

SUN Magnetics

JoSIM®-Pro v2.9.0

User Manual

Copyright © 2026 SUN Magnetics (Pty) Ltd. All Rights Reserved for:
JoSIM®-Pro v2.9.0.

SUN Magnetics (RF) (Pty) Ltd
15 De Beer Street
Stellenbosch, 7600
Republic of South Africa

www.sun-magnetics.com

Permission is granted to anyone to make or distribute verbatim copies of this document as received, in any medium, provided that the copyright notice and the permission notice are preserved, and that the distributor grants the recipient permission for further redistribution as permitted by this notice.

JoSIM® is a registered trademark of SUN Magnetics (RF) (Pty) Ltd.
All other trademarks are the property of their respective owners.

Date: October 22, 2026

Credits

JoSIM®-Pro is the product of the combined research and development efforts of the JoSIM Development Team, including contributors from SUN Magnetics (Pty) Ltd, with additional support from industry partners and researchers.

Contents

1 Introduction **9**
1.1 About This Manual . . . 9
1.2 Find the workflow you need . . . 9
1.3 Who JoSIM®-Pro Is For . . . 9
1.4 What This Manual Covers . . . 9

2 System Requirements **11**
2.1 Operating Systems and Packages . . . 11
2.2 Hardware Requirements . . . 11
2.3 Software Requirements . . . 11
2.4 Installed Command and Launcher Expectations . . . 11
2.5 Network and Licensing . . . 12
2.6 Practical Guidance . . . 12
2.7 Unsupported or Non-Standard Systems . . . 12

3 Getting Started: Install, License, and Launch **13**
3.1 Choose the package for your platform . . . 13
3.2 Windows install and launch . . . 13
3.3 Linux install and launch . . . 13
3.4 macOS install and launch . . . 13
3.5 Licensing and first-run behavior . . . 14
3.6 Generate a System ID from the GUI . . . 14
3.7 Install or select the issued license . . . 14
3.8 Optional command-line validation . . . 15
3.9 Continue to the first GUI simulation . . . 15

4 GUI Quick Start: First Simulation **16**
4.1 Before starting . . . 16
4.2 Launch the GUI . . . 16
4.3 Open the installed examples directory . . . 16
4.4 Open the JTL example . . . 16
4.5 Optional: check run settings . . . 17
4.6 Run the active netlist . . . 17
4.7 Inspect the waveform . . . 17
4.8 Export or save the result . . . 17
4.9 Continue from here . . . 18

5 GUI Workspace and Files **19**
5.1 Workspace overview . . . 19
5.2 Creating, opening, and saving files . . . 19
5.3 Browse projects and recent files . . . 19
5.4 Arrange tabs and editor groups . . . 20
5.5 Use the toolbar and command palette . . . 21
5.6 Read the application status bar . . . 22
5.7 Protect source files . . . 22
5.8 External file changes . . . 23
5.9 Restore a workspace and recover edits . . . 24
5.10 Choose an autosave policy . . . 24
5.11 Compare and restore local save history . . . 25

6 Writing and Editing Netlists **26**
6.1 Start with the source . . . 26
6.2 Complete fields and include paths . . . 26
6.3 Inspect arguments and symbol information . . . 28
6.4 Insert a snippet . . . 28
6.5 Use optional language annotations . . . 29
6.6 Edit lines and literal text . . . 29
6.7 Work with multiple and column selections . . . 30
6.8 Expand and shrink a syntax selection . . . 31
6.9 Insert and navigate paired delimiters . . . 31
6.10 Indent blocks and continuations . . . 31
6.11 Format source deliberately . . . 32
6.12 Keep text Undo and cursor history distinct . . . 33

7 Navigation, Search and Refactoring **34**
7.1 Open files and jump to a location . . . 34
7.2 Follow scopes and source links . . . 34
7.3 Find declarations and references . . . 35
7.4 Return through navigation history . . . 35
7.5 Fold source safely . . . 36
7.6 Search and replace in one document . . . 36
7.7 Search a workspace . . . 37
7.8 Review workspace replacements . . . 38
7.9 Mark review locations . . . 38
7.10 Rename a resolved symbol . . . 38
7.11 Edit paired subcircuit names . . . 39
7.12 Preview a circuit refactoring . . . 39

8 Diagnostics, Checks and Tasks **40**
8.1 Read live diagnostics . . . 40
8.2 Check the whole workspace . . . 40
8.3 Understand checking, stale and incomplete states . . . 41
8.4 Preview and apply a quick fix . . . 41
8.5 Understand static checks . . . 42
8.6 Run an explicit saved-circuit check . . . 43
8.7 Inspect and cancel Tasks and Output . . . 43

9 Editor Preferences and Display **45**
9.1 Choose defaults and setting scopes . . . 45
9.2 Customize keyboard shortcuts . . . 45
9.3 Choose font, spacing and zoom . . . 46
9.4 Wrap and scroll long source lines . . . 46
9.5 Configure line numbers, rulers and guides . . . 46
9.6 Reveal whitespace and control characters . . . 47
9.7 Use the overview ruler and minimap . . . 47
9.8 Keep scope visible while scrolling . . . 47
9.9 Read syntax and semantic colours . . . 47
9.10 Adjust diagnostic severity and suppression . . . 48
9.11 Popup behavior in small windows . . . 48

10 Git Review and External Editors **49**
10.1 Inspect repository state . . . 49
10.2 Review whole-file stage and unstage . . . 49
10.3 Choose individual lines or hunks . . . 49
10.4 Recover a prior index action . . . 50
10.5 Review and validate a merge result . . . 50
10.6 Use JoSIM-Pro from an external editor . . . 51

11 Schematic Visualization **52**
11.1 Switching to the netlist visualizer . . . 52
11.2 Visualizer controls and layout edits . . . 52

11.3 Subcircuit navigation . . . 52
11.4 Layout persistence . . . 52
11.5 Schematic export . . . 53

12 GUI Simulation and Results 54
12.1 Simulation lifecycle . . . 54
12.2 Preconditions for running . . . 54
12.3 Run settings . . . 54
12.4 Create multiple runs . . . 55
12.5 Result states and diagnostics . . . 55
12.6 Waveform result view . . . 56
12.7 Trace Manager basics . . . 56
12.8 Visibility, order, groups, and comparison plots . . . 56
12.9 Analysis outputs . . . 56
12.10 Manual annotations . . . 57
12.11 AC magnitude and angle views . . . 58
12.12 Imported CSV and TSV result data . . . 58
12.13 Plot export studio . . . 58
12.14 Troubleshooting hooks . . . 59

13 GUI-to-CLI Equivalence 60
13.1 Simulation workflow equivalence . . . 60
13.2 Run Settings equivalence . . . 60
13.3 Output, streaming, and import equivalence . . . 61
13.4 Licensing equivalence . . . 61
13.5 Optimum handoff equivalence . . . 61

14 Optimum CLI Workflows: Verify, Margin, Yield, and Optimize 62
14.1 GUI handoff model . . . 62
14.2 Installed DFF example . . . 62
14.3 CLI invocation and licensing . . . 62
14.4 Mode overview . . . 63
14.5 Runtime contract . . . 63
14.6 File paths and project layout . . . 63
14.7 Minimal configurations . . . 63
14.8 Configuration reference . . . 64
14.9 Verification inputs and tolerances . . . 66
14.10 Margin workflow . . . 68
14.11 Yield workflow . . . 69
14.12 Optimize workflow . . . 69
14.13 Worked DFF example . . . 69

15 CLI Usage and Command Reference 71
15.1 When to use the CLI . . . 71
15.2 Command structure . . . 71
15.3 Default behavior . . . 71
15.4 Quick command patterns . . . 71
15.5 Option summary . . . 72
15.6 Operational notes . . . 73
15.7 Common tasks . . . 75

16 Netlist Syntax Reference 76
16.1 Netlist anatomy . . . 76
16.2 Quick syntax index . . . 76
16.3 Detailed reference . . . 78
16.4 Element reference . . . 78
16.5 Independent source reference . . . 82
16.6 Dependent source reference . . . 84
16.7 Control-command reference . . . 85
16.8 Built-in constants . . . 95
16.9 Summary . . . 95

17 Worked Netlist Examples **96**
17.1 First transient run and result inspection . . . 96
17.2 Hierarchical design with subcircuits and included files . . . 96
17.3 Device and model characterization with `.IV` . . . 97
17.4 Noise, spread, and reproducibility with a fixed seed . . . 98
17.5 Variable-step and streamed output for larger runs . . . 98
17.6 Derived outputs and multiple output files . . . 98
17.7 Full logic-cell examples . . . 99
17.8 Josephson Transmission Line (JTL) . . . 99
17.9 RSFQ Splitter Cell . . . 100
17.10 RSFQ AND Gate . . . 101
17.11 Summary . . . 102

18 Simulation Feature Reference **103**
18.1 Core Simulation Capabilities . . . 103
18.2 Circuit Description and Modelling . . . 103
18.3 Noise, Spread, and Realism . . . 104
18.4 Characterization and Analysis Workflows . . . 104
18.5 Output, Post-Processing, and Data Handling . . . 105
18.6 Summary . . . 106

19 Python Automation **107**
19.1 Making the module available . . . 107
19.2 Process-wide settings and licensing . . . 107
19.3 Smallest useful file-based run . . . 107
19.4 Runtime control from Python . . . 107
19.5 Explicit runtime example . . . 108
19.6 Reading results into NumPy . . . 108
19.7 Getting named traces from the print list . . . 108
19.8 Transient measurements . . . 109
19.9 Parameter sweeps . . . 109
19.10 Pausing and saving a simulation . . . 109
19.11 Writing CSV, BIN, RAW, or normal print output . . . 110
19.12 Static operating points . . . 110
19.13 Complex AC results . . . 110
19.14 Large runs, streaming, and variable-step restrictions . . . 111
19.15 Repeated runs and batch automation . . . 111
19.16 End-to-end plotting example . . . 111
19.17 Practical reminders . . . 112

20 Troubleshooting **113**
20.1 The GUI launcher does not start . . . 113
20.2 The GUI reports a licensing problem or opens in degraded mode . . . 113
20.3 Command-line launch exits immediately . . . 114
20.4 Input files, `.include` paths, and subcircuit boundaries . . . 114
20.5 Netlist syntax, component definitions, and model lines . . . 114
20.6 Parameters, expressions, and control commands fail to parse . . . 115
20.7 Source functions and noise expressions are rejected . . . 115
20.8 Output requests are rejected or produce the wrong signals . . . 116
20.9 Variable-step mode or runtime options are incompatible with the netlist . . . 116
20.10 Transmission-line initial conditions or delayed breakpoints are rejected . . . 117
20.11 The simulation starts but fails, diverges, or produces no useful result . . . 117
20.12 Large runs, output files, compression, and streaming issues . . . 117
20.13 Optimum exits before Margin, Yield, or Optimize runs . . . 118
20.14 A practical order for debugging difficult runs . . . 118

A Native Integration Appendix **119**
A.1 What the native interfaces are for . . . 119
A.2 When to use this instead of Python . . . 119
A.3 Minimal file-based workflow . . . 119
A.4 Path and encoding notes . . . 119

A.5 Authoritative reference 119

Introduction

JoSIM®-Pro is a professional circuit simulation environment for superconducting electronics and Josephson-junction-based designs. It is intended for engineers, researchers, and technical users who need a reliable way to model circuit behaviour, study transient response, evaluate design choices, and analyse simulation results across a range of workflows.

At its core, JoSIM®-Pro is designed to make superconducting circuit simulation practical for real design work. This manual begins with graphical workflows for installation, first simulation, waveform inspection, and export, then covers command-line simulation, structured netlist support, multiple output formats, automation interfaces, and optimisation tools as complementary workflows.

JoSIM®-Pro is suitable for work ranging from small test circuits to larger and more complex superconducting systems. Whether the task is validating a basic concept, exploring timing and phase behaviour, generating output for post-processing, or preparing data for optimisation and yield studies, JoSIM®-Pro provides the tools needed to move from circuit description to useful simulation results efficiently.

JoSIM® is a registered trademark of SUN Magnetics (RF) (Pty) Ltd and is used throughout this manual to identify the simulator and associated software tools.

1.1 About This Manual

This manual is a practical guide to using JoSIM®-Pro. It is written to help users understand the software, build effective simulation workflows, and make full use of the features available in the installed package.

The emphasis throughout is on what JoSIM®-Pro does, how to use it, and how its different components fit together in normal engineering work. Where a feature has important operating limits or usage requirements, those are presented as practical operating guidance.

1.2 Find the workflow you need

Task	Start here
First installation and run	Getting Started; GUI Quick Start (Chapter 4).
Files, split views and recovery	GUI Workspace and Files (Chapter 5).
Completion, snippets and formatting	Writing and Editing Netlists (Chapter 6).
Search, references and Rename	Navigation, Search and Refactoring (Chapter 7).
Problems, checks and cancellation	Diagnostics, Checks and Tasks (Chapter 8).
Appearance and project defaults	Editor Preferences and Display (Chapter 9).
Repository review or VS Code	Git Review and External Editors (Chapter 10).
Plots and export	GUI Simulation and Results (Chapter 12).

1.3 Who JoSIM®-Pro Is For

JoSIM®-Pro is intended for:

- engineers and researchers working with superconducting circuits and Josephson junction devices
- users who prefer a graphical environment for editing, running, and reviewing simulations
- users who need a simulator that can be driven from the command line as part of repeatable design flows
- users who want to automate simulation and post-processing from Python
- users who need design-space exploration, verification, margin, yield, or optimisation workflows through the Optimum CLI

While the manual assumes a technical audience, it is organised so that new users can begin with installation and GUI-based first-use workflows, and then move on to netlist controls, scripting, CLI automation, Python, native integration, and optimisation as needed.

1.4 What This Manual Covers

This manual covers the principal user workflows available in JoSIM®-Pro, including:

- installation, licensing, and first-time setup
- GUI-based editing, execution, and visualisation workflows
- command-line simulation with `josim-pro-cli`

- circuit netlists, control statements, and supported simulation inputs
- output files, streaming, and result handling
- Optimum CLI workflows for verification, margin, yield, and optimisation
- Python-based automation and scripting
- troubleshooting and reference material for common usage issues

The chapters that follow are arranged so that the manual can be read from front to back as an introduction to the product, or used selectively as a reference while working.

Key strengths of JoSIM®-Pro include:

- **High Performance**: JoSIM®-Pro is built to handle demanding superconducting circuit simulation workloads efficiently.
- **Cross-Platform Availability**: The software is designed for use on Windows, Linux, and macOS.
- **Flexible Workflows**: Users can work through the GUI first, then move to the command line, Python, or Optimum CLI when automation or deeper control is needed.
- **Practical Output Handling**: JoSIM®-Pro supports workflows that range from direct console output to file-based export and downstream analysis.

With JoSIM®-Pro, users have access to a capable and flexible environment for superconducting circuit simulation, from first experiments through to detailed design and analysis tasks.

System Requirements

This chapter summarizes the operating-system, hardware, software, and licensing requirements for installed use of JoSIM®-Pro. The GUI supports interactive editing, simulation, and result inspection; the CLI, Python module, Optimum CLI, and native APIs support automation, scripted operation, and integration workflows.

2.1 Operating Systems and Packages

Use the package built for your operating system and architecture.

- **Windows**: 64-bit Windows 10 or later. The installer is an MSI that installs under `Program Files/SUN Magnetics/JoSIM-Pro` and can create `JoSIM-Pro` shortcuts for launching the GUI.
- **Linux**: a recent 64-bit Linux environment compatible with the supplied DEB or RPM package. The active enterprise target is EL8/RHEL8-compatible. GUI packages require the desktop GTK/OpenGL runtime libraries declared by the package metadata.
- **macOS**: a current macOS version matching the architecture of the supplied macOS package. Current package builds are architecture-specific unless the accompanying notes explicitly state that a universal binary is supplied.

Use the installer or package supplied for your operating system rather than copying files from another platform or build layout.

2.2 Hardware Requirements

2.2.1 Minimum Requirements

The following configuration is suitable for installation, learning, and smaller simulation jobs:

- **Processor (CPU)**: dual-core 64-bit processor
- **Memory (RAM)**: 4 GB
- **Storage**: at least 500 MB for software plus additional space for projects, examples, output files, and temporary files
- **Display**: desktop display suitable for the GUI workspace, editor, and waveform views

2.2.2 Recommended Requirements

For larger circuits, longer transient runs, Monte Carlo style studies, or heavier post-processing, use:

- **Processor (CPU)**: modern quad-core or better 64-bit processor
- **Memory (RAM)**: 16 GB or more
- **Storage**: SSD storage with at least 1 GB available for installation, outputs, examples, and temporary files
- **Display**: an engineering desktop environment with enough space for netlists, logs, visualizer views, and plots

2.3 Software Requirements

- **Windows runtime**: required runtime components are supplied or expected by the MSI package. If a site image is missing Microsoft runtime components, repair the installation or install the required redistributable supplied by your administrator.
- **Linux GUI runtime**: install DEB/RPM packages with `apt install`, `dnf install`, or `yum install` so dependencies such as GTK and OpenGL runtime libraries are resolved automatically.
- **macOS package runtime**: use the supplied package layout. The application bundle contains the GUI and app-local runtime files; wrappers in `/usr/local/bin` expose the GUI and CLI tools.
- **Python (optional)**: Python is required only if you plan to use the JoSIM®-Pro Python interface or user-local Python integration helpers.
- **External plotting tools (optional)**: spreadsheet tools, Python plotting libraries, GNUPlot, MATLAB, or similar tools may be used for post-processing exported data. The GUI can inspect waveform results directly.

2.4 Installed Command and Launcher Expectations

- **Windows**: launch the GUI with the `JoSIM-Pro` shortcut; it targets `JoSIM_Pro_GUI.exe`. Command-line discovery depends on the MSI command-line PATH integration option.
- **Linux**: launch the GUI through `/usr/local/bin/josim-pro-gui` or the desktop entry named `JoSIM-Pro`. Use the wrapper rather than starting the GUI binary directly.
- **macOS**: launch the GUI from `/Applications/SUN Magnetics/JoSIM-Pro.app` or through `/usr/local/bin/josim-pro-gui` when the macOS package is installed.

The command-line simulator is `josim-pro-cli`. The optimization tool is `optimum-cli`. These tools are fully supported for automation, batch simulation, reproducibility checks, and TOML-driven optimisation workflows.

2.5 Network and Licensing

A valid JoSIM®-Pro license is required before simulations can run. Current packages also require the latest SUN Magnetics License Manager to be installed, because JoSIM-Pro delegates license checks and System ID generation to the `smlicensed` licensing daemon. Without `smlicensed`, neither the GUI nor the command-line tools can generate `SM_SysID.txt`.

- **License Manager prerequisite**: install or update the SUN Magnetics License Manager before launching JoSIM-Pro for licensing, System ID generation, or simulation work.
- **Default daemon locations**: JoSIM-Pro looks for `C:\Program Files\SUN Magnetics\Licenses\bin\smlicensed.exe` on Windows, `/usr/local/share/sunmagnetics/licenses/bin/smlicensed` on Linux, and `/Applications/SUN Magnetics/Licenses/bin/smlicensed` on macOS. Site deployments can override this with `SMLICENSED_PATH` when required.
- **Offline use**: once installed and licensed correctly, normal simulation work can be performed without a permanent network connection unless your licensing arrangement requires online validation.
- **License fulfilment**: network access may be required during deployment or license administration, depending on how the software is issued and managed in your environment.
- **Licensing service**: if the GUI reports that it cannot contact the licensing service, use **Retry** only after correcting the License Manager installation, service state, network, firewall, license-file, or seat-availability problem.
- **Support**: for licensing or installation assistance, contact support@sun-magnetics.com.

2.6 Practical Guidance

For the best experience with JoSIM®-Pro:

- keep project files in a user-writeable working directory
- use the installed examples directory for manual workflows
- ensure that output locations have sufficient free space
- use SSD storage for larger projects or repeated simulation runs
- use additional memory where large netlists or extensive output data are expected

2.7 Unsupported or Non-Standard Systems

If your system falls below the minimum requirements, reliable operation cannot be guaranteed. If you need to deploy JoSIM®-Pro in an unusual environment, contact SUN Magnetics for guidance before relying on the software in production work.

Getting Started: Install, License, and Launch

This chapter covers first use after installation: install JoSIM®-Pro, launch the GUI, resolve licensing, and then continue to the GUI quick start. Terminal commands remain available for validation, automation, and scripted simulation.

3.1 Choose the package for your platform

Use the installer or package supplied for your operating system. Do not infer support for one operating system from files intended for another platform.

- **Windows**: use the signed `JoSIM-Pro-<version>-Windows-AMD64.msi` package.
- **Linux**: use the supplied distribution package. Install it with a dependency-resolving package manager so GTK/OpenGL GUI runtime dependencies are installed with the application.
- **macOS**: use the supplied macOS package when one is included for your platform. The macOS package installs an application bundle and command wrappers; it is not interchangeable with the Windows MSI or Linux package.

3.2 Windows install and launch

The Windows package is a custom MSI. It installs JoSIM®-Pro under the SUN Magnetics program directory and can create shortcuts that launch the GUI.

1. Run the `JoSIM-Pro-<version>-Windows-AMD64.msi` installer.
2. Keep **Create Start Menu shortcut** selected if you want a launcher under **SUN Magnetics** in the Start Menu.
3. Keep **Create Desktop shortcut** selected if you want a Desktop launcher.
4. Keep **Command-line PATH integration** selected if you want new terminals to discover `josim-pro-cli` and `optimum-cli` automatically.
5. After installation, launch `JoSIM-Pro` from the Start Menu or Desktop shortcut.

The shortcut target is the installed GUI executable `JoSIM_Pro_GUI.exe`. If the command-line PATH option was disabled, or if a terminal was already open before installation, run command-line tools from the installed `bin` directory or open a new terminal after installation.

3.3 Linux install and launch

The Linux package installs runtime files below the SUN Magnetics application prefix and exposes launch commands through `/usr/local/bin`. Launch the GUI through the wrapper so the runtime library path is prepared correctly.

For a Debian-style package, prefer a dependency-resolving install:

```
sudo cp JoSIM-Pro-*.deb /tmp/
sudo apt install /tmp/JoSIM-Pro-*.deb
```

For an RPM-style package, use `dnf` or `yum` rather than `rpm -i`:

```
sudo dnf install ./JoSIM-Pro-*.rpm
# or, on systems using yum:
sudo yum install ./JoSIM-Pro-*.rpm
```

After installation, launch from the desktop entry named `JoSIM-Pro`, or run:

```
josim-pro-gui
```

The public Linux command entries are:

- `/usr/local/bin/josim-pro-gui`
- `/usr/local/bin/josim-pro-cli`
- `/usr/local/bin/optimum-cli`

The GUI wrapper prepares the runtime library search path before starting the application. If the wrapper reports that `JoSIM_Pro_GUI` is missing or not executable, repair or reinstall the package rather than bypassing the wrapper.

3.4 macOS install and launch

When a macOS package is supplied, it installs JoSIM®-Pro as an application bundle:

```
/Applications/SUN Magnetics/JoSIM-Pro.app
```

Launch the GUI from the application bundle or run the wrapper:

```
josim-pro-gui
```

The macOS package also exposes `josim-pro-cli` and `optimum-cli` through `/usr/local/bin`. Current macOS packages are architecture-specific. Use the package built for your Mac architecture; do not assume a universal binary unless the package notes explicitly say one is supplied.

3.5 Licensing and first-run behavior

A valid JoSIM®-Pro license is required before simulations can run. The latest SUN Magnetics License Manager must also be installed, because the GUI and CLI use the `smlicensed` daemon for license checks and System ID generation. On startup, the GUI checks the configured license path and contacts `smlicensed`. A successful check opens the main workspace normally.

License renewal continues in the background while simulation matrices remain open, including matrices retained by Python or C API applications. Repeated simulations can reuse a verified authorization for up to 60 seconds. If renewal fails, a simulation already running is allowed to finish; subsequent simulations require a valid authorization. Applications that retain matrices between runs remain active for licensing purposes until those matrices are released.



Figure 3.1: Normal startup license check from the installed GUI.

If licensing is unavailable, the startup shell presents recovery choices:

- **Retry**: check again after correcting the license file, network, or licensing-service problem.
- **Continue degraded**: open the GUI for limited inspection and recovery tasks. Simulations still require a valid license.
- **Quit**: close the application.

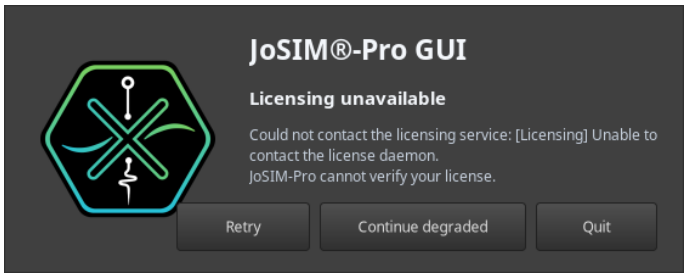


Figure 3.2: Degraded-mode startup recovery choices when licensing is unavailable.

Use degraded mode when you need to inspect installation state before support can issue or repair a license. System ID generation still requires the License Manager: if `smlicensed` is missing, outdated, blocked, or installed outside the expected location, no `SM_SysID.txt` file will be produced.

3.6 Generate a System ID from the GUI

If you do not yet have a license, generate a system identifier from the GUI after installing the latest SUN Magnetics License Manager:

1. Install or update the SUN Magnetics License Manager so that `smlicensed` is available on the machine.
2. Launch `JoSIM-Pro`.
3. If the startup shell reports a license problem, choose **Continue degraded**.
4. Open `Help > About`.
5. Click **Generate System ID...**. The GUI asks `smlicensed` to create `SM_SysID.txt`; it does not generate the file independently.
6. Save the file as `SM_SysID.txt`.
7. Send `SM_SysID.txt` to support@sun-magnetics.com when requested for licensing support.

If this step fails or the selected folder does not contain `SM_SysID.txt`, repair the License Manager installation first. On non-standard deployments, confirm that `SMLICENSED_PATH` points to the installed `smlicensed` executable.

The About dialog also reports the current license state. When a license is verified, it shows the licensed use type, status, licensee, and validity date. In degraded mode it reports that verified license information is unavailable.



Figure 3.3: The About dialog exposes current license state and the **Generate System ID...** action.

JoSIM-Pro requires SUN Magnetics License Daemon 1.5.2 or newer. It launches the daemon with the selected license before requesting authorization; an existing daemon enrolls the supplied license.

3.7 Install or select the issued license

After SUN Magnetics issues a license, place the license file where your site or installation instructions expect it. If the GUI does not find it automatically, set the license path once with the command-line tool:

```
josim-pro-cli -l /path/to/sm_license.txt
```

The GUI reads the configured license path during startup. After setting the path, start `JoSIM-Pro` again and confirm that the startup shell reaches the main workspace without a licensing warning.

3.8 Optional command-line validation

Command-line validation is useful for administrators, existing CLI users, and automation workflows. It is not required for the first GUI simulation.

Check the installed simulator version:

```
josim-pro-cli -v
```

Generate a System ID from a terminal instead of the GUI. This command also requires the latest SUN Magnetics License Manager, because it invokes `smlicensed`:

```
josim-pro-cli -s
```

Run a direct CLI simulation only after licensing is working. From the installed examples directory, use the packaged JTL example:

```
josim-pro-cli -o JTL_Example.csv JTL_Example.cir
```

The full command reference appears in the CLI chapter. `optimum-cli` is covered separately in the Optimum chapter.

3.9 Continue to the first GUI simulation

After the GUI launches and licensing is resolved, continue to the **GUI Quick Start** chapter. That chapter opens an installed example netlist, runs it in the GUI, and inspects the waveform result without requiring a terminal.

If the first launch fails, use the troubleshooting chapter in this order: GUI launcher problems, licensing or degraded mode, then file/netlist/simulation issues.

GUI Quick Start: First Simulation

This chapter gives a first complete JoSIM®-Pro workflow: launch the graphical interface, open the installed `JTL_Example.cir` netlist, run it, inspect the waveform, and export a result from the GUI. The command-line tools remain available for users who automate runs from a terminal.

4.1 Before starting

This tutorial assumes that JoSIM®-Pro is installed and that the latest SUN Magnetics License Manager is installed and reachable. The GUI can be opened in degraded mode for inspection and recovery, but simulation and System ID generation still depend on the installed `smlicensed` daemon and a valid licensing setup.

Use an installed example from the examples directory. That directory includes these first-use example files:

Installed file	Use in this manual
`JTL_Example.cir`	Recommended first run. It is self-contained and has explicit plot requests.
`AND_Example.cir`	Larger logic-cell example; it depends on `base_cells.cir`.
`Splitter_Example.cir`	Additional small logic-cell example for later exploration.
`DFF_Example.cir`, `DFF_Example.sp`, and `DFF_Example_config.toml`	Canonical installed Optimum handoff example; use it after the GUI first run.

4.2 Launch the GUI

Start the application using the installed launcher:

- On Windows, use the `JoSIM-Pro` Start Menu entry or Desktop shortcut when that installer option was selected. The launcher target is the installed `JoSIM_Pro_GUI.exe` binary.
- On Linux, use the desktop entry named `JoSIM-Pro` or run `josim-pro-gui`. The command is a wrapper that prepares the GUI runtime library search path; use it instead of starting the GUI binary directly.
- On macOS, use the application package or `josim-pro-gui` wrapper supplied for the target architecture.

If the startup license check reports a missing or invalid license, return to the install and license chapter before attempting this tutorial. `Continue degraded` is suitable for opening files and inspecting the interface, but it is not the normal simulation path.

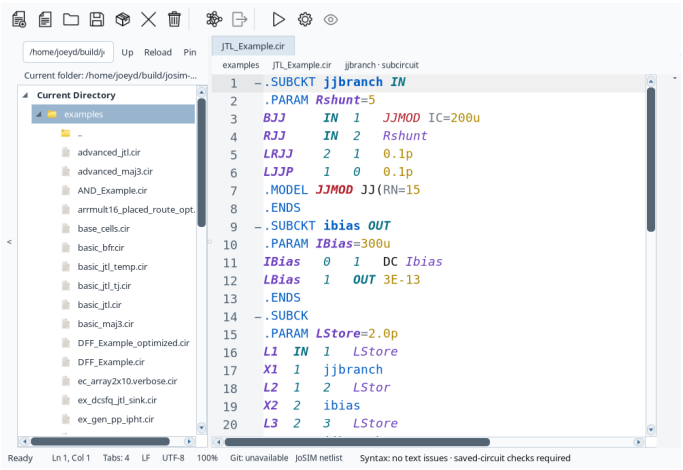


Figure 4.1: GUI workspace after opening the installed examples folder.

4.3 Open the installed examples directory

Use `File > Open Folder...` to point the file browser at the examples directory installed by the package. The exact parent directory is platform-dependent, but the examples are installed under an `examples` directory.

After the folder opens, the left file browser can be used to select netlists. If you already know the exact file location, use `File > Open...` instead. The open dialog accepts JoSIM netlists and result data files; for this tutorial, select `JTL_Example.cir`.

4.4 Open the JTL example

Open `JTL_Example.cir`. A new workspace tab appears, and the netlist is loaded into the editor. Before running it, inspect the file rather than treating it as a black-box demo.

The important lines for the first run are:

- `VIN ... PULSE(...)` defines the input pulse source.
- `XJTL 1 2 jtl` instantiates the JTL subcircuit.
- `.TRAN 0.25ps 250ps` defines the transient run length and time step.
- `.PLOT V(VIN) P(BJJ.X1.XJTL) P(BJJ.X3.XJTL) V(ROUT)` defines the traces that should appear after a successful run.

This example is a good first test because it does not require `base_cells.cir` or any other include file.

> **CLI side note.** The equivalent terminal run is `josim-pro-cli -o JTL_Example.csv JTL_Example.cir`. This is

useful for automation, but it is not required for the GUI quick start.

4.4.1 Make a small edit

Place the caret, edit the source and use Ctrl/Cmd+Z to undo. The Edit menu and right-click menu group editing, navigation and diagnostic actions. Ctrl/Cmd+Space requests completion; Ctrl/Cmd+F opens document search. Save a copy before changing installed examples.

The editor uses syntax-aware indentation and configurable tab stops. For snippets, formatting, multiple selections and source navigation, see Chapters 6 and 7.

4.5 Optional: check run settings

For the first run, use the default GUI settings unless you are deliberately testing a specific solver configuration. To inspect them, open `Run > Settings`. These are deck defaults; netlist and control-block settings may override them. The dialog exposes:

- analysis type: `Phase` or `Voltage`;
- integration method: `BDF` or `TRAP`;
- variable-step mode, valid only for compatible phase/BDF configurations;
- `Use UIC`;
- output precision in the supported range `0` to `17`;
- compressed text output;
- deterministic seed control for reproducible runtime behavior.

Select `Cancel` if you only inspected the settings. Select `OK` only when you intentionally changed a setting.

4.6 Run the active netlist

With `JTL_Example.cir` as the active tab, press the Run toolbar button. Every explicit analysis runs once; multiple analyses execute serially. To create named runs with different integration methods and output cadences, use `Run > Add analysis blocks...`, described in Create multiple runs. The GUI prepares the input file, starts the simulation session, shows progress in the active tab, and updates the result view when the run completes.

If a run is already active for the tab, the GUI prevents a second concurrent run for that tab. If the active tab contains imported CSV/TSV result data instead of a netlist, simulation is unavailable; open the source `.cir` file instead.

A failed run usually points to one of these causes:

- licensing is unavailable or degraded mode was used for inspection only;
- the netlist path or included file path is not available;
- a netlist edit introduced a syntax or model error;
- run settings were changed to an incompatible combination.

Use the diagnostics/status information in the tab before changing unrelated settings.

4.7 Inspect the waveform

After a successful run, the result view displays the traces requested by the `.PLOT` line. For `JTL_Example.cir`, expect traces for the input source, internal Josephson-junction branch phases, and the output resistor voltage.

Use the waveform area for the first inspection pass:

- verify that the expected traces appear;
- use trace visibility controls to hide or show traces;
- use `Run > Trace Manager` when result traces exist and you need a structured visibility/selection dialog;
- use `View > Grid` and `View > Sync Crosshairs` to control waveform-view behavior.

Do not switch to the CLI just to see the first waveform. CSV/TSV export and command-line processing are later workflow choices, not prerequisites for first use.

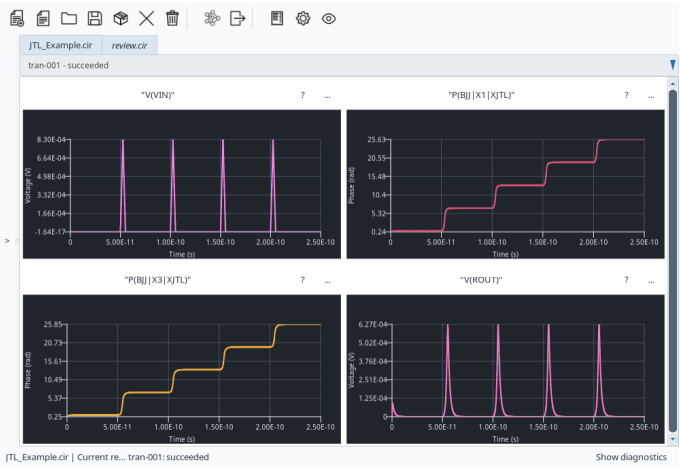


Figure 4.2: Successful GUI waveform result from the installed JTL example.

4.8 Export or save the result

When the waveform view is active and output is ready, use the toolbar export button to open the plot export workflow. The same export button is context-sensitive: in visualizer view it exports the schematic, while in waveform/result view it opens plot export for the current results.

For a first saved result:

1. keep the export scope at visible or all-visible traces unless you intentionally selected a subset;
2. choose a visual format such as `PNG image` for a report figure, or a data format such as `CSV data` / `TSV data` for downstream analysis;

3. save the file into your project directory rather than into the installed examples directory;
4. confirm from the tab status/diagnostics that the export succeeded.

Supported plot-export formats include `PNG`, `JPEG`, `SVG`, `PDF`, `EPS`, `TikZ/LaTeX`, `CSV`, and `TSV`. Data exports write selected trace data and x-axis values; visual style controls apply to raster/vector plot exports, not to CSV/TSV output.

Use `File > Save As...` only when you want to save a modified copy of the netlist. It does not replace the plot/data export path for waveform results.

4.9 Continue from here

After the first successful run, use Chapter 5 for files, split views and recovery, Chapter 6 for netlist authoring, and Chapter 11 for schematic inspection. Chapter 12 covers result organization, Trace Manager and export. Chapter 13 maps these workflows to command-line automation.

GUI Workspace and Files

Open a project, arrange its views and keep source files safe. For writing netlists, continue with Chapter 6; for execution and plots, see Chapter 12. Screenshots show version 2.9.0 on Linux. Windows and macOS use the same workflows; native file dialogs and modifier keys can differ.

5.1 Workspace overview

The GUI is organized around workspace tabs. A tab can contain an editable netlist, an imported result data file, a waveform result produced by simulation, or a visualizer view built from the active netlist. The left file browser is a project navigation aid; it does not replace the normal `File > Open...` dialog.

Use the GUI workspace for interactive work:

- create or open a `.cir` netlist;
- edit and save the netlist in a workspace tab;
- open a folder into the file browser for project navigation;
- switch between editor and visualizer views for the active netlist;
- import prior `.csv` or `.tsv` result data for GUI-side waveform inspection;
- preserve recent files, recent places, pinned roots, and session state between launches.

Scope note. The visualizer is a viewing and layout tool for existing netlists. It is not documented as a full schematic-capture replacement in the current release.

5.2 Creating, opening, and saving files

The `File` menu is the primary workspace entry point. Use these actions for normal file work:

File > New Creates a new untitled netlist tab. The tab must be saved before the first simulation run.

File > Open... Opens a JoSIM netlist or result data file. The dialog title is `Open netlist or result data`.

File > Open Folder... Changes the file browser root to a selected directory. It does not automatically open every file in that directory.

File > Save Saves the active editor tab. Untitled tabs prompt for a `.cir` save path.

File > Save As... Saves the active netlist tab to a new `.cir` path.

File > Save All Saves every modified netlist tab, prompting where needed.

The normal open path accepts these file types:

- `.cir`: opened as an editable netlist tab;
- `.csv`: opened as imported result data;
- `.tsv`: opened as imported result data.

5.3 Browse projects and recent files

The file browser provides a collapsible project tree on the left side of the workspace. Set its root with `File > Open Folder...` or by entering a directory path in the address bar. Double-click a supported file to open it. Double-click a directory to navigate into it.

The browser lists folders, JoSIM netlists, and imported-result file types. Hidden entries and unsupported files are omitted unless the GUI preference for hidden entries is enabled. The browser recognizes the same supported open-file set as the normal open path: `.cir`, `.csv`, and `.tsv`.

The file browser also maintains navigation memory:

- **Pinned Roots**: persistent project roots selected by the user. Use the pin control or `Ctrl/Cmd+P` on a selected folder.
- **Recent Places**: directories used by recent open or save operations.
- **Recent Files**: supported files recently opened through the GUI.

Use `Alt+Left` and `Alt+Right`, or the browser right-click menu's `Back` and `Forward` actions, to revisit folders within the current session. A new folder visit replaces forward history. Unavailable folders leave the current folder unchanged. Long status paths use an ellipsis; hover over the status to read the full path.

The browser right-click menu provides `New netlist`, `New folder`, `Duplicate file`, and `Rename`. New netlists use the `.cir` extension. Creation, duplication and rename reject existing destination names. Rename is also available with `F2`. Close affected workspace files before renaming or trashing a file or folder. A folder containing open files, the current working directory, its ancestors and filesystem roots cannot be renamed or trashed. Duplicating copies the saved file contents.

Use `Preview / details` to inspect the path, size, modification time and the first 4096 bytes of a supported UTF-8 text file without opening or changing it. Binary and unsupported file types show metadata only. `Move to system Trash` asks for confirmation and uses Windows Recycle Bin, macOS Trash or Linux desktop Trash. Restore items through the system file manager. An unavailable system Trash is reported as an error; the browser has no permanent-delete action.

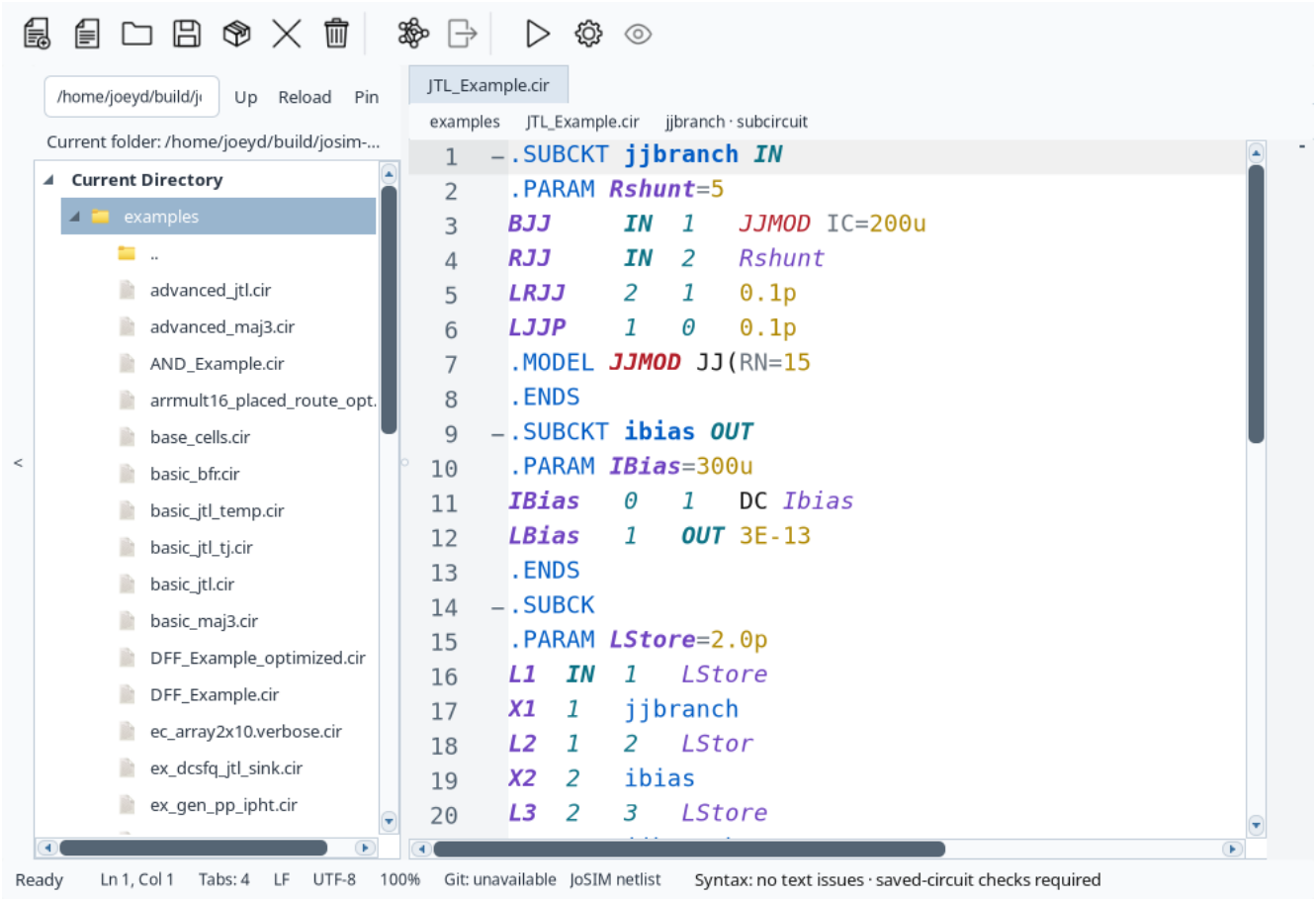


Figure 5.1: The examples browser, shared toolbar, source tabs and editor in one workspace.

Use pinned roots for stable project locations such as installed examples, active design directories, and shared benchmark folders. Use recent places for short-lived navigation.

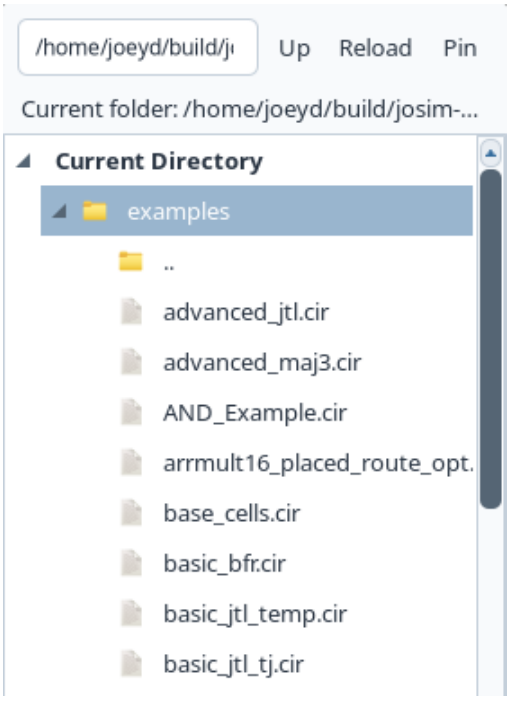


Figure 5.2: The examples tree with supported netlists; folders expand independently of source tabs.

Files can also be dragged from the operating-system file manager into the GUI window. Dropped `.cir`, `.csv`, and `.tsv` files are routed through the normal open path. Unsupported dropped items are ignored and reported in the context/status area.

This behavior is useful for reopening CLI-produced data in the GUI. For example, a `josim-pro-cli` run can write a `.csv` file, and that file can then be dropped into the GUI for waveform inspection and export.

5.4 Arrange tabs and editor groups

Use `View > Editor groups` to split the editor right or down. Each group has its own tabs, and the divider resizes the groups. Splitting opens another view of the current netlist. Clicking an editor selects its group; New, Open, Save and Edit commands act in that group. `Ctrl+Tab` switches tabs within the active group.

Opening an already-open netlist reuses its view in the active group, or adds a shared view there. Views share text, document identity, dirty state and one Undo/Redo history. Each keeps its own caret, selections, scroll position and folds. Edits from different views have separate typing Undo boundaries. Saving either view saves the shared document and clears both dirty indicators.

Closing one view keeps the document and history in the remaining view. Unsaved-change approval is needed when the last view closes; Close All asks once per document. `Single editor group` retains unique tabs and unsaved buffers, and merges duplicate idle views. Imported result tabs are not split. Split layouts, tab order and pinned/preview state persist across restart. Recovery retains one text entry per document with independent state for each split view.

Use `View > Editor tabs` or right-click a tab for grouped Navigate, Move, Keep and Close commands. `Ctrl+Page Up/Down` selects the previous/next tab; `Ctrl+Shift+Page Up/Down` reorders it within its pinned or unpinned section. `Ctrl+F6` and `Ctrl+Alt+Shift+F6` switch editor groups. `Ctrl+Alt+F6` moves the current view to the other group, creating a right-hand group if needed. Moving preserves its content and history and merges an idle duplicate view in the destination.

`Ctrl+Shift+F11` pins or unpins a tab. Pinned tabs appear first with a `[pin]` label and are protected from Close Others, Close Right and Close Saved. Explicit Close Tab and Close All still close pinned tabs after the usual checks. Source navigation opens a `[preview]` tab when the target is not already open in that group. A later navigation replaces a clean preview only after the new file opens successfully. Editing, pinning, double-clicking the tab, or Keep Preview Open makes it permanent; normal File Open also keeps the tab. Use as Preview Tab explicitly marks a clean, unpinned, idle tab as replaceable.

`Ctrl+Shift+F12` reopens the most recently closed tab; up to 20 tabs are remembered during this session. Clean files reload their current disk contents. Unsaved or untitled buffers retain their text; an existing open document always takes precedence over a closed snapshot. Reopening restores the caret, selection, scroll and folds, and the former group if it still exists. Missing files leave the history entry available for a later retry. `Ctrl+Shift+T` retains its existing Transpose Text command.

Close Others and Close Right apply to the active group. Close Saved covers both groups and skips dirty or pinned tabs. Bulk close checks all selected views before asking about unsaved changes, asks once per document whose last view would close, and removes no tabs if any prompt is cancelled. Choosing Save may save a document even if a later prompt is cancelled.

Imported `.csv` and `.tsv` result tabs are for waveform inspection, Trace Manager analysis, and plot export. They are not source netlists and cannot be simulated; open the corresponding `.cir` file to run a simulation.

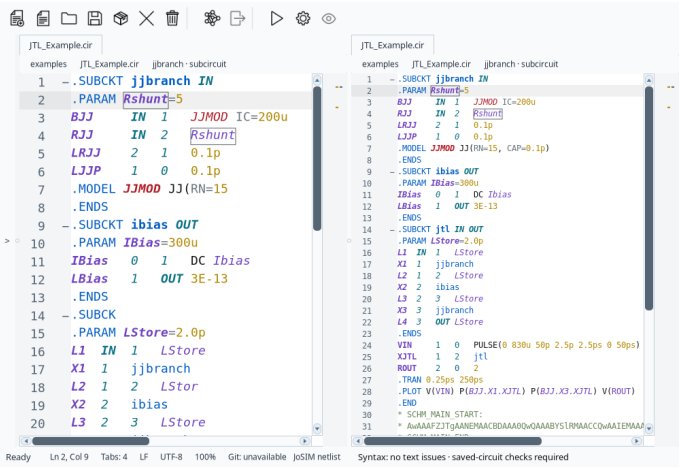


Figure 5.3: Two views share the netlist and Undo history; their carets, scroll positions and folds remain independent.

5.5 Use the toolbar and command palette

The toolbar provides shortcuts for common file, view, run, trace, and export operations. Some buttons are disabled until the active tab supports the requested action. For example, schematic export is available only while the visualizer is active, while plot export is available when waveform output is ready.

`F6` and `Shift+F6` move between the toolbar, browser address, file tree and active document view. Use `Ctrl/Cmd+Shift+E` to show the browser and focus its address, and `Ctrl+Tab` or `Ctrl+Shift+Tab` to switch documents on all platforms. A schematic tab retains schematic focus; an imported-results tab retains its plot view. Toolbar buttons accept keyboard focus and `Space` activation. Inside the editor, `Tab` still indents; use `F6` to leave that region.

Schematic brackets (`[` and `]`) select components and reveal their nodes and parameters. Use `Enter` to enter a selected subcircuit and `Backspace` to return. In the plot list, `Up/Down` select a plot; `Left/Right` inspect adjacent samples and `Home/End` inspect the first or last sample. Sample context uses each trace's axis and physical units.

On Linux, screen readers can read committed netlist text and its selection. Schematic components expose their parameters and connected nodes; subcircuits provide an open action. Plot samples expose each trace's value, axis and physical units, with a select action for inspection.

Common dialogs fit their display work area. The analysis-block dialog scrolls its fields and read-only preview while retaining its action footer. The preview uses a platform monospaced font.

The export toolbar button is context-sensitive:

- in visualizer view, it opens the `Export schematic` workflow for PNG or SVG schematic export;
- in waveform/result view, it opens the plot export workflow for result figures or data files.

5.5.1 Find an action

Open `Tools > Command palette` or press `Ctrl/Cmd+Shift+P` to search the available menu actions by name or category. The palette includes file, editor, language, workspace, view and tool commands. Each row shows its category and assigned shortcut, with `Unavailable` for disabled actions and `On` for checked actions.

Use Up/Down to select, Enter or **Run command** to execute, and Escape to cancel. Cancellation restores focus without editing the source. Commands run through their existing handlers, including normal save prompts and license/setup requirements.

The GUI refreshes command state before execution; if the active view or availability changed, reopen the palette. No circuit checks or simulation run merely by opening it.

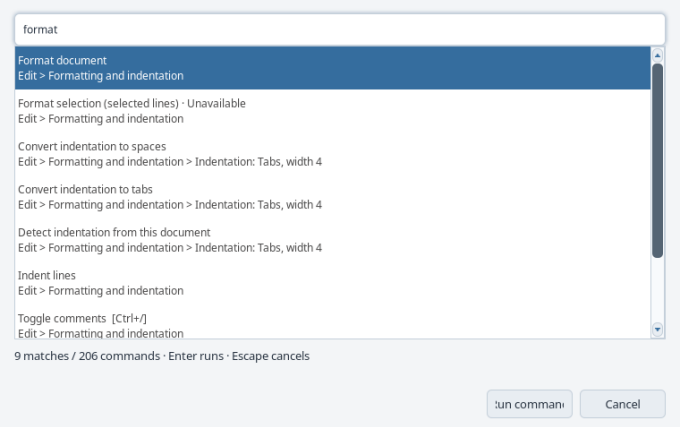


Figure 5.4: Search by action or category. The palette shows the effective shortcut and whether a command is available.

5.6 Read the application status bar

The application uses one compact status bar. It shows workspace or active-view state outside the editor. When the editor or one of its status actions has focus, it shows the physical `Ln`/`Col`, selected length, additional cursor count, indentation, line endings, encoding, `JoSIM netlist` language, disk access and editor zoom.

Caret movement, character deletion and selection keep combining marks and emoji sequences together as Unicode grapheme clusters. Text stays unnormalized; stored columns and source offsets remain exact. The fixed display grid retains its UTF-16 cell advance, with tabs expanded to configured tab stops.

Position and selection counts use UTF-16 units; each logical newline counts once and overlapping selections count once. Activate the position field to enter a one-based `line:column`; navigation reveals folded targets. Activate `Spaces`/`Tabs` to choose width 1–16, insertion mode, indentation detection or explicit conversion.

Conversion uses normal Undo; width and insertion mode do not rewrite source. Persistent defaults remain in Preferences. Status actions are flat text links with hover and keyboard-focus feedback.

The bar retains one row in narrow windows; the overflow menu keeps additional details and actions available. Hints provide complete values, including analysis budgets. Switching focus does not change the source height.



Figure 5.5: Two details of the status bar: position, indentation, file format and zoom, followed by language and analysis state. Activate a flat link to change its value.

5.7 Protect source files

Read-only locks, explicit file-format choices and guarded saves protect the original source. Review them before overwriting a file.

5.7.1 Read-only sources and editable copies

Read-only files open locked. The status bar and tab caption visibly identify read-only documents. For writable library sources, toggle `Edit > Editor view > Read-only mode`. This lock applies to all views of the document during the current session. Disk protection cannot be removed with this toggle; permission changes are checked before editing and reflected on focus and periodically. Editing, Undo/Redo, replacement, formatting, snippets, refactoring and Save commands are disabled while locked. Selection, Copy, search, navigation and explicit Reload disk remain available.

Choose `Edit > Editor view > Create editable copy` to open a separate, unsaved, editable tab. It keeps the source text, encoding, line endings and view position, with its own document identity and Undo history. The original remains unchanged.

Save the copy to a different path; saving it over its remembered original path is refused even after closing the original tab. The manual lock survives recovery checkpoints; copies saved to disk are ordinary documents when reopened. `Disk missing` describes a missing source whose buffer is still editable. The analysis row retains checking, stale, incomplete, failed and completed states, with budget details in its hint; it never invents a percentage or starts a simulation.

5.7.2 Choose line endings and encoding

The line-ending and encoding buttons show the format that Save will use. Loading preserves LF, CRLF or CR, UTF-8 with/without BOM, and BOM-marked UTF-16 byte order. The loaded final-newline presence is retained. Activate the line-ending button to convert the whole document or toggle `Append final newline`; activate the encoding button to choose UTF-8 without BOM, UTF-8 with BOM,

UTF-16 LE or UTF-16 BE. These conversions use document Undo/Redo, mark the document dirty, and apply to all shared views. Internal caret and search offsets do not change. Save As uses the selected format too.

UTF-8 is the default when no BOM is present; legacy code pages are not guessed. Malformed UTF-8, incomplete UTF-16 units, unpaired surrogates and unsupported UTF-32 BOMs produce an explicit error before replacing the editor buffer. Saving also rejects character loss before opening the target file. A literal Unicode replacement character is valid text and is not itself a decoding error.

The encoding menu also offers `Reopen with encoding` for UTF-8 or UTF-16 LE/BE, including BOM-less UTF-16. Reopening requires a saved, unchanged file; it refuses unsaved edits, changes on disk and a choice conflicting with the BOM. Reinterpretation uses one Undo step across shared views and remains dirty until saved. Save encoding conversion changes output bytes without reinterpreting text. No legacy encoding support is currently required by the supplied netlists.

`Mixed` reports multiple line-ending styles. An unchanged mixed file saves byte-for-byte. After editing mixed source, choose LF or CRLF explicitly before saving; the editor refuses the write until then. Undoing the edit restores exact mixed-file saving. BOM-only conversion of unchanged mixed source preserves its individual endings. The final-newline setting adds an implicit ending after the last editor row; disabling it does not delete explicit blank rows or newlines typed into the text. Changing this setting requires a uniform ending choice. Recovery copies retain encoding, ending style and final-newline settings; a recovered mixed copy requires explicit conversion because its original separator bytes are unavailable. Files recovered from older checkpoints without format metadata use the existing default format. JoSIM runtime decks should use UTF-8; UTF-16 preservation/conversion is editor file handling, not runtime encoding support.

5.7.3 Understand safe saving

Save writes the complete encoded document to an exclusively created sibling file, flushes it, then replaces the destination. It never falls back to truncating the original. On local POSIX filesystems, replacement publishes the complete source with one rename; existing readers retain the original file until they reopen it. New files are published without overwriting a concurrently created destination. Existing POSIX ownership, permission bits, ACLs and extended attributes are preserved; inability to preserve metadata aborts the save. The destination folder must permit creation and replacement, even when the source itself is writable.

Windows uses the native replacement operation with permission/metadata merging and an original-source backup. A replacement error reports any retained backup path. Save refuses read-only files, symbolic/reparse links, hard-linked files and special files; create an editable copy of a read-only source before saving to another ordinary file. A failed save keeps the editor content, document path and dirty state. Temporary-file cleanup is best effort; a process interruption can leave a hidden `.josim-save-` sibling. Atomic publication is not a guarantee against power loss, network filesystem behavior or concurrent writers. Platform failure details remain in the Save Failed message.

5.8 External file changes

The workspace watches opened files for external changes. Both clean and dirty netlists require a reviewed choice before refreshing; clean files are not silently reloaded. Use `Edit > Navigation > Compare to disk` at any time. The dialog shows a changed-region diff, read-only editor/disk panes and an editable merge preview, with detected encoding and line endings.

- **Reload disk**: replaces the buffer with disk text and its encoding, line endings and final-newline state. Reload is one undoable edit. Undo restores local content as unsaved relative to the reviewed disk version.
- **Keep editor**: preserves local text and format, acknowledges the reviewed disk version and marks differences unsaved. A later explicit save can overwrite that reviewed version.
- **Apply merge**: combines separate changed-line regions automatically. Overlapping regions retain local text for manual editing. Review the preview and check the confirmation before applying it as one undoable buffer edit. Merge keeps editor encoding/line endings and never writes disk automatically.
- **Cancel**: preserves the buffer and leaves reconciliation pending. The same automatic prompt is deferred; protected actions ask again.

Applying a choice rechecks the editor revision and disk content hash; a stale review applies nothing. Shared editor views receive the same content, disk baseline and Undo history while retaining their own view positions. Missing, malformed or oversized disk sources preserve local content and report the reason; use Save As when reconciliation is unavailable.

Review is limited to 2 MiB of source bytes and about two million text characters per pane; diff output is bounded and may be truncated. Distributed/overlapping changes are deliberately conservative. Background polling uses file size and modification time; protected actions and save also check content hashes, including equal-size/time changes. These checks do not lock out a writer changing the file during the final save.

Imported result files retain their existing reload/keep prompt.

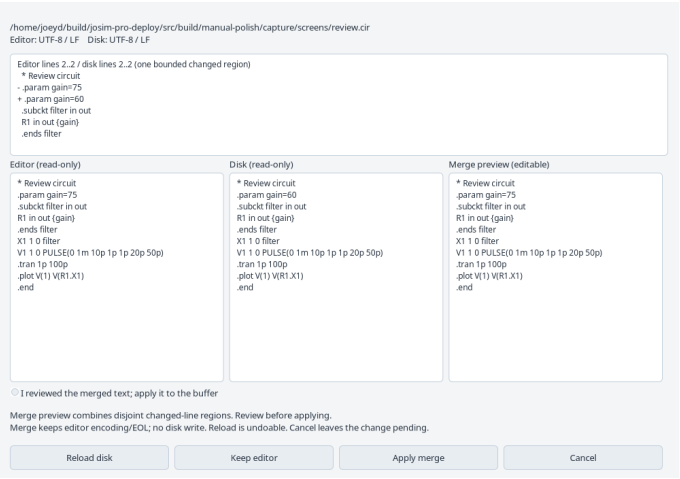


Figure 5.6: Compare local edits with the current disk source before choosing Reload, Keep editor or a reviewed merge.

5.9 Restore a workspace and recover edits

The GUI stores user workspace configuration in the user's configuration directory. Persisted state includes the working directory, window size/state, file-tree panel width and visibility, theme, visualizer grid style and size, waveform grid and synchronized-crosshair preferences, hidden-file browser preference, and whether workspace tabs are restored on launch.

When session restore is enabled, the GUI reopens persisted tabs on startup. Session state records both editor groups, their orientation and divider ratio, active tabs, tab order, pin/preview state, editor positions, selections and view modes, waveform preferences, and analysis recipes. Each editor view retains zoom, indentation and detection policy, wrapping, read-only state and folds. Global editor preferences remain in the Preferences settings. If files are missing or fail to reload, startup continues and the GUI reports a `Session Restore Incomplete` warning rather than treating the problem as a fatal launch failure.

`Close All` and application Quit collect the required save or discard decisions before removing tabs. Cancelling a prompt keeps every tab open and restores the active tab. Files already saved in response to an earlier prompt remain saved.

Saved-tab session state stores file paths, editor positions and view modes. Separately, unsaved editor buffers are checkpointed in the configuration directory every ten seconds and when workspace session state is persisted. When session restore is enabled, these buffers reopen as modified, unsaved copies with their caret, selection, scroll position and editor or visualizer view.

The active unsaved copy takes focus after restoration. Split recovery views share one recovered buffer, with independent selections, scroll positions and settings. Recovered buffers never overwrite the original file; shortened saved files clamp restored positions to the current text.

Edits since the last successful checkpoint may be lost after a crash. A failed checkpoint retains the last successful snapshot. Invalid recovery data is retained and reported. Folds reopen only when the source matches the checkpoint and current syntax analysis confirms their ranges; changed or malformed source remains visible.

Invalid saved view metadata falls back to defaults with a warning; invalid recovery metadata retains the journal. Text and cursor Undo/Redo histories are session-only: files reopened after startup and recovered copies start with empty history. Local save history remains available for saved versions.

Keep moved or renamed project files reachable if you expect saved tabs to reopen automatically.

5.10 Choose an autosave policy

In `Tools > Preferences > General > Autosave (writes named source files)`, choose `Off` (the default), `After typing delay`, `When leaving an editor`, or `When leaving the window`. The delay is 1000–60000 ms (default 2000 ms), measured from the last text or format change. Focus policies queue a save attempt for the next safe timer check. Checks run about once per second and attempt at most one document per check; active dialogs, simulations, multi-file edits, snippet placeholders and text composition defer writes. Shared views save one document and retain Undo.

Autosave writes only existing, named, editable netlists with a verified disk baseline. It never guesses a destination for an untitled, recovered or editable copy buffer, never opens Save As, and never silently resolves a disk conflict. The status bar shows waiting, saved, untitled, read-only or blocked state; hover for details. Missing, externally changed, malformed, oversized or unwritable sources stay dirty. A blocked attempt is not retried continuously; editing, explicit reconciliation, lock changes or changed autosave settings rearm it. Read-only and imported result sources are skipped.

Source review is limited to 2 MiB of disk bytes and 2,097,152 UTF-16 buffer units. Save uses the selected encoding, line endings, enabled save formatting and the normal safe-save rules; Undo after autosave can make the document dirty again. The byte checks do not lock out a concurrent writer during the final save.

Recovery checkpoints are separate: they preserve dirty named and untitled buffers in the configuration directory without writing their source files. Disabling autosave keeps checkpointing enabled. Checkpoints refresh every ten seconds and when session state is persisted; a completed autosave is reflected at the next checkpoint.

5.11 Compare and restore local save history

In the editor's `Navigation` menu, choose `Local save history` to browse versions captured by successful Save, Save As and autosave. History preserves exact source bytes, encoding, BOM and line endings. The first save also captures the previous file when it fits the snapshot limit; unchanged consecutive versions are deduplicated. Each full file path has its own timeline, so Save As starts a separate destination history. Opening, checkpointing and editing an untitled buffer do not create save history.

Select a version to compare the current buffer and read-only saved source. After checking the review confirmation, `Restore to buffer` replaces text and file format in one shared Undo transaction. It does not write the source; normal Save or the configured autosave policy writes the restored buffer later. A changed buffer or path requires reopening the review. Read-only sources can be compared but cannot be restored into; use an editable copy instead.

`Move to Trash` removes an entry from the timeline without deleting source. Enable `Show deleted versions (Trash)` and choose `Recover from Trash` to return it and restart its active retention period. History lives in the configuration directory separately from crash recovery. It retains up to 30 active versions per path for 30 days, plus up to 30 deleted versions for seven days, within shared limits of 8 MiB per path, 64 MiB overall and 4096 entries. Each snapshot is limited to 2 MiB; oldest entries, including Trash, expire when budgets are reached. This is local history, not a backup guarantee. A history storage failure leaves a successful source save intact and displays a status warning with details.

Writing and Editing Netlists

Use the editor to author the actual JoSIM netlist. Completion, signatures and snippets help with supported syntax; they do not choose circuit values for you. Unless noted, Ctrl is the primary modifier on Windows/Linux and Cmd on macOS. Tools > Keyboard shortcuts shows effective bindings.

6.1 Start with the source

A `.cir` netlist opens in the text editor. The editor is designed for direct netlist editing and review, not for hiding the source syntax. It provides:

- line numbers and current-line indication;
- syntax highlighting for JoSIM netlist structure;
- bracket match and bracket error highlighting;
- live text-syntax diagnostics and context-sensitive completion;
- search highlighting;
- find and replace with next/previous navigation;
- copy, cut, paste, delete, select-all, undo, and redo;
- keyboard navigation, word deletion, tab insertion, and line movement;
- modified-state tracking so unsaved tabs are visibly distinct.

The editor accepts desktop input-method composition. Marked text is provisional until the input method commits it; live static analysis and recovery use committed source. Cancelling Linux composition retains the original selection and source. Committing a replacement is one undo step.

6.2 Complete fields and include paths

1. Place the caret in the field you want to complete and press `Ctrl/Cmd+Space`.
2. Use `Up/Down` to select a suggestion. Its kind and reference explain what it inserts.
3. Press `Tab` or `Enter` to replace the token, or `Escape` to leave it unchanged.
4. Use `F1` for the full reference or `F12` for an available declaration link.

Use `Ctrl/Cmd+Space` for suggestions. Suggestions also appear after the configured typing pause when entering a directive or a prefix of at least two characters, once static analysis is current. The list includes supported directives/devices and suitable models, parameters, nodes or subcircuits from the current document, included source and scope.

Expression positions offer supported mathematical functions; source positions offer waveforms; model definitions offer supported types and type-specific fields. `.OPTION` and `.OPTIONS` offer runtime option names and aliases, with boolean, analysis and integration values where applicable. `.TRAN` offers the supported `STARTUP`, `DST` and `UIC` flags. Completion suggestions appear beside the caret without reducing the text viewport. The list fits above or below the caret and moves inside the visible editor bounds.

Each item shows its kind and usage; hover an item for its full text. Matching ignores ASCII letter case and accepts letters in order with gaps. Exact matches rank before prefixes, then fuzzy matches; within each class, local scope and nearby declarations in the current file take priority over included symbols and built-in vocabulary.

Explicit requests retain an exact match; accepting unchanged text creates no edit. Use `Up/Down` to select, `Tab` or `Enter` to insert, and `Esc` to dismiss. Double-click also inserts. `Shift+Tab` dismisses suggestions and unindents.

Scrolling the caret out of view, changing the selection or leaving the editor dismisses the list. Acceptance replaces the entire current token and can be undone in one step. In Preferences > Assistance, *Editor suggestions* controls automatic suggestions, a delay of 0–2000 ms (default 300 ms), and Tab/Enter acceptance independently. A disabled acceptance key performs its normal editing action. `Ctrl/Cmd+Space` bypasses the typing delay and remains available when automatic suggestions are disabled.

In a `.INCLUDE` path (or `include` inside a control block), suggestions show files and folders relative to the containing source file. Save an unnamed source first to use relative paths; absolute paths remain available. Type part of a filename or folder and use `Ctrl/Cmd+Space`; automatic suggestions follow the same delay preference.

Filename prefix matching ignores ASCII letter case and preserves each filename's actual spelling. Choosing a folder adds a slash and opens its suggestions. Choosing a file replaces the full path argument in one undo step, preserves existing single or double quotes, and closes an unfinished quote.

Paths containing spaces gain quotes. Existing trailing comments remain intact. Inserted separators use forward slashes on Windows, except a UNC root retains backslashes because the runtime treats `//` as a comment.

Directory listing runs in the background and inspects one folder at a time. Results are bound to the source, document revision and caret; edits, navigation or `Esc` discard pending results. Checking, unavailable, empty and partial listings appear as information rows that cannot insert text.

Refresh suggestions after a directory changes. Each request stops after 256 matching entries, 4,096 scanned entries or 150 ms of listing work; a partial-listing notice discloses a reached limit. A filesystem operation already in progress may take longer, but it does not run in the key handler. Sym-

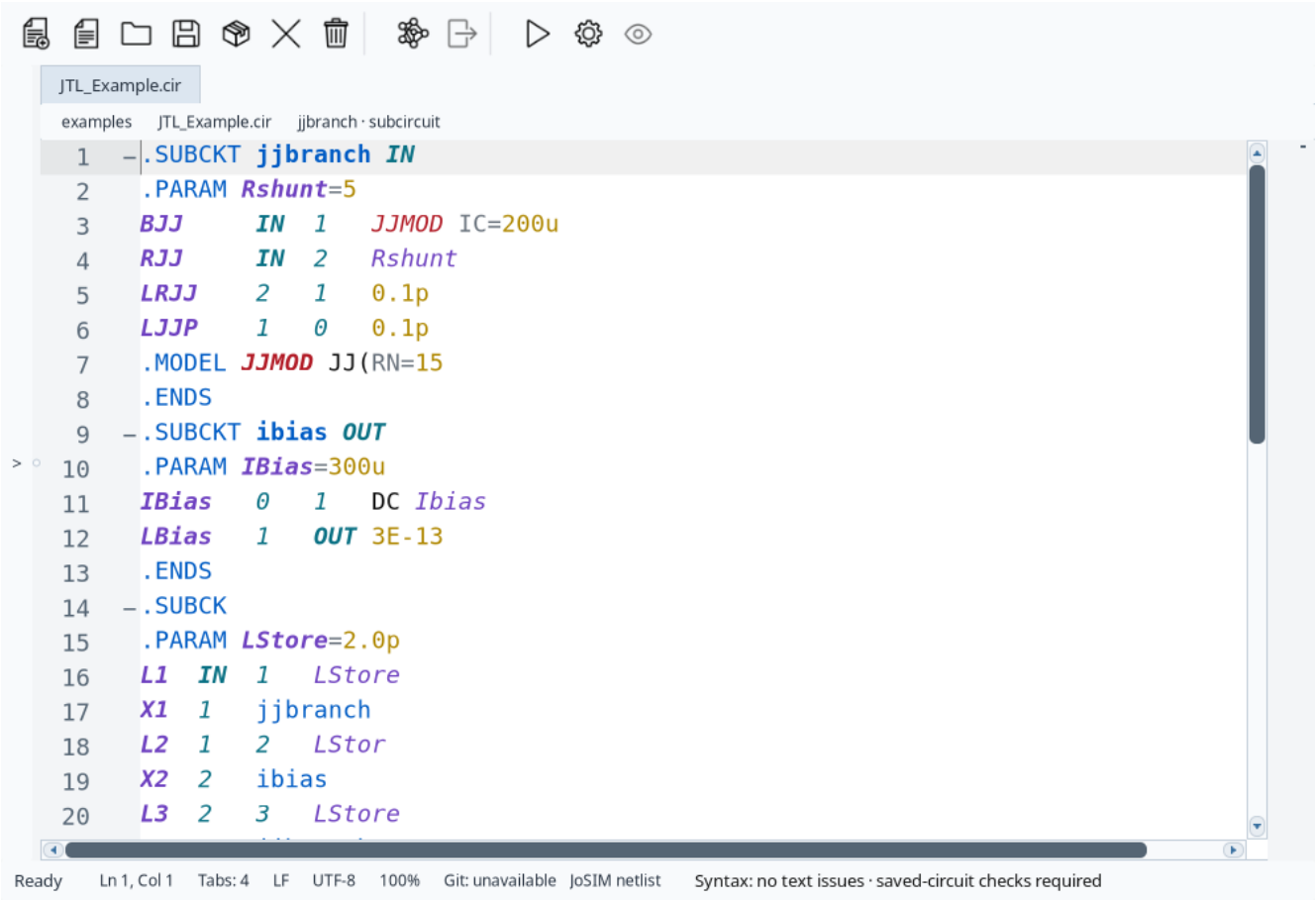


Figure 6.1: The JTL example with syntax colours, real scope breadcrumbs and source line numbers.

link, reparse-point and system entries are omitted; names beginning with a dot require a dot prefix.

Candidate file contents are not read by listing, and folders are not recursively scanned. Names containing control characters, repeated spaces or an unrepresentable quote are omitted because runtime path syntax cannot preserve them reliably. Paths in comments and unrelated strings do not trigger listing.

Completion follows the whole logical statement, including continuation lines. It suggests nodes in device terminal positions, compatible effective models, inductors for K references, subcircuit targets with matching port counts, and the selected target's parameter overrides. Override values use the caller's parameter scope.

Output suggestions distinguish arithmetic functions, signal targets and legacy selectors; power and energy targets exclude nodes. Known MTL conductor counts provide current selectors. Instance-qualified output names follow actual calls, including nested instances; mapped formal ports are omitted.

Hierarchical suggestions are bounded to 512 traversal frames, 512 names, 32 levels and 1,024 bytes per name. Reaching a limit shows an incomplete-analysis warning; it does not expand or simulate the circuit. Option suggestions follow supported names and values, including comma-separated and continued statements. ASCII letters compare without case; other Unicode letters retain their runtime case identity.

Accepting a suggestion replaces the current token in one undo step and preserves adjacent arithmetic operators.

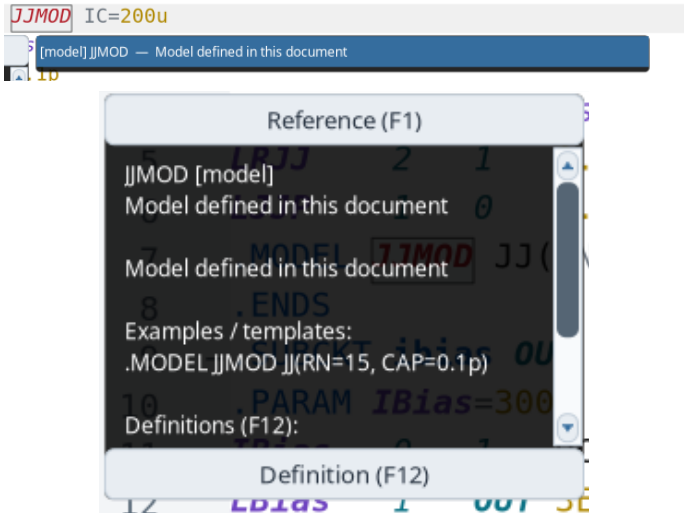


Figure 6.2: Completion in the junction model field (top), with the selected model's syntax, example and definition link (below).

6.3 Inspect arguments and symbol information

Signature help follows the caret in supported waveform and parameter-function calls, and in positional device or known subcircuit fields. It shows the complete signature and names the active argument with its role, units and optional status. PWL/PWS time/level pairs retain their pair number; grouped expressions, quoted paths and continued physical lines retain their argument position. Transmission-line and `@MTL` named fields are identified by name, including reordered fields. Known subcircuit targets use the definition's formal port order and supported parameter overrides.

An unresolved target reports that port order is unavailable. Use `Ctrl/Cmd+Shift+Space` to request help and `Escape` to dismiss it. Help fits within the text viewport, hides when space is insufficient, and clears when source analysis becomes stale. It does not edit text or run simulation.

Hover over a device, parameter, model, node or directive to inspect its declaration, scope and usage. Known values identify whether they are literals, evaluated constants or subcircuit defaults that instance overrides may change. Numeric scale factors follow JoSIM's parser. Unknown, inactive and context-dependent values are labelled explicitly. Included declarations retain their source locations; occurrence counts apply to the current root deck and are qualified when analysis is incomplete. Diagnostics remain visible alongside symbol information. Press `F1` for the full information at the pointer or caret. Hover information clears when its source changes and does not run a simulation.

Selecting a suggestion shows a read-only reference beside the popup when space permits. References include syntax, argument roles, units, aliases and examples from the shared JoSIM-Pro language catalog. Press `F1` while suggestions are open for the full reference. Press `F12`, or use the Definition button, to navigate to a source declaration. When several definitions are ambiguous, choose a location explicitly. Source links preserve exact source ranges and use the editor navigation history. Parameter units and unknown values remain context-dependent; references never run a simulation. Large decks may omit optional documentation while retaining semantic analysis. On narrow viewports the details pane hides; the full reference remains available through `F1`.

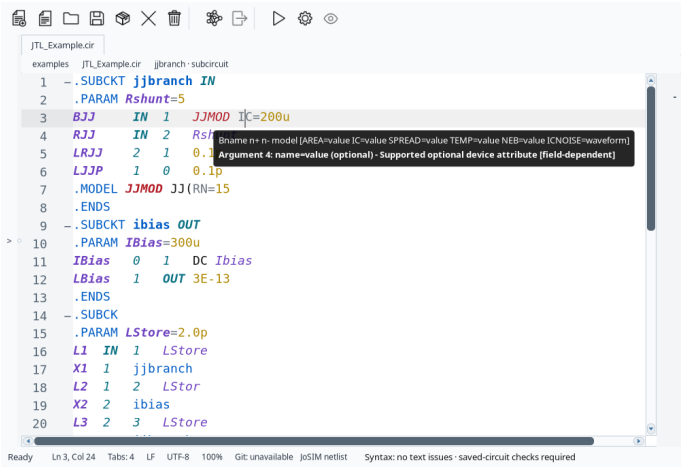


Figure 6.3: Signature help lists junction fields and identifies the active argument, its role and its units without changing the circuit.

6.4 Insert a snippet

Choose *Edit* > *Insert snippet* (also available in the editor context menu) for grouped device, source, model and block templates. The 47 built-ins cover R/L/C, Josephson junctions, K references, lossless/wideband T lines, `@MTL`, controlled sources and subcircuit calls; DC, PULSE, SIN, PWL/PWS, EXP, NOISE, CUS, PNOISE, FNOISE, STAIR and junction ICNOISE forms; JJ/RCSJ, TJM, WIDEBAND and MTL models; parameterized `.SUBCKT`, `.CONTROL`, transient/IV analysis, output selection, parameters and options. Editable defaults are selected in field order. Use `Tab` and `Shift+Tab` to move between fields; repeated fields update together, including the subcircuit's matching `.ENDS` name.

Choice fields show their alternatives in the snippet status line; use `Alt+Up/Down` to cycle, or type a custom value. `Escape` ends the session and keeps the inserted text. Reaching the final stop or moving the caret outside the current field also ends the session. One Undo restores the text and selection from before insertion; Redo restores all edits without reopening the field session. Automatic suggestions pause during the session; `Ctrl/Cmd+Space` remains available explicitly.

6.4.1 Add user templates

Choose *Insert snippet* > *Edit user snippets* to add or edit custom entries. The library is UTF-8 JSON at `.josimpro/netlist-snippets.json` in your home directory. You may also edit that file externally. Save validates the whole library; Cancel leaves it unchanged. The next menu opening reloads it under *User snippets*. Invalid libraries leave built-ins available and expose a repair action. A file changed since the dialog opened cannot be overwritten; close and reopen the dialog. A changed menu entry must be selected again from a freshly opened menu. Saving a library does not edit or save the netlist.

Use an array of objects with `name`, optional `description`, and `body` as text or an array of lines. For example:

```
[
  {"name":"My resistor","description":"50 ohm default",
   "body":"R${1:custom} ${2:node} ${3:0} ${4:50}$0"},
  {"name":"My block","body":[".SUBCKT ${1:cell} a b",
    "R1 a b ${2|50,100|}",".ENDS $1","$0"]}
]
```

Fields use `$n`, `${n:default}` or `${n|first,second|}`; `$0` is the final stop. Escape a literal dollar, brace, comma in a choice, or backslash with a backslash (and escape that backslash again in JSON). Nested fields, variables and transformations are unsupported. Names are unique without case and contain 1–80 characters; descriptions have at most 512 characters. Limits are 128 entries and 1 MiB per file, 65,536 UTF-16 units and 256 explicit fields per body, field numbers 0–999, 64 alternatives per choice and 1,024 body lines. Unknown object fields and malformed entries reject the whole library. Templates only insert text: choose actual nodes, targets, model types, port/matrix sizes and paths for your circuit. The CUS template needs a readable waveform file; TJM fit data must be supplied separately when required. Inserting a snippet does not load those resources or run a simulation.

6.5 Use optional language annotations

6.5.1 Inlay hints

Tools > Preferences > Assistance provides separate switches for positional port names and known values/units. Both are off by default. Display-only hints appear after the physical source line, or its final wrapped row. Subcircuit calls show formal port names paired with actual nodes only when the target and port count resolve unambiguously. Values come from bounded static analysis; local subcircuit values are labeled `default`, since instance overrides may change them. Unknown, invalid and policy-dependent values have no value hint. Units describe documented passive-device or supported analysis argument roles; parameters do not acquire an inferred dimensional type.

Hints disappear while their analysis is stale. They are clipped to the editor, limited to 259 UTF-16 units per line and 4096 items per document, and never enter selection, clipboard text, saved netlists, recovery content or text Undo. The two switches support the usual user, netlist-language and project precedence.

6.5.2 Code lenses

Tools > Preferences > Assistance provides `Show code lenses`, off by default. Declaration annotations show root/include reference counts and, for subcircuits, source instance counts. These count written call sites, not the expanded instance hierarchy. Possible ambiguous/context-dependent uses are separate; incomplete analysis is labeled. The first explicit declaration on a physical line owns that line's lens; use Find All References for other declarations. Click a count to open References. The first source line provides explicit `Check saved circuit` and `Run circuit` links, using the usual save, license and simulation guards. They act on the active circuit, not a subcircuit definition; analysis never runs it automatically. Run follows the normal simulation command, including displaying existing current results.

Links follow the final wrapped row and never enter source, clipboard or saved content. Stale lenses disappear until analysis completes. The editor's `Diagnostics and refactoring > Code lenses` submenu offers keyboard access to references at the current declaration and the check/run actions. The setting supports user, netlist-language and project precedence.

6.5.3 Inline template suggestions

Tools > Preferences > Assistance > Language annotations provides `Show inline suggestions`, off by default. A muted statement tail and `Tab` label identify a shipped snippet template. These are example defaults, not inferred circuit values. Supplied names, nodes and fields are preserved; missing targets, paths, ambiguous tails, comments and quoted expressions have no suggestion. Common numeric device tails, waveform arguments, transient analysis, model fields and supported option defaults use the snippet library. Suggestions also follow continued statements on the last wrapped row.

Press Tab to accept the visible tail in one undo step; Escape dismisses it until an edit or caret movement. Completion popups and active snippet tab stops take priority. Selection, multiple carets, read-only/overwrite mode and native marked text suppress suggestions. Ghost text never enters clipboard, disk, recovery, source offsets or history until accepted. Acceptance rechecks source revisions. The boolean setting supports user, netlist-language and project precedence.

6.6 Edit lines and literal text

The `Edit` menu and editor right-click menu also provide whole-line commands:

- `Go to line...` (`Ctrl/Cmd+L`) moves to the start of a numbered line.
- `Select line` (`Ctrl/Cmd+Shift+L`) selects the current or touched physical lines, including the following line break when present. Repeating the command extends to the next line.
- `Insert line above/below` inserts an indented blank line without splitting the current line or replacing selected text. Use `Ctrl/Cmd+Shift+Enter` above and `Ctrl/Cmd+Enter` below. The caret moves to the new line.
- `Duplicate lines upward/downward` (`Alt/Option+Shift+Up`/`Alt/Option+Shift+Down`) copies the current or touched lines

and keeps the caret or directional selection in the new copy.

- With no selection, `Copy`/`Cut` copies the complete current line and a line break; Cut removes that physical row. Selected fragments retain ordinary clipboard behavior. Pasted clipboard text follows the editor's normal paste and indentation settings.
- `Duplicate lines` (`Ctrl/Cmd+Shift+D`) copies the current or selected lines below them.
- `Delete lines` (`Ctrl/Cmd+Shift+K`) removes the current or selected lines.
- `Join lines` (`Ctrl/Cmd+J`) joins selected lines, or the current and next line, with a space while retaining the first line's indentation.
- `Toggle comments` (`Ctrl/Cmd+/`) inserts a netlist `*` after indentation on uncommented lines. If every nonblank target line is commented, it removes the first `*`. Blank lines are unchanged.
- `Trim trailing whitespace` removes trailing spaces and tabs from selected lines, or the entire document when there is no selection.

Whole-line commands use the primary caret/selection and return to one cursor. Undo restores preceding secondary ranges for inserted or duplicated lines. A selection ending at the start of a line excludes that line from whole-line commands. Each edit is a single undo step; undo restores the preceding caret and selection.

Double-click selects a word or punctuation character; drag from the double-click selection to extend by whole words. Click the line-number gutter to select a physical row, including its following newline where present; dragging in the gutter extends by whole rows, including backward selections. Diagnostic and fold markers retain their separate actions.

Drag inside a single nonempty selection to move its literal text within the editor. Hold `Alt/Option` when starting the drag to copy it. The insertion marker shows the target. Dropping inside the original range leaves source unchanged; `Escape` cancels. Text dropped from another editor or application is copied at the target. Dragging text to another application exports a copy where the platform supports text drag/drop. File drops retain their file-opening behavior. Drag/drop does not apply paste formatting or delete an external source. Each local move or copy is one Undo action.

Edit > Transpose characters (`Ctrl/Cmd+Shift+T`) swaps the Unicode scalar before the caret with the one after it. At the end of a row, it swaps the final two scalars. It does not cross newlines or transpose selected text. Multiple carets can transpose separate pairs; overlapping pairs reject the action. *Convert to uppercase/lowercase* changes selected text, or the word at each unselected caret, using Unicode case conversion. Overlapping ranges are changed once, and selection direction is retained. These are literal text commands; they do not perform semantic symbol renaming.

Sort selected lines ascending/descending sorts the physical rows touched by the primary selection, using case-sensitive text comparison. *Remove duplicate selected lines* keeps the first occurrence of each exact row, preserving case distinctions and the order of remaining rows. These commands require a selection; a range ending at the next row's column zero excludes that row. They retain the resulting row selection and support one-step Undo/Redo. Other cursors are restored on Undo. **Source order can change circuit behavior.** Sorting and duplicate removal are explicit commands only; no save, paste, formatter or analysis action invokes them. Check that the selected rows can be reordered or removed safely before using either command.

6.7 Work with multiple and column selections

Occurrence selection: Edit > Multiple cursors offers Select next occurrence (Ctrl/Cmd+D), Skip occurrence (Ctrl/Cmd+K), and Select all occurrences (Ctrl/Cmd+Shift+A). With an empty selection, next/skip first selects the word (or character) at the caret. Next adds the next case-sensitive literal match; skip moves the primary range to the next match and preserves earlier ranges. Both wrap and exclude ranges already selected or overlapping; exhaustion leaves the selections unchanged. All selects nonoverlapping matches in source order, including adjacent matches. Matching uses the current document, including comments, strings and multiline selections. It is separate from semantic occurrence highlighting and scoped Rename. No Find-panel option changes it. Selection alone does not modify text or add an undo entry. Replacing the ranges is one undoable edit and restores the full selection on Undo. All refuses more than 256 matches with a visible limit message instead of silently truncating; next also stops at 256 ranges. Use Find or a narrower selection for larger sets. Ctrl/Cmd+A still selects the whole document; Shift+D duplicates lines and Shift+K deletes lines with the Ctrl/Cmd modifier.

6.7.1 Add independent cursors

Use `Alt/Option+click` to add a cursor, then drag to select its range. **Edit > Multiple cursors** offers **Add cursor above**, **Add cursor below** and **Keep primary cursor**. The add shortcuts are `Ctrl+Alt+Up/Down` on Windows/Linux and `Cmd+Option+Up/Down` on macOS. The most recently added cursor is primary; repeated add commands keep their column across short lines. Up to 256 cursors are available.

Arrow keys and `Shift+arrows` move or extend each range. Typing, committed input, paste, Backspace/Delete, word deletion and overwrite apply independently to every range; Enter uses each range's syntax context and Tab uses its own tab stop at each range. Overlapping ranges and duplicate cursors merge before replacement, so source is edited once.

Each edit has one Undo/Redo action that restores all ranges.

Copy/Cut joins nonempty selected fragments in source order with line breaks; Multiline paste rebases indentation in-

dependently at each eligible range. Empty ranges add no clipboard fragment. `Escape` keeps the primary range, and an ordinary click starts a single selection. Whole-line commands and snippets use the primary range. Automatic pairs, completion and signature help are inactive while multiple cursors are present. Cursors remain in the open editor and its undo and navigation history; reopening/recovering a document restores its persisted ranges. New/load resets added cursors before applying saved view state.

6.7.2 Select a rectangle

Column selection: Hold `Alt/Option+Shift` and drag across the editor to select a rectangle. Use `Ctrl+Alt+Shift+arrows` on Windows/Linux, or `Cmd+Option+Shift+arrows` on macOS, to extend or shrink one from the caret. Edit > Multiple cursors also offers Column select up/down/left/right. Each visible row receives its own range; folded rows are excluded. Tabs advance to the next configured tab stop. Endpoints inside a tab or grapheme cluster snap to the nearer complete boundary, choosing the right boundary at the midpoint. Partial tabs are not expanded into spaces. Short and empty rows clamp at their ends: virtual space is disabled and editing never adds padding. Requested columns remain available when extending onto longer rows. A zero-width rectangle provides a cursor on each row. Typing, paste, Tab and deletion use one multi-selection edit; Copy/Cut joins nonempty fragments in source order. Undo restores all ranges and requested columns. View/navigation and fix checkpoints preserve the geometry during the session; recovered documents restore their primary, secondary and rectangular ranges. At most 256 visible rows are selected; a larger rectangle is refused with a visible limit message. Escape keeps the primary range and ends column selection; ordinary movement converts ranges to regular multiple selections, and a plain click establishes one selection. Editing ends rectangle mode while retaining the resulting cursors. Whole-line commands and snippets continue to use the primary range. Automatic pairs, completion and signature help remain inactive during column selection.

6.8 Expand and shrink a syntax selection

Expand/shrink selection: Use `Alt+Shift+Right` on Windows/Linux and `Ctrl+Cmd+Shift+Right` on macOS to expand from a token to enclosing quoted contents, bracketed expressions or fields, the complete logical statement (including `+` continuations), matched subcircuit/control blocks and finally the document. Equal ranges are skipped. Comments remain plain text; incomplete expressions stop at their statement, and crossed block boundaries do not create an enclosing block. Edit offers the same Expand semantic selection and Shrink semantic selection actions.

Use `Alt+Shift+Left` (`Ctrl+Cmd+Shift+Left` on macOS) to restore each previous range and finally the original caret. Backward selection direction and independent cursors are preserved. Expanding a rectangle produces ordinary syntax ranges; shrinking restores its original geometry.

Selected folded source becomes visible. Navigation, edits or a new selection end this shrink history; Copy preserves it. These actions do not modify the buffer or consume text Undo/Redo. Selection accepts at most 10,000 rows, 1,048,576 character positions and 65,536 characters per row, with 256 nested groups/blocks and 256 expansion steps. Oversized input remains selected as it was. This is local syntax selection; no include files or simulation are read or run.

6.9 Insert and navigate paired delimiters

Typing an opening parenthesis, bracket, brace or quote inserts its partner at safe token boundaries and places the caret inside. Typing a matching closing delimiter steps over it. Selecting text before an opening delimiter surrounds it and keeps the inner text selected; quote surrounding requires one line without the same quote or a comment marker.

Backspace or Delete between an empty pair removes both delimiters. Each operation supports Undo/Redo. Comments and quoted text keep literal bracket input. JoSIM treats backslashes literally and starts a comment at `//`, including inside quotes. Continuation lines retain the preceding statement's quote/bracket context. Overwrite mode, pasted text and IME commits use ordinary literal editing.

Use `Indent lines` and `Unindent lines` from either edit menu, or `Tab` and `Shift+Tab`, to adjust indentation. `Go to matching bracket` (`Ctrl/Cmd+Shift+Backslash`) moves between matching parentheses, brackets or braces, skipping strings and comments and revealing hidden targets. Unmatched brackets leave the caret unchanged.

6.10 Indent blocks and continuations

Indentation settings: Tools > Preferences > Editing > Editor indentation sets a tab width of 1–16 columns (default 4), whether Tab inserts spaces (default: tabs), and whether opening a file detects its indentation (default: off). A tab advances to the next stop, even in the middle of a line. Text, caret, selections, highlights and mouse positioning use the same stops. Vertical movement and Add cursor above/below retain display columns through short rows. Changing width updates layout without changing source text and ends a column-selection session while retaining its individual ranges.

With spaces selected, Tab inserts enough spaces to reach the next stop at each cursor. Selected-line indentation adds one configured unit; unindent removes up to one unit of leading spaces/tabs. Single-line clipboard text, committed text and snippet contents remain literal. Edit > Indentation shows the effective document mode and width, and offers

Detect indentation from this document, Convert indentation to spaces and Convert indentation to tabs.

Detection inspects leading whitespace on the first 10,000 rows, with at most 256 characters per prefix. Tabs/spaces use the majority of nonblank indented rows. A clear spaces pattern may infer widths 2, 3, 4 or 8; tab width cannot be inferred from tab characters alone. An ambiguous mode uses the configured defaults; an unsupported spaces pattern keeps the configured width. The status explains the result. Detection never rewrites text. Detected settings apply to the current document; new files use the configured defaults.

Conversion applies to leading indentation throughout the current document, preserving its displayed width. Tabs inside statements, strings and paths are unchanged. Conversion to tabs retains any remaining spaces before the next stop. Each conversion is one undoable edit that restores source and selection; its chosen insertion mode remains in effect until settings change or a file is opened. Conversion uses the primary selection, as other whole-line commands do. Save explicitly to write converted text to disk.

Automatic indentation: Enter after `.SUBCKT`, `.SUBC` or `.CONTROL` adds one indentation unit using the document's width and tabs/spaces mode. Ordinary lines retain their leading whitespace. Typing `.ENDS`, `.ENDC` or a control block's bare `ENDC` aligns the closing line with its matching opener. Enter also aligns an existing closer and the following line in one edit. Comments, quoted text and directive-like words in continuation arguments do not define blocks. Unmatched or crossed blocks retain their whitespace.

Typing a leading `+` indents the continuation one unit from its containing statement, including across blank/comment gaps. Enter after a continuation returns to the statement's base indentation, or to the body after a continued block header. Enter does not insert a `+` marker; add one explicitly when continuing a statement.

Multiline paste at a leading-whitespace position rebases the clipboard's minimum nonblank indentation to the destination and preserves relative indentation. An empty line immediately after a block header uses body indentation. Leading prefixes use the document's tabs/spaces mode; inline whitespace, tokens and quoted paths remain unchanged. Blank pasted rows stay empty. Paste inside a string/comment, after circuit text, or with an unfinished quoted clipboard statement remains literal. Paste does not format unindented block contents or add continuation markers. Source changes use one undoable paste transaction, including independent cursors and linked snippet fields.

Indentation matching inspects at most 2,048 previous rows and 65,536 characters; paste adjustment accepts at most 2,048 rows and 65,536 characters. Oversized or ambiguous input retains its whitespace. Undo restores source and prior cursor/selection state. Save remains explicit.

6.11 Format source deliberately

Formatting: Edit > Format document formats the current buffer. Edit > Format selection (selected lines) formats physical lines intersecting any nonempty selection; an endpoint at column zero excludes that final line. Overlapping selections format each row once. Unselected source stays unchanged. Formatting uses the document's tabs/spaces mode and width for subcircuit/control bodies and explicit continuations. Directives, control commands and continuation markers use spaces even in tabs mode: leading tabs prevent their recognition by runtime input preprocessing. Existing tab-prefixed boundaries leave the document unchanged; Convert indentation to spaces is a separate explicit edit. Adjacent devices with the same prefix and field count align their fields; other statements use single field separators. Expression and waveform interiors, quoted paths, names, number spellings, comments and blank rows stay literal. Statements and continuation markers retain their order; formatting does not insert markers or save files.

Repeated formatting makes no further edits. One Undo restores the complete source and prior cursor/selection state; Redo reapplies it. Formatting ends an active snippet. Unclosed/crossed scopes leave the document unchanged; unfinished or mismatched expressions and cross-line quoted text retain their rows. Documents above 10,000 rows or 1,048,576 characters, or with a row above 65,536 characters, are left unchanged. Individual rows with 256 or more fields retain their text; fields wider than 80 display cells use single separators.

6.11.1 Opt in to automatic formatting

Automatic formatting: Preferences > Editing > Automatic editor formatting offers independent Format document on save, Format pasted rows and Format completed rows on Enter settings. All are disabled by default. Paste formatting starts from literal clipboard content and formats only the inserted rows; Enter formats the completed row and the new row. Both share the originating edit's Undo step. Printable characters and committed input do not trigger formatting. Active snippet fields retain their linked editing and history.

Trim safe trailing whitespace on save removes only spaces/tabs outside comments, strings and expressions. Keep one final newline on save removes safe extra final blank rows and explicitly enables a final newline. Without this setting, the loaded final-newline presence is preserved. These independent cleanup settings are disabled by default. Unfinished or mismatched expression/string content and oversized source prevent cleanup. Save formatting and cleanup change the visible buffer before writing and share one Undo step. A failed write retains those visible, undoable changes and leaves the document unsaved.

6.12 Keep text Undo and cursor history distinct

Cursor/selection history: *Edit* > *Cursor Undo* (`Ctrl/Cmd+U`) restores previous caret and selection states without changing source or consuming text Undo. *Cursor Redo* (`Ctrl/Cmd+Shift+F6`) restores them forward. History includes backward, multiple and column selections and the preferred vertical column through short lines; a mouse drag is one step. Up to 256 completed changes are retained in the current document. Movement after Cursor Undo discards its redo branch; Copy and unchanged actions preserve it. Source/file identity changes, reload and text Undo/Redo clear cursor history. Text redo remains available across cursor-only changes.

Saving ends the current typing group. Undo back to saved content clears the dirty indicator; Undo before that content makes the buffer dirty, and Redo reconciles it again. Undo/Redo edits the buffer, leaving disk content unchanged. A failed save retains dirty, undoable edits. Enabled save formatting remains an ordinary source edit in text history.

Multiple-selection Copy/Cut joins nonempty fragments in source order; overlapping ranges are handled once. Empty cursors add no fragments. Paste broadcasts the complete clipboard text to each merged range, including multiline text; it does not map separate fragments to individual cursors. Each Cut/Paste is one Undo action and restores the earlier selections.

Navigation, Search and Refactoring

Choose source navigation for declarations, literal search for text, and scoped Rename for a symbol identity. Previews show exactly what will change. Source links reveal folded regions and preserve existing dirty buffers.

7.1 Open files and jump to a location

Quick Open File, `Ctrl/Cmd+P`, searches fuzzy file names and relative paths. Queries without a path separator match file names; queries with a separator match paths. Rows show containing paths; full paths are available in hints so duplicate include names remain distinguishable. Open and untitled buffers and existing recent netlists are included. With an empty query, open buffers rank first, then recent files in recency order, then discovered files. With a query, match quality ranks first. Recent files outside the workspace are included without scanning their folders; missing files and non-netlist recent entries are skipped. Opening an existing buffer preserves unsaved text and its view unless a location is requested.

Append `:line` or `:line:column`, for example `models/jj.inc:12:4`, to jump to a physical line and column. Both are 1-based; columns count UTF-16 text positions rather than tab-expanded display columns. Omitted column means 1.

Positions beyond source bounds clamp to the last line/column; positions inside a grapheme cluster move to its start. A Windows drive colon remains part of the path. A trailing colon introduces location syntax and requires positive integers. Use `:line[:column]` with an empty path to navigate the selected result. Invalid locations disable opening.

Location conversion uses the dirty buffer when open; closed sources are read with their encoding/BOM and a 2 MiB limit. Navigation reveals folds and does not edit or save source.

The picker discovers `.cir`, `.inc`, `.lib`, `.sp`, `.spice` and `.net` files under the current workspace without parsing or simulation. Hidden folders, `build`, `bin` and filesystem links are skipped. Up/Down selects, Enter opens, Escape cancels and F5 refreshes added or moved files and recent entries. Discovery is limited to 4,096 files, 1,024 directories and 8 MiB of results; at most 512 matches display. Reached limits and stale targets are labelled. Use `Ctrl/Cmd+Shift+O` for symbols in the current file or `Ctrl/Cmd+T` for workspace symbols.

7.2 Follow scopes and source links

Breadcrumbs above the editor show the workspace, current file and enclosing subcircuit or control block. Click the workspace or file to search its symbols; click a scope to follow its declaration. Use *Edit > Focus Breadcrumbs* or `Ctrl/Cmd+Shift+B` to focus the current scope or file. Left/Right and Tab move between buttons; Enter or Space activates, and Escape returns to the editor. Full paths appear in tooltips. Narrow views shorten captions. Scope links clear while analysis is checking or stale; failed and incomplete analysis is labelled. Untitled files retain a file-symbol action.

Choose *Edit > Outline* or press `Ctrl/Cmd+Shift+U` to toggle the current source's Outline panel. Subcircuits contain their ports, parameters, models and devices according to actual scope. Named and unnamed control blocks appear as runs; variables use their run context. Required enclosing scopes from includes remain visible. Choose source order or name sorting and filter names with fuzzy matching; matching declarations retain their ancestors.

Opening the panel focuses its filter. Arrow keys enter and explore the tree, `Enter` follows the exact declaration, and `Escape` returns to the editor. The current declaration or enclosing scope follows the caret, including continued statements. Filtered results keep the selected match.

Dirty buffers stay unsaved; changed sources clear the tree while analysis refreshes. Partial analysis and the 4,096-item limit are labelled. Panel visibility belongs to the current tab.

Choose *Edit > Circuit hierarchy* to open the Outline panel's *Instances* view. Its view selector also offers *Dependencies* and *Includes*, alongside *Declarations*. Instances start at the circuit root and retain distinct paths such as `/Xleft/Xchild` and `/Xright/Xchild`. Dependencies start at each effective subcircuit definition, including unused definitions. Includes follow physical include statements; shared files expand once, with shared and cyclic edges remaining visible. Missing, unresolved, ambiguous and incomplete entries are labelled rather than expanded as known circuit instances.

Source-order/name sorting and fuzzy filtering retain matching ancestors. *Open source* or `Enter` visits the call site, definition or include statement. *Open target* or `F12` visits a resolved subcircuit definition or included file. Navigation reveals folded targets and uses editor history; changed buffers or include files invalidate these actions. `Escape` returns to the editor. Paths and source locations appear in tooltips.

Each hierarchy view has at most 512 rows, 32 expansion levels and 1,024 bytes per instance path; a partial or limited view is labelled. This is source analysis, with unsaved include buffers, and does not run a simulation or evaluate instance parameters.

Use `F12`, `Ctrl/Cmd+click`, or `Follow include / definition` to navigate a source link. Included-file issues open the corresponding source tab; existing unsaved tabs are preserved.

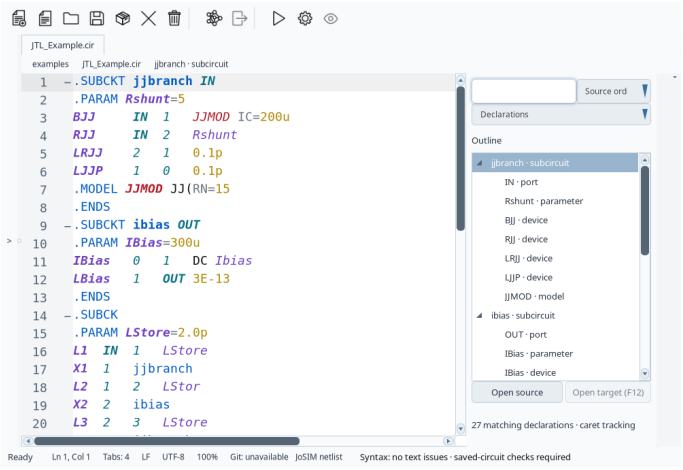


Figure 7.1: The Outline follows parsed subcircuit scopes. Its selector also exposes Instances, Dependencies and Includes.

7.3 Find declarations and references

Choose *Edit* > *Go to Symbol* or press `Ctrl/Cmd+Shift+O` to search declarations in the current source. Use *Tools* > *Go to Workspace Symbol* or `Ctrl/Cmd+T` for open buffers, unopened workspace decks and their includes. Filter by subcircuits, models, parameters, devices or named runs. Names use ASCII case-insensitive exact, prefix and ordered fuzzy matching; fuzzy queries need at least two characters. Rows show scope, source and declaration state. `Up/Down` selects, `Enter` opens the exact declaration range and `Escape` cancels.

Source jumps use navigation history and reveal folded lines. Dirty buffers stay unsaved. Changed sources disable results; use `F5` or Refresh, including after adding files.

Incomplete analysis and result limits are labelled. Symbol search is license-free and does not simulate.

Use **Edit > Peek definition / include** or `Alt+F12` to preview source inside the current editor without opening another tab. Peek reads current unsaved buffers and shows a read-only excerpt with the target selected. Ambiguous or context-dependent references require an explicit target choice. Choose **Open in editor** or press `F12` to follow that exact range; `Escape` closes Peek. Edits or changed includes disable stale actions; **Refresh / F5** checks the original editor caret again. Preview is bounded to nine lines, 4096 characters per line, 128 targets and 2 MiB per source.

Use **Edit > Find All References** or `Shift+F12` to list locations across the current root circuit and its includes, including unsaved buffers. Results are grouped by file and show statement context with *Definition*, *Use* or *Possible use* labels. Shadowed or replaced declarations retain separate identities.

Ambiguous targets require an explicit declaration choice; context-dependent uses remain labelled as possible. Select a location and press `Enter` or choose **Open in editor** to follow its exact range. `Escape` returns to the editor; `F5` refreshes from its caret. Changed sources disable stale locations. Incomplete analysis, 128-target and 2048-location limits are labelled.

Reference analysis does not save buffers or run a simulation.

Place the caret on a resolved parameter, model, device or node to highlight its occurrences in the current source. Translucent boxes retain syntax colours and outline the matching ranges. Matching follows declarations and actual scopes, including definitions in unsaved included buffers; identical names in unrelated scopes are not highlighted.

Ground aliases `0` and `GND` share one identity. Ambiguous, context-dependent, inactive, stale and incomplete-analysis results stay clear. Selecting text suppresses occurrence marks.

Highlighting does not edit text, alter find results or run simulation. The local occurrence index is limited to 16,384 physical ranges; a reached limit is labelled.

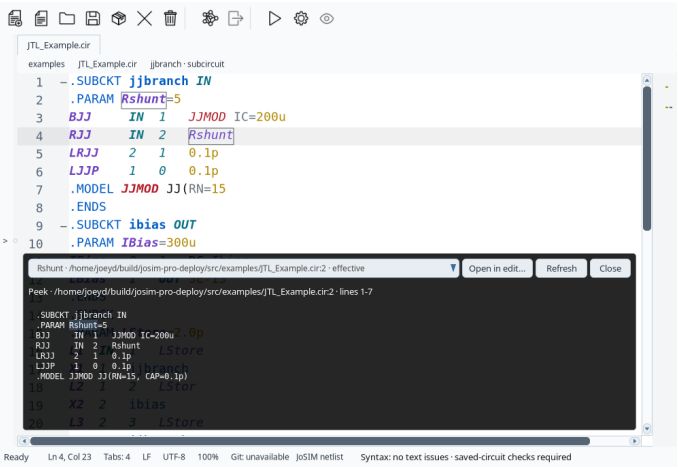


Figure 7.2: Peek keeps the reference and its declaration visible together; Open in editor follows the exact source location.

7.4 Return through navigation history

Use **Edit** > **Go Back / Go Forward** or `Alt+Left/Right` (`Cmd+Alt+Left/Right` on macOS) to return through source jumps, Go to Line and quick-open locations. Navigation restores the caret, selection and scroll position; folded target regions are revealed and positions clamp when a source has become shorter. Saved closed files reopen; unavailable locations are skipped. A new successful jump replaces the forward branch. This history is separate from browser folder history.

Previous Edit Location, `Ctrl/Cmd+Alt+Z`, visits earlier edits across open and saved source documents. Consecutive edits on the same line coalesce. Each source and edit history retains up to 100 locations during the session. Navigating does not undo, save or change source text.

7.5 Fold source safely

Completed subcircuit/control blocks, consecutive full-line comments and contiguous continued statements can collapse with the gutter marker or `Toggle code fold`. Use `Ctrl/Cmd+Shift+[` to toggle the current region and `Ctrl/Cmd+Shift+]` to unfold the current region. `Edit > Folding` offers fold/unfold current, recursive current, all and nesting levels 1–9. Level 1 is the outermost region; child folds retain their state when a parent unfolds.

Folding preserves source text and undo history. Unaffected folds follow inserted/deleted lines; touched regions expand while analysis checks their boundaries, then restore valid collapse choices. Removed or malformed boundaries remain visible.

Source navigation, search and selection reveal their targets. New/load resets folds; fold choices apply to the current session.

7.6 Search and replace in one document

Common shortcuts follow desktop-editor conventions. Use `Ctrl/Cmd+F` for find, `F3` or `Ctrl/Cmd+G` for the next match, `Shift+F3` or `Ctrl/Cmd+Shift+G` for the previous match, and `Esc` to clear search highlighting or close the search panel.

Choose `Whole document`, `Selection` or `Editable range` in the search scope row. Whole-document search includes folded source and reveals matching ranges. Selection scope captures the current primary, multiple or column selections; finding a match does not change these bounds.

Edit the From/To fields as one-based `line:column` positions, then choose `Apply scope`; the end position is exclusive. Overlapping selections are searched once. Replacement updates captured bounds as text grows or shrinks.

Other source edits, including Undo/Redo, pause scoped search until the range is applied or selections are recaptured. Invalid or empty bounds never fall back to whole-document replacement.

`Unfolded text only` excludes folded rows within the chosen scope. It includes unfolded rows outside the viewport. Multiline matches cannot cross hidden gaps. Folding or unfolding refreshes this option immediately. Each selected range or contiguous unfolded region is a separate regex subject, so `\A`/`\z` and clipped row anchors refer to that region; lookarounds cannot inspect excluded text. Whole-word checks still use the original surrounding source. Scoped and unfolded-only searches share the source, result, output and traversal limits below, including single-line literal searches.

The search panel offers `Match case`, `Whole word` and `Regex`. Regex uses Perl-compatible expressions within the chosen scope. Find and replacement fields accept pasted multiline text; `Ctrl/Cmd+Enter` inserts a line break, while `Enter` finds or replaces and `Shift+Enter` in Find selects the previous match. Fields expand to at most three visible rows and scroll for longer input. A multiline editor selection seeds Find. For continued statements, include their physical line breaks and `+` markers: search never flattens or rewrites circuit syntax. In Regex, `\n` matches a line break; `^` and `$` anchor physical rows.

Dot excludes line breaks unless `(?s)` enables dotall. `\A` and `\z` anchor the document. Search uses normalized line breaks and selects exact physical ranges, including Unicode and folded source; replacements use the document line endings. The storage-only final terminator is excluded, while an actual final empty row is searchable.

For example, find `R([0-9]+)` and replace with `R$1_new`. Replacements support `$0` for the complete match, `$1`–`$99` for captures, `${name}` for a named capture and `$$` for a literal dollar. Unmatched optional captures insert empty text.

Backslashes and other dollar signs remain literal in replacement text. Literal search mode does not expand captures. Replace and Replace All use current source and form one text undo operation.

Zero-length matches advance at Unicode character boundaries, including line-end matches. Regex matches and captures that split a Unicode character are rejected.

Enable `Preserve case` in the replacement controls to adapt each replacement to an uppercase, lowercase or initial-capital match. For example, replacing `gain` with `level` changes `GAIN`, `gain` and `Gain` to `LEVEL`, `level` and `Level`. Mixed-case and uncased matches keep the replacement as entered. Digits and punctuation do not determine case. Regex capture expansion happens before this case adjustment; a zero-length match keeps the template case. The option defaults to off and is saved across document and workspace search. Replacement history stores the text entered.

Whole-word search keeps prefixes and label punctuation together: `TRAN` does not match inside `.TRAN`, nor `MTL` inside `@MTL`. Dots, colons, hierarchy separators, bus brackets, signs and path separators remain part of ordinary netlist fields. Letters, digits, underscores and combining marks remain joined.

In parameter assignments, braces and single-quoted expressions, operators separate identifiers, so `gain` matches in `gain+1` but not `gain_extra`. Physical `+` continuations retain this lexical context. Boundary context is limited to 4,096 characters; longer fields use conservative field boundaries.

These are text-search boundaries, independent of semantic symbol identity.

Use the Find history and Replace history lists to recall executed searches and replacements without changing source. Each list keeps the 20 newest distinct entries across launches, preserving multiline text and case; entries longer than 16,384 characters are not stored. An empty replacement can be recalled for deletion.

Typing alone does not add history. Match case, Whole word, Regex, Preserve case and Unfolded text only are saved; document-specific selection and editable bounds are

not saved. The status shows the total count before navigation and the current match number afterward.

Crossing either end explicitly reports `Wrapped to first match` or `Wrapped to last match`; changing the query, scope or source clears this feedback. Search errors take priority over counts.

Pattern, replacement and execution errors appear in the search status and disable replacement. A reached limit clears partial results; Replace All never applies a truncated match list. Regex limits are 2,048 pattern characters, 99 captures, 16,384 characters per source line, 2 Mi characters of source, 10,000 matches and 8 Mi characters of expanded replacement text. Replacement templates are limited to 16,384 characters.

The engine permits 1,000 match steps per starting position and 128 recursive calls; a 100 ms budget is checked between engine calls and after each call. An individual engine call is bounded by those work limits rather than interrupted at exactly 100 ms. All literal searches and replacements use a 16,384-character pattern/template limit, 2 Mi characters of source, 10,000 matches, 8 Mi characters of replacement output and a 100 ms traversal budget.

Simplify the pattern or reduce the source when a limit is reported.

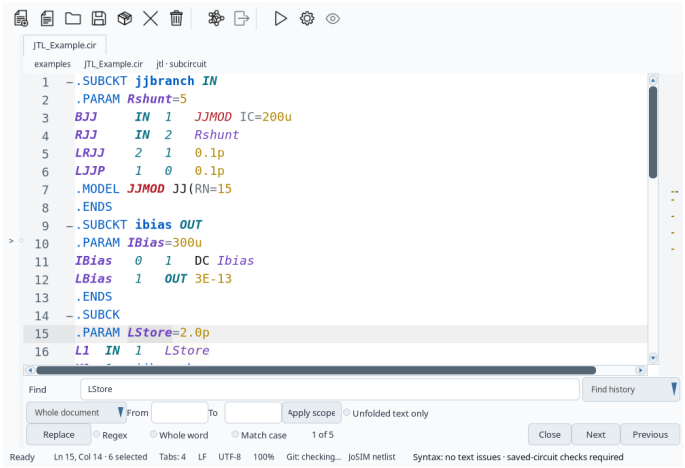


Figure 7.3: Document search exposes scope, regex and preserve-case options. Whole-document search includes folded source.

7.7 Search a workspace

Use *Tools* > *Search Workspace* or `Ctrl/Cmd+Shift+F` to search literal text in the workspace folder and open netlist buffers. Enter starts a search; `Ctrl/Cmd+Enter` inserts a line break. Match case and Whole word refine the query. Results show source paths, positions and text previews. Select a result and choose Open, or press Enter, to reveal its exact range, including folded source. Unsaved buffers override their disk files; untitled buffers are included. Search does not edit source or run simulation.

Include and Exclude accept comma-separated path globs: `*` matches within one path segment, `**` crosses folders, and `?` matches one character. Basename masks apply at any depth; paths use forward slashes. The default Include lists `*.cir,*.inc,*.lib,*.sp,*.spice,*.net`. Use `**/*` to include custom filenames.

Already-open sources are eligible with the default Include; custom Include masks restrict named buffers too. Generated build/bin, results, output, runs, simulation and plots folders are excluded by default. Hidden, linked and build/bin directories are always skipped during folder discovery. Search includes files beneath the workspace and open buffers; unopened include files outside that folder must be opened or searched from their containing workspace.

Glob matching follows platform path case rules. Masks are limited to 16 entries of 256 characters; braces and character classes are not supported.

The latest search-results view is retained for the current workspace session. Reopen *Tools* > *Search Workspace* or press `Ctrl/Cmd+Shift+F` to return to its query, filters, replacement text and selected result without scanning again. Closing the window or pressing Escape keeps completed results; closing during a search cancels unfinished work. Closing the application discards this view.

In the results list, Up/Down, Home/End and Page Up/Down move through every retained match. The view displays at most 512 rows per page and changes pages as selection crosses their boundaries. Enter or Open reveals the selected source range and returns to the editor. `Copy result` or `Ctrl/Cmd+C` copies the selected path, position and preview; `Copy all` or `Ctrl/Cmd+Shift+C` copies all retained results, including other pages. These shortcuts apply while the results list has focus. Clipboard output includes the original query and known stale/incomplete state.

`Remove result` or Delete removes only the selected entry from the view and from later replacement previews. `Clear results` removes the retained result list. These actions do not edit or delete source, save files or change editor undo history. Typing a new query or changing filters keeps prior results labelled as stale until Search explicitly replaces them.

Changed workspace/buffer snapshots remain visible as stale. Opening a result rechecks its complete source: unchanged sources can still be navigated after other tabs are opened; changed, moved or closed unnamed sources require another search. Named sources can reopen from unchanged disk content. Replacement still requires a current, complete search and the reviewed operation below. Removing entries never makes an incomplete search complete.

Cancel work stops between filesystem or matching operations. Search supersedes previous work. Limits are 4,096 files, 1,024 directories, 2 MiB per disk file, 2 Mi characters per buffer, 16 MiB total source, 10,000 matches, 8 MiB of results and a five-second matching budget after discovery. Skipped unreadable, binary or oversized files and reached limits show Incomplete; narrow the query or filters to inspect remaining matches. Blocking filesystem calls cannot be interrupted immediately.

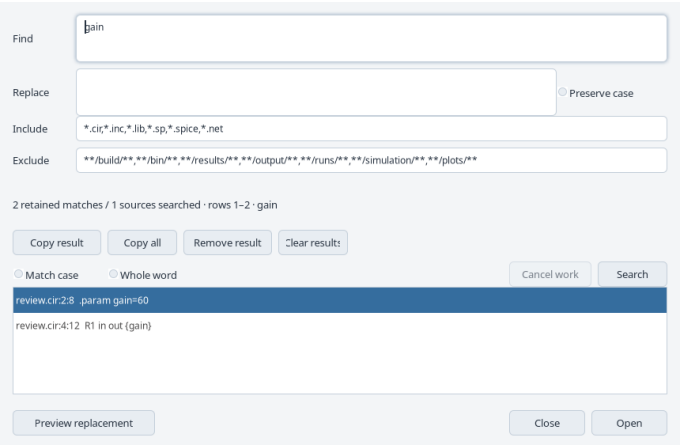


Figure 7.4: Workspace results retain paths, exact locations and source previews; unsaved buffers take precedence over disk.

7.8 Review workspace replacements

In Search Workspace, enter literal replacement text and choose `Preview replacement`. An empty replacement deletes matches. The optional `Preserve case` setting adjusts each match before its Before/After preview. The review window groups matches by file and shows complete read-only Before/After source. Select each file explicitly with its checkbox; no file is preselected. Read-only and unchanged files cannot be selected. Review the selected files, then choose `Apply selected files`; Cancel changes nothing. Replacement supports Unicode and physical line breaks, and does not expand regex captures.

Apply checks buffer revisions, expected text, disk hashes and writability again before changing any participant. Changed, missing or read-only sources reject the selected operation. Reviewed unopened sources become unsaved editor tabs; existing dirty buffers retain their contents and history.

Disk files are saved only by an explicit Save action. A successful operation uses one transaction: Undo or Redo from any participating buffer changes the entire group. Independent edits must be undone before stepping the group. Renamed or read-only participants block grouped history; no partial group Undo is performed. Closing a participating tab discards its buffer history and makes the group unavailable; reopening it does not restore that history.

Incomplete search results cannot drive replacement. Preview is limited to 256 files, 10,000 disjoint matches, 16,384 replacement characters, 2 Mi characters of output per file and 8 MiB of preview data. Narrow the search when a limit is reached. A no-op file creates no history entry. Search and replacement preserve statement order; this is literal editing and does not infer circuit intent.

7.9 Mark review locations

Use `Edit > Navigation > Bookmarks and source marks` or the command palette to mark circuit-review locations. `Ctrl/Cmd+F2` toggles a bookmark on the current physical line. `F4` and `Shift+F4` visit the next and previous resolved marks in source order, wrapping at the document ends. Navigation reveals folded targets and records the normal editor navigation history. The gutter diamond identifies marked physical lines; gutter clicks retain their existing selection, diagnostic and folding actions.

`Add or label source mark` attaches a label to the current caret. `Review source marks` lists labels, line/column locations and source previews. Filter the list, use Enter to open a resolved location, or use the actions to change its label, remove it or explicitly reattach it to the editor's current caret. Clear all asks for confirmation. Escape closes the review. Marks are shared by split views of the same document, including read-only views. They do not change netlist text, its dirty state or its text undo history, and are never passed to the simulation engine.

Edits before an anchor move it, including ordinary undo/redo of those edits. Replacing or deleting the anchored text makes its mark unresolved. Text undo does not automatically revive an unresolved mark; reattach it explicitly after checking the intended source. Unresolved marks remain listed and are skipped by next/previous navigation. Broad replacements conservatively invalidate anchors inside the replaced region. A review whose source or marks change rejects its pending action.

Named-file marks remain in the local GUI config store after closing a tab. Open-tab marks also participate in workspace-session restoration; untitled marks participate in unsaved-buffer recovery. Restored anchors require the same normalized source hash; changed or invalid locations remain unresolved.

Labels and locations are local metadata, not portable circuit annotations. Limits are 1,024 marks per document, 128 UTF-16 code units per label and persistent named-file records for 256 paths. Labels cannot contain control characters.

Clear marks from another file before adding a new path when the persistent limit is reached. Source edits and metadata remain usable when a config write fails; a metadata-save error appears in editor status. Text undo history is not persisted across reopen.

7.10 Rename a resolved symbol

Use **Edit** > **Rename Symbol** or `F2` on a resolved source symbol. Enter the new name and select **Preview** (`Enter` in the name field). The pane groups exact replacements by file. **Apply rename** changes all participating unsaved buffers together; `Escape` cancels. Rename checks declaration collisions and reference capture/shadowing across the current root and include graph. It preserves local subcircuit node and

port identity, caller port bindings, ground aliases, component device prefixes, qualified instance paths, parameter override keys and matching named `.ENDS` boundaries.

Other roots outside this graph are not analyzed. Ground, built-ins, ambiguous/context-dependent uses, incomplete analysis, stale revisions and read-only files prevent application. Names contain at most 128 ASCII letters, digits or underscores; only node names may begin with a digit. Previews are limited to 256 physical edits and never apply a partial result.

Rename does not save files. Grouped Undo/Redo requires every participating buffer to remain open, writable and at the same transaction; later independent edits can block grouped history.

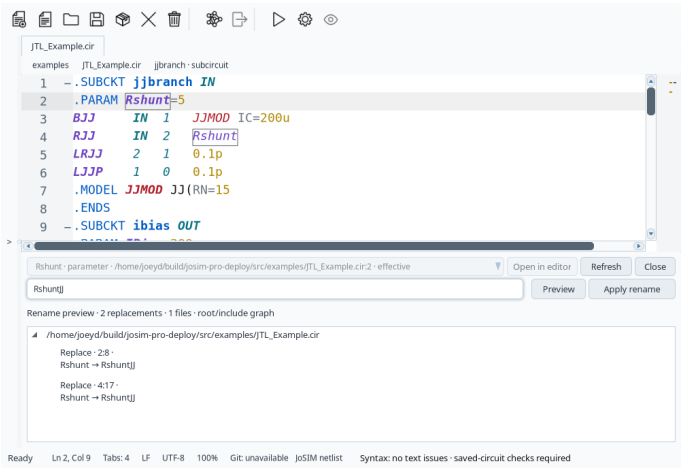


Figure 7.5: Rename previews replacements by file and checks scoped collisions before changing any buffer.

7.11 Edit paired subcircuit names

Place the caret in a named `.SUBCKT`/`.SUBC` declaration or its matching named `.ENDS`, then choose `Edit > Diagnostics and refactoring > Edit paired subcircuit names`. The same command is searchable in the command palette and can be assigned a keybinding. The selected name is editable; typing, deletion and paste mirror it into the paired range. Tab or Escape finishes. The whole session is one undo/redo transaction.

Names use the core Rename policy: 1–128 ASCII letters, digits or underscores, starting with a letter or underscore. Invalid names and collisions with active subcircuit declarations in the root/include graph are rejected before either range changes. Source revisions and read-only state are checked on entry and before each linked mutation. A peer view's text edit ends the session.

Both names must be matched, explicit and in the same source. Unnamed, mismatched, incomplete, ambiguous or cross-file pairs are rejected; continued statements and legacy name-last headers use core physical ranges. Instance calls remain unchanged: use `Rename Symbol (F2)` instead when references must follow the declaration. Normal snippet placeholders keep their existing behavior; this command adds validation to paired editing of existing blocks.

7.12 Preview a circuit refactoring

The `Edit > Diagnostics and refactoring` menu and command palette offer `Extract selected devices into subcircuit`, `Inline subcircuit instance` and `Fold constant parameter expression`. Each action opens a read-only before/after preview. Choose Preview, review the complete changed region, then Apply; Escape cancels. Apply changes only the current buffer as one undo/redo transaction. Save remains explicit, and dirty buffers are preserved.

Extraction requires complete root-level physical lines containing flat R/L/C statements with literal values. All non-ground nodes become ports in first-use order; no internal/external connectivity is guessed. Enter a unique subcircuit name; the instance label is X followed by that name. Comments, device values and statement order are retained. Inlining replaces one root instance with prefixed primitive labels and mapped ports/internal nodes; it keeps the definition and other instances. Body comments and literal spellings remain intact.

These initial transforms reject parameterised, nested, model-dependent, continued or cross-file forms, device-output/coupling references, spread policies and name collisions. Constant folding accepts only a nonliteral, dependency-free, statically known root parameter expression; named dependencies and instance defaults remain unchanged. Place the caret within the expression before invoking the command.

Complete analysis and core validation before and after are required. Previews are bounded to 128 KiB of root source and 128 devices; truncated review cannot be applied. Source/include revisions, disk identity and read-only state are checked again at Apply. No simulation or arbitrary external evaluation runs implicitly.

Diagnostics, Checks and Tasks

Live analysis is license-free and does not simulate. Use its exact source locations to repair authoring issues. Check Circuit is a separate, explicit licensed setup check; Run executes the simulation. A clean diagnostic list does not establish numerical correctness.

8.1 Read live diagnostics

Live static analysis runs shortly after editing pauses and refreshes included source periodically. Issues appear below the editor with line and column locations and are underlined in the source. Click an issue, or use `F8` and `Shift+F8`, to navigate between them.

In the focused problem list, `Enter` follows the selected source location and `Esc` returns to the editor. Checks cover unsupported directives/devices, missing required arguments, quoted strings and delimiters, duplicate component labels within a definition, subcircuit/control block structure, parameter expression grammar, supported model types and fields (including duplicate aliases), required MTL/WIDEBAND fields, paired WIDEBAND tables, and source waveform argument counts. Missing required option values are marked as incomplete; ignored option tokens are advisory warnings.

Unfinished statements show an `Incomplete` label with the expected input. Later declarations remain available for suggestions and navigation. Missing delimiters produce a primary issue rather than repeated follow-on errors.

Unresolved-symbol warnings are withheld while source syntax or include analysis is incomplete and resume after repair; known source links remain available.

Each live issue shows a stable rule code and a syntax, semantic or advisory category. Severity remains separate: unfinished syntax and unresolved semantics may be warnings. Use `Explain diagnostic` from the editor's edit menu or `F1` to open built-in Help for the hovered issue, the issue at the caret, or the selected problem. The first problem is used when none applies. In the focused problem list, `F1` explains its selected row. Help retains related declaration and include origins. Saved-circuit failures open a separate explanation; runtime provenance stays in the message, and unavailable exact source ranges are identified explicitly.

Errors use wavy underlines; warnings use dashes; information uses dots. Boxed `E`, `W` and `I` markers and written severity labels distinguish issues without relying on color. Hover over a marked range or gutter marker for its severity, rule and explanation.

Click a gutter marker to follow the highest-severity issue on that line. The overview strip at the right maps physical document lines in three severity lanes; hover explains and clicking reveals the source, including folded or offscreen lines. Continuation fragments are marked separately.

Included-file issues remain in the problem list until their source tab is opened. `F8` and `Shift+F8` reach all problems and related notes when markers overlap. Editing, leaving the marked surface, scrolling or resizing dismisses the hover explanation.

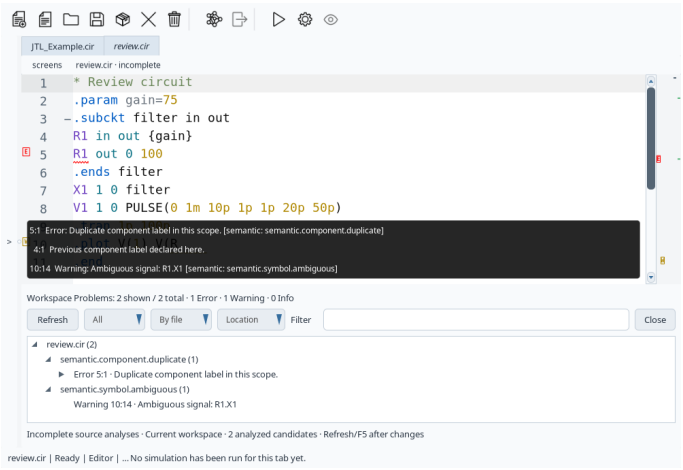


Figure 8.1: Workspace Problems groups diagnostics by source or rule. Related rows lead to previous declarations and include origins.

8.2 Check the whole workspace

Open `Tools > Workspace Problems` or press `Ctrl/Cmd+Shift+M` to inspect the working folder's `.cir` files, its subfolders and open netlist documents without opening every source tab. Open unsaved buffers override disk text, including included files. Hidden folders, `build`, `bin` and symbolic links are excluded from folder discovery; explicitly opened documents and reachable includes still participate. Included library files are analyzed in their root-deck context rather than as extra standalone roots. Identical shared-include issues appear once, with their root decks and include origins retained; context-dependent diagnostics remain separate.

Refreshing while a workspace scan is pending cancels that scan and keeps the latest requested snapshot. Workspace output is limited to 16 MiB; omitted roots are disclosed as incomplete. Per-root limits match the editor. Unopened roots larger than 2 MiB are skipped without opening tabs.

Choose `By file` or `By rule`, filter by severity or text, and sort by `Location`, `Severity` or `Rule`. Text searches match source paths, rule codes, messages and root decks. Counts distinguish shown issues from the total and list total Error, Warning and Info counts; related locations do not add to issue counts.

Expand an issue for related declarations and include origins. Double-click a location or press `Enter` to open only

that source, preserving existing unsaved tabs. F1 explains the selected issue, F5 refreshes the workspace snapshot and Esc closes the panel. Use Refresh after edits, disk changes or changing the working folder.

Results are bound to the document and include revisions inspected. Checking clears previous issues; changed source or folder revisions produce a Stale state and disable old source actions. Incomplete results identify skipped or unresolved source; Failed identifies an analysis failure.

Open-buffer edits invalidate dependent editor results immediately. Disk source changes are checked periodically and again before source actions; byte content is checked even when size and timestamps are unchanged. Use Refresh/F5 to analyze a stale workspace again; live editor analysis schedules a new check.

Workspace analysis does not start automatically after invalidation. Discovery is bounded to 128 candidate files and 1,024 directories; analysis is bounded to 128 roots and 16 MiB of root text, with a 2 MiB limit for an unopened root. Limits, skipped reads and incomplete source analyses are identified in the status.

Each root retains the include and expression limits described below. Workspace inspection does not run simulations or saved-circuit checks.

8.3 Understand checking, stale and incomplete states

Live analysis uses a 300 ms editing pause and cooperatively cancels superseded work. Completion, source navigation, diagnostic navigation and folding schedule missing analysis without waiting in the key handler. A pending action runs after the current result arrives if the document and caret still match; editing or moving the caret discards that action. Syntax status distinguishes checking, incomplete and failed analysis.

Hover over it to see the fixed budgets: 2 MiB of source, 64 files, 32 include levels, 16,384 physical lines, 8,192 statements, 64 KiB per statement, 512 diagnostics and 8 MiB of output. Oversized sources retain checks on a physical source prefix. A limit warning cannot be suppressed; incomplete results do not establish circuit validity. If symbol output exceeds the budget, symbol tables and source links are omitted together.

Automatic fixes are unavailable when an analysis or output limit is reached. Reduce the deck for complete semantic actions.

Diagnostic navigation selects the reported source range, including complete Unicode characters and continued physical lines, and reveals folded source. Source jumps show the text editor when the destination tab displays a diagram or waveform. Related declarations and include origins appear inline in the Problems views; select their rows to visit those ranges.

Use Alt+Left and Alt+Right (Cmd+Alt+Left/Right on macOS) to go back and forward through source jumps. History preserves caret, selection and scroll position, keeps unsaved tabs intact, and clamps saved positions after edits. Closed unnamed documents are skipped; named files can reopen.

Each workspace retains up to 100 locations for the current session. Starting a new jump discards the forward branch. History records views rather than analysis results; new diagnostic jumps still require current source analysis.

8.4 Preview and apply a quick fix

1. Follow an issue with F8, or select its row in Problems.
2. Press Ctrl/Cmd+. or choose Quick fix....
3. Review the exact replacement, then accept it or cancel.
4. Recheck the diagnostics and save the changed source explicitly.

Use Quick fix... in the editor's edit menu or Ctrl/Cmd+. for the hovered issue, issue at the caret or selected editor problem. Available actions cover a unique supported directive spelling correction, matching the known open subcircuit name, or one missing inner closing delimiter before an existing outer closer. Every action previews its exact replacement and asks before applying it.

Cancel leaves the source unchanged. Changed analysis or source cancels the action; check diagnostics again. Included source opens as an unsaved editor buffer when needed.

Each accepted fix is one undoable edit; it does not save or run simulation. Ambiguous spelling, unfinished quotes, missing end-of-line delimiters, model/node names and circuit values require manual editing.

Use Review available fixes across files... to preview every available unambiguous fix in the root deck's current diagnostics, including its source files. Accepting applies the complete set to editor buffers together. Dirty buffers retain their other edits; closed includes open as buffers. No file is saved automatically. Before applying, the editor checks root/include revisions, exact source bytes, expected text and file writability. A changed, missing, read-only or locked participant, overlapping edits or an invalid range rejects the entire set. A failure during commit restores text, selections and existing undo/redo histories. Sets are limited to 256 edits and 2 MiB per source file.

In hosted editors, single and multi-file fixes create a linked undo entry. Use ordinary editor undo/redo from any participating buffer to undo/redo the complete transaction. All participants must remain open under the same document identity and path, writable, unchanged on disk and at the corresponding undo/redo boundary.

Undo later local edits first. Closing/reopening, reloading or renaming a participant, or branching its redo history, prevents partial group undo/redo and reports the conflict. Groups last only for the current workspace session; recovery copies do not retain them.

Explicit saves remain independent: grouped undo changes buffers only, even after a participant was saved, and dirty markers reflect each file's saved baseline. Save each changed file through the usual save/conflict checks.

8.5 Understand static checks

Static analysis reads include files and uses unsaved buffers from open tabs. Relative includes require a saved document path. Missing or cyclic includes make analysis incomplete; unresolved-symbol warnings are suppressed until the source graph is complete.

Analysis is limited to 64 files, 32 include levels and 2 MiB of source. Conflicting component/control labels, duplicate model fields or aliases, mismatched subcircuit names and invalid parameter/model replacements include related source locations where available. These notes appear beneath the issue and support the same click, `Enter` and `F8`/`Shift+F8` navigation, including files read through includes.

Related notes do not count as additional issues. ASCII names are case-insensitive; non-ASCII letters retain their case. Parameters and models resolve locally before globally, with the last declaration in a scope taking precedence; subcircuit targets are global. Parameter/model replacement and local shadowing are permitted.

Repeated subcircuit names follow the runtime's legacy header-name selection and ignored-body rules; they are not rejected merely for repeating a name. Nodes and devices remain within their own circuit scope. Named control blocks have distinct run labels.

Suggestions hide shadowed globals and unrelated nodes/devices. Source navigation covers parameters, models, subcircuits, ports, implicit nodes, mutual-inductance devices and hierarchical output targets using either dots or vertical bars. Ground aliases `0` and `GND` share a built-in identity.

Ambiguous device references show an advisory and have no navigation target. Instance overrides retain their own identity; source links point to declarations rather than evaluated instance values.

8.5.1 Subcircuit calls and device shapes

Subcircuit calls check formal port counts, override assignments, caller-scope parameter references, missing targets and recursive instantiation. Too few ports are errors; extra port tokens and bare tokens after overrides are warnings because the runtime ignores them. Overrides may introduce new names; repeated overrides use the last value. References supplied by instance overrides are marked context-dependent instead of guessed from one instance. Reachable recursion is an error; recursion in unused definitions is advisory. Recursive diagnostics are bounded to 32 cycles and 32 call-chain locations per cycle; omitted cycles are reported explicitly and leave analysis incomplete. These checks use a finite graph of source definitions and call sites, without expanding the circuit. Source waveform arguments accept the core parenthesized and flat forms, such as `DC 1m`. Device checks cover supported node and argument counts, junction models and inline TJM coefficient counts, inductor references in K couplings, transmission-line field combinations, and `@MTL` port and initial-condition dimensions. Model compatibility follows the effective declaration in the current scope; related model/component locations are navigable. Statically known MTL conductor counts, including resolved global parameters, are checked against model matrices and ports. Unknown or instance-dependent dimensions and external TJM coefficient files remain deferred to the saved-circuit check.

8.5.2 Parameter dependencies and known values

Parameter analysis uses the core numeric expression grammar and supported math functions. Suggestions display proven constant values or subcircuit defaults; unknown, incomplete, instance-dependent and division-policy-dependent values are labeled without guessing.

8.5.3 Forward references

Forward references and effective replacements participate in dependency analysis. Cycles include navigable declaration locations; known invalid math domains and non-finite results are diagnosed. Only a conditional's condition and selected branch contribute to value dependencies. Missing includes prevent value evaluation. Analysis is bounded to 4,096 effective parameter declarations, 64 KiB per expression and 8 MiB of cumulative expression-evaluation work; expression nesting is limited to 256. A reached limit leaves analysis incomplete. Brace-wrapped `.PARAM` declarations are rejected by runtime grammar and cannot supply a constant value. Subcircuit values are defaults rather than expanded instance values; overrides may replace them. Numeric constant inspection uses the core default zero tolerance and does not apply configurable division fallbacks. External files, arbitrary function calls, random input and simulation are never evaluated by this check.

8.5.4 Numbers, suffixes and waveform domains

Numeric suffix explanations use the actual parser: `M` means milli, while `MEG` and `X` mean mega. Trailing letters do not declare physical units; unrecognized suffixes retain the parser's no-scaling behavior. Known pulse timing, colored-noise band limits, model scalar domains, WIDEBAND pole order/loss values and transmission-line scalar constraints are checked. Accepted runtime clamps and decreasing PWL/PWS time pairs are warnings; equal-time discontinuities remain allowed. Missing waveform defaults need simulation context and are not guessed. Model scalar expressions retain the runtime grammar: brace wrappers are rejected. Matrix dimensions are checked without running ma-

trix passivity solvers. A clean syntax status is not simulation qualification.

8.5.5 Control blocks and output targets

Control and output checks cover supported JoSIM-Pro commands, independent-run placement, initial-condition assignments, literal IV step counts, and transient or option constraints when values are known. Run options inherit global settings; later assignments retain runtime replacement rules. Output targets resolve against the main circuit and its instance hierarchy.

Voltage and phase may select nodes or supported devices; `P(...)` means phase. Power and energy use `W(...)` and `E(...)`. Multiconductor-line currents require `I(line[n])`; ambiguous, out-of-range and unsupported aggregate selectors are warnings with related declarations.

Bare legacy targets containing vertical bars may be interpreted as expressions by the runtime; use signal calls such as `P(B1|X1)`. Lexical output-path conflicts are advisory; runtime admission still checks effective paths and filesystem aliases. Unknown settings remain deferred.

The shared catalog includes `.OP`, `.DC` and `.AC`, plus transient measurements and parameter sweeps. AC runs provide frequency-domain magnitude/angle plots and typed complex CSV/gzip export. Unsupported expressions and model combinations produce source-linked runtime diagnostics.

8.5.6 Circuit advisories

Circuit advisories require a closed `.END` deck with complete includes and clean resolved semantics. They identify effective parameters or models with no active source reference, subcircuits unreachable from the main circuit, formal ports with no internal terminal, and non-port nodes with a single visible device terminal. Definition-body references count conservatively; instance bindings count as possible connections.

Output and initial-condition references add no terminals. These warnings may describe intentional circuits. They use core node/ground rules and source terminal incidence; they do not expand instances, infer conductive paths, prove matrix singularity, qualify convergence or establish numerical correctness.

Unfinished or uncertain decks defer these checks.

8.6 Run an explicit saved-circuit check

Use `Tools > Check Circuit (saved input)` after saving the active circuit and its included files. A valid JoSIM-Pro license is required. This explicit action reads disk includes and builds the default circuit/matrix setup through the simulation core. Workspace Problems shows the saved-circuit result separately from live analysis, with passed, failed, license-unavailable or not-checked status.

Dirty referenced buffers and externally changed open sources must be saved or reloaded first; other dirty tabs are preserved. Source verification uses the static inspector's file, depth and size limits; a reached limit prevents setup checking. Results track the inspected saved root/includes and open document revisions.

Editing or disk changes make results stale; rerun Check Circuit to check setup again. `Refresh/F5` switches to license-free live analysis and never implicitly runs a saved-circuit check. Failures retain core source files, line numbers and include stacks in the message and `F1` Help where available. Runtime ranges are labeled unavailable; no source location is guessed.

External setup resources such as TJM coefficient files are not part of the netlist revision snapshot; rerun Check Circuit after changing them. The check does not run simulations, write simulation outputs or qualify independent analysis jobs, convergence or numerical accuracy.

8.7 Inspect and cancel Tasks and Output

Use `Tools > Tasks and Output` or the command palette to inspect built-in Check Circuit and simulation tasks. The panel retains up to 64 recent task records during the workspace session; active tasks are retained. Select a task for its status, elapsed time and read-only output. Output is limited to 64 Ki characters per task and truncation is labeled. Simulation progress and cancellation use the existing Run owner. Closing the panel does not stop a simulation; closing a source tab retains task records already collected by the panel. Task history is not persisted across application restarts.

Check Circuit runs in background after explicit invocation. It requires a valid license and saved, current root/include files, and does not simulate. `Cancel task` cancels source inspection and requests cancellation between setup stages; an already executing input or matrix call must finish before cancellation completes. Cancelled, superseded or closed-source checks do not publish current Problems. `Latest check Problems` opens the latest published saved-circuit diagnostics without starting another check. `Workspace Problems` and its Refresh action still request live analysis.

`Open source`, or Enter in the task list, opens the task's root netlist and preserves an existing dirty buffer. `Open failure source` follows the first explicit `Source: file:line` location reported by the engine. Stale output opens the file without applying its old line offset. Check links revalidate saved source bytes; dirty buffers and simulation includes without a captured source receipt open by file only. Truncated output does not offer an exact failure offset. Exact runtime ranges are not guessed. Available include paths, line numbers and include stacks remain in failure output and Problems. Source edits conservatively label collected tasks stale; the latest check's saved root/include hashes are rechecked in background while the panel is visible. Stale tasks remain reviewable as historical output. Escape closes the

panel. These actions use only built-in licensed operations; the panel does not execute arbitrary shell tasks or attach a debugger.

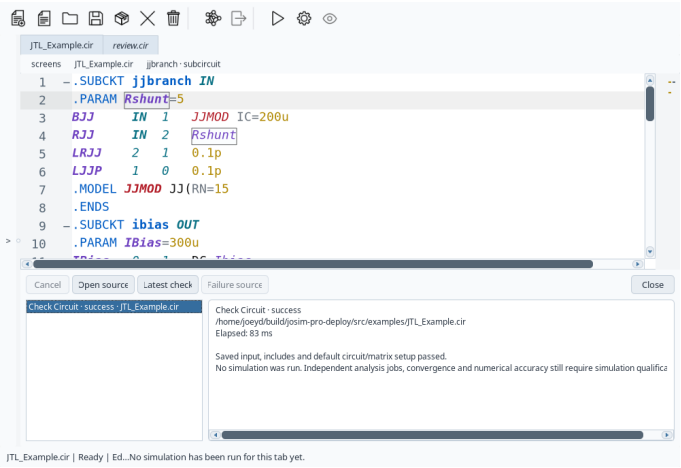


Figure 8.2: Tasks retains Check Circuit and simulation output with explicit status and source links. Closing the panel does not cancel a run.

Editor Preferences and Display

Start with Tools > Preferences for user defaults. Use Tools > Settings scopes when a project or netlist needs an override. Display choices change how source is shown; explicit format, conversion and save actions change the buffer or file.

9.1 Choose defaults and setting scopes

Preferences tabs: Tools > Preferences groups settings into General, Display, Editing, Assistance and Diagnostics. Display has Font, Layout, Guides and Scrolling pages. Each page states the shipped defaults; changes remain staged until OK, and Cancel keeps the current settings. Each page scrolls independently, while the tabs and OK/Cancel buttons stay visible. Display > Layout includes *Show folding controls in the gutter* (default on). Turning it off hides only the clickable markers; existing folds and keyboard/menu folding remain available. Editor settings apply across open views and future editors; per-view zoom and persisted document state retain their established behavior.

Setting scopes: Open Tools > Settings scopes to inspect the current netlist's effective values and their sources. Precedence is user Preferences, user netlist overrides, project editor settings, then project netlist overrides. The dialog shows staged winners, supports filtering and lets you Set override or Remove override for a setting. Removing an override inherits the next lower scope. Save commits the selected scope; Cancel discards staged changes. Save or Cancel before switching scopes after making edits. User Preferences remain in the existing Preferences dialog and config file.

Project overrides use `.josim-pro/settings.json`. The nearest containing settings file wins; lookup stops at a Git root or after 32 parent directories. If no file or Git root is found, the source folder is the project folder. Untitled editors use the configured working directory. Only the selected project's file is read; parent projects are not merged. The following example sets project defaults and then a netlist-specific override:

```
{
  "editor": {"EditorTabWidth": 4},
  "netlist": {"EditorTabWidth": 8}
}
```

Only supported editor, suggestion and diagnostic settings are accepted; project files cannot change global paths, licensing, autosave or keybindings. Values must have the displayed setting's correct type and range. Objects such as `DiagnosticRules` replace the lower scope's whole value. Project files are strict UTF-8 JSON, bounded to 64 KiB and 32 nesting levels. Unknown or repeated keys, invalid values and lossy decoding invalidate the whole project file; user settings remain effective. The dialog identifies invalid input, and the editor language status shows *settings!* with details in its hint. Repair invalid project files in their source before saving overrides.

Project saves check the original file hash and use the existing safe-save contract, including read-only/link refusal. User netlist saves check the current language layer and preserve other preferences. Reopen the dialog after a changed file rejection.

Opening a dialog or previewing settings never creates a project file. Effective changes apply to open views and on source open, Save As or editor focus; unchanged settings and unrelated appearance changes preserve per-view indentation/wrapping choices. Local view/session choices remain above configured defaults until that setting changes.

Live and workspace diagnostics carry the resolved root settings and reject results when those settings become stale.

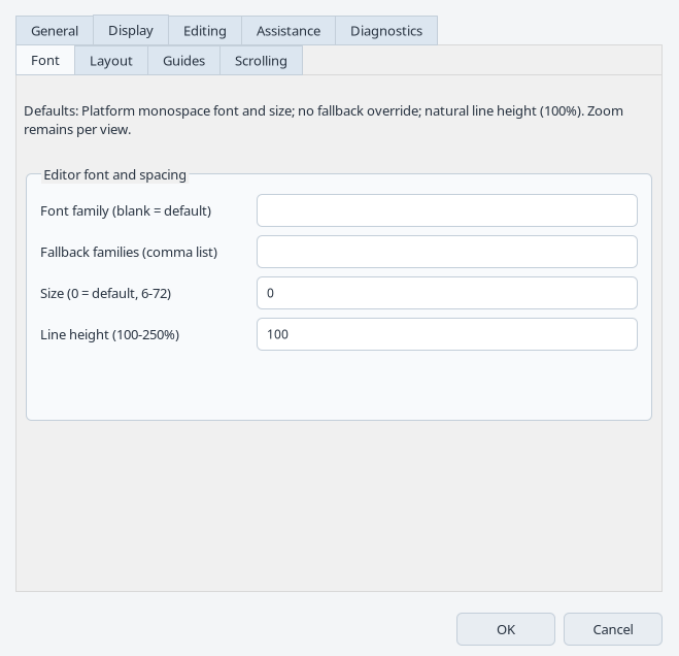


Figure 9.1: Preferences groups defaults into General, Display, Editing, Assistance and Diagnostics. Changes apply only after OK.

9.2 Customize keyboard shortcuts

`Tools > Keyboard shortcuts` lists commands and their platform defaults. Search by command name or category, select a command, enter a shortcut, and choose **Set binding**. Leave the binding blank to disable it. **Save** applies staged changes; **Cancel** keeps the existing settings. **Reset selected** restores a command's defaults, and **Reset all** removes custom bindings. Conflicting shortcuts and chord prefixes are rejected; disable the conflicting command first.

Use `Ctrl/Cmd` or `Primary` for the platform modifier, or explicit `Ctrl`, `Cmd`, `Alt` and `Shift`. Two-step chords require a modifier on both steps, for example `Ctrl+Shift+9 Ctrl+Shift+8`. Pending chords expire after 1.5 seconds and cancel on Escape, focus loss or input composition.

Unmatched second keys continue through normal input handling. Custom bindings apply in the source editor and toolbar; find fields, completion choices, snippets and IME composition retain their contextual keys. Unmodified typing, selection keys, AltGr printable combinations, F1 help and F8 diagnostic navigation remain reserved.

Disabled actions remain unavailable. The command palette displays effective custom shortcuts.

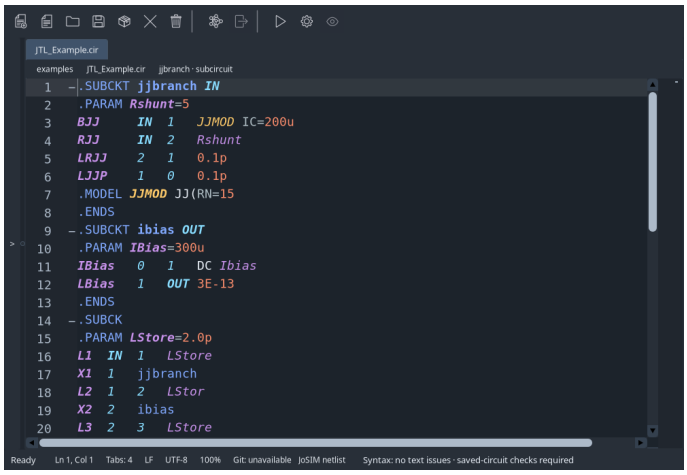


Figure 9.2: The same source in the dark theme. Display preferences affect presentation; source positions, file contents and Undo remain unchanged.

9.3 Choose font, spacing and zoom

Open `Tools > Preferences > Display > Font > Editor font and spacing` to choose the font family, comma-separated fallback families, size and line height. A blank family uses the platform editor default; a size of 0 uses the theme's default size. Explicit sizes range from 6–72 logical units. Line height is 100–250% of the natural font height, including its normal padding.

The first installed family in the requested order is used, followed by the platform editor default. Unicode glyph substitution is provided by the native text renderer. Monospaced fonts are recommended: the editor uses a fixed text grid for rendering, tabs, Unicode offsets, selections and mouse positioning. Settings are saved and apply to existing and newly opened tabs without rewriting source.

Use `Edit > Editor view > Editor zoom`, `Ctrl/Cmd+Plus`, `Ctrl/Cmd+Minus` or `Ctrl/Cmd+0` to enlarge, reduce or reset the active editor. Zoom changes in 10-percentage-point steps within 50–300%; reset returns to 100% of the configured base size. The status-bar percentage opens zoom in, zoom out and reset controls when activated, and its hint shows the selected font, effective size and line height. Zoom is kept separately per open tab and is not saved; new tabs start at 100%. Changing font, spacing or zoom preserves source text, selection and text Undo. Soft wrapping and display-scale changes use the same refreshed metrics as painting, navigation and hit testing.

9.4 Wrap and scroll long source lines

Open `Tools > Preferences > Display > Layout > Editor soft wrapping` to wrap long source lines. Wrapping is off by default. Set `Wrap column` to 0 for the viewport width, or 1–1000 for a maximum column; narrow viewports wrap earlier. At least two columns remain available. Set `Continuation indent` to 0–32 display columns; narrow rows reduce the indent to leave room for text. Wrapping prefers spaces or tabs and splits longer fields only at character boundaries. Tabs keep the physical line's tab stops. These settings apply to open editors and persist across launches.

Wrapped rows share one physical source line. Only its first row shows the line number, fold control and diagnostic gutter marker. Up/Down and Page Up/Down move through displayed rows, including with Shift selections; Home/End retain physical-line navigation. Mouse clicks, search ranges, completion and input anchors follow the displayed text. Column editing retains physical columns and applies one edit per source line. Resizing, folding or changing wrapping never inserts newlines, changes source order, creates an Undo entry or marks the deck modified. Saved netlists and copied selections contain only source text.

Open `Tools > Preferences > Display > Scrolling > Editor scrolling` to enable smooth scrolling, choose 1–20 rows per wheel notch (default 3), and reserve 0–100 blank rows below the last line (default 10). Wheel distance follows the editor's text size rather than viewport height. `Shift+Wheel` scrolls horizontally; native horizontal wheel events also keep the vertical position unchanged. Fine wheel deltas accumulate without discarding small movements. Navigation and edits cancel pending wheel movement and reveal the caret. Blank space, scrolling and zoom do not add source lines or text Undo actions.

9.5 Configure line numbers, rulers and guides

Open `Tools > Preferences > Display > Guides > Editor gutter and rulers` to choose `Absolute`, `Relative`, `Hybrid` or `Off` line numbers. Absolute uses physical source lines; Relative shows distance from the caret line (0 at the caret), while Hybrid shows the current absolute line number and relative distances elsewhere. Wrapped rows do not receive extra numbers. Gutter padding adds 0–24 logical units on each side of the number lane. Diagnostic, line-selection and folding lanes remain separate when numbers are hidden. Click/drag in the number lane selects whole physical lines; gutter double-clicks do not become word selections.

Column rulers accept up to 16 unique comma-separated columns from 1–1000, for example `80, 120`; blank disables

them. They mark display-cell boundaries after tab expansion and stay aligned with editor zoom. Wrapped rows use columns from the visual text margin. Active-line highlighting can cover `All`, `Text`, `Gutter` or `Off`; it follows the caret's physical line, including its wrapped rows. These preferences apply to open and new tabs without changing source or text Undo.

Open `Tools > Preferences > Display > Guides > Editor indentation and brackets` to enable `Show indentation guides`, `Show bracket-pair guides` or `Color bracket pairs by nesting depth`. These three options default to off and persist across launches. Indentation guides follow physical leading spaces and tab stops. Artificial wrap indentation receives no guide. The active indentation region extends through adjacent rows at the same or deeper indentation; it describes layout, not subcircuit identity.

Bracket colors repeat by nesting depth for matched parentheses, square brackets and braces outside comments and quoted strings. Guides connect pairs across physical continuations and visual wrapped rows. Independent statements never share a pair; malformed or unfinished pairs receive no connector. Hidden folded endpoints suppress their connector. `Emphasize active guides` defaults to on: the innermost enclosing pair and active indentation guide use thicker lines, and active bracket endpoints are underlined. Existing matching-bracket navigation still reveals folded targets. These settings preserve source text, caret and selection geometry, dirty state and Undo/Redo.

Guides cover up to 100,000 physical rows, 1,048,576 source characters, 256 nesting levels, 32,768 pairs and 262,144 pair/row crossings. Beyond a limit, completed pairs in the scanned prefix remain visible; later guides may be absent.

9.6 Reveal whitespace and control characters

Open `Tools > Preferences > Display > Guides > Editor whitespace and controls` to choose `Whitespace visibility`. `None` is the default. `All` shows every ASCII space and tab. `Boundary` shows tabs, leading/trailing spaces and runs of adjacent spaces, while hiding single interior spaces. `Selection` shows whitespace within the current primary and additional selections; `Trailing` shows only the final spaces/tabs of each physical source line. Spaces use dots and tabs use arrows across their configured tab stops. Trailing spaces use hollow circles; trailing tabs add a hollow starting marker. Soft-wrap indentation has no source characters and receives no markers.

Enable `Show control characters` independently to display boxed two-digit hexadecimal labels for C0 controls (`00–1F`, excluding tab and line breaks), DEL (`7F`) and C1 controls (`80–9F`). For example, `01` means U+0001. Hidden control characters occupy their source cell without a printed glyph. Printable Unicode keeps its normal rendering. Markers also reveal whitespace in comments and quoted text; they do not alter netlist syntax, selection offsets, copied/saved text, dirty state or Undo. Settings persist across launches and apply to open editors. `Cancel` leaves the previous settings intact.

9.7 Use the overview ruler and minimap

The overview ruler beside the editor has separate lanes for errors, warnings, information, search or semantic matches, selections and unsaved changes. Diagnostic shapes and selection outlines supplement colour. Click a marker to reveal its physical source line, including folded source; diagnostic and search markers select their exact ranges. Unsaved changes are compared with the last load or save. Deleted lines have a boundary marker. Very large changed regions use a bounded comparison; their hover hint explains that unchanged lines within the region can also be marked.

Choose `Tools > Preferences > Display > Layout > Editor minimap` for `Auto`, `Always` or `Off`. Auto shows the source preview only when the total scrollable document height exceeds one full wheel revolution: 24 notches at the editor's current scrollbar step. It stays visible near the document end. The shaded band shows the viewport; click or drag to navigate physical source lines. Wrapping, resizing and zoom update visibility without inserting source newlines. The overview ruler remains available when the minimap is off.

9.8 Keep scope visible while scrolling

The sticky scope header follows the top of the editor viewport as a subcircuit or control declaration scrolls out of view. It uses the parser's validated block ranges and shows up to four innermost headers, limited to one quarter of the viewport. Continued declarations must scroll fully out of view before their header appears. Malformed or unclosed blocks without a safe range do not produce a header; stale analysis clears the header. Known safe ranges from incomplete analysis have an explanatory hint.

Click a header to open its declaration. Press `Ctrl/Cmd+Shift+H` to focus it; `Enter`/`Space` opens the declaration. `Up`/`Down` moves among headers; `Escape` returns to the editor. The mouse wheel also scrolls when over the header. It is a separate navigation control, not another editable source line: source offsets, selections and text Undo remain unchanged. Caret navigation accounts for the overlay height. Choose `Tools > Preferences > Display > Layout > Sticky editor scope` to turn it off or on (default on) for existing and new tabs.

9.9 Read syntax and semantic colours

Semantic highlighting refines the lexical colours for resolved parameter, model, device, node and scope names. Declarations and formal ports are bold; symbols declared

within a subcircuit are italic. Parameter references use the component colour and model names use the string colour. Exact physical ranges also cover continuations and parameter expressions. Comments, quoted file paths and unclassified text keep their lexical colours. Ambiguous or superseded symbols are not assigned an effective semantic style. Incomplete, stale or limited analysis retains lexical colouring until a current complete result is available. Highlighting does not change source text or selection.

9.10 Adjust diagnostic severity and suppression

Open `Tools > Preferences > Diagnostics > Editor diagnostics` to set display severity or suppression for a rule. Enter the exact rule code shown beside a problem, choose `error`, `warning`, `info` or `off`, then press `Set rule`. Select a configured rule to edit it.

Choose `default` and press `Set rule` to remove its override. Press `OK` to apply and persist the changes; `Cancel` leaves the current settings intact. Settings apply to live authoring across open tabs and unopened workspace decks.

Workspace Problems requires `Refresh/F5` after a change; editors schedule a fresh check. Unknown codes have no effect until a matching rule is emitted. Up to 256 rule overrides are supported.

Suppression hides the issue's row, markers, hover and quick-fix actions; status text still shows incomplete analysis and the number of suppressed issues. Workspace counts identify suppressed root issues, which can include the same source in several roots. Severity overrides retain the issue's exact range and core default severity; diagnostic Help explains the override.

Incomplete input, include failures, analysis-limit notices and output targets that can conflict with input or other outputs keep their core severity and cannot be suppressed or demoted. Saved-circuit checks, save/conflict prompts and recovery/data-loss warnings are independent of these settings. Severity and suppression do not change parsing, symbol resolution, core completeness or simulation behavior.

No inline suppression directive is recognized; ordinary parser comment rules are unchanged.

Suspicious-character warnings appear in the editor diagnostics and workspace Problems with the exact physical location, Unicode code point (`U+XXXX`) and character name. Bidirectional controls and unusual line separators are reported even in comments or paths. Invisible controls, non-ASCII spacing and ASCII lookalikes in executable text are advisory: source is never rewritten. Ordinary Unicode comments and path names remain accepted; emoji joining and variation characters in those contexts are not treated as malformed input. Confusable checks compare individual characters with ASCII netlist spellings, not whole Unicode identifiers.

Diagnostic rule settings can change or suppress `advisory.unicode.bidi`, `advisory.unicode.invisible`, `advisory.unicode.confusable` and `advisory.unicode.line-terminator`. The first 128 character warnings are listed across the source graph; a visible, unsuppressible limit warning identifies omitted results. Existing analysis budgets also apply. No automatic character replacement is offered.

9.11 Popup behavior in small windows

Editor completion, hover, signature, diagnostic and peek surfaces use a rounded, semi-translucent background with opaque text, following the file browser's hover presentation. Popups fit the source viewport and avoid the status bar and other open panels. In small windows completion documentation may be hidden; `F1` still opens the full reference. Long signature help scrolls inside its popup. Passive popups keep editor focus. Explicit peek navigation keeps its keyboard controls; `Escape` returns to the editor. Peek and live diagnostics overlay the source without resizing it. A peek that cannot fit its controls closes safely when the window becomes too small.

Git Review and External Editors

Git actions affect deliberately staged content. Begin with read-only comparisons, save source explicitly, then review the full proposed staged file. Index recovery is separate from text Undo. The GUI does not commit, fetch or continue a merge/rebase.

10.1 Inspect repository state

For a saved netlist inside a Git working tree, the Git status button and `Edit > Git` show repository changes. Git 2.49 or newer must be installed on the application host. The status list uses Git's two-column state: the first column describes staged changes and the second describes saved working-tree changes. Unsaved buffer changes remain separate. Filter paths to narrow large result lists; Enter opens the selected source.

Git gutter markers compare the staged version with the current buffer: plus marks denote additions, bars modifications and chevrons deletions. Next/Previous Git change reveal folded source without editing it. The existing unsaved overview lane continues to compare disk with the buffer.

The comparison window shows read-only baseline/result panes for staged versus buffer, HEAD versus buffer, HEAD versus staged, disk versus buffer and up to 20 recent revisions of the current path. Refresh obtains a new snapshot; repository revisions are read as raw bytes, with display line endings normalized. Versions above 1 MiB and unsupported text representations are identified rather than silently truncated.

Very large differences use labelled coarse regions. Filters and encoding attributes are identified; their raw repository versions remain viewable where decodable, but their Git gutter is suppressed. When filter drivers are configured, raw status may differ from Git's converted-content status; the view identifies this limitation.

Symbolic links and submodule entries are not text baselines. Escape closes the view. Discard, commit, merge continuation and automatic fetch are not provided.

10.2 Review whole-file stage and unstage

Stage and unstage change the index. Review the complete proposed staged file before applying either action.

1. Save or reconcile the current source file.
2. Choose `Preview stage disk` or `Preview unstage`.
3. Compare the current staged content with its complete proposed replacement.
4. Acknowledge the review and choose `Apply reviewed`. Reopen the preview if a stale-state check rejects it.

Preview stage disk stages the saved file, including all changes in that file; it never saves a dirty buffer implicitly. Preview unstage restores the current HEAD version in the index, or removes an added entry before the first commit. Both operations leave disk bytes and editor text unchanged. Actions concern the current editor file, not the selected status-list row. Save dirty documents explicitly first; untitled and read-only documents cannot change the index.

The read-only panes show the current staged version and its proposed replacement. Review both panes, acknowledge replacing the staged content and choose Apply reviewed. The action rejects changes to disk, index, HEAD, configuration or attributes since the preview. Existing index/ref locks cause refusal; the editor never removes another client's locks.

10.3 Choose individual lines or hunks

Choose lines to stage compares the index with the saved disk file; Choose lines to unstage compares the index with HEAD. Check added or removed lines independently. Space toggles a line; Shift+Space or double-click toggles its whole contiguous hunk. Replacements contain separate removal and addition choices. Preview chosen lines shows the complete proposed staged file; acknowledge that preview before Apply reviewed. Unselected staged content and the current file mode are preserved. The selection is frozen during review; choose lines again to change it. No fuzzy patch matching is performed. Source changes, concurrent index changes and stale selections require a new review.

Partial actions require non-binary UTF-8 text and preserve the selected lines' raw LF/CRLF bytes and final-newline state. Their exact changed-middle diff is limited to four million comparison cells and 2048 changed lines, with no coarse fallback. Mode-only changes use whole-file actions. A selection that would join unterminated lines is refused; select the corresponding replacement too. An absent target removes the entry only when all its line removals are selected.

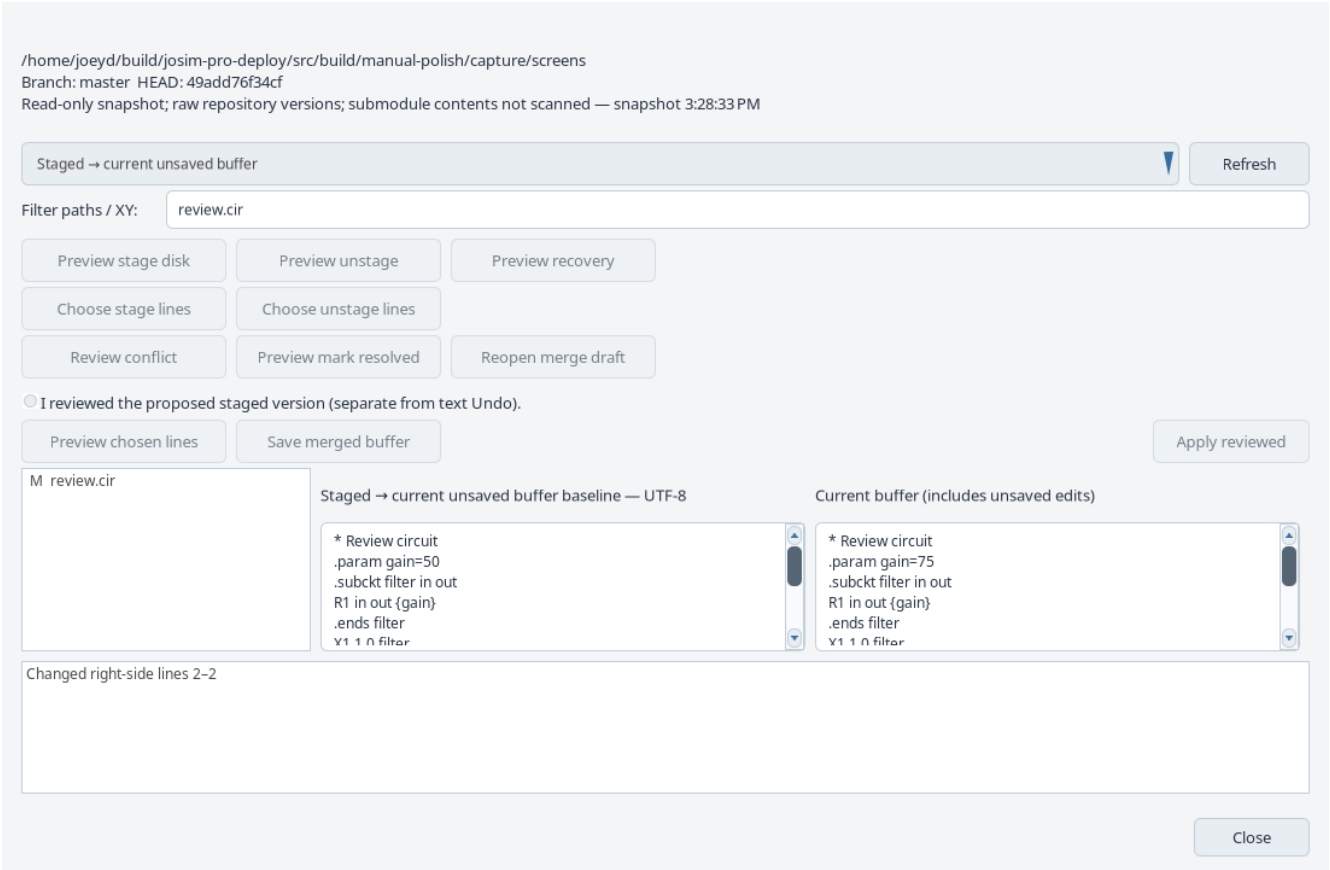


Figure 10.1: Git status and read-only comparison keep staged content, saved source and unsaved buffer changes distinct.

10.4 Recover a prior index action

Preview recovery restores the previous staged version for this file from the latest local action record, including after restarting the application. Recovery requires the file's staged version and HEAD identity to still match that action; other files' staged changes are preserved. Recovery is another reviewed index action, separate from text Undo.

Records retain previous blob bytes in `jpro-index-history` inside the worktree's Git metadata directory. Records are not automatically deleted. An interrupted operation may leave Git lock files; inspect repository state with Git before removing confirmed abandoned locks.

Preview preparation may create unreferenced local Git objects, but does not change the real index or any source file.

Index actions require ordinary regular-file entries and file-based Git refs. Ordinary staging refuses conflicts. Renames/copies, split/sparse indexes, special index flags, filter/encoding/ident/EOL attributes and automatic Git EOL conversion are refused. Use an external Git client for these cases. Preview and recovery blobs are limited to 1 MiB; built-in text Undo does not undo staging.

10.5 Review and validate a merge result

Review conflict shows the current file's immutable base (stage 1), stage 2 and stage 3 snapshots alongside the original conflicted disk text and an editable result. The source selector shows each full stage; absent stages are labelled explicitly. During rebase, stage 2 is upstream and stage 3 is the replayed commit.

Move the result caret into a conflict, then use stage 2, stage 3 or both; Use both places stage 2 before stage 3 and requires review for duplicate statements. Next conflict finds the next marker block, wrapping at the end. Diff3 base sections are excluded from side choices. Malformed blocks require manual editing. These choices change only the preview. Validate result checks UTF-8, remaining conflict markers and the existing non-simulating single-file syntax rules. Syntax errors and analysis limits block application. Semantic messages are shown for review and may depend on includes; validation is not proof of circuit correctness and does not run simulation.

Acknowledge the entire validated result and Apply reviewed to place it in the editor as one undoable edit. Base, both sides, original disk bytes and the reviewed result are preserved in `jpro-merge-history` inside the worktree's Git metadata directory before changing the buffer. Reopen merge draft retrieves the latest preserved result only when its conflict sides still match. Autosave is paused while the

Git review window is open and resumes after closing it. Closing or refreshing discards unapplied preview edits; applied results use the existing editor recovery and text Undo.

10.5.1 Save and mark the file resolved

Save merged buffer explicitly uses the ordinary guarded save and local-history workflow, including configured save formatting. Preview mark resolved then checks and displays the exact saved version and all preserved conflict sides. A separate acknowledgement and Apply reviewed replace only this file's unmerged index entries. Other staged files and other conflicts remain intact. Preview recovery restores the exact previous conflict stages without changing the saved result; compare or restore source text separately. Recovery requires the same HEAD and merge/rebase operation metadata. No commit, abort or continue operation is performed.

10.5.2 Know when to use an external Git client

Each blob/result is limited to 1 MiB. Binary or non-UTF-8 sides, unavailable objects, symlinks/submodules, conflicting modes, conversion-managed files, renames and deletion resolutions require an external Git client. Dirty buffers must be saved or reconciled before starting. Disk, index, HEAD, operation metadata, configuration or attribute changes invalidate a review; changes to the preview invalidate its acknowledgement. Index publication uses the existing Git locks and recovery contract. A source save and index replacement remain separate operations; no atomic transaction across both is claimed.

10.6 Use JoSIM-Pro from an external editor

The optional `josim-pro-lsp` executable shares the GUI's static authoring API with external editors over local standard input/output. The supplied VS Code client provides live Problems, documented token completion, hover values, definitions, references, document symbols, folding, include links and scoped Rename. It does not run simulation or write source files. The GUI continues using its direct authoring API.

Install the supplied VSIX through VS Code's Extensions menu and configure the user-level `JoSIM-Pro: Server Path` with the native executable's absolute path. Development Linux builds place it in `bin/linux`; Windows uses `josim-pro-lsp.exe`; macOS places it in the app's `Contents/MacOS` directory. A trusted workspace is required. Choose the JoSIM-Pro language mode for additional include filenames. Use the matching 2.9.0 server and native libraries; no runtime license is required for static authoring.

Analysis uses unsaved buffers and explicit includes. References cover the selected root/include graph. Open the containing deck to retain include scope; multiple open root contexts are reported explicitly. UTF-8 source with LF or CRLF endings is required. Source/message/open-buffer budgets remain visible through diagnostics or the language-server Output channel. Source changes cancel older work and invalidate prior diagnostics. The external client supplies token/scope completion; it does not reproduce every GUI snippet, positional suggestion or visual overlay.

Rename requires complete, unambiguous analysis and every affected file open. The server returns edits with exact document versions; VS Code owns applying, undoing and saving them. There are no server-side Git, shell, task, save or simulation commands. The native server communicates over STDIO without opening a network port.

Schematic Visualization

Inspect a parsed netlist as component blocks and connections, navigate subcircuits and prepare schematic images. The visualizer is a viewing and layout tool; source remains the owner of circuit connectivity.

11.1 Switching to the netlist visualizer

Use the visualizer toolbar button to switch the active netlist tab from text editor view to schematic visualizer view. The visualizer parses the current netlist, creates component blocks and connections, and displays the circuit as a schematic-like layout.

If parsing fails, the GUI keeps the tab in editor view and reports `Visualization Failed`. Fix the netlist syntax or include-path problem before trying again. Imported result tabs cannot be visualized because they do not contain a netlist.

When the netlist text changes, the visualizer is rebuilt from the current document revision the next time it is opened. Existing layout data is reused where possible; newly added components are placed automatically.

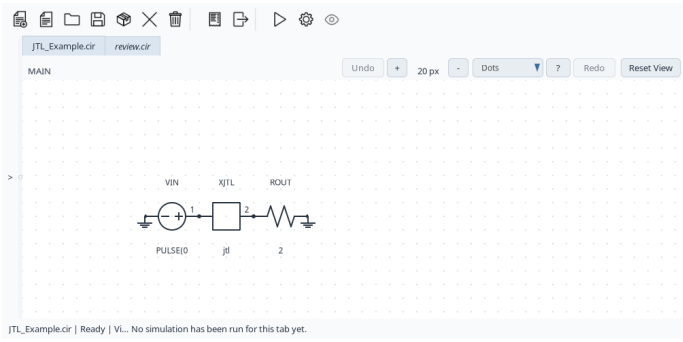


Figure 11.1: Visualizer view for the installed JTL example.

11.2 Visualizer controls and layout edits

The visualizer supports interactive inspection and cosmetic layout edits:

- drag a component to reposition it;
- drag empty canvas space to pan;
- use the mouse wheel to zoom around the pointer;
- use `Ctrl/Cmd+0` to reset zoom and pan;
- select a component and use arrow keys to move it;
- hold `Shift` with arrow keys for larger movement steps;
- press `R` or `Shift+R` to rotate the selected component;
- use `Ctrl/Cmd+Z` and `Ctrl/Cmd+Y` for visualizer layout undo/redo;
- press `G`, or use the grid-style selector, to cycle dots, lines, and no grid;
- use `Ctrl/Cmd+Mouse Wheel` or the grid controls to change grid spacing.

These edits are layout edits. They affect how the netlist is drawn in the GUI and exported as a schematic; they do not change the electrical meaning of the circuit unless the underlying netlist text is edited separately.

11.3 Subcircuit navigation

Subcircuits are drawn as blocks with pins and a subcircuit label. Double-click a subcircuit block to enter the referenced subcircuit. The visualizer then redraws that subcircuit scope. Use the visible `Back` button, `Backspace`, or `Alt+Left` to return to the parent schematic.

If a subcircuit reference cannot be resolved, the GUI reports `Open Subcircuit Failed` or `Subcircuit Not Available`. In that case, check the subcircuit name, included files, and whether the parser can see the referenced `.SUBCKT` definition.

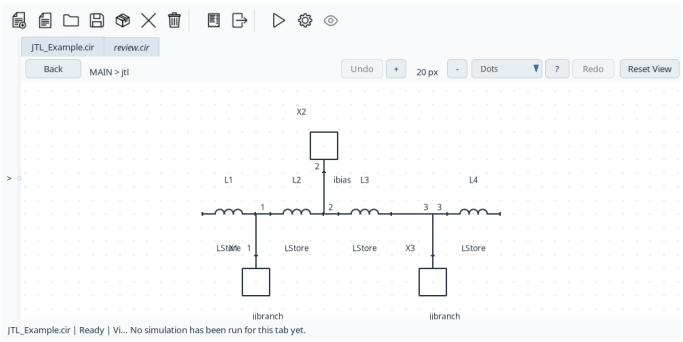


Figure 11.2: Subcircuit navigation view with parent/child schematic context.

11.4 Layout persistence

Visualizer layout is persisted into the netlist as tagged comment data. When layout changes are saved, the GUI writes an encoded `SCHM_<design>` comment block for the active design scope. Subcircuit layouts can be saved independently for the subcircuit scope being viewed.

This means the layout travels with the netlist file and can be restored when the same design is opened later. It also means layout changes make the file modified even though the circuit semantics are unchanged. The GUI labels these as cosmetic layout changes so they can be distinguished from semantic netlist edits.

Do not manually edit or delete the encoded schematic comment block unless you intend to discard the stored layout. If the netlist topology changes, the visualizer reuses stored positions for matching components and places new components automatically.

11.5 Schematic export

While the visualizer is active, use the toolbar export button to open `Export schematic`. The schematic export dialog supports:

- `PNG image (*.png)` for raster output;
- `SVG vector schematic (*.svg)` for scalable vector output.

If no extension is supplied, the GUI uses the selected filter to add `.png` or `.svg`. The export path defaults from the active netlist path when available. Exported schematics use the prepared visualizer scene rather than a screen capture, so the saved file is suitable for reports and presentation material.

Use plot export, not schematic export, for waveform/result figures. The same toolbar button chooses the correct workflow based on whether the active tab is showing the visualizer or waveform results.

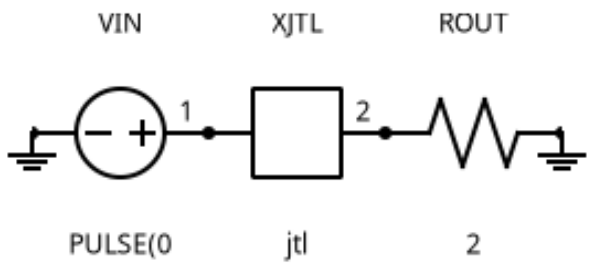


Figure 11.3: The JTL main scope exported as a schematic image. Export includes the prepared circuit scene rather than the application chrome.

GUI Simulation and Results

This chapter documents the GUI-side simulation, waveform, Trace Manager, imported-result, analysis, and plot-export workflows. The quick-start chapter gives a first run; this chapter is the task reference for day-to-day result work.

12.1 Simulation lifecycle

Use the toolbar simulate button, or the equivalent run command in the active tab, to run the current netlist. The GUI runs the same JoSIM-Pro runtime used by `josim-pro-cli`; the difference is that progress, diagnostics, and waveform output are managed by the active GUI tab.

The GUI first reads the saved netlist and validates every analysis and output destination. It then executes analyses serially in source order, with fresh simulation state for each run. Each successful run publishes its own files and retains its requested traces. The result selector identifies runs by block name, analysis identifier and completion status.

During a run, the tab shows a progress overlay with a progress bar and `Cancel` button. If the runtime reports percent-style progress, the overlay shows the percentage and sample count; otherwise it shows the current stage text. The context/status area also reports the active stage.

Cancelling requests an abort of the active analysis and changes the overlay button to `Cancelling...`. Wait for the worker to finish before starting another execution. Completed analyses from that execution remain available; the interrupted analysis does not publish incomplete staged output. Diagnostics identify failed, cancelled and unstarted analyses.

12.2 Preconditions for running

Simulation is available only for netlist tabs, not imported result tabs. A `.csv` or `.tsv` tab can be inspected, analysed, and exported, but it cannot be simulated because it does not contain source netlist text.

The GUI runs from a saved netlist file. If the active tab is untitled, the GUI prompts to save it before the first run. If the tab has unsaved semantic netlist edits, the GUI asks whether to save and run the current text, run the last saved file, or cancel. Running the last saved file leaves the displayed result state marked stale when the unsaved text differs from the saved file.

Resolve external file changes before running. If the file on disk changed outside the GUI, the tab reports an external-change state and asks whether to reload or keep the current in-memory copy before simulation continues.

> **License requirement.** GUI simulation requires the latest SUN Magnetics License Manager and a working `smlicensed` daemon. Degraded mode is for inspection and recovery; it is not a licensed simulation mode.

12.3 Run settings

Open `Run > Settings` from a netlist tab to edit defaults for every analysis in the active netlist. Netlist options and block-local options can override these defaults. Run settings are disabled while that tab is simulating and are unavailable for imported result tabs. Changing run settings after a successful run marks existing output stale until the tab is re-run.

Analysis type Chooses `Phase` or `Voltage` analysis for the runtime.

Integration method Chooses `BDF` or `TRAP`.

Variable-step mode Enables variable-step mode only for the supported `Phase` + `BDF` combination. Other combinations are rejected by inline validation.

Use UIC Passes the UIC flag through to the simulation runtime.

Output precision Sets output digits in the runtime-supported range `0` to `17`. Invalid values block `OK`.

Compressed output Requests compressed text output when supported by the runtime and output path.

Deterministic seed Enables a fixed unsigned seed for reproducibility. If disabled, the seed field is ignored and the runtime receives seed `0`.

Defaults are loaded from the installed runtime configuration when the tab is created. Keep the defaults for normal first-use simulations. Change settings when you need CLI-equivalent reproducibility, a support-requested solver configuration, or a deliberate comparison between analysis/integration modes.

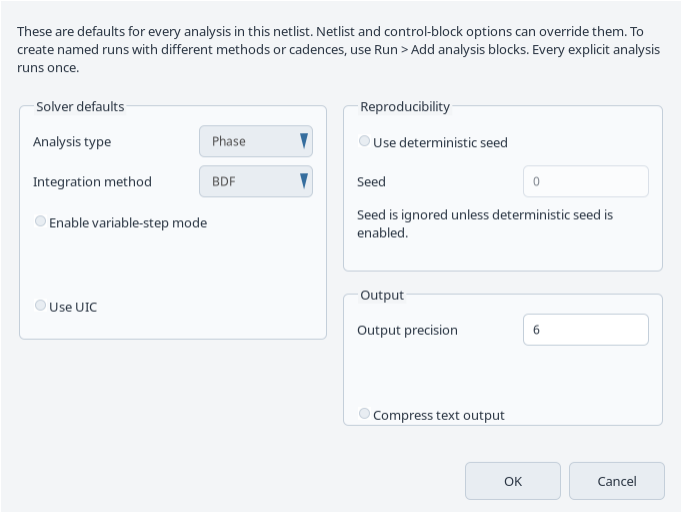


Figure 12.1: Run Settings supplies deck defaults; control blocks can override them.

12.4 Create multiple runs

Open `Run > Add analysis blocks...` from an editable netlist tab. The dialog inserts ordinary netlist commands that can be inspected and edited. It is disabled while a simulation is running and for imported result tabs.

To compare TRAP and BDF at three output cadences:

1. Enter a unique block name prefix, for example `compare`.
2. Select `TRAP and BDF`, a simulation step of `1p`, a stop time of `20p`, and an output start of `0`.
3. Enter `2p, 4p, 5p` as the output cadences and choose traces present in the circuit, for example `V(1)`.
4. Review the preview and select `Insert into netlist`. Save the netlist, then press Run.

This creates `compare_trap` and `compare_bdf`, each with three independent transients: six runs in total. Each block uses fixed simulation steps and its own integration method and traces. Separate output files receive automatic names. The simulation step controls numerical advancement; the output cadence controls the retained sample grid.

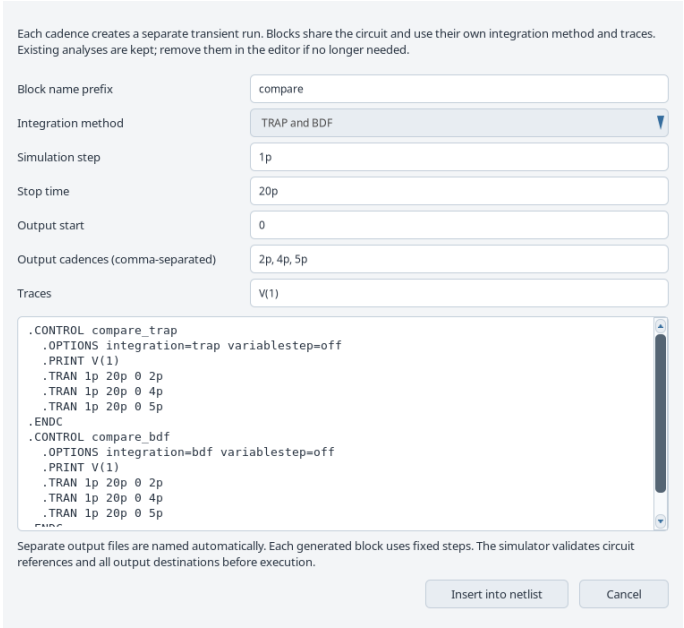


Figure 12.2: Two named blocks, each containing three transient analyses.

The dialog accepts literal times in seconds or with suffixes such as `p` and `n`. It keeps existing analyses and inserts the generated blocks before `.END`; remove unwanted existing analyses in the editor. A repeated block name is rejected. The simulator checks circuit references and output destinations before execution.

For other combinations, add more blocks with distinct prefixes or edit the generated commands. Blocks may contain multiple supported analyses and their own options and output directives. Use the netlist syntax chapter for parameters, explicit `.FILE` destinations and IV analyses.

After execution, choose a run from the selector above the plots. Each selection loads that run's own traces and sampling grid; IV results use voltage as the horizontal axis. In this example the three cadences produce 11, 6 and 5 samples respectively, including zero. Changing the selection rebuilds the plot workspace: export derived traces or customised plots before switching runs.

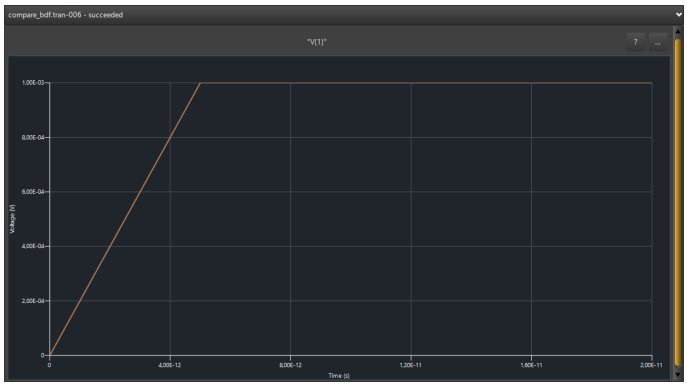


Figure 12.3: Selecting the BDF run with a 5 ps output cadence. Other run results remain separately selectable.

12.5 Result states and diagnostics

The active tab reports both document state and result state. Use these states to decide whether the waveform view reflects the current netlist and run settings.

No results The tab has not produced or imported waveform data.

Running The tab has an active simulation worker. The result view updates only after the run completes.

Cancelling A cancel request was issued; wait for the worker to finish before starting another run.

Results current The waveform output matches the saved netlist and current run settings.

Results stale Semantic netlist text or run settings changed after the last successful run. Re-run before trusting the waveform as current output.

Run failed The runtime reported an error. Diagnostics include the latest failure stage and error text when available.

Run cancelled The last run was cancelled by the user. Previous successful results may still be visible.

Imported results The tab contains loaded `.csv` or `.tsv` result data rather than a source netlist.

Use the diagnostics toggle in the tab/status area when a run fails, output looks stale, an external-change warning appears, or an export reports a problem. Diagnostics include the active file, surface mode, document state, result state, recent progress stage, last run/export summary, latest issue, warnings, and recent run history.

Native output-file routing is also reported in diagnostics. Each successful analysis publishes its own output plan. If `.FILE` appears only after existing `.PRINT`, `.SAVE`, or `.PLOT` commands, the GUI warns that `.FILE` routes only subsequent output requests; move `.FILE` before the output commands when file routing is intended.

12.6 Waveform result view

Successful simulation opens the waveform result view in the same tab. Imported result files also open directly into this waveform surface. The waveform workspace supports multiple plot panels, visible/hidden traces, trace grouping, crosshair inspection, grid display, analysis outputs, annotations, and export.

Use `View > Grid` to toggle plot-grid display for the active waveform workspace. Use `View > Sync Crosshairs` to synchronize crosshair movement across plot panels. These view preferences are stored with GUI preferences and reapplied to new result workspaces.

Traces can be inspected directly in the plot area. The screenshot below shows the waveform surface with multiple plot panels and visible traces from the installed JTL example.

12.7 Trace Manager basics

Open `Run > Trace Manager` or the toolbar trace-manager button after results exist. Trace Manager is disabled while the tab is simulating and is enabled for both simulated results and imported `.csv`/`.tsv` result tabs.

The Trace Manager dialog contains a trace list on the left and task tabs on the right:

- **Analysis**: create derived traces, measurements, masks, FFT outputs, and comparison plots.
- **Annotations**: add exact vertical, horizontal, point, or interval annotations.
- **Selection**: rename selected items, toggle visibility, assign groups, reorder traces, and override trace colors.
- **Appearance**: choose plot-wide theme and color settings.
- **Layout**: choose plot-panel layout mode.

The trace list supports search, multi-selection, `Show All`, `Hide All`, removal of derived outputs, and non-destructive comparison plots. Use `Ctrl/Cmd+A` to select all items and `Ctrl/Cmd`-click for multi-selection.

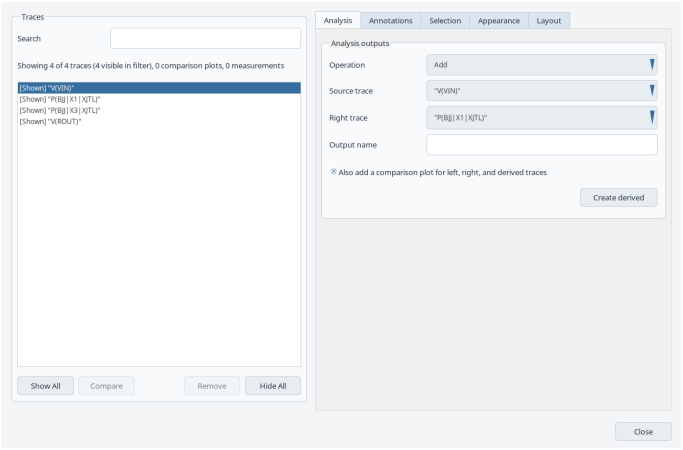


Figure 12.5: Trace Manager with trace selection and analysis controls.

12.8 Visibility, order, groups, and comparison plots

Use the **Selection** tab for practical trace organization:

- toggle `Visible for selected trace(s)` to show or hide selected traces;
- edit `Selected item name` to rename a trace or comparison plot for display/export;
- use `Assign group` and `Reset` to move traces into named plot groups or return them to default grouping;
- use `Move Up` and `Move Down` to change plotting order;
- apply or clear selected-trace line color overrides;
- use `Compare` to create an additive comparison plot from the current selection.

Trace grouping is a display organization tool. It does not modify the source netlist or simulation output. Comparison plots are also non-destructive; they add alternate views over existing or derived traces rather than replacing the original data.

The waveform workspace remembers layout, color, and analysis recipes through session state. If a later run or import cannot restore a previous analysis recipe because a trace name or axis is missing, the tab reports restore warnings in diagnostics.

12.9 Analysis outputs

The **Analysis** tab creates new outputs from existing traces. Choose a source trace, optional right-hand trace, an operation, scalar parameters where required, and an output name. For trace-producing operations, `Create derived` adds a derived trace to the workspace. For measurement-producing operations, the button changes to `Measure` and records a measurement output, often with markers.

Available operation families include:

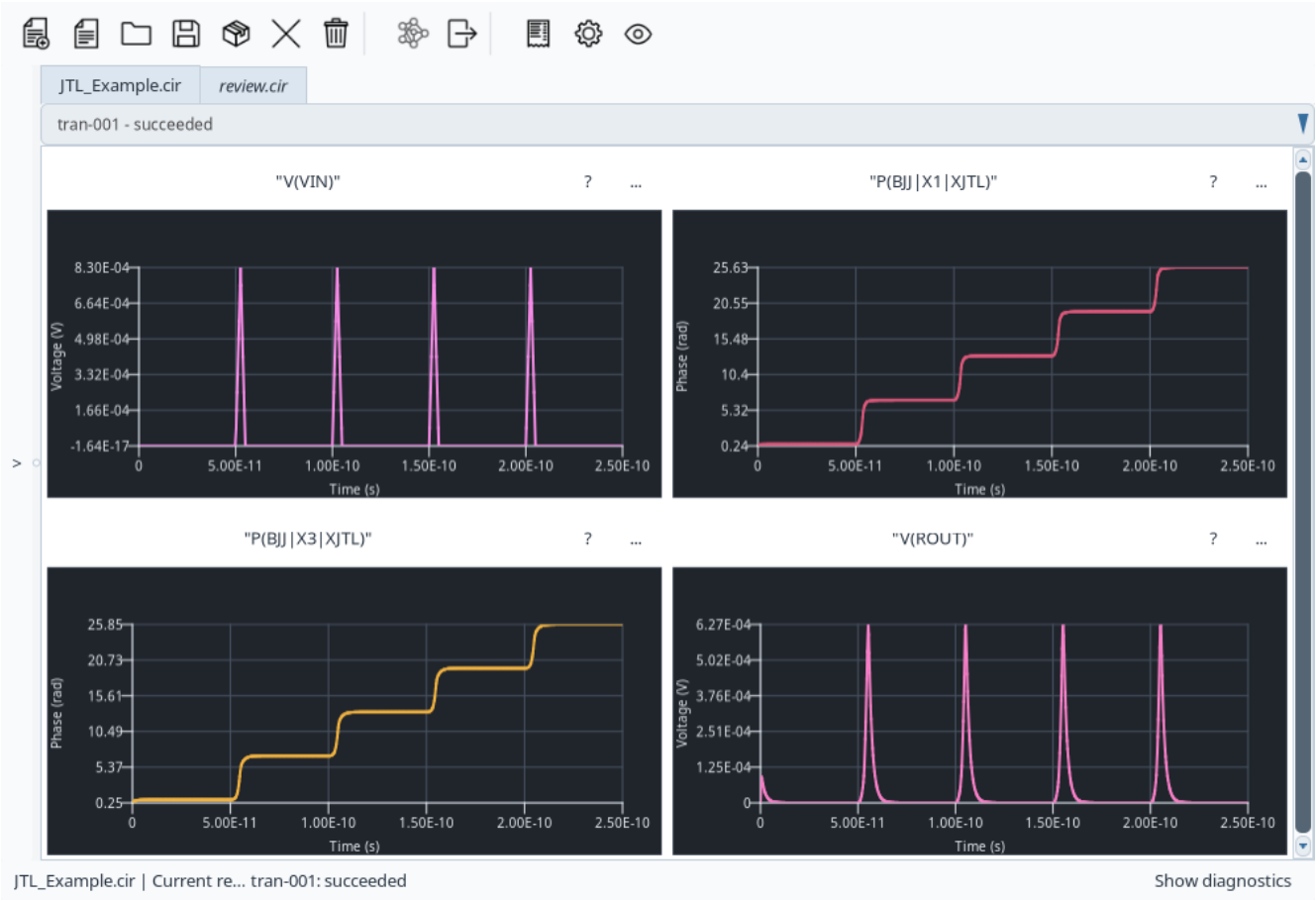


Figure 12.4: The JTL result with separate voltage and phase axes. The run selector identifies the completed analysis.

- arithmetic and scalar arithmetic such as add, subtract, multiply, divide, `x+k`, `x-k`, `x*k`, and `x/k`;
- transforms such as absolute value, negation, DC removal, subtract-initial, logarithmic forms, derivative, and cumulative integral;
- normalization and smoothing operations;
- threshold masks, range masks, trace-comparison masks, and mask logic;
- FFT magnitude, FFT dB magnitude, FFT phase, and power spectrum;
- measurements such as min, max, mean, RMS, peak-to-peak, crossing time/count, error metrics, rise/fall time, propagation delay, pulse width, duty cycle, period/frequency estimate, overshoot, and settling time.

Operations that combine traces require compatible x-axis metadata. FFT operations require a uniform time axis. The dialog exposes FFT controls for window selection, zero-padding policy, optional DC removal, and dB floor epsilon. Invalid choices are rejected in the analysis validation area rather than producing a silent output.

Derived traces and measurements are workspace outputs. They can be hidden, renamed, grouped, exported, removed, and restored as analysis recipes when session state can match their source traces.

Static operating points. A deck containing a standalone `.OP` reports voltage, phase, current and convergence scalars in Tasks/Output. The editor remains visible and the run summary reads *Operating point ready*. No time-domain plot is created. See Section 16.7.3 for the supported devices and equilibrium rules.

12.10 Manual annotations

Use the **Annotations** tab to add exact markers from numeric coordinates. Choose a trace target, marker kind, label, and coordinates, then click `Add exact`. Supported marker kinds are:

- `Vertical marker`;
- `Horizontal marker`;
- `Point annotation`;
- `Interval marker`.

Annotations can be visible or hidden. Enable `Include in visual exports` when the marker should appear in image/vector plot exports. This flag affects visual exports only; CSV/TSV data export writes numeric trace data, not rendered annotation geometry.

12.11 AC magnitude and angle views

Run a standalone `.AC` deck with explicit `.PRINT AC` selectors. Each complex trace produces a linear magnitude panel in its physical unit and a phasor-angle panel in degrees. The horizontal axis is frequency in Hz, separate from time-domain FFT plots. Physical phase excursion `P(...)` is measured in radians, independently of its phasor angle.

The frequency axis starts logarithmic. Use the plot menu's `Logarithmic frequency` option to switch every AC panel between log and linear scales. Zoom, pan and annotations use the selected frequency scale. Enable the existing synchronized-cursor setting to link panels at the same frequency, including panels with different visible ranges. Exactly zero response displays zero magnitude and an undefined angle, with a gap in the angle curve. Tasks/Output includes the resolved stimulus, effective seed and bias/model declarations.

For gain or impedance, request a supported complex ratio in `.PRINT AC`. The ordinary real-trace analysis operations are unavailable for AC projections: FFT, DC removal, triggers, calculus, energy and transient measurements produce an explicit frequency-domain diagnostic. This prevents a magnitude curve from being treated as a time signal or complex gain.

Image and vector exports use the displayed magnitude/angle views. For data export choose CSV with `All traces`: this saves every original Re/Im trace and its metadata. Use a `.csv.gz` filename for gzip. Partial data scopes and TSV are rejected for AC. Reopen typed CSV/CSV.gz through `File > Open...`, the browser or drag-and-drop to restore complex frequency results. Ordinary CSV/TSV waveform import remains separate.

12.12 Imported CSV and TSV result data

Use `File > Open...`, the file browser, or drag-and-drop to open existing `.csv` and `.tsv` result files. Imported result files become waveform tabs immediately; the editor surface is hidden because the tab represents data, not source text.

For ordinary numeric CSV/TSV, the import reader accepts comma-delimited and tab-delimited data. Blank lines and lines beginning with `#` are ignored. A header row is optional; when present, the first column is used as the x-axis label and every remaining column becomes a trace. When no header is present, the first column is treated as `Time (s)` and trace names are generated automatically.

CSV extension Uses comma as the delimiter.

TSV extension Uses tab as the delimiter.

Unknown delimiter Uses the more frequent comma/tab character in the first data line.

Header row present Uses header text for axis and trace names. Units in parentheses are recognized where possible.

No header row Generates `Time (s)` and `Trace 1`, `Trace 2`, and so on.

Non-numeric cell Import fails with the offending line number.

Mismatched column count Import fails with the offending line number and expected column count.

Too few samples Ordinary waveform imports require at least two numeric samples. Typed AC files may contain one frequency.

Ordered axis required Time, frequency, lag, and sample axes must be non-decreasing.

Imported result tabs support Trace Manager analysis and plot export. They do not support run settings, simulation, save/save-as as netlists, or netlist visualization. If an imported file changes on disk, the GUI asks whether to reload the CSV/TSV data and rebuild the plot workspace.

12.13 Plot export studio

When the active tab is showing waveform results, the toolbar export button opens `Plot export studio`. This is separate from schematic export: schematic export is available from the visualizer; plot export is available from the waveform/result view.

Plot export studio provides a live preview and file output using the same plot-export backend. The left side controls preset, format, scope, trace selection, style, dimensions, and advanced format disclosure. The right side shows the preview and export-readiness summary.

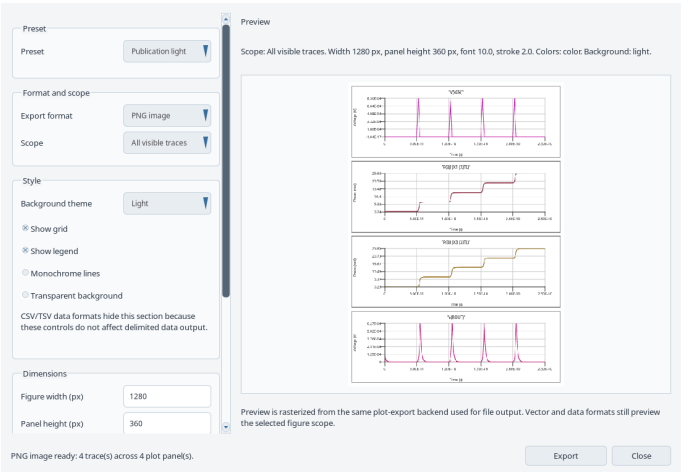


Figure 12.6: Plot Export Studio configured for a PNG plot export with preview.

Supported export formats are:

- raster figures: `PNG image`, `JPEG image`;
- vector figures: `SVG vector`, `PDF vector`, `EPS vector`, `TikZ/LaTeX`;
- data files: `CSV data`, `TSV data`.

Supported export scopes are:

- `Selected traces`;
- `Chosen visible group`;
- `All visible traces`;
- `All traces`.

For image/vector exports, use the presets `Publication light`, `Presentation dark`, or `Monochrome print`, then adjust grid, legend, monochrome, transparent background, figure width, panel height, font size, and stroke thickness as needed. Minimum values are enforced: width at least `480` px, panel height at least `180` px, font size at least `7`, and stroke thickness at least `1`. JPEG export does not support transparent background.

For CSV/TSV export, style and dimension controls are hidden because delimited data output writes x-axis values and selected trace series only. Delimited export requires compatible x-axis metadata across the selected groups/traces; if axes are incompatible, the export fails with an explanatory message.

The save dialog uses the selected export format to choose the default extension. If no extension is supplied, the GUI adds the format's default extension. Successful and failed exports are recorded in tab diagnostics.

12.14 Troubleshooting hooks

Use these checks when simulation or result handling does not behave as expected:

Simulation button is disabled Confirm the active tab is a netlist tab, not an imported `.csv`/`.tsv` result tab.

Run settings are disabled Wait for the active simulation to finish, or switch from an imported result tab to a netlist tab.

Run prompts for save The tab is untitled or has unsaved semantic edits. Save the netlist if the current text should be simulated.

Results are marked stale Re-run after changing semantic netlist text or run settings. Cosmetic visualizer layout changes do not invalidate simulation results.

No waveform appears Check diagnostics for the failing runtime stage and confirm the netlist contains output requests such as `.PLOT`, `.PRINT`, or `.SAVE`.

Native output file is missing If using `.FILE`, place it before the output requests it should route. Check diagnostics for native-output routing messages.

CSV/TSV import fails Check the reported line for non-numeric cells, mismatched column count, too few samples, or decreasing x-axis values.

Trace Manager cannot create output Check operation compatibility. Binary operations require compatible axes; FFT requires a uniform time axis; scalar operations require valid numeric parameters.

Plot export fails Check export scope and axis compatibility. Data export cannot combine traces with incompatible x-axis metadata. JPEG cannot use transparent background.

Session restore warns about analysis A saved analysis recipe referenced a trace or axis that is no longer present. Recreate the derived output or measurement from current traces.

For licensing, package launch, or System ID problems, use the installation and troubleshooting chapters first. For CLI-equivalent run settings, use the GUI-to-CLI equivalence chapter.

GUI-to-CLI Equivalence

This chapter maps interactive GUI workflows to the command-line tools used for automation, batch runs, reproducibility checks, and Optimum. Use the GUI for interactive editing, visualization, simulation, trace analysis, and result export. Use `josim-pro-cli` for scripted simulation and native output generation. Use `optimum-cli` for TOML-driven verification, margin, yield, and optimization.

The GUI and CLI execute the same analyses and control blocks. `Run > Settings` supplies deck defaults; `Run > Add analysis blocks...` inserts `.CONTROL`, `.OPTIONS`, `.PRINT` and `.TRAN` commands. Save that netlist and pass it unchanged to `josim-pro-cli` to execute the same run plan. See Create multiple runs for the GUI workflow.

13.1 Simulation workflow equivalence

GUI workflow	CLI-oriented equivalent
Open a `.cir` file and run it in the active tab	`josim-pro-cli -o results.csv design.cir`
Open saved CSV/TSV result data for inspection	Generate result data with `josim-pro-cli -o results.csv design.cir`, then open the result data in the GUI
Save a netlist before running	Use a stable path as the positional input or with `-i`/`--input`
Run a netlist from standard input	`josim-pro-cli -i STDIN < design.cir`
Generate output that the GUI can import later	Prefer `CSV` or `TSV`-compatible text output; the GUI opens `.csv` and `.tsv` as result-data tabs
Use the GUI diagnostics/status area after a failed run	Read the terminal diagnostics and fix the first parse, license, runtime-option, or output-routing error before rerunning

The full command-line option reference is Chapter 15. Netlist syntax remains in Chapter 16.

13.2 Run Settings equivalence

GUI Run Settings control	CLI equivalent
Analysis: `Phase`	`josim-pro-cli -a 0 design.cir`
Analysis: `Voltage`	`josim-pro-cli -a 1 design.cir`
Integration: `BDF`	`josim-pro-cli -x 0 design.cir`
Integration: `TRAP`	`josim-pro-cli -x 1 design.cir`
Enable variable-step mode	`josim-pro-cli --varstep design.cir`; valid only for `PHASE + BDF`
Output precision	`josim-pro-cli -P 8 design.cir`, changing the integer value as needed
Compress text output	`josim-pro-cli -c -o results.csv design.cir`; use `CSV` or `DAT`, not `BIN`/`RAW`
Use deterministic seed	`josim-pro-cli -r 12345 design.cir`
Disable Josephson-junction Newton solving	`josim-pro-cli --no-newtonj design.cir`

CLI overrides are run-local. If the same behavior is a design requirement, record it in the netlist with the appropriate `.OPTION` or simulation-control statement instead of relying only on a shell command.

13.3 Output, streaming, and import equivalence

GUI-side goal	CLI-oriented equivalent
Save result data for later GUI inspection	`josim-pro-cli -o results.csv design.cir`, then open `results.csv` in the GUI
Write primary output to the terminal	Omit `-o`/`--output`, or use `-o STDOUT`
Write a specific output file	`josim-pro-cli --output=results.csv design.cir` or `josim-pro-cli -o results.csv design.cir`
Split output groups into multiple files	Use netlist `.FILE` directives before the `.PRINT`, `.SAVE`, or `.PLOT` statements they should route
Stream a long text-output run	`josim-pro-cli -p -o results.csv design.cir`
Generate buffered binary or raw data	Keep streaming disabled: `josim-pro-cli -o results.bin design.cir` or `josim-pro-cli -o results.raw design.cir`
Compress text output	`josim-pro-cli -c -o results.csv design.cir`; the written file receives a `.gz` suffix when needed
Avoid output/input overwrite mistakes	Use distinct paths; the CLI rejects identical input and output file names

Streaming currently targets text workflows: `CSV`, `DAT`, and `STDOUT`. If a streamed target asks for `BIN`, `RAW`, an extensionless raw path, or another unsupported extension, the runtime falls back to CSV and reports the fallback.

13.4 Licensing equivalence

GUI licensing task	CLI-oriented equivalent
Generate a system identifier from `Help > About > Generate System ID...`	`josim-pro-cli -s`; this asks the installed `smlicensed` daemon to generate `SM_SysID.txt`
Set or correct a license-file path	`josim-pro-cli -l /path/to/sm_license.txt`; run this by itself to store the path
Check that the executable is present	`josim-pro-cli -v`; this prints version/build information and exits
Diagnose license-daemon problems	Confirm that the latest SUN Magnetics License Manager and `smlicensed` daemon are installed before debugging the GUI or CLI executable

Do not use `josim-pro-cli -l /path/to/sm_license.txt -v` to save a license path. The version option exits before a license-path update is performed.

13.5 Optimum handoff equivalence

Optimum is launched with `optimum-cli`. The GUI handoff is file based.

GUI-side step	Optimum CLI step
Create or edit the design netlist in the GUI	Save the netlist as a stable `.cir` file and reference it from `verify.circuit` in the TOML file
Keep related files together with `File > Open Folder...`	Run `optimum-cli` from that project/examples directory when using relative output paths
Use the installed DFF example as an Optimum starting point	Run `optimum-cli DFF_Example_config.toml` from the installed examples directory
Switch between Verify, Margin, Yield, and Optimize	Edit only the TOML `mode` value: `"verify"`, `"margin"`, `"yield"`, or `"optimize"`
Inspect the optimized circuit	Open the generated `DFF_Example_optimized.cir` or other `optimize.output` file in the GUI
Inspect generated CSV/TSV data	Open the generated `.csv` or `.tsv` file in the GUI for waveform inspection, Trace Manager analysis, and plot export
Generate an Optimum-side system identifier	Use `optimum-cli -s`; this also delegates to the installed `smlicensed` daemon

For details on TOML keys, supported Optimum modes, installed DFF example files, and output handling, see Chapter 14.

Optimum CLI Workflows: Verify, Margin, Yield, and Optimize

Optimum is the specification-checking and design-tuning companion to JoSIM®-Pro. Optimum is driven by `optimum-cli` and a TOML configuration file. The GUI can prepare and inspect the files used by Optimum, while the Optimum run itself is launched from the command line.

Use the GUI to edit and save netlists, inspect an optimized netlist, or open CSV/TSV result data generated by CLI workflows. Use `optimum-cli` when the task is specification verification, margin search, yield estimation, or automatic parameter optimization.

14.1 GUI handoff model

A normal Optimum workflow in a project that uses both the GUI and command-line tools is:

1. Open or create the design netlist in the GUI.
2. Save the netlist as a stable `.cir` file. If the design uses includes, keep the TOML file, circuit, specification file, and included files in a predictable project directory.
3. Create or edit an Optimum TOML configuration that points to the saved circuit and the verification specification. For EVENT verification, `--event-config` can list the circuit's available Josephson-junction phase signals and generate the rule block interactively.
4. Run `optimum-cli` from a terminal.
5. Reopen the resulting optimized `.cir` file or any generated CSV/TSV result data in the GUI for inspection, waveform analysis, and plot export.

This is a handoff workflow, not a hidden GUI integration. The GUI's `File > Open...` workflow accepts `.cir`, `.csv`, and `.tsv` files, so it is suitable for reviewing Optimum inputs and downstream result data. It does not replace `optimum-cli` for the optimization run itself.

14.2 Installed DFF example

The package installs the canonical Optimum example files:

- `DFF_Example.cir` – circuit netlist,
- `DFF_Example.sp` – SP-style verification specification, and
- `DFF_Example_config.toml` – Optimum configuration.

These files are installed with the other examples under the package's `examples` directory. Use them for the first Optimum test because they are available after installation.

From the installed examples directory, run:

```
optimum-cli DFF_Example_config.toml
```

The provided configuration uses `mode="optimize"` by default. To test the other modes without changing file paths, edit only the top-level `mode` line:

```
#mode="verify"
#mode="margin"
#mode="yield"
mode="optimize"
```

If `optimize.output` remains set to `"DFF_Example_optimized.cir"`, an optimization run writes the optimized netlist beside the example files. `DFF_Example_optimized.cir` is generated output, not a required installed example file. Open that generated `.cir` file in the GUI to inspect the changed `.PARAM` values or to run a normal GUI simulation.

14.3 CLI invocation and licensing

`optimum-cli` accepts either a positional TOML file or `-i/--input`. A valid JoSIM-Pro license and the installed SUN Magnetics License Manager are still required for simulation-backed verification. The `--event-config` helper only parses the circuit and exits after writing an EVENT rule; it does not run a simulation or perform license validation.

Option	Meaning
`input_file`	Positional TOML input file.
`-i, --input`	TOML input file path. Equivalent to the positional file.
`--event-config`	Interactively list Josephson-junction phase signals available in the input circuit and write one EVENT `[[verify.rules]]` block to standard output.
`-l, --license`	Save or use a license-file path. If no input file is supplied, the path is saved and the tool exits after validation.
`-s, --sysid`	Ask the installed `smlicensed` daemon to generate `SM_SysID.txt` in the current directory.
`-r, --seed`	Override the random-number seed used by stochastic Optimum paths.
`-m, --minimal`	Reduce normal execution feedback.
`-V, --verbose`	Set verbose execution feedback, with `0` for off and `1` for on.
`-v, --version`	Print version information.
`-h, --help`	Print help and exit.

Examples:

```
# Positional TOML file
optimum-cli DFF_Example_config.toml

# Explicit input option
optimum-cli -i DFF_Example_config.toml

# Deterministic stochastic run
optimum-cli -r 123456 DFF_Example_config.toml

# Generate a licensing system identifier through smlicensed
optimum-cli -s

# Generate an EVENT rule from the DFF circuit
optimum-cli --event-config DFF_Example.cir
```

The EVENT helper writes prompts and the phase-signal inventory to standard error, while the generated TOML rule is written to standard output. This allows the rule fragment to be redirected without mixing it with prompts:

```
optimum-cli --event-config DFF_Example.cir > event_rule.toml
```

The helper accepts either the displayed signal number or its label. It supports `transfer`, `fanout`, `merge`, `delay`, `phase_shift`, `logic`, `clocked_logic`, and `clocked_transfer`. For logic rules it also asks for the function. It does not infer expected event times, waveform shapes, or signal roles.

14.4 Mode overview

At the top level, set `mode` to one of the four supported mode names. The executable lower-cases the mode value before dispatching.

Mode	Use case
`verify`	Run a nominal circuit and check it against a specification. This is the gate that all other modes must pass first.
`margin`	Search how far selected parameters can move down and up before verification fails.
`yield`	Estimate the pass fraction under randomized parameter variation.
`optimize`	Search for a better parameter center using verification and margin analysis as the score path.

All four modes use the same top-level execution sequence:

1. parse the TOML configuration,
2. load the circuit and verification specification,
3. run baseline verification on the nominal circuit, and
4. continue with the selected mode only if that nominal verification passes.

If the nominal circuit does not verify, Optimum stops before running Margin, Yield, or Optimize.

14.5 Runtime contract

Optimum currently enforces fixed-step transient `PHASE` analysis with `BDF` integration. Variable-step execution is intentionally disabled inside Optimum because the verification paths assume uniformly sampled time data.

In practice:

- Optimum is not the path for comparing `PHASE` against `VOLTAGE` operation.
- Optimum is not the path for comparing `BDF` against trapezoidal integration.
- Netlist or runtime settings that would move the run outside fixed-step `PHASE + BDF` are overridden or rejected by the Optimum runtime contract.

Use the GUI or `josim-pro-cli` for interactive solver-option comparisons. Use Optimum after the verification circuit is known to run correctly under the fixed-step `PHASE + BDF` contract.

14.6 File paths and project layout

The TOML configuration file is the anchor for Optimum inputs. Relative values in `verify.circuit`, and in `verify.file` when SP or CSV verification uses it, are resolved relative to the configuration file location. The circuit's own includes are then resolved through the normal simulator input handling from the circuit working directory.

Recommended layout:

```
my_dff_project/
  DFF_Example_config.toml
  DFF_Example.cir
  DFF_Example.sp
  base_cells.cir
```

For output files, prefer explicit names in the same project directory. Margin and Optimize write relative output paths against the circuit working directory. Yield roll-off output uses the process working directory, so run `optimum-cli` from the project directory when using relative `yield.output` paths.

14.7 Minimal configurations

14.7.1 Verify

```
mode = "verify"

[verify]
method = "spec_file"
file = "DFF_Example.sp"
circuit = "DFF_Example.cir"
threshold = 0.35
```

Use this when you only need to confirm that the circuit satisfies the reference behavior.

For EVENT verification, no SP or CSV reference file is required:

```toml
mode = "verify"

[verify]
method = "event"
circuit = "DFF_Example.cir"

[[verify.rules]]
type = "clocked_transfer"
input = "B1"
clock = "B6"
output = "B7"
```

Bare phase-signal names such as `B1` are accepted. Use `--event-config DFF_Example.cir` to list the circuit's available Josephson-junction phase signals and generate this rule block interactively.

14.7.2 Margin

```toml
mode = "margin"

[verify]
method = "spec_file"
file = "DFF_Example.sp"
circuit = "DFF_Example.cir"
threshold = 0.35

[margin]
accuracy = 0.01
binary_search_steps = 7
output = "DFF_Example_margins.txt"

[parameters]
search_string = "^(B\\d+|IB\\d+|L\\d+)$"
```

Use this when you want a per-parameter robustness summary around the nominal design.

14.7.3 Yield

```toml
mode = "yield"

[verify]
method = "spec_file"
file = "DFF_Example.sp"
circuit = "DFF_Example.cir"
threshold = 0.35

[yield]
num_samples = 2000
spread = 0.10

[parameters]
search_string = "^(B\\d+|IB\\d+|L\\d+)$"
```

Use this when you want a Monte-Carlo pass-rate estimate for a chosen spread.

14.7.4 Optimize

```toml
mode = "optimize"

[verify]
method = "spec_file"
file = "DFF_Example.sp"
circuit = "DFF_Example.cir"
threshold = 0.35

[margin]
binary_search_steps = 7

[optimize]
method = "cog"
spread = 0.60
samples = 1000
max_iterations = 500
output = "DFF_Example_optimized.cir"

[parameters]
search_string = "^(B\\d+|IB\\d+|L\\d+)$"
```

Use this when you want Optimum to search automatically for a better margin center.

14.8 Configuration reference

14.8.1 `[verify]` – common to all modes

`method` string; default `"spec_file"`

Verification input format: `"spec_file"`, `"csv_file"`, or `"event"`.

`file` string; no default

Specification or CSV reference file. Required for `spec_file` and `csv_file`; not used by `event`.

`circuit` string; no default

Circuit netlist to simulate. Required.

`rules` array of tables; no default

EVENT semantic rules. Required and non-empty when `method="event"`.

`threshold` float; default 0.05

Global tolerance used by SP checks and as a base tolerance in CSV comparison.

`current_threshold` float; default 0.05

Current tolerance factor used for `L...` columns in SP-style specifications.

`amplitude_threshold` float; default 0.50

CSV pulse-detection amplitude fraction.

`slope_threshold` float; default 0.02

CSV slope threshold used when identifying stable and transition regions.

`min_pulse_width` float; default 2.5e-12

Minimum event width considered meaningful in CSV processing.

min_stable_samples int; default 3

Minimum consecutive stable samples for CSV processing.

merge_gap_threshold float; default 1e-12

Merge adjacent CSV pulses separated by less than this gap.

edge_time_tol_ps float; default 5.0

CSV edge-pairing tolerance in picoseconds.

edge_rise_tol_frac float; default 0.10

Accepted by the current parser, but not presently active in the comparison path.

edge_min_amp_frac float; default 0.20

Minimum normalized edge amplitude used by the adaptive CSV edge detector.

edge_window_size int; default 200

Window size, in samples, for adaptive CSV edge detection.

EVENT rules are configured as `[[verify.rules]]` tables. Section 14.9.3 gives the complete EVENT rule reference, including the required keys, supported logic functions, timing behavior, and configuration examples.

14.8.2 `[margin]` – robustness search

subcircuit string; no default

Limit parameter selection to this subcircuit scope. Yield and Optimize read the same scope value.

accuracy float; default 0.01

Margin search tolerance or exhaustive-search step size.

binary_search_steps int; default 10

Maximum number of binary-search refinement steps.

min_search float; default 0.10

Lowest scale factor tested on the negative side.

max_search float; default 1.90

Highest scale factor tested on the positive side.

exhaustive bool; default false

Scan the interval in fixed `accuracy` steps instead of using binary search.

pocket_search bool; default false

Emit full pass/fail pocket maps instead of only summarized margins.

output string; no default

Optional text file for the printed margin summary or pocket map.

parameter_dependence bool; default false

Re-evaluate dependent parameter expressions when a searched parameter changes.

14.8.3 `[yield]` – Monte-Carlo yield

num_samples int; default 100

Number of Monte-Carlo trials per yield estimate.

spread float; default 0.0

Global fractional spread. If left at zero, explicit parameter entries may provide spreads; otherwise Optimum falls back to 10% of each nominal value.

roll_off bool; default false

Sweep spread from `step_size` to 1.0 and report yield at each step.

step_size float; default 0.1

Roll-off sweep step.

output string; no default

Optional roll-off CSV output path. Used only for roll-off output.

14.8.4 `[optimize]` – automatic improvement search

method string; default `"cog"`

Optimization method selector. Optimum uses the center-of-gravity workflow.

spread float; default 0.50

Global sampling spread around the current center.

spread_decay float; default 0.75

Multiplicative spread reduction during stalled iterations; internally clamped.

spread_grow float; default 1.05

Multiplicative spread increase after improvement; internally clamped.

samples int; default 10000

Number of samples per optimization iteration; raised internally to at least 100.

samples_max int; default 20000

Maximum adaptive sample count when the search upsamples after stalled progress.

seed int; default 1234567

Base seed for repeatable optimization clouds.

resample_every int; default 5

Reuse a common-random-number epoch for this many iterations before reseeding.

trim_frac float; default 0.10

Two-sided trim fraction for the trimmed mean.

ema_alpha float; default 0.50

Exponential moving average blend between the current center and the new estimate.

sig_digits int; default 4

Significant digits used when writing optimized numeric `.PARAM` values.

max_iterations int; default 500

Hard iteration cap.

patience int; default 10

Number of non-improving iterations allowed before early stopping.

converge float; default -1.0

Stop when the gap to the current best margin stays within this threshold. Set `>0` to enable.

target float; default -1.0

Stop when the best critical margin reaches this target. Values `<=1.0` are interpreted as fractions and converted to percent.

conv_k int; default 4

Number of consecutive within-threshold iterations required before `converge` triggers.

min_spread float; default 0.02

Minimum allowed spread during spread decay.

output string; no default

Optional optimized circuit filename. When set, Optimum writes updated `.PARAM` values into a new netlist.

14.8.5 `[parameters]` – choosing the parameter set

Margin, Yield, and Optimize need a parameter set to work on. If `[parameters]` is omitted, Optimum falls back to all parameters in the active scope. That is valid, but it is usually slower and less informative than selecting a focused set.

Use `search_string` to select a family of parameter names:

```
[parameters]
search_string = "^(IBIAS|L_.*|C_.*)$"
```

Alternatively, enumerate parameters explicitly:

```
[parameters]
B1 = {}
IB1 = {}
L1 = {}
```

Explicit entries can provide values:

```
[parameters]
B1 = { nominal = 120e-6 }
IB1 = { nominal = 180e-6, spread = 15e-6 }
```

For these settings:

- `nominal` overrides the circuit nominal value used as the baseline for Margin, Yield, and Optimize,
- `spread` is used by Yield when drawing that parameter, and
- Optimize does not currently read per-parameter `spread`; it samples from the global `optimize.spread` around the current center.

14.9 Verification inputs and tolerances

14.9.1 `method = "spec_file"`

Use an SP-style expectation file when you want to check explicit values at specific times. The first column is time. Each remaining column names a trace and gives the expected value at that time.

Optimum interprets labels as follows:

- labels starting with `L` are treated as currents and compared using `verify.current_threshold`,
- labels starting with `B` are treated as phase outputs, with SP values interpreted in cycles and converted internally to radians, and
- any other label is treated as an amplitude-style comparison.

Example:

```
time     B1  B6  B7
2e-11    0   0   0
5e-11    0   1   0
1.5e-10  0   2   0
2e-10    1   2   0
```

14.9.2 `method = "csv_file"`

Use a CSV reference when you want Optimum to compare waveform behavior rather than exact checkpoint values. The first column is time. The remaining columns are numeric reference traces.

For CSV verification, Optimum extracts events from the reference and simulated traces, pairs them in time, and compares their behavior using the tolerance settings in `[verify]`. This is where `threshold`, `amplitude_threshold`, `slope_threshold`, `min_pulse_width`, `min_stable_samples`, `merge_gap_threshold`, and the `edge_*` settings matter most.

14.9.3 `method = "event"`

Use EVENT verification when the required behavior can be described directly from Josephson-junction phase events. Unlike SP and CSV verification, EVENT does not need a reference waveform or a table of expected event times. Instead, each `[[verify.rules]]` entry states what relationship must hold between named signals.

EVENT accepts bare junction labels such as `B1` and explicit phase expressions such as `P(B1)`. A configuration must contain at least one rule.

Rule options

The keys available inside a `[[verify.rules]]` table are:

`type` string

Rule type. One of `transfer`, `fanout`, `merge`, `delay`, `phase_shift`, `logic`, `clocked_logic`, or `clocked_transfer`.

`input` string

Single input signal used by `transfer`, `fanout`, `delay`, `phase_shift`, and `clocked_transfer`.

`inputs` string array

Multiple input signals used by `merge`, `logic`, and `clocked_logic`.

`clock` string

Clock signal used by `clocked_logic` and `clocked_transfer`.

`output` string

Single output signal used by every rule except `fanout`.

`outputs` string array

Output branches used by `fanout`. At least one output is required.

`function` string

Boolean function used by `logic` and `clocked_logic`.

Only the keys required by the selected rule are needed. Rule types, function names, and key names are written in lower case as shown below. There are no EVENT-specific timing thresholds to enter for normal use; event grouping and timing consistency use the simulation's numerical time resolution.

Supported rules

The rule set is intentionally small. Choose the rule that describes the cell behavior rather than looking for a rule named after a particular cell.

`transfer` requires `input`, `output`

One input event is reproduced at one output. Event order is preserved and propagation delay must remain consistent.

`fanout` requires `input`, `outputs`

One input event is reproduced on every listed output branch. Branch timing must remain consistent.

`merge` requires `inputs`, `output`

Events from two or more inputs converge onto one output. Inputs that arrive within the numerical time resolution are treated as one logical arrival.

`delay` requires `input`, `output`

The input is reproduced at the output with a stable delay. This has the same verification behavior as `transfer`, but identifies intentional delay as the purpose of the cell.

`phase_shift` requires `input`, `output`

The input is reproduced at the output across a clock or excitation phase boundary. No user-entered phase number or golden delay is required.

`logic` requires `function`, `inputs`, `output`

Event-triggered combinational logic. Near-coincident input events form one logical input group.

`clocked_logic` requires `function`, `inputs`, `clock`, `output`

Combinational logic evaluated at each clock event using inputs received since the previous clock. Repeated events on one input within a clock interval count as one logical input.

`clocked_transfer` requires `input`, `clock`, `output`

An input event is stored until the next clock. Several input pulses before the same clock represent one stored SFQ event and require one output event.

Logic functions

`logic` supports functions whose result is determined by an arriving event group:

`buffer`, `and`, `or`, `xor`, `majority`

`clocked_logic` supports the complete function set because the clock gives an explicit evaluation point, including intervals where no input event arrived:

`buffer`, `not`, `and`, `nand`, `or`, `nor`, `xor`, `xnor`, `majority`, `minority`

Input-count requirements are:

- `buffer` and `not`: exactly one input,
- `majority` and `minority`: an odd number of at least three inputs, and
- all other logic functions: at least two inputs.

Unclocked `logic` does not support `not`, `nand`, `nor`, `xnor`, or `minority`. Those functions can require an output when no input event occurs, so use `clocked_logic` when the result depends on an empty input interval.

Rule examples

A simple transfer check is shown below. The `delay` and `phase_shift` rules use the same `input`/`output` form with only the `type` value changed:

```
[[verify.rules]]
type = "transfer"
input = "B1"
output = "B2"
```

A fanout check uses an output array:

```toml
[[verify.rules]]
type = "fanout"
input = "B1"
outputs = ["B2", "B3", "B4"]
```

A merge check uses an input array:

```toml
[[verify.rules]]
type = "merge"
inputs = ["B1", "B2"]
output = "B3"
```

Unclocked logic adds a Boolean function:

```toml
[[verify.rules]]
type = "logic"
function = "xor"
inputs = ["B1", "B2"]
output = "B3"
```

Clocked logic adds the clock signal:

```toml
[[verify.rules]]
type = "clocked_logic"
function = "nand"
inputs = ["B1", "B2"]
clock = "B6"
output = "B7"
```

A clocked storage/transfer rule is:

```toml
[[verify.rules]]
type = "clocked_transfer"
input = "B1"
clock = "B6"
output = "B7"
```

Timing, polarity, and use with Optimum modes

EVENT verification checks event direction as well as event count and ordering. A reverse-polarity phase transition does not satisfy a positive event. Transfer-style rules establish propagation timing from the simulated behavior and require that timing to remain consistent; they do not require a golden delay value in the TOML file.

The same EVENT rules can be used with `verify`, `margin`, `yield`, and `optimize`. For margin searches, Optimum records the nominal logical behavior before parameter variation. Rule inputs must preserve their event count and polarity. For clocked rules, the clock stimulus must preserve its nominal event count, polarity, logical event slots, and cadence, and each input event must remain associated with the same capture clock as in the nominal design. A common propagation-latency shift at the observed clock junction is allowed when it does not change the clock cadence or logical slot. Configured outputs must continue to produce the nominal event cardinality, and every candidate must still satisfy the selected rule. This allows events to move within their intended logical window without requiring golden timestamps while rejecting cycle slips or changes in the applied stimulus.

EVENT currently operates on Josephson-junction phase events. Use SP or CSV verification when acceptance must be based on ordinary voltage or current levels rather than phase events.

Use `optimum-cli --event-config circuit.cir` when the available phase-signal labels are not already known. The helper lists the Josephson-junction phase signals in the circuit, asks for the rule type and required roles, and writes the corresponding `[[verify.rules]]` block. It does not infer signal roles, expected waveform timing, or expected pulse sequences.

14.9.4 Choosing tolerances

Start conservatively and tighten only after the nominal design is stable.

- For SP-style specifications, start with `threshold = 0.05` and only widen it when expected timing or amplitude differences are known and acceptable.
- For CSV verification, adjust one class of control at a time: first event detection, then edge-pairing tolerance, then overall acceptance threshold.
- If the circuit passes visually but Optimum misses an event, inspect `amplitude_threshold`, `min_pulse_width`, and `edge_min_amp_frac` before loosening the final pass/fail threshold.

14.10 Margin workflow

Margin finds the largest verified window around a scale factor of 1.0 for each selected parameter. The negative side searches `[min_search, 1.0]`; the positive side searches `[1.0, max_search]`.

Reported values are converted to percent around nominal, so a lower scale of `0.88` becomes a negative margin of 12%, while an upper scale of `1.14` becomes a positive margin of 14%.

14.10.1 Binary, exhaustive, and pocket search

- **Binary search** is the default and is the right first choice for most parameter sets.
- **Exhaustive search** steps through the interval in `accuracy` increments. Use it when you want direct control over the tested grid.
- **Pocket search** prints the pass/fail pattern across the searched interval. Use it when a circuit is not well described by one continuous margin on each side of nominal.

If parameter expressions are coupled, enable `parameter_dependence = true` so dependent parameter expressions are re-evaluated as the searched parameter moves.

14.10.2 Margin output

A margin run prints one line per selected parameter, followed by the critical and average margins:

```
IBIAS :  12.50[               ************|**************
         ]  14.00
L_IN  :   5.00[                     *****|*************
         ]  12.00
Critical margin: 5.00%   ['L_IN-']
Average margin: 10.75%
```

The critical margin is the worst side of the worst parameter. That is normally the first value to inspect.

14.11 Yield workflow

Yield draws random parameter sets from normal distributions centered on the chosen nominal values, verifies each draw, and reports the fraction that pass. Optimum uses a Gaussian model with standard deviation $\sigma = \text{spread}/3$, so the configured spread corresponds roughly to a $\pm 3\sigma$ range. Negative sampled values are clamped to zero.

14.11.1 Choosing the spread model

- Set `yield.spread` when you want a single fractional spread applied to all selected parameters.
- Leave `yield.spread = 0.0` when you want explicit parameter entries to provide spreads.
- If neither global nor per-parameter spread is supplied, Optimum falls back to 10% of each nominal parameter value.

14.11.2 Roll-off runs

Set `roll_off = true` to sweep the spread from `step_size` to `1.0`. If `yield.output` is set, Optimum writes a two-column CSV with `Spread,Yield`.

14.11.3 Yield output

A standard yield run reports the number of passing and failing samples plus the resulting percentage:

```
Passes: 1932
Fails: 68
Yield: 96.6%
```

A roll-off run produces a table or CSV such as:

```
Spread,Yield
0.10,96.0
0.20,90.5
0.30,78.0
```

Open the roll-off CSV in the GUI when you want to inspect or export it alongside other result data.

14.12 Optimize workflow

Optimize uses Margin as its score function. It begins from the nominal design, repeatedly samples around the current parameter center, keeps the passing samples, forms a new candidate center, and accepts that candidate only when the critical margin improves.

14.12.1 Current optimization method

Optimum uses a center-of-gravity style search:

1. draw a Monte-Carlo cloud around the current center,
2. keep passing samples and also test reflected versions of failing samples,
3. compute a trimmed mean of the surviving cloud,
4. smooth that candidate using `ema_alpha`,
5. re-run Margin on the new candidate, and
6. keep it only if the critical margin improves.

14.12.2 Stopping conditions

The run stops when one of the following occurs:

- `patience` is exhausted,
- `converge` remains satisfied for `conv_k` consecutive iterations,
- `target` is reached, or
- `max_iterations` is reached.

14.12.3 Optimized netlist output

When `optimize.output` is set, Optimum writes a new circuit file with numeric `.PARAM` values back-annotated into the netlist. Parameters that were directly optimized are written with the new best values. Dependent parameters that need to be locked for reproducibility are written with baseline numeric values. Other lines are left unchanged.

The optimized output is a normal `.cir` file. Open it in the GUI to inspect parameter changes, visualize the netlist, or run a standard GUI simulation. If the optimized file references include files, keep it in the same project layout expected by the original circuit.

14.13 Worked DFF example

The installed `DFF_Example_config.toml` file illustrates all Optimum configuration blocks. It defaults to Optimize, but the commented mode lines make it easy to switch to Verify, Margin, or Yield.

```
#mode="verify"
#mode="margin"
#mode="yield"
mode="optimize"

[optimize]
method = "cog"
spread = 0.6
converge = 0.01
samples = 1000
max_iterations = 5000
output = "DFF_Example_optimized.cir"
sig_digits = 4

[parameters]
search_string="^(?!(?:B0$|B0Rs$))(?:B\\d+|IB\\d+|L\\d+)$"

[margin]
binary_search_steps = 7
output = "DFF_Example_margins.txt"

[yield]
spread = 0.1
num_samples = 2000
roll_off = true
output = "DFF_Example_yield_roll_off.csv"
step_size = 0.05

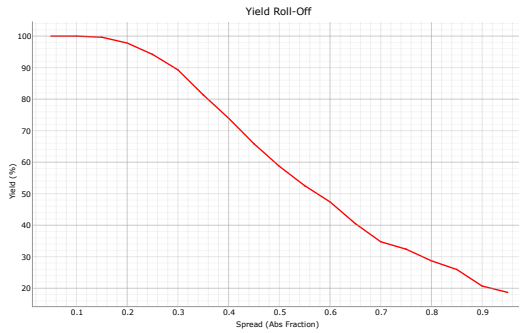
[verify]
method = "spec_file"
file = "DFF_Example.sp"
circuit = "DFF_Example.cir"
threshold=0.35
```

A margin run on this example produces output of this form:

```
Mode: Margin
B1 :  90.00[
     ************************|************************   ]
     90.00
B2 :  90.00[   ************************|**********
          ] 37.27
B3 :  49.92[              **************|***********
          ] 38.67
B4 :  47.81[               *************|*******
          ] 24.61
B5 :  47.11[               *************|****
          ] 14.06
...
Critical margin: 14.06% ['B5+']
Average margin: 72.70%
```

The critical margin tells you which parameter side fails first and therefore where the design is weakest.

A yield roll-off on the same circuit produces a spread-versus-yield curve such as the one below:



An optimization run reports progress as it searches for a better critical margin:

```
Mode: Optimize
Original critical margin: B5+ 14.0625%
Iteration   20/5000 (Best critical margin:   37.27%      B4+)
     spread=1  samples=1000000
Convergence reached
Final critical margin: B4+ 37%
```

The exact numeric result is not deterministic unless the seed is controlled. Compare the final critical margin against the original nominal margin, then inspect the written optimized circuit when `optimize.output` is enabled.

CLI Usage and Command Reference

JoSIM®-Pro can be driven from the command line through `josim-pro-cli`. This chapter documents terminal workflows for automation, scripting, CI, batch simulation, reproducibility checks, and result generation for later GUI import. Interactive editing, visualization, first simulation, trace analysis, and plot export are covered in the GUI chapters.

15.1 When to use the CLI

Use `josim-pro-cli` when one of these requirements matters more than interactive inspection:

- repeatable batch runs from a shell script, build job, CI runner, or parameter sweep;
- producing `.csv` or `.tsv`-style text data for later import into the GUI or another analysis tool;
- streaming long text-output runs instead of buffering all output before writing;
- forcing a run-local analysis mode, integration method, precision, seed, or output destination without editing the netlist;
- checking CLI-only deployment behavior on systems where no desktop session is available.

For a GUI-to-terminal mapping of the same controls, see Chapter 13.

15.2 Command structure

The basic form is:

```
josim-pro-cli [options] [input_file]
```

The input netlist may be supplied in any one of the following ways:

- as a positional `input_file`;
- with `-i` or `--input`;
- as `STDIN`, usually with `-i STDIN` or shell redirection.

If both `-i`/`--input` and a positional input file are supplied, the explicit `-i`/`--input` value is the runtime input. If the program is started with no arguments at all, it prints version and usage information and exits instead of waiting on *STDIN*.

Keep durable circuit intent in the netlist itself with commands such as `.TRAN`, `.IC`, `.OPTION`, `.FILE`, and `.IV`. Use the CLI for per-run controls such as output destination, analysis-mode override, streaming, precision, verbosity, and reproducibility.

15.3 Default behavior

Unless overridden, JoSIM®-Pro behaves as follows:

- analysis mode defaults to `PHASE`;
- integration method defaults to `BDF`;
- variable-step mode is **OFF**;
- Josephson-junction Newton solving is **ON**;
- formatted output precision defaults to `6`;
- output is written to *STDOUT* unless an output target is supplied by `-o`/`--output` or by netlist `.FILE` routing;
- if `-o`/`--output` is provided without a value, the output file defaults to `out.csv`;
- the configured license path is read from `~/.josimpro/config.ini`, or from the built-in install default when no user configuration exists.

15.4 Quick command patterns

The commands below cover the most common operating patterns. Where a concrete input is useful, they use the installed `JTL_Example.cir` file from the examples directory.

Command	Purpose
`josim-pro-cli JTL_Example.cir`	Run a circuit with the default `PHASE + BDF` configuration and write results to *STDOUT*.
`josim-pro-cli -o results.csv JTL_Example.cir`	Run a circuit and write formatted CSV output to a file.
`josim-pro-cli –output=results.csv -i STDIN`	Read a netlist from *STDIN* and write CSV output to a named file.
`josim-pro-cli -a 1 -o results.dat JTL_Example.cir`	Override the analysis domain to `VOLTAGE` and write plain-text data output.
`josim-pro-cli -p -o streamed.csv JTL_Example.cir`	Stream a long text-output run directly to disk instead of buffering the full result set first.
`josim-pro-cli –varstep -o results.csv JTL_Example.cir`	Enable the supported variable-step path for `PHASE + BDF`.
`josim-pro-cli -r 123456 -o repeatable.csv JTL_Example.cir`	Reproduce spread- and noise-related random choices with a fixed seed.
`josim-pro-cli -l /path/to/sm_license.txt`	Store a license-file path in the user CLI configuration, then exit.
`josim-pro-cli -s`	Ask the installed `smlicensed` daemon to generate `SM_SysID.txt`.
`josim-pro-cli -h`	Print the built-in help summary.

15.5 Option summary

The full option surface is easier to scan when it is grouped by purpose rather than expanded one flag at a time.

15.5.1 Input, licensing, and information

Short	Long	Value	Purpose
`-i`	`–input`	path or STDIN	Input netlist path. Use `STDIN` when piping or redirecting netlist text.
`-l`	`–license`	path	License-file path. The selected path is stored in the user configuration for future runs. When no input/output run is requested, the path is checked, saved, and the command exits.
`-s`	`–sysid`	none	Ask the installed SUN Magnetics License Manager `smlicensed` daemon to generate `SM_SysID.txt` in the current working directory for licensing support.
`-h`	`–help`	none	Print the built-in help summary.
`-v`	`–version`	none	Print product name, version, build timestamp, and git revision when available.

15.5.2 Analysis and runtime control

Short	Long	Value	Purpose
`-a`	`–analysis`	0 or 1	Select the transient analysis domain: `0` for `PHASE`, `1` for `VOLTAGE`.
`-x`	`–integration`	0 or 1	Select the integration method: `0` for `BDF`, `1` for trapezoidal/`TRAP`.
	`–varstep`	none	Enable the supported variable-step path. This is valid only for `PHASE + BDF`.
	`–no-newtonjj`	none	Disable Josephson-junction Newton solving for the run.
`-r`	`–seed`	unsigned integer	Set the random seed used for spread- and noise-related random choices. `0` means the runtime may use non-deterministic/time-based seeding.

15.5.3 Output and reporting

Short	Long	Value	Purpose
`-o`	`--output`	optional target	Select the primary output target. Use a filename or `STDOUT`. If the flag is present without a value, the target is `out.csv`; if the flag is omitted, netlist `.FILE` routing or *STDOUT* is used.
`-p`	`--stream`	none	Stream text output as the run progresses instead of buffering the full result set first.
`-c`	`--compressed`	none	Store `CSV` or `DAT` output in a GZIP container. Compression is not applied to *STDOUT*, `BIN`, or `RAW` output.
`-P`	`--precision`	0 to 17	Set the number of digits used for formatted floating-point output.
`-m`	`--minimal`	none	Suppress the standard execution banner, run summaries, progress feedback, and other non-essential terminal text.
`-V`	`--verbose`	optional 0 to 3	Increase execution feedback. Use an explicit value, for example `-V 1`. A bare `-V` is accepted but resolves to the default value `0`.

15.6 Operational notes

15.6.1 Analysis and integration overrides

`PHASE` with `BDF` is the default and is the normal starting point for superconducting transient work. Use `VOLTAGE` when a voltage-domain solution is the better match for the circuit or for the quantities you intend to inspect directly.

```
josim-pro-cli -a 0 JTL_Example.cir
josim-pro-cli --analysis 1 JTL_Example.cir
josim-pro-cli -x 0 JTL_Example.cir
josim-pro-cli --integration 1 JTL_Example.cir
```

These CLI values are run-local overrides. They should not be used as a substitute for documenting durable design assumptions in the netlist.

15.6.2 Variable-step operation

Variable-step mode is useful when fast events are localized in time and a uniform fixed-step print cadence would oversample quiet regions. Keep fixed-step mode for uniformly sampled outputs, direct point-to-point comparison against earlier runs, or any run that must remain in `VOLTAGE` mode or use trapezoidal integration.

```
josim-pro-cli --varstep JTL_Example.cir
```

Constraint: variable-step mode is supported only for `PHASE + BDF`. Unsupported combinations are rejected explicitly.

15.6.3 Output targets and file formats

The output target determines the file format:

- `.csv` produces comma-separated text output;
- `.dat` produces space-separated text output;
- `.bin` produces binary output;
- `.raw` or an extensionless target produces raw output in buffered mode;
- `STDOUT` writes text output to the terminal using space-separated columns.

Unsupported extensions are normalized to CSV and the target extension is replaced with `.csv`. A `.gz` suffix is treated as a compression suffix: the logical format still comes from the preceding extension, for example `results.csv.gz` is CSV output.

Because `-o`/`--output` takes an optional value, prefer one of these unambiguous forms when piping from *STDIN*:

```
josim-pro-cli --output=results.csv -i STDIN
josim-pro-cli -i STDIN -o results.csv
```

A command such as `josim-pro-cli -o results.csv` is ambiguous to humans and is not a good pattern: use `--output=results.csv -i STDIN` when there is no positional input file.

15.6.4 Netlist `.FILE` routing

A netlist `.FILE` directive routes subsequent `.PRINT`, `.SAVE`, or `.PLOT` requests into the selected target. If no CLI `-o`/`--output` target is supplied, `.FILE` targets can define the native output plan by themselves.

Relative `.FILE` paths are resolved near the source that contains the directive, then against the input-file directory, then against the process working directory. This is important for included files: a `.FILE` inside an included file may route output relative to that included file's directory.

```
.FILE waveforms.csv
.PRINT V(q)
.FILE derived.dat
.PRINT P(j1)
```

Place `.FILE` before the output requests it is meant to route. It does not retroactively move earlier output requests.

15.6.5 Streaming output

Streaming is intended for `CSV`, `DAT`, and `STDOUT` workflows:

```
josim-pro-cli -p -o results.csv JTL_Example.cir
```

When streaming is enabled, selecting `.bin`, `.raw`, an extensionless raw target, or another unsupported extension falls back to streamed CSV output and emits a warning when fallback warnings are enabled. Use buffered output if a downstream tool specifically requires `BIN` or `RAW`.

15.6.6 Compression

Compressed output is valid for file-backed `CSV` and `DAT` output. When compression is enabled, JoSIM®-Pro ensures the written target has a `.gz` suffix.

```
josim-pro-cli -c -o results.csv JTL_Example.cir
josim-pro-cli --compressed --output=results.dat JTL_Example.cir
```

Do not request compression for `BIN` or `RAW`. Compression is ignored for *STDOUT*.

15.6.7 Reproducibility and randomness

Use a fixed seed when you need spread- and noise-related random decisions to be repeatable:

```
josim-pro-cli -r 123456 JTL_Example.cir
```

This is useful for side-by-side comparisons, regression tests, CI baselines, and Optimum-adjacent workflows where you want nominally identical random choices across repeated runs.

15.6.8 Saved testbenches and CI verification

A saved testbench checks existing `.MEASURE TRAN` results against electrical requirements. Save this definition as `bench.json` beside the input netlist:

```
{
  "schema": "josim-testbench/1",
  "name": "Bias current",
  "input": "bias.cir",
  "requirements": [{
    "id": "bias-current-range",
    "measurement": "bias_current",
    "unit": "A",
    "min": 0.00099,
    "max": 0.00101
  }]
}
```

For example, `bias.cir` can contain:

```
.OPTION ANALYSIS=1 SEED=321
I1 0 N DC 1m
R1 N 0 1
.TRAN 1p 10p
.PRINT V(N)
.MEASURE TRAN bias_current FIND I(I1) AT=5p
.END
```

Run it with:

```
josim-pro-cli --testbench bench.json --results run-001
```

The parent of `run-001` must exist; `run-001` itself must be new. Use a different directory for each generation. The input path resolves beside the JSON definition, independently of the shell's working directory. Numerical CLI overrides, such as `-a`, `-x`, and `-r`, retain their usual precedence. Omit positional input, `-i`, `-o`, and parallel-worker options.

Each definition has 1–256 requirements with unique IDs. Measurement names are case-insensitive; units must match the measurement's reported unit exactly. Limits are finite JSON numbers in SI units; strings such as `"1m"` are rejected. Use inclusive `min`/`max` bounds, including a single bound, or a `target` with explicit nonnegative `absolute_tolerance` and/or `relative_tolerance`:

$$|m - t| \le a + r|t|,$$

where m is the measured value, t the target, a the absolute tolerance, and r the relative tolerance. An omitted tolerance is zero. There is no implicit tolerance or unit conversion. A target cannot be combined with bounds. Unknown fields are rejected.

The first contract supports serial, buffered transient jobs, including existing `.STEP` cases. Every requirement applies to every case. The testbench owns waveform destinations under `waveforms/`; remove `.FILE` directives and disable streaming. Operating-point, DC, IV, and AC jobs are outside this contract. GUI review and reference-waveform comparison are not yet provided.

Exit code	Run status	Meaning
0	pass	Every requirement passes in every case.
1	execution_failed	Input, numerical execution, configuration, or output publication failed.
2	fail	At least one measured value violates its specification.
3	failed_to_evaluate	Missing measurement, unit mismatch, or measurement-evaluation failure.
130	cancelled	Cancellation was observed.

Specification failures do not stop remaining cases. Execution and measurement-evaluation failures retain the existing analysis-job stop policy; subsequent cases are `not_run`. Each check retains its own state, so a cancelled or unexecuted case cannot count as a pass. Cancellation and execu-

tion failure take precedence over specification outcomes; failed evaluation takes precedence over a specification failure.

The directory contains a definition snapshot, the normalized expanded netlist, waveform files, and versioned `josim-testbench-results/1` JSON and CSV results. JSON records source/include byte hashes, model-declaration hashes and origins, the expanded-netlist hash, effective settings, actual case seeds and sweep assignments, core version/Git/binary identity, diagnostics, and file commit flags/hashes. CSV has one row per requirement and case, including overall status, process exit code, parameters, artifact links, and a link to JSON. Preflight failures have a diagnostic row without a case.

`results.json` is published last and marks a complete generation; an interrupted or failed publication may leave a partial directory without that marker. Invalid definitions and failure to publish return code 1 without a completed report. Source hashes describe the capture before execution; the expanded snapshot identifies the logical input used. Reusing an existing results directory is rejected. Saved results are not a database or a promise of identical waveform bytes across numerical methods.

15.6.9 Licensing commands

Generate a system identifier from the terminal with:

```
josim-pro-cli -s
```

This delegates to the installed SUN Magnetics License Manager daemon, `smlicensed`. If the daemon is missing or outdated, `SM_SysID.txt` generation can fail even though the CLI executable itself is present.

To store a license path for future CLI runs, use `-l`/`--license` by itself:

```
josim-pro-cli -l /path/to/sm_license.txt
```

Do not combine that configuration-only form with `-v`; the version option exits before a simulation or license-path update is performed.

15.7 Common tasks

15.7.1 Run a normal fixed-step simulation

```
josim-pro-cli -o results.csv JTL_Example.cir
```

15.7.2 Run a voltage-domain simulation

```
josim-pro-cli -a 1 -P 12 -o results.dat JTL_Example.cir
```

15.7.3 Run a variable-step PHASE + BDF simulation

```
josim-pro-cli --varstep -o results.csv JTL_Example.cir
```

15.7.4 Stream a long run directly to disk

```
josim-pro-cli -p -o streamed.csv JTL_Example.cir
```

15.7.5 Read from *STDIN* and write to *STDOUT*

```
josim-pro-cli -i STDIN < JTL_Example.cir
```

15.7.6 Generate a CSV file for GUI import

```
josim-pro-cli -o results.csv design.cir
```

Then open `results.csv` in the GUI with `File > Open...`, drag it into the main window, or select it from a folder opened with `File > Open Folder...`.

For interactive editing, simulation and plot export, see the GUI chapters.

Netlist Syntax Reference

This chapter is the quick-reference guide to the JoSIM®-Pro netlist language. Use it when you need the required form of an element, source, model, or control command, or when you want to confirm which optional attributes are accepted on a given line.

For workflow guidance such as when to choose phase or voltage operation, when variable-step mode is appropriate, or how streamed output behaves, see the Features chapter. For command-line overrides and concrete execution examples, see the Usage chapter.

16.1 Netlist anatomy

A typical JoSIM®-Pro netlist contains four kinds of content:

- element and source lines,
- model definitions,
- control commands, and
- the terminating `.END` statement.

A compact example is:

```
* Example Josephson junction circuit
.PARAM IBIAS=0.18m
B1 1 2 JJMOD area=1.2 spread=0.1 temp=4.2
L1 1 0 1e-9 spread=0.05
R1 2 0 50 spread=0.1 temp=4.2 neb=10GHz
I1 2 0 DC IBIAS
.MODEL JJMOD JJ(RN=10, CAP=1e-12)
.PRINT V(1) I(R1)
.TRAN 0.1ps 1ns
.END
```

Read the file from top to bottom:

1. declare any parameters and included files,
2. define the circuit elements and sources,
3. define any required models and output requests,
4. add the transient-analysis command and any runtime options, and
5. end the file with `.END`.

16.2 Quick syntax index

Use the tables below when you need to locate the right family of statement quickly before reading the detailed notes and examples later in the chapter.

16.2.1 Elements and models

Prefix / command	Basic form	Purpose
`R`	`Rname n+ n- value`	Resistor, with optional `spread=`, `temp=`, and `neb=` attributes.
`L`	`Lname n+ n- value`	Inductor, with optional `spread=`.
`C`	`Cname n+ n- value`	Capacitor, with optional `spread=`.
`D`	`Dname n+ n- model [AREA=value]`	Two-terminal diode current and charge.
`Y`	`Yname n+ n- model`	Cubic polynomial current and charge.
`B`	`Bname n+ n- model ...`	Josephson junction instance using a `JJ`, `RCSJ`, or `TJM` model.
`T`	`Tname n1+ n1- n2+ n2- ...`	Transmission line using the ideal `LOSSLESS`, finite `LCLADDER`, differential `RLGC`, frequency-dependent `WIDEBAND`, or two-mode `MODAL` model.
`@MTL`	`@MTL name P1=(...) P2=(...) ...`	JoSIM-Pro multiconductor RLGC matrix line with an explicit reference node.
`K`	`Kname L1 L2 k`	Mutual inductive coupling.
`.MODEL`	`.MODEL name type(parameters)`	Junction, line, `D`, or `POLY` model definition.

16.2.2 Diode and polynomial current/charge models

`Dname p n model [AREA=value]` selects a diode. Positive current flows from `p` to `n`; $v = V(p) - V(n)$. Its terminal equation and small-signal admittance are

$$i = I(v) + \frac{dQ(v)}{dt}, \qquad Y(\omega) = \frac{dI}{dv} + j\omega\frac{dQ}{dv}.$$

The diode uses $I(v) = I_S \operatorname{expm1}(v/(Nk_BT/q))$, SPICE depletion charge and `TT` diffusion charge. `AREA` scales current and charge and defaults to 1.

```
.MODEL DM D(IS=1p N=1 T=300.15 CJO=2p VJ=.7 M=.5 FC=.5 TT=1n)
V1 IN 0 DC .8
R1 IN OUT 1k
D1 OUT 0 DM AREA=1
.OP
```

Field	Default	Meaning and domain
IS	10^{-14} A	Saturation current at the specified temperature; positive.
N	1	Emission coefficient; positive.
T	300.15 K	Model temperature in kelvin; positive.
CJO	0 F	Zero-bias depletion capacitance; nonnegative.
VJ	1 V	Junction potential; positive.
M	0.5	Grading coefficient, $0 \leq M \leq 1$.
FC	0.5	Forward capacitance coefficient, $0 \leq FC < 1$.
TT	0 s	Diffusion transit time; nonnegative.

IS is supplied at T; no nominal-temperature rescaling is performed. If .TEMP is present, it must match the model's T. Series resistance, reverse breakdown, semiconductor noise and self heating are outside this model. Unknown fields are errors. These equations do not establish measured-device or cryogenic validity.

Yname p n model selects a native POLY model:

$$I(v) = \sum_{r=0}^{3} I_r v^r, \qquad Q(v) = \sum_{r=0}^{3} Q_r v^r.$$

Fields I0–I3 and Q0–Q3 default to zero and have units A/V^r and C/V^r. Coefficients must be finite; negative values are permitted without a passivity or uniqueness guarantee. B remains the Josephson designator.

```
.MODEL PM POLY(I1=1m I3=2m Q0=7p Q1=2p Q3=3p)
Y1 OUT 0 PM
V1 OUT 0 PWL(0 0 1n .4)
.OPTION ANALYSIS=VOLTAGE INTEGRATION=BDF
.TRAN 10p 1n 0 10p
.PRINT V(OUT) I(Y1)
```

Both families support .OP, fresh-point .DC and bias-linearized .AC, with current and voltage outputs through the shared analysis interfaces. Transients require fixed-step VOLTAGE/BDF; charge history is committed only after acceptance. PHASE, TRAP, adaptive stepping, mixed Josephson transients and checkpoint pause/resume are unsupported for these devices. A failed nonlinear solve publishes no partial analysis file. NEWTONMAXITERS and NEWTONDAMPING control iteration; diode Newton voltage corrections are bounded to $4Nk_BT/q$ without clipping the model voltage or changing its equations. Ordinary AC requires a valid selected equilibrium.

16.2.3 Sources and controlled sources

Prefix	Basic form	Purpose
V	Vname n+ n- source type	Independent voltage source.
I	Iname n+ n- source type	Independent current source.
P	Pname n+ n- source type	Independent phase source.
F	Fname n+ n- cp cn gain	Current-controlled current source.
H	Hname n+ n- cp cn r	Current-controlled voltage source.
G	Gname n+ n- cp cn gm	Voltage-controlled current source.
E	Ename n+ n- cp cn gain	Voltage-controlled voltage source.

16.2.4 Control commands

Command	Basic form	Purpose
.OP	.OP	Solve a standalone static operating point and report scalar quantities.
.DC	.DC source start stop increment	Sweep one DC current/voltage source through static equilibria.
.AC	.AC LIN|DEC|OCT count start stop	Solve a standalone small-signal frequency sweep.
.TRAN	.TRAN *tstep tstop* ...	Define the transient run window, print cadence, and startup flags.
.MEASURE	.MEASURE TRAN name ...	Evaluate a named transient statistic, point or event.
.STEP	.STEP PARAM name LIST ...	Run independent parameter combinations.
.IC	.IC target=value	Apply explicit initial conditions to nodes or devices.
.PARAM	.PARAM *name=expr*	Define reusable parameter expressions.
.SUBCKT / X	.SUBCKT ... / X...	Define and instantiate hierarchical subcircuits.
.INCLUDE	.INCLUDE "file"	Include another netlist file.
.TEMP / .NEB	.TEMP ... / .NEB ...	Set global thermal-noise temperature and bandwidth.
.SPREAD	.SPREAD ...	Set global or family-specific spread controls.
.PRINT / .SAVE / .PLOT	.PRINT ...	Request outputs and derived expressions.
.IV	.IV *model* [...]	Generate an IV sweep for a model.
.FILE	.FILE *path*	Route subsequent outputs to a named file.
.OPTION	.OPTION key=value ...	Set runtime, analysis, output, and adaptive-step controls in the netlist.
.END	.END	End the netlist.

16.3 Detailed reference

The remainder of this chapter provides the detailed forms, notes, and examples for each supported netlist statement.

16.4 Element reference

16.4.1 Resistor

Syntax:

```
Rname node1 node2 resistance [spread=spread] [temp=temp] [neb=
    freq]
```

Example:

```
R1 1 0 50 spread=0.1 temp=4.2 neb=10GHz
```

16.4.2 Inductor

Syntax:

```
Lname node1 node2 inductance [spread=spread]
```

Example:

```
L1 1 0 1e-9 spread=0.05
```

16.4.3 Capacitor

Syntax:

```
Cname node1 node2 capacitance [spread=spread]
```

Example:

```
C1 1 0 1e-12 spread=0.03
```

16.4.4 Josephson junction

Syntax:

```
Bname node1 node2 model [area=area] [spread=spread] [ic=ic] [
    temp=temp] [neb=freq] [icnoise=ICNOISE(...)]
```

Example:

```
B1 1 2 JJMOD area=1.2 spread=0.1 temp=4.2 icnoise=ICNOISE(0.01 1
    0 1p 10n 64)
.MODEL JJMOD JJ(RN=10, CAP=1e-12)
```

JoSIM®-Pro supports both `JJ`/`RCSJ` and `TJM` model definitions for Josephson junctions.

RCSJ/JJ Model Parameters

Parameter	Default	Description
VG or VGAP	2.8E-3	Junction gap voltage
IC or ICRIT	1E-3	Junction critical current before area scaling
RN	5	Junction normal resistance
R0	30	Junction subgap resistance
C or CAP	2.5E-12	Nonnegative junction capacitance; zero gives the resistively shunted limit
T	4.2	Junction temperature in Kelvin; enables temperature-dependent RCSJ behaviour
TC	9.1	Critical temperature of the material; enables temperature-dependent RCSJ behaviour
RTYPE	1	Selects how the subgap branch is handled
DELV	0.1E-3	Transition voltage between subgap and normal regions
D	0.0	Transparency parameter affecting the phase relationship
ICFCT or ICFACT	$\pi/4$	Ratio of critical current to the transition step height
PHI	0	Phase offset, including π-junction style modelling
CPR	{1}	Harmonic amplitudes for the current-phase relationship

TJM Model Syntax

```
.MODEL KIT TJM()
.MODEL KIT_EXPLICIT TJM(FIT="builtin:kit2024-v1")
.MODEL TJ1 TJM(VG=2.8m,IC=1m,RN=5,CAP=2.5p,FIT="legacy.txt")
.MODEL TJ2 TJM(RN=5,CAP=200f,FIT="real-kernel.json")
```

Omitting `FIT` selects the embedded `builtin:kit2024-v1` reference. It requires no kernel file. This versioned, 215-row model uses measured quasiparticle transport from Adam, Enss and Kempf (2024), Figure 3, DOI 10.1088/1361-6668/ad59cf, with an assumed BCS pair response at 4.2 K. Its default `RN` is 305.7863344021282 Ω and default `CAP` is 68.38738657266714 fF; explicit overrides are accepted. The derived `IC` is approximately 5.390520 μA. The gap knee is calibrated to the positive running branch; capacitance, pair spectrum and unmeasured spectral tails remain assumptions. It is a research reference, with open low-current and polarity limits, rather than a qualified

fabrication-process or switching model. Built-in IDs are case-insensitive; unknown IDs are rejected.

FIT paths are relative to the including netlist. Legacy numeric files contain one scalar per nonempty line, with at least 80 values in the AP/QP/AQ/QQ layout. The corresponding inline FIT={...} form is also accepted. Legacy coefficients retain their numerical meaning; nominal model IC does not renormalize their memory current when instance AREA is explicit.

Real microscopic kernels. A JSON file with "format":"josim-tjm-real/1" selects shared-pole real pair/quasiparticle kernels. The following is a minimal analytic example, not a material calibration:

```
{
  "format": "josim-tjm-real/1",
  "temperature_K": 4.2,
  "gap1_meV": 1.4,
  "gap2_meV": 1.4,
  "pair_suppression": 0.7,
  "fmax_Hz": 1e11,
  "source": "Analytic single-pole example",
  "material": "synthetic",
  "coefficients": [[-1, 0, 1, 0, 0, 0]]
}
```

All fields are required. Use one or more coefficient rows within the 65,536-byte FIT file limit, each containing Re(p), Im(p), Re(A), Im(A), Re(B), Im(B). With $\tau = \omega_g t$ and $\omega_g = eV_G/\hbar$, the dimensionless kernels are $k_p(\tau) = \mathrm{Re}\sum_n A_n e^{p_n\tau}$ and $k_q(\tau) = \mathrm{Re}\sum_n B_n e^{p_n\tau}$. Every pole must satisfy $\mathrm{Re}(p_n) < 0$. B_n describes the quasiparticle kernel with its normal-current term removed; JoSIM®-Pro adds V/R_N and $C\,dV/dt$ once each. Pair suppression $0 < \alpha \le 1$ multiplies A_n only.

The gap voltage and nominal critical current are derived:

$$V_G = (\Delta_1 + \Delta_2) \times 10^{-3}\ \mathrm{V},$$
$$I_{c,\mathrm{fit}} = \frac{V_G}{R_N}\left[-\sum_n \mathrm{Re}\,\frac{\alpha A_n}{p_n}\right],$$

where Δ_1, Δ_2 are the numeric gap values in meV. Omit model IC; an explicit model VG must match the gaps. RN, CAP and the derived critical current must be finite and positive. Choose either instance AREA or instance IC, which scales area relative to $I_{c,\mathrm{fit}}$. Area/spread scale critical current and capacitance together and resistance inversely. Quiescent prehistory, including an explicit .IC phase, initializes the real memory at equilibrium.

The fitted temperature is fixed. A requested instance/global TEMP must match temperature_K; use instance NEB=0 when supplying TEMP. Real-kernel fits support deterministic, fixed-step PHASE/BDF transients, stationary .OP/.DC equilibria and small-signal .AC within the declared fit band. Stationary analysis is independent of the transient integration method. Adaptive transients, ICNOISE and thermal noise are unavailable.

fmax_Hz records the declared fit band; transient inputs are not automatically band-limited. Stable poles and finite coefficients do not establish passivity, material accuracy or accuracy outside that band. Supply source and material identifiers with the fit. The JSON file is limited to 65,536 bytes; each identifier requires 1–1,024 UTF-8 bytes without control characters.

Area and Critical-Current Handling

If area is omitted but ic= is provided on the instance, JoSIM®-Pro derives the effective area from the requested critical current and the model's nominal IC. Instance area then scales the model quantities in the expected direction:

- IC and CAP scale with area,
- RN scales inversely with area,
- R0 either follows RN or scales inversely with area depending on RTYPE.

16.4.5 Transmission line

A transmission line connects port p1/n1 to port p2/n2. The TYPE attribute selects the model. Omitting TYPE selects LOSSLESS, preserving existing untyped T elements. New netlists should use the explicit TYPE= form. The legacy standalone LOSSLESS token remains accepted.

Scalar and two-mode line syntax

```
Tname p1 n1 p2 n2 TD=<time> Z0=<impedance>
Tname p1 n1 p2 n2 TD=<time> Z0=<impedance> TYPE=LOSSLESS
Tname p1 n1 p2 n2 TD=<time> Z0=<impedance> TYPE=LCLADDER NSEG=<integer>
Tname p1 n1 p2 n2 TYPE=RLGC RPERL=<ohm_per_length> \
      LPERL=<henry_per_length> GPERL=<siemens_per_length> \
      CPERL=<farad_per_length> LEN=<length> NSEG=<integer>
Tname p1 n1 p2 n2 TYPE=WIDEBAND MODEL=<model_name> \
      LEN=<length> NSEG=<integer>
Tname p1 n1 p2 n2 TYPE=MODAL ZDM=<impedance> TDDM=<time> \
      ZCM=<impedance> TDCM=<time> NSEG=<integer>
Tname p1 n1 p2 n2 <model-attributes> ICV=<voltage-state> ICI=<current-state>
```

Model	Required attributes	Use
LOSSLESS	TD, Z0	Ideal differential delay line. This is the default when TYPE is omitted.
LCLADDER	TD, Z0, NSEG	Finite lossless differential LC ladder. It models spatial dispersion and energy storage but no common-mode current path.
RLGC	RPERL, LPERL, GPERL, CPERL, LEN, NSEG	Uniform differential line with constant per-unit-length loss and storage values.
WIDEBAND	MODEL, LEN, NSEG	Uniform differential line with a named passive real-pole loss model.
MODAL	ZDM, TDDM, ZCM, TDCM, NSEG	Independent lossless differential and common-mode propagation for one conductor pair.

All required numeric values may be parameter expressions. Values must be finite and physically valid. NSEG must resolve to a positive integer and cannot change during parameter stepping because it controls generated topology.

Common attributes

Attribute	Applies to	Description
TD	LOSSLESS, LCLADDER	One-way propagation delay. Must be greater than zero.
Z0	LOSSLESS, LCLADDER	Differential characteristic impedance. Must be greater than zero.
RPERL	RLGC	Series resistance per chosen length unit. Must be greater than or equal to zero.
LPERL	RLGC	Series inductance per chosen length unit. Must be greater than zero.
GPERL	RLGC	Shunt conductance per chosen length unit. Must be greater than or equal to zero.
CPERL	RLGC	Shunt capacitance per chosen length unit. Must be greater than zero.
LEN	RLGC, WIDEBAND	Physical length in the same length unit used by the model values. Must be greater than zero.
MODEL	WIDEBAND	Name of a .MODEL ... WIDEBAND(...) definition.
ZDM, TDDM	MODAL	Differential-mode impedance and one-way delay.
ZCM, TDCM	MODAL	Common-mode impedance and one-way delay.
NSEG	Finite models	Number of generated sections. Required for LCLADDER, RLGC, WIDEBAND, and MODAL; invalid for LOSSLESS.
ICV, ICI	See initial state below	Paired explicit voltage and current state. Both must be present or both omitted.

Wideband model definition

```
.MODEL model_name WIDEBAND(RPERL=<ohm_per_length> \
      LPERL=<henry_per_length> GPERL=<siemens_per_length> \
      CPERL=<farad_per_length> RFREQ=(<frequency_hz> ...) \
      RDELTA=(<ohm_per_length> ...) GFREQ=(<frequency_hz> ...) \
      GDELTA=(<siemens_per_length> ...) FMAX=<frequency_hz>)
```

RPERL, LPERL, GPERL, and CPERL define the constant base model. RFREQ/RDELTA add frequency-dependent series-loss terms, and GFREQ/GDELTA add frequency-dependent shunt-loss terms. Each frequency list must have the same number of entries as its matching increment list. Frequencies must be strictly increasing and positive. Increments must be nonnegative. Each list may contain at most 64 entries.

FMAX declares the highest frequency for which the supplied fit is intended. It does not filter the simulated waveform. Results above FMAX are outside the declared model range; standalone .AC rejects a sweep containing any such frequency. Omitting both dynamic term pairs reduces the model to the constant RLGC case.

JoSIM-Pro accepts an already fitted passive real-pole model. It does not import sampled network data, perform vector fitting, repair non-passive coefficients, or support complex poles, arbitrary rational multiports, or frequency-dependent multiconductor matrices.

Initial state and startup

The default LOSSLESS DC solution uses the zero-frequency differential relation

$$V(p1) - V(n1) = V(p2) - V(n2), \qquad I_1 = -I_2,$$

where both currents are defined as entering the line. This does not constrain common-mode voltage or create a common-mode current path. The solved state seeds transient history, so a line under constant DC bias does not begin with an artificial empty-line fill transient.

UIC skips the DC solve. STARTUP uses the existing startup procedure instead of a computed DC operating point. Use either option when an empty or externally initialized line is intended.

A scalar finite line or LOSSLESS line may declare a uniform state with paired ICV and ICI values. LOSSLESS, LCLADDER, and RLGC use one differential voltage and one current. MODAL uses two values in each vector:

```
T1 p1 n1 p2 n2 TYPE=MODAL ... ICV=(Vd Vc) ICI=(Id Ic)
```

Positive declared current flows from port 1 toward port 2. Explicit line state is supported with ANALYSIS=VOLTAGE. WIDEBAND does not accept ICV/ICI, because voltage and current alone do not define its internal relaxation state.

In fixed-step LOSSLESS runs, JoSIM-Pro inserts solve points for delayed discontinuities in addition to the nominal .TRAN grid. This improves edge timing but does not replace timestep-convergence checks. LOSSLESS remains fixed-step only.

Multiconductor RLGC matrix line

The JoSIM-Pro extension element @MTL represents a uniform coupled line with an explicit reference conductor.

```
@MTL MTL1 P1=(p1 p2 ... pM) P2=(q1 q2 ... qM) \
      REF=0 MODEL=MTLMODEL LEN=<length> NSEG=<integer> \
      [ICV=(v1 v2 ... vM) ICI=(i1 i2 ... iM)]

.MODEL MTLMODEL MTL(NCOND=M \
      R=(r11 r12 ... rMM) \
      L=(l11 l12 ... lMM) \
      G=(g11 g12 ... gMM) \
      C=(c11 c12 ... cMM))
```

P1 and P2 must contain exactly NCOND nodes. Their order defines the conductor order at both ports, in every matrix, and in current selectors. REF is the shared reference conductor. LEN is in metres. Matrix values are row-major and use SI units per metre:

- R: ohms per metre,
- L: henries per metre,
- G: siemens per metre,
- C: farads per metre.

Each matrix must contain exactly M^2 finite values. R and G must be symmetric positive semidefinite. L and C must be symmetric positive definite. Invalid dimensions, nonsymmetric data, and non-passive data are rejected rather than repaired.

Optional paired ICV and ICI vectors define a uniform conductor-domain initial state. Each vector must contain exactly NCOND values. Voltages are relative to REF; positive currents flow from P1 toward P2. The same ANALYSIS=VOLTAGE restriction used by scalar explicit line state applies.

Conductor currents are selected explicitly:

```
.PRINT I(MTL1[1])       * P1 conductor 1, entering the line
.PRINT I(MTL1[M])       * P1 conductor M, entering the line
.PRINT I(MTL1[M+1])     * P2 conductor 1, entering the line
.PRINT I(MTL1[2*M])     * P2 conductor M, entering the line
```

Use an integer in an actual output request. Bare I(MTL1) is rejected because it does not identify a conductor. Aggregate POWER and ENERGY selectors are not defined for @MTL.

Outputs

For a scalar or two-mode T line, I(Tname) is the current entering terminal p1, and V(Tname) is V(p1)-V(n1). Either port voltage can also be requested from its external nodes, for example V(p1,n1) or V(p2,n2). Generated ladder elements, transforms, and internal nodes are hidden from normal .PRINT ALL and .SAVE ALL output.

Solver support

Model	Fixed step	Variable step
LOSSLESS	Supported	Not supported.
LCLADDER	Supported	PHASE + BDF.
RLGC	Supported	PHASE + BDF.
WIDEBAND	BDF only	PHASE + BDF.
MODAL	Supported	PHASE + BDF.
@MTL	Supported	PHASE + BDF.

Variable-step operation also requires the normal global variable-step settings. Use a finite STEPMAXDT; an unbounded maximum timestep can under-resolve a finite line even when the adaptive error test passes.

Choosing NSEG and timestep

A finite ladder is dispersive, so delay error increases with frequency. For a lossless line with $N =$ NSEG and one-way delay T_D, the approximate ladder cutoff is

$$f_c = \frac{N}{\pi T_D}.$$

For angular frequency ω below cutoff, the approximate relative phase-delay and group-delay errors are

$$\epsilon_{\text{phase}} = \frac{\arcsin(x)}{x} - 1,$$
$$\epsilon_{\text{group}} = \frac{1}{\sqrt{1-x^2}} - 1,$$
$$x = \frac{\omega T_D}{2N}.$$

Use the larger section count required by the phase-delay and group-delay limits. For MODAL, apply the estimate to both modal delays and use one NSEG that satisfies both modes.

For RLGC and WIDEBAND, an initial lossless transit estimate is

$$T_D \approx \text{LEN}\sqrt{\text{LPERL CPERL}}.$$

Loss changes attenuation and phase, so verify results at the frequencies of interest and repeat the simulation with a larger NSEG. For @MTL, the fastest supported mode controls spatial and timestep resolution.

Use at least ten timesteps per section as an initial limit:

$$\Delta t \le \frac{T_{\text{section}}}{10}.$$

For LCLADDER, $T_{\text{section}} = T_D/NSEG$. For RLGC and WIDEBAND, use the lossless transit estimate divided by NSEG. For MODAL, use the shorter modal delay divided by NSEG. JoSIM-Pro warns when the configured fixed timestep or STEPMAXDT exceeds its estimate, but it does not change the timestep or section count.

Choose NSEG from the required bandwidth and delay-error target, then check timestep convergence and section-count convergence separately. Larger NSEG values increase matrix size, runtime, and memory use.

Model limits

- LOSSLESS is an ideal differential delay line and has no physical loss or common-mode path.
- LCLADDER is a finite lossless differential model and has no common-mode path.
- RLGC uses constant scalar per-unit-length values. It does not model skin effect, dielectric dispersion, geometry, or coupling to other lines.
- WIDEBAND supports fitted scalar real-pole loss terms only. It does not fit data or represent a frequency-dependent multiconductor matrix.
- MODAL represents two independent constant lossless modes. It does not model modal loss or mode conversion along the line.
- @MTL represents a uniform coupled line with constant RLGC matrices and an explicit reference conductor.

Examples

```
.PARAM ZLINE=4
.PARAM TLINE=2p
V1 source 0 PULSE(0 1m 1p 0.1p 0.1p 4p 20p)
RSOURCE source 1 ZLINE
T1 1 0 2 0 TD=TLINE Z0=ZLINE TYPE=LCLADDER NSEG=20
RLOAD 2 0 ZLINE
```

```
T1 1 0 2 0 TYPE=RLGC RPERL=0.8 LPERL=8p \
   GPERL=0.05 CPERL=0.5p LEN=1 NSEG=80
```

```
T1 p1 n1 p2 n2 TYPE=MODAL \
   ZDM=4 TDDM=2p ZCM=10 TDCM=6p NSEG=80
```

16.4.6 Mutual inductance

Syntax:

```
Kname inductor1 inductor2 coupling_factor
```

Example:

```
K1 L1 L2 0.99
```

Description: The signed coupling factor satisfies $-1 \leq k \leq 1$, with mutual inductance $M_{ij} = k_{ij}\sqrt{L_i L_j}$. Coupled inductances must be finite and nonnegative. Repeated declarations for the same pair add their mutual inductances.

Each connected group must be passive: its symmetric inductance matrix $\mathbf{L}$ must satisfy $\frac{1}{2}\mathbf{i}^T\mathbf{L}\mathbf{i} \geq 0$ for every current vector $\mathbf{i}$. Pairwise bounds alone do not guarantee this condition. Semidefinite groups, including consistent perfect coupling, are accepted with a scale-aware floating-point tolerance. Invalid groups report their inductors and coupling source locations. The same validation applies when parameters are updated.

16.5 Independent source reference

16.5.1 Voltage source

Syntax:

```
Vname node1 node2 source_type
```

Example:

```
V1 1 0 DC 1.0
```

16.5.2 Current source

Syntax:

```
Iname node1 node2 source_type
```

Example:

```
I1 1 0 PWL(0 0 1n 1)
```

16.5.3 Phase source

Syntax:

```
Pname node1 node2 source_type
```

Example:

```
.param TWO_PI=2*PI
P1 1 0 SIN(0 TWO_PI 1GHz)
```

16.5.4 Waveform and source functions

JoSIM®-Pro supports various types of functions that can be used as sources in circuits. Below are the waveform forms supported directly by the code.

Piece-wise Linear (PWL)

Generates a signal by linearly interpolating between the provided time–value pairs.

Syntax:

```
PWL(T1 A1 T2 A2 T3 A3 ... Tn An)
PWL(0 0 1n 1 2n 0)
```

Pulse

Generates a single or repeating pulse train between two amplitudes with a specified delay, rise time, fall time, pulse width, and period.

Syntax:

```
PULSE(A1 A2 [Td [Tr [Tf [Pw [Per]]]]])
PULSE(0 1 1n 0.1n 0.1n 2n 5n)
```

- A1 - Initial amplitude
- A2 - Final amplitude
- Td - Time delay before the first pulse

Default: 0
- Tr - Rise time
  Default: Tstep
- Tf - Fall time
  Default: Tstep
- Pw - Pulse width
  Default: Tstop
- Per - Period
  Default: Tstop

Sinusoid

Creates a sinusoidal waveform with offset, amplitude, frequency, delay, and exponential damping.

Syntax:

```
SIN(A0 A [f [Td [theta]]])
SIN(0 1 1GHz 0 0)
```

- A0 - Offset
- A - Amplitude
- f - Frequency
  Default: 1/TStep
- Td - Time delay
  Default: 0
- theta - Exponential damping factor
  Default: 0

The resulting waveform is:

$$f(t) = A_0 + A\sin\left(2\pi f(t - T_d)\right) e^{-\theta(t-T_d)}$$

Custom Waveform

Loads a waveform from an external text file, with time step, scale factor, and interpolation control.

Syntax:

```
CUS(file.txt Ts Sf Im [Td [Per]])
CUS(waveform.txt 1n 1 0)
```

- file.txt - Path to a plain-text file containing amplitude samples
- Ts - Separation time between the samples in *file.txt*
  Default: Tstep
- Sf - Scaling factor applied to the file samples
  Default: 1
- Im - Interpolation method:
  - 0 - None (step-like hold)
  - 1 - Linear
  - 2 - Cubic

  Default: 0
- Td - Start delay
  Default: 0
- Per - Repeat period
  Default: Tstop

Custom waveform file paths are resolved relative to the netlist or included file that references them.

DC

Provides a constant value. A bare scalar is also accepted anywhere a source function is expected.

Syntax:

```
DC A
DC 1.0
```

Noise

Generates sampled white-noise style excitation with a specified spectral amplitude and sample interval.

Syntax:

```
NOISE(A [Td [Ts]])
NOISE(1 1n 1p)
```

- A - Noise amplitude (spectral amplitude)
- Td - Time delay before noise starts
  Default: 0
- Ts - Noise sample time
  Default: Tstep

Power-Law Noise

Generates $1/f^{\alpha}$-style stochastic excitation.

Syntax:

```
PNOISE(A1 alpha [Td [dt [tcorr [N]]]])
PNOISE(1u 1 0 1p 10n 64)
```

- A1 - Amplitude spectral density at 1 Hz
- alpha - Power-law exponent
- Td - Start delay
- dt - Noise update interval
- tcorr - Correlation time used to set the low-frequency band edge
- N - Number of internal processes used to synthesize the spectrum

Flux-Noise Style Source

Generates $1/f^{\alpha}$-style source excitation referenced to Φ_0. This is primarily useful in phase or flux-oriented workflows.

Syntax:

```
FNOISE(Aphi0 alpha PHIDC [Td [dt [tcorr [N]]]])
FNOISE(1e-6 1 0 0 1p 10n 64)
```

- Aphi0 - Amplitude spectral density at 1 Hz, referenced to Φ_0
- alpha - Power-law exponent
- PHIDC - DC phase or flux offset, referenced to Φ_0

Exponential

Produces an exponentially rising and optionally falling waveform.

Syntax:

```
EXP(A1 A2 [Td1 [tau1 [Td2 [tau2]]]])
EXP(0 1 1n 1p 2n 1p)
```

The resulting waveform uses the standard decaying exponent form for both the rising and falling segments.

Piece-wise Sinusoidal

Generates a sinusoidal transition between each pair of successive time–value points.

Syntax:

```
PWS(T1 A1 T2 A2 T3 A3 ... Tn An)
PWS(0 0 10p 0 15p 1)
```

Stair

Generates a staircase waveform that increases by a fixed increment at a fixed interval, with optional delay, rise time, and initial offset.

Syntax

```
STAIR(A Per [Td [Tr [A0]]])
```

- A - Step increment
- Per - Step period
  Default: Tstep
- Td - Delay time
  Default: 0
- Tr - Rise time
  Default: Tstep
- A0 - Starting amplitude

Junction Critical-Current Noise Attribute

Josephson junction instances also support a dedicated `ICNOISE(...)` attribute for fractional critical-current noise:

```
B1 1 2 JJMOD ICNOISE=ICNOISE(0.01 1 0 1p 10n 64)
```

The argument structure matches `PNOISE(...)`.

16.6 Dependent source reference

16.6.1 Current-controlled current source (CCCS)

Syntax:

```
Fname node1 node2 control_pos control_neg gain
```

Creates a current source scaled by the specified gain and controlled by the current between the named control nodes.

16.6.2 Current-controlled voltage source (CCVS)

Syntax:

```
Hname node1 node2 control_pos control_neg transresistance
```

Creates a voltage source whose output is controlled by the current flowing between the named control nodes and scaled by the specified transresistance.

16.6.3 Voltage-controlled current source (VCCS)

Syntax:

```
Gname node1 node2 control_pos control_neg transconductance
```

Creates a current source whose output is controlled by the voltage across the named control nodes and scaled by the specified transconductance.

16.6.4 Voltage-controlled voltage source (VCVS)

Syntax:

```
Ename node1 node2 control_pos control_neg gain
```

Creates a voltage source scaled by the specified gain and controlled by the voltage across the named control nodes.

16.7 Control-command reference

Command names are case-insensitive and must match a supported command exactly. Inside a `.CONTROL` block, commands may omit the leading dot. Unsupported analysis or control commands stop the run with a source-location diagnostic before simulation output is written.

16.7.1 CLI analysis jobs and control blocks

The CLI executes each explicit `.TRAN` and `.IV` as a separate job, serially by default, in source order after includes are expanded. Each job starts with fresh simulation state. An IV-only deck does not run a transient; its requested model can be defined without a main circuit. With no explicit analysis, an unscoped deck or a single unlabeled control block uses one default transient.

For several runs of one circuit with shared settings and traces, place several explicit transient commands at top level. Each command supplies its own timing:

```
I1 0 1 DC 1m
R1 1 0 1
.PRINT V(1)
.TRAN 1p 20p 0 2p
.TRAN 2p 20p 0 4p
.END
```

For a deck named `compare.cir`, the default files are `compare.tran-001.csv` and `compare.tran-002.csv`. Settings at top level are shared defaults, not sequential changes between runs.

For a named run with its own settings and outputs, put one analysis in each control block. The following complete deck runs the same circuit twice:

```
I1 0 1 DC 1m
R1 1 0 1
.CONTROL fine
  .OPTIONS integration=trap
  .TRAN 1p 20p 0 2p
  .FILE fine.csv
  .PRINT V(1)
.ENDC
.CONTROL coarse
  .OPTIONS integration=bdf
  .TRAN 2p 20p 0 4p
  .FILE coarse.csv
  .PRINT V(1)
.ENDC
.END
```

Here `fine` requests a 1 ps simulation step and 2 ps output cadence; `coarse` requests a 2 ps simulation step and 4 ps output cadence. The fourth `.TRAN` argument sets output cadence independently of the simulation step. Local `.PRINT` directives select each run's columns; local `.FILE` directives select its destinations.

Use one explicit analysis per block for this named-run pattern. The CLI also accepts several analyses in a block, producing one job per analysis with shared block settings, and settings-only blocks, which produce no job when the deck contains explicit analyses. A block is declarative: it is not an executable control script, and there are no loops, conditionals or sequential assignments.

Block labels start with an ASCII letter and contain only letters, digits, underscores or hyphens. Labels are unique ignoring case. Blocks must balance and cannot nest. Named or multiple blocks require an explicit analysis. Top-level settings apply to all jobs; block settings override them only in that block, regardless of their position relative to the analysis. Parameters, models and circuit declarations belong outside named or multi-analysis blocks. A legacy single-analysis unlabeled block retains its declaration handling.

A block inherits the complete top-level output-directive sequence unless it provides its own output directives, which replace that sequence. IV jobs use their own destination and voltage/current columns. Transient output directives in an IV-only scope are rejected. An explicit nonzero seed starts equivalent fresh random streams for each job. Automatic seeding selects one seed for the deck and reports the effective seed for each job.

A single transient retains its output name; a single IV defaults to `base.iv.csv`. Multiple transient jobs derive names such as `base.fine.tran-001.csv`; multiple IV jobs without paths use names such as `base.iv-002.csv`. The number is the position in the complete job sequence. Unlabeled blocks use ordinal names such as `block-1.tran-001`. For repeated transients, CLI `-o` supplies the base and format; without it, the input filename supplies the base and CSV is used. Explicit `.FILE` and IV paths are never automatically renamed. Only one job may write STDOUT.

The CLI checks all resolved file destinations before simulation, including format/compression changes, input files and temporary-file names. Collisions are errors. Files are staged until their job succeeds. A failure or cancellation stops the sequence and preserves earlier successful jobs. Existing files for a failed job remain intact unless some renames have already completed during a multi-file commit; the summary lists each file's outcome. Already-emitted STDOUT data cannot be recalled. Ctrl-C and termination requests cancel active CLI jobs cooperatively. Abrupt process termination and filesystem changes by other processes are outside this cleanup guarantee.

These deck-job rules apply to CLI execution, the explicit deck-job APIs and GUI execution. The low-level C++, C API and Python simulation operations retain their single-analysis behavior.

16.7.2 GUI and API analysis jobs

The GUI runs every analysis in the deck and provides a result selector showing each job ID and outcome. Several analyses in one block share that block's settings and traces, while each retains its own timing and results. Select a successful waveform run to view its plots. Transient plots use time; IV plots use voltage on the horizontal axis and current on the vertical axis. Switching runs opens that run's source plots; plot edits and derived traces are not retained across run selection. Failed or cancelled runs have no selectable result, but earlier successful results remain available. Run details list file destinations and whether they were committed. Cancellation also stops an active IV analysis.

C++ clients use `AnalysisJobs(input, retainResults)`, Python clients use `AnalysisJobs(input, retain_results)`, and C clients use the `JPRO_AnalysisJobs_*` functions. Creation validates the complete plan; `run()` (C: `JPRO_AnalysisJobs_Run`) executes it once. Create a fresh plan for another execution. Existing low-level simulation calls do not execute a deck's other analyses.

Results expose job identity, block label, source, analysis kind, effective seed, integration method, transient timing, state and file publication outcomes. Inspect results before or after execution, including after an execution error; only cancellation may be requested concurrently. Planning errors do not return a plan. A plan snapshots its input settings and shares the supplied runtime's cancellation token. Clear a reused runtime's cancellation request before creating a new plan, or use a fresh runtime.

Retained results are optional and require buffered output. The GUI retains results for selection; clients can disable retention and use output files for streamed execution. Retention stores every successful run's visible series and therefore requires memory for their combined results. C++ and Python result snapshots and C output handles obtained from a plan remain valid after that plan is destroyed. For IV, the retained axis (the C output "time vector") is voltage in volts and the series is current in amperes; the analysis kind must be checked before interpreting the axis.

16.7.3 Static operating point

Use `.OP` to solve a static circuit without advancing a transient or constructing a time-domain waveform:

```
.MODEL JMOD JJ(IC=1m RN=5 R0=100 C=1p VG=2.8m)
V1 supply 0 DC 1m
R1 supply bias 2
L1 bias junction 2p
B1 junction 0 JMOD
.OP
.END
```

Run the saved deck with `josim-pro-cli -o bias.csv bias.cir`. The report contains one header row and one numeric row. CLI output and API scalar results include `V(node)` in volts, `P(node)` in radians for phase nodes, `I(component)` in amperes, `OP.iterations` and `OP.normalized_residual`. Currents are positive from the first terminal to the second. This example gives 0.5 mA through the resistor, inductor and junction, zero voltage at the superconducting nodes, and a junction phase of $\arcsin(0.5)$. Inductor phase difference is its flux linkage divided by $\Phi_0/(2\pi)$, including mutual inductance where present.

The supported circuit contains `R`, `C`, `L`, `K`, DC current/voltage sources, `JJ`/`RCSJ` junctions and versioned real-kernel TJM junctions. Capacitors are exactly open. Nodes touching an inductor or junction are phase nodes; their physical voltage is zero. Every connected inductor/junction network must reach circuit ground through inductors or junctions. Other nodes carry normal DC voltage. Resistors provide the interface between normal and superconducting nodes, as in the example; voltage sources cannot touch phase nodes directly.

RCSJ junctions use their model current-phase relationship, phase offset and supported temperature dependence. Real-kernel TJM uses its fit-derived sinusoidal equilibrium at the fitted temperature. Parameter spread retains its configured seed. Thermal and time-varying noise samples are excluded from the static mean. The solver starts from zero and reports the selected equilibrium. It does not search all roots, certify stability or select a nonzero trapped-flux sector. A running junction requires transient analysis; its average voltage is not a static operating point.

Temperature dependence requires $0 < T < T_C$. Nonzero `D` supports only a unit first `CPR` harmonic; custom Fourier harmonics require `D=0`.

Success requires `OP.normalized_residual` ≤ 1. Equation tolerances are 10^{-12} A, 10^{-9} V or 10^{-9} rad, plus 10^{-9} relative scale. Each junction phase correction must be at most 10^{-9} rad plus 10^{-9} times its magnitude. `NEWTONMAXITERS` and `NEWTONDAMPING` bound the solve. Transient timestep, integrator and error tolerances do not define this equilibrium. Failure reports the worst equation and preserves existing reports.

This interface accepts one argument-free `.OP` and no other analysis or `.STEP` in that deck. It returns all supported static quantities; `.PRINT`, `.PLOT`, `.SAVE`, `.FILE`, `.MEASURE` and explicit `.IC` are not supported. Legacy TJM FIT, transmission lines/MTL, controlled/phase sources and time-varying source waveforms are outside this analysis. Streaming and parallel scheduling are not supported. Specify one `.csv` or `.csv.gz` report path; without one, the input basename and job ID form a separate `.op-001.csv` filename. STDOUT and waveform file formats are rejected as report targets.

Execute `.OP` through `AnalysisJobs`; low-level `Matrix`/`Simulation` calls do not execute it. C++ `kind(index)`, C/Pascal `JPRO_AnalysisJobs_GetKind` and Python `jobs.kind(index)` report kind 2. `isIV` remains false, timing fields are zero and integration method is -1. The `measurements` array contains the scalars; retained results have empty time/trace arrays. The GUI keeps the editor visible and reports *Operating point ready*, with scalar and convergence details in Tasks/Output, without a waveform plot.

16.7.4 Circuit DC sweeps

`.DC` sweeps one existing independent DC current or voltage source:

```
.MODEL JMOD JJ(IC=1m RN=5 R0=100 C=1p VG=2.8m)
I1 0 junction DC 0
B1 junction 0 JMOD
.DC I1 -0.9m 0.9m 0.1m
.END
```

The syntax is `.DC source start stop increment`. Source names are case-insensitive native component labels. Range values are finite numeric constants with engineering suffixes; parameter names are not accepted here. The increment must be nonzero and point toward the stop value. Both ascending and descending ranges are supported, with at most 4,096 points. Equal start and stop values produce one point. An aligned endpoint is included within roundoff; an unaligned stop does not create a shortened final interval.

Each point uses the static solver and all model, topology and convergence limits in Section 16.7.3. Model parameters and seeded spread stay fixed across points. Every solve starts from zero. The example gives $P(\text{junction}) = \arcsin(I_1/I_C)$ and zero junction voltage. This is a curve of selected mathematical equilibria, without a stability or uniqueness certificate. It does not predict switching, hysteresis, a running junction's average voltage or measured-device agreement. Use transient analysis for dynamical behavior. Near-critical iteration limits and the perfect transparency start singularity remain applicable.

Run with `josim-pro-cli -P 17 -o curve.csv bias.cir`. The CSV begins with `Sweep(I1) (A)` or the swept voltage source in volts, followed by all static quantities and per-point convergence data. One header precedes the rows in sweep order. Specify one `.csv` or `.csv.gz` target; the default uses the input basename and `.dc-001.csv` job ID, with a named control block prefix where present. A failed or cancelled point reports its index, source and value. No partial report or retained curve is published, and an existing report remains intact.

Use one standalone `.DC` per deck. Nested/two-source, resistor and temperature sweeps, `.STEP`, other analyses, explicit `.IC`, waveform output directives, `.MEASURE`, streaming, checkpoints and parallel scheduling are outside this interface. A time-varying source cannot serve as the sweep source; all independent sources must use DC waveforms.

Execute through `AnalysisJobs`; analysis kind is 3 (Python `AnalysisJobKind.dc_sweep`). C++/Python `axis_metadata(index)` returns the axis name and unit. C/Pascal `JPRO_AnalysisJobs_GetText`, field 6, returns these as a JSON object. The existing retained axis (C "time vector") contains source values, not seconds; traces contain the corresponding static quantities. The scalar measurements array is empty. Timing fields are zero, integration method is -1 and `isIV` is false. The GUI plots against the source axis in amperes or volts; CSV import preserves this axis and supports descending sweeps.

16.7.5 Small-signal AC frequency sweeps

`.AC` solves the complex small-signal response of stationary circuits. The CLI, analysis APIs and GUI support native R/L/C/K, stationary RCSJ junctions, independent I/V/P sources, E/F/G/H controlled sources, T lines and `@MTL`. WIDEBAND requires all frequencies at or below its `FMAX`; TJM junctions require a versioned `josim-tjm-real/1` FIT file; legacy numeric FIT files are excluded. Run one standalone analysis per deck:

```
V1 IN 0 DC 2 AC 1
R1 IN OUT 1k
C1 OUT 0 1u
.AC DEC 20 1 10k
.PRINT AC V(OUT) V(IN) I(V1) P(OUT)
.END
```

Use `josim-pro-cli -o response.csv input.cir`. Select one grid form:

```
.AC LIN count start stop
.AC DEC points_per_decade start stop
.AC OCT points_per_octave start stop
```

Frequency bounds are finite positive numeric constants in Hz, with ordinary engineering suffixes. `M` means milli; use `MEG` or `X` for mega. Counts are decimal integer tokens from 1 to 65,536. LIN includes both endpoints; one LIN point requires equal bounds. DEC/OCT include both bounds and append a shortened last interval when necessary. Equal log bounds give one point. Frequencies must remain strictly increasing after conversion; zero frequency and grids exceeding 65,536 emitted points are rejected. Grid parameters and expressions are excluded.

Independent source attributes have this order:

```
V1 A 0 DC 2m AC 1m 90 PWL(0 7m 3p 10m)
I1 0 B AC(1u,-90)
P1 C 0 DC .2 AC .01
```

DC and transient clauses are optional. A bare scalar is a DC alias. AC magnitude must be finite and nonnegative; its optional angle is in degrees and defaults to zero. Scalar parameters are supported in source values. At least one resolved nonzero independent AC drive is required. Multiple drives superpose without automatic normalization. Unused transient waveforms are not sampled or opened to obtain bias: absent explicit DC means zero DC. In a transient run, the explicit waveform supplies the signal; otherwise DC supplies it. DC is not added to an explicit waveform, and AC attributes do not generate a transient signal. Combined AC attributes remain excluded from standalone `.OP`/`.DC`.

The phasor convention is

$$x(t) = x_0 + \Re\{\hat{x}e^{j2\pi ft}\}.$$

Amplitudes are peak values. For a phase source, DC and AC magnitudes are radians, with $\hat{V} = j2\pi f\hbar/(2e)\,\hat{P}$; constant phase imposes zero DC voltage. `P(A,B)` reports phase excursion in radians, $[\hat{V}(A) - \hat{V}(B)]/[j2\pi f\hbar/(2e)]$, rather

than a phasor angle. Controlled sources sense physical volts/amps. F/H retain their native four-terminal, zero-voltage current-sensing ports; negative and zero gain are supported in AC.

Request explicit `.PRINT AC V(node), V(node1,node2), I(component)` or `P(node1,node2)` selectors. One-node V/P selectors use ground as the second terminal. Repeated selectors appear once in source order. One explicit ratio is also supported: equal-unit signals give a dimensionless gain, V/I gives impedance in ohms, and I/V gives admittance in siemens. For example:

```
.PRINT AC V(OUT) V(IN) V(OUT)/V(IN) V(OUT)/I(R1)
```

A zero denominator in a requested output trace fails the entire staged analysis. Other arithmetic, power, energy and AC measurements are excluded. The output uses one `.csv` or `.csv.gz` target, selected by `-o` or one `.FILE`, with a default `.ac-001.csv` suffix. Named control blocks add their ordinary job prefix. STDOUT, RAW, streaming, `.STEP`, mixed analyses, initial conditions, checkpoints and parallel scheduling are excluded.

The CSV starts with a `# josim-ac-csv/1` JSON metadata line, then `frequency (Hz)` and paired real/imaginary columns for each selector. Numbers retain 17 significant digits and always use an ASCII decimal point, independently of the GUI locale. Gzip decompresses to the same typed content. Metadata records stimulus values, effective seed, bias convention, model values and numerical validity. A failed or cancelled bias, frequency or write publishes no partial curve and preserves an existing target.

Bias uses a fresh zero-start stationary state. A bounded positive-R/C/I network may have a consistent nonunique DC common mode; it is admitted only if every requested finite-frequency operator has a unique physical response. Its metadata identifies the DC solution class, without inventing a unique OP. Constant-parameter lines also permit a numerically verified linear DC solution class under the same finite-frequency uniqueness requirement. Singular line bias reduction is limited to 256 unknowns; a larger singular case is rejected explicitly. General nonlinear singular bias topologies remain excluded. Phase gauges never impose an electrical ground. No stability or global root-uniqueness certificate is provided.

RCSJ junctions use the selected stationary phase ϕ_0 and the same effective AREA, CPR, temperature and spread values throughout the sweep:

$$Y_{\mathrm{JJ}}(s) = G_{\mathrm{qp}} + sC + \frac{K}{s\,\hbar/(2e)}, \qquad K = \left.\frac{dI_s}{d\phi}\right|_{\phi_0}.$$

Here $s = j2\pi f$. For a sinusoidal CPR, $K = I_C\cos(\phi_0 - \mathrm{PHI})$ and $L_J = [\hbar/(2e)]/K$ when $K \neq 0$. Exactly zero stiffness gives the RC response directly. Zero capacitance is supported. Fourier harmonics and phase offsets enter the derivative; an instance `IC=` sets AREA from the model current scale.

`RTYPE=0` uses $G_{\mathrm{qp}} = 1/R_N$. `RTYPE=1` uses $1/R_0$ when $V_G - \mathrm{DELV}/2$ is positive, and the transition slope $I_C/(\mathrm{ICFCT}\,\mathrm{DELV})$ when it equals zero. A transition crossing zero is rejected unless its signed current offset vanishes exactly. Temperature-dependent models require $0 < T < T_C$; finite transparency requires `CPR=`{1}, with zero trailing coefficients permitted. Custom thermal Fourier CPRs require `D=0`.

JJ metadata records the bias and effective coefficients. Singular static stiffness requires a structurally complete system, a zero-stiffness JJ, zero DC forcing and an exactly zero residual at zero start; every AC operator must remain uniquely solvable. Negative stiffness has no stability guarantee. Running junctions require periodic analysis and are rejected. Existing convergence limits apply. Thermal gap accuracy and measured-device calibration remain unassessed.

Real-kernel TJM junctions retain their fixed-phase memory $F_n = \cos(\phi_0/2)$, $G_n = \sin(\phi_0/2)$ and equilibrium $I_0 = I_{c,\mathrm{fit}}\sin\phi_0$. With $q = s/\omega_g$, their small-signal response is

$$\begin{aligned} Y_{\mathrm{TJM}}(s) &= \frac{1}{R_N} + sC + \frac{K(s)}{s\,\hbar/(2e)}, \\ K(s) &= \frac{V_G}{2R_N}\big\{\cos\phi_0[H_p(0) + H_p(q)] \\ &\qquad + H_q(0) - H_q(q)\big\}, \\ H_p(q) &= \frac{1}{2}\sum_n \left[\frac{\alpha A_n}{q - p_n} + \frac{\overline{\alpha A_n}}{q - \overline{p_n}}\right]. \end{aligned}$$

H_q uses B_n in place of αA_n. Conjugation applies to the real-kernel coefficients, not to the response phasor. Normal current and capacitance appear once each. Every requested frequency must be at most the FIT's `fmax_Hz`; an out-of-band request publishes no curve. Results retain the fitted coefficients, source, material, temperature, suppression, effective area and stationary memory. Legacy FIT, noise and finite-voltage running states are excluded. Pole decay alone does not establish passivity, root stability or measured-device accuracy.

T lines support `LOSSLESS`, `LCLADDER`, `RLGC`, `MODAL` and `WIDEBAND`; `@MTL` supports its full native constant R/L/G/C matrices. Finite models retain their specified `NSEG`, endpoint half sections and native modal normalization. Increasing `NSEG` refines the finite model; it does not select an exact distributed-line replacement. LOSSLESS uses the continuous propagation factor $e^{-s\,\mathrm{TD}}$, including half-wave frequencies when the terminated circuit is solvable. For MODAL, $V_{\mathrm{DM}} = V_p - V_n$, $V_{\mathrm{CM}} = (V_p + V_n)/2$, $I_{\mathrm{DM}} = (I_p - I_n)/2$ and $I_{\mathrm{CM}} = I_p + I_n$.

WIDEBAND sections use the continuous per-length functions

$$\begin{aligned} Z'(s) &= R' + sL' + \sum_k \Delta R'_k \frac{s}{s + p_{R,k}}, \\ Y'(s) &= G' + sC' + \sum_k \Delta G'_k \frac{s}{s + p_{G,k}}, \end{aligned}$$

where $p_k = 2\pi f_k > 0$. Both real loss and reactive dispersion are included. Endpoint half-section weights also apply to the loss increments. The declared fit range is enforced before solving the sweep; `FMAX` is not a filter, and a small residual does not establish fit accuracy.

Stationary relaxation states equal their section current or voltage. AC bias uses a zero relaxation-flux reference. In a series section with zero base resistance, the static inductance per length is $L' + \sum_k \Delta R'_k / p_{R,k}$. This stored flux participates in superconducting-loop bias. For a matching transient comparison, reach the bias from zero state with a ramp. Nonzero-current transient DC priming seeds a different relaxation-flux reference. Frequency analysis does not depend on the transient timestep or integration method.

`V(T1)` and `P(T1)` select the first differential port; a physical node named `T1` takes precedence. `I(T1)` is the current entering its first positive terminal. For an n-conductor `@MTL X`, use `I(X[k])`: indices $1 \ldots n$ enter P1, and $n+1 \ldots 2n$ enter P2. Select MTL voltages with explicit conductor and reference nodes. Generated internal sections are not output selectors. Parent metadata records effective line parameters and determined DC port values; indeterminate port values are null. Line initial conditions are excluded. Ambiguous zero-resistance modes and unrepresentable static reductions fail explicitly; no phase gauge creates an electrical ground.

Admission allows at most 4,096 traces, 256 MiB of raw retained samples and 1 MiB of metadata plus headers. These limits do not bound sparse factor fill or total process memory. Nonfinite, singular or unrepresentable frequency equations fail explicitly.

Retained complex results. Retrieve a detached owner after successful execution with retention enabled:

C++ `AnalysisJobs::ac_result(index)`.
C/Pascal `JPRO_AnalysisJobs_GetAcResult`.
Python `jobs.ac_result(index)`.

The result owns frequency and complex arrays beyond plan or input destruction. Analysis kind is 4 and the axis is Frequency/Hz. The existing real `Output` and `Print` interfaces retain their real signal semantics; requesting a real Output for AC is rejected. C++ `ac_result_json(index)` remains available.

C++ `AcResult` exposes read-only frequency, trace name/unit and complex vectors, metadata JSON, `view_json`, and typed `load`/`save`. C/Pascal `JPRO_AcResult_*` functions provide shape, borrowed read-only frequency, copied separate Re/Im arrays, UTF-8 text/views, load/save and explicit destruction. A borrowed frequency pointer is valid until that result handle is destroyed. Trace copy requires separate nonoverlapping buffers; a size query reports the required sample count without writing data. Text queries use the standard two-call byte count including the terminating NUL. Each retrieved handle must be destroyed independently; handles remain valid after plan closure.

Derived views use the complex values rather than real-only trace arithmetic. Magnitude is linear in the trace's physical unit. Phasor angle is in $(-180, 180]$ degrees; exactly zero response has no defined angle. Optional unwrapping resets across undefined points. Interactive ratios of retained operands report a null value with `zero_denominator` validity when needed. Canonical stored/exported samples stay finite. Optional dB views require an explicit reference: `1 V` for dimensionless gain, `1 V` for dBV (peak), or `1 A` for dBA (peak). Zero magnitude is negative-infinite dB, represented by a null numeric value and an explicit status. No average complex-signal power result is offered.

Typed load/save preserves Re/Im, frequency, units, stimulus, bias/root declarations, effective seed and case/source identity in CSV or CSV.gz. Import validates the format and finite arrays; it does not independently requalify the recorded physical model or selected bias.

16.7.6 Transient analysis

Syntax:

```
.TRAN time_step stop_time [print_start_time] [print_step] [UIC]
     [STARTUP|DST]
```

Examples:

```
.TRAN 0.01ns 10ns
.TRAN 0.01ns 10ns 0 0.02ns UIC
.TRAN 0.01ns 10ns STARTUP
```

Each explicit command starts an independent CLI run; see the named-run examples on page 85.

The optional trailing flags may appear in any order:

- `UIC` — use the user-specified initial conditions directly instead of the default consistent start,
- `STARTUP` — enable legacy startup priming,
- `DST` — explicitly disable legacy startup priming.

If `print_step` is omitted, output is emitted at the accepted simulation steps. If it is provided, it must be positive.

Initial conditions are supplied separately with `.IC`. Supported forms include node-oriented aliases and explicit typed targets:

```
.IC n1=0
.IC V(n2)=0
.IC P(n3)=PI/2
.IC B1[2]=0
```

16.7.7 Parameter sweeps

`.STEP` repeats an explicit `.TRAN` or `.IV` analysis with different values of an existing global `.PARAM`. Use a list or a linear range:

```
.STEP PARAM name LIST value ...
.STEP PARAM name start stop increment
```

Values must be finite numeric constants. Numeric suffixes and constant expressions without spaces are accepted; references to other parameters and random functions are not accepted as sweep values. The default parameter declarations must also resolve normally. Dependent parameters, models and subcircuit instances are evaluated for each case.

```
.PARAM bias=1m
.PARAM resistance=1
I1 0 1 DC bias
R1 1 0 resistance
.STEP PARAM bias LIST 1m 2m
.STEP PARAM resistance 3 1 -1
.TRAN 1p 5p
.PRINT V(1)
.MEASURE TRAN peak MAX V(1)
.END
```

This example runs six cases: resistance 3, 2 and 1 for bias 1 mA, followed by the same resistances for bias 2 mA. Multiple directives form a Cartesian product; the last dimension varies fastest. Lists preserve order and may repeat a value. Linear ranges include aligned endpoints, allowing floating-point roundoff, and do not add a shortened final interval. The increment must be nonzero and point toward the stop value. Equal endpoints produce one case.

Global directives apply to every analysis. Directives inside a control block add dimensions for that block; a parameter may appear only once in an analysis's effective sweep. A local sweep requires an analysis in that block. The limits are eight dimensions per analysis, 256 values per dimension, 256 total jobs per deck including unswept analyses, and 65536 bytes per directive. These limits bound run count; transient duration and retained waveform memory still depend on the analysis settings.

Each case starts a fresh simulation and uses the effective analysis seed. An unspecified seed is chosen once per deck and reported with each job. Cases share that seed so repeated cases use the same random stream. Sweeps run serially by default, in analysis source order. They do not carry device history between cases and do not implement Monte Carlo or continuation analysis.

Each swept job ID ends in `.case-N`. Its output filenames include the complete job ID before the extension, including explicit `.FILE`, CLI and IV paths. For example, `.FILE results.csv` produces names such as `results.tran-001.case-1.csv`. Compressed filenames retain their `.csv.gz` suffix. Sweep output requires files; `STDOUT` is rejected. All cases and destinations are checked before execution. Failure or cancellation stops the plan, preserves earlier committed results and leaves later cases unrun.

CLI and GUI Tasks/Output show each case's assignments and measurements. C++ and Python job results expose ordered `parameters` pairs and the existing `measurements` list, including when waveform retention is disabled. C API `JPRO_AnalysisJobs_GetText` field 5 returns a JSON array of objects with `name` and `value` fields using the usual two-call buffer convention; field 4 returns measurements. Single-Matrix API workflows reject `.STEP`; use `AnalysisJobs` to execute sweeps.

16.7.8 Parallel transient sweeps

On Linux, independent buffered fixed-step transient sweep cases can run in isolated worker processes. Serial execution is the default. IV, mixed or unswept plans, adaptive stepping, streaming, waveform `STDOUT`, verbose generated-file output and shared solver diagnostic files are rejected for parallel execution. The GUI uses serial execution.

Supply all three CLI options. Worker count is a maximum from 1 through 4; memory values are in MiB, where 1 MiB is 1,048,576 bytes:

```
josim-pro-cli --jobs=4 --memory-budget=2048 \
  --job-memory=256 -o sweep.csv circuit.cir
```

The matching JoSIM-Pro CLI must be installed beside the core library, or in the sibling `bin` directory. Other platforms reject parallel requests.

The total budget covers conservative reservations for the plan, active workers, completed cases awaiting publication and all requested retained results. It is not a cap on the host application's RSS or memory allocated by user callbacks. Input reading and plan construction precede execution admission. Each worker has a separate enforced virtual-address-space limit of at least 64 MiB, including its libraries and simulation storage. A case that exceeds this limit fails without publishing its results.

Fewer workers can run when the remaining reservation budget is insufficient. A plan that cannot admit one case and its required retained results fails before dispatch. Retained output without a positive print cadence needs a larger reservation because fixed-step reductions can increase its sample count. An explicit `.TRAN` print step permits a tighter bound.

Cases keep their original IDs, assignments and effective seeds. Results and file commits follow source order even when computation finishes out of order. On failure, preceding cases settle first; later speculative cases are discarded and reported as cancelled, while never-started cases are not run. The earliest failing case in source order supplies the failure. Cancellation preserves committed files. A multi-file commit can retain some committed files if cancellation or an I/O error interrupts the rest; measurements and retained plots require the whole case to succeed.

In C++, call `set_parallel` on `AnalysisJobs`; in C, call `JPRO_AnalysisJobs_SetParallel`. Supply the worker count, total budget and per-worker limit before execution and before requesting plan cancellation. All API memory limits use bytes. Only `request_abort()` may overlap `run()`; query results after it returns. Parallel plan cancellation leaves other plans' tokens unchanged. Explicit cancellation through a shared caller Runtime token affects plans that share that token. Progress and detail callbacks run on the thread calling `run()`, with case identity and coalesced progress. Keep settings and the process environment unchanged during execution. Per-case feedback logs retain at most 65,536 bytes.

16.7.9 Transient measurements

.MEASURE TRAN evaluates named scalar results from buffered transient samples. It uses accepted solver samples from the complete run, independently of the .TRAN print start and print cadence. Measurement expressions use the same signal references and parameter evaluator as derived .PRINT expressions. Enclose expressions containing spaces in braces or quotes.

```
.MEASURE TRAN name AVG expression [FROM=time] [TO=time]
.MEASURE TRAN name RMS expression [FROM=time] [TO=time]
.MEASURE TRAN name MIN expression [FROM=time] [TO=time]
.MEASURE TRAN name MAX expression [FROM=time] [TO=time]
.MEASURE TRAN name PP expression [FROM=time] [TO=time]
.MEASURE TRAN name INTEG expression [FROM=time] [TO=time]
.MEASURE TRAN name FIND expression AT=time
.MEASURE TRAN name WHEN expression=value [RISE=n|FALL=n|CROSS=n]
  + [TD=time] [FROM=time] [TO=time]
.MEASURE TRAN name TRIG expression VAL=value [RISE=n|FALL=n|
    CROSS=n]
  + [TD=time] TARG expression VAL=value [RISE=n|FALL=n|CROSS=n]
  + [TD=time]
```

Names match [A-Za-z_][A-Za-z0-9_]* and are unique within a job, ignoring case. TRAN is required. FROM defaults to zero and TO to the transient stop time; require $0 \leq \mathrm{FROM} < \mathrm{TO} \leq t_{\mathrm{stop}}$. Numeric settings accept finite parameter expressions and the usual numeric suffixes. AT and TD must lie within the transient interval.

Values between adjacent samples are linearly interpolated, including window endpoints. AVG is the time integral divided by the window duration; it is not a sample-count average. RMS is the square root of the time average of the squared, piecewise-linear signal. INTEG integrates that signal. MIN, MAX and PP include interpolated endpoints; PP returns maximum minus minimum. FIND interpolates the value at AT.

WHEN returns a crossing time in seconds. Select one of RISE, FALL or CROSS per event. Counts start at one after the event's TD and within its window; the default is CROSS=1. LAST selects the last qualifying event. Reaching the threshold from strictly below or above counts once; consecutive samples on the threshold do not add events. Equality at the first sample alone is not an event. Crossing times are linearly interpolated and boundary events are included.

TRIG/TARG returns target time minus trigger time. Each event has its own count and delay. Counts are independent, so a target before its trigger produces a negative result. For example:

```
.MEASURE TRAN peak MAX V(out) FROM=10p TO=100p
.MEASURE TRAN average AVG {V(out) - V(ref)} FROM=10p
.MEASURE TRAN sample FIND I(R1) AT=25p
.MEASURE TRAN switching WHEN V(out)=100u RISE=1
.MEASURE TRAN delay TRIG V(in) VAL=100u RISE=1
  + TARG V(out) VAL=100u RISE=1
```

Top-level measurements apply to every transient job. A control block adds its local measurements to those inherited definitions. Duplicate names in the effective job are errors. Local measurements require a transient in that scope; measurements do not run on IV jobs. With mixed IV/transient jobs, top-level measurements apply only to the transients.

Single-signal values report V, A, rad, W or J; integrals append *s. General expressions have no inferred unit label. Event times and delays report s. CLI reports use normal diagnostic output, separate from waveform STDOUT; quiet mode suppresses them. The GUI includes results in Tasks and Output. API results retain their names, numeric values and unit labels even when waveform retention is disabled.

Streaming measurements, other analysis types and unsupported measurement forms are rejected. There are at most 256 measurements per job and 65,536 bytes per directive. A missing event, uncovered interval or non-finite measured value fails the job without returning partial measurement results. File publication follows the job's ordinary success and cancellation rules. These commands do not schedule sweeps, stop the simulation early or change the solver.

16.7.10 Parameter definition

Syntax:

```
.PARAM name=mathematical_expression
```

Example:

```
.PARAM Rval=50*cos(2*PI)
R1 1 0 Rval
```

16.7.11 Subcircuits

Syntax:

```
.SUBCKT name io_nodes
elements
.ENDS name
```

Example:

```
.SUBCKT amplifier 1 2 3
R1 1 2 100
L1 2 3 1e-9
.ENDS amplifier
```

Usage syntax:

```
Xname io_nodes [component_name=parameter]
```

Usage example:

```
X1 1 2 3 amplifier R1=400
```

16.7.12 Include files

Syntax:

```
.INCLUDE "filename"
```

Example:

```
.INCLUDE "common_components.cir"
```

16.7.13 Noise settings

Syntax:

```
.TEMP temperature_in_K
.NEB bandwidth_in_Hz
```

Example:

```
.TEMP 4.2
.NEB 1GHz
R1 1 0 50 temp=4.2 neb=1GHz
B1 1 2 JJMOD temp=4.2 neb=1GHz
```

These commands set the global temperature and effective noise bandwidth used when thermal noise is enabled on supported resistive devices and junctions. Individual resistors and Josephson junctions can override the global values with `temp=` and `neb=` attributes.

16.7.14 Parameter spread

Syntax:

```
.SPREAD [global_spread] [B=junction_spread] [C=capacitor_spread]
      [L=inductor_spread] [R=resistor_spread]
```

Examples:

```
.SPREAD 0.02
.SPREAD 0.02 L=0.10 C=0.05 R=0.10 B=0.20
R1 1 0 50 spread=0.05
```

The optional leading scalar applies a global spread. Family-specific overrides apply to Josephson junctions, capacitors, inductors, and resistors respectively. Individual devices can also specify `spread=` directly on the instance.

16.7.15 Output control

Syntax:

```
.PRINT type(device)
.SAVE type(node)
.PLOT type(node/device)
```

Example:

```
.PRINT V(1) I(R1)
.SAVE P(B1)
.PLOT NODEP B1
```

`.PRINT`, `.PLOT` and `.SAVE` perform the exact same function and can be used interchangeably.

Output Types

The following output types can be used within the **.PRINT**, **.SAVE** or **.PLOT** commands:

- **NODEV**: Nodal voltage between a specified node and ground, or between two nodes.
- **NODEP**: Nodal phase between a node and ground, or between two nodes.
- **DEVV**: Device voltage across a specific element, such as a resistor or Josephson junction.
- **DEVI**: Device current through an element.
- **DEVP**: Device phase associated with a particular element.
- **V()**: Voltage at a node (or between two nodes).
- **I()**: Current through a device.
- **P()**: Phase of a node or element.
- **W()**: Instantaneous apparent power of element. Only works on elements and not nodes.
- **E()**: Energy consumed/produced by element. Only works on elements and not nodes.

Example:

```
.PRINT V(1) I(R1) P(J1)
.PRINT NODEV(1) NODEP(2, 3)
.SAVE DEVV(R1) DEVI(L1) DEVP(J1)
.PRINT E(B01) W(IB01)
```

These commands allow users to monitor and analyze specific quantities in their circuit simulations, providing detailed insight into the behavior of both nodes and devices.

Subcircuit Output Referencing

Referencing the nodes or devices within a subcircuit requires unrolling the hierarchy of the subcircuit instantiations to the top level. In JoSIM®-Pro, subcircuits can be nested within each other, meaning that devices or nodes within a deeply nested subcircuit must be referenced using their complete hierarchical path.

Syntax for Subcircuit Referencing:

```
device.subckt1.subckt2.subcktN
```

Here, the device name refers to the component inside the innermost subcircuit, and each subcircuit name identifies the containing hierarchy as you move outward to the top-level netlist.

Example:

```
R1.X1.X2.X3
```

In this example:

- `R1` is a resistor inside the subcircuit `X1`.
- `X1` is instantiated inside another subcircuit `X2`.
- `X2` is instantiated within the subcircuit `X3`.
- `X3` is part of the top-level netlist.

To output the voltage across `R1`, you would use the following command:

```
.PRINT V(R1.X1.X2.X3)
```

Explanation: When JoSIM®-Pro unrolls the subcircuits during simulation, it treats each nested subcircuit as a hierarchical block. In order to refer to a specific device or node within such a hierarchy, you must fully specify the path from the innermost device to the top level. This ensures that the simulator correctly identifies the element within the nested structure.

More Examples: If you want to print the phase of a Josephson junction `B1` inside a deeply nested subcircuit, you would reference it like this:

```
.PRINT P(B1.X1.X2)
```

To save the current through an inductor `L2` inside the subcircuit `X4`:

```
.SAVE I(L2.X4)
```

Why This is Important: Proper subcircuit referencing ensures that the simulator can correctly locate the elements, especially when dealing with large and complex designs. Nested subcircuits allow modular design, but without clear referencing, it would be impossible to track specific elements within the hierarchy during the simulation. JoSIM®-Pro's ability to handle these references also makes it easier to debug and analyze simulations that involve reused or replicated components.

Expanded Nested Example:

```
.SUBCKT inner 1 2
R1 1 2 50
.ENDS inner

.SUBCKT middle 3 4
X1 3 4 inner
L1 3 4 1e-9
.ENDS middle

.SUBCKT outer 5 6
X2 5 6 middle
V1 5 0 DC 1
.ENDS outer

X3 1 2 outer
.TRAN 0.01ns 1ns
.END
```

In this example:

- `R1` is located inside the `inner` subcircuit.
- The `inner` subcircuit is instantiated as `X1` within the `middle` subcircuit.
- The `middle` subcircuit is instantiated as `X2` within the `outer` subcircuit.
- Finally, `outer` is instantiated as `X3` in the top-level netlist.

To print the voltage across `R1`, the correct reference would be:

```
.PRINT V(R1.X1.X2.X3)
```

This ensures that JoSIM®-Pro traces through the entire hierarchy to correctly locate `R1`.

Output Operations: Mathematical operations can now be performed on output traces with the same level of capability seen in `.PARAM` expression parsing. This enables users to extract complex outputs directly from the simulation without the need for post processing.

```
.PRINT V(R01) + V(R02)
.PLOT SIN(P(B01))
```

16.7.16 IV curve generation

Syntax:

```
.IV model_name [max_current] [filepath] [current_steps]
```

Example:

```
.IV JJMOD 300u example_iv.csv 400
```

This command generates a current-versus-voltage sweep for the named model to $\pm$ the requested maximum current, using the requested number of steps and storing the result at the requested file path. When omitted, the code supplies defaults for the maximum current, output path, and step count.

16.7.17 File output

Syntax:

```
.FILE filepath
```

Example:

```
.FILE results1.csv
.PRINT V(1) I(R1)
.FILE results2.csv
.PRINT P(J1)
```

The **.FILE** command allows for multiple output files. Each output command following a **.FILE** line stores results in the specified file.

The chosen file extension still determines the intended format for that target. When streaming output is enabled from the CLI, text targets are the intended path; requests for BIN or RAW output normalize to streamed CSV instead. See the Usage chapter for the CLI-side streaming controls.

16.7.18 Runtime and .OPTION controls

The .OPTION command is used to control simulation behavior directly from the netlist. This section documents the settings most relevant to everyday use, including analysis mode, integration method, output formatting, and variable-step controls.

General syntax:

```
.OPTION key=value [key=value ...]
```

General Runtime Controls

The following .OPTION controls are commonly used when configuring a run:

- analysis=phase, voltage, 0, or 1
- integration=BDF, TRAP, 0, or 1
- seed=*integer*
- outputprecision=*integer* or precision=*integer*
- compressed=ON, OFF, 1, or 0; alternatively nocompress
- newtonjj=ON, OFF, 1, or 0
- newtonmaxiters=*integer*
- newtonreltolphase=*value*
- newtonabstolphase=*value*
- newtondamping=*value*
- variablestep=ON, OFF, 1, or 0

Examples:

```
.OPTION analysis=PHASE integration=BDF
.OPTION seed=123456 outputprecision=12 compressed=ON
.OPTION newtonjj=ON newtonmaxiters=20 newtondamping=0.5
.OPTION variablestep=ON
```

Practical selection: PHASE is the default and is the mode to use when a workflow depends on Josephson phase evolution or variable-step operation. VOLTAGE remains available for voltage-domain workflows. BDF is the default integration method and is also the required method for variable-step runs, while TRAP remains available for fixed-step use.

Curated Adaptive-Step Surface

The following adaptive-step controls are the main settings users are most likely to adjust when tuning variable-step behaviour:

- stepreltolphase, stepabstolphase
- stepreltoldphi, stepabstoldphi
- stepreltoldphidt, stepabstoldphidt, stepltedphidtweight
- stepreltoljjvolt, stepabstoljjvolt, stepltejjvoltweight
- stepsafetyfactor, stepmaxgrow, stepminshrink
- stepgrowththreshold, stepgoodstepsbeforegrow
- stepmindt, stepmaxdt, stepstartupdt
- stepcapstartupbreakpoint, stepstartupbreakpointfrac
- stepcapjjphaseadvance, stepjjetaphi
- stepcapjjvgapproach, stepjjvgapproachfrac, stepjjvgapproachsafety

Example:

```
.OPTION variablestep=ON integration=BDF analysis=PHASE
.OPTION stepreltolphase=1e-3 stepabstolphase=1e-6
.OPTION stepmindt=1e-15 stepmaxdt=1e-12 stepstartupdt=5e-15
.OPTION stepcapjjphaseadvance=ON stepjjvgapproachfrac=0.85
```

Constraints and Invalid Combinations

Variable-step mode is only supported for PHASE + BDF. If variablestep=ON is combined with a different analysis type or integration method, JoSIM®-Pro rejects the run explicitly.

CLI overrides still take precedence when the corresponding CLI override was set for the run. For example, --varstep or --no-newtonjj on the command line overrides the same setting in the netlist.

Additional advanced variable-step controls are also available for breakpoint fencing, local growth clamps, BDF1 restart behaviour, oscillation recovery, JJ-activity gating, and startup holdoffs. These controls are intended for specialised tuning rather than everyday use, but they are part of the supported .OPTION surface.

Recommended Practice

Use .OPTION to keep simulation intent close to the netlist, and use the CLI for per-run overrides such as output location, verbosity, and licensing. In most cases, it is best to start with the default adaptive-step settings and only tune them when a specific circuit requires it. For exact command-line forms of --varstep, output streaming, and analysis selection, see the Usage chapter.

16.8 Built-in constants

Constant	Symbol	Value
PI	π	3.141592653589793
PHI_ZERO	Φ_0	2.067833831170082E-15
BOLTZMANN	k_B	1.38064852E-23
EV	e	1.6021766208E-19
HBAR	$\hbar$	1.0545718001391127E-34
C	c	299792458
MU0	μ_0	12.566370614E-7
EPS0	ϵ_0	8.854187817E-12
RFQ	$\frac{\Phi_0}{2\pi}$	3.291059757E-16

16.9 Summary

This chapter provided a detailed explanation of the JoSIM®-Pro netlist syntax, including component definitions, dependent sources, model parameters, and built-in constants.

Worked Netlist Examples

This chapter presents practical example workflows for JoSIM®-Pro. Each example is organized around a task rather than around a standalone netlist dump. For every workflow, the goal is the same: show what problem the example solves, what command to run, what output to inspect, and what behavior to expect from the results.

Example paths in this chapter have two explicit scopes:

- **Installed examples** are files that the package installs under its `examples` directory. Use these for first-use GUI work, quick CLI validation, and Optimum handoff.
- **Manual-only snippets** are small self-contained netlists printed in this chapter. They are not installed as named files; copy them into your own working directory when you want to reproduce the workflow.

The installed examples directory contains the following files:

Installed file	Primary use
`JTL_Example.cir`	First simulation and quick CLI validation.
`AND_Example.cir`	Installed RSFQ AND-gate example; requires the installed `base_cells.cir` in the same directory.
`Splitter_Example.cir`	Installed RSFQ splitter example.
`DFF_Example.cir`	Installed DFF circuit used by the Optimum example.
`DFF_Example.sp`	Installed DFF verification specification used by Optimum.
`DFF_Example_config.toml`	Installed Optimum configuration for the DFF example.
`base_cells.cir`	Installed reusable cell library used by `AND_Example.cir`.

Additional example files outside this installed list are not assumed to be present after installation. Some are large stress designs, generated netlists, or environment-specific examples. They are therefore not used by this manual's guided workflows.

The examples below are intentionally small enough to adapt into your own netlists. For larger superconducting logic cells and system-level designs, the same workflow patterns apply: build the hierarchy, choose the analysis mode and stepping method, select the outputs you need, and run the simulation in the format that best fits the downstream toolchain. The chapter closes with installed full logic-cell examples.

17.1 First transient run and result inspection

Problem this solves: You want to confirm that a simple transient simulation runs correctly and that you understand how to read the resulting waveform file.

Example netlist (save as `first_run.cir`):

```
* first_run.cir
.MODEL JJMOD JJ(IC=200u RN=15 R0=30 CAP=0.2p)
VIN in 0 PULSE(0 830u 20ps 2ps 2ps 8ps 40ps)
L1 in j 2p
B1 j 0 JJMOD
R1 j 0 2

.TRAN 0.25ps 120ps
.PRINT V(in) V(j) I(B1) P(B1)
.END
```

Command to run:

```
josim-pro-cli -o first_run.csv first_run.cir
```

What to inspect:

- the time column in `first_run.csv`
- the input drive `V(in)`
- the junction node response `V(j)`
- the junction current and phase: `I(B1)` and `P(B1)`

What to expect:

- The CSV file will contain the requested output traces together with the simulation time points.
- The input pulse at `V(in)` drives the junction branch through `L1`.
- The phase and current traces provide a quick way to see whether the junction remains in a quiet operating region or switches dynamically during the transient.

Why this example matters: This is the smallest useful end-to-end workflow in JoSIM®-Pro: define a model, drive a circuit, choose outputs, run the simulator, and inspect the output file.

17.2 Hierarchical design with subcircuits and included files

Problem this solves: You want to organize reusable building blocks in separate files and still reference internal devices cleanly in your output requests.

Library file (save as `base_cells.cir`):

```
* base_cells.cir
.SUBCKT jjbranch IN OUT
.PARAM RSH=5
BJJ IN N1 JJMOD IC=200u
RJJ IN N2 RSH
LRJJ N2 N1 0.1p
LRET N1 OUT 0.1p
.MODEL JJMOD JJ(RN=15 R0=30 CAP=0.1p)
.ENDS
```

Top-level netlist (save as `hierarchy_run.cir`):

```
* hierarchy_run.cir
.include base_cells.cir

.SUBCKT delay_stage IN OUT
L1 IN N0 2p
X1 N0 OUT jjbranch
.ENDS

VIN in 0 PULSE(0 830u 20ps 2ps 2ps 8ps 50ps)
XSTAGE1 in mid delay_stage
XSTAGE2 mid out delay_stage
RLOAD out 0 2

.TRAN 0.25ps 150ps
.PRINT V(in) V(mid) V(out)
.PRINT P(BJJ.X1.XSTAGE1) P(BJJ.X1.XSTAGE2)
.END
```

Command to run:

```
josim-pro-cli -o hierarchy_run.csv hierarchy_run.cir
```

What to inspect:

- the stage-to-stage propagation from `V(in)` to `V(mid)` to `V(out)`
- the hierarchical phase traces `P(BJJ.X1.XSTAGE1)` and `P(BJJ.X1.XSTAGE2)`

What to expect:

- The design is built from a reusable cell file and a higher-level subcircuit wrapper.
- The output file shows the delay through each stage.
- The fully qualified names demonstrate how internal devices are referenced once the hierarchy is unrolled.

Why this example matters: Real JoSIM®-Pro designs are rarely single flat netlists. This workflow shows how to split designs into reusable files while still inspecting internal activity when debugging or characterizing a larger circuit.

17.3 Device and model characterization with `.IV`

Problem this solves: You want to characterize a junction model directly instead of embedding it first inside a larger circuit.

Example netlist (save as `iv_run.cir`):

```
* iv_run.cir
.MODEL JJMOD JJ(IC=200u RN=15 R0=30 CAP=0.2p)
.IV JJMOD 1m jjmod_iv.csv 200
.END
```

Command to run:

```
josim-pro-cli iv_run.cir
```

What to inspect:

- the generated file `jjmod_iv.csv`
- the voltage-current sweep shape for the model under test

What to expect:

- JoSIM®-Pro generates a helper circuit internally and writes the IV data to the requested file.
- The sweep covers positive and negative current excursions and returns through zero, which is useful when checking symmetry and transition behavior.
- The `steps` argument controls the resolution of the generated curve.

Why this example matters: `.IV` is the quickest way to validate that a model has the electrical behavior you expect before using it in a larger circuit or optimization flow.

17.4 Noise, spread, and reproducibility with a fixed seed

Problem this solves: You want to introduce randomized variation and noise into a run while still being able to reproduce the same realization later.

Example netlist (save as `noise_seeded.cir`):

```
* noise_seeded.cir
.OPTION seed=123456
.SPREAD 0.05 R=0.02 B=0.03
.MODEL JJMOD JJ(IC=180u RN=12 R0=25 CAP=0.15p)

IBIAS j 0 DC 250u
B1 j 0 JJMOD icnoise=ICNOISE(0.02 1 20ps 1ps 200ps 64)
R1 j 0 2 temp=4.2 neb=20GHz

.TRAN 0.25ps 150ps
.PRINT V(j) I(B1) I(IBIAS)
.END
```

Command to run:

```
josim-pro-cli -o noise_seeded.csv noise_seeded.cir
```

What to inspect:

- the waveform file from one run to the next
- the bias point and current fluctuations in `V(j)` and `I(B1)`

What to expect:

- With the same seed, the same randomized spread realization and noise sequence are reproduced.
- Changing the seed produces a different realization while keeping the same nominal circuit.
- The resistor contributes thermal-noise behavior through `temp` and `neb`, while the junction uses an explicit `ICNOISE(...)` model for critical-current variation in time.

Practical note: When you want deterministic reruns from the command line, `-r` can be used to override the netlist seed for a particular run. The same workflow also applies to explicit source-side noise functions such as `NOISE(...)`, `PNOISE(...)`, or `FNOISE(...)` when those are more appropriate for the quantity being injected.

17.5 Variable-step and streamed output for larger runs

Problem this solves: You want to run a longer transient efficiently and avoid buffering a large result set entirely in memory.

Example netlist (save as `long_run.cir`):

```
* long_run.cir
.OPTION analysis=PHASE integration=BDF variablestep=ON
.MODEL JJMOD JJ(IC=200u RN=15 R0=30 CAP=0.2p)

VIN in 0 PULSE(0 830u 20ps 2ps 2ps 10ps 200ps)
L1 in j 2p
B1 j 0 JJMOD
R1 j 0 2

.TRAN 0.25ps 2ns
.PRINT V(in) V(j) I(B1)
.END
```

Command to run:

```
josim-pro-cli --varstep -p -o long_run.csv long_run.cir
```

What to inspect:

- the spacing of the time points in `long_run.csv`
- file generation during the run rather than only at the end

What to expect:

- Variable-step mode uses non-uniform time steps, so adjacent rows in the output file do not need to be evenly spaced in time.
- Streaming writes text output progressively, which is useful for longer runs and large waveform sets.
- This workflow is only valid for the supported `PHASE + BDF` combination.

Practical note: For streamed output, a text format such as CSV is the safest choice. When you need binary or raw output, use the normal buffered output path instead of streaming.

17.6 Derived outputs and multiple output files

Problem this solves: You want JoSIM®-Pro to write both direct waveforms and derived quantities without relying on a separate post-processing script.

Example netlist (save as `derived_outputs.cir`):

```
* derived_outputs.cir
.MODEL JJMOD JJ(IC=200u RN=15 R0=30 CAP=0.2p)
VIN in 0 PULSE(0 830u 20ps 2ps 2ps 8ps 40ps)
L1 in j 2p
B1 j 0 JJMOD
R1 j 0 2

.TRAN 0.25ps 120ps

.FILE waveforms.csv
.PRINT V(in) V(j) I(B1)

.FILE derived.csv
.PRINT V(j)-V(in) SIN(P(B1)) E(B1) W(R1)
```

```
.END
```

Command to run:

```
josim-pro-cli derived_outputs.cir
```

What to inspect:

- `waveforms.csv` for the direct simulation traces
- `derived.csv` for computed expressions and device-derived quantities

What to expect:

- The `.FILE` directive routes subsequent output requests into the selected file.
- Derived expressions such as `V(j)-V(in)` and `SIN(P(B1))` are evaluated during output generation.
- Energy and power style outputs such as `E(B1)` and `W(R1)` make it easier to inspect device behavior without exporting every raw trace first.

Why this example matters: Output expressions and multiple output files reduce friction in downstream analysis. They are especially useful when different consumers need different result sets, such as plotting tools, scripting workflows, or spreadsheet review.

17.7 Full logic-cell examples

The workflow examples above are compact and task-oriented. The three full logic-cell examples below are installed with the package. Open them from the installed `examples` directory in the GUI, or run the listed CLI commands from that same directory. Together they show both focused simulation tasks and complete cell-level designs.

17.8 Josephson Transmission Line (JTL)

Installed file: `JTL_Example.cir`. Open it from the installed examples directory with `File > Open...`, or run it from that directory with:

```
josim-pro-cli -o JTL_Example.csv JTL_Example.cir
```

The listing below shows the simulation-relevant content. The installed file may also contain `SCHM_<design>` comment blocks that preserve GUI visualizer layout; those comments do not change the circuit semantics.

```
* Josephson Transmission Line (JTL)
.SUBCKT jjbranch IN
.PARAM RSHUNT=5
BJJ     IN  1   JJMOD IC=200u
RJJ     IN  2   RSHUNT
LRJJ    2   1   0.1p
LJJP    1   0   0.1p
.MODEL JJMOD JJ(RN=15, CAP=0.1p)
.ENDS

.SUBCKT ibias OUT
.PARAM IBIAS=300u
IBIAS   0   1   DC IBIAS
LBIAS   1   OUT 3E-13
.ENDS

.SUBCKT jtl IN OUT
.PARAM LSTORE=2.0p
L1      IN  1   LSTORE
X1      1       jjbranch
L2      1   2   LSTORE
X2      2       ibias
L3      2   3   LSTORE
X3      3       jjbranch
L4      3   OUT LSTORE
.ENDS

VIN     1   0   PULSE(0 830u 50p 2.5p 2.5ps 0 50ps)
XJTL    1   2   jtl
ROUT    2   0   2

.TRAN 0.25ps 250ps
.PLOT V(VIN) P(BJJ.X1.XJTL) P(BJJ.X3.XJTL) V(ROUT)
.END
```

Description: The Josephson Transmission Line (JTL) is a key building block in superconducting circuits, allowing the efficient propagation of flux quanta. This example demonstrates a more advanced JTL design with shunt resistors, inductors, bias currents, and multiple Josephson junctions.

Explanation: This advanced JTL design is composed of the following key elements:

- **Josephson Branch (`jjbranch`)**:
  - A Josephson junction (`BJJ`) modeled by `JJMOD`.
  - A shunt resistor (`RJJ`) to maintain stability.
  - Two inductors, `LRJJ` and `LJJP`, forming part of the branch.
- **Bias Current Subcircuit (`ibias`)**:
  - Provides a steady bias current to the circuit using an ideal current source and an inductor.
- **JTL Subcircuit (`jtl`)**:
  - Multiple `jjbranch` and `ibias` subcircuits are connected with inductors (`L1`, `L2`, `L3`, `L4`) to propagate flux quanta through the line.
- **Top-Level Netlist**:
  - A pulse source (`VIN`) drives the input.
  - The output voltage is monitored across the resistor `ROUT`.

Key Simulation Insights:

- The `.TRAN` command defines a transient simulation with a step size of `0.25ps` and a total duration of `250ps`.
- The `.PLOT` command outputs:
- The input voltage at `VIN`.
- The phase across the Josephson junctions `BJJ` in the first and third branches of the JTL.
- The output voltage across the load resistor `ROUT`.

Why This Example is Useful:

- This example illustrates the construction of a superconducting transmission line using multiple Josephson junctions and bias currents.
- It demonstrates how to organize complex circuits into hierarchical subcircuits.
- It shows how to reference devices within nested subcircuits using full paths such as `BJJ.X1.XJTL`.

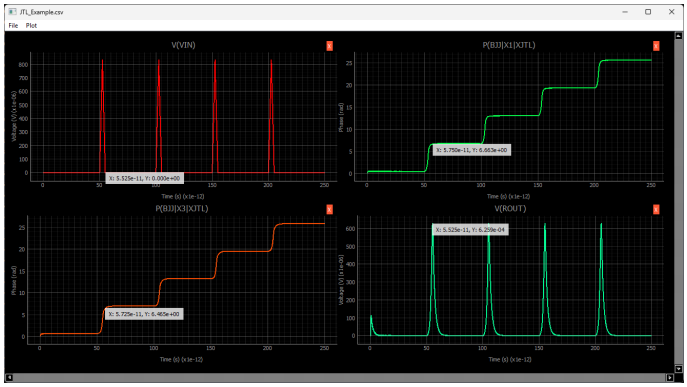


Figure 17.1: JTL example output traces from the installed example

---

17.9 RSFQ Splitter Cell

Installed file: `Splitter_Example.cir`. Open it from the installed examples directory with `File > Open...`, or run it from that directory with:

```
josim-pro-cli -o Splitter_Example.csv Splitter_Example.cir
```

```
* RSFQ Splitter Cell
.SUBCKT jjbranch IN
.PARAM Rshunt=5
BJJ     IN  1   JJMOD IC=200u
RJJ     IN  2   Rshunt
LRJJ    2   1   0.1p
LJJP    1   0   0.1p
.MODEL JJMOD JJ(RN=15, CAP=0.1p)
.ENDS

.SUBCKT ibias OUT
.PARAM IBias=300u
IBias   0   1   DC Ibias
LBias   1   OUT 3E-13
.ENDS

.SUBCKT splitter a Q0 Q1
LA      a   6   0.3E-12
XJJ1    6       jjbranch    BJJ=100u
XBIAS1  6       ibias       IBIAS=125u
L1      6   7   1.5E-12
XJJ2    7       jjbranch    BJJ=175u
L2      7   18  3E-12
XBIAS2  18      ibias       IBIAS=350u
L3      18  19  0.5E-12
L4      4   19  1.3E-12
XJJ3    4       jjbranch    BJJ=125u
XBIAS3  5       ibias       IBIAS=75u
L5      4   5   1.3E-12
XJJ4    5       jjbranch    BJJ=175u
LQ0     5   q0  2.2E-12
L6      19  8   1.3E-12
XJJ5    8       jjbranch    BJJ=125u
XBIAS4  9       ibias       IBIAS=75u
L7      8   9   1.3E-12
XJJ6    9       jjbranch    BJJ=175u
LQ1     9   q1  2.2E-12
.ENDS

VIN     a 0  PULSE(0 830u 50p 2.5p 2.5ps 0 50ps)
Xsplit  a Q0 Q1 splitter
R1      Q0 0 2
R2      Q1 0 2

.TRAN 0.25ps 250ps
.PLOT V(VIN) V(R1) V(R2)
.END
```

Description: This example demonstrates a splitter cell based on RSFQ technology, which splits the input signal into two outputs. The circuit uses the same **Josephson Junction branch** (`jjbranch`) and **bias current subcircuit** (`ibias`) in multiple parts of the design, allowing for efficient parameterization and reusability.

Explanation: This RSFQ splitter design is composed of the following key elements:

- **Josephson Branch (`jjbranch`)**:
  - A Josephson junction (`BJJ`) modeled by `JJMOD`.
  - A shunt resistor (`RJJ`) for stability.
  - Two inductors, `LRJJ` and `LJJP`, forming the Josephson branch.
- **Bias Current Subcircuit (`ibias`)**:
  - Provides steady bias current to the circuit using an ideal current source and an inductor.
- **Splitter Subcircuit (`splitter`)**:
  - This subcircuit uses multiple instances of `jjbranch` and `ibias` subcircuits to build a splitter that routes the input signal `a` into two outputs `Q0` and `Q1`.
  - The inductors `LA`, `L1`, `L2`, `L3`, and `LQ0`, `LQ1` form the core of the splitting operation.
- **Top-Level Netlist**:
  - A pulse source (`VIN`) drives the input.
  - The output voltages are monitored across resistors `R1` and `R2`.

Key Simulation Insights:

- The `.TRAN` command defines a transient simulation with a step size of `0.25ps` and a total duration of `250ps`.
- The `.PLOT` command outputs:
  - The input voltage at `VIN`.
  - The output voltage at `R1` and `R2`, corresponding to `Q0` and `Q1`.

Why This Example is Useful:

- This example illustrates how to efficiently reuse subcircuits by parameterizing them (e.g., modifying Josephson junction critical current `BJJ` and bias current `IBIAS` for different instances).
- It demonstrates how to organize complex circuits into hierarchical subcircuits, simplifying the design.
- It shows how to split a signal in RSFQ logic, a fundamental operation in superconducting digital circuits.

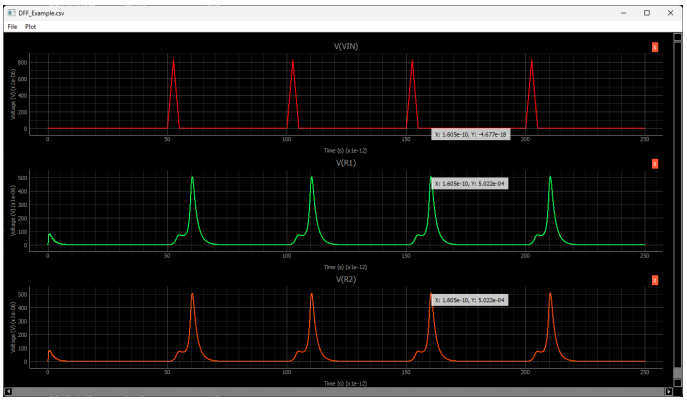


Figure 17.2: Splitter example output traces from the installed example

17.10 RSFQ AND Gate

Installed files: `AND_Example.cir` and `base_cells.cir`. Keep both files in the same directory, because the AND netlist includes `base_cells.cir`. Open `AND_Example.cir` from the installed examples directory with `File > Open...`, or run it from that directory with:

```
josim-pro-cli -o AND_Example.csv AND_Example.cir
```

```
* RSFQ AND Cell
.include base_cells.cir

.subckt and a b clk q
LA      a   1   2p
XJJ1    1       jjbranch
XBIAS1  1       ibias       IBIAS=175u
L1      1   2   3p
B1      2   4   JJMOD       IC=180u
RB1     2   3   4
LRB1    3   4   2.2p
XJJ2    4       jjbranch    IC=250u
XBIAS2  4       ibias       IBIAS=175u
L2      4   5   10p
XJJ3    5       jjbranch    IC=250u
B2      5   7   JJMOD       IC=180u
RB2     5   6   4
LRB2    6   7   2.2p
L3      7   8   1p
L4      5   9   3p
B3      9   11  JJMOD       IC=180u
RB3     9   10  4
LRB3    10  11  2.2p
LB      b   12   2p
XBIAS3  12      ibias       IBIAS=175u
XJJ4    12      jjbranch    IC=250u
L5      12  13  3p
B4      13  15  JJMOD       IC=180u
RB4     13  14  4
LRB4    14  15  2.2p
XJJ5    15      jjbranch    IC=250u
XBIAS4  15      ibias       IBIAS=175u
L6      15  16  10p
B5      16  18  JJMOD       IC=180u
RB5     16  17  4
LRB5    17  18  2.2p
L7      18  8   1p
XJJ6    16      jjbranch    IC=250u
L8      16  19  3p
B6      19  11  JJMOD       IC=180u
RB6     19  20  4
LRB6    20  11  2.2p
LCLK    clk 21  2p
XJJ7    21      jjbranch    IC=250u
XBIAS5  21      ibias       IBIAS=175u
L9      21  22  3p
XJJ8    22      jjbranch    IC=250u
XBIAS7  22      ibias       IBIAS=175u
L10     22  8   1p
L11     11  23  1p
XJJ9    23      jjbranch    IC=250u
XBIAS6  23      ibias       IBIAS=175u
LQ      23  q   2p
.ends

VCLK    clk 0   PULSE(0 830u 50p 2.5p 2.5ps 0 50ps)
VA      a   0   PWL(0 0 80p 0 82.5p 830u 85p 0 180p 0 182.5p 830
    u 185p 0)
VB      b   0   PWL(0 0 130p 0 132.5p 830u 135p 0 180p 0 182.5p
    830u 185p 0)
XAND    a   b   clk     q     and
RQ      q   0   2

.TRAN 0.125ps 250ps
.PLOT V(VA) V(VB) V(VCLK) V(RQ)
.END
```

Description: This example demonstrates a basic RSFQ AND gate. The AND logic is achieved by using Josephson junctions and bias currents to combine two inputs. The design showcases the reuse of subcircuits like `jjbranch` and `ibias` from the included base file (`base_cells.cir`).

Explanation: This RSFQ AND gate design includes the following key elements:

- **Josephson Branch (`jjbranch`):**
  - A Josephson junction modeled by `JJMOD`.
  - A shunt resistor and inductors forming the Josephson branch.
- **Bias Current Subcircuit (`ibias`):**

  - Provides a steady bias current using an ideal current source.
- **AND Gate Subcircuit (and)**:
  - This subcircuit combines the two inputs, `a` and `b`, and processes them through Josephson junctions and bias currents to output a logic AND at node `q`.
  - The inputs are routed through the inductors (`L1`, `L2`, `LA`, etc.) and the Josephson junction branches.

Key Simulation Insights:

- The `.TRAN` command defines a transient simulation with a step size of `0.125ps` and a total duration of `250ps`.
- The `.PLOT` command outputs:
  - The input voltages at `VA` and `VB`.
  - The clock signal at `VCLK`.
  - The output signal at `RQ`.

Why This Example is Useful:

- This example demonstrates the construction of a rudimentary RSFQ AND gate.
- It showcases the reuse of subcircuits and the inclusion of external files using the `.include` command.
- It provides a basic example of how logic gates are implemented in RSFQ technology, demonstrating the potential for building more complex logic circuits.

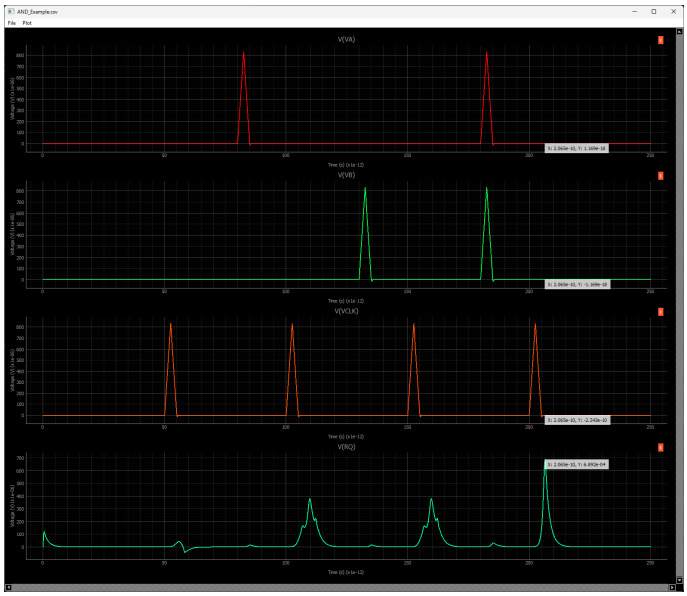


Figure 17.3: RSFQ AND gate output traces from the installed example

17.11 Summary

This chapter combines two kinds of examples. The workflow-oriented snippets at the beginning are manual-only starting points that you can copy into your own working directory. The full logic-cell examples at the end are installed files and should be used when the manual asks for a packaged example. Additional examples outside the installed list are not assumed to be present after installation.

Simulation Feature Reference

JoSIM®-Pro is a professional superconducting circuit simulator designed for practical circuit design, analysis, and characterization. The software combines a SPICE-like netlist flow with device models and analysis features tailored to superconducting electronics. This chapter summarizes the main capabilities of JoSIM®-Pro and explains how they fit into typical simulation workflows.

18.1 Core Simulation Capabilities

At its core, JoSIM®-Pro is built for transient simulation of superconducting circuits. Users describe a circuit in a SPICE-like netlist, choose the desired analysis settings, and generate output traces for inspection, plotting, or post-processing.

Key simulation capabilities include:

- **Transient simulation**: Simulate switching, biasing, clock propagation, pulse interaction, and waveform evolution in the time domain.
- **Phase and voltage workflows**: Run in either phase mode or voltage mode depending on the circuit and the quantity of interest.
- **Multiple integration methods**: Choose between backward differentiation and trapezoidal integration.
- **Fixed-step and variable-step operation**: Run with a fixed print cadence or enable adaptive stepping for the currently supported subset of problems. In the current JoSIM®-Pro v2.9.0 release, variable-step mode is supported for `PHASE + BDF`.
- **Practical startup control**: Use the default consistent start at $t = 0$, preserve user-specified initial conditions with `UIC`, or enable legacy startup priming when a workflow benefits from it.
- **Large-circuit support**: Work with hierarchical designs, reusable subcircuits, multiple output targets, and streamed text output for long or memory-intensive runs.

These capabilities make JoSIM®-Pro suitable for both early-stage design exploration and detailed verification of larger superconducting systems. The Usage chapter explains how to select these modes from the CLI, while the Syntax chapter documents the netlist commands that capture the same intent inside a circuit file.

18.1.1 Choosing Between Phase and Voltage Workflows

`PHASE` mode is the default operating mode in JoSIM®-Pro and is the natural choice for most superconducting transient studies, especially when the workflow depends on Josephson-junction phase evolution, SFQ-style pulse propagation, or variable-step operation. `VOLTAGE` mode remains available when a voltage-domain representation is the better fit for the circuit under study or for the way results need to be reported.

In practice, a good rule is to start in `PHASE` mode unless there is a specific reason to prefer voltage-domain solving. The selected analysis mode affects both the solved state representation and the default meaning of output requests. Users can choose the mode from the CLI with `-a`/`--analysis` or from the netlist with `.OPTION analysis=...`.

18.1.2 Fixed-Step and Variable-Step Operation

Fixed-step runs are the straightforward choice when a workflow needs uniformly spaced samples, easy run-to-run comparison, or a known print cadence for downstream tools. Variable-step runs are more appropriate when switching events are localized in time, long quiet intervals would otherwise be oversampled, or the goal is to reduce point count while still refining around fast dynamics.

The current JoSIM®-Pro v2.9.0 release supports variable-step mode only for `PHASE + BDF`. That means variable-step is not the right choice for voltage-mode runs or trapezoidal integration. For most circuits, it is best to begin with the default adaptive settings and only tune the detailed `.OPTION` controls if a specific circuit requires it.

18.2 Circuit Description and Modelling

18.2.1 SPICE-like Netlists, Parameters, and Hierarchy

JoSIM®-Pro uses a SPICE-like netlist syntax so that circuit definitions remain compact, readable, and easy to version-control. Users can define scalar parameters with `.PARAM`, reuse common content with `.INCLUDE`, and structure larger designs with subcircuits.

```
.param scalar=2
.param R1=50*scalar
.param L1=10E-9*0.25*scalar
R1 1 2 R1
L1 2 0 L1
```

This style of netlisting is useful for design scaling, design-space exploration, and maintaining repeated constants across larger projects.

18.2.2 Sources and Circuit Elements

The simulator supports the component and source models needed to build practical superconducting circuits, including Josephson junctions, resistors, inductors, capacitors, transmission lines, mutual inductive coupling, independent voltage/current/phase sources, and the standard dependent source families.

Independent sources can be driven with a broad set of built-in waveform definitions, including:

- **DC** values,
- **PWL** and **PWS** piecewise waveforms,
- **PULSE**, **SIN**, **EXP**, and **STAIR** excitations,
- **CUS** waveforms loaded from external text files,
- **NOISE**, **PNOISE**, and **FNOISE** for stochastic excitation.

This allows the same simulator to cover straightforward bias setups, pulse-driven logic experiments, external waveform replay, and noise-driven studies.

18.2.3 Josephson Junction Models

JoSIM®-Pro supports more than one Josephson-junction model family. Standard `JJ` and `RCSJ` models cover the usual resistively and capacitively shunted workflow, while `TJM` models support tunnel-junction modelling with fitted parameter sets.

For practical modelling work, the code supports:

- standard junction parameters such as critical current, normal resistance, subgap resistance, capacitance, and gap voltage,
- temperature-dependent RCSJ behaviour through `T` and `TC`,
- transition shaping with `DELV` and `RTYPE`,
- phase-offset and non-sinusoidal current-phase relationships through `PHI` and `CPR`,
- TJM fit data supplied inline or loaded from an external file.

This gives users enough modelling range to move from simple circuit studies to more specialised device characterisation work without changing tools.

18.3 Noise, Spread, and Realism

18.3.1 Thermal Noise

JoSIM®-Pro can include Johnson–Nyquist noise through global `.TEMP` and `.NEB` commands, and resistive devices and junctions can also override those settings locally with `temp=` and `neb=` attributes when needed.

```
.temp 4.2
.neb 10GHz
R1 1 2 50 temp=4.2 neb=10GHz
B1 2 3 JJMOD temp=4.2 neb=10GHz
```

This makes it possible to move from idealized switching studies to temperature- and bandwidth-aware transient simulations.

18.3.2 Explicit Noise Waveforms and Junction Noise

In addition to thermal noise, JoSIM®-Pro supports explicit stochastic waveform functions for source-driven studies and device-level perturbation:

- `NOISE(...)` for sampled white-noise style excitation,
- `PNOISE(...)` for $1/f^{\alpha}$-style amplitude noise,
- `FNOISE(...)` for flux-noise style excitation referenced to Φ_0,
- `ICNOISE(...)` on Josephson junctions for fractional critical-current noise.

```
I1 1 0 PNOISE(1u 1 0 1p 10n 64)
P1 2 0 FNOISE(1e-6 1 0 0 1p 10n 64)
B1 2 3 JJMOD ICNOISE=ICNOISE(0.01 1 0 1p 10n 64)
```

These capabilities are useful when characterising jitter sensitivity, low-frequency drift, or noise-limited switching behaviour.

18.3.3 Parameter Spread and Reproducibility

Manufacturing variation and device mismatch are often just as important as nominal behaviour. JoSIM®-Pro supports both global spread controls and per-component spread attributes.

```
.param R1=50
.param L1=10E-9
R1 1 2 R1 spread=0.05
L1 2 0 L1
.spread 0.02 R=0.05 B=0.10
```

This is useful when evaluating robustness, estimating yield, or understanding sensitivity to fabrication tolerances. When reproducibility matters, the random seed can be fixed from the CLI or from `.OPTION seed=...`.

18.4 Characterization and Analysis Workflows

Use transient, IV and variation workflows to inspect different aspects of circuit behavior.

18.4.1 Transient and Startup Workflows

The normal workflow in JoSIM®-Pro is transient analysis driven by `.TRAN`. Within that flow, users can choose how the run is initialized:

- use the default consistent start at $t = 0$,
- apply explicit initial conditions with `.IC`,
- use `UIC` when user-specified initial conditions should be taken directly,
- enable `STARTUP` when a legacy startup-priming flow is desired.

This makes the transient engine practical for both clean nominal runs and more specialised startup or state-initialisation studies.

18.4.2 IV Curve Generation

JoSIM®-Pro can generate IV data directly from a model definition. This is useful for characterising junction behaviour, validating model parameters, and checking expected operating ranges before a device is placed into a larger circuit.

```
.iv JJMOD 300E-6 IV.csv 400
```

This command sweeps the named model to $\pm 300\mu\mathrm{A}$ and stores the resulting current–voltage data in `IV.csv`. Because the workflow is model-based, it is useful both as a first-pass device check and as a way to compare fitted model behaviour before running a larger transient simulation.

18.4.3 Variation, Noise, and Repeatability Studies

Characterisation in JoSIM®-Pro is not limited to a single nominal transient run. Users can combine `.SPREAD`, explicit source or device noise, fixed random seeds, and expression-based outputs to study robustness, sensitivity, and repeatability.

A typical workflow is to begin with a nominal run, then introduce spread or noise one step at a time while keeping the seed fixed so that results remain comparable across revisions. This is particularly useful when evaluating switching margins, timing sensitivity, or the stability of a design under realistic perturbations.

18.4.4 Derived Outputs and Diagnostics

Output requests are not limited to raw node traces. Users can request voltages, currents, phases, apparent power, and accumulated energy, and may also build expressions directly in output statements.

```
.print V(R1) + V(R2)
.print sin(P(B01))
.print W(R1) E(B01)
```

This is useful when a workflow depends on derived traces, compact diagnostics, or direct reporting of power- and energy-related quantities without extra post-processing.

18.5 Output, Post-Processing, and Data Handling

18.5.1 Flexible Output Formats

Simulation results can be written in multiple formats depending on the downstream workflow:

- **CSV and DAT**: convenient for plotting and importing into analysis tools such as Python, MATLAB, or spreadsheets,
- **BIN**: compact binary output for offline workflows,
- **RAW**: extensionless raw output for minimal-overhead file generation.

Text formats can also be compressed in a GZIP container when file size matters.

18.5.2 Streaming Output and Multiple Files

For long simulations or large output sets, JoSIM®-Pro can stream text output during the run rather than waiting until the end of the simulation. In practice, streaming is intended for `CSV`, `DAT`, and `STDOUT` workflows. Offline generation remains the practical choice when `BIN` or `RAW` output is required.

When streaming is enabled, file-format choice becomes part of the workflow design. If a downstream tool requires exact `BIN` or `RAW` output, keep streaming disabled and let the run complete in buffered mode. If the priority is lower peak memory usage or immediate text output during a long run, choose streamed `CSV`, `DAT`, or `STDOUT` instead.

The `.FILE` command allows output to be split across multiple files, which is useful when separating waveform groups or organizing results by function. The same format rules apply per target, so a multi-file workflow can mix logical output groupings while still following the runtime rules for streaming versus buffered output.

18.5.3 Hierarchical Referencing and Expressions

When working with hierarchical circuits, JoSIM®-Pro can reference signals inside nested subcircuits using fully qualified hierarchical names. Output statements can also include mathematical expressions built from supported signal references.

This allows large designs to remain modular while still exposing the exact internal traces needed for debugging, verification, and reporting.

18.6 Summary

JoSIM®-Pro combines superconducting-device simulation, hierarchical netlist support, multiple Josephson-junction model paths, thermal and explicit noise modelling, spread analysis, IV characterization, and flexible output handling in a single workflow. Together, these capabilities make it suitable for research, design exploration, and engineering use where accurate time-domain simulation of superconducting circuits is required.

For day-to-day execution choices such as analysis mode selection, output destinations, streaming, and reproducibility controls, see the Usage chapter. For the exact netlist forms of `.TRAN`, `.IC`, `.OPTION`, `.FILE`, output expressions, and `.IV`, see the Syntax chapter.

Python Automation

The `josimpro` module exposes the JoSIM®-Pro simulation engine for scripted and repeatable workflows. In practice, the Python layer is most useful when you want to run the same circuit repeatedly, inspect results in NumPy, or export traces directly from a script.

This chapter is organized around normal user tasks:

1. make the module importable,
2. configure process-wide settings such as the license path,
3. load a circuit file,
4. adjust runtime settings for a run,
5. execute the simulation,
6. inspect results in Python, and
7. write output files for later analysis.

The netlist syntax, analysis modes, print directives, and output behavior are the same ones described in the Usage, Syntax, Features, and Examples chapters. The Python layer does not define a separate circuit language.

19.1 Making the module available

The Python extension module is named `josimpro`. Product installs place the compiled module in the JoSIM®-Pro binary location. The helper scripts in `utils/install_josimpro_python.sh` and `utils/install_josimpro_python.bat` add that binary directory to the Python user site-packages path by writing a `josimpro.pth` file.

If you are using a custom installation layout, make sure the directory containing the compiled `josimpro` module is on `PYTHONPATH` before importing it.

A quick import and version check looks like this:

```
import josimpro as jp

print("Version:", jp.VERSION)
print("Built:", jp.BUILD_TIMESTAMP)
print("Git:", jp.GIT_HASH)
```

19.2 Process-wide settings and licensing

`jp.Settings` holds process-wide settings. These are not the per-run simulation controls. In the current Python binding, the main process-wide settings are:

- `Settings.verbosity`
- `Settings.minOutput`
- `Settings.license`
- `Settings.licensee` (read-only)
- `Settings.validity` (read-only)
- `Settings.endUse` (read-only)

A typical setup block is:

```
import josimpro as jp

jp.Settings.license = "/path/to/sm_license.txt"
jp.Settings.verbosity = 1
jp.Settings.minOutput = False
```

Use `Settings` for process-wide presentation and licensing state. Use `Runtime` for the settings that control a specific simulation run.

19.3 Smallest useful file-based run

The simplest Python workflow is file-based. Construct an `Input` from a circuit file, build a `Matrix`, then run a `Simulation`:

```
import josimpro as jp

jp.Settings.license = "/path/to/sm_license.txt"

inp = jp.Input("JTL_Example.cir")
mat = jp.Matrix(inp)
mat.validate_simulation_ready()

sim = jp.Simulation(mat)
sim.simulate(mat)

print("Simulation OK:", sim.isSimOK)
print("Accepted steps:", sim.stepCount)
```

The `Input("file")` constructor reads and parses the circuit immediately. This is the most convenient entry point when the netlist already contains the `.TRAN`, `.OPTION`, and `.PRINT` directives you want to use.

19.4 Runtime control from Python

Use `jp.Runtime` when you want to set or inspect run-specific controls from Python.

```python
import josimpro as jp

rt = jp.Runtime()
# also available:
rt2 = jp.Runtime.fromGlobalSettings()
```

In the current release, `Runtime.fromGlobalSettings()` is effectively a convenience constructor for the default runtime object. It does not copy the process-wide `Settings` fields such as license location or verbosity.

Two details matter immediately:

1. the Python binding currently exposes analysis and integration selection as integer fields, not Python enums;
2. unlike the CLI, the Python layer does not expose separate override masks for `.TRAN` and `.OPTION`. If the netlist sets the same field during parse, the netlist value remains authoritative.

The integer mappings are:

- `analysisType`: `0 = PHASE`, `1 = VOLTAGE`
- `integrationMethod`: `0 = BDF`, `1 = TRAP`
- `exprDivZeroPolicy`: `0 = ReturnZero`, `1 = ClampToMax`, `2 = NaN`, `3 = Error`

The runtime fields most users adjust from Python are:

- `inputFile`, `workingDir`
- `tStep`, `tStop`, `pStart`, `pStep`, `useUIC`
- `analysisType`, `integrationMethod`, `variableStepEnabled`
- `newtonJJEnabled`, `newtonMaxIters`, `newtonRelTolPhase`, `newtonAbsTolPhase`, `newtonDamping`
- `outputPrecision`, `compressedOutput`, `streamOutput`
- `globalTemp`, `globalNeb`, `globalSpread`, `globalRSpread`, `globalCSpread`, `globalLSpread`, `globalBSpread`
- `seed`

The module also exposes the full supported runtime field list through `jp.PUBLIC_RUNTIME_FIELDS`.

Important: variable-step mode is currently supported only for `PHASE + BDF`. In Python, that means `analysisType = 0`, `integrationMethod = 0`, and `variableStepEnabled = True`. Other combinations are rejected by the core runtime checks.

19.5 Explicit runtime example

When you want to set the run configuration from Python before reading the file, create an `Input` manually and attach the runtime first:

```python
import josimpro as jp

rt = jp.Runtime()
rt.inputFile = "JTL_Example.cir"
rt.workingDir = "."
rt.analysisType = 0          # PHASE
rt.integrationMethod = 0     # BDF
rt.variableStepEnabled = False
rt.outputPrecision = 12
rt.seed = 12345

inp = jp.Input()
inp.runtime = rt
inp.read_input()
inp.parse_input()

mat = jp.Matrix(inp)
mat.validate_simulation_ready()

sim = jp.Simulation(mat)
sim.simulate(mat)
```

This pattern is the safest starting point for scripted automation because the runtime is explicit and the input file path is tied to that runtime object.

19.6 Reading results into NumPy

After a run, `sim.results` provides NumPy-friendly access to the stored solution.

The two most useful helpers are:

- `results.get_time()` — returns the stored time vector as a NumPy array
- `results.get_x_full(matrix)` — returns the stored solver state as a 2D NumPy array

Example:

```python
import josimpro as jp
import numpy as np

inp = jp.Input("JTL_Example.cir")
mat = jp.Matrix(inp)
sim = jp.Simulation(mat)
sim.simulate(mat)

time = sim.results.get_time()
x = sim.results.get_x_full(mat)

print(type(time), time.shape)
print(type(x), x.shape)
```

The array returned by `get_x_full(matrix)` is the stored solver state. Its rows follow the matrix column ordering used internally for the simulation, or the relevant filtered subset when output pruning is active. When you want named traces that follow the circuit's `.PRINT` plan, build an `Output` object and evaluate the print list instead.

19.7 Getting named traces from the print list

`Output(inp, mat)` parses the same `.PRINT` and `.FILE` directives used by the command-line flow. This is the easiest way to obtain named traces in Python.

```python
import josimpro as jp
import numpy as np

inp = jp.Input("JTL_Example.cir")
mat = jp.Matrix(inp)
sim = jp.Simulation(mat)
sim.simulate(mat)

out = jp.Output(inp, mat)
out.format_time_vector(sim.results, mat)
out.compute_all_print_values(mat)

traces = {
    p.name: np.asarray(p.values, dtype=float)
    for p in out.plist
    if not p.hidden
}

for name, values in traces.items():
    print(name, values.shape)
```

Use this route when you want the same signal naming and derived-expression behavior that JoSIM®-Pro would produce from the netlist itself.

19.8 Transient measurements

For a deck containing `.MEASURE TRAN` commands, each explicit analysis-job result exposes a `measurements` list. Values remain available when `retain_results=False`:

```python
jobs = jp.AnalysisJobs(inp, retain_results=False)
jobs.run()
for run in jobs.results:
    for metric in run.measurements:
        print(run.id, metric.name, metric.value, metric.unit)
```

In the low-level buffered workflow, construct `out = jp.Output(inp, mat)` before simulation so measurement signals are retained. After `sim.simulate(mat)`, call `out.compute_measurements(sim, mat)` and read `out.measurements`. This evaluates measurements without writing waveform files. `out.print_output(sim, mat)` also evaluates them. See Section 16.7.9 for syntax, interpolation and failure behavior.

19.9 Parameter sweeps

For a deck containing `.STEP`, use `AnalysisJobs`. Each job result includes ordered `(name, value)` pairs in `parameters`, its actual `seed`, output files and per-case measurements:

```python
jobs = jp.AnalysisJobs(inp, retain_results=False)
jobs.run()
for run in jobs.results:
    print(run.id, run.parameters, run.seed)
    for metric in run.measurements:
        print(metric.name, metric.value, metric.unit)
```

Assignments are available after planning; measurements require successful execution. Unswept jobs have an empty `parameters` list. See Section 16.7.7 for ordering, limits, filenames and seed policy.

Configure bounded parallel execution on Linux:

```python
MiB = 1024 * 1024
jobs = jp.AnalysisJobs(inp, retain_results=False)
jobs.set_parallel(max_workers=4,
                  memory_budget_bytes=2048 * MiB,
                  worker_memory_bytes=256 * MiB)
jobs.run()
```

`run()` releases the Python GIL. Each case executes in a matching native CLI worker; this does not require Python multiprocessing. The maximum worker count, reservation budget, per-worker virtual-memory limit, supported decks and cancellation/publication rules are described in Section 16.7.8. Retaining waveforms reserves space for all case results before dispatch; use a positive print cadence when appropriate.

19.10 Pausing and saving a simulation

The low-level `Simulation` API can pause a buffered fixed-step lumped circuit and save it for a later process. PHASE and VOLTAGE analysis with BDF or TRAP integration support R/L/C, mutual inductances, JJs and independent or controlled sources. Existing model restrictions apply; TJM requires PHASE with BDF. Adaptive stepping, transmission lines, streaming, `AnalysisJobs` plans and GUI pause controls are outside this interface.

Choose an explicit nonzero seed before building the matrix. Construct the `Output` plan before stepping so printed and measured signals are retained. For a deck with a stop time greater than 50 ps:

```python
inp = jp.Input("circuit.cir")
rt = jp.Runtime()
rt.seed = 12345
mat = jp.Matrix(inp, rt)
out = jp.Output(inp, mat)
sim = jp.Simulation(mat)
sim.simulate_until(mat, 50e-12)
if sim.paused:
    sim.save_checkpoint(mat, "run-001.checkpoint")
```

The pause occurs at the first accepted sample at or after the requested time; it does not split a step or change the stop time. A request must lie in `[0, tStop)`. If the next accepted sample completes the run, `paused` is false. `request_pause()` can request an earlier pause from another thread while `simulate_until` or `resume` is running. Wait for stepping to return before querying or saving state.

In a later process, rebuild the same circuit, runtime settings, seed and output relevance plan with the same core binary and platform:

```
inp = jp.Input("circuit.cir")
rt = jp.Runtime()
rt.seed = 12345
mat = jp.Matrix(inp, rt)
out = jp.Output(inp, mat)
sim = jp.Simulation(mat)
sim.load_checkpoint(mat, "run-001.checkpoint")
sim.resume(mat)
out.compute_measurements(sim, mat)
out.print_output(sim, mat)
```

Loading requires fresh matrix/simulation owners and replaces parsed-matrix views. Keep these owners intact while paused. Use `resume(mat, time)` to pause again at a later time; `simulate(mat)` starts from the beginning. Cancellation remains separate from pause.

Each save requires a new filename on a filesystem supporting hard links. Existing files are never replaced. The checkpoint payload is limited to 256 MiB and includes the retained trajectory, device/integrator histories, noise state and continuation position. No waveform files are opened by checkpoint stepping. After completion, ordinary output and measurement calls operate on the complete trajectory. Saved samples and times are preserved exactly; loading rebuilds the linear solver factors, so bitwise identity of future values is not guaranteed. Checkpoints are specific to the exact core binary and are not portable between builds.

19.11 Writing CSV, BIN, RAW, or normal print output

The `Output` helpers let a Python script write the same kinds of files discussed in the output chapters.

The simplest choice is:

```
out = jp.Output(inp, mat)
out.print_output(sim, mat)
```

That uses the parsed print plan and whatever output routing follows from the netlist and runtime state.

When you want explicit file names from Python, first format the output time vector, then call the file writer you need:

```
import josimpro as jp

inp = jp.Input("JTL_Example.cir")
mat = jp.Matrix(inp)
sim = jp.Simulation(mat)
sim.simulate(mat)

out = jp.Output(inp, mat)
out.format_time_vector(sim.results, mat)

out.print_CSV_DAT(',', "jtl_run.csv", mat)
out.print_BIN("jtl_run.bin", mat)
out.print_RAW("jtl_run.raw", mat)
```

Use `print_CSV_DAT(',', ...)` for CSV and `print_CSV_DAT(' ', ...)` for DAT-style text output.

Practical note: `Runtime.outputFiles` is exposed read-only in Python as a record of the active parsed output targets. If you want to choose file names directly from Python, the explicit `print_*` helpers shown above are the straightforward option.

19.12 Static operating points

Execute a saved deck containing a standalone `.OP` through the shared job interface:

```
import josimpro as jp

inp = jp.Input("bias.cir")
jobs = jp.AnalysisJobs(inp, True)
jobs.run()
assert jobs.kind(0) == jp.AnalysisJobKind.operating_point
for value in jobs.results[0].measurements:
    print(value.name, value.value, value.unit)
assert jobs.results[0].axis == []
assert jobs.results[0].traces == []
```

The default CSV report is separate from the input file. Retained results contain named voltage, phase, current and convergence scalars, without a time axis. Use the supported circuit and equilibrium rules in Section 16.7.3. Low-level transient simulation calls do not execute `.OP`.

19.13 Complex AC results

For a standalone AC deck (Section 16.7.5), retain a detached complex result rather than constructing a real `Output`:

```
import josimpro as jp

jobs = jp.AnalysisJobs(jp.Input("response.cir"), True)
jobs.run()
assert jobs.kind(0) == jp.AnalysisJobKind.ac
ac = jobs.ac_result(0)
frequency = ac.frequency_hz  # Hz, float64 copy
values = ac.values(0)        # complex128 copy
print(ac.trace_name(0), ac.trace_unit(0), ac.metadata)
view = ac.view("V(OUT)/V(IN)", unwrap=True, db_reference="1")
ac.save("response-copy.csv.gz")
del jobs
reopened = jp.AcResult.load("response-copy.csv.gz")
```

View operands must have been retained by `.PRINT AC`, unless the requested ratio itself was retained. Views return validity arrays, nullable angles in degrees, and an explicit reference/status for requested dB values. Physical `P(...)` remains in radians. Modifying a returned NumPy array does not modify the retained result. Metadata and JSON preserve source, stimulus, seed and the declared bias/model limits. Real-signal FFT, integration, energy and transient measurement interfaces do not accept this complex owner.

19.14 Large runs, streaming, and variable-step restrictions

For runs where you want to use the streaming-capable output path, pass an `Output` object to `simulate()`:

```
import josimpro as jp

rt = jp.Runtime()
rt.inputFile = "JTL_Example.cir"
rt.analysisType = 0          # PHASE
rt.integrationMethod = 0     # BDF
rt.variableStepEnabled = True
rt.streamOutput = True

inp = jp.Input()
inp.runtime = rt
inp.read_input()
inp.parse_input()

mat = jp.Matrix(inp)
out = jp.Output(inp, mat)
sim = jp.Simulation(mat)
sim.simulate(mat, out)
```

This is the Python-side route into the streaming-aware output path.

Keep these restrictions in mind:

- variable-step mode currently supports `PHASE + BDF` only,
- the same `.PRINT`, `.FILE`, and output-format rules from the Usage and Syntax chapters still apply,
- if the netlist sets conflicting `.TRAN` or `.OPTION` values, those directives still govern the parsed run.

19.15 Repeated runs and batch automation

Python is most valuable when you need to run the same circuit repeatedly with controlled changes. A simple example is exporting one output file per random seed:

```
import josimpro as jp

for seed in (1001, 1002, 1003):
    rt = jp.Runtime()
    rt.inputFile = "JTL_Example.cir"
    rt.seed = seed
    rt.outputPrecision = 12

    inp = jp.Input()
    inp.runtime = rt
    inp.read_input()
    inp.parse_input()

    mat = jp.Matrix(inp)
    sim = jp.Simulation(mat)
    sim.simulate(mat)

    out = jp.Output(inp, mat)
    out.format_time_vector(sim.results, mat)
    out.print_CSV_DAT(',', f"jtl_seed_{seed}.csv", mat)
```

The same structure works for sweeps over output precision, analysis mode, spread settings, temperature, Newton settings, or other runtime fields that are already available on `Runtime`.

19.16 End-to-end plotting example

The following script runs a circuit, pulls the stored solver state into NumPy, and plots every stored row. It is intentionally simple; for publication-quality plots you will usually select a smaller set of named traces from `Output.plist` instead.

```
import josimpro as jp
import matplotlib.pyplot as plt

rt = jp.Runtime()
rt.inputFile = "JTL_Example.cir"
rt.analysisType = 0          # PHASE
rt.integrationMethod = 0     # BDF
rt.variableStepEnabled = True

inp = jp.Input()
inp.runtime = rt
inp.read_input()
inp.parse_input()

mat = jp.Matrix(inp)
sim = jp.Simulation(mat)
sim.simulate(mat)

time = sim.results.get_time()
y = sim.results.get_x_full(mat)

fig, axes = plt.subplots(y.shape[0], 1, sharex=True)
for i, ax in enumerate(axes):
    ax.plot(time, y[i])
plt.show()
```

Which produces the resulting plot:

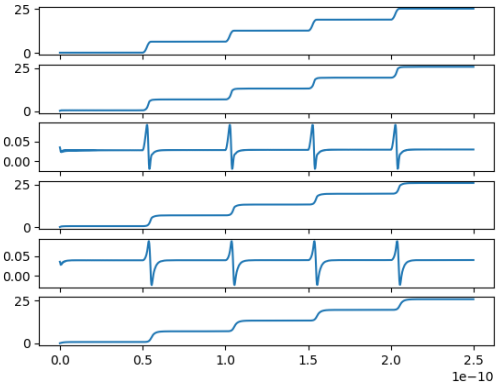


Figure 19.1: JTL example simulated and plotted using the JoSIM®-Pro Python interface

19.17 Practical reminders

For day-to-day Python use, the main points are:

- use `Settings` for licensing and process-wide presentation state,
- use `Runtime` for run-specific controls,
- prefer `Input("file") -> Matrix -> Simulation` for the simplest scripts,
- use `Output` when you want named traces or exported files,
- treat `analysisType` and `integrationMethod` as integer fields in the current binding,
- remember that variable-step mode requires `PHASE + BDF`, and
- remember that netlist directives still remain authoritative when they set the same fields during parse.

Troubleshooting

This chapter is organized by what the user sees when launch, licensing, simulation, or result generation fails. Start with GUI launch and licensing issues, then move to file, netlist, runtime-option, and CLI-specific problems.

20.1 The GUI launcher does not start

Typical symptoms

- The `JoSIM-Pro` Start Menu or Desktop shortcut is missing after Windows installation.
- Running `josim-pro-gui` on Linux prints `missing or non-executable GUI binary`.
- The Linux desktop entry appears, but the application does not open.
- A terminal cannot find `josim-pro-cli` or `optimum-cli` even though the GUI shortcut exists.

What this usually means

- The optional Windows shortcut or PATH feature was not selected, or the terminal was opened before installation finished.
- The Linux package is incomplete, was installed without dependency resolution, or the installed GUI executable is missing or not executable.
- The Linux desktop entry is present, but the wrapper cannot reach the installed GUI files.
- Command-line discovery and GUI launcher discovery are separate installation concerns.

Most likely fixes first

- On Windows, repair or rerun the MSI and select the Start Menu/Desktop shortcut options if you want launchers.
- If a Windows terminal cannot find CLI tools, open a new terminal after installation, enable command-line PATH integration, or run the tools from the installed `bin` directory.
- On Linux, install the package with `apt install`, `dnf install`, or `yum install`; avoid `dpkg -i` or `rpm -i` because they do not resolve dependencies automatically.
- On Linux, launch the GUI with `/usr/local/bin/josim-pro-gui`; use the wrapper rather than starting the GUI binary directly.
- If the wrapper still reports a missing executable, reinstall or repair the package. That error indicates a packaging or staging problem, not a netlist problem.

20.2 The GUI reports a licensing problem or opens in degraded mode

Typical symptoms

- The startup shell stays at `Checking license...` or reports `Licensing unavailable`.
- The startup shell offers **Retry**, **Continue degraded**, and **Quit**.
- The main workspace opens with a degraded-mode status message.
- `Help > About` reports `Status: Degraded mode / licensing unavailable`.
- **Generate System ID...** fails or no `SM_SysID.txt` file appears in the selected folder.

What this usually means

- The SUN Magnetics License Manager is not installed, is outdated, or its `smlicensed` daemon cannot be started.
- The configured license path is empty, incorrect, or points to a file the application cannot read.
- The licensing service cannot be contacted.
- The license is expired, inactive, tied to different hardware, or not valid for this installation.
- All available concurrent seats are already in use.

Most likely fixes first

- Install or update the SUN Magnetics License Manager first. System ID generation depends on `smlicensed`; JoSIM-Pro does not generate `SM_SysID.txt` by itself.
- Choose **Retry** after correcting the License Manager installation, license file, licensing service, network, firewall, or seat-availability issue.
- Choose **Continue degraded** when you need to inspect the GUI. Simulations still require a valid license, and System ID generation still requires `smlicensed`.
- Open `Help > About` and use **Generate System ID...** to ask `smlicensed` to save `SM_SysID.txt` when support requests a system identifier.
- On non-standard deployments, set `SMLICENSED_PATH` to the installed `smlicensed` executable and restart JoSIM-Pro.
- If the license file was issued but is not detected automatically, set the license path once with `josim-pro-cli -l /path/to/sm_license.txt` by itself, then restart the GUI. Do not add `-v`; version display exits before the license path is saved.
- Contact support@sun-magnetics.com for expired, deactivated, hardware-mismatched, or account-side license issues.

20.3 Command-line launch exits immediately

Typical symptoms

- `Unknown switch: ...`
- `Input file and output file names are the same.`
- The program prints usage/version information and exits.

What this usually means

- A command-line option was misspelled or is not supported.
- The same file was accidentally used for both input and output.
- The command was run without a valid input file or without the intended options.

Most likely fixes first

- Check the switch spelling against Chapter 15.
- Make sure the input and output paths are different.
- If you intended to pipe a netlist through standard input, provide input explicitly with `-i STDIN` instead of running the executable with no arguments.

Example

```
# Wrong: same path used for both input and output
josim-pro-cli circuit.cir -o circuit.cir

# Correct
josim-pro-cli -o results.csv circuit.cir
```

20.4 Input files, `.include` paths, and subcircuit boundaries

Typical symptoms

- `The file ... could not be found.`
- `Invalid .include statement: ...`
- `Missing end of subcircuit ...`
- `Mismatched .ends name: ...`
- `Unknown subcircuit specified in line: ...`
- `Incorrect number of I/O nodes for subckt ...`
- A cyclic include warning or cyclic subcircuit expansion error.
- The GUI opens a netlist, but simulation fails immediately because an included file is missing.

What this usually means

- A referenced file does not exist at the expected path.
- An included file is being resolved from the wrong working directory.
- A `.subckt` block was not closed properly, or the `.ends` name does not match.
- A subcircuit call uses the wrong subcircuit name or the wrong number of connection nodes.
- Two files include each other, or a subcircuit expands into itself recursively.

Most likely fixes first

- In the GUI, prefer `File > Open Folder...` for projects that depend on nearby include files.
- Check the path exactly as written in the netlist.
- Keep included files near the main netlist or use explicit relative paths that are stable.
- Verify every `.subckt` has a matching `.ends`.
- Check that each `X...` line provides exactly the number of I/O nodes expected by the target subcircuit.
- Remove cyclic include or cyclic subcircuit relationships.

Example

```
.include models/junctions.inc

.subckt JJ_STAGE in out bias
B1 in out JJMOD
R1 out bias 2
.ends JJ_STAGE

X1 n1 n2 nbias JJ_STAGE
```

If the included file is not found, or if `X1` does not provide the correct node count, JoSIM®-Pro will stop before simulation begins.

20.5 Netlist syntax, component definitions, and model lines

Typical symptoms

- `Unknown parameter ... found in component ...`
- `Duplicate component ... detected.`
- `Missing model for ...`
- `Invalid model line specified: Missing parameters.`
- `Unknown model type: ...`
- `Unknown model parameter in ...`
- `Invalid mutual inductance line specified: ...`

What this usually means

- A component instance uses an attribute that the component does not support.
- Two devices share the same identifier.
- A Josephson junction or other model-backed element references a model that does not exist.
- A `.MODEL` line is missing the type, parentheses, or one or more name=value pairs.
- A mutual inductance entry references bad syntax or invalid parameters.

Most likely fixes first

- Compare the line against the syntax reference rather than editing it by guesswork.
- Make all component names unique.
- Confirm that the model name on the component line exactly matches a defined `.MODEL`.
- For model definitions, check the type first: `JJ`, `RCSJ`, or `TJM` as appropriate.
- For mutual inductance, verify that both inductors exist and that the coupling value is parseable.

Example

```
.MODEL JJMOD JJ(RN=10 CAP=2.5p IC=1m)
B1 1 2 JJMOD area=1.2
```

A line such as `B1 1 2 MISSING_MODEL` will fail immediately because the model does not exist.

20.6 Parameters, expressions, and control commands fail to parse

Typical symptoms

- `String (...) cannot be converted to double.`
- `Mismatched parentheses in expression.`
- `Invalid parameter definition found: ...`
- `Unparseable parameters found.`
- `Invalid spread command defined.`
- `Invalid .IC syntax ...`
- `Invalid .option ... value: ...`

What this usually means

- A value that should evaluate to a number does not.
- A parameter expression contains missing parentheses or an unsupported form.
- A control statement such as `.SPREAD`, `.IC`, or `.OPTION` is malformed.

Most likely fixes first

- Reduce the failing expression to the smallest possible form and rebuild it.
- Check every `name=value` pair for missing delimiters.
- For `.OPTION analysis=` and `.OPTION integration=`, use only supported values.
- For `.OPTION outputprecision=`, keep the value in the documented range.
- For `.IC`, make sure the left-hand side targets a valid node or typed quantity and the right-hand side evaluates to a finite number.

Examples

```
.PARAM scale = 2
.PARAM Rbias = 50*scale
.SPREAD 0.02 R=0.05 B=0.10
.OPTION analysis=PHASE integration=BDF outputprecision=12
.IC V(n1)=0
```

If a parameter expression cannot be evaluated, fix that expression first before debugging downstream errors.

20.7 Source functions and noise expressions are rejected

Typical symptoms

- `Unsupported function type: ...`
- `Too few values to form ... function.`
- `Too many values to form ... function.`
- `The file ... could not be found.`
- `Invalid DC function value.`

What this usually means

- The waveform name is misspelled.
- The argument count does not match the selected function.
- A custom waveform file referenced by `CUS(...)` cannot be opened.
- A numeric argument in `NOISE(...)`, `PNOISE(...)`, `FNOISE(...)`, `ICNOISE(...)`, `PULSE(...)`, or another function is missing or malformed.

Most likely fixes first

- Start from the documented function signature and reenter the arguments in order.
- Check commas, spaces, and parentheses before changing the function itself.
- For `CUS(...)`, verify the file path and that the file contains numeric sample values.
- For noise functions, check every numeric argument, especially the sample/update interval and correlation-time style terms.

Examples

```
I1 1 0 NOISE(1u 0 1p)
I2 2 0 PNOISE(1u 1 0 1p 10n 64)
P1 3 0 FNOISE(1e-6 1 0 0 1p 10n 64)
B1 4 5 JJMOD ICNOISE=ICNOISE(0.01 1 0 1p 10n 64)
```

If the parser reports a function error, correct that line first. Later simulation errors often disappear once the source expression is valid.

20.8 Output requests are rejected or produce the wrong signals

Typical symptoms

- `Invalid .print directive: ...`
- `The component ... requested for output does not exist in the circuit.`
- Output appears empty or does not contain the signals you expected.
- The GUI waveform area opens with fewer traces than expected.

What this usually means

- The `.PRINT` or `.PLOT` form is invalid.
- A node or component name was mistyped.
- The request uses the wrong output form for the quantity being asked for.

Most likely fixes first

- Check the exact print form rather than editing the line incrementally.
- Make sure the requested node or device actually exists in the flattened circuit.
- Use device-oriented forms such as `W(R1)` or `E(R1)` for power and energy on supported two-terminal devices.
- In the GUI, check run diagnostics before assuming the plot view is at fault.

Examples

```
.PRINT V(n1)
.PRINT I(R1)
.PRINT P(B1)
.PRINT W(R1) E(R1)
.PRINT @R1[I]
```

If a request references a name that is not present in the circuit, fix the name first before investigating the output format.

20.9 Variable-step mode or runtime options are incompatible with the netlist

Typical symptoms

- `Variable-step mode currently supports PHASE + BDF only.`
- `Variable-step mode is not supported for LOSSLESS transmission lines.`
- `Blocking LOSSLESS lines: T1, T2.`
- `Invalid .option analysis value: ...`
- `Invalid .option integration value: ...`

What this usually means

- Variable-step mode was enabled for an unsupported analysis or integration method.
- The circuit contains one or more `LOSSLESS` transmission lines. The diagnostic lists the blocking line names.
- A runtime option was given an unsupported value.

Most likely fixes first

- If you want variable-step mode, use `PHASE` analysis with `BDF` integration.
- Use fixed-step mode for `LOSSLESS` transmission lines, including untyped `T` elements because `LOSSLESS` is their default model.
- Where the finite LC-ladder model is physically appropriate, select `TYPE=LCLADDER NSEG=...`; it supports variable-step `PHASE + BDF` through its generated inductors and capacitors.
- Re-check boolean and enumerated `.OPTION` values instead of trying nearby spellings.

Example

```
.OPTION analysis=PHASE integration=BDF variablestep=ON
```

If that run contains a `LOSSLESS` transmission line, disable variable-step mode or use a validated fixed-step setup. Do not change to `LCLADDER` only to avoid the restriction: the two models have different numerical and physical approximations.

20.10 Transmission-line initial conditions or delayed breakpoints are rejected

Typical symptoms

- `ICV and ICI must be specified together.`
- `Invalid initial-condition dimensions ...`
- `Explicit transmission line initial conditions require ANALYSIS=VOLTAGE.`
- `Explicit initial conditions are not supported for WIDEBAND ...`

Most likely fixes first

- Supply both `ICV` and `ICI`; do not specify only voltage or only current.
- Use one value for scalar lines, two differential/common values for `MODAL`, and exactly `NCOND` values for `@MTL`.
- Use `ANALYSIS=VOLTAGE`. Remove the explicit line state from `PHASE` runs.
- Remove `ICV`/`ICI` from `WIDEBAND` lines. Their recursive relaxation state is not defined by voltage and current alone.
- Delayed events and required interpolation history have no fixed application count cap. JoSIM-Pro retains the history needed by the line delay; sufficiently large runs still require adequate system memory.

Delayed-event breakpoints improve edge placement but do not prove timestep convergence. Repeat the run with a smaller nominal `.TRAN` step when validating edge shape or amplitude.

20.11 The simulation starts but fails, diverges, or produces no useful result

Typical symptoms

- `Simulation failed. Matrix has no solution.`
- `Simulation failed. Matrix has no solution during Newton iteration.`
- The run completes, but the output is clearly not what you expected.

What this usually means

- The circuit description is internally inconsistent.
- A model, bias condition, or initial condition is driving the solver into an impossible or unstable state.
- The requested analysis settings are not suitable for the circuit as written.

Most likely fixes first

- Check for missing grounds, missing models, or impossible parameter values.
- Reduce the problem to a smaller reproducible netlist.
- Remove optional spread/noise terms and verify the nominal circuit first.
- Try a simpler startup path before adding `UIC`, `STARTUP`, or advanced runtime tuning.
- For difficult circuits, confirm that the requested output and analysis modes are appropriate.

Practical debugging approach

1. Get the circuit running without spread or explicit noise.
2. Verify that all models and included files resolve correctly.
3. Add complexity back one feature at a time.

If a very small, valid-looking reproducer still fails in the same way after these checks, that is the point where it is reasonable to treat the behavior as a likely defect and report it.

20.12 Large runs, output files, compression, and streaming issues

Stored-result memory grows with the number of retained samples and traces. Advance reservation is bounded, but there is no fixed total trajectory memory budget. Use `-stream` with CSV or DAT to avoid retaining the complete trajectory; solver state, delay history and output buffers still require memory. If allocation fails, the CLI and C API report insufficient memory. Failed file streams discard temporary output and preserve an existing destination. Data already written to standard output cannot be retracted.

Typical symptoms

- `Compressed output is not supported for RAW output.`
- `Compressed output is not supported for BIN output.`
- A warning that streaming does not support the requested extension and will write CSV instead.
- Streaming output stops unexpectedly or the produced files are not in the format you expected.

What this usually means

- Compression was requested for a format that does not support it.
- Streaming mode was used with `.raw`, `.bin`, or another unsupported extension.
- The output target fell back to CSV because the requested format is not available in streaming mode.

Most likely fixes first

- Use `CSV` or `DAT` if you want compressed text output.
- Use buffered output instead of streaming if you need `RAW` or `BIN` output.
- Read warnings about fallback formats carefully; they are often telling you exactly what file was written instead.
- For very large runs, keep output requests focused so you are not writing unnecessary traces.

Examples

```
# Good streaming target
josim-pro-cli -p -o results.csv design.cir

# Better for buffered binary/raw workflows
josim-pro-cli -o results.bin design.cir
josim-pro-cli -o results.raw design.cir
```

20.13 Optimum exits before Margin, Yield, or Optimize runs

Typical symptoms

- `Configuration file missing mode value.`
- `Configuration file missing verify table.`
- `Verify table missing file value.`
- `Verify table missing file circuit.`
- `... is not a valid mode of operation.`
- Optimum prints `Mode: ...`, then stops before Margin, Yield, or Optimize output appears.
- An optimized netlist is not written even though `optimize.output` is set.

What this usually means

- The TOML file is missing the top-level `mode` value or the required `[verify]` table.
- `verify.file` or `verify.circuit` does not point to a readable file from the TOML file location.
- The nominal verification failed. Optimum intentionally stops before Margin, Yield, or Optimize when the baseline circuit does not verify.
- A selected parameter does not exist in the active circuit or subcircuit scope.
- `optimize.output` points to a directory that does not exist or cannot be written.

Most likely fixes first

- Start from the installed `DFF_Example_config.toml`, `DFF_Example.cir`, and `DFF_Example.sp` files before adapting a larger project.
- Keep the TOML, circuit, specification file, and include files in one project directory while validating relative paths.
- Confirm that `mode` is exactly one of `"verify"`, `"margin"`, `"yield"`, or `"optimize"`.
- Run `mode="verify"` first. Only move to Margin, Yield, or Optimize after nominal verification passes.
- Check `parameters.search_string`, explicit `[parameters]` entries, and `margin.subcircuit` against the actual parameter names in the netlist.
- Remember that Optimum runs through `optimum-cli` and a TOML configuration file. Use the GUI to save netlists or inspect generated files, then run Optimum from a terminal.

20.14 A practical order for debugging difficult runs

When several errors appear at once, fix them in this order:

1. GUI launcher and package-installation problems
2. licensing, degraded mode, and System ID generation problems
3. missing files and `.include` errors
4. syntax, model, and parameter errors
5. source-function and output-request errors
6. runtime-option compatibility issues
7. solver failures and result-quality problems
8. CLI-specific invocation, output, compression, or streaming problems
9. Optimum TOML, verification, parameter-selection, or output-path problems

That order matters. Many later-stage failures are simply downstream effects of an earlier launch, license, parse, or setup error.

Native Integration Appendix

This appendix keeps the native integration surfaces in view without turning them into a second automation manual. For most scripting and batch workflows, the Python chapter should remain the first choice. Use the native interfaces when you need to embed JoSIM®-Pro into an existing compiled application, when Python is not a suitable dependency, or when you are integrating from C, C++, Delphi, or Free Pascal.

A.1 What the native interfaces are for

The native interfaces are provided in two forms:

- a **C API** based on opaque integer handles for input, matrix, simulation, output, and runtime objects;
- a **Pascal import unit** that exposes the same exported functions for Delphi- and Free Pascal-style consumers.

These interfaces are useful when you want to call the simulator directly from another compiled program. They complement the Python chapter, which covers script-level automation.

A.2 When to use this instead of Python

Prefer the native interfaces when one or more of the following is true:

- your application is already written in C, C++, Delphi, or Free Pascal;
- you want to embed simulation into a GUI, service, or larger native tool;
- you need a C-callable boundary for another language or framework;
- Python is not available or not desirable in the deployment environment.

If your goal is mainly repeatable simulation runs, result inspection, and file export from scripts, the Python interface is simpler and should usually be preferred.

A.3 Minimal file-based workflow

The smallest practical native workflow is file-based and closely follows the normal simulator pipeline:

1. create an input handle from a circuit file with `JPRO_Input_CreateFromFileA(...)` or, on Windows, `JPRO_Input_CreateFromFileW(...)`;
2. optionally create a runtime handle when you need per-run overrides, then build the matrix with `JPRO_Matrix_CreateFromInputEx(...)`; otherwise use `JPRO_Matrix_CreateFromInput(...)`;
3. create a simulation handle with `JPRO_Simulation_CreateFromMatrix(...)`;
4. run the solve with `JPRO_Simulation_Run(...)` or `JPRO_Simulation_RunWithOutput(...)`;
5. either:
   - retrieve raw result arrays with `JPRO_Simulation_GetResults(...)` and the `JPRO_Results_*` accessors, or
   - create an output handle with `JPRO_Output_CreateFromInputMatrix(...)` and write CSV, binary, RAW, or console output with the `JPRO_Output_*` functions.
6. destroy all handles when complete.

All exported functions return a status code. When a call fails, retrieve the message with `JPRO_GetLastError()` before continuing.

A.4 Path and encoding notes

The file-based entry points are the recommended way to create inputs.

- Functions with an `A` suffix take UTF-8 paths.
- `JPRO_Input_CreateFromFileW(...)` is available on Windows for UTF-16 paths.
- The current runtime path setters exposed in the native API use UTF-8 `A`-style entry points.

In practice, this means UTF-8 paths are the portable default for native integrations, while Windows callers can use the wide-character input constructor when needed.

A.5 Authoritative reference

For exact signatures, types, calling conventions and exports, use the matching installed API files:

- `CAPI/jpro_capi.hpp` for the C API;
- `CAPI/JoSIM_CAPI.pas` for the Pascal import unit.

If this appendix and those files ever disagree, follow the header or Pascal unit for the package version you are using.