actually executes.

8.1.2.1 Parameter Syntax Basics

The `.PARAM` syntax is mostly common across all SPICE-compatible simulation engines.

For a definitive reference, see the ngspice manual.

You may also find the LTSpice Wiki helpful as a quick reference.

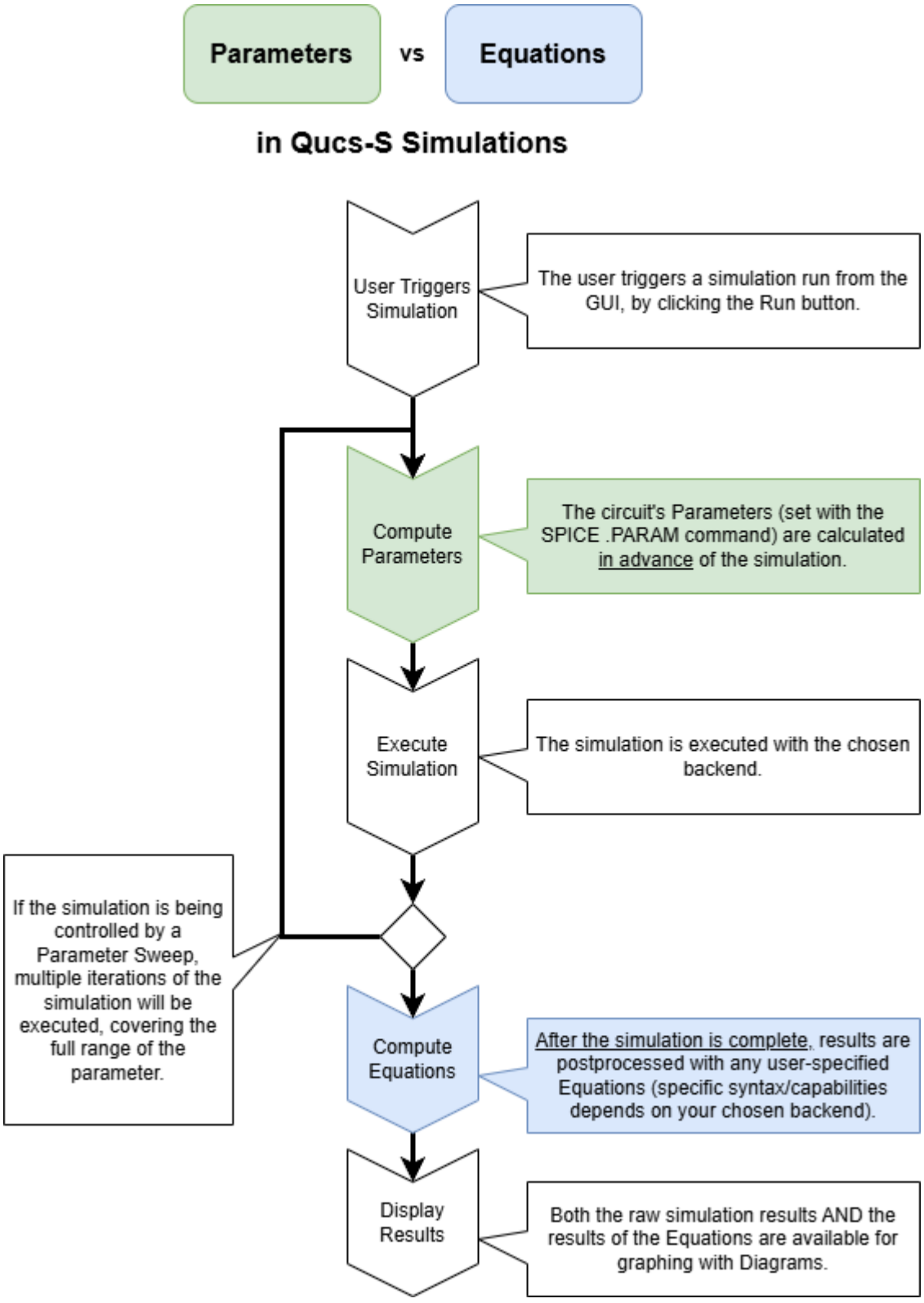


Fig. 1: A simplified diagram of the steps in the Qucs-S simulation process, highlighting the times where *Parameters* and *Equations* are computed.

Operators

Table 1: Operators available in SPICE .PARAM statements, with their "precedence" in the order of operations listed as well. Taken from the ngspice manual.

Operator	Alias	Precedence	Description
-		1	unary -
!		1	unary not
**	^	2	power, like pwr
*		3	multiply
/		3	divide
%		3	modulo
\		3	integer divide
+		4	add
-		4	subtract
==		5	equality
!=	<>	5	non-equal
<=		5	less or equal
>=		5	greater or equal
<		5	less than
>		5	greater than
&&		6	boolean and
||		7	boolean or
c?x:y		8	ternary operator

Functions

Table 2: A list of functions available in SPICE .PARAM statements. Taken from the ngspice manual.

Built-in function	Notes
`sqrt(x)`	`y = sqrt(x)`
`sin(x)`, `cos(x)`, `tan(x)`	
`sinh(x)`, `cosh(x)`, `tanh(x)`	
`asin(x)`, `acos(x)`, `atan(x)`	
`asinh(x)`, `acosh(x)`, `atanh(x)`	
`arctan(x)`	`atan(x)`, kept for compatibility
`exp(x)`	
`ln(x)`, `log(x)`	
`abs(x)`	
`nint(x)`	Nearest integer, half integers towards even
`int(x)`	Nearest integer rounded towards 0
`floor(x)`	Nearest integer rounded towards -∞
`ceil(x)`	Nearest integer rounded towards +∞
`pow(x,y)`	x raised to the power of y (pow from C runtime library)
`pwr(x,y)`	pow(fabs(x), y)
`min(x, y)`	
`max(x, y)`	
`sgn(x)`	1.0 for x > 0, 0.0 for x == 0, -1.0 for x < 0
`ternary_fcn(x, y, z)`	x ? y : z
`gauss(nom, rvar, sigma)`	nominal value plus variation drawn from Gaussian distribution with mean 0 and standard deviation rvar (relative to nominal), divided by sigma
`agauss(nom, avar, sigma)`	nominal value plus variation drawn from Gaussian distribution with mean 0 and standard deviation avar (absolute), divided by sigma
`unif(nom, rvar)`	nominal value plus relative variation (to nominal) uniformly distributed between +/-rvar
`aunif(nom, avar)`	nominal value plus absolute variation uniformly distributed between +/-avar
`limit(nom, avar)`	nominal value +/-avar, depending on random number in [-1, 1[being > 0 or < 0

Scaling Suffixes

Typical SI Suffixes can be used to scale units. See the table below for syntax.

Table 3: Built-in keywords for scaling units according to the International System of Units (SI).

Suffix	Name	Value
g	Giga	1e9
meg	Mega	1e6
k	Kilo	1e3
m	Milli	1e-3
u	Micro	1e-6
n	Nano	1e-9
p	Pico	1e-12
f	Femto	1e-15

8.1.2.2 Example Circuit using Parameters (`.PARAM`)

Consider a simple low-pass filter made up of a resistor and a capacitor. According to the well-known design equations for this type of filter, the half-power cutoff frequency can be described with the equation below:

$f_c = \frac{1}{2\pi RC}$

With some algebra, we can define C in terms of f_c and R:

$C = \frac{1}{2\pi f_c R}$

At this point, we can use the *Parameters* feature to calculate the value of C for a given cutoff frequency automatically.

> **Tip**
>
> If you need to use Pi (π) in a SPICE `.PARAM` component, use the following code: `pi={4*atan(1)}`

After adding the code to create the constant Pi, and adding the design equation from above, we arrive at this `.PARAM` code (with desired cutoff frequency set to 2kHz):

```
R = 1k
fc = 2k
pi = {4*atan(1)}
C = (1)/(2*pi*fc*R)
```

After building up the rest of the filter circuit using an AC Simulation, we can see the result is a success:

8.1.3 Equations in ngspice with Nutmeg

In contrast to the *Parameters* feature from the previous section, *Equations* are calculated *AFTER* your simulation is complete. They are useful for analyzing results, and manipulating your simulation data for display on a *Diagram* (graph).

When using ngspice, Qucs-S leverages its built in "Nutmeg" postprocessor system for an "Equations" functionality.

> **Warning**
>
> Note that Nutmeg is *NOT* the same as the recently-deprecated `ngnutmeg` command-line utility. The ngspice manual mentions several times that `ngnutmeg` is obsolete, but that does *NOT* mean that the full Nutmeg functionality is obsolete, and it does not affect the *Nutmeg Equation* function in Qucs-S.

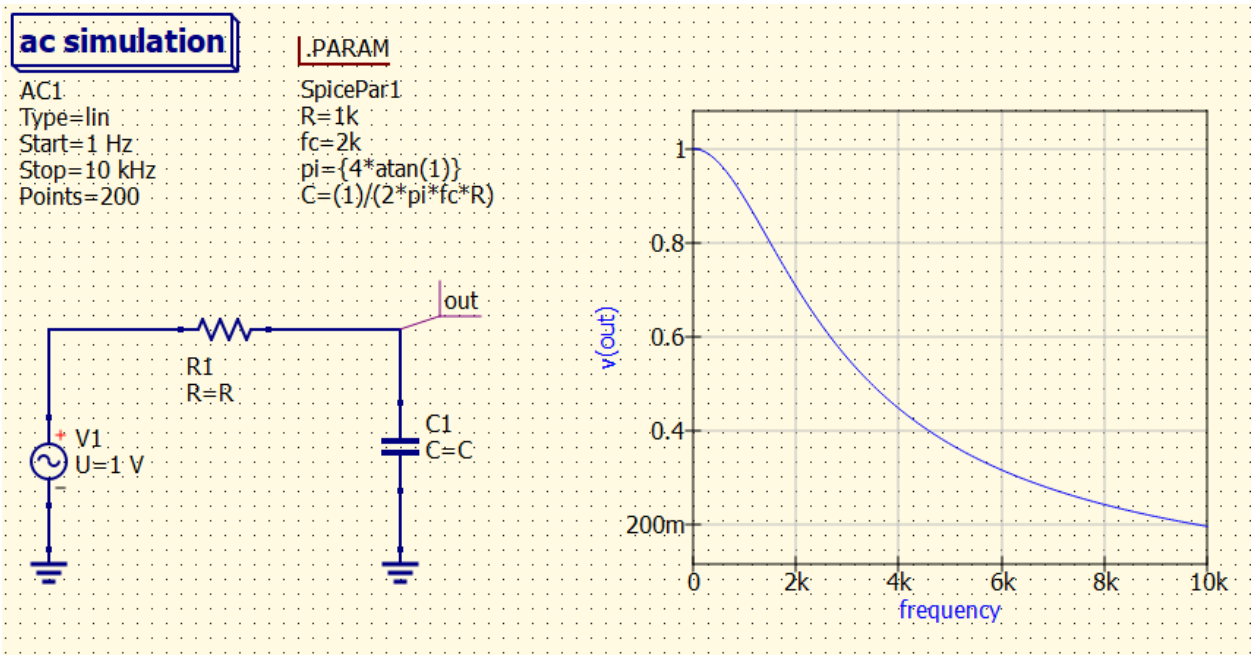


Fig. 2: Example using the Parameters (`.PARAM`) feature, to automatically calculate the capacitor value in an RC Low-Pass filter based on a desired cutoff frequency.

To use this feature, place a *Nutmeg Equation* component on your schematic page. You can find this component under the *Equations* category. An example of the component is shown below, with its Properties dialog annotated.

Tip

You can select which *Simulation Components* a given *Nutmeg Equation* block applies to, within the Properties dialog.

By default, *Nutmeg Equation* blocks apply to all simulations on the schematic page. In most cases, this will suffice. However, in particularly complex simulation cases, limiting the scope to certain simulations may reduce simulation time, or reduce the complexity of the output.

8.1.3.1 Nutmeg Syntax Basics

Tip

For an exhaustive syntactical reference on ngspice Nutmeg equations, see the ngspice manual, Section 13 "Interactive Interpreter".

The Nutmeg equation syntax is relatively simple, and supports all the common mathematical operators, as well as many special functions (trigonometric, logarithmic, etc).

First, it's necessary to understand that ngspice stores all data as a vector. Even scalars are vectors "under the hood", they simply have a length of 1.

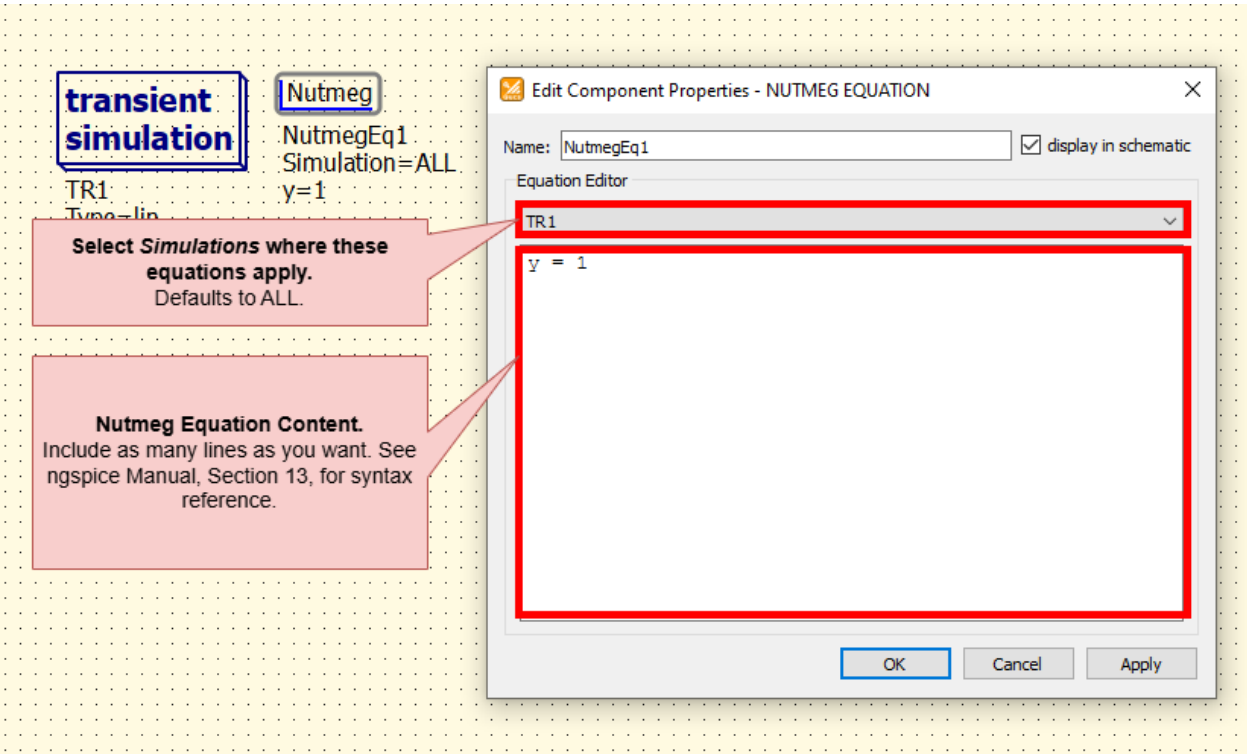


Fig. 3: An example of a *Nutmeg Equation* block, and its Properties. Annotated for clarity.

Algebraic Operators

- `+` : Addition
- `-` : Subtraction
- `*` : Multiplication
- `/` : Division
- `^` : Exponent
- `%` : Modulo (Remainder)
- `,` : Dual-purpose operator, depending on context.
  - If it is used in the arguments list of a user-definable function, it separates the arguments (like most programming languages).
  - In all other cases, it serves as a way to express complex numbers/vectors. For example, `x , y` is synonymous with `x + j(y)`.

Boolean/Relational Operators

- `gt` : Equivalent to $>$
- `lt` : Equivalent to $<$
- `ge` : Equivalent to $>=$
- `le` : Equivalent to $<=$
- `ne` : Equivalent to $<>$

- `and` : Equivalent to &
- `or` : Equivalent to |
- `not` : Equivalent to !
- `eq` : Equivalent to =

Comments

You can place comments anywhere you like in a *Nutmeg Equation*. Simply begin the line with `;`, and that line will be considered a comment. There is no support for "inline" comments - comments must take an entire line. An example of commenting is shown below.

```
;This line is a comment, and will be ignored
y=x*log(a)
;The equation above will be parsed normally
```

Built-In Constants and Variables

ngspice provides a number of pre-defined constants, which you can reference by name. Note that they are all given in MKS units. See table below.

Table 4: Built-in constants available in ngspice. Taken from the ngspice manual.

Name	Description	Value
`pi`		3.14159…
`e`	e (the base of natural logarithms)	2.71828…
`c`	c (the speed of light)	299,792,458 m sec
`i`	i (the square root of -1)	$\sqrt{-1}$
`kelvin`	(absolute zero in centigrade)	-273.15 °C
`echarge`	q (the charge of an electron)	1.60219e-19 C
`boltz`	k (Boltzmann's constant)	1.38062e-23 J K
`planck`	h (Planck's constant)	6.62607e-34 J s
`yes`	boolean	1
`no`	boolean	0
`TRUE`	boolean	1
`FALSE`	boolean	0

Functions

Name	Function
`mag(vector)`	Magnitude of vector (same as abs(vector)).
`ph(vector)`	Phase of complex vector, in radians.
`cph(vector)`	Phase of complex vector, in radians. Continuous values, no discontinuity at $\pm$.
`unwrap(vector)`	Phase of vector with real phase vector in degrees as input and output. Continuous values, no discont
`j(vector)`	i (sqrt(-1)) times vector.
`real(vector)`	The real component of vector.
`imag(vector)`	The imaginary part of vector.
`conj(vector)`	The complex conjugate of a vector

Name	Function
db(vector)	20 log10(mag(vector)).
log10(vector)	The logarithm (base 10) of vector.
ln(vector)	The natural logarithm (base e) of vector.
exp(vector)	e to the vector power.
abs(vector)	The absolute value of vector (same as mag).
sqrt(vector)	The square root of vector.
sin(vector)	The sine of vector.
cos(vector)	The cosine of vector.
tan(vector)	The tangent of vector.
atan(vector)	The inverse tangent of vector.
sinh(vector)	The hyperbolic sine of vector.
cosh(vector)	The hyperbolic cosine of vector.
tanh(vector)	The hyperbolic tangent of vector.
atanh(vector)	The inverse hyperbolic tangent of vector.
floor(vector)	Largest integer that is less than or equal to vector.
ceil(vector)	Smallest integer that is greater than or equal to vector.
norm(vector)	The vector normalized to 1 (i.e, the largest magnitude of any component is 1).
mean(vector)	The result is a scalar (a length 1 vector) that is the mean of the elements of vector (elements values a
avg(vector)	The average of a vector.Returns a vector where each element is the mean of the preceding elements
stddev(vector)	The result is a scalar (a length 1 vector) that is the standard deviation of the elements of vector .
group_delay(vector)	Calculates the group delay -dphase[rad]/d[rad/s] . Input is the complex vector of a system transfe
vector(number)	The result is a vector of length number, with elements 0, 1, ... number - 1. If number is a vector the
cvector(number)	Return a vector of length number, containing complex elements, with the real part values increasing
unitvec(number)	The result is a vector of length number, all elements having a value 1.
length(vector)	The length of vector.
interpolate(plot.vector)	The result of interpolating the named vector onto the scale of the current plot. This function uses th
integ(vector)	Integrates over the given vector (versus the real component of the scale vector), using the trapezioda
deriv(vector)	Calculates the derivative of the given vector. This uses numeric differentiation by interpolating a po
vecd(vector)	Compute the differential of a vector.
vecmin(vector)	Returns the value of the vector element with minimum value. Same as minimum.
minimum(vector)	Returns the value of the vector element with minimum value. Same as vecmin.
vecmax(vector)	Returns the value of the vector element with maximum value. Same as maximum.
maximum(vector)	Returns the value of the vector element with maximum value. Same as vecmax.
fft(vector)	fast fourier transform (13.5.33)
ifft(vector)	inverse fast fourier transform (13.5.33)
sortorder(vector)	Returns a vector with the positions of the elements in a real vector after they have been sorted into i
timer(vector)	Returns CPU-time minus the value of the first vector element.
clock(vector)	Returns wall-time minus the value of the first vector element.
rnd(vector)	A vector with each component a random integer between 0 and the absolute value of the input vecto
sgauss(vector)	Returns a vector of random numbers drawn from a Gaussian distribution (real value, mean = 0 , star
sunif(vector)	Returns a vector of random real numbers uniformly distributed in the interval [-1 .. 1[. The length o
poisson(vector)	Returns a vector with its elements being integers drawn from a Poisson distribution. The elements o
exponential(vector)	Returns a vector with its elements (real numbers) drawn from an exponential distribution. The elem

Example Expressions

Below are a few example expressions, to show syntax. Taken from the ngspice manual.

```
cos(TIME) + db(v(3))
sin(cos(log([1 2 3 4 5 6 7 8 9 10])))
```

(continues on next page)

(continued from previous page)

```
TIME * rnd(v(9)) - 15 * cos(vin#branch) ^ [7.9e5 8]
not ((ac3.FREQ[32] & tran1.TIME[10]) gt 3)
(sunif(0) ge 0) ? 1.0 : 2.0
mag(fft(v(18)))
```

8.1.3.2 Example Circuit with Nutmeg Equations

For a simple demonstration, the example circuit below graphs the ratio of output to input voltage across a range of frequencies, for a simple RC filter. All that was necessary to achieve this was to add the *Nutmeg Equation* block, write an equation to define our new variable (called `K`, in this example), and then set the *Cartesian Diagram* to graph `K` instead of a directly "measurable" variable from the circuit.

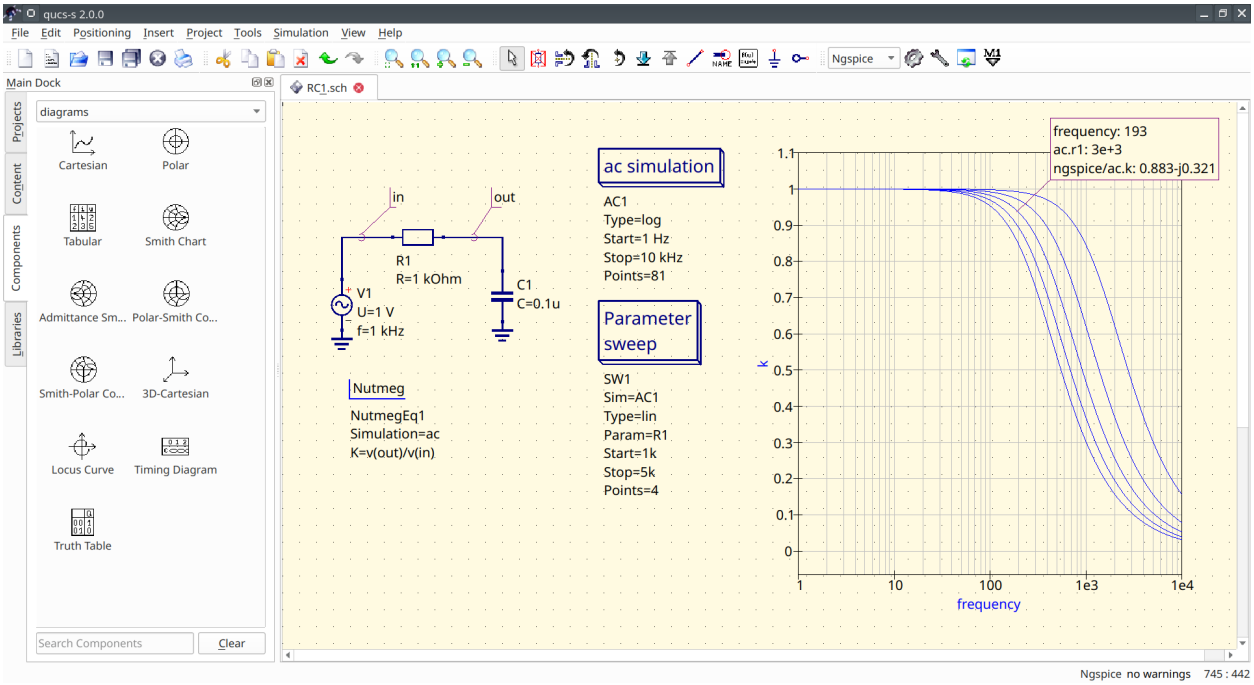


Fig. 4: An example of a *Nutmeg Equation* block being used in a simple RC filter circuit.

Of course, using multiple lines in your *Nutmeg Equation* block, and using the mathematical functions from the tables above, you can extend the functionality far beyond simple division.

8.2 Using Equations in QucsatorRF

Like ngspice, QucsatorRF also has an Equations feature.

However, unlike ngspice, QucsatorRF does *not* separate things into *Parameters* and *Equations* based on when they are computed. In QucsatorRF, all variables/data are accessible from the *Equations* feature. There is no need to use a different feature for pre-simulation and post-simulation computations.

The QucsatorRF *Equations* feature is very similar to the *Equations* feature in the upstream Qucs project. The QucsatorRF simulation backend is a fork of Qucsator, the only simulation backend supported by the upstream Qucs project. This means that most equations written for upstream Qucs will "just work" in Qucs-S, *when used with QucsatorRF as the backend.*

8.2.1 Example: RC Filter

When showing how to utilize Parameters and Equations in ngspice, an RC filter was used an example circuit. In that example, *Parameters* were used to manipulate component values before the simulation, while *Equations* were used to manipulate the results for graphing.

With QucsatorRF, the pre-processing and post-processing features are combined into a single *Equations* feature. This is accessible via the *Qucsator Equation* component, as shown below.

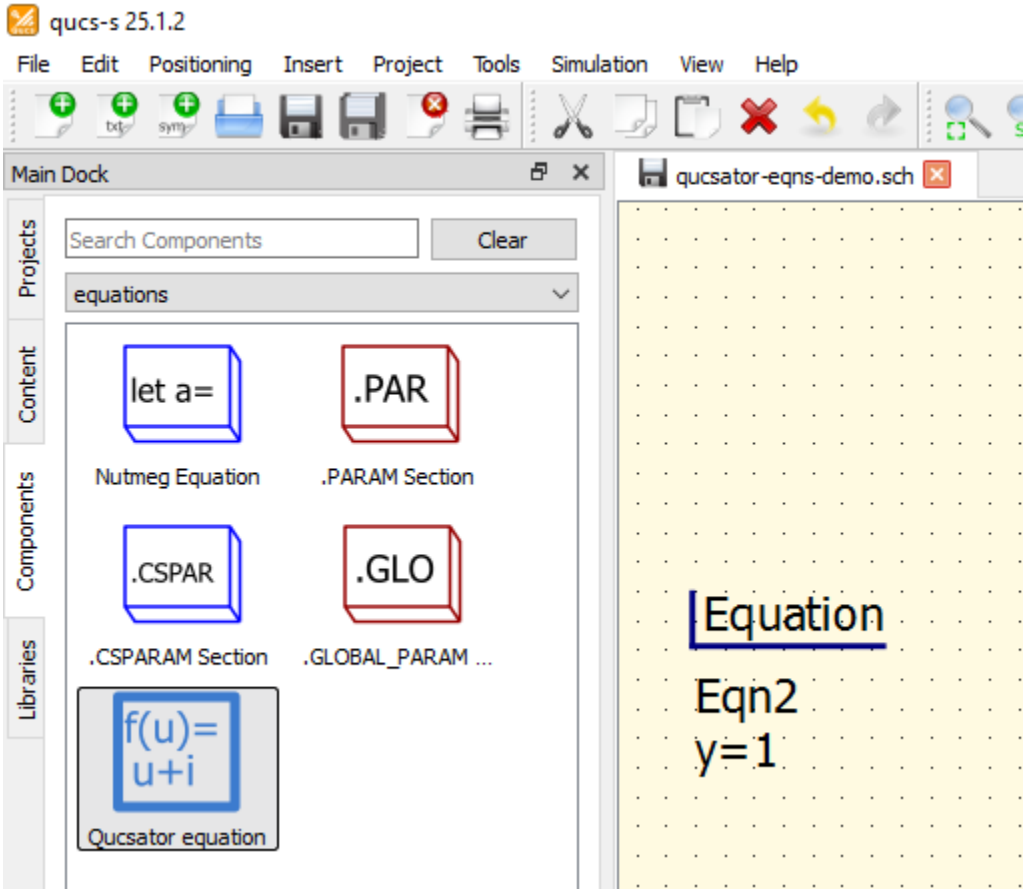


Fig. 5: The *Qucsator Equation* schematic component, shown in the component browser at left, and placed on the schematic page at right.

To demonstrate this, we'll use the same RC filter as the ngspice example. In Qucsator, it's possible to create a single *Equation* component containing both the design equations (governing component values) *and* the postprocessing equation (the ratio of output to input voltage). The text of this equation is shown below (note the syntax difference in QucsatorRF equations compared to ngspice *Nutmeg Equations* and *Parameters*).

> **Warning**
>
> **Qucsator equations are case-sensitive!** This includes references to built-in functions.
>
> For example, `db(foo)` will not work, you need to call `dB(foo)` instead.

```
r_1 = 1k
fc = 2k
```

(continues on next page)

(continued from previous page)

```
C = 1/(2*pi*fc*r_1)
ratio_linear = out.v/in.v
ratio_dB = dB(out.v/in.v)
```

Using the equation text from above, we can create the following circuit. Note the graphed results contain both dB and linear format, utilizing the `dB()` function in QucsatorRF to easily produce a graph in dB format.

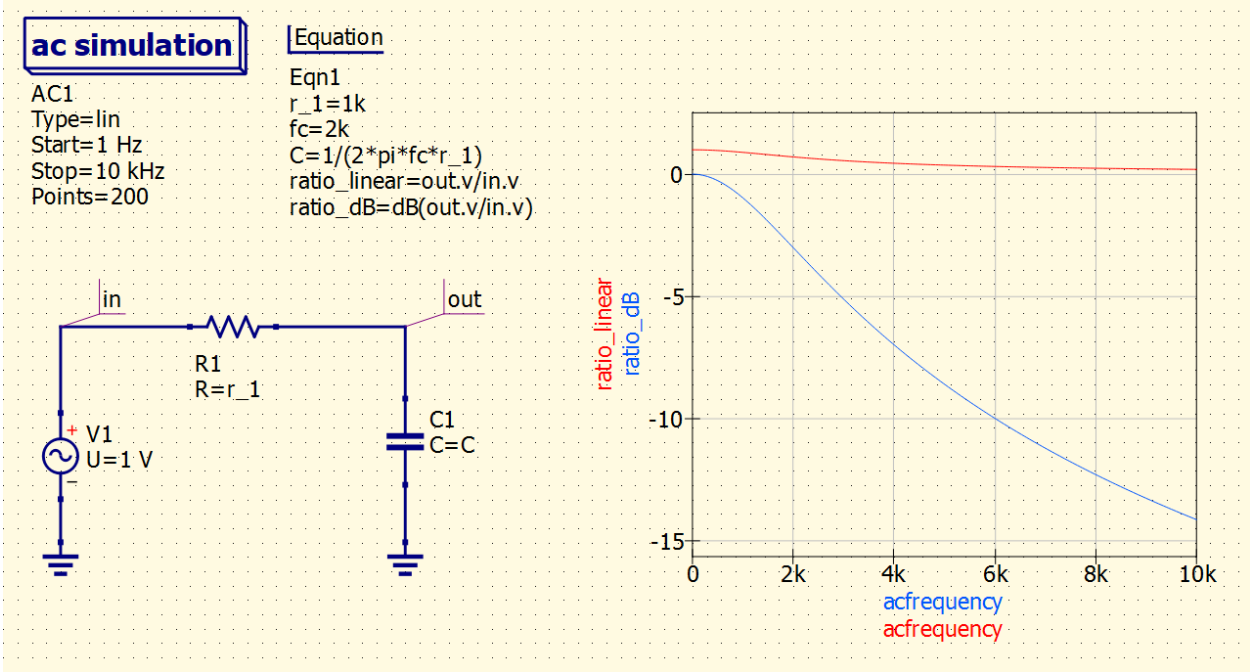


Fig. 6: An example circuit utilizing the *Equations* feature with Qucsator.

8.2.2 Additional Examples

Since the syntax has remained unchanged since Qucs `v0.0.19`, many of the tutorials for original Qucs still apply to Qucsator Equations in Qucs-S. For additional examples, you may wish to visit these references:

- A Tutorial: Component, compact device and circuit modeling using symbolic equations
- Measurement Expressions Reference Manual

8.2.3 Syntax Basics

The following operations and functions can be applied in QucsatorRF equations. Parameters in brackets `[...]` are optional.

> **Tip**
>
> **For a more exhaustive reference, please see the Measurement Expressions Reference Manual**, a document written by the original Qucs team. The syntax for QucsatorRF equations in Qucs-S is unchanged compared to upstream Qucs.

8.2.3.1 Operators

Arithmetic Operators

`+x`	Unary plus
`-x`	Unary minus
`x+y`	Addition
`x-y`	Subtraction
`x*y`	Multiplication
`x/y`	Division
`x%y`	Modulo (remainder of division)
`x^y`	Power

Logical Operators

`!x`	Negation
`x&&y`	And
`` `x ``	
`x^^y`	Exclusive or
`x?y:z`	Abbreviation for conditional expression - if `x` then `y` else `z`
`x==y`	Equal
`x!=y`	Not equal
`x<y`	Less than
`x<=y`	Less than or equal
`x>y`	Larger than
`x>=y`	Larger than or equal

8.2.3.2 Math Functions

Vectors and Matrices: Creation

`eye(n)`	Creates `n` x `n` identity matrix
`length(y)`	Returns the length of the `y` vector
`linspace(from,to,n)`	Real vector with `n` lin spaced components between `from` and `to`
`logspace(from,to,n)`	Real vector with `n` log spaced components between `from` and `to`

Vectors and Matrices: Basic Matrix Functions

`adjoint(x)`	Adjoint matrix of `x` (transposed and conjugate complex)
`det(x)`	Determinant of a matrix `x`
`inverse(x)`	Inverse matrix of `x`
`transpose(x)`	Transposed matrix of `x` (rows and columns exchanged)

Elementary Mathematical Functions: Basic Real and Complex Functions

`abs(x)`	Absolute value, magnitude of complex number
`angle(x)`	Phase angle in radians of a complex number. Synonym for `arg()`
`arg(x)`	Phase angle in radians of a complex number
`conj(x)`	Conjugate of a complex number
`deg2rad(x)`	Converts phase from degrees into radians
`hypot(x,y)`	Euclidean distance function
`imag(x)`	Imaginary part of a complex number
`mag(x)`	Magnitude of a complex number
`norm(x)`	Square of the absolute value of a vector
`phase(x)`	Phase angle in degrees of a complex number
`polar(m,p)`	Transform polar coordinates `m` and `p` into a complex number
`rad2deg(x)`	Converts phase from radians into degrees
`real(x)`	Real part of a complex number
`sign(x)`	Signum function
`sqr(x)`	Square (power of two) of a number
`sqrt(x)`	Square root
`unwrap(p[,tol[,step]])`	Unwrap angle `p` (radians) – defaults `step` = 2pi, `tol` = pi

Elementary Mathematical Functions: Exponential and Logarithmic Functions

`exp(x)`	Exponential function to basis e
`limexp(x)`	Limited exponential function
`log10(x)`	Decimal logarithm
`log2(x)`	Binary logarithm
`ln(x)`	Natural logarithm (base e)

Elementary Mathematical Functions: Trigonometry

`cos(x)`	Cosine function
`cosec(x)`	Cosecant
`cot(x)`	Cotangent function
`sec(x)`	Secant
`sin(x)`	Sine function
`tan(x)`	Tangent function

Elementary Mathematical Functions: Inverse Trigonometric Functions

`arccos(x)`	Arc cosine (also known as "inverse cosine")
`arccosec(x)`	Arc cosecant
`arccot(x)`	Arc cotangent
`arcsec(x)`	Arc secant
`arcsin(x)`	Arc sine (also known as "inverse sine")
`arctan(x[,y])`	Arc tangent (also known as "inverse tangent")

Elementary Mathematical Functions: Hyperbolic Functions

`cosh(x)`	Hyperbolic cosine
`cosech(x)`	Hyperbolic cosecant
`coth(x)`	Hyperbolic cotangent
`sech(x)`	Hyperbolic secant
`sinh(x)`	Hyperbolic sine
`tanh(x)`	Hyperbolic tangent

Elementary Mathematical Functions: Inverse Hyperbolic Functions

`arcosh(x)`	Hyperbolic area cosine
`arcosech(x)`	Hyperbolic area cosecant
`arcoth(x)`	Hyperbolic area cotangent
`arsech(x)`	Hyperbolic area secant
`arsinh(x)`	Hyperbolic area sine
`artanh(x)`	Hyperbolic area tangent

Elementary Mathematical Functions: Rounding

`ceil(x)`	Round to the next higher integer
`fix(x)`	Truncate decimal places from real number
`floor(x)`	Round to the next lower integer
`round(x)`	Round to nearest integer

Elementary Mathematical Functions: Special Mathematical Functions

`besseli0(x)`	Modified Bessel function of order zero
`besselj(n,x)`	Bessel function of first kind and `n`-th order
`bessely(n,x)`	Bessel function of second kind and `n`-th order
`erf(x)`	Error function
`erfc(x)`	Complementary error function
`erfinv(x)`	Inverse error function
`erfcinv(x)`	Inverse complementary error function
`sinc(x)`	Sinc function (sin(`x`)/`x` or 1 at `x` = 0)
`step(x)`	Step function

Data Analysis: Basic Statistics

`avg(x[,range])`	Average of vector `x`. If range given `x` must have a single data dependency
`cumavg(x)`	Cumulative average of vector elements
`max(x,y)`	Returns the greater of the values `x` and `y`
`max(x[,range])`	Maximum of vector `x`. If range given `x` must have a single data dependency
`min(x,y)`	Returns the lesser of the values `x` and `y`
`min(x[,range])`	Minimum of vector `x`. If range is given `x` must have a single data dependency
`rms(x)`	Root Mean Square of vector elements
`runavg(x)`	Running average of vector elements
`stddev(x)`	Standard deviation of vector elements
`variance(x)`	Variance of vector elements
`random()`	Random number between 0.0 and 1.0
`srandom(x)`	Give random seed

Data Analysis: Basic Operation

Data Analysis: Differentiation and Integration

`ddx(expr,var)`	Derives mathematical expression `expr` with respect to the variable `var`
`diff(y,x[,n])`	Differentiate vector `y` with respect to vector `x` `n` times. Defaults to `n` = 1
`integrate(x,h)`	Integrate vector `x` numerically assuming a constant step-size `h`

Data Analysis: Signal Processing

`dft(x)`	Discrete Fourier Transform of vector `x`
`fft(x)`	Fast Fourier Transform of vector `x`
`fftshift(x)`	Shuffles the FFT values of vector `x` to move DC to the center of the vector
`Freq2Time(V,f)`	Inverse Discrete Fourier Transform of function `V(f)` interpreting it physically
`idft(x)`	Inverse Discrete Fourier Transform of vector `x`
`ifft(x)`	Inverse Fast Fourier Transform of vector `x`
`kbd(x[,n])`	Kaiser-Bessel derived window
`Time2Freq(v,t)`	Discrete Fourier Transform of function `v(t)` interpreting it physically

8.2.3.3 Electronics Functions

Unit Conversion

`dB(x)`	dB value
`dbm(x)`	Convert voltage to power in dBm
`dbm2w(x)`	Convert power in dBm to power in Watts
`w2dbm(x)`	Convert power in Watts to power in dBm
`vt(t)`	Thermal voltage for a given temperature `t` in Kelvin

Reflection Coefficients and VSWR

`rtoswr(x)`	Converts reflection coefficient to voltage standing wave ratio (VSWR)
`rtoy(x[,zref])`	Converts reflection coefficient to admittance; default `zref` = 50 ohms
`rtoz(x[,zref])`	Converts reflection coefficient to impedance; default `zref` = 50 ohms
`ytor(x[,zref])`	Converts admittance to reflection coefficient; default `zref` = 50 ohms
`ztor(x[,zref])`	Converts impedance to reflection coefficient; default `zref` = 50 ohms

N-Port Matrix Conversions

`stos(s,zref[,z0])`	Converts S-parameter matrix to S-parameter matrix with a different Z0
`stoy(s[,zref])`	Converts S-parameter matrix to Y-parameter matrix
`stoz(s[,zref])`	Converts S-parameter matrix to Z-parameter matrix
`twoport(m,from,to)`	Converts a two-port matrix: `from` and `to` are 'Y', 'Z', 'H', 'G', 'A', 'S' and 'T'.
`ytos(y[,z0])`	Converts Y-parameter matrix to S-parameter matrix
`ytoz(y)`	Converts Y-parameter matrix to Z-parameter matrix
`ztos(z[,z0])`	Converts Z-parameter matrix to S-parameter matrix
`ztoy(z)`	Converts Z-parameter matrix to Y-parameter matrix

Amplifiers

`GaCircle(s,Ga[,arcs])`	Available power gain `Ga` circles (source plane)
`GpCircle(s,Gp[,arcs])`	Operating power gain `Gp` circles (load plane)
`Mu(s)`	Mu stability factor of a two-port S-parameter matrix
`Mu2(s)`	Mu' stability factor of a two-port S-parameter matrix
`NoiseCircle(Sopt,Fmin,Rn,F[,Arcs])`	Noise Figure(s) `F` circles
`PlotVs(data,dep)`	Returns data selected from `data`: dependency `dep`
`Rollet(s)`	Rollet stability factor of a two-port S-parameter matrix
`StabCircleL(s[,arcs])`	Stability circle in the load plane
`StabCircleS(s[,arcs])`	Stability circle in the source plane
`StabFactor(s)`	Stability factor of a two-port S-parameter matrix
`StabMeasure(s)`	Stability measure B1 of a two-port S-parameter matrix

8.2.3.4 Nomenclature

Ranges

`LO:HI`	Range from `LO` to `HI`
`:HI`	Up to `HI`
`LO:`	From `LO`
`:`	No range limitations

Matrices and Matrix Elements

`M`	The whole matrix `M`
`M[2,3]`	Element being in 2nd row and 3rd column of matrix `M`
`M[:,3]`	Vector consisting of 3rd column of matrix `M`

Immediate

`2.5`	Real number
`1.4+j5.1`	Complex number
`[1,3,5,7]`	Vector
`[11,12;21,22]`	Matrix

Number suffixes

`E`	exa, 1e+18
`P`	peta, 1e+15
`T`	tera, 1e+12
`G`	giga, 1e+9
`M`	mega, 1e+6
`k`	kilo, 1e+3
`m`	milli, 1e-3
`u`	micro, 1e-6
`n`	nano, 1e-9
`p`	pico, 1e-12
`f`	femto, 1e-15
`a`	atto, 1e-18

Name of Values

`S[1,1]`	S-parameter value
nodename`.V`	DC voltage at node *nodename*
name`.I`	DC current through component *name*
nodename`.v`	AC voltage at node *nodename*
name`.i`	AC current through component *name*
nodename`.vn`	AC noise voltage at node *nodename*
name`.in`	AC noise current through component *name*
nodename`.Vt`	Transient voltage at node *nodename*
name`.It`	Transient current through component *name*

Note: All voltages and currents are peak values. Note: Noise voltages are RMS values at 1 Hz bandwidth.

8.2.3.5 Constants

`i, j`	Imaginary unit (“square root of -1”)
`pi`	4*arctan(1) = 3.14159…
`e`	Euler = 2.71828…
`kB`	Boltzmann constant = 1.38065e-23 J/K
`q`	Elementary charge = 1.6021765e-19 C

CHAPTER

NINE

DISPLAYING RESULTS

After running your simulation in Qucs-S, a *Dataset* is generated. This contains the raw output data from the simulation. See *Understanding File Structure* for more information.

To display your simulation *Dataset* in a human-readable form, use the *Diagrams* feature. *Diagrams* are graphs/plots/tables that you can place on your schematic page. An example is shown in the figure below.

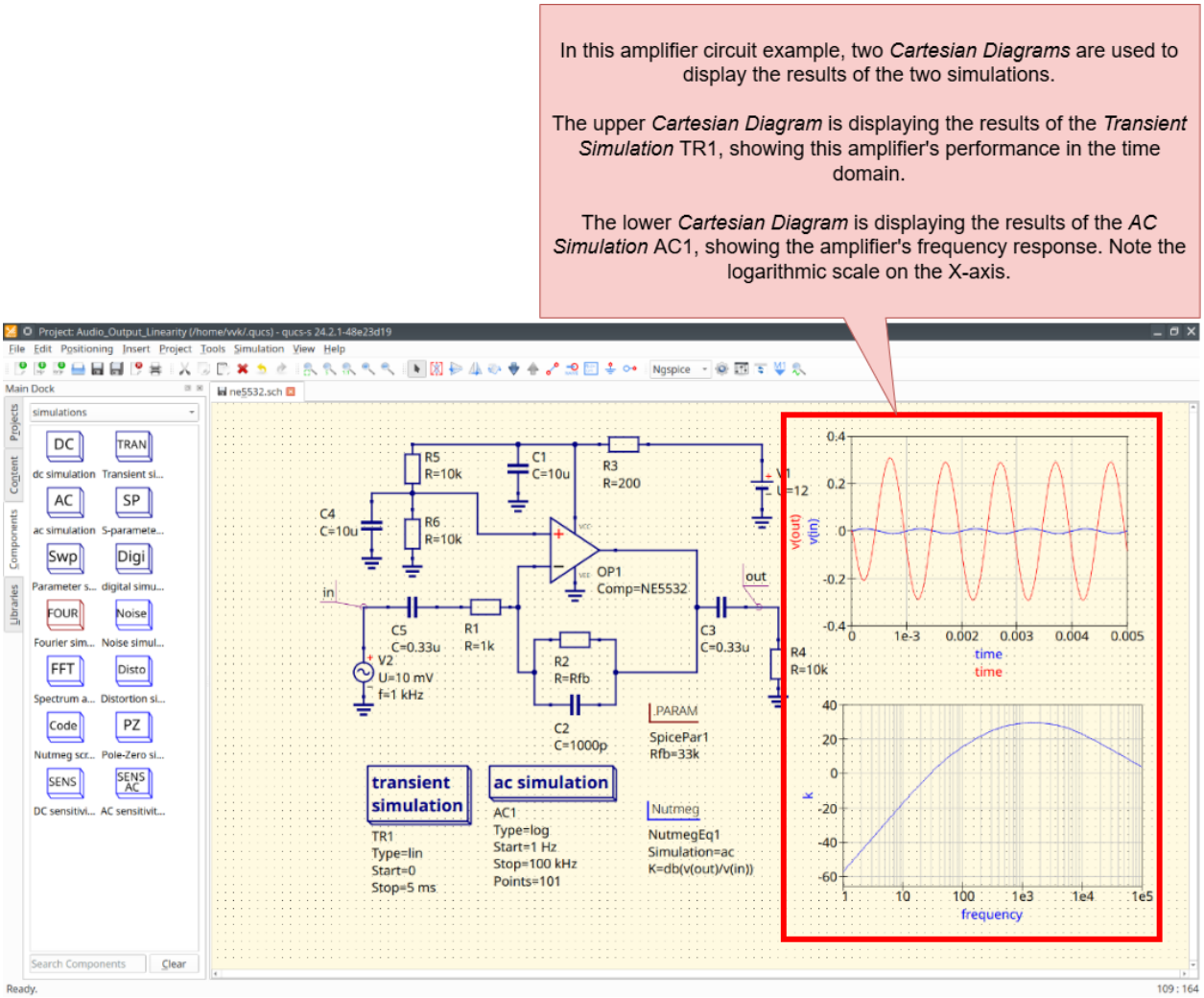


Fig. 1: An amplifier being simulated in Qucs-S, with *Cartesian Diagrams* to display its performance, in both the time domain and the frequency domain.

9.1 Available Diagrams

Qucs-S supports the following diagrams, regardless of which *backend you choose*:

9.1.1 Basic Diagrams

- **Cartesian Graph (2D)**
- **Polar Graph**
- **3D Cartesian Graph**
- **Tabular**: A simple table. Particularly useful for DC simulations.

9.1.2 Smith Charts

- **Smith Chart**
- **Admittance Smith Chart**
- **Polar-Smith Combination Chart**
- **Smith-Polar Combination Chart**

9.1.3 Specialty Diagrams

- **Locus Curve**

9.1.4 Digital-Focused Diagrams

See the *introduction to Digital Simulation.*

- **Timing Diagram**
- **Truth Table**

9.2 Creating Diagrams

In this example, we'll be using the simple RC circuit shown below to demonstrate the *Diagrams* feature.

> **Warning**
>
> **You MUST run a simulation at least once before attempting to create a *Diagram*.** If you don't do this, no data/variables will be available for you to select (because without running at least one simulation, there is no *Dataset* to graph). Qucs-S will not allow you to place a *Diagram* without having a *Dataset* for it to depict.

9.2.1 Placing on the Page

To show this filter's frequency response, we can graph V_{out} on a normal 2-dimensional Cartesian plot.

> **Tip**
>
> In reality, it's more common to graph a ratio to show frequency response, perhaps $\frac{V_{out}}{V_{in}}$. It's also very common to convert the ratio to dB.

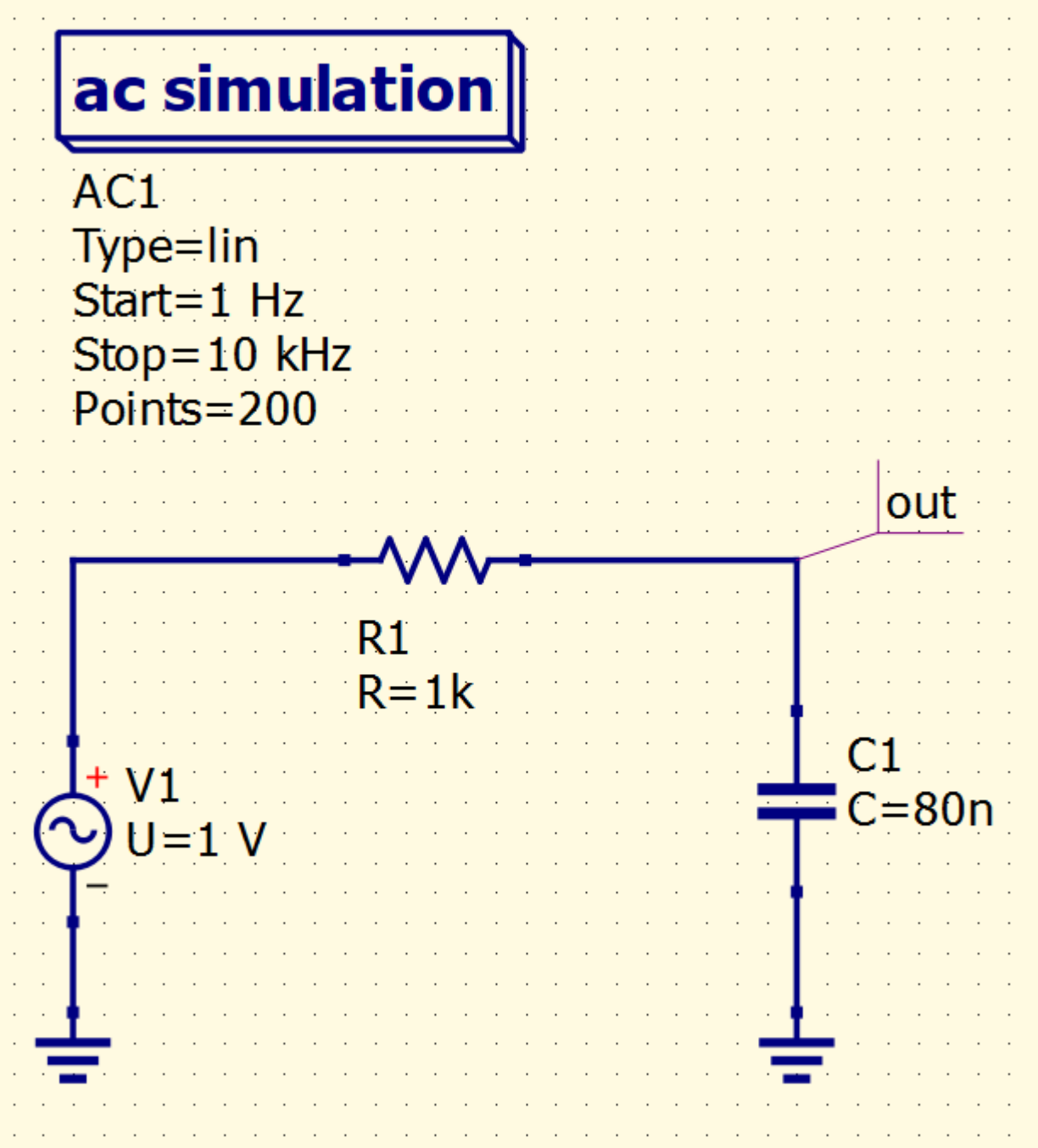


Fig. 2: Simple RC filter circuit, which will serve as an example of how to use *Diagrams*.

This is all achievable within Qucs-S, you just need to *use the* Equations *feature.* However, for this example, we will simply graph V_{out}, to keep things simple.

To do this, we can use a *Cartesian Diagram*. This is available in the *Diagrams* section of the *Components* tab. To place one on your schematic page, single-click on the diagram in the *Components* tab, then click somewhere on your schematic page to place it there.

After placing it on your schematic page, you'll be greeted with the *Properties* dialog for your diagram. This dialog is shown below, with annotations on how to navigate it.

You'll need to choose at least one variable to graph on the *Diagram*. In this case, we'll graph the voltage at the node labeled `out`. We can do this by double-clicking `ac.v(out)` in the "Dataset" section of the dialog.

With the ngspice backend in the example above, variables are in the form `simtype.datatype(netname)`, where `simtype` is the simulation type (AC, transient, etc), `datatype` is the electrical data type (voltage, current, etc), and `netname` is the label applied to the circuit node.

Tip

Variables may appear with a different naming convention when using other simulation backends, but navigating the Properties dialog is the same.

After adding the `ac.v(out)` variable to this graph, the Properties dialog looks like this:

At this point, we can click "OK" to exit the Properties dialog, and the graph will appear on the schematic page as shown below.

9.2.2 Customizing Diagrams

A number of additional customizations can be made to the *Diagrams*, from within the Properties dialog. Available customizations may vary across the *Diagram* types. The most common customizations are described in the sections below.

9.2.2.1 Setting Trace Thickness/Color/Style

Each trace on a graph can be set to any arbitrary color, thickness, or line style (solid, dashed, dotted, circles, stars, and more). This is all adjusted within the Data tab of the Properties dialog. First, select the trace you wish to adjust under the "Graph" section at the bottom right, then its settings will appear in the "Graph Input" section (top center of the window).

Color can be set by clicking the "Color" rectangle in the "Graph Input" section. This opens an additional window (shown to the right of the main Properties window in the figure below). The color can be selected manually, from common presets, or specified numerically using hexadecimal, RGB, or HSV format. It can also be chosen with an eyedropper-style tool by clicking "Pick Screen Color".

9.2.2.2 Setting Graph Limits

The limits/window of a graph can be set on the Limits tab of the Properties dialog. In most cases, Qucs-S will attempt to automatically select appropriate graph limits, unless you open the Properties dialog and override it. The exact settings present in this tab will vary depending on the type of *Diagram* you are interacting with. The figure below shows the settings for a simple 2D *Cartesian Diagram*.

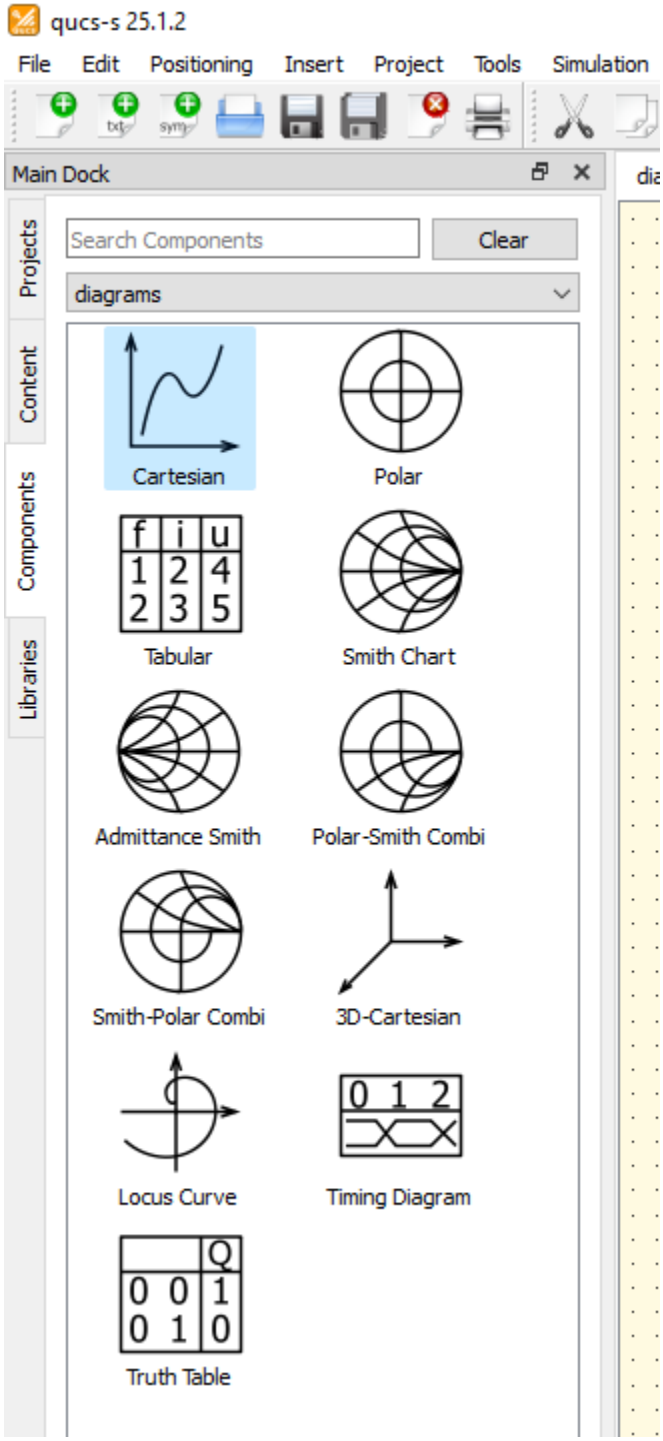


Fig. 3: The *Diagrams* section of the *Components* tab, with the *Cartesian Diagram* selected.

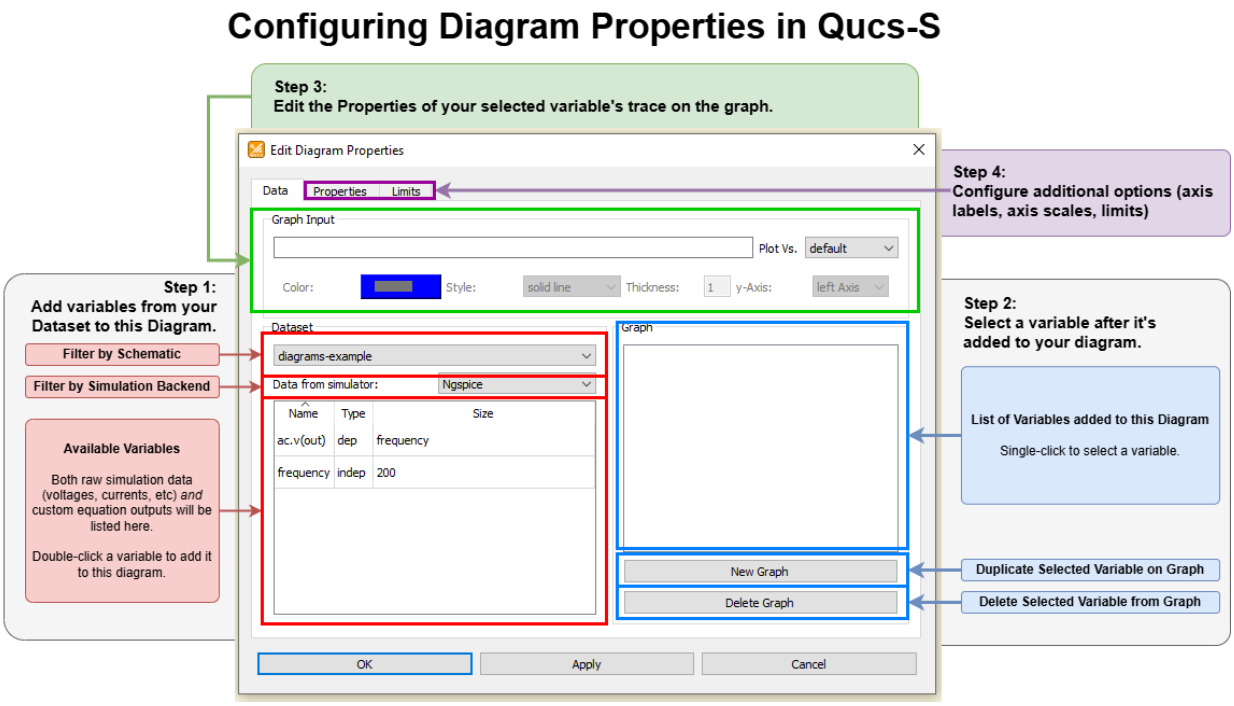


Fig. 4: An example of the *Properties* dialog for a *Cartesian Diagram* in Qucs-S, with annotations explaining how to configure it. Most of these options will be similar across other diagram types.

9.2.2.3 Configuring Axis Labels and Scales

The scaling for each axis, and the label applied to it, can be edited in the Properties tab of the Properties dialog. The figure below shows the settings for a 2D *Cartesian Diagram*, but of course the available settings will vary depending on which *Diagram* type you are using.

9.3 Exporting Diagrams

There are two ways you can export a *Diagram*:

- You can export *a single trace* to a CSV file.
- You can export *the entire graph* to a PNG or SVG image file.

9.3.1 Image Export

Right-click anywhere on the graph's background (NOT on a trace line itself), and choose "Export to Image".

> **Tip**
>
> To change the image export type from `SVG` to `PNG`, you must manually edit the file name in the "Save to file" box.

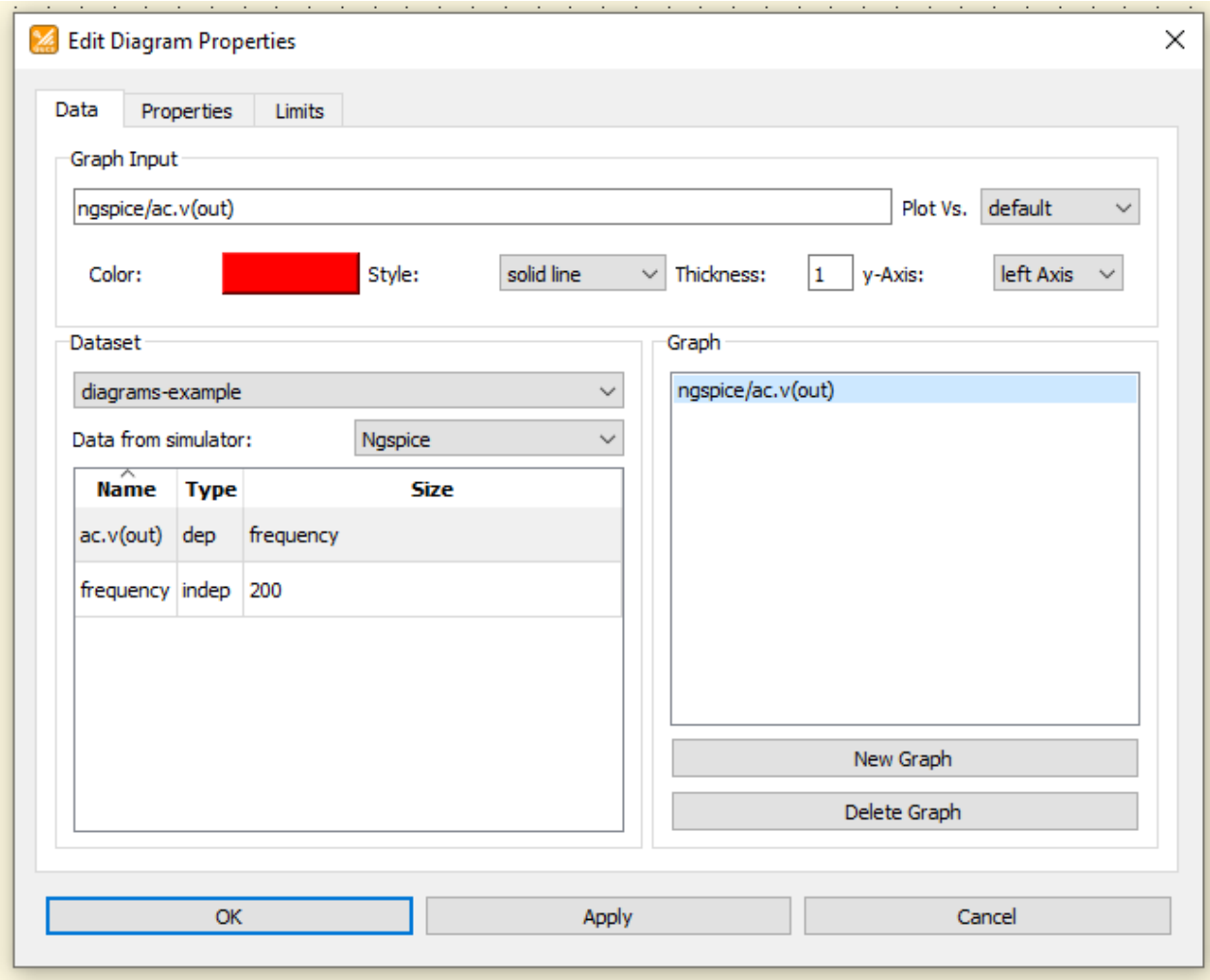


Fig. 5: The Properties dialog for the example *Cartesian Diagram*, after adding the `ac.v(out)` variable to the *Diagram.*

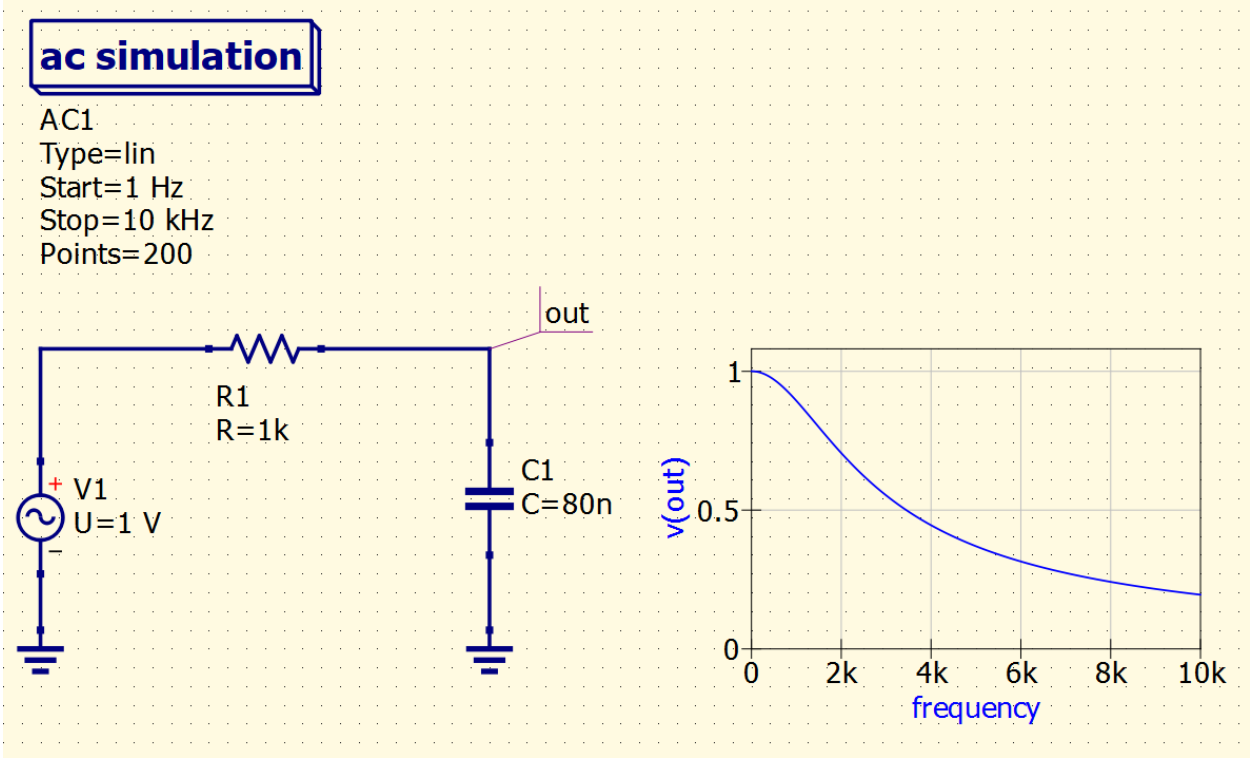


Fig. 6: The example RC filter circuit, with a *Cartesian Diagram* succesfully placed on the page, graphing the frequency response of the voltage at the `out` node.

9.3.2 CSV Export

Right click *on a trace itself* (NOT on the graph's background) and choose "Export to CSV". Note that this will NOT export the entire graph, but only the selected trace.

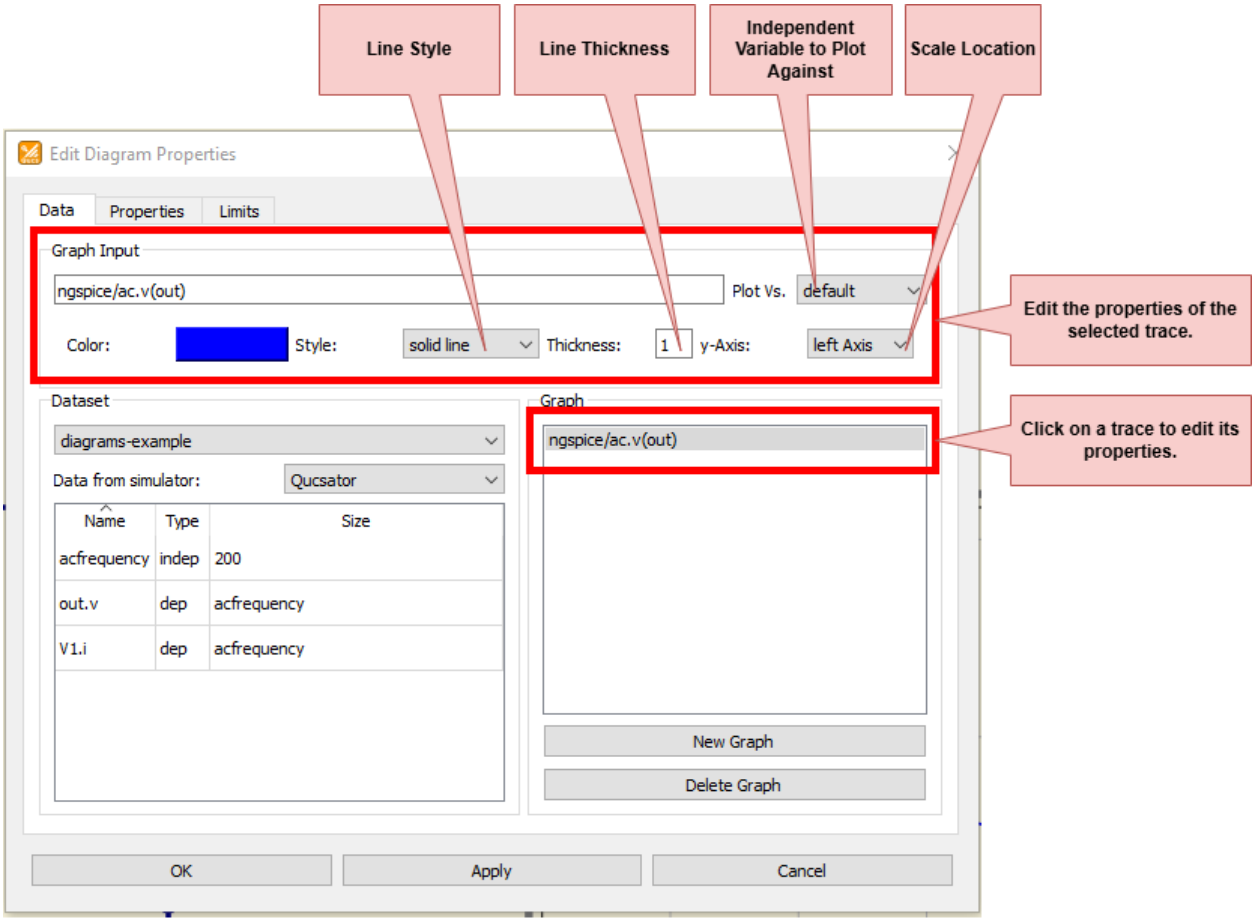


Fig. 7: Editing properties of a trace on a *Cartesian Diagram*.

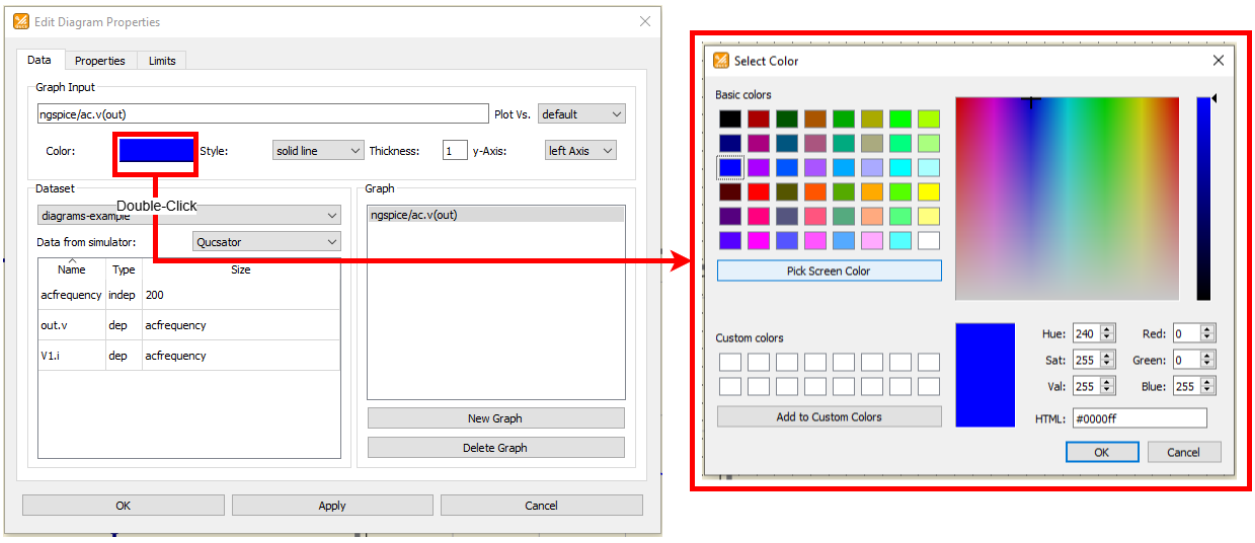


Fig. 8: Editing color of a trace on a *Cartesian Diagram*.

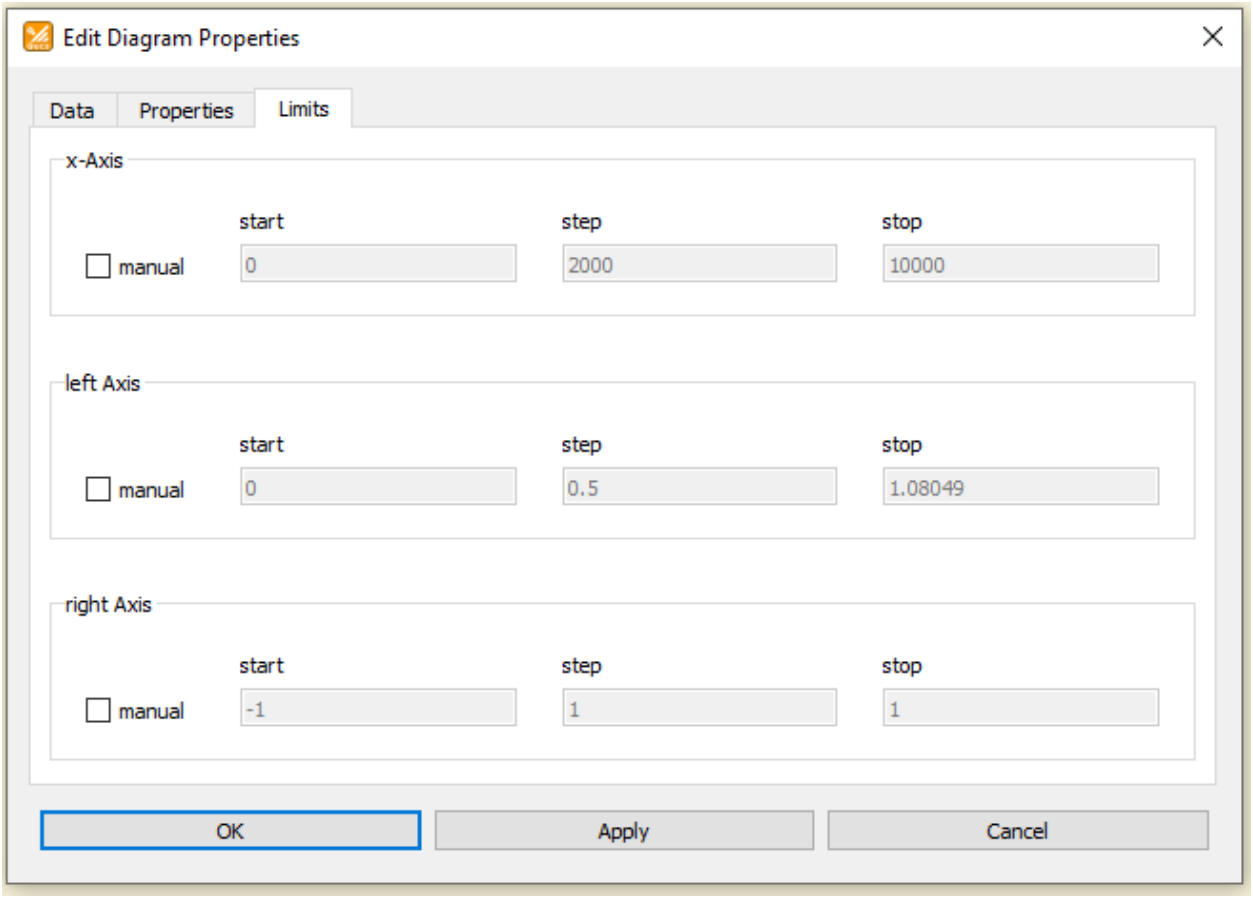


Fig. 9: The Limits section of the Properties dialog for a *Cartesian Diagram*. Here, you can manually select the limits of the graph.

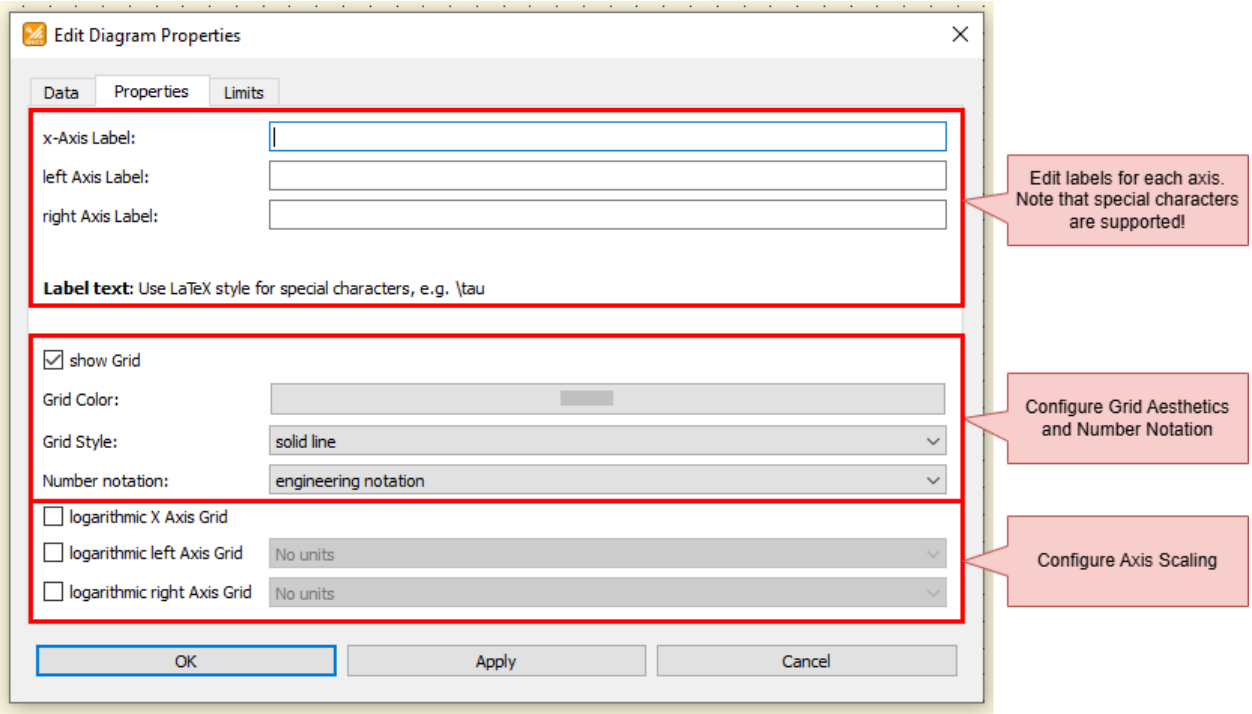


Fig. 10: The Properties section of the Properties dialog for a *Cartesian Diagram*. Here, you can add custom labels to each axis, and configure the scaling of each axis.

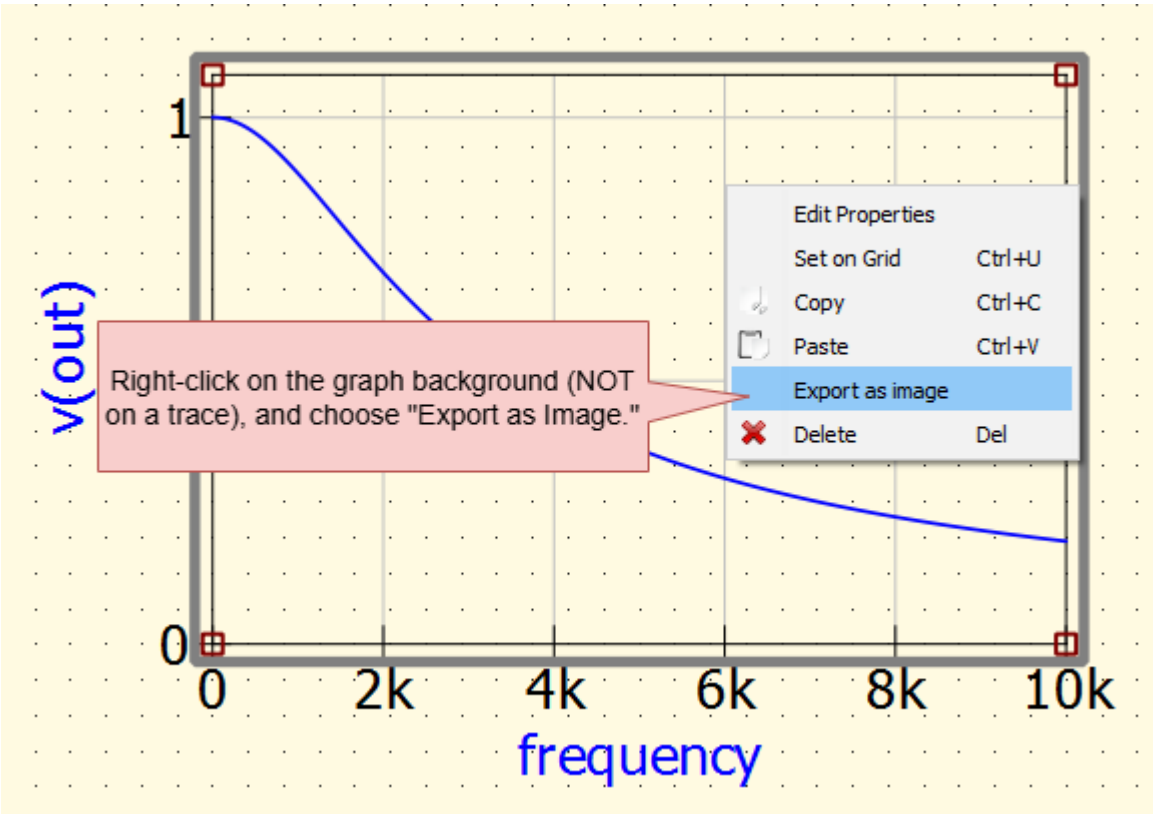


Fig. 11: Exporting an entire graph to an image file.

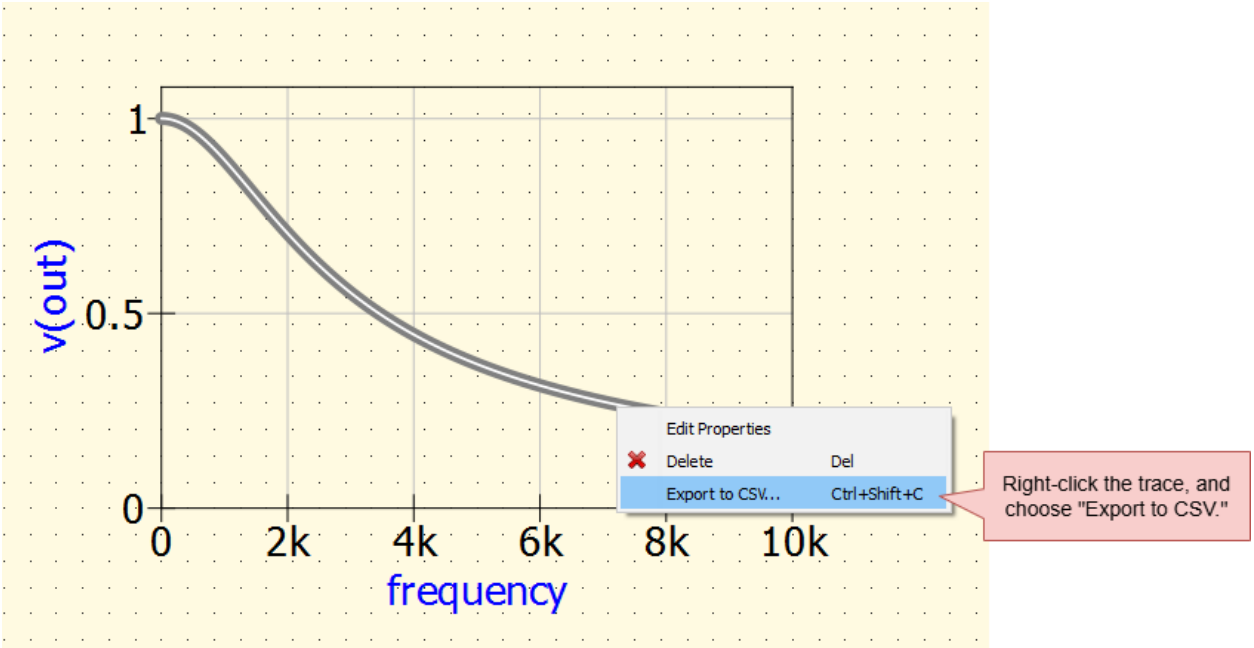


Fig. 12: Exporting one trace from your graph to a CSV file.

CHAPTER

TEN

WORKING WITH SUBCIRCUITS

Complex electrical circuits are often difficult to manage on a single schematic page. You may also wish to reuse the same circuit across multiple situations, following a hierarchical design paradigm. For these reasons, Qucs-S provides a "Subcircuits" feature.

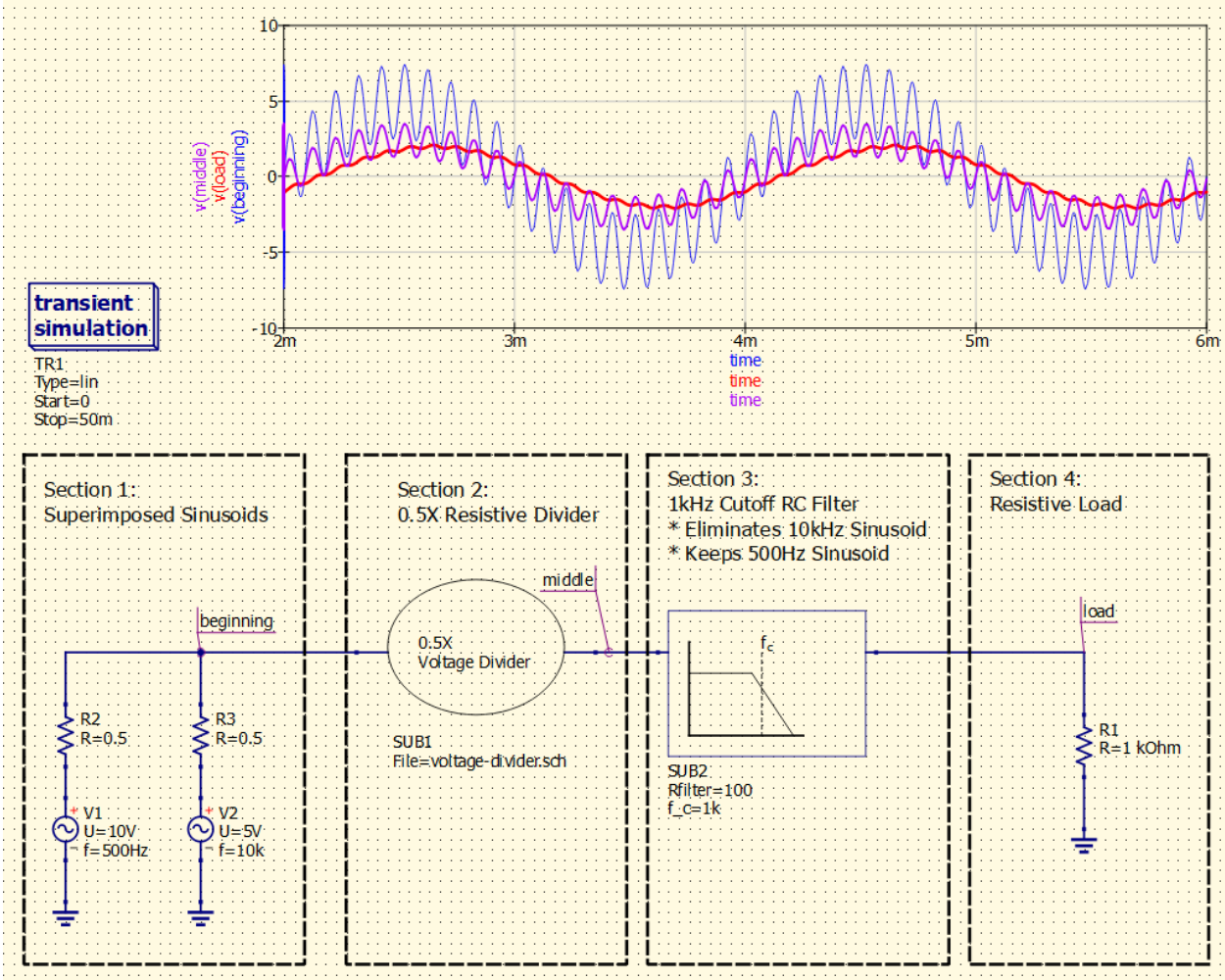


Fig. 1: An example of a "hierarchical design" utilizing Qucs-S subcircuits.
The design uses a subcircuit for a resistive voltage divider, and then uses a parameterized subcircuit for a simple RC low-pass filter. Note that the cutoff frequency and resistance values of the filter are set in the top-level context with the `f_c` and `Rfilter` parameters, instead of being "hardcoded" into the subcircuit itself.

Subcircuits are simply references to another standard Qucs-S Schematic file (`.sch`). Any standard `.sch` can be used as a subcircuit, as long as at least one "port" is added to the schematic.

Warning

This page refers to a different feature from the SPICE `.SUBCKT` directive! Qucs-S subcircuits facilitate hierarchical designs using the standard Qucs-S schematic editor, it is unrelated to the SPICE `.SUBCKT` feature.

10.1 Preparing a Schematic for use as a Subcircuit

The standard Qucs-S schematic editing tools are also used when creating subcircuits (since subcircuits are really just standard `.sch` files being referenced in another schematic). However, there are a few steps to take when preparing a circuit to be used as a subcircuit in a larger schematic.

1. **You MUST create at least one *Subcircuit Port*.** This is your circuit's interface to higher-level schematic files - it is a bidirectional component that allows you to pass signals in and out of your subcircuit. Note that it is NOT necessary to create a port for Ground.
2. **Optionally, you can create a custom symbol for your circuit.** Qucs-S will automatically generate a simple symbol, so this is not a requirement, but it usually improves the usability and readability of your schematic.
3. **Optionally, you may add parameters to your subcircuit.** Qucs-S allows you to pass parameters through into an instance of your subcircuit, from the parent context. This can be useful when reusing a common design multiple times, with slight variations.

Below is a diagram showing the underlying `.sch` files, and their schematics and custom symbols, for the voltage divider/RC filter example from earlier in this tutorial.

10.1.1 Creating Ports

Any Qucs-S `.sch` schematic file can be used as a subcircuit, as long as at least one *Subcircuit Port* is added (creating an externally-accessible interface). This component can be accessed from two places:

- The *Lumped Components* section of the *Components* tab.
- The quick access symbol toolbar at the top of the Qucs-S window.

As an example, we can create a simple RC filter, with an input and output port. This makes it possible to use it as a subcircuit in another schematic. The figure below shows the filter with the necessary *Subcircuit Ports*.

Note that all *Subcircuit Ports* have a *Number* attribute, which is exposed "externally" (when your circuit is used as a subcircuit). The numbers are auto-generated upon placement of the component, but they can be manually edited after the fact through the *Properties* dialog. This can be useful when creating a subcircuit to mimic an off-the-shelf device or integrated circuit, since the pin numbers can be edited to match the real hardware.

The port name (e.g. `P1`, `P2`, etc) is NOT exposed externally. However, if you wish to use text to label a subcircuit port, you can add arbitrary text when creating a custom symbol for your circuit.

Tip

It is not necessary to create a *Subcircuit Port* for your main ground. The standard *Ground* component is considered one single net across the entire simulation; it automatically crosses the boundaries of a subcircuit.

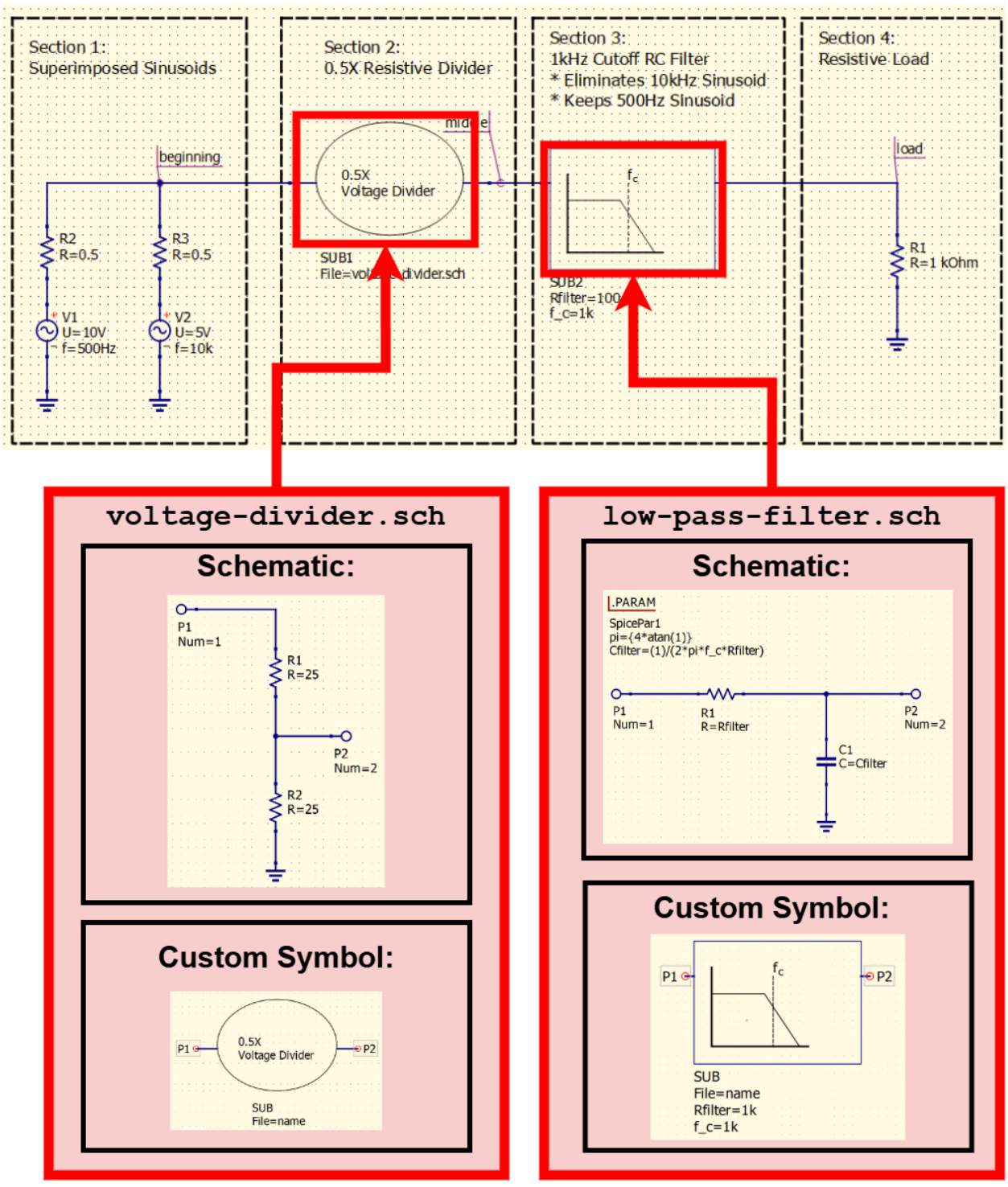


Fig. 2: A diagram showing the underlying files, circuits, and custom symbols involved in the voltage divider/RC filter example.

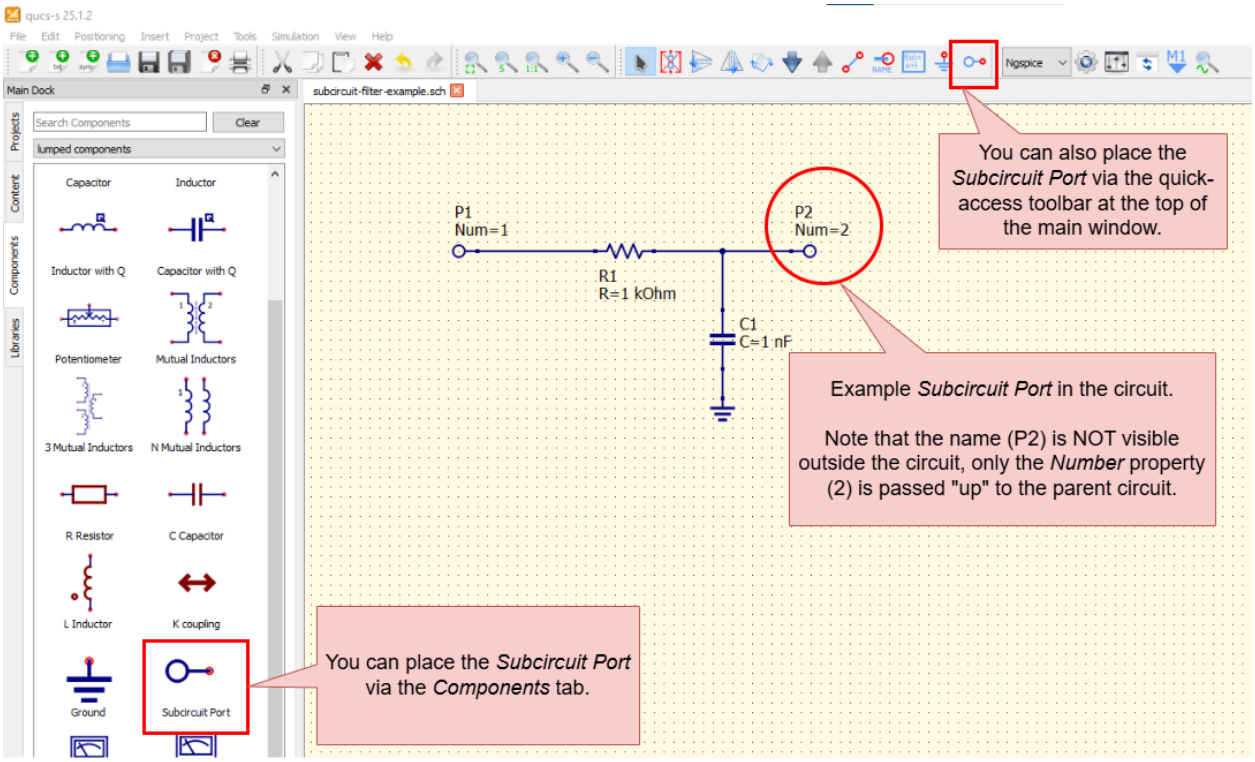


Fig. 3: Using *Subcircuit Ports* to make a schematic usable as a subcircuit.

10.1.2 Customizing a Schematic's Symbol

When used as a subcircuit, a Qucs-S schematic file (`.sch`) does not show its entire contents on the parent schematic page where it's referenced. Instead, it exposes a symbol to represent it. If you do not intervene, Qucs-S will automatically generate a simple rectangular symbol, and distribute the *Subcircuit Ports* across it, as shown in the example below.

The simulation will work using the default auto-generated symbol, but a symbol like this is not very clear. It's difficult to tell what the subcircuit's function is in the parent schematic. For this reason, Qucs-S allows you to customize the symbol that any schematic (`.sch`) exposes when used as a subcircuit.

To access the symbol editor, navigate to your subcircuit `.sch` file in the editor, and navigate to *File* > *Edit Circuit Symbol*. You can also press the `F9` keyboard shortcut. This will open the editor for this schematic's symbol. From this window, you can use the normal page editing tools and the graphics components in the *Paintings* section of the *Components* tab to customize your circuit symbol.

10.1.3 Parameterizing a Subcircuit

10.1.3.1 Creating and Modifying Parameters

When using subcircuits, it is also possible to pass parameters from the parent circuit into the subcircuit. These parameters can be integrated with the standard parameterization features in QUCS-S, such as the SPICE `.PARAM` block commonly used with ngspice.

To create new parameters that your circuit will expose when used as a subcircuit, visit the "Edit Circuit Symbol" dialog (`F9` hotkey). Then right-click the text underneath your circuit symbol, and choose "Edit Properties" as shown in the image below.

Each parameter has 5 properties:

1. `display`: Set to "Yes" to show this parameter by default on your parent schematic page.

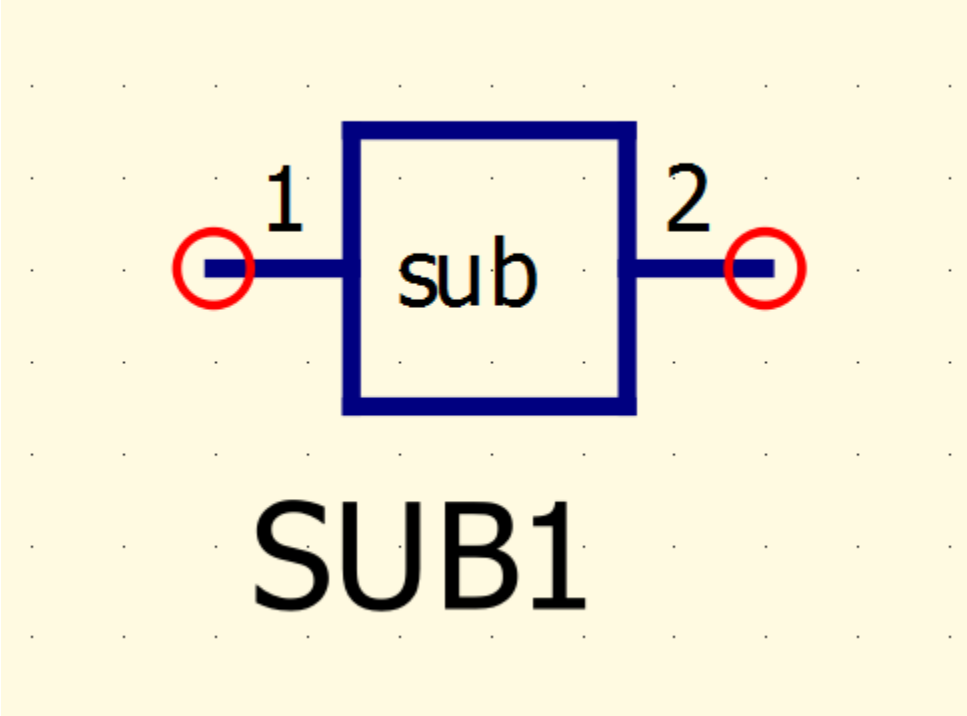


Fig. 4: Example of a subcircuit symbol generated automatically by Qucs-S. These symbols can be customized to enhance readability and clarity.

2. `Name`: This is the actual parameter/variable name, which you'll need to reference inside your subcircuit.
3. `Default`: The default value for this parameter when your subcircuit is initially placed. Of course, you can modify the actual value for each instance of the subcircuit that you place in a parent schematic, this is simply the default/initial value.
4. `Description`: A human-readable, arbitrary comment describing your parameter. This is visible when setting the parameter from a parent circuit. Set this to whatever you like, there is no specific format required.
5. `Type`: This is an additional human-readable, arbitrary comment. You don't have to use it, but it may be useful to note the intended units for a parameter.

All the parameters created here are automatically made available within the subcircuit. The method for accessing them varies slightly depending on simulation backend:

- For Qucsator-based simulations, they are accessed in the same way as *standard Qucsator Equation variables (see the relevant content for more details on syntax)*.
- For SPICE-based simulations (ngspice, Xyce, SpiceOpus), they are accessed in the same way as *`.PARAM` variables (see the relevant content for more details on syntax).*

10.1.3.2 Example Parameterized Subcircuit

As an example, you might wish to create an RC low-pass filter as a subcircuit, and parameterize the series resistance and the desired cutoff frequency. Then, inside the subcircuit, a standard SPICE `.PARAM` component can be used to select the capacitor's value automatically, based on the desired cutoff frequency. An example of such a circuit is shown below.

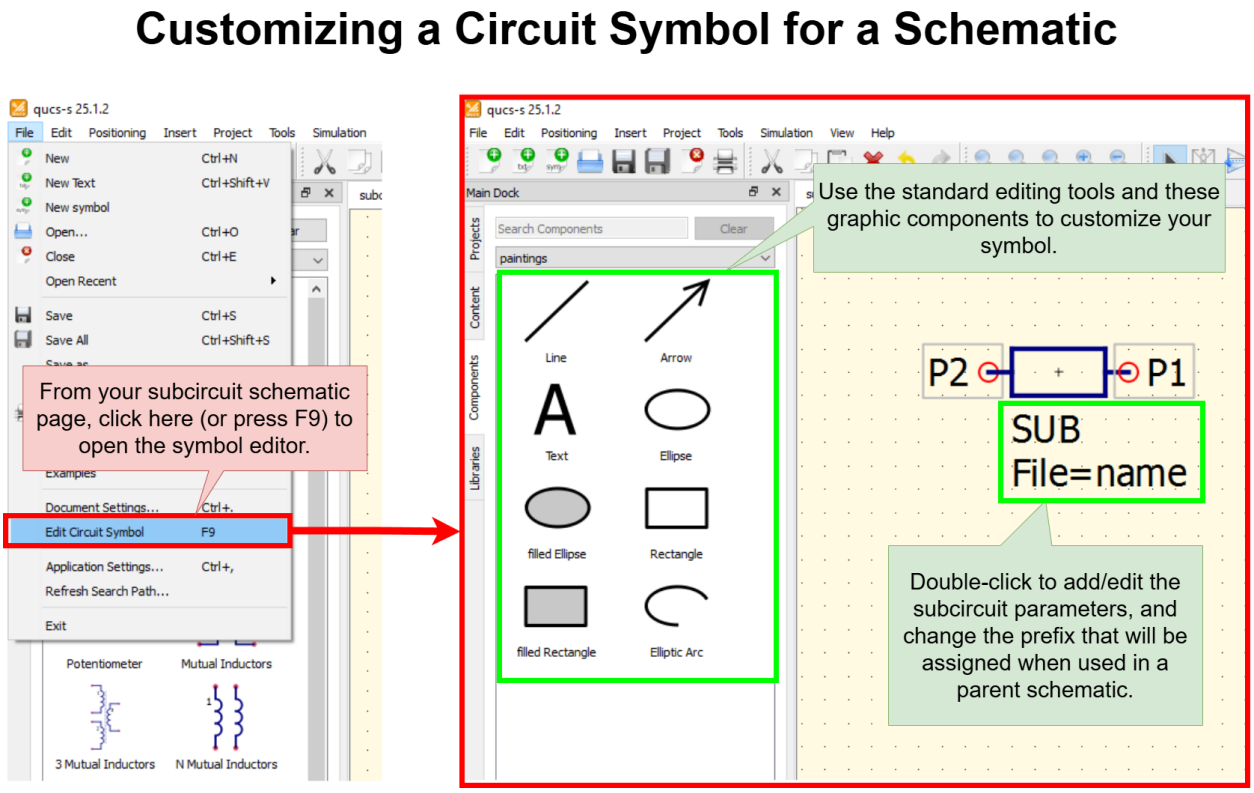


Fig. 5: Navigating to the *Circuit Symbol Editor*. This allows you to customize the symbol that a schematic (`.sch`) file exposes when used as a *Subcircuit* in a parent schematic.

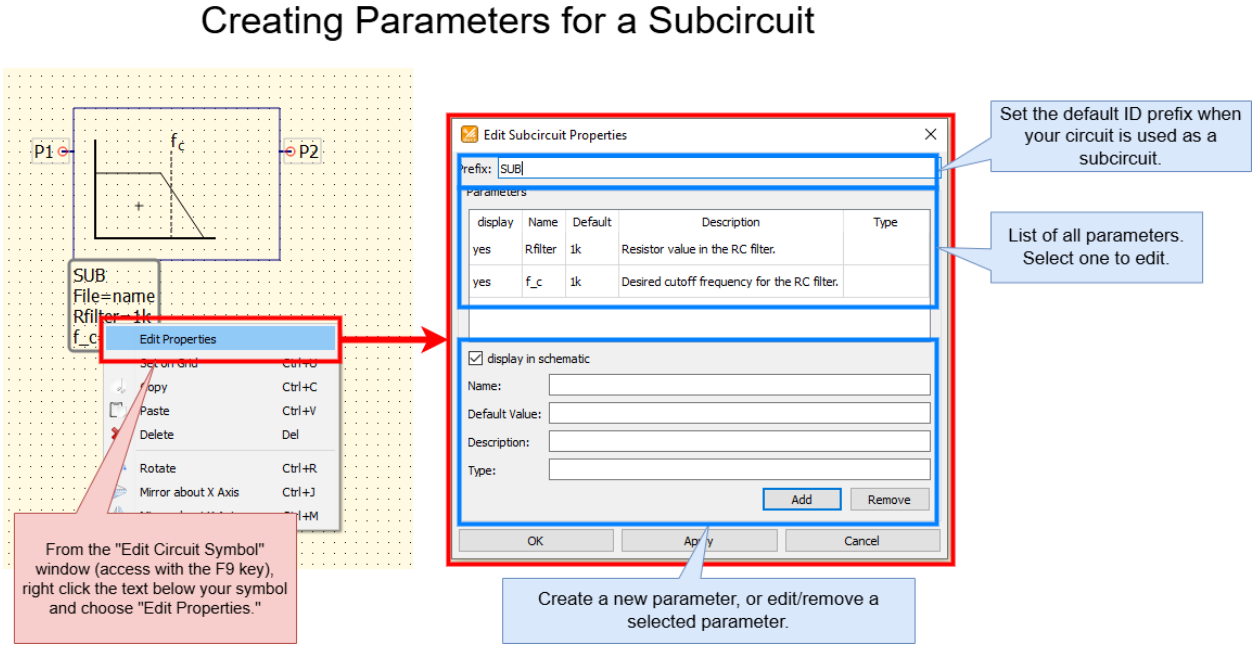


Fig. 6: How to add or modify parameters for your subcircuit to expose to a parent circuit.

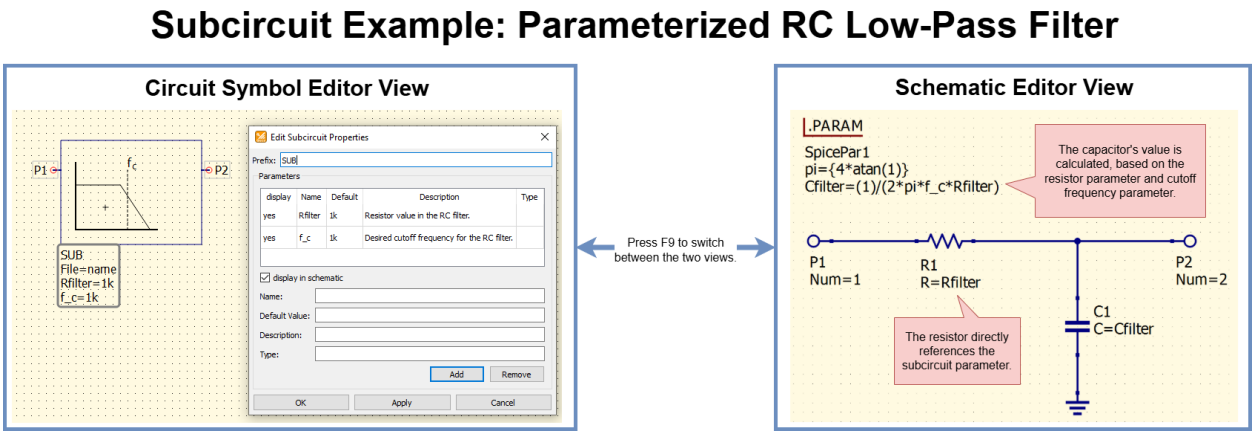


Fig. 7: An example use case for subcircuit parameters. This example shows an RC filter, which has been parameterized to allow passing in a desired series resistance and a desired cutoff frequency from the parent circuit. The capacitor's value is calculated based on the resistance and the desired cutoff frequency.

10.2 Using a Subcircuit in a Schematic

Once you have prepared your `.sch` file for use as a subcircuit, there are two ways to place it into a parent schematic as a subcircuit. See the sections below.

10.2.1 Placing a Subcircuit from the Content tab (recommended)

If the file you want to use as a subcircuit is inside the same QUCS-S project folder as your parent schematic, you can place it conveniently from *the* Content *tab (see the UI overview if you are unsure where this is.)*

To do this, simply navigate to the *Content* tab, and find the circuit you wish to place as a subcircuit under the *Schematics* dropdown. Right-click it, choose "Insert", then left-click anywhere on your schematic page to place it as a subcircuit. You can place multiple instances if you like - press the ESC key when you are done placing instances of the subcircuit.

10.2.2 Placing a Subcircuit with a Custom File Path

If the file you want to use as a subcircuit is *NOT* inside the same QUCS-S Project folder as your parent schematic, use this method to place it, since it will allow to manually specify any arbitrary path to the subcircuit file.

Navigate to the *File Components* section of the *Components* tab, and place the *Subcircuit* component on your schematic page. When it is first placed, it will not perform any function, since it has not been linked to a QUCS-S Schematic (`.sch`) file yet.

Once the *Subcircuit* component is placed, right-click it and choose "Edit Properties". You can select your desired Schematic (`.sch`) file from within the "Edit Properties" window, as shown in the screenshots below.

After you select a `.sch` file, the *Subcircuit* symbol will change to the symbol specified by your file, and any necessary parameters and circuit ports will appear on the page. At that point, the *Subcircuit* is ready to be used in your design.

10.2.3 Manipulating Parameters of a Subcircuit Instance

After placing an instance of your subcircuit in a parent schematic, simply right-click the component and choose "Edit Properties" to modify the parameters (as shown in the figure below). Note that the parameter names, descriptions, default values, and default visibility settings are all passed up from inside the subcircuit.

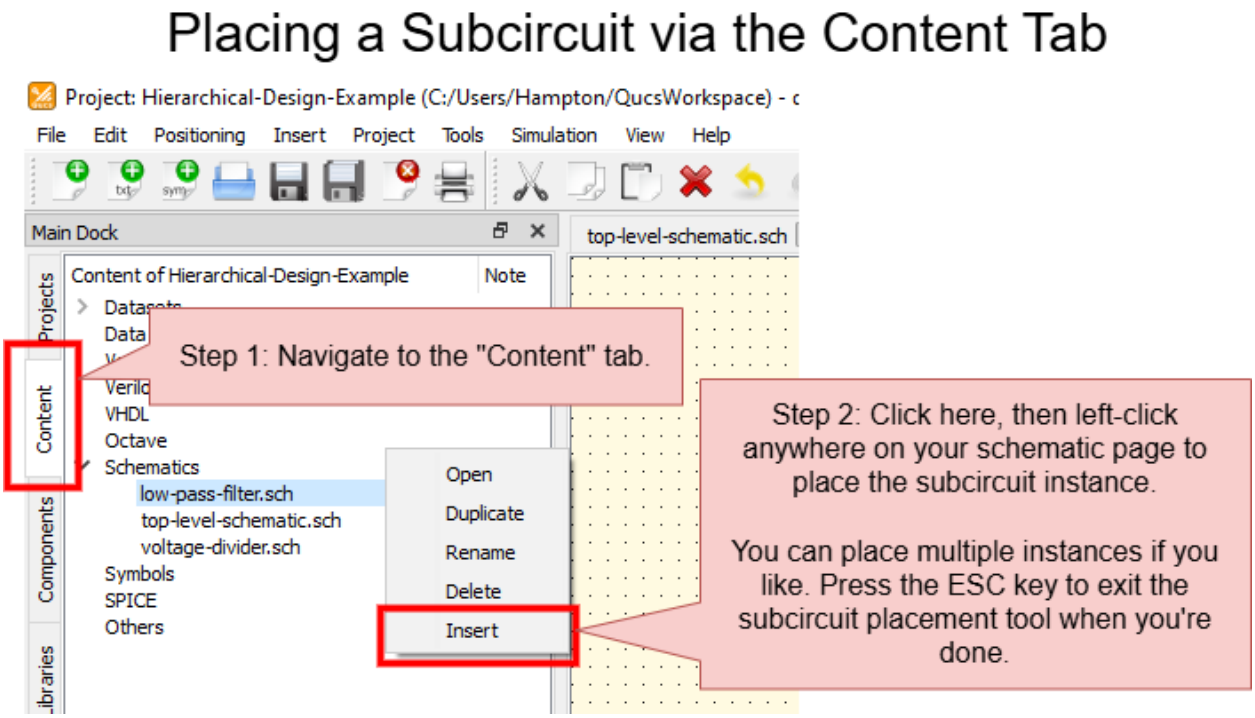


Fig. 8: Accessing and placing a *Subcircuit* on a Qucs-S schematic page, via the *Content* tab.

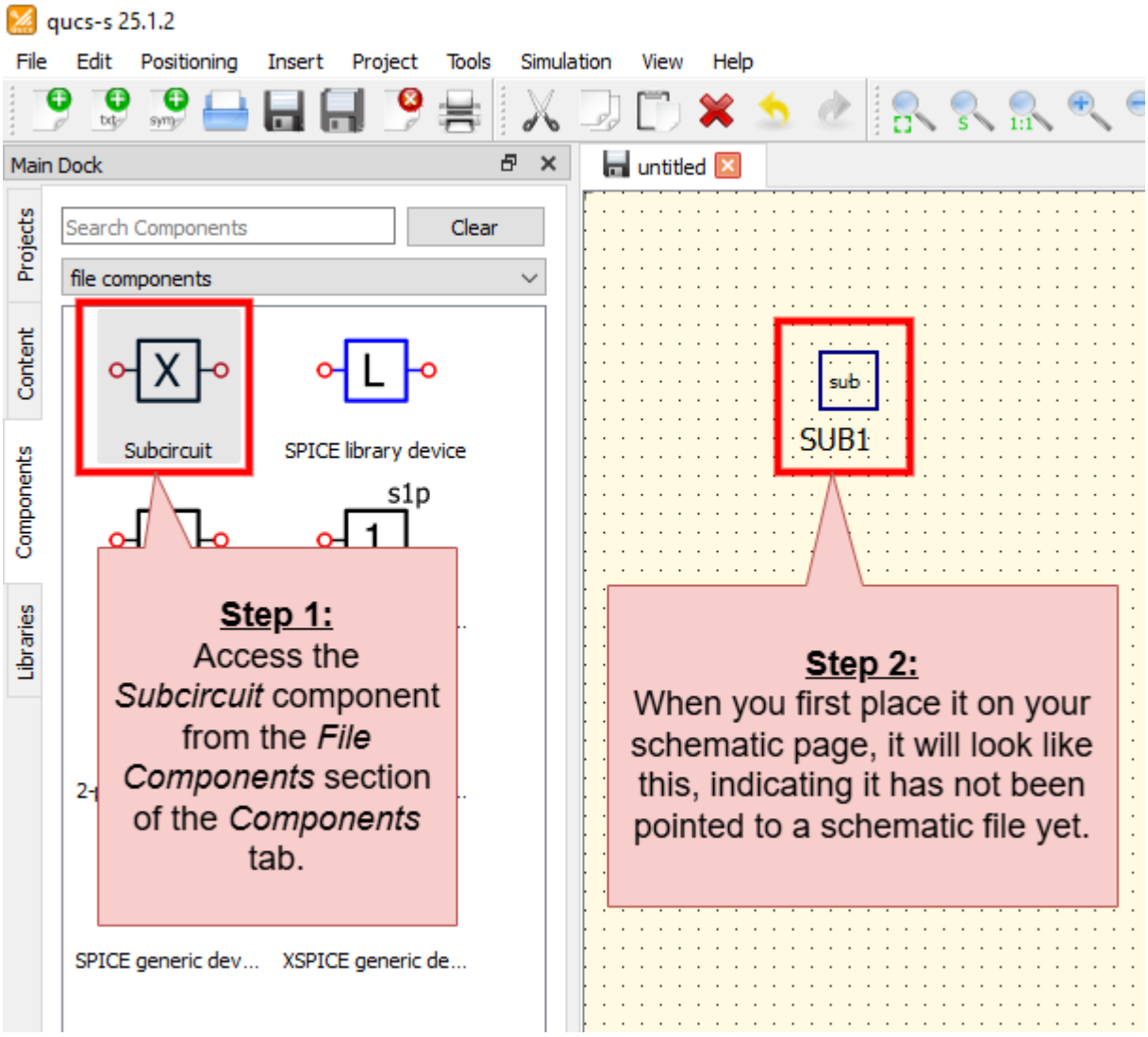


Fig. 9: Accessing and placing a *Subcircuit* on a Qucs-S schematic page, without a particular `.sch` file specified yet.

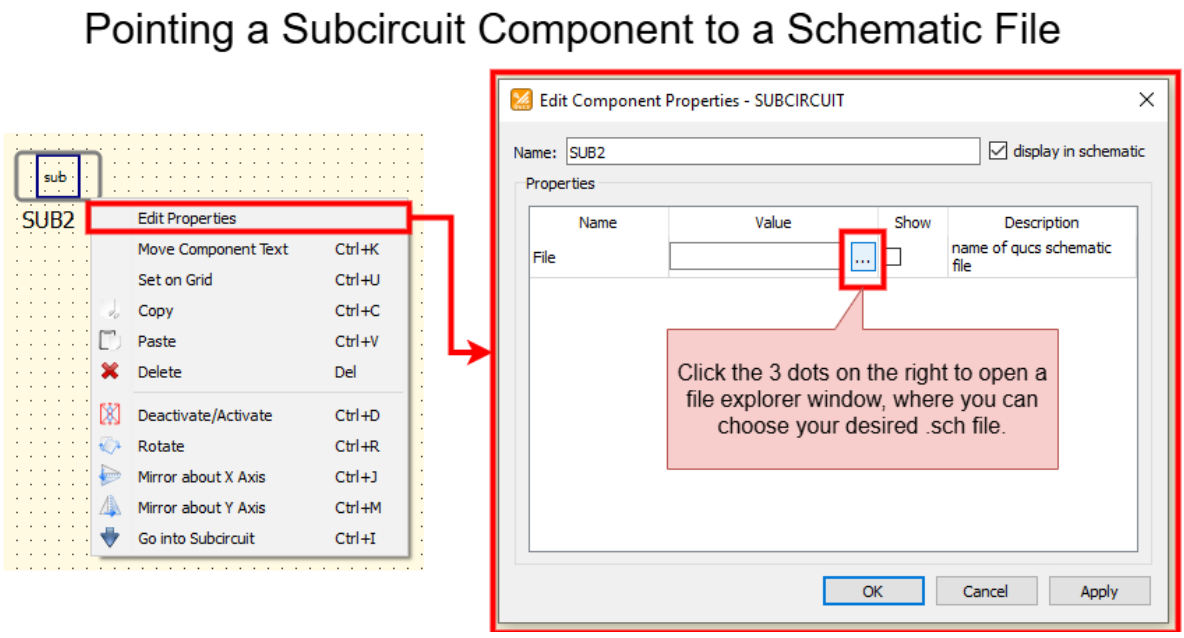


Fig. 10: Pointing a newly-placed *Subcircuit* component to a schematic file.

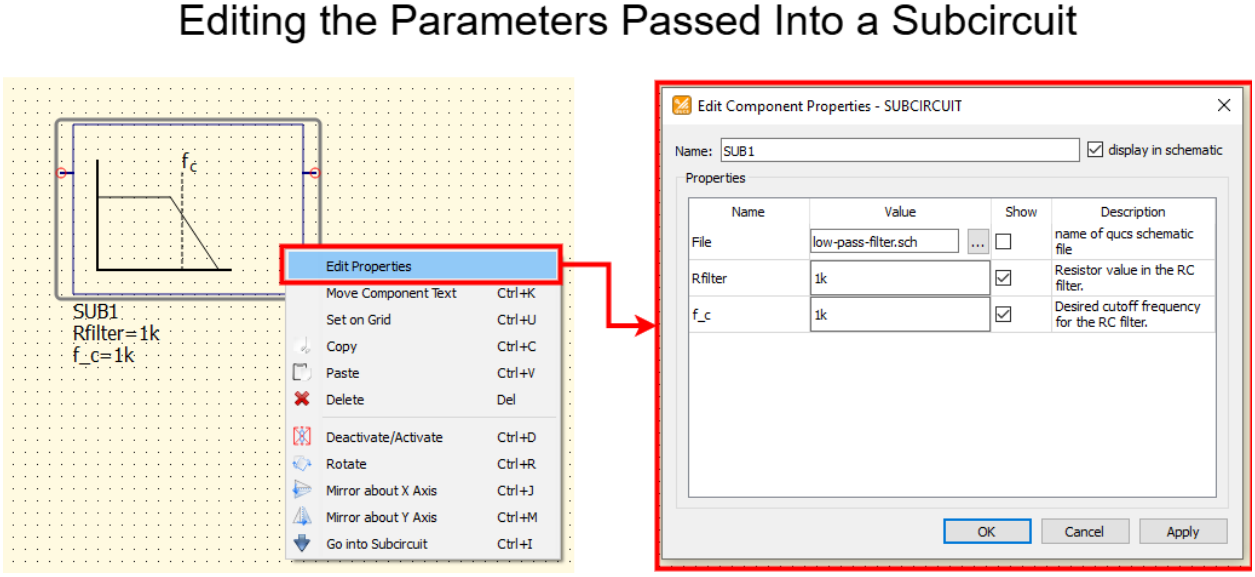


Fig. 11: An example of how to edit the parameters for an instance of a subcircuit, using the "Edit Properties" dialog from within the parent schematic.

CHAPTER

ELEVEN

USING SPICE MODELS

Perhaps the biggest benefit to using the SPICE-based *Simulation Backends* in QUCS-S is the ability to use industry-standard models for real devices. It's very common for electronics manufacturers to provide free SPICE models for their devices, which you can utilize in simulating your designs.

Read on to learn how to bring these models into QUCS-S.

> **Warning**
>
> **The examples in this section of the documentation have all been performed with the ngspice *Simulation Backend*.**
>
> Many of these concepts may also apply to the other SPICE-based backends (Xyce and SPICEOpus), however the syntax may be subtly different in some cases.
>
> None of these examples will work in the QucsatorRF backend (it is not SPICE-compatible).

11.1 Modeling Methods

There are essentially two methods for modeling real devices in SPICE-based simulators:

1. **For discrete semiconductors and passives (FETs, BJTs, diodes, etc), you can *use a* Device Model.**
   - This is most often done with the `.MODEL` directive, although sometimes `.INCLUDE` may be used as well.
2. **For complex devices (like integrated circuits), you can *use a* Subcircuit Model.**
   - This is done with the SPICE `.SUBCKT` directive.
   - Note that this is NOT the same thing as a *QUCS-S Subcircuit!*

11.2 Additional Reading

11.2.1 Using SPICE Discrete Component Models (.MODEL Directive)

11.2.1.1 What is the SPICE .MODEL Directive?

The SPICE `.MODEL` directive allows you to specify the parameters necessary to accurately model simple discrete semiconductor devices. This feature allows you to tune the parameters of a device model that's intrinsically understood by SPICE, such as NPN/PNP transistors, diodes, and various MOSFETs.

For example, below is a model of the popular 2N2222 NPN transistor.

```
.model 2N2222A NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=255.9 Ne=1.307
+ Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1 Cjc=7.306p
+ Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75 Tr=46.91n Tf=411.1p
+ Itf=.6Vtf=1.7 Xtf=3 Rb=10 Vceo=40)
```

Models like these are commonly distributed by electronics manufacturers. Read on to learn how to use them in QUCS-S simulations.

> **Tip**
>
> **A model for the component you want to use may already exist in the out-of-the-box library!** QUCS-S ships with a robust library of models for common parts - check it out before importing additional models.
>
> See the *documentation section on Libraries* for more information on this part of QUCS-S.

11.2.1.2 Importing with "Populate Parameters from SPICE File" (recommended)

> **Warning**
>
> This section depends on a feature that was not added until QUCS-S `v24.3.0` (released July 2024).
>
> **If you are using a QUCS-S version older than `v24.3.0`, this feature is not available!** You will need to use *the more manual method described in the next section.*

For many of *the blue* Universal Components *(see here if you're not sure what this means)*, a SPICE model can be directly imported after placing the component on your schematic. To do this, double-click on the component to open the *Edit Properties* dialog.

Applicable Components

Out of all the *blue* Universal Components in QUCS-S, only the semiconductor devices include the *Populate Parameters from SPICE File* feature. At the time of this writing (July 2025), the latest release is QUCS-S `v25.1.2`, so the following components include this feature:

- `Diode`
- `npn transistor` (with and without substrate connection)
- `pnp transistor` (with and without substrate connection)
- `n-JFET` (with and without substrate connection)
- `p-JFET` (with and without substrate connection)
- `n-MOSFET` (with and without substrate connection)
- `p-MOSFET` (with and without substrate connection)
- `depletion MOSFET` (with and without substrate connection)

All these components are accessible via the *Nonlinear Components* section of the Components Tab.

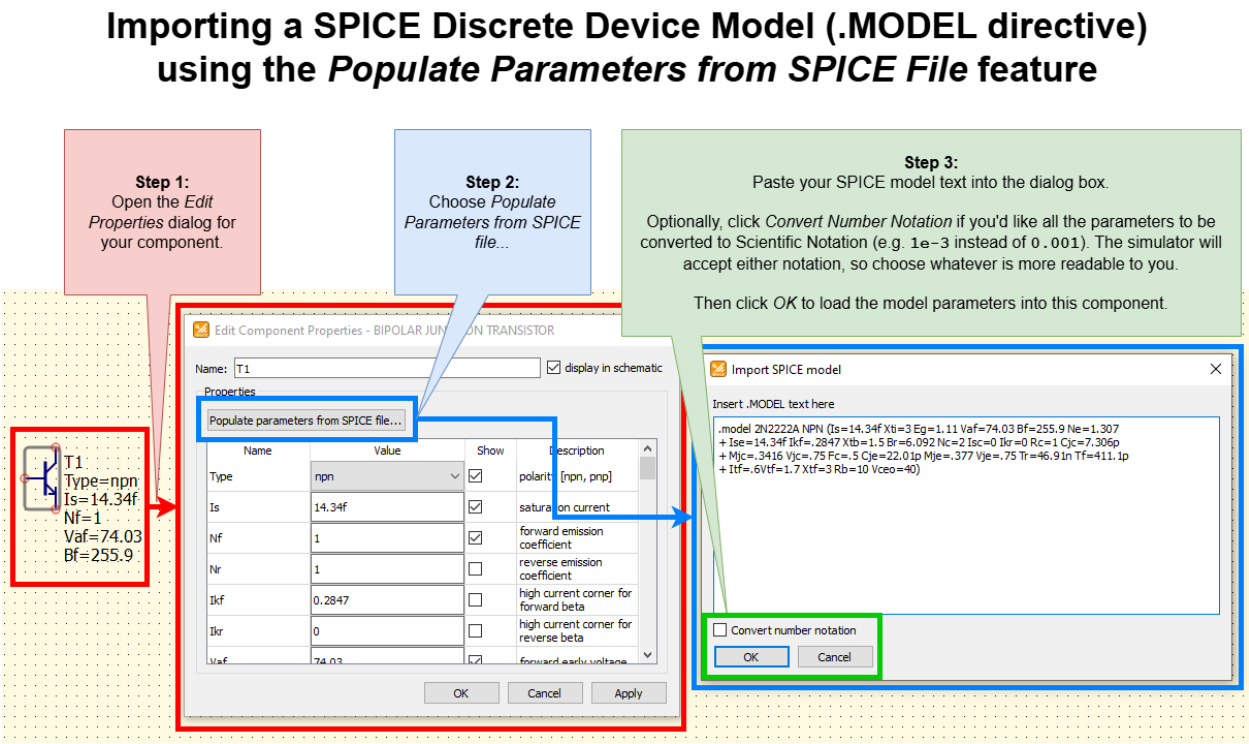


Fig. 1: Annotated screenshots showing how to import a SPICE discrete device model (.MODEL directive) using the new *Populate Parameters from SPICE File* feature (available in QUCS-S `v24.3.0` and later).

This specific example shows how to import the model for the popular 2N2222A for an NPN transistor component.

11.2.1.3 Importing with Manual SPICE Netlist Addition

Let's consider the example of the popular 2N2222A NPN transistor. *The SPICE model for this device is shown in an earlier section.*

You could manually copy and paste each parameter from the SPICE `.MODEL` text into the appropriate place in the QUCS-S component properties. Obviously, this is tedious and error-prone. A more desirable method might be to use special QUCS-S components that can inject additional text into the SPICE Netlist. Two such methods are described in the following sections.

Embedding a Model into a Schematic with *.MODEL Section*

Instead of manually copying/pasting each parameter, you can use the *.MODEL Section* from the *SPICE Netlist Sections* category of components. Placing this on your schematic allows you to define an arbitrary SPICE model for a given device ID.

To use this method, place a *.MODEL Section* on your schematic, and populate it with the text from your model, as shown below.

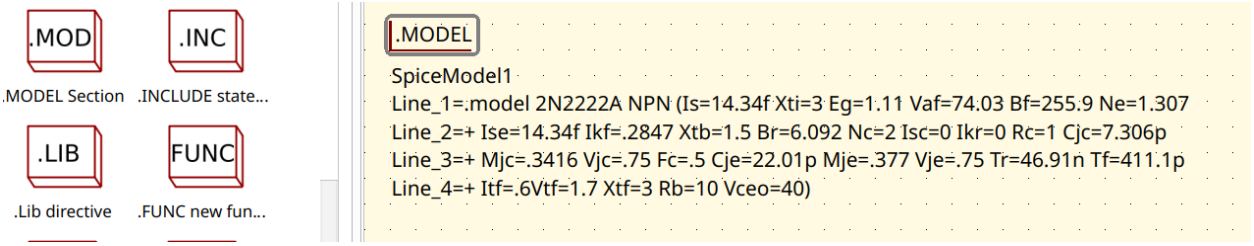


Fig. 2: Example of the *SPICE .MODEL Section* component, placed on the schematic with the model text for the 2N2222A transistor.

Now, to use the model, you will need a special transistor component that supports the full SPICE specification (color-coded red). The blue *Universal* components will NOT work here. See *the Interface Overview section* if you're not familiar with this distinction.

We place a red transistor (Q NPN BJT) on the diagram and enter the model name 2N2222A into its properties. This transistor could be found in *microelectronics* group on the left panel. The screenshot shows a test circuit with such a transistor, which simulates a family of output I-V characteristics.

Referencing an External File with .INCLUDE or .LIB

If you have many models to import, your schematic could quickly get cluttered with *SPICE .MODEL Section* components. To avoid this, you can reference an external SPICE file with one of two methods:

- ***.LIB Section* (Recommended)**: This inserts a SPICE `.LIB` directive, which causes SPICE to *ONLY* parse the `.MODEL` directives in the referenced text file (other directives will be ignored). This is recommended since it limits the possibility for unintended modifications to your circuit with other SPICE directives.
- ***.INCLUDE Section***: This inserts a SPICE `.INCLUDE` directive, which respects the full set of SPICE directives in the referenced file. This will work, but risks confusing behavior if the referenced file contains additional SPICE directives that modify the behavior of your circuit.

Whichever method you choose to use, place the text file containing your model in `$HOME/QucsWorkspace/user_lib` (see *QUCS-S Home Directory* for more details on this location). Then, place either a *.INCLUDE Section* or *.LIB Section* on your schematic, and set it up to reference your text file, similar to the example below.

Once you've done this, you can reference the `.MODEL` identifiers in your schematic components, just like in the `2N2222A` example from the preceding section.

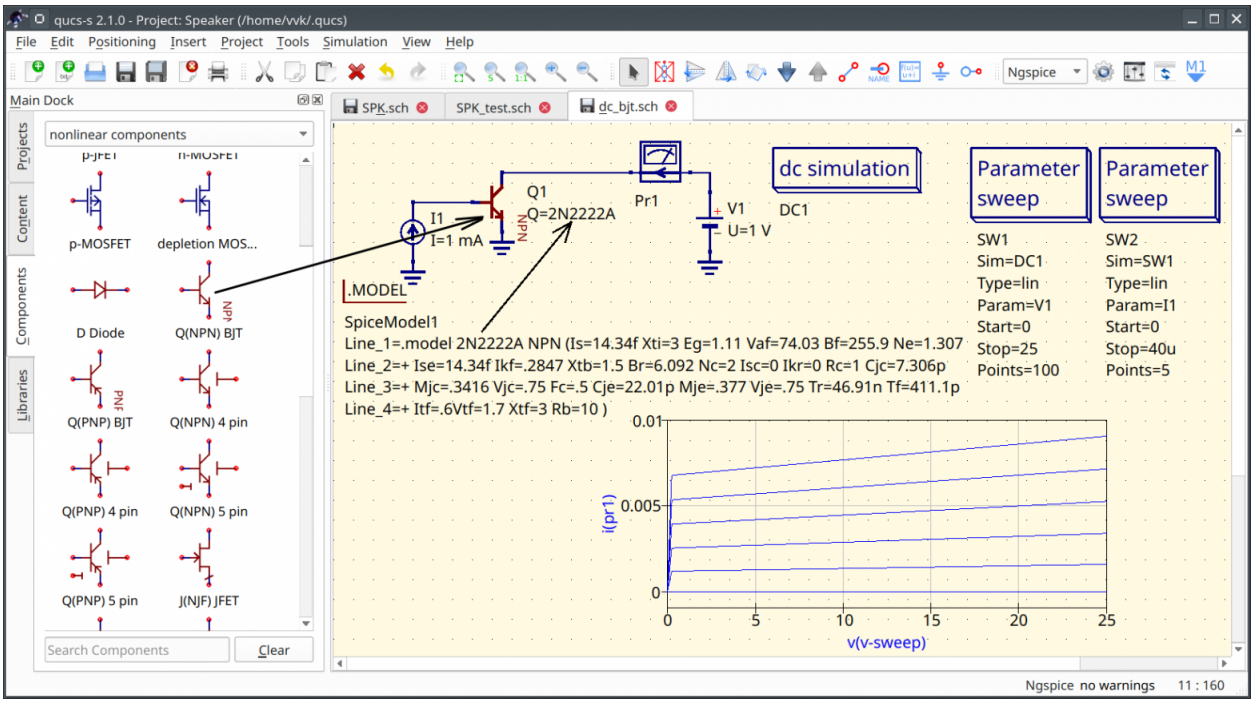


Fig. 3: Example of the *SPICE .MODEL Section* component being used in a schematic. Note that the transistor `Q1` references the model ID `2N2222A` - this is what links the schematic component to the SPICE `.MODEL` text.

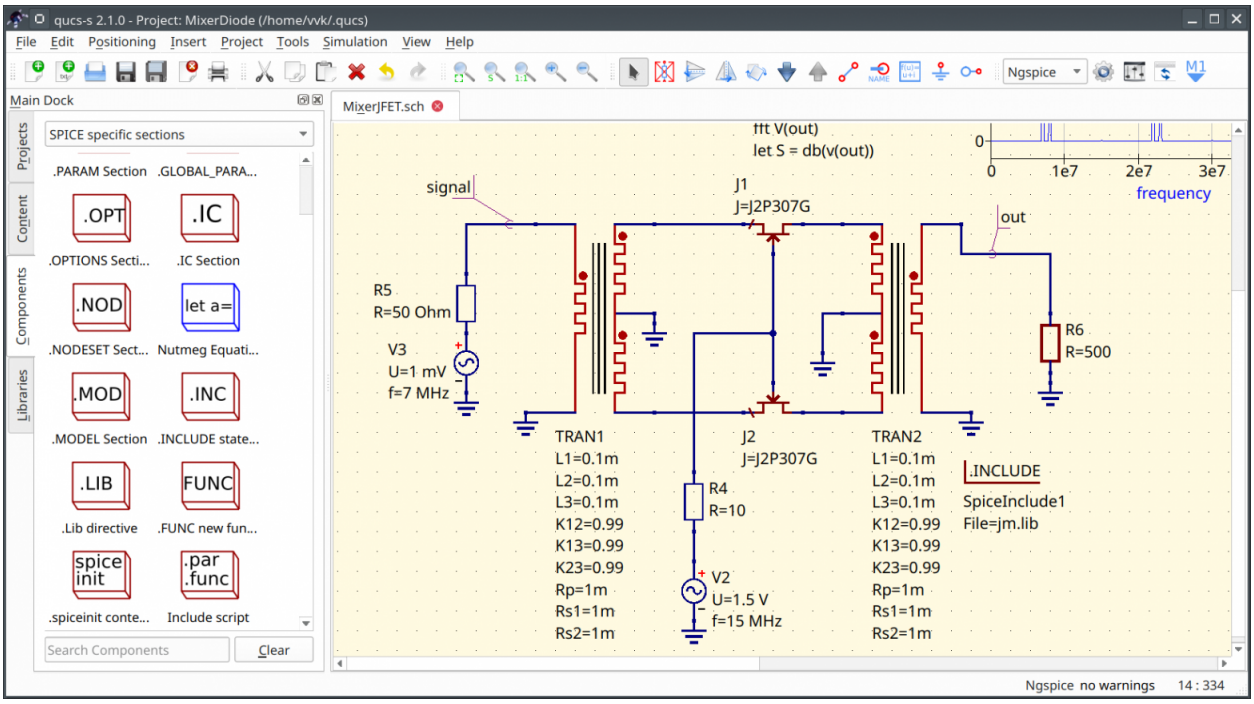


Fig. 4: Example of including an external text file containing a SPICE model (in this case, using the *.INCLUDE Section* component). Note that the transistors `J1` and `J2` reference the model ID `J2P307G`, referring to the `.MODEL` text in the `jm.lib` file.

11.2.2 Using SPICE Complex Device Models (.SUBCKT Directive)

11.2.2.1 Introduction

What is the SPICE .SUBCKT Directive?

> **Warning**
>
> Although their names are similar, the SPICE `.SUBCKT` directive is different from the [QUCS-S Subcircuits] feature!

The SPICE netlist syntax supports subcircuits/hierarchical designs using the `.SUBCKT` directive.

Every `.SUBCKT` block has a name, and exposes some number of "external ports", each with their own integer "port number". The purpose of each port is often described in the comments of the SPICE netlist file containing the `.SUBCKT`.

In contrast to the `.MODEL` directive, which can only adjust the parameters of standard SPICE device models, a `.SUBCKT` can contain an arbitrary circuit with any of the standard SPICE devices, including transistors and diodes with their own customized `.MODEL` parameters. `.SUBCKT`s can also be nested to model more complex devices.

How are .SUBCKT models distributed?

Most electronics manufacturers will provide SPICE `.SUBCKT` models for their integrated circuits (ICs), and even sometimes for complete pluggable electronic modules. These models are simply textual SPICE netlists containing the necessary `.SUBCKT` and other directives, typically with the `.lib` or `.cir` extension.

> **Tip**
>
> Many commercial SPICE-based simulators (such as LTspice or PSPICE) include their own extensions to the standard SPICE netlist syntax.
>
> Unfortunately, some manufacturer-provided SPICE models are designed specifically for one of these commercial simulators, and utilize some of their non-standard syntax.
>
> If you try to run these models in QUCS-S (with ngspice/Xyce/SpiceOpus backends), you may receive some cryptic SPICE netlist syntax errors. You can often get around this problem by *enabling one of the "Compatibility Modes" in your chosen simulation backend.*

Example Model

Below is a model for the popular LM358 operational amplifier.

Note that the identifier of the `.SUBCKT` directive is `LM358`, and the exposed ports are `1`, `2`, `99`, `50`, and `28`. The functions of each port are described in the comments above the `.SUBCKT` directive.

```
*////////////////////////////////////////////////////////////////////////
* (C) National Semiconductor, Inc.
* Models developed and under copyright by:
* National Semiconductor, Inc.

*////////////////////////////////////////////////////////////////////////
* Legal Notice: This material is intended for free software support.
* The file may be copied, and distributed; however, reselling the
*  material is illegal
```

(continues on next page)

(continued from previous page)

```
*////////////////////////////////////////////////////////////////////
* For ordering or technical information on these models, contact:
* National Semiconductor's Customer Response Center
*                 7:00 A.M.--7:00 P.M.  U.S. Central Time
*                                (800) 272-9959
* For Applications support, contact the Internet address:
*  amps-apps@galaxy.nsc.com

*////////////////////////////////////////////////////////
*LM358 DUAL OPERATIONAL AMPLIFIER MACRO-MODEL
*////////////////////////////////////////////////////////
*
* connections:      non-inverting input
*                   |   inverting input
*                   |   |   positive power supply
*                   |   |   |   negative power supply
*                   |   |   |   |   output
*                   |   |   |   |   |
*                   |   |   |   |   |
.SUBCKT LM358       1   2  99  50  28
*
*Features:
*Eliminates need for dual supplies
*Large DC voltage gain =             100dB
*High bandwidth =                    1MHz
*Low input offset voltage =           2mV
*Wide supply range =       +-1.5V to +-16V
*
*NOTE: Model is for single device only and simulated
*      supply current is 1/2 of total device current.
*      Output crossover distortion with dual supplies
*      is not modeled.
*
****************INPUT STAGE**************
*
IOS 2 1 5N
*^Input offset current
R1 1 3 500K
R2 3 2 500K
I1 99 4 100U
R3 5 50 517
R4 6 50 517
Q1 5 2 4 QX
Q2 6 7 4 QX
*Fp2=1.2 MHz
C4 5 6 128.27P
*
***********COMMON MODE EFFECT***********
*
I2 99 50 75U
*^Quiescent supply current
EOS 7 1 POLY(1) 16 49 2E-3 1
```

(continues on next page)

(continued from previous page)

```
*Input offset voltage.^
R8 99 49 60K
R9 49 50 60K
*
*********OUTPUT VOLTAGE LIMITING********
V2 99 8 1.63
D1 9 8 DX
D2 10 9 DX
V3 10 50 .635
*
**************SECOND STAGE**************
*
EH 99 98 99 49 1
G1 98 9 POLY(1) 5 6 0 9.8772E-4 0 .3459
*Fp1=7.86 Hz
R5 98 9 101.2433MEG
C3 98 9 200P
*
***************POLE STAGE***************
*
*Fp=2 MHz
G3 98 15 9 49 1E-6
R12 98 15 1MEG
C5 98 15 7.9577E-14
*
*********COMMON-MODE ZERO STAGE*********
*
*Fpcm=10 KHz
G4 98 16 3 49 5.6234E-8
L2 98 17 15.9M
R13 17 16 1K
*
**************OUTPUT STAGE**************
*
F6 50 99 POLY(1) V6 300U 1
E1 99 23 99 15 1
R16 24 23 17.5
D5 26 24 DX
V6 26 22 .63V
R17 23 25 17.5
D6 25 27 DX
V7 22 27 .63V
V5 22 21 0.27V
D4 21 15 DX
V4 20 22 0.27V
D3 15 20 DX
L3 22 28 500P
RL3 22 28 100K
*
***************MODELS USED**************
*
.MODEL DX D(IS=1E-15)
```

(continues on next page)

(continued from previous page)

```
.MODEL QX PNP(BF=1.111E3)
*
.ENDS
*$
```

11.2.2.2 Importing .SUBCKT Models with "SPICE Library Device" (recommended)

> **Warning**
>
> This section depends on a feature that was not added until QUCS-S `v24.3.0` (released July 2024).
>
> **If you are using a QUCS-S version older than `v24.3.0`, this feature is not available!** You will need to use *the more manual method described in the next section.*

With `.SUBCKT`-based models, you can use the *SPICE Library Device* component to quickly import your model and select a symbol. This component includes templates for common electronics symbols, and the option to input a custom symbol file if you desire.

To get started, navigate to the *File Components* section of the *Components* tab, and place a *SPICE Library Device* on your schematic. Initially, the symbol will be a simple square with no pins, as shown in the screenshot below.

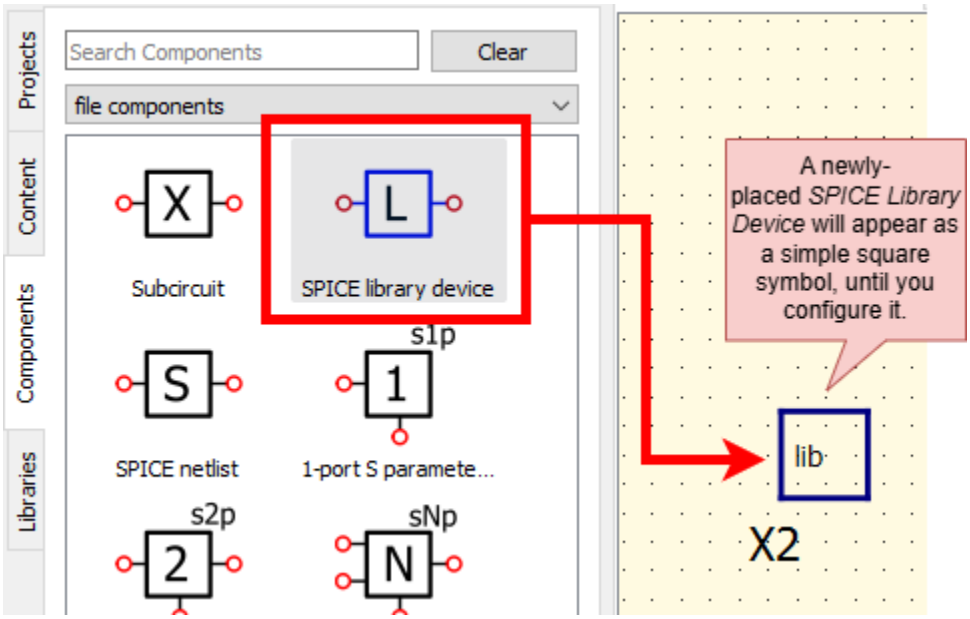


Fig. 5: Annotated screenshot showing how to place a *SPICE Library Device* on a schematic. Note that a new instance of this component will appear as a simple square symbol with no terminals, until you configure it with a real SPICE model and choose a symbol.

To configure the *SPICE Library Device*, double-click the symbol to open its *Properties* dialog. Then:

1. **Select your SPICE model file** (typically `.lib` extension).
2. **If your file contains multiple `.SUBCKT` directives, use the bottom right section of the dialog to select the appropriate one.**
3. **Configure a symbol.** Select from the following options:

- **Automatic Symbol**: This will generate a rectangular symbol, with pin names matching the SPICE `.SUBCKT`. This is usually a good fit for complex integrated circuits which are hard to describe with traditional symbols.
- **Symbol from Template**: This will let you choose from a list of common electronics symbols, including op-amps, transistors, logic gates, and diodes. You can configure the mapping between symbol pins and SPICE `.SUBCKT` pins in the bottom left portion of the window.
- **Symbol from File**: If the previous options are insufficient, you can specify a custom QUCS-S Symbol file (`.sym` extension). *See the next section for instructions on how to create such a file.*

4. **Map your symbol's pins to the appropriate pins in your `.SUBCKT` model.** It is often necessary to read the comments in the SPICE netlist to determine the function of each pin, so a preview of the SPICE model is provided in the bottom right of the window for your convenience.
5. **(Optional) If your model accepts parameters, pass the appropriate parameters in using the "Component Parameters" text box.**
   - Not all models require parameters. You can tell if your `.SUBCKT` accepts parameters by looking for parameter definitions right after the main `.SUBCKT portname1 portname2` portion of the SPICE file. See the relevant sections of the ngspice manual for examples of a `.SUBCKT` with parameters.
   - The standard SPICE parameter syntax applies. *See the section on Equations and Parameters for more information.*

The annotated screenshot below highlights each of the major sections of the configuration dialog, with an example LM358 op-amp model selected.

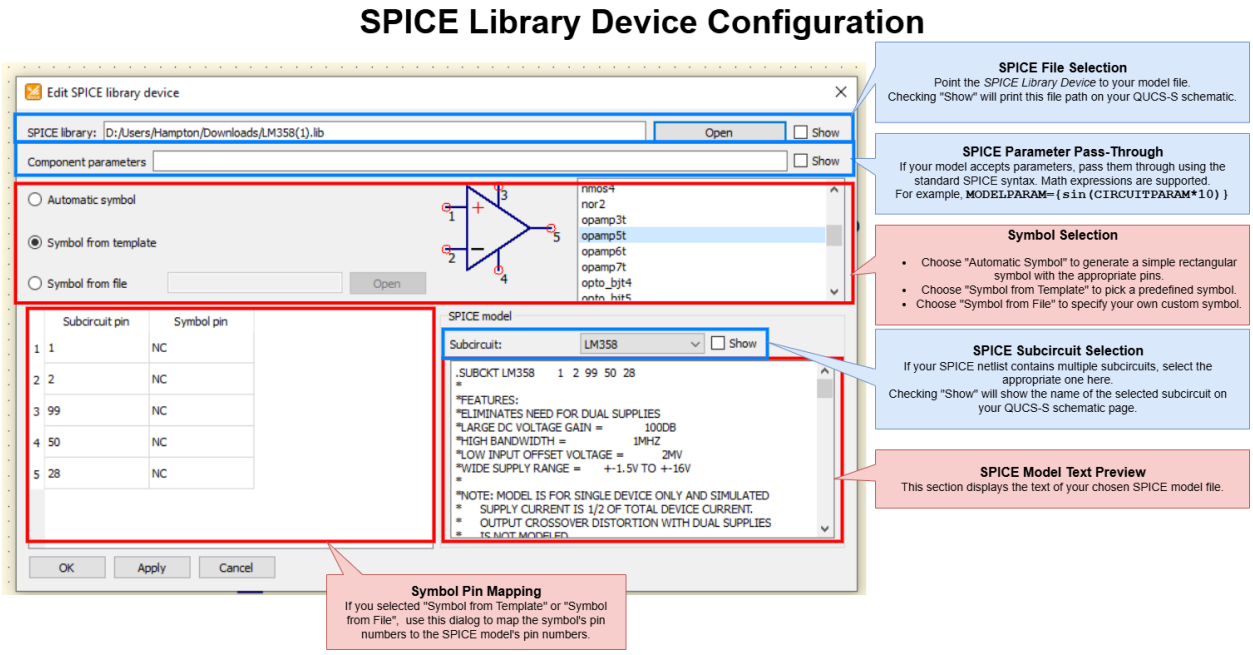


Fig. 6: Annotated screenshot showing the configuration options for a *SPICE Library Device*, using the common LM358 op-amp as an example.

In the case of the LM358, the `opamp5t` symbol in the "Use Symbol from Template" feature is a good fit. After mapping the terminals and clicking *OK* to exit the *SPICE Library Device* configuration dialog, the schematic symbol now looks like an op-amp, and the device is usable in the schematic.

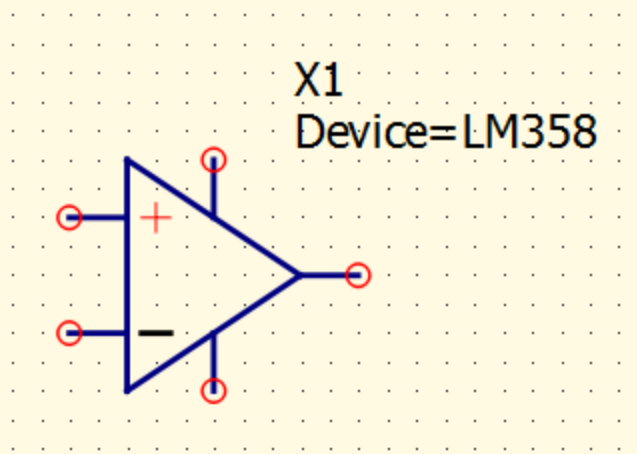


Fig. 7: The *SPICE Library Device* from the previous examples, after configuring with the LM358 model file, appropriate symbol, and appropriate symbol pin mapping. This device is now usable within the QUCS-S schematic.

Creating a Custom Symbol File

In some cases, you may wish to create a custom symbol for a *SPICE Library Device* instead of relying on the auto-generated symbols or the pre-made templates. To do this, create a new *Symbol File* using the button on the toolbar. This will open a new document, as shown below, with a default square symbol (the same as the default symbol for a *QUCS-S subcircuit*).

Use the "Paintings" components to create the graphics of your symbol. To add electrical ports, use the *Insert Port* button, just like when you *edit a QUCS-S subcircuit symbol.*

When you are happy with the symbol, save it, and return to your main schematic. Open the configuration for your *SPICE Library Device*, and point it to your new custom symbol file. *See the earlier section on configuring SPICE Library Devices for more details.*

11.2.2.3 Importing .SUBCKT Models with "SPICE File Component"

> **Warning**
>
> **If you are using QUCS-S version `v24.3.0` (released July 2024) or later, see *the previous section* for a much more efficient method of using `.SUBCKT` models!** The method described below will still work in newer versions of QUCS-S, but it generally offers no advantage over the new *SPICE Library Device* method.
>
> If you are using a version of QUCS-S prior to `v24.3.0`, the method described in this section is the *only* method of using `.SUBCKT` models.

Part 1: Using "SPICE File Component"

The first part of the process is to place a *SPICE File Component* in your schematic. Generally, you'll want to create a new, empty schematic (`.sch`) for this purpose, since you'll need to use this schematic as a subcircuit if you wish to customize the symbol.

1. Ensure your SPICE model file is saved in a convenient location on your filesystem. In this example, it's saved in `/home/vvk/LM358.cir`.
2. Using the *File Components* section of *the* Components *tab*, place a *SPICE File Component* on your schematic page. Once you place it, it should appear as a square symbol with no pins.

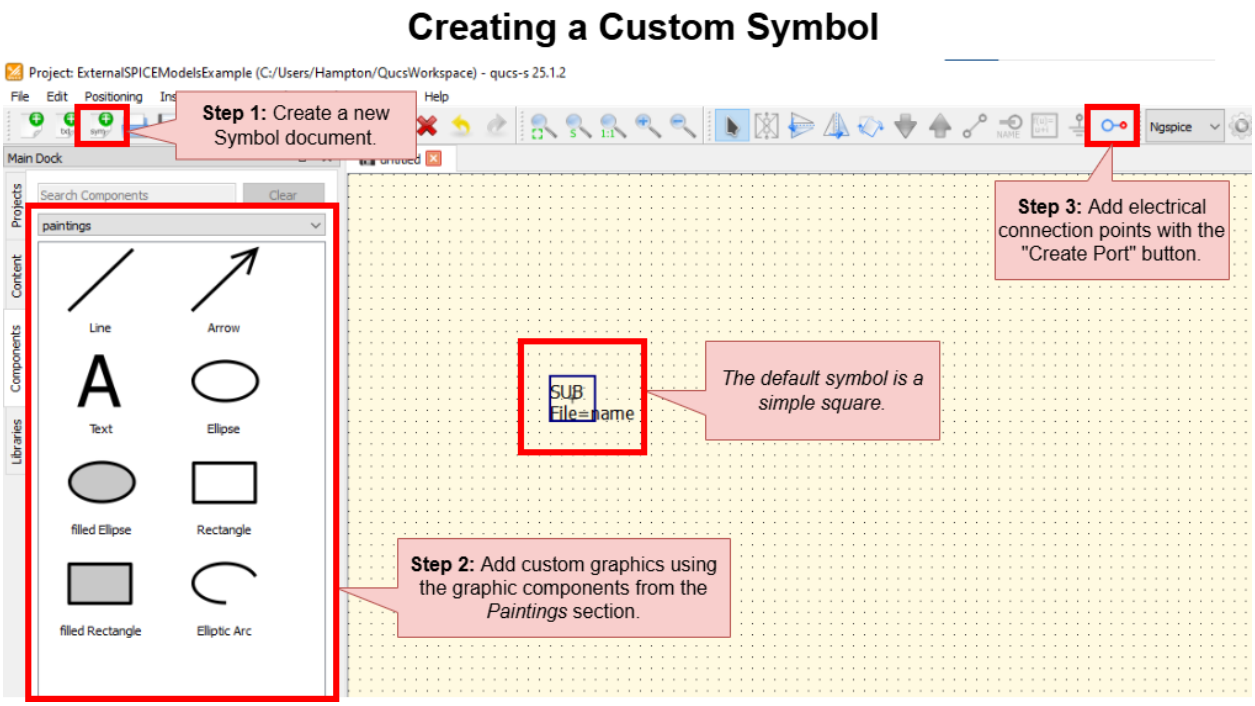
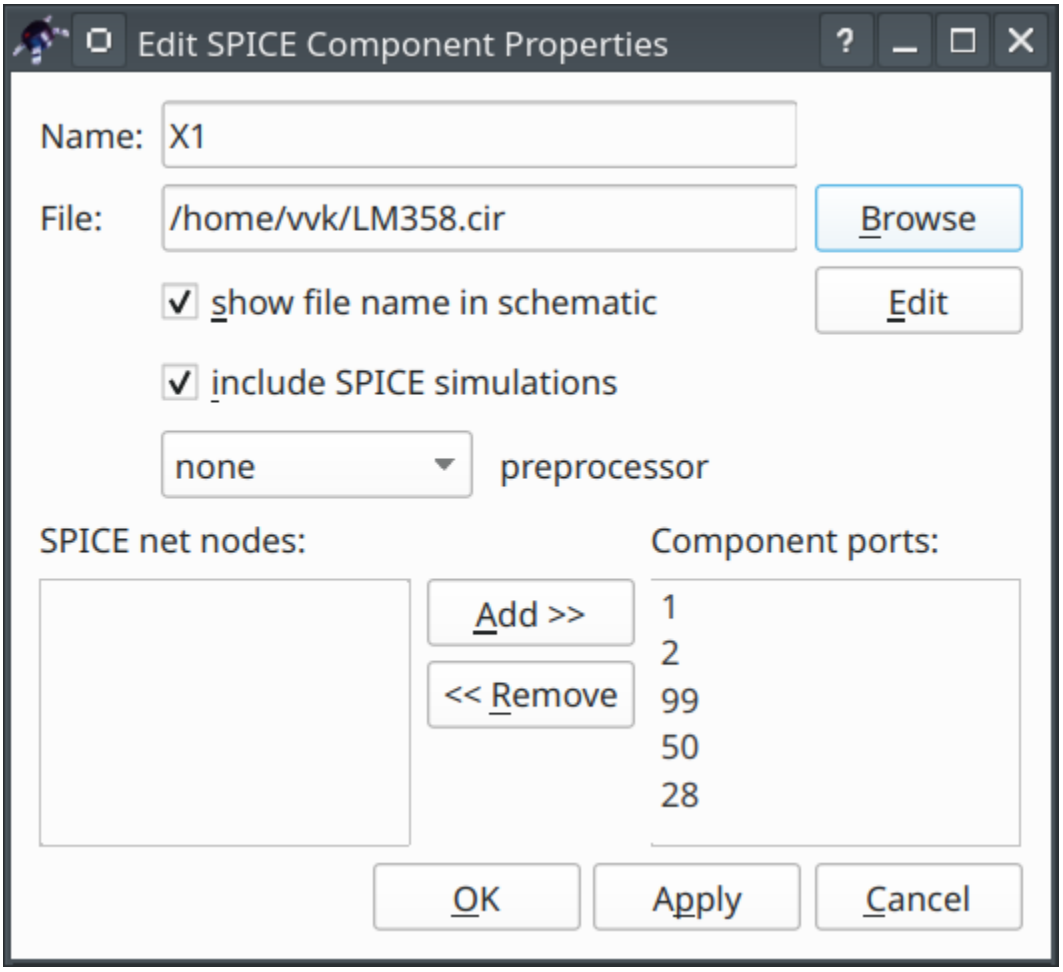


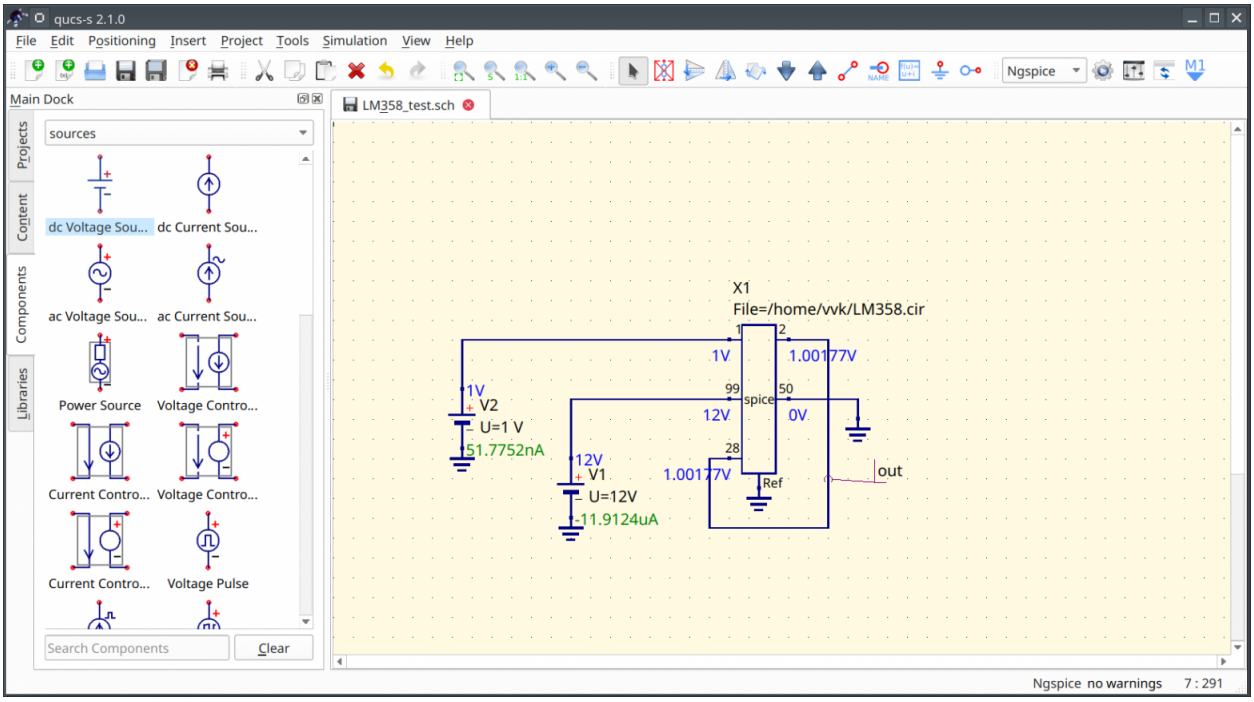
Fig. 8: Creating a custom symbol document in QUCS-S. These symbols can be used with SPICE library devices (among other use cases).

3. Double-click the newly-placed *SPICE File Component* symbol to open its *Component Properties* dialog. Specify the path to your SPICE model file by clicking the *Browse* button.

4. Once you select your model, you'll see a list of the model's ports appear in the *Component Properties* dialog. In most cases, you'll want to simply add all the model's ports to the "Component Ports" list, which makes them appear in the QUCS-S symbol. After completing steps 1-4, your *Component Properties* dialog should look like the screenshot below.



5. Once the component is configured, click OK to exit. The symbol should have changed - it should now be showing all the terminals you added to the "Component Ports" list. At this point, the model is usable in simulations. A simple *DC operating point simulation* has been performed in the screenshot below, which confirms the model is functioning correctly.

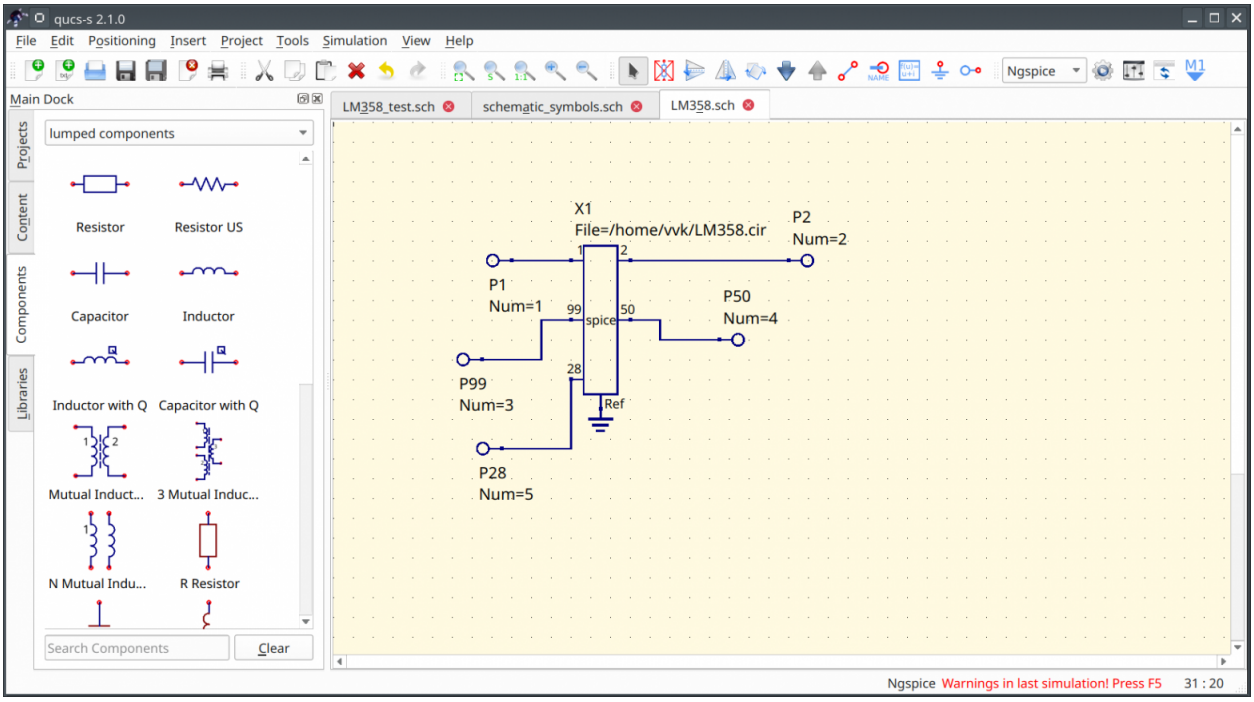


At this point, the model is usable for simulations. However, if you wish to create a customized symbol, continue on to the next section.

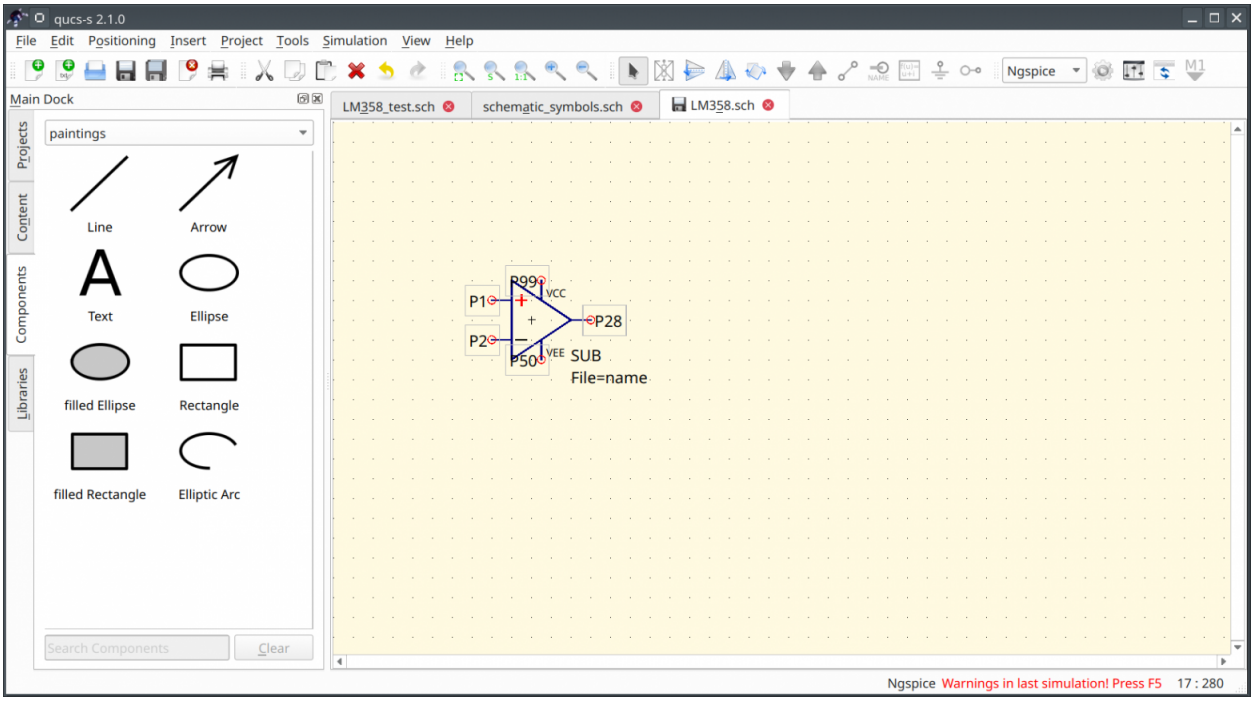
Part 2: Custom Symbol by Wrapping in a QUCS-S Subcircuit

The *SPICE File Component* offers no direct provision for customizing its symbol. It simply auto-generates a basic rectangular symbol. To create a custom symbol, the *SPICE File Component* will need to be wrapped in a *QUCS-S Subcircuit,* and then the subcircuit's symbol can be customized to suit your needs.

To do this, first ensure your *SPICE File Component* is the only component on your schematic. Then add the necessary QUCS-S Ports for it to function as a subcircuit. The end result will be something like the screenshot below.



After doing that, press `F9` to enter the symbol editor for your `.sch` file. Draw your desired symbol here (*see the subcircuit documentation for more details on how to do this*). An example op-amp symbol is shown below.



After doing that, simply place an instance of your new subcircuit anywhere you want to use your SPICE model. It will appear with your custom symbol, and under the hood it will still be referencing the SPICE model you configured in the *SPICE File Component*. An example simulation using this method is shown below.

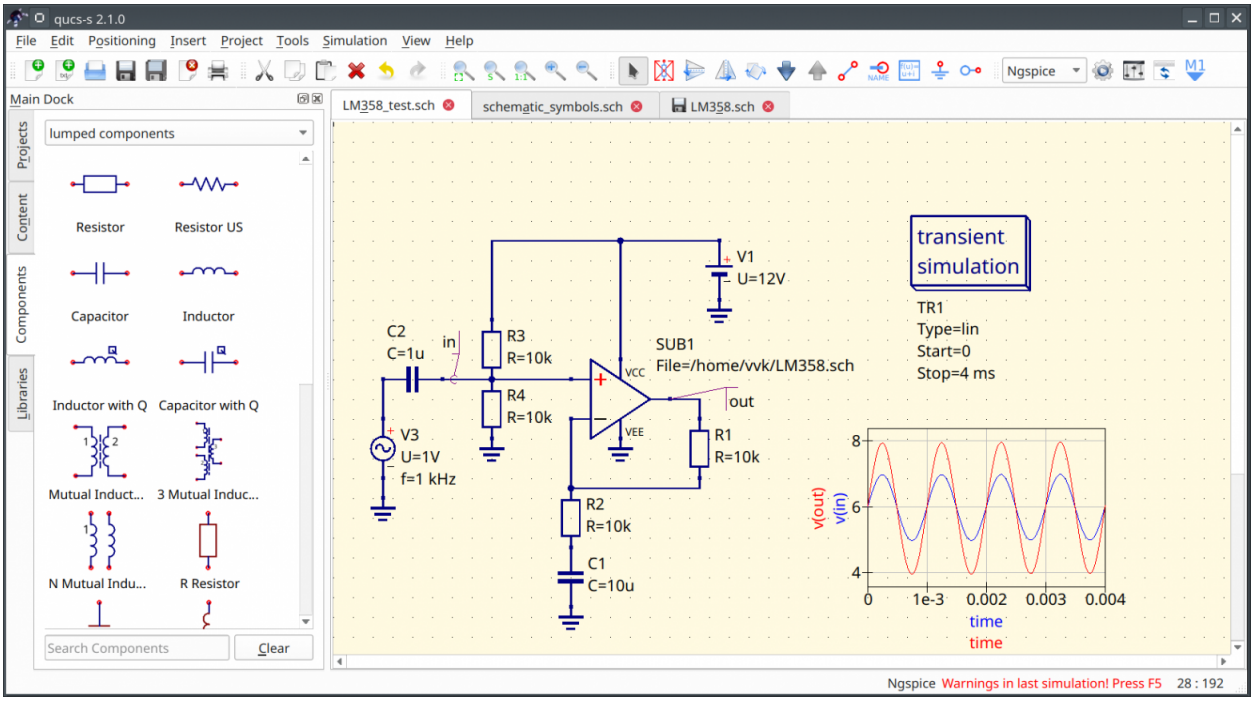


Fig. 9: An example of a simulation using an external SPICE model, achieved with the "*SPICE File Component* wrapped in a QUCS-S subcircuit" method.

11.2.3 SPICE Model Compatibility Modes & Troubleshooting

11.2.3.1 Introduction

The netlist syntax in most modern SPICE simulators, including ngspice, is compliant with the original SPICE `3f5` standard (the last version of Berkeley SPICE, released in 1993). However, most simulators have continued adding new features on top of the original SPICE standard. For example, the popular LTspice netlist syntax supports the `IF` operator, while Ngspice implements a ternary operator instead.

Unfortunately, many manufacturers release device models that utilize the special features of a particular modern SPICE simulator. This gives additional flexibility and more sophisticated modeling, but it can also cause models to break when used with simulators other than the one they were originally designed for.

ngspice Compatibility Modes

To improve model compatibility, ngspice offers various "compatibility modes", which allow netlist syntax from other simulators to operate in an ngspice simulation. Multiple different compatibility modes can be enabled at the same time, and compatibility modes can be limited in scope to certain portions of the netlist if you desire. See the Compatibility section of the ngspice manual for details.

One of the most common compatibility mode commands is to add the following to the `.spiceinit` file:

```
set ngbehavior=ltpsa
```

This command has 4 parts:

- `set ngbehavior=` : Set the compatibility mode variable
- `lt` : Enable LTspice compatibility mode
- `ps` : Enable PSPICE compatibility mode

- `a` : Make any compatibility modes apply to the entire SPICE netlist

Please note that the compatibility modes are not 100% effective, since it's impossible for ngspice to keep pace with commercial simulator development perfectly.

Enabling ngspice Compatibility Mode in QUCS-S

In QUCS-S, ngspice's Compatibility Modes can be enabled for the entire QUCS-S window (with no change to your project files), or they can be enabled for a particular schematic.

Enabling Globally (Application-Wide)

If you wish, one of the ngspice Compatibility Modes can be selected from the *Simulator Settings* window. You can access this dialog from the top toolbar (`Simulate -> Simulator Settings`).

Note that with this method, only one compatibility mode can be selected at a time. If you want to enable multiple compatibility modes at a time, *see the method described in the next section.*

Enabling for a Particular Schematic

To add an *ngspice compatibility mode command* to a particular QUCS-S schematic, place a *.spiceinit* component on your schematic. This component is available from the *SPICE netlist sections* category of the *Components Tab*.

> **Warning**
>
> If you are using hierarchical design/QUCS-S subcircuits, place the *.spiceinit* component in the top-level schematic (not the lower-level subcircuit schematics) or you may get unpredictable behavior.

An example is shown below, using a model of the TDA2003 amplifier which is designed for the LTspice simulator.

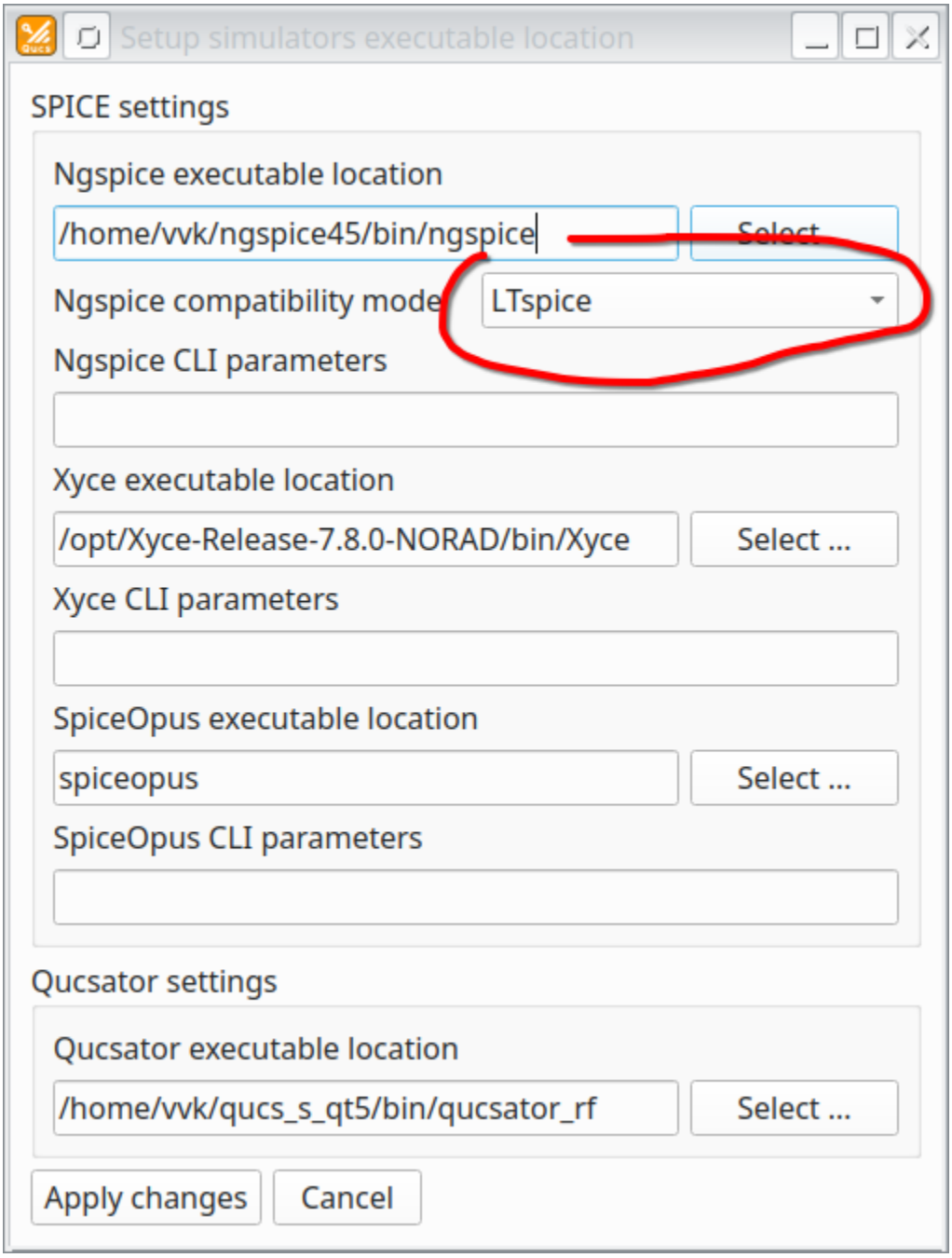


Fig. 10: A screenshot of the *Simulator Settings* dialog, with the dropdown to choose an ngspice Compatibility Mode highlighted.

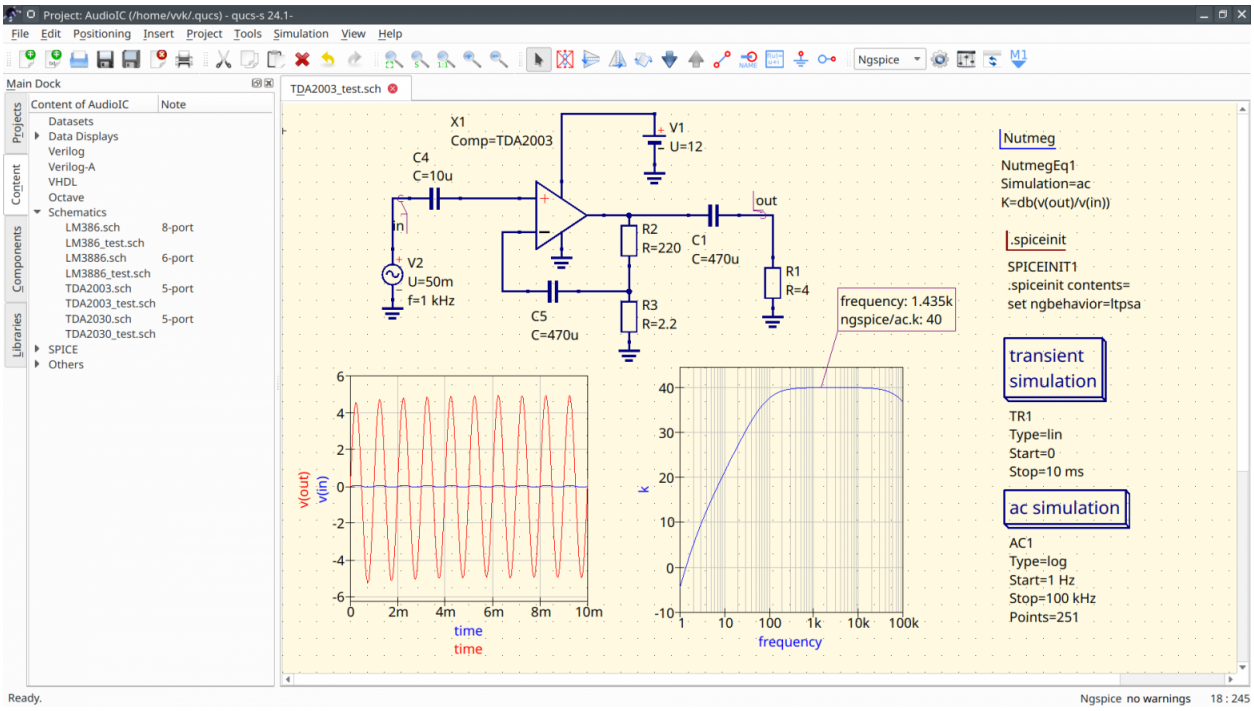


Fig. 11: An example of a simulation with an ngspice Compatibility Mode enabled. The model for the TDA2003 amplifier is designed for the LTspice simulator, and will not work in ngspice unless Compatibility Modes are used. Note the `set ngbehavior=ltpsa` command (enabling both LTspice and PSPICE compatibility mode for the entire simulation netlist) in the special *.spiceinit* component, on the right side of the schematic.

CHAPTER

TWELVE

INTRODUCTION TO LIBRARIES

12.1 Purpose of Libraries

To simulate "real-world" circuit designs, it is usually necessary to incorporate models of "real-world" components. For example, you may need a model for a specific voltage regulator IC from Texas Instruments, or a specific diode from OnSemi (as opposed to a genericized/ideal model).

Bringing external SPICE models into QUCS-S simulations was discussed in earlier sections of the documentation. But for common parts used across many projects, it's a waste of time to go back through the process of importing the model and configuring the symbol for each project. This is where Libraries come in - they allow you to build a collection of frequently-used "real-world" models, complete with all the necessary symbols to use them in QUCS-S simulations.

12.2 How Libraries Work

You can think of QUCS-S Libraries (`.lib` files) as "compiled" products. Creating a Library is a one-way process, just like compiling source code. If you wish to update a library, you cannot do so with just the "compiled" library file, you need the "source code".

The input to the "compilation" (i.e. the "source code") is *a standard QUCS-S "Project"*, containing multiple `.sch` files which are *set up as Subcircuits*. There should be one of these `.sch` files for each Device you want in your Library. The Subcircuits should have all the necessary symbols and ports configured - all the Library will do is make it easier to drop these Subcircuits into your projects.

Tip

Generally speaking, there's no special/quick process to generate these Subcircuits for use in a Library, especially if you are importing models that utilize a SPICE `.SUBCKT` directive. It's a manual process, and you cannot use *the new "SPICE Library Device" component to accelerate the process.*

The only time where there *is* a valid "shortcut" to create a Library is if you have SPICE Discrete Device models (i.e. the `.MODEL` data) for your devices. In that case, you can *use the "Convert Data Files" utility.*

You can follow *this section of the Subcircuit documentation* to learn how to create a subcircuit with a customized symbol based on an external SPICE model.

The output of this "compilation" process is the Library file (`.lib` file), which contains all the Subcircuits of the source Project.

Warning

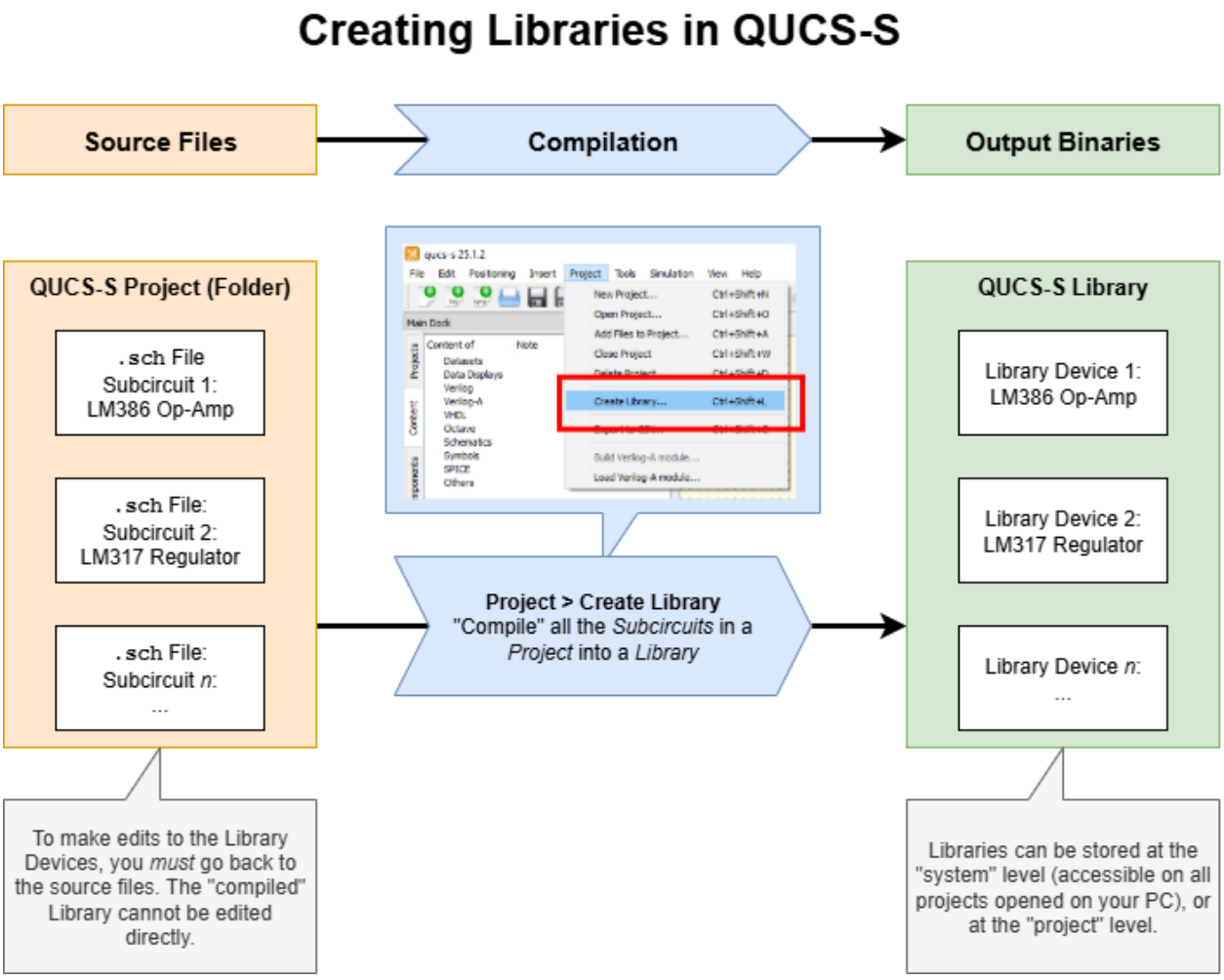


Fig. 1: Process diagram, showing the one-way process of creating *Libraries* from *Projects* containing *Subcircuits*.

While the Library syntax is XML-like, the exact syntax is not documented at this time, and the files are not intended to be human-editable. Attempting to manually edit a "compiled" Library file may result in unexpected behavior.

CHAPTER

THIRTEEN

CREATING LIBRARIES

There are two methods to create Libraries in QUCS-S:

1. ***Create a Library Manually, with fully customizable circuitry and symbology.*** This is the most common method, since it gives you the most power to customize the Library components.
2. ***Create a Library by Importing SPICE Discrete Component Models (`.MODEL` data), using the "Convert Data Files" tool (QucsConv).*** This is most commonly used to import model libraries from other tools, such as LTSpice or Micro-Cap.

13.1 Creating Libraries Manually

13.1.1 Preparing the Source Project

To create a *Library*, you must first create an empty *Project*. You can do this with `Project > New Project` in the menu at the top of the main QUCS-S window. Name the project whatever you like, it does not matter.

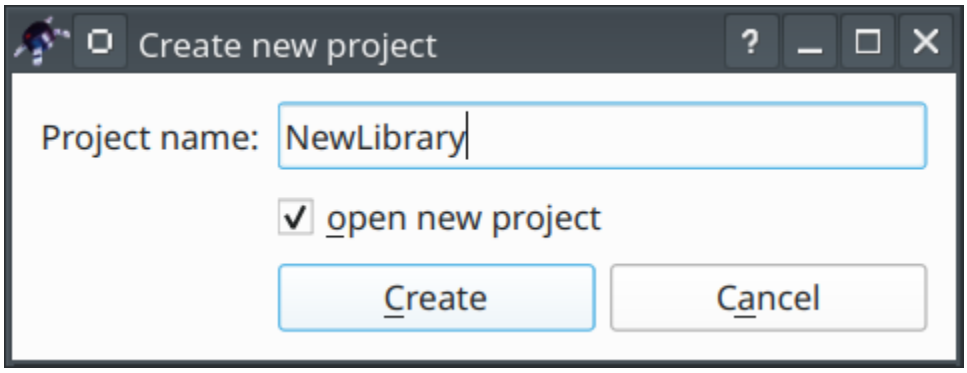


Fig. 1: An example of the dialog for creating a new Project in QUCS-S.

Once you have an empty *Project*, you need to import a *Schematic* file (`.sch`), configured for use as a *Subcircuit*, for each component you want to include in your *Library*. There are two methods to do this:

1. If you have not already created any `.sch` files for this Library, create them as you would in any normal project. Follow the *documentation on Subcircuits* if you are unsure how to do this.
2. If you already have `.sch` files prepared for your parts, you can include them in your project by simply copying the files into your project's folder on your filesystem.
    - This will be something like `$HOME/QucsWorkspace/MyNewProject_prj`, where `$HOME` is your operating system home directory, and `MyNewProject` is the name you assigned when you created your project.

- If you choose this approach, you will need to force QUCS-S to "refresh" the Project after you copy the files into it. The easiest way to do this is to close the project (Project > Close), then re-open it by double-clicking on `MyNewProject_prj` in the *Projects Tab* at the left side of the window.

The figure below shows an example of what a Project might look like when it is properly prepared to be "compiled" into a Library.

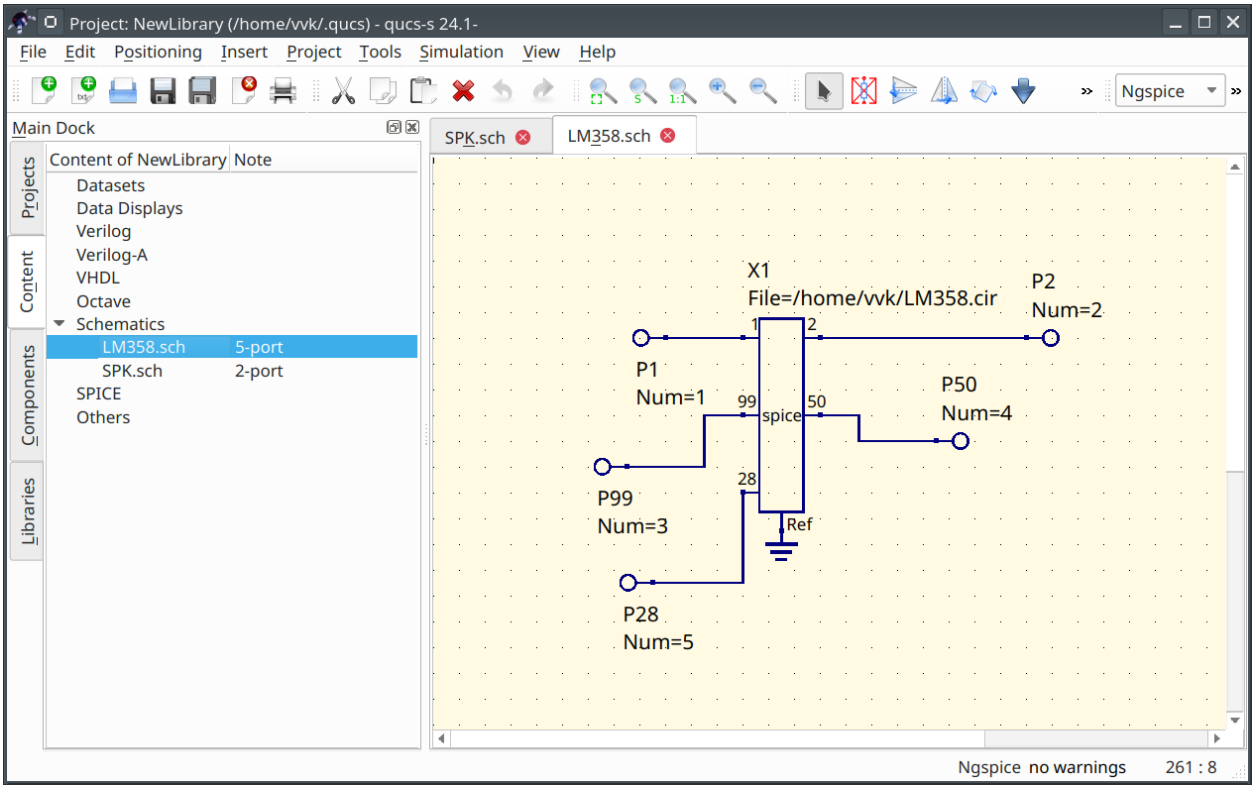


Fig. 2: An example of a Project that is ready to be "compiled" into a Library. In this example, the resulting Library would contain two parts, `LM358.sch` and `SPK.sch` (not necessarily under these names - names can be configured during the Library creation process).

13.1.2 Compiling a Library from a Project

Once your Project is prepared with all the parts you wish to include (as their own `.sch` files), navigate to `Project > Create Library`. This will open the Library Creation dialog (shown in the figure below), which allows you to name your Library, and choose which `.sch` files from your Project should be included in the compiled Library file.

After clicking "Next", you will be given the opportunity to assign a description to each component (`.sch` file) in the library. These can be relatively long descriptions. They will be visible in the bottom left of the QUCS-S window whenever a component is selected for insertion into your schematic (see the figure below for an example).

After you specify a description for every *Component* being included in your *Library*, the actual "compilation" process will begin. Relevant logs and errors from this process are shown in the pop-up window (as shown in the figure below).

> **Tip**
>
> If you see errors like `ERROR: Component "Example" has no digital model`, you can usually disregard them. This is just QUCS-S warning you that no VHDL or Verilog model for your components. If you are creating a Library of analog components, such a model is obviously not required.

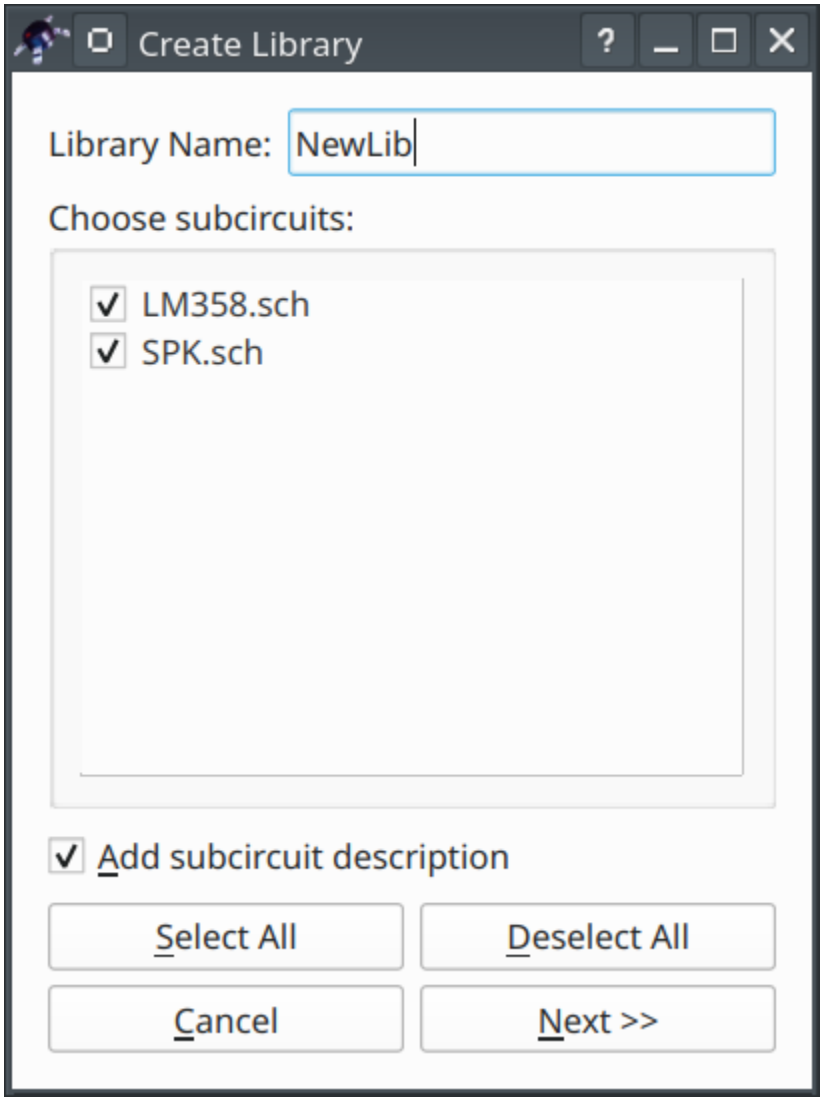


Fig. 3: The first dialog in the Library compilation process, which prompts you to select the `.sch` files for inclusion in the Library, and give the Library a name.

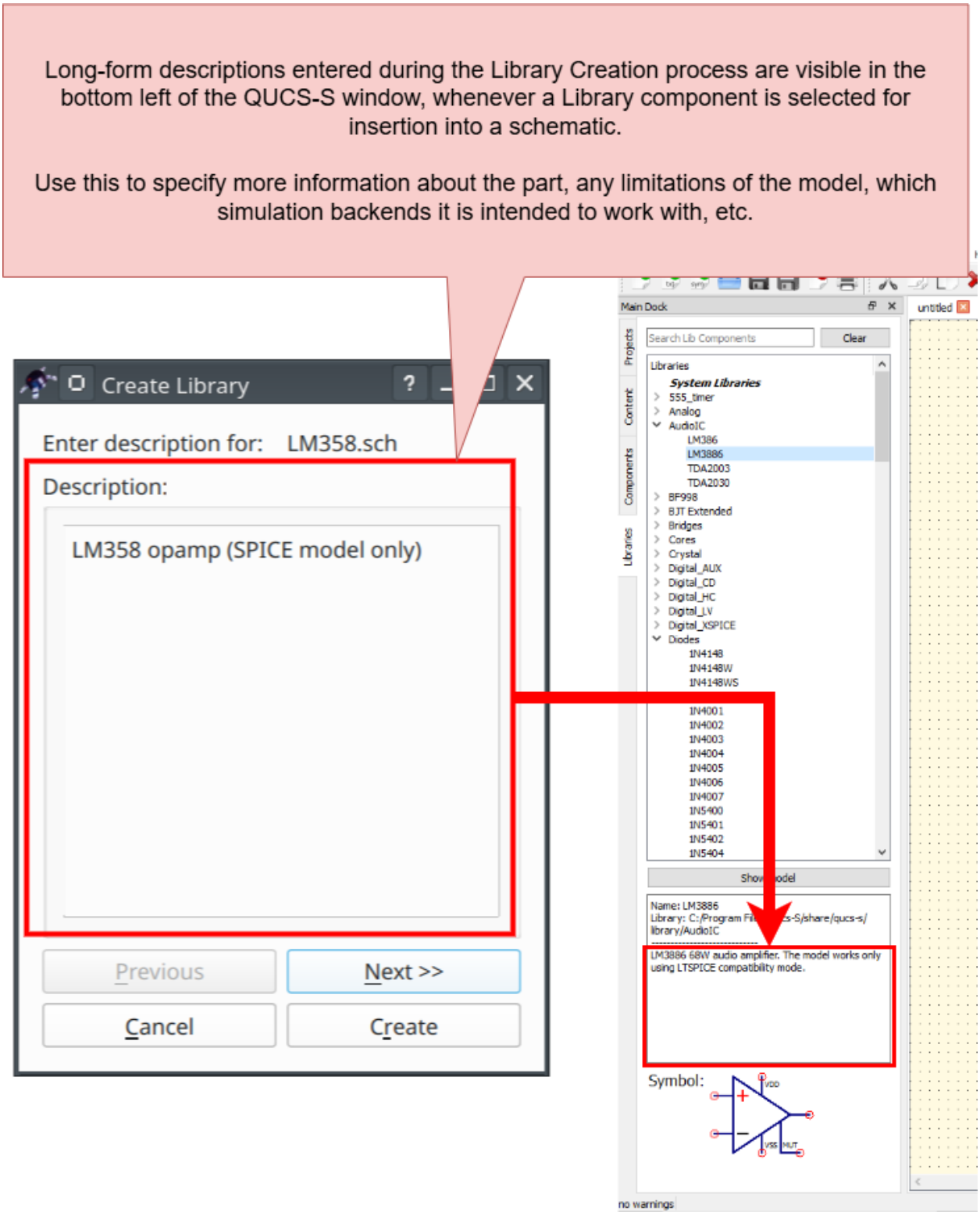


Fig. 4: Annotated screenshots showing where you can specify a long text description of each Library component, and where these descriptions are available once your Library has been created.

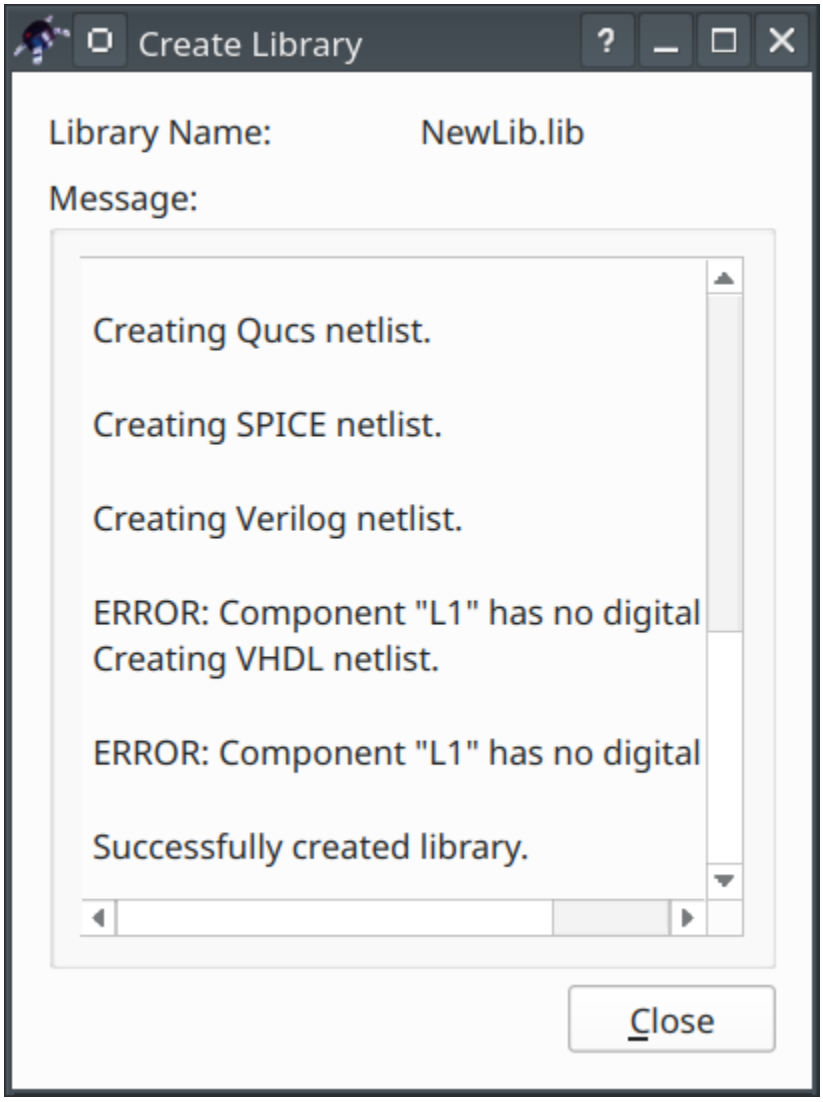


Fig. 5: Example of the logs from the “compilation” of a new Library. The “no digital model” errors can be disregarded since, in this example, we are creating models for analog components.

After the compilation process completes, your new *Library* should be usable from the "User Libraries" section of the *Libraries Tab in the main window,* as shown in the figure below.

Warning

Remember that Library "compilation" is a one-way operation! If you ever need to make edits to your Library's components, you will need the original source project files, so be sure to keep them in a safe place. The `.lib` file cannot be directly edited.

See *Introduction to Libraries* for more information.

13.2 Creating Libraries from Discrete Component Data

13.2.1 Introduction: Data Files Converter Utility (QucsConv)

A number of useful secondary tools are bundled with QUCS-S, under the `Tools` menu at the top of the main window. One of these tools is a Data File Converter, also known as QucsConv.

This utility can perform a variety of common conversion tasks, and can be accessed via the GUI (`Tools > Data Files Converter`) *or* via the command line, by calling the `qucsconv_rf` executable.

This tool ships with the QucsatorRF simulation backend, so visit QucsatorRF's GitHub repository to see the source code.

13.2.2 Converting SPICE Netlists to QUCS-S Libraries

It is possible to use the QucsConv utility to automatically create a *QUCS-S Library* from SPICE model data, with a major caveat: **The SPICE model data *must only contain* `.MODEL` directives.** It should be a single file with a `.MODEL` directive for each part.

Note

SPICE `.MODEL` directives are a common method of modeling discrete components such as BJTs, MOSFETs, and diodes in SPICE simulators.

Integrated circuits and other complex devices typically require SPICE `.SUBCKT` directives, which are much more complex and therefore cannot be processed by the QucsConv utility. If you need to include SPICE `.SUBCKT` models in a QUCS-S Library, this is still possible, you will just need to *create the Library manually.*

13.2.2.1 Example: LTspice BJT Library

As an example, the standard LTspice BJT library can be converted into a QUCS-S Library using the QucsConv tool.

Tip

The entire collection of standard libraries that ship with LTspice is available on ltwiki.org.

If you are experienced with LTspice and want to retain the device libraries you're familiar with, you can use these files with the method shown in this section to create QUCS-S Libraries containing all the same components (except any components that can't be described with a simple `.MODEL` directive).

To do this, follow the steps in the figure below, and click Convert. See *Using Libraries* for details on where to put the resulting Library (`.lib`) file to make it accessible in your QUCS-S projects.

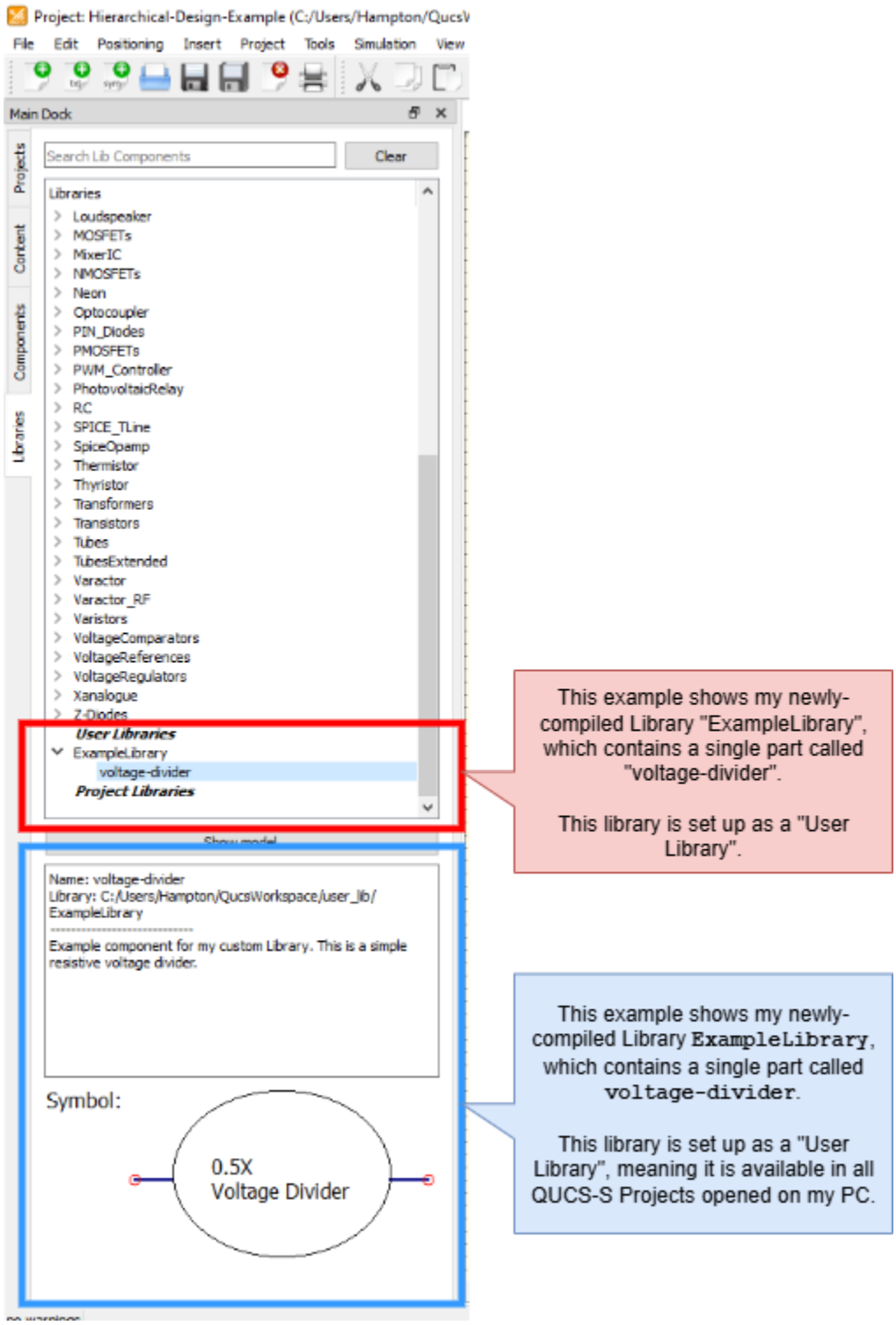


Fig. 6: Example of what a successfully-created Library would look like. This example shows a library `ExampleLibrary` containing a single part, `voltage-divider`. The Library has been set up as a *User Library*, which means it's available in all QUCS-S projects opened on this PC. See the next section for more information on this.

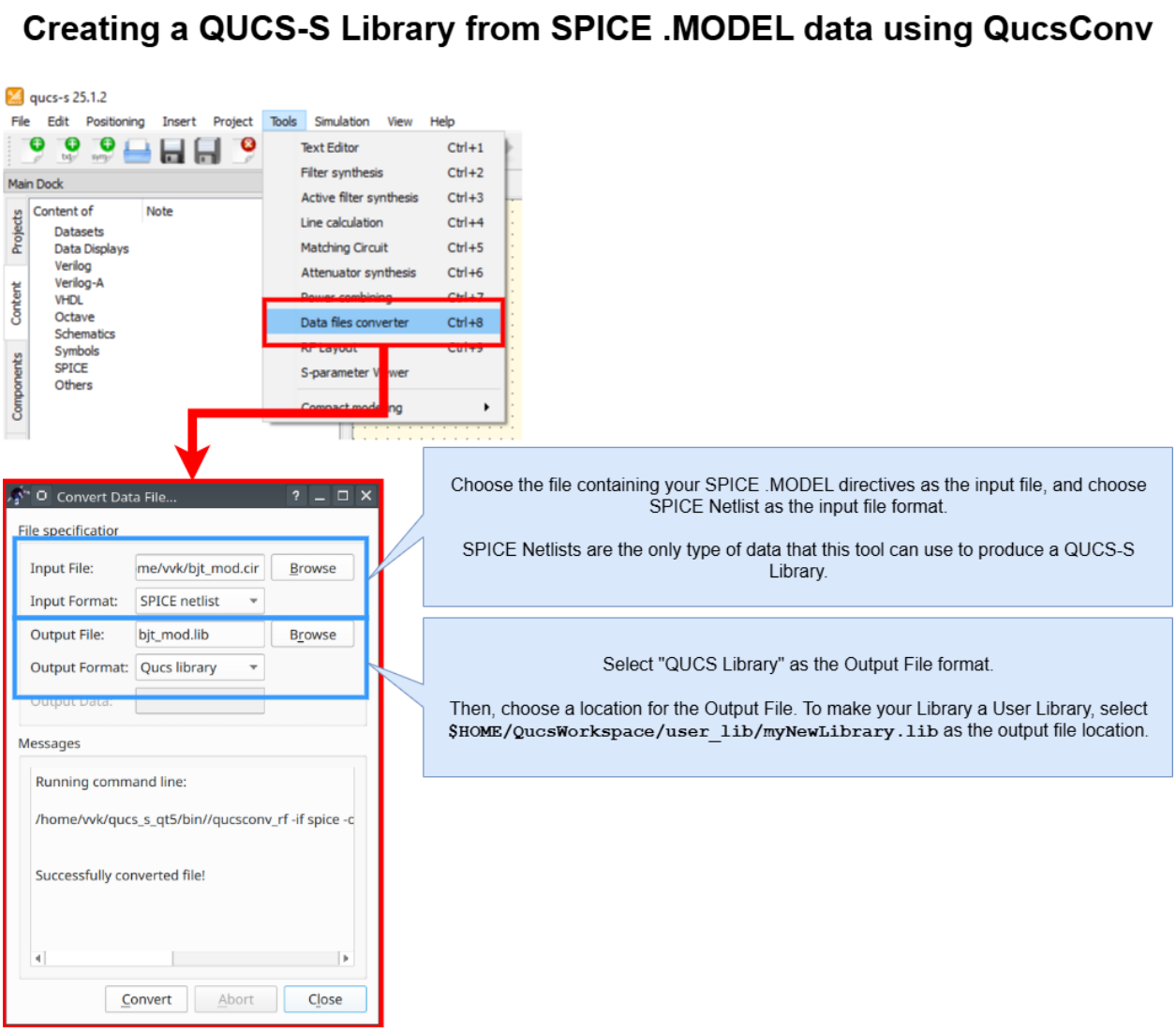


Fig. 7: Collection of images and annotations, showing how to use the QucsConv utility (included with QUCS-S) to convert SPICE `.MODEL` data into a QUCS-S Library.

Warning

Remember that *QUCS-S Library creation is a one-way process*, much like compiling source code.

There is no way to edit the generated `.lib` file. If you wish to edit any portion of the library, you must make the edit to the source files and then re-create the Library file.

It is therefore recommended to keep the original SPICE `.MODEL` data in a safe location, in case you ever need to make an adjustment to your library.

CHAPTER

FOURTEEN

USING LIBRARIES

Once you have a Library (`.lib` file), you can do one of two things with it:

1. *Set it up as a **User Library***, which is available across *all* QUCS-S projects opened on your PC.
    - *User Libraries* will NOT travel with your QUCS-S projects when transferred from one computer to another.
    - **This is what QUCS-S does with new Libraries by default.**
2. *Set it up as a **Project Library***, which is stored within another QUCS-S *Project*.
    - *Project Libraries* are "portable". They travel anywhere your *Project* goes, including to another PC.

14.1 Setting up a User Library

If you are *creating a brand new Library manually*, it will be configured as a User Library by default.

If you are importing a pre-existing Library (for example, if a peer has sent you a `.lib` file that you want to use), simply copy the file into `$HOME/QucsWorkspace/user_lib/` and restart QUCS-S for the changes to take effect.

14.2 Setting up a Project Library

Project Libraries have the advantage of being "portable" with the rest of your QUCS-S Project, unlike User Libraries. This is because Project Libraries are stored in a *QUCS-S Project folder*, and therefore travel alongside the rest of the Project (in contrast to a User Library which is stored at the system level).

To configure a Library as a Project Library, simply paste the `.lib` file into the Project folder. You may need to close and re-open the Project in QUCS-S for the Library to be recognized.

To access a Project library, first open the Project it's stored in (via the Projects tab). Then, navigate to the Libraries tab. Any Project Libraries should be listed in the "Project Libraries" section of the Libraries tab, as shown in the figure below.

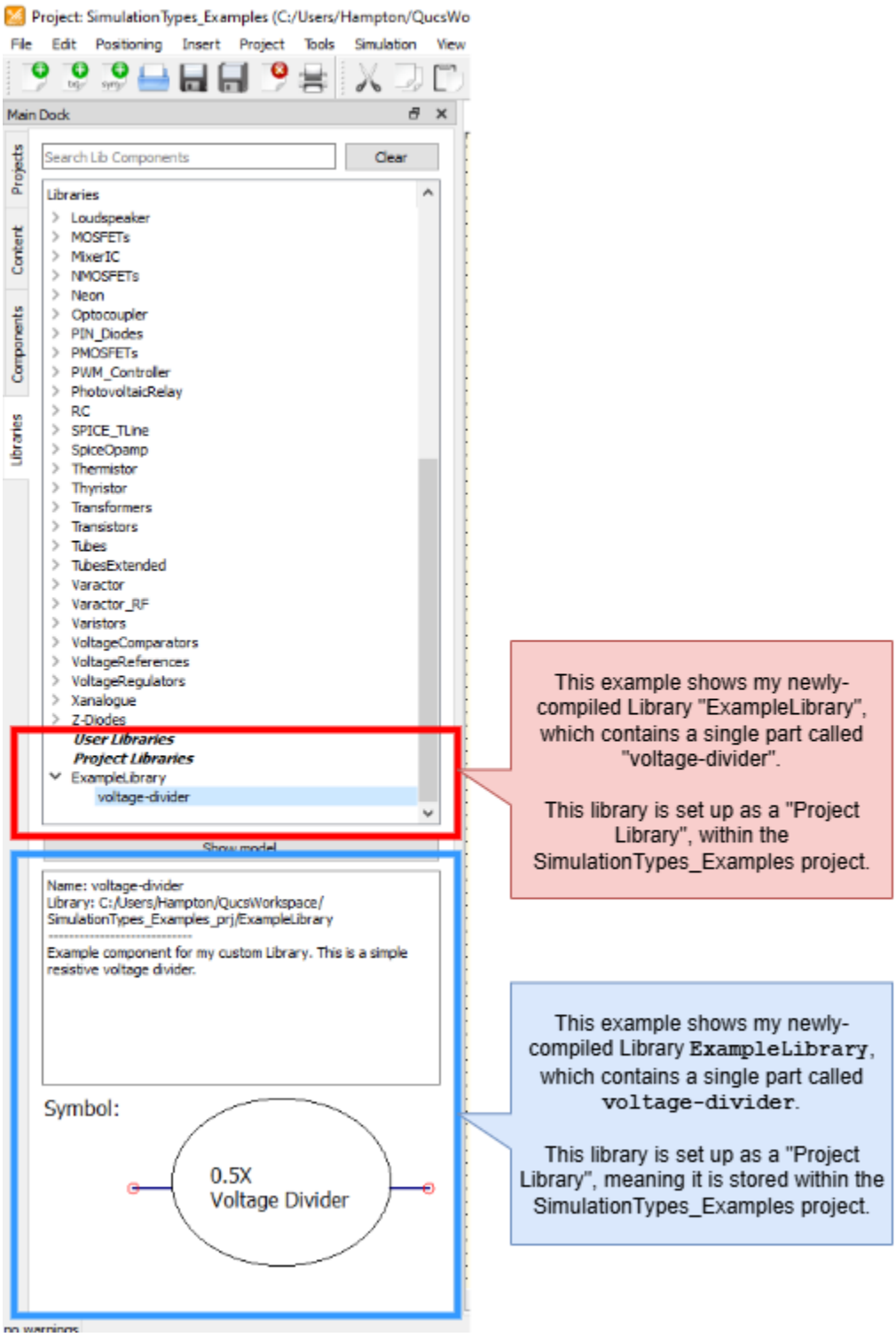


Fig. 1: Example of what a Library configured as a Project Library would look like. This example shows a library `ExampleLibrary` containing a single part, `voltage-divider`. It is stored within the `SimulationTypes_Examples` project.