LAMMPS-GUI Documentation

Release v1.12.0

Axel Kohlmeyer

akohlmey@gmail.com

LAMMPS-GUI Documentation

---

1 About LAMMPS-GUI

LAMMPS-GUI is a graphical text editor with syntax highlighting, auto-completion, inline help, and indentation support for LAMMPS input files. It is programmed using the Qt Framework and customized for running, monitoring, and visualizing LAMMPS simulations. It calls LAMMPS directly using the LAMMPS library interface instead of launching an external LAMMPS executable. Therefore it can retrieve and display information from LAMMPS *while it is running* and *immediately* display visualizations created by a dump image command in the input.

The primary motivation for implementing LAMMPS-GUI is to facilitate teaching LAMMPS to beginners using only LAMMPS-GUI and to have a consistent behavior across major platforms like Linux, macOS, and Windows. This way one can focus on teaching LAMMPS and avoid having to spend time explaining different tools (for editing inputs, plotting graphs, visualizing systems) on the different platforms. Also, LAMMPS-GUI is fully integrated with a collection of LAMMPS tutorials.

Many of the features in LAMMPS-GUI are useful beyond working on tutorials. For instance, it can streamline the process of prototyping new simulation projects or debugging misbehaving simulations. It also has been found extremely useful in creating, debugging, and tweaking advanced visualizations with LAMMPS using the built-in visualization facility of the dump image command.

---

2 About this document

This document contains the documentation of LAMMPS-GUI and how to compile, install, use, configure, and modify it. Suggestions for new features and reports of bugs are always welcome. You can use the same channels as for LAMMPS itself for that purpose or submit bug reports or pull requests in the LAMMPS-GUI GitHub repository.

This document describes LAMMPS-GUI version 1.12.0.

---

3 Citing LAMMPS-GUI

There is currently no citation specifically describing LAMMPS-GUI, but an introduction to LAMMPS-GUI is included in the following publication in LiveCoMS for the LAMMPS tutorials that are linked from LAMMPS-GUI, so the suggestion is to cite that publication for now:

> Gravelle, S., Alvares, C. M. S., Gissinger, J. R., & Kohlmeyer, A. (2025). A Set of Tutorials for the LAMMPS Simulation Package [Article v1.0]. Living Journal of Computational Molecular Science, 6(1), 3027. https://doi.org/10.33011/livecoms.6.1.3037

or in BibTeX format:

```
@article{lammps_tutorials_2025,
  author={Gravelle, Simon and Alvares, Cecilia M. S. and Gissinger, Jacob R. and␣
↪Kohlmeyer, Axel},
  title={A Set of Tutorials for the {LAMMPS} Simulation Package [Article v1.0]},
  journal={Living Journal of Computational Molecular Science},
  pages={3027},
  volume={6},
  number={1},
  year={2025},
  month={Sep.},
  url={https://livecomsjournal.org/index.php/livecoms/article/view/v6i1e3037},
  DOI={10.33011/livecoms.6.1.3037}
}
```

---

4 User's Guide

4.1 Installation

LAMMPS-GUI is distributed as source code on GitHub and can be compiled as part of compiling LAMMPS, where it will be linked to the corresponding version of LAMMPS directly. Pre-compiled packages of LAMMPS with LAMMPS-GUI included are available for download (see below).

LAMMPS-GUI can also be compiled as a standalone package and load the LAMMPS library dynamically at runtime. This enables using LAMMPS-GUI with customized, patched, or extended LAMMPS versions containing features not available in the official LAMMPS distribution packages. It also allows to use LAMMPS-GUI with LAMMPS shared libraries compiled using the traditional makefile based build process (which does not support compiling LAMMPS-GUI directly). Pre-compiled packages of standalone LAMMPS-GUI versions with some basic LAMMPS shared library included are also available for download (see below).

Prerequisites and portability

LAMMPS-GUI is programmed in C++ based on the C++17 standard and using the Qt GUI framework. Currently, Qt version 5.15LTS or later is required; compiling with Qt version 6.x is preferred. When compiled with Qt version 6.x, LAMMPS-GUI can switch between a "light" and a "dark" theme according to the settings of the desktop environment. Otherwise, there are no changes in functionality between using either major version of Qt. Building LAMMPS-GUI requires CMake version 3.20 or later.

LAMMPS-GUI 1.12.0 has been successfully compiled and tested on

- Ubuntu Linux 22.04LTS x86_64 using GCC 11, Qt version 5.15
- Fedora Linux 43 x86_64 using GCC 14 and Clang 17, Qt version 5.15
- Fedora Linux 43 x86_64 using GCC 15, Qt version 6.10
- Apple macOS 12 (Monterey) with Xcode on arm64 and x86_64, Qt version 5.15
- Windows 11 x86_64 with Visual Studio 2026 and Visual C++ 14.50, Qt version 6.10
- Windows 11 x86_64 with MinGW / GCC 15.2 cross-compiler on Fedora 43, Qt version 6.10

Pre-compiled executables

Packages including a full LAMMPS version

For many users and especially for beginners learning to use LAMMPS, it is most convenient to install and use one of the pre-compiled packages that include both, LAMMPS-GUI and the command-line version of LAMMPS. In these packages LAMMPS-GUI is linked directly to the included LAMMPS library and thus it *cannot* be changed in the *LAMMPS-GUI preferences dialog*. Such pre-compiled LAMMPS executable packages are available for download for Linux x86_64 (Ubuntu 22.04LTS or later and compatible), macOS (version 11 aka Big Sur or later), and Windows (version 10 or later) from the LAMMPS releases page on GitHub. A backup download location is at https://download.lammps.org/static/ but may not always be up-to-date. Occasionally, also test version packages previewing recently added features are available at https://download.lammps.org/testing/ .

Standalone packages with a basic LAMMPS library

LAMMPS-GUI packages containing *only* LAMMPS-GUI compiled in plugin mode are available from the LAMMPS-GUI releases page on GitHub. These packages include a LAMMPS shared library with some subset of LAMMPS' features that do not depend on additional libraries.

If you want to override that choice of LAMMPS library, you can use the `-p` command line flag to tell LAMMPS-GUI

which other LAMMPS shared library file you want it to load. By using `-p ""` you can also reset any previous choice and thus trigger loading the default library again. Once LAMMPS-GUI is running, you can also change the path to the LAMMPS shared library from the *Preferences dialog*.

Since LAMMPS-GUI version 1.8.4, the minimum LAMMPS version required is 22 July 2025 update 2, but using a more recent LAMMPS release version is recommended to unlock all advanced features in LAMMPS-GUI.

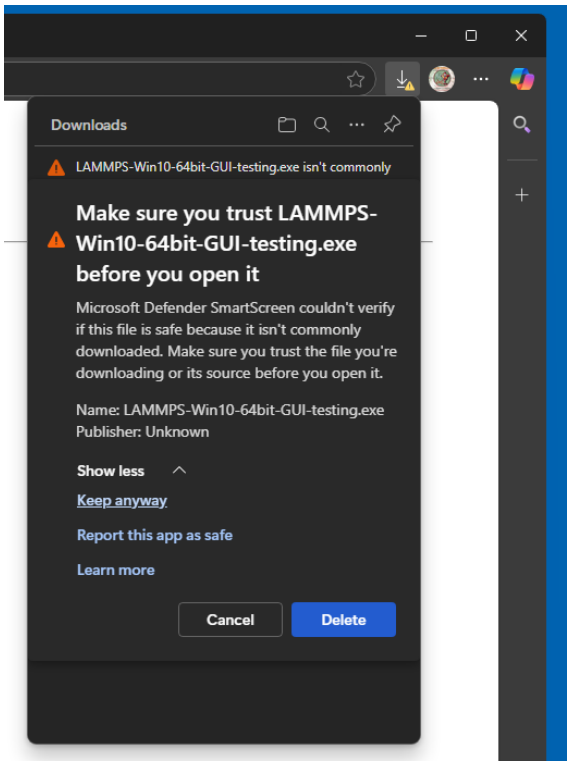
GPU support and MPI parallelization

The pre-compiled packages include support for GPUs through the GPU package with OpenCL (in mixed precision). However, this requires that you have a compatible driver and the OpenCL runtime installed. This is not always available and when using the flatpak bundle, the flatpak sandbox usually prevents accessing the GPU. GPU support through the KOKKOS package is currently not available for technical reasons, but serial and OpenMP multi-threading use of KOKKOS is available.

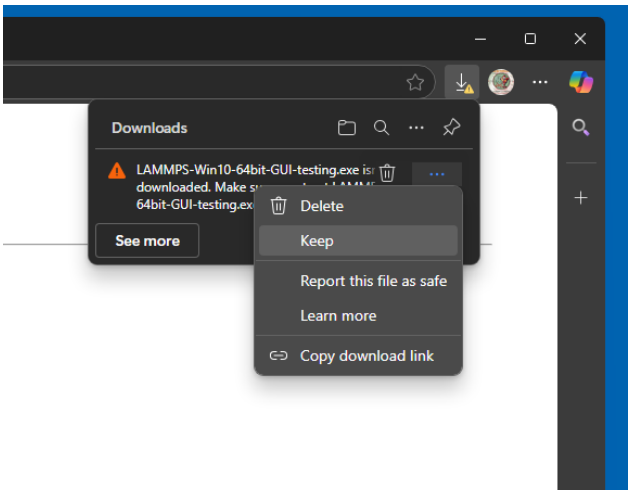
The design decisions for LAMMPS-GUI and how it launches LAMMPS conflict with parallel runs using MPI. You have to use a regular LAMMPS executable compiled with MPI support for that. For the use cases that LAMMPS-GUI has been conceived for (learning LAMMPS, testing or debugging LAMMPS inputs, prototyping new projects or complex workflows), this is not a significant limitation. Many supercomputing centers and high-performance computing clusters have parallel LAMMPS pre-installed.

Platform notes

Windows 10 and later

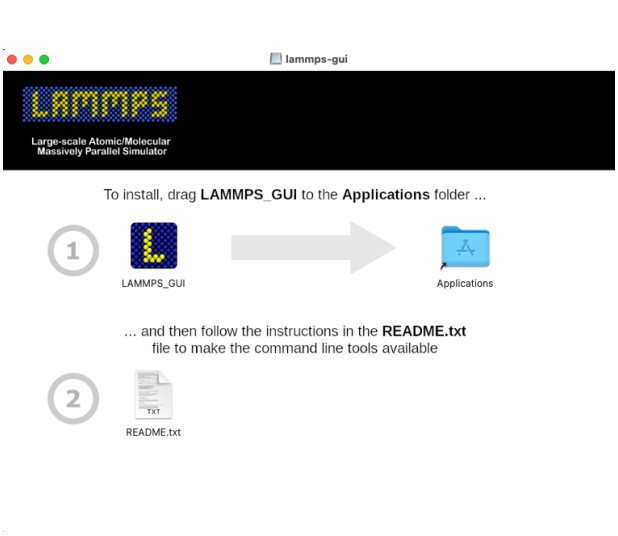


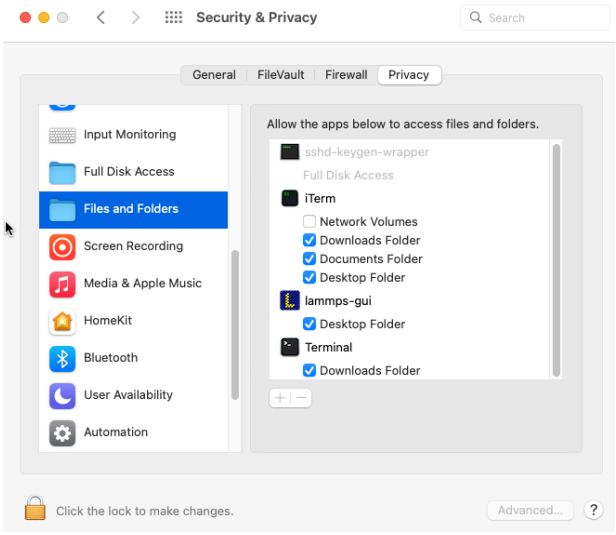
After downloading either the `LAMMPS-Win10-64bit-GUI-<LAMMPS version>.exe` or the `LAMMPS-GUI-Win10-x86_64-<LAMMPS-GUI version>.exe` installer package, you need to execute it, and start the installation process. Depending on your security settings of your web browser, you may have to explicitly tell it to download the file and then confirm **twice** to *keep the downloaded file* despite the claims that it may be dangerous and insecure. Since the installer packages are currently not cryptographically signed, you may also have to enable "Developer Mode" in the Windows System Settings to be able to run the installer.



MacOS 11 and later

After downloading the `LAMMPS-macOS-multiarch-GUI-<LAMMPS version>.dmg` or `LAMMPS-GUI-multiarch-<LAMMPS-GUI version>.dmg` application bundle disk image, you need to double-click it and then - in the window that opens - drag the app bundle as indicated into the "Applications" folder. Afterwards, the disk image can be unmounted or ejected. Then follow the instructions in the "README.txt" file to get access to the other included command-line executables, if desired.





Linux on x86_64

For Linux with x86_64 CPU there are currently two variants of pre-compiled LAMMPS-GUI: 1) a tar file with binaries and a wrapper script and 2) a flatpak bundle. The first is currently compiled on Ubuntu 22.04LTS (the oldest popular Linux distribution that provides the required C++17 compatibility out of the box) and depends on the backward compatibility of the core libraries between different releases on Linux distributions, and should be compatible with most recent Linux distributions. The second uses the flatpak sandbox environment to maintain binary compatibility across platforms.

Linux binary tarball

After downloading and unpacking the `LAMMPS-Linux-x86_64-GUI-<LAMMPS version>.tar.gz` or the `LAMMPS-GUI-Linux-x86_64-<LAMMPS-GUI version>.tar.gz` package, you can switch into the

"LAMMPS_GUI" folder and execute "./lammps-gui" directly:

```
$ cd ~/Downloads
$ tar -xzvvf LAMMPS-Linux-x86_64-GUI-22Jul2025.tar.gz
$ cd LAMMPS_GUI
$ ./lammps-gui &
```

The `LAMMPS_GUI` folder may also be moved around and added to the `PATH` environment variable so the executables will be found automatically.

Installing required compatibility packages

Since software is constantly evolving, it may be required to install additional software packages for your Linux distribution to achieve compatibility with binaries compiled on older distributions. For example the libraries `libxcb-xinput.so.0` and `libxcb-xinerama.so.0` may be missing and you thus get the error

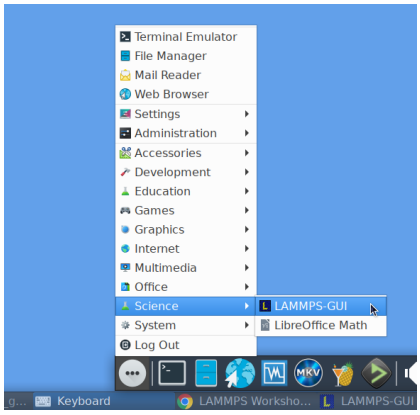
```
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was␣
↪found.
```

On Ubuntu 24.04 for example, those libraries are in the packages `libxcb-xinput0` and `libxcb-xinerama0` which are not installed by default. Using the flatpak bundle (see below) avoids these kind of issues by compiling and running the application in a standardized sandbox which is maintained by the flatpak software manager.

Linux flatpak bundle

The second Linux package variant uses flatpak and requires the flatpak management and runtime software to be installed. After downloading the `LAMMPS-GUI-Linux-x86_64-GUI-<version>.flatpak` flatpak bundle, you can install it with:

```
$ cd ~/Downloads
$ flatpak install --user LAMMPS-GUI-Linux-x86_64-GUI-<version>.flatpak
```



After installation, LAMMPS-GUI should be integrated into your desktop environment under "Applications > Science" but also can be launched from the console with `flatpak run org.lammps.lammps-gui`. The flatpak bundle also includes the console LAMMPS executable `lmp` which can be launched to run simulations with, for example with:

```
flatpak run --command=lmp org.lammps.lammps-gui -in in.melt
```

Other bundled command-line executables are run the same way and can be listed with:

```
ls $(flatpak info --show-location org.lammps.lammps-gui )/files/bin
```

Compilation from source

The source for LAMMPS-GUI was included with the LAMMPS source code distribution until LAMMPS version 22 July 2025 in the folder `tools/lammps-gui`. Starting with LAMMPS-GUI version 1.8.0 and LAMMPS version 10 September 2025 the LAMMPS-GUI sources are distributed separately through its own git repository at https://github.com/akohlmey/lammps-gui.

LAMMPS-GUI can still be built as part of a regular LAMMPS compilation. It will be automatically downloaded from its git repository and configured. This is usually the most convenient way. Since CMake is *required* to build LAMMPS-GUI, you need to build LAMMPS with CMake as well. To enable its compilation during compiling LAMMPS, the CMake variable `-D BUILD_LAMMPS_GUI=on` must be set when creating the CMake configuration. All other settings (compiler, flags, compile type) for LAMMPS-GUI are then inherited from the regular LAMMPS build. If the Qt library is installed as packaged for Linux distributions, then its location is typically auto-detected since the required CMake configuration files are stored in a location where CMake can find them without additional help. Otherwise, the location of the Qt library installation must be indicated by setting `-D Qt5_DIR=/path/to/qt5/lib/cmake/Qt5`, which is a path to a folder inside the Qt installation that contains the file `Qt5Config.cmake`. Similarly, for Qt6 the location of the Qt library installation can be indicated by setting `-D Qt6_DIR=/path/to/qt6/lib/cmake/Qt6`, if necessary. When both, Qt5 and Qt6 are available, Qt6 will be preferred unless `-D LAMMPS_GUI_USE_QT5=yes` is set.

Added in version 1.11.3.

Since the QtCharts module of the Qt library has been deprecated with Qt version 6.10, LAMMPS-GUI includes an alternate implementation of the charts display based on the QtGraphs module. This alternate version is enabled by default when compiling LAMMPS-GUI for Qt 6.10 and later. Setting `-D LAMMPS_GUI_US_QTCHARTS=yes` will enforce using the QtCharts based chart display.

LAMMPS-GUI plugin version

It is possible to compile a standalone LAMMPS-GUI executable (e.g. when LAMMPS has been compiled with traditional make). Rather than linking to the LAMMPS library during compilation, it includes a plugin loader that will load a LAMMPS shared library file dynamically at runtime during the start of the GUI; e.g. `liblammps.so.0` or `liblammps.0.dylib` or `liblammps.dll` (depending on the operating system). This has the advantage that the LAMMPS library can be built from updated or modified LAMMPS source without having to (re-)compile the GUI.

The ABI of the LAMMPS C-library interface is very stable and generally backward compatible. However, features used in LAMMPS-GUI may require a minimum LAMMPS version of the library. LAMMPS-GUI will print a suitable error message and exit if an incompatible LAMMPS library is loaded. You can override the path to the LAMMPS library with the `-p <path>` or `--pluginpath <path>` command-line flag. This is usually auto-detected on the first run and can be changed in the LAMMPS-GUI *Preferences* dialog. The command-line flag allows to reset this path to a valid value in case the original setting has become invalid. An empty path ("") as argument restores the default setting.

It is also possible to link the standalone compiled LAMMPS-GUI version to the LAMMPS library directly. This feature is enabled by setting `-D LAMMPS_GUI_USE_PLUGIN=off` (default setting is on). This is also the setting for compilation within LAMMPS. In this case, the CMake configuration needs to be told where to find the LAMMPS headers and the LAMMPS library, via `-D LAMMPS_SOURCE_DIR=/path/to/lammps/src`.

Platform notes

macOS

When building on macOS, the build procedure will try to manufacture a drag-n-drop installer, `LAMMPS-GUI-macOS-multiarch-<version>.dmg`, when using the 'dmg' target (i.e. `cmake --build <build dir> --target dmg` or `make dmg`.

To build multi-arch executables that will run on both, arm64 and x86_64 architectures natively, it is necessary to set the CMake variable `-D CMAKE_OSX_ARCHITECTURES=arm64;x86_64`. To achieve wide compatibility with different macOS versions, you can also set `-D CMAKE_OSX_DEPLOYMENT_TARGET=11.0` which will set compatibility to macOS 11 (Big Sur) and later, even if you are compiling on a more recent macOS version.

Windows

On Windows either native compilation from within Visual Studio 2022 with Visual C++ is supported and tested, or compilation with the MinGW / GCC cross-compiler environment on Fedora Linux.

Visual Studio

Using CMake and Ninja as build system are required. Qt needs to be installed, tested was a binary Qt package downloaded from https://www.qt.io, which installs into the `C:\\Qt` folder by default. There is a custom *x64-GUI-MSVC* build configuration provided in the `CMakeSettings.json` file that Visual Studio uses to store different compilation settings for project. Choosing this configuration will activate building the *lammps-gui.exe* executable in addition to LAMMPS through importing package selection from the `windows.cmake` preset file and enabling building LAMMPS-GUI and disabling building with MPI. When requesting an installation from the *Build* menu in Visual Studio, it will create a compressed `LAMMPS-GUI-Win10-amd64.zip` zip file with the executables and required dependent .dll files. This zip file can be uncompressed and `lammps-gui.exe` run directly from there. The uncompressed folder can be added to the `PATH` environment and LAMMPS and LAMMPS-GUI can be launched from anywhere from the command-line.

MinGW64 Cross-compiler

The standard CMake build procedure for cross-compilation can be applied. By using the `mingw64-cmake` wrapper the CMake configuration will automatically include a suitable CMake toolchain file (the regular cmake command can be used after that to modify the configuration settings, if needed). After building the libraries and executables, you can build the target 'nsis' (i.e. `cmake --build <build dir> --target nsis` or `make nsis` to build an Nullsoft installer package executable that can be executed on a Windows 10 or later machine with x86_64 CPU and will then install LAMMPS-GUI including a basic LAMMPS shared library file and all required dependencies.

Linux

Binary tarball package

Version 5.15LTS or later of the Qt library is required. Those are provided by, e.g., Ubuntu 22.04LTS or later. Thus older Linux distributions are not likely to be supported, while more recent ones will work, even for pre-compiled executables (see above). After compiling with `cmake --build <build folder>`, use `cmake --build <build folder> --target tgz` or `make tgz` to build a `LAMMPS-Linux-amd64.tar.gz` file with the executables and their support libraries.

Flatpak bundle

It is also possible to build a flatpak bundle which is a way to distribute applications in a way that is compatible with most Linux distributions (provided the flatpak system is installed). Use the "flatpak" target to trigger a compile (`cmake --build <build folder> --target flatpak` or `make flatpak`). Please note that this will not build from the local sources but from the repository and branch listed in the `org.lammps.lammps-gui.yml` LAMMPS-GUI source folder. Flatpak builds are currently only supported when building LAMMPS-GUI from within LAMMPS due to restrictions imposed by the flatpak sandbox.

4.2 Overview

LAMMPS-GUI is a graphical text editor customized for editing LAMMPS input files that are linked to the LAMMPS C-library and thus can run LAMMPS directly using the contents of the editor's text buffer as input. It can retrieve and display information from LAMMPS while it is running, display visualizations created with the dump image command, and is adapted specifically for editing LAMMPS input files through syntax highlighting, text completion, and reformatting, and linking to the online LAMMPS documentation for known LAMMPS commands and styles.

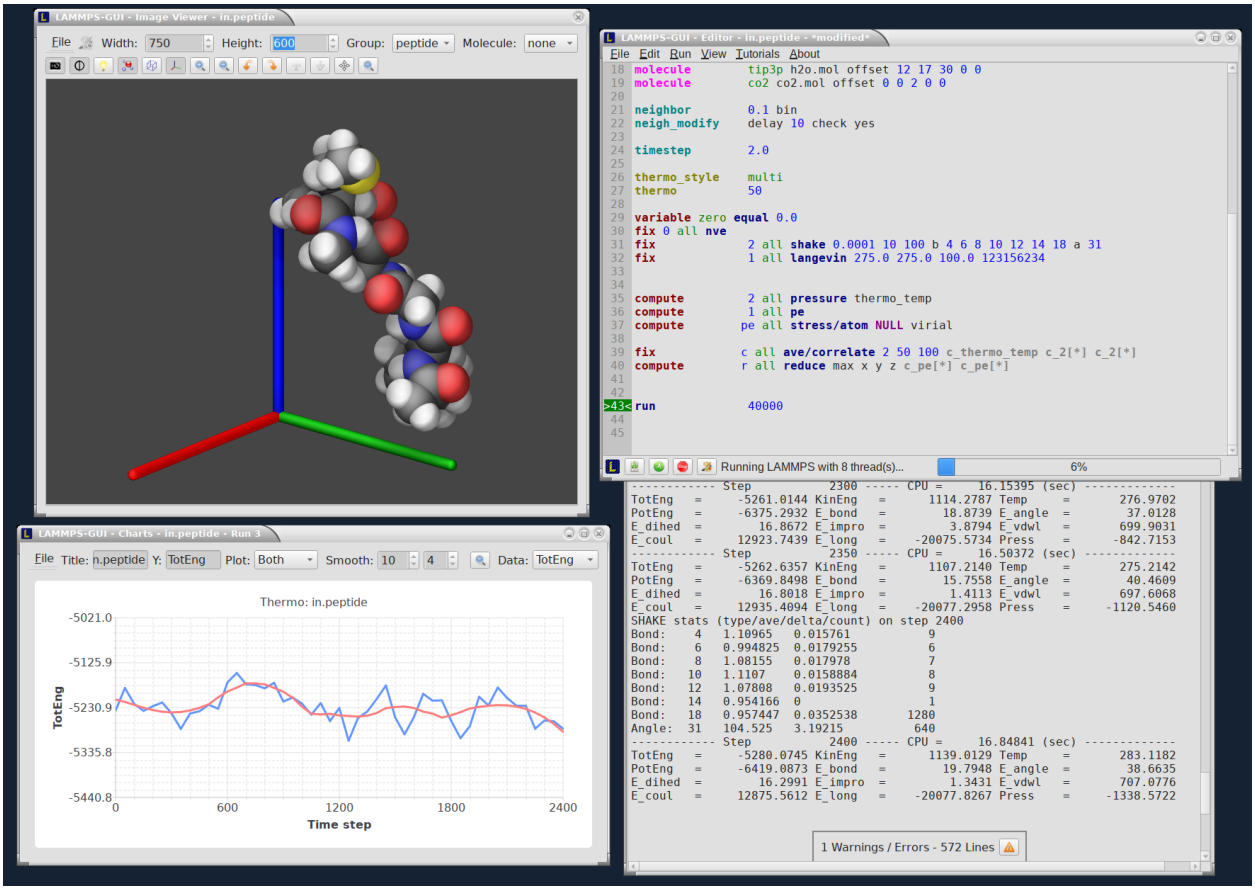
LAMMPS-GUI aims to support a workflow similar to the traditional experience of running LAMMPS using a text editor, a command-line window, launching the LAMMPS text-mode executable printing output to the screen, and post-processing and visualizing LAMMPS' output but just integrated into a single application.

LAMMPS-GUI integrates well with graphical desktop environments where the *.lmp* filename extension can be registered with LAMMPS-GUI as the executable to launch when double clicking on such files using a file manager. LAMMPS-GUI will launch and read the file into its buffer. Input files can also be dropped into the editor window of the running LAMMPS-GUI application, which will close the current file and open the new file.

LAMMPS-GUI makes it easier for beginners to get started running LAMMPS and is well-suited for LAMMPS tutorials, since you only need to work with a single, ready-to-use program for most of the tasks. Plus it is available for download as pre-compiled package for popular operating systems (Linux, macOS, Windows). This saves time and allows users to focus on learning LAMMPS itself, without the need to learn how to compile LAMMPS, learn how to use the command line, or learn how to use a separate text editor, plotting or visualization program.

The tutorials at https://lammpstutorials.github.io/ are specifically updated for use with LAMMPS-GUI and their tutorial materials can be downloaded and edited directly from within the GUI while automatically loading the matching tutorial instructions into a web browser.

While making it easy for beginners to start with LAMMPS, it is also the expectation that LAMMPS-GUI users will eventually transition to workflows that most experienced LAMMPS users employ. That traditional procedure is effective for people proficient in using the command-line, as it allows them to use the tools for the individual steps that they are most comfortable with. In fact, it is often *required* to adopt this workflow when running LAMMPS simulations on high-performance computing facilities.



Most features in LAMMPS-GUI have been exposed to keyboard shortcuts, so that there is also appeal for experienced LAMMPS users for prototyping and testing simulation setups.

Features

A detailed discussion and explanation of all features and functionality are in the following pages. Here are a few highlights of LAMMPS-GUI:

- Text editor with line numbers and syntax highlighting customized for LAMMPS
- Text editor features command completion and indentation for known commands and styles
- Text editor will switch its working directory to folder of file in buffer
- Indicator for currently executed command
- Indicator for line that caused an error
- Progress bar indicates how far a run command is completed and how CPUs are utilized
- Context-sensitive help for LAMMPS commands via the online documentation
- Auto-adapting to features and packages available in the LAMMPS library in use
- LAMMPS is running in a concurrent thread, so the GUI remains responsive
- LAMMPS can be started and stopped with a mouse click or a hotkey
- Screen output is captured in an *Output* Window
- Many adjustable settings and preferences that are persistent including the 5 most recent files
- Thermodynamic output is captured and displayed as line graph in a *Chart* Window
- Interactive visualization of current state via calling write_dump image
- Capture of images created by dump image in Slide show window
- Dialog to set variables, similar to the LAMMPS command-line flag '-v' / '-var'
- Support for GPU, INTEL, KOKKOS/OpenMP, OPENMP, and OPT accelerator packages
- Inspection of binary restart files created by LAMMPS
- Integration with LAMMPS tutorials

4.3 Basic usage of LAMMPS-GUI

Command-line options

LAMMPS-GUI supports the following command-line options:

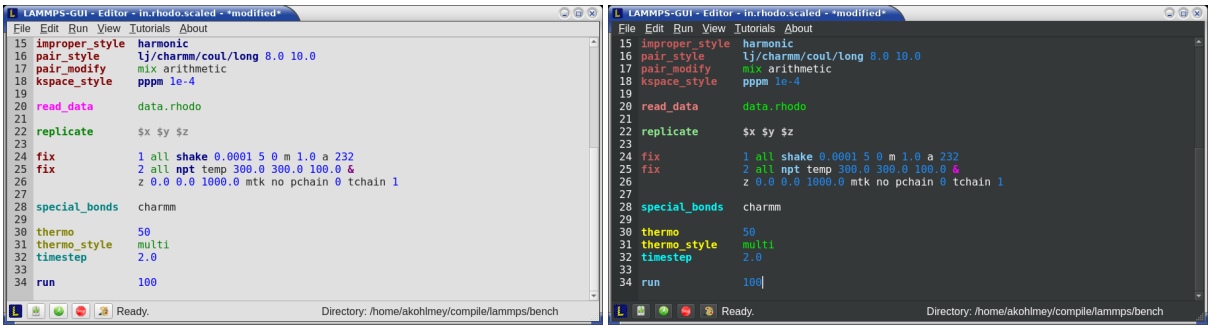
```
lammps-gui [options] [file]
```

Option	Description
`-x <width>`, `--width <width>`	Override the editor window width in pixels
`-y <height>`, `--height <height>`	Override the editor window height in pixels
`-s <style>`, `--style <style>`	Set the visual style of the application (default: `Fusion`)
`-p <path>`, `--pluginpath <path>`	Set the path to the LAMMPS shared library (plugin mode only)
`-v`, `--version`	Print version information and exit
`-h`, `--help`	Print usage information and exit

The optional `file` argument specifies a LAMMPS input file to open on startup. If no file is provided, LAMMPS-GUI starts with an empty editor buffer. Available choices for the visual style depend on the platform and Qt configuration. On most platforms, there is also the option `Windows` which is a visual style somewhat resembling Windows 95.

Launching LAMMPS-GUI

When LAMMPS-GUI starts, it shows the main window, labeled *Editor*, with either an empty buffer or the contents of the file used as argument. In the latter case it may look like the following:



There is the typical menu bar at the top, then the main editor buffer, and a status bar at the bottom. The input file contents are shown with line numbers on the left and the input is colored according to the LAMMPS input file syntax. The status bar shows the status of LAMMPS execution on the left (e.g. "Ready." when idle) and the current working directory on the right. The name of the current file in the buffer is shown in the window title; the word **modified** is added if the buffer edits have not yet saved to a file. The geometry of the main window is stored when exiting and restored when starting again.

Opening and saving files

The LAMMPS-GUI application can be launched without command-line arguments and then starts with an empty buffer in the *Editor* window.

If a file argument is given, LAMMPS-GUI will use it as the file name for the *Editor* buffer and read its contents into the buffer, provided a file of that name exists; otherwise the buffer will be empty, but set up to save any added content to that file. Files can also be opened via the *File* menu, the *Ctrl-O* (*Command-O* on macOS) keyboard shortcut or by drag-and-drop of a file from a graphical file manager into the editor window. If a file extension (e.g. `.lmp`) has been registered with the graphical environment to launch LAMMPS-GUI, an existing input file can be launched with LAMMPS-GUI through double clicking.

Only one file can be edited at a time, so opening a new file with a file already loaded into the buffer closes that buffer. If the buffer has unsaved modifications, you are asked to either cancel the operation, discard the changes, or save them. A buffer with modifications can be saved any time from the "File" menu, by the keyboard shortcut *Ctrl-S* (*Command-S* on macOS), or by clicking on the "Save" button at the very left in the status bar.

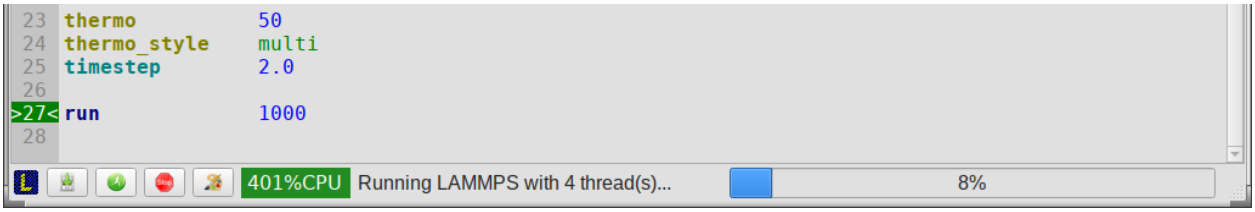
Running LAMMPS

From within the LAMMPS-GUI main window LAMMPS can be started either from the *Run* menu using the *Run LAMMPS from Editor Buffer* entry, by the keyboard shortcut *Ctrl-Enter* (*Command-Enter* on macOS), or by clicking on the green "Run" button in the status bar. All of these operations cause LAMMPS to process the entire input script in the editor buffer, which may contain multiple run or minimize commands.

LAMMPS runs in a separate thread, so the GUI stays responsive and is able to interact with the running calculation and access data it produces. It is important to note that running LAMMPS this way is using the contents of the input buffer for the run (via the lammps_commands_string function of the LAMMPS C-library interface), and **not** the original file it was read from. Thus, if there are unsaved changes in the buffer, they *will* be used. As an alternative, it is also possible to run LAMMPS by reading the contents of a file from the *Run LAMMPS from File* menu entry or with *Ctrl-Shift-Enter*. This option may be required in some rare cases where the input uses some functionality that is not compatible with

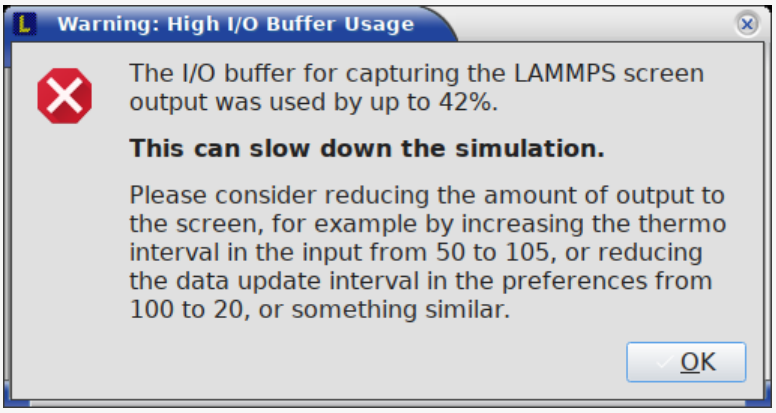
running LAMMPS from a string buffer. For consistency, any unsaved changes in the buffer must be either saved to the file or undone before LAMMPS can be run from a file.

The line number of the currently executed command is highlighted in green in the line number display for the *Editor* Window.



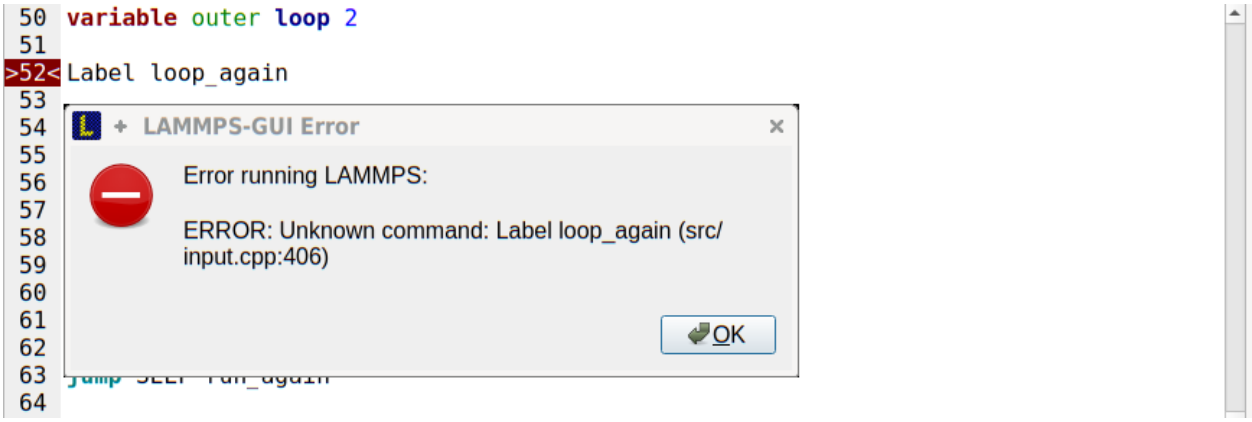
While LAMMPS is running, the contents of the status bar change. The text fields that normally show "Ready." and the current working directory, change into an area showing the CPU utilization in percent. Next to it is a text indicating that LAMMPS is running, which also indicates the number of active threads (in case thread-parallel acceleration was selected in the *Preferences* dialog). On the right side, a progress bar is shown that displays the estimated progress for the current run or minimize command.

CPU Utilization



The CPU Utilization should ideally be close to 100% times the number of threads like in the screenshot image above. Since the GUI is running as a separate thread, the CPU utilization *may* be higher, for example when the GUI needs to work hard to keep up with the output produced by the simulation; for example when there is frequent thermo output or the simulation runs very fast. In the *Preferences* dialog, the polling interval for updating the *Output* and *Charts* windows can be adjusted. The intervals may need to be lowered to not miss data between *Charts* data updates or to avoid stalling when the thermo output is not transferred to the *Output* window fast enough. You could also make LAMMPS run slower by reducing or turning off thread parallelization. It is also possible to reduce the amount of data by increasing the thermo interval. LAMMPS-GUI detects if the associated I/O buffer is significantly full, and will print a warning *after* the run with suggested adjustments. The CPU utilization can also be lower than expected, when some significant parts of the code paths in use are not multi-threaded, when the simulation is slowed down by the GUI or other processes also running on the host computer and competing with LAMMPS-GUI for resources.

If an error occurs (in the example below the command label was incorrectly capitalized as "Label"), an error message dialog is shown and the line of the input which triggered the error is highlighted in red. The state of LAMMPS in the status bar is set to "Failed." instead of "Ready."



Up to three additional windows may open during a run

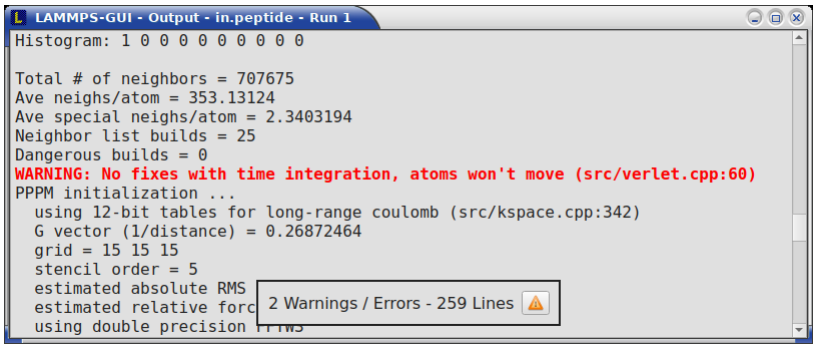
- An *Output window* with the captured screen output from LAMMPS
- A *Charts window* with a line graph created from thermodynamic output of the run
- A *Slide Show window* with images created by a dump image command in the input

More information on those windows and how to adjust their behavior and contents is given in *the next pages*.

An active LAMMPS run can be stopped cleanly by using either the *Stop LAMMPS* entry in the *Run* menu, the keyboard shortcut *Ctrl-/* (*Command-/* on macOS), or by clicking on the red button in the status bar. This will cause the running LAMMPS process to complete the current timestep (or iteration for energy minimization) and then complete the processing of the buffer while skipping all run or minimize commands. This is equivalent to the input script command timer timeout 0 and is implemented by calling the lammps_force_timeout function of the LAMMPS C-library interface. Please see the corresponding documentation pages to understand the implications of this operation.

4.4 Monitoring LAMMPS output

Output Window



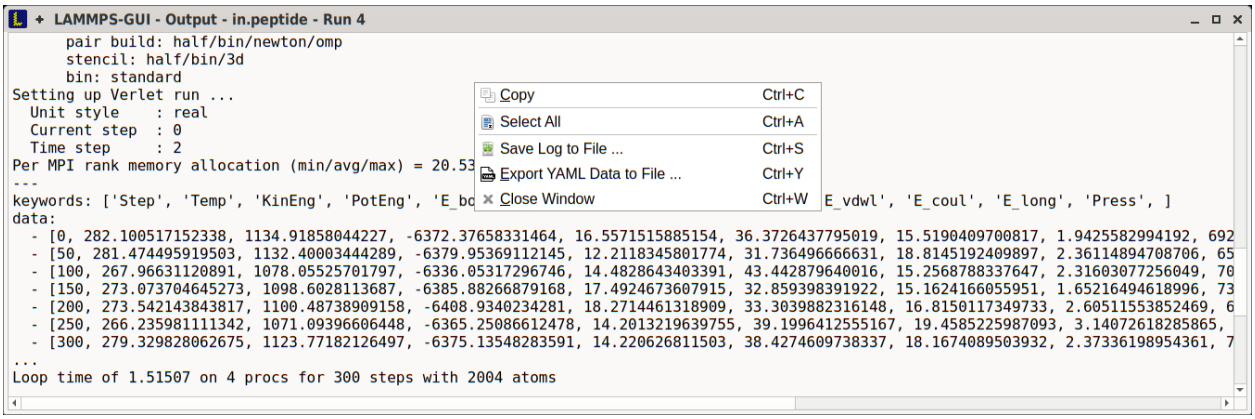
By default, when starting a run, an *Output* window opens that displays the screen output of the running LAMMPS calculation, as shown below. This text would normally be seen in the command-line window.

LAMMPS-GUI captures the screen output from LAMMPS as it is generated and updates the *Output* window regularly during a run. If there are any warnings or errors in the LAMMPS output, they are highlighted by using bold text colored in red. There is a small panel at the bottom center of the *Output* window showing how many warnings and errors were detected and how many lines the entire output has. By clicking on the button on the right with the warning symbol or by using the keyboard shortcut *Ctrl-N* (*Command-N* on macOS), you can jump to the next line with a warning or error. If there is a URL pointing to additional explanations in the online manual, that URL will be highlighted and

double-clicking on it shall open the corresponding manual page in the web browser. The option is also available from the context menu.

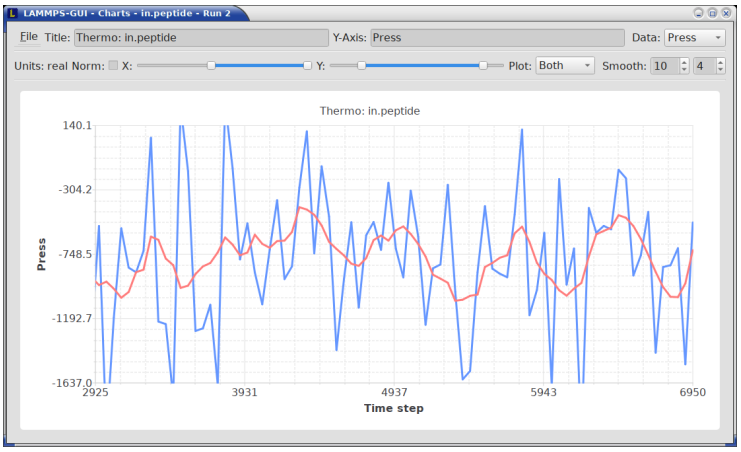
By default, the *Output* window is replaced each time a run is started. The runs are counted and the run number for the current run is displayed in the window title. It is possible to change the behavior of LAMMPS-GUI in the preferences dialog to create a *new Output* window for every run or to not show the current *Output* window. It is also possible to show or hide the *current Output* window from the *View* menu.

The text in the *Output* window is read-only and cannot be modified, but keyboard shortcuts to select and copy all or parts of the text can be used to transfer text to another program. Also, the keyboard shortcut *Ctrl-S* (*Command-S* on macOS) is available to save the *Output* buffer to a file. The "Select All" and "Copy" functions, as well as a "Save Log to File" option are also available from a context menu by clicking with the right mouse button into the *Output* window text area.



Should the *Output* window contain embedded YAML format text (see above for a demonstration), for example from using thermo_style yaml or thermo_modify line yaml, the keyboard shortcut *Ctrl-Y* (*Command-Y* on macOS) is available to save only the YAML parts to a file. This option is also available from a context menu by clicking with the right mouse button into the *Output* window text area.

Charts Window



By default, when starting a run, a *Charts* window opens that displays a plot of thermodynamic output of the LAMMPS calculation as shown below.

The "Data:" drop down menu on the top right allows selection of different properties that are computed and written as thermodynamic output to the output window. Only one property can be shown at a time. The plots are updated regularly with new data as the run progresses, so they can be used to visually monitor the evolution of available properties. The update interval can be set in the *Preferences* dialog. By default, the raw data for the selected property is plotted as a

blue graph. From the "Plot:" drop menu on the second row and on the left, you can select whether to plot only raw data graph, only a smoothed data graph, or both graphs on top of each other. The smoothing process uses a Savitzky-Golay convolution filter. The convolution window width (left) and order (right) parameters can be set in the boxes next to the drop down menu. Default settings are 10 and 4 which means that the smoothing window includes 10 points each to the left and the right of the current data point for a total of 21 points and a fourth order polynomial is fitted to the data in the window.

The "Title:" and "Y:" input boxes allow to edit the text shown as the plot title and the y-axis label, respectively. The text entered in the "Title:" box is applied to *all* charts, while the "Y:" text changes only the y-axis label of the currently *selected* plot.

The window title shows the current run number that this chart window corresponds to. Same as for the *Output* window, the chart window is replaced on each new run, but the behavior can be changed in the *Preferences* dialog.

From the *File* menu on the top left, it is possible to save an image of the currently displayed plot or export the data in either plain text columns (for use by plotting tools like gnuplot or grace), as CSV data which can be imported for further processing with Microsoft Excel, LibreOffice Calc, or with Python via pandas, or as YAML which can be imported into Python with PyYAML or pandas.

Thermo output data from successive run commands in the input script is combined into a single data set unless the format, number, or names of output columns are changed with a thermo_style or a thermo_modify command, or the current time step is reset with reset_timestep, or if a clear command is issued. This is where the YAML export from the *Charts* window differs from that of the *Output* window: here you get the compounded data set starting with the last change of output fields or timestep setting, while the export from the log will contain *all* YAML output but *segmented* into individual runs.

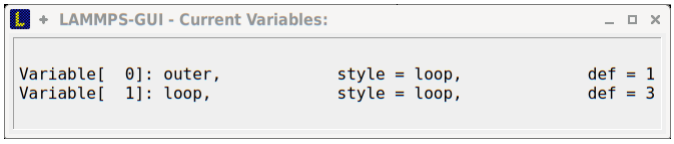
The *Preferences* dialog has a *Charts* tab, where you can configure multiple chart-related settings, like the default title, colors for the graphs, default choice of the raw / smooth graph selection, and the default chart graph size.

Slowdown of Simulations from Charts Data Processing

Using frequent thermo output during long simulations can result in a significant slowdown of that simulation since it is accumulating many data points for each of the thermo properties in the chart window to be redrawn with every update. The updates are consuming additional CPU time when smoothing enabled. This slowdown can be confirmed when an increasing percentage of the total run time is spent in the "Output" or "Other" sections of the MPI task timing breakdown. It is thus recommended to use a large enough value as argument *N* for the thermo command and to select plotting only the "Raw" data in the *Charts Window* during such simulations. It is always possible to switch between the different display styles for charts during the simulation and after it has finished.

Changed in version 1.7: As of LAMMPS-GUI version 1.7 the chart data processing is significantly optimized compared to older versions of LAMMPS-GUI. The general problem of accumulating excessive amounts of data and the overhead of too frequently polling LAMMPS for new data cannot be optimized away, though. If necessary, the command line LAMMPS executable needs to be used and the output accumulated on a very fast disk (e.g. a high-performance SSD).

Variable Info



During a run, it may be of interest to monitor the value of input script variables, for example to monitor the progress of loops. This can be done by enabling the "Variables Window" in the *View* menu or by using the *Ctrl-Shift-W* keyboard shortcut. This shows info similar to the info variables command in a separate window as shown below.

Like for the *Output* and *Charts* windows, its content is continuously updated during a run. It will show "(none)" if there are no variables defined. Note that it is also possible to *set* index style variables, that would normally be set via command-line flags, via the "Set Variables…" dialog from the *Run* menu. LAMMPS-GUI automatically defines the variable "gui_run" to the current value of the run counter. That way it is possible to automatically record a separate log for each run attempt by using the command

```
log logfile-${gui_run}.txt
```

at the beginning of an input file. That would record logs to files `logfile-1.txt`, `logfile-2.txt`, and so on for successive runs.

4.5 Visualization

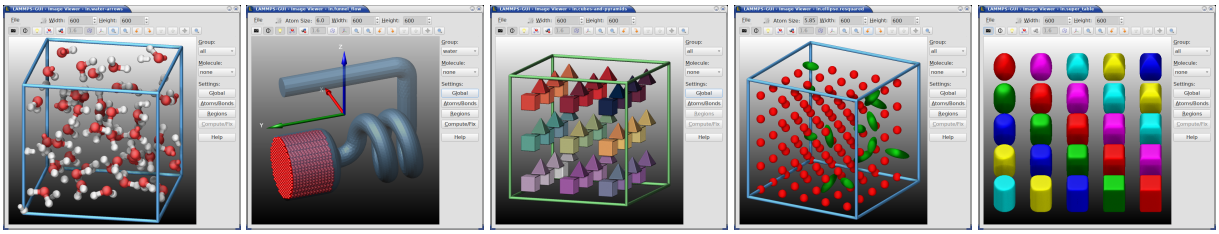
Snapshot Image Viewer

By selecting the *Create Image* entry in the *Run* menu, or by hitting the *Ctrl-I* (*Command-I* on macOS) keyboard shortcut, or by clicking on the "palette" button in the status bar of the *Editor window*, LAMMPS-GUI sends a custom write_dump image command to LAMMPS and reads back the resulting snapshot image with the current state of the system into an image viewer. This functionality is *not* available *during* an ongoing run. In case LAMMPS is not yet initialized, LAMMPS-GUI tries to identify the line with the first run or minimize command and execute all commands in the editor up to that line, and then executes a "run 0" command. This initializes the system so an image of the initial state of the system that can be rendered. If there was an error in that process, a dialog with the error message will appear.

Automatic settings

When possible, LAMMPS-GUI tries to detect which elements the atoms correspond to (via their mass) and then colorizes them in the image and sets their atom diameters accordingly. If this is not possible - for instance when using reduced (= 'lj') units - then LAMMPS-GUI will check the current pair style and if it is a Lennard-Jones type potential, it will extract the *sigma* parameter for each atom type and assign atom diameters from those numbers. When using an atom style where the atom diameters are set directly on a per-atom basis, LAMMPS will use that value. For cases where atom diameters are not auto-detected or you want to override the choice, you can configure it in the *Atom/Bond* settings dialog (see below). The default value is inferred from the x-direction lattice spacing.

For particles that use atom styles "body", "ellipsoid", "line", or "tri" LAMMPS will visualize the particles according to their atom style information by default. Other particles types will be visualized as sphere. In the *Atom/Bond* settings dialog, this can be further customized (or disabled).



It is also possible to visualize regions, graphics from computes and fixes, and have bonds computed dynamically for potentials, where the bonds are determined implicitly (like AIREBO. Please see the documentation of the dump image command for more details on these and other features and the LAMMPS Visualization Howto for more general discussions on how to generate advanced visualizations with LAMMPS directly.

If elements cannot be detected the default sequence of colors of the dump image command is assigned to the different atom types.

Customizations

The Image Viewer controls described below support significant customization of the default visualization through the toolbar buttons, editable text fields, and additional dialogs. This covers a wide variety of possible customizations. After each change is applied, LAMMPS-GUI will have LAMMPS re-create the displayed image with the updated settings. The resulting image can be saved or copied to the clipboard and pasted into a compatible application.

Delays in updating the image

Some visualization settings, especially enabling SSAO and FSAA (see below) or massively enlarging the image size, can significantly increase the time required for LAMMPS to render the image. While in most cases, the image will be updated in a fraction of a second, complex visualizations may take multiple seconds. While LAMMPS is active to render an updated image, a small color palette icon in the menu bar is colored and will be grayed out when rendering is complete.

For further customization or making the visualization available when running the simulation with LAMMPS directly (e.g. when running on a cluster after enlarging the system), you can copy the current dump image and dump_modify commands to the clipboard so they can be pasted into a LAMMPS input file in either the included *text editor window* or some other text editor and adjusted according to the documentation.

The resulting images will be shown automatically in the *slide show viewer* when running the simulation with the thus modified input from LAMMPS-GUI. This strategy has been used to great effect to create many of the simulation snapshot images shown in this documentation and the LAMMPS manual.

Available controls

The Image Viewer window consists of three main areas: a menu/toolbar strip at the top, the rendered image in the center, and a settings panel on the right side. Following the general theme of LAMMPS-GUI of extensive keyboard shortcut support, you can select most text fields by using the *Alt* key and the underlined letter. For example the *File* menu is opened with *Alt-F* and its entries can be also selected the same way or using the cursor keys and *Enter*. Keyboard shortcuts starting with *Ctrl* usually work globally inside the image window, that is even when the corresponding menu item is not visible.

Note

Some options (for example, axes location, transparency, top background color, enabling visualizations of regions or computes and fixes) require that LAMMPS-GUI is interfaced with LAMMPS version 10 December 2025 or 11 February 2026 or later. These fields are grayed out and disabled when an older version of LAMMPS is used.

The **menu/toolbar strip** consists of two rows: the first row with the *File* menu, atom and bond size controls, image dimension controls, and a second row of toggle and action buttons.

The **menu bar row** has:

- **The File menu with the following entries:**
  - **Save As...** (*Ctrl-S*): Save the rendered image to a file. The file format is inferred from the file name extension. When the ImageMagick software is installed, additional file formats beyond those natively supported by the Qt library become available.
  - **Copy Image** (*Ctrl-C*): Copy the rendered image to the clipboard for pasting it into another application. This requires support from the receiving applications, but many applications like document editors or web browsers are.

  - **Copy dump image command** (*Ctrl-D*): Copy the current dump image and dump_modify commands to the clipboard so they can be pasted into a LAMMPS input file in either the included *text editor window* or some other text editor. This allows the current visualization settings to be reproduced during a simulation run, including in the *slide show viewer*.
  - **Close** (*Ctrl-W*): Close the Image Viewer window.
  - **Quit** (*Ctrl-Q*): Quit the entire application.

- The **busy indicator**, a small palette icon that is colored  while LAMMPS is rendering a new image and grayed out  when rendering is complete.
- The **Atom size** text field, where the atom diameter can be adjusted. This field is only visible when the atom diameter is not automatically set.
- The **Bond size** text field, where the bond diameter can be adjusted. This field is only visible when the bond diameter is not automatically set and display of explicit or implicit bonds is enabled
- The **Width** spin box where the image width can be set. It can be accessed using the *Alt-W* keyboard shortcut.
- The **Height** spin box, where the image height can be set. It can be accessed using the *Alt-H* keyboard shortcut.

The **toolbar buttons** row below the menu bar provide quick access to several rendering options and view manipulations. From left to right there are:

- **SSAO** (toggle): Enable or disable Screen Space Ambient Occlusion rendering for a more spatial, depth-shaded appearance images at the expense of more CPU time.
- **Anti-aliasing** (toggle): Render the image at double resolution and scale down for smoother edges. Full Scene Anti-Aliasing (FSAA) produces higher quality images at the expense of more CPU time. It is particularly recommended in combination with any transparent objects.
- **Shininess** (toggle): Switch between shiny and matte surface rendering of graphics objects like atoms and bonds.
- **VDW style** (toggle): Switch between space-filling (Van der Waals) sphere representation of atoms and the smaller ball-and-stick style of atoms and bonds.
- **Dynamic bonds** (toggle): Automatically compute bonds from atom distances. This is useful for force fields with implicit bonds. When enabled, the adjacent text field allows setting the bond cutoff distance. This feature depends on existing neighbor list data and thus may not always work as expected when the system has explicit bonds and thus neighbors may be automatically excluded from neighbor lists due to the special_bonds settings
- **Box** (toggle): Show or hide the simulation box drawn as colored cylinders.
- **Axes** (toggle): Show or hide the labeled coordinate axes arrows.
- **Zoom in / Zoom out**: Adjust the zoom level between in 10 percent increments between 0.1x and 10.0x.
- **Rotate left / Rotate right**: Rotate the view horizontally by 10 degrees per click.
- **Rotate up / Rotate down**: Rotate the view vertically by 10 degrees per click.
- **Recenter**: Recenter the view on the center of mass of the currently selected group.
- **Reset**: Reset the view to the default orientation and zoom level.

The default image size, some default image quality settings, the view style and some colors can be changed in the *Preferences* dialog window. From the image viewer window further adjustments can be made: actual image size, high-quality (SSAO) rendering, anti-aliasing, view style, display of box or axes, zoom factor. The view of the system can be rotated horizontally and vertically.

The **settings panel** on the right side of the window provides additional controls (most are explained in detail below):

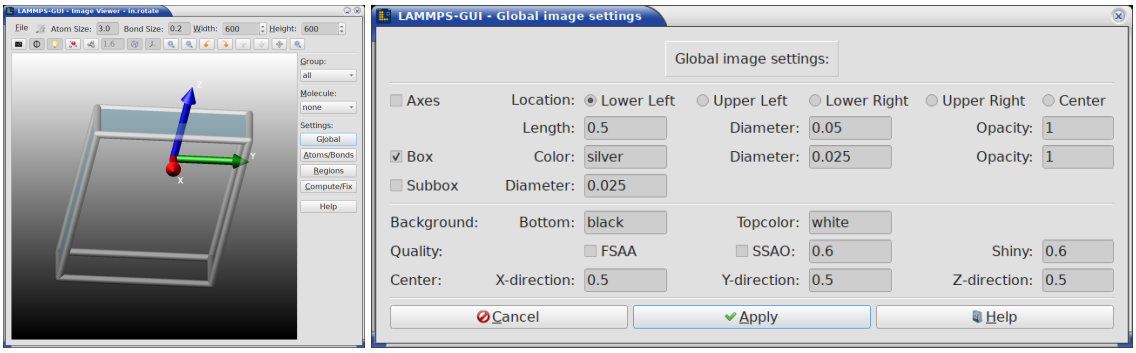
- **Group**: A drop-down list to select which group of atoms to display (default is "all"). Only atoms belonging to the selected group are rendered.

- **Molecule**: A drop-down list to select a molecule to visualize (default is "none"). When a molecule is selected, it is shown at the center of the simulation box, and the group selection is disabled. Selecting "none" restores normal group-based display.
- **Global**: Opens the *Global image settings* dialog for fine-grained control of axes, box, background, quality, and center settings.
- **Atoms/Bonds**: Opens the *Atom and bond settings* dialog for detailed atom, bond, VDW, and special atom style visualization options.
- **Regions**: Opens the *Region settings* dialog to configure visualization of regions defined in the simulation.
- **Compute/Fix**: Opens the *Compute and fix graphics* dialog to enable and configure extra graphics objects provided by selected compute and fix styles.
- **Help**: Opens this online documentation page for the visualization features in LAMMPS-GUI in a web browser.

The image is re-rendered after each change to the buttons, text fields or settings dialogs, and when there are many atoms to render and high quality images with anti-aliasing are requested, re-rendering may take several seconds. Some time consuming rendering steps are multi-threaded, but there is no GPU acceleration.

Global image settings

While some persistent default settings for the image output can be configured in the "Snapshot Image" tab of the *Preferences dialog*, more fine-grained configuration is possible by opening the "Global image settings" dialog. However, settings not stored by the preference are reset when the image viewer window is close. This dialog is opened by pressing the "Global" button in the settings panel or by using the *Alt-L* keyboard shortcut. The settings in this dialog correspond to options of the LAMMPS dump image and dump_modify commands.



The dialog is organized into the following sections:

Axes

Controls the display of coordinate axes arrows in the image.

- **Axes** (checkbox): Enable or disable rendering of coordinate axes.
- **Location** (radio buttons): Select where the axes are drawn in the image (only available for LAMMPS versions that support this feature). Possible choices are: *Lower Left* (default), *Lower Right*, *Upper Left*, *Upper Right*, or *Center*.
- **Length**: The length of the axes arrows as a fraction of the box size (range: 0.00001 – 5.0).
- **Diameter**: The diameter of the axes arrows as a fraction of the box size (range: 0.00001 – 5.0).
- **Opacity**: The transparency of the axes (range: 0.0 – 1.0, where 1.0 is fully opaque and 0.0 is fully transparent; only available for LAMMPS versions that support this feature).

Box
Controls the display of the simulation box.

- **Box** (checkbox): Enable or disable rendering of the simulation box.
- **Color**: The color used to draw the box edges. Accepts named colors.
- **Diameter**: The diameter of the box edge sticks as fraction of the box size (range: 0.000001 – 5.0).
- **Opacity**: The transparency of the box edges (range: 0.0 – 1.0, where 1.0 is fully opaque and 0.0 is fully transparent; only available for LAMMPS versions that support this feature).

Subbox
Controls the display of the per-processor sub-domain boxes (relevant for MPI parallel simulations, will coincide with the regular box for LAMMPS-GUI runs).

- **Subbox** (checkbox): Enable or disable rendering of the sub-domain box.
- **Diameter**: The diameter of the sub-domain box edge sticks as fraction of the box size (range: 0.00001 – 5.0).

Background
Sets the background color(s) of the rendered image.

- **Bottomcolor**: The background color at the bottom of the image.
- **Topcolor**: The background color at the top of the image (Only available for LAMMPS versions that support this feature). If the two colors differ, a vertical gradient is applied from bottom to top.

Quality
Controls rendering quality options.

- **FSAA** (checkbox): Enable or disable full-scene anti-aliasing.
- **SSAO** (checkbox): Enable or disable Screen Space Ambient Occlusion for depth-shaded rendering.
- **SSAO strength**: The strength of the SSAO effect (range: 0.0 – 1.0).
- **Shiny**: The shininess factor for surface rendering (range: 0.0 – 1.0, where 0.0 is matte and 1.0 is fully shiny).

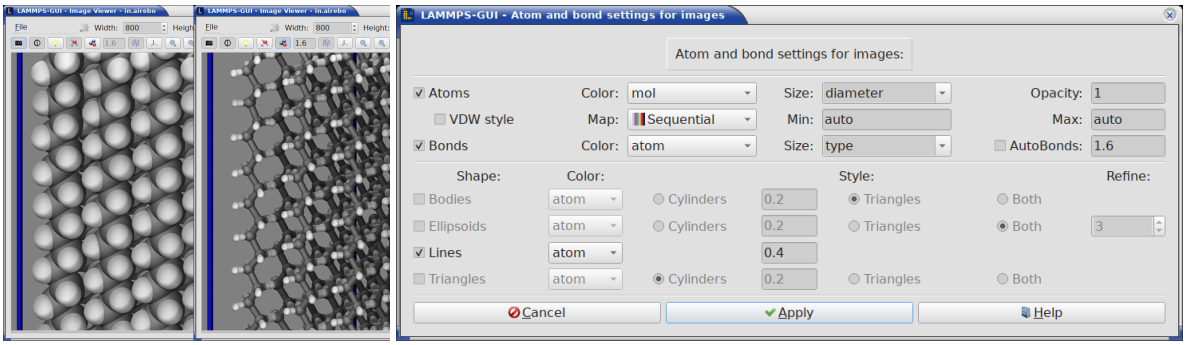
Center
Adjusts the center point of the rendered view.

- **X-direction**, **Y-direction**, **Z-direction**: Fractional coordinates (range: 0.0 – 1.0) specifying the center of the view relative to the simulation box.

Press **Apply** to apply the current settings and re-render the image, or **Cancel** to discard changes. The **Help** button opens the LAMMPS dump image documentation.

Atoms/bonds settings

This dialog offers more detailed customizations for atom and bond visualization that are not directly accessible from the main Image Viewer toolbar. It is opened by pressing the "Atoms/Bonds" button in the settings panel or by using the *Alt-A* keyboard shortcut.



The dialog contains the following sections:

Atoms

Controls how atoms are rendered.

- **Atoms** (checkbox): Enable or disable rendering of atoms.
- **Color**: Select the per-atom property used for coloring. Options include *type*, *element* (if detected), *mol*, *q* (charge), *diameter*, *id*, *mass*, *x*/*y*/*z* (coordinates), as well as atom-style variables (*v_<name>*), per-atom compute results (*c_<name>* or *c_<name>[col]*), and per-atom fix results (*f_<name>* or *f_<name>[col]*). The contents of the list depends on which are available in the current simulation state. When selecting *type* or *element* the colors are fixed and can be only changed manually for `dump_image` output in the input using `dump_modify acolor` or `dump_modify element`, respectively. Otherwise, the colors are determined by the colormap selection described below.
- **Size**: Select the property used for atom sizing. Options include *auto* (when element, diameter, or sigma data is available), *type*, *element*, and a few pre-defined choices for custom atom diameters. The text field can be edited and a different custom diameter entered.
- **Opacity**: The transparency of atoms *and* bonds (range: 0.0 – 1.0, where 1.0 is fully opaque and 0.0 is fully transparent; only available for LAMMPS versions supporting this feature).
- **VDW style** (checkbox): Enable or disable space-filling sphere rendering. When unchecked, the ball-and-stick style is used.
- **Colormap**: Select the colormap used for coloring by a per-atom property. This option is *not* available for atom color selections *type* and *element*. Currently available continuous colormaps are: *BWR* (blue-white-red), *RWB* (red-white-blue), *PWT* (purple-white-teal), *BWG* (blue-white-green), *BGR* (blue-green-red), “Viridis” (from matplotlib), “Plasma” (from matplotlib), “Inferno” (from matplotlib), “Teal”, “Rainbow”, and *Grayscale*. *Sequential* is a map with discrete colors. These are pre-defined colormap settings and cannot be adjusted from LAMMPS-GUI. As for *all* image settings, further customizations can be realized by copying the dump image command line as customized by the Image Viewer to the editor and then run LAMMPS and observe the resulting images in the Slideshow Viewer window. Then the color map setting can be fully customized according to the dump_modify colormap documentation.
- **Min / Max**: Set the range of the colormap. Use *auto* to have LAMMPS determine the range automatically or specify an explicit numeric value.

Bonds

Controls bond visualization.

- **Bonds** (checkbox): Enable or disable bond rendering. This option is only available when the atom style supports explicit bonds.
- **Color**: Select bond coloring mode – *atom* (colored by the atom type at each end) or *type* (uniform color per bond type).
- **Size**: Select bond diameter mode. Options include *atom*, *type*, and a few pre-defined choices for custom bond diameters. The text field can be edited and a different custom diameter entered.

- **AutoBonds** (checkbox): Automatically determine bonds from atom distances, useful for many-body force fields with implicit bonds like AIREBO or Tersoff. This feature depends on existing neighbor list data and thus may not always work as expected when the system has explicit bonds and thus neighbors may be automatically excluded from neighbor lists due to the special_bonds settings
- **Cutoff**: The distance cutoff used for automatic bond detection (range: 0.001 – 10.0 in distance units). Only available when auto-bonds are enabled.

Bodies

Controls visualization of body particles (when present in the simulation).

- **Bodies** (checkbox): Enable or disable rendering of body particle shapes. When disabled, the particles are rendered as spheres like regular atoms.
- **Color** (selection): Use coloring by the *atom* color choice, the body *index*, or the atom *type* of the body particles.
- **Style** (radio buttons): Select the body rendering style – *Cylinders*, *Triangles*, or *Both*. For cylinders - when used for body particle rendering - also their diameter can be set (range: 0.1 – 10.0).

Ellipsoids

Controls visualization of aspherical particles (when present in the simulation). Particles flagged as ellipsoids are represented as a triangle mesh, others as spheres.

- **Ellipsoids** (checkbox): Enable or disable rendering of ellipsoid particle shapes. When disabled, the particles are rendered as spheres like regular atoms.
- **Color** (selection): Use coloring by the *atom* color choice, the ellipsoid *index*, or the atom *type* of the ellipsoid particles.
- **Style** (radio buttons): Select the ellipsoid rendering style – *Cylinders*, *Triangles*, or *Both*. For cylinders - when used for ellipsoid particle rendering - also their diameter can be set (range: 0.1 – 10.0).
- **Refine** (spinbox): Level of triangle mesh refinement. At level 1 the ellipsoids are represented by a deformed octahedron. With a level increase, each triangle is replaced by 4 triangles following the ellipsoid shape more closely (max: 6). At the maximum level each ellipsoid is represented by 8192 triangles. At high refinement level, there may be artifacts from rounding due to limitations of the image rasterizer included in LAMMPS. These can be made less prominent by enabling anti-aliasing.

Lines

Controls visualization of line segment particles (when present in the simulation).

- **Lines** (checkbox): Enable or disable rendering of line segment particle shapes as connected cylinders. When disabled, the particles are rendered as spheres like regular atoms. Also the cylinder diameter can be set (range: 0.1 – 10.0).
- **Color** (selection): Use coloring by the *atom* color choice, the line *index*, or the atom *type* of the line particles.

Triangles

Controls visualization of triangulated particles (when present in the simulation).

- **Triangles** (checkbox): Enable or disable rendering of triangulated particle shapes. When disabled, the particles are rendered as spheres like regular atoms.
- **Color** (selection): Use coloring by the *atom* color choice, the triangulated particle *index*, or the atom *type* of the triangulated particles.
- **Style** (radio buttons): Select the particle rendering style – *Cylinders*, *Triangles*, or *Both*. For cylinders - when used for triangle particle rendering - also their diameter can be set (range: 0.1 – 10.0).

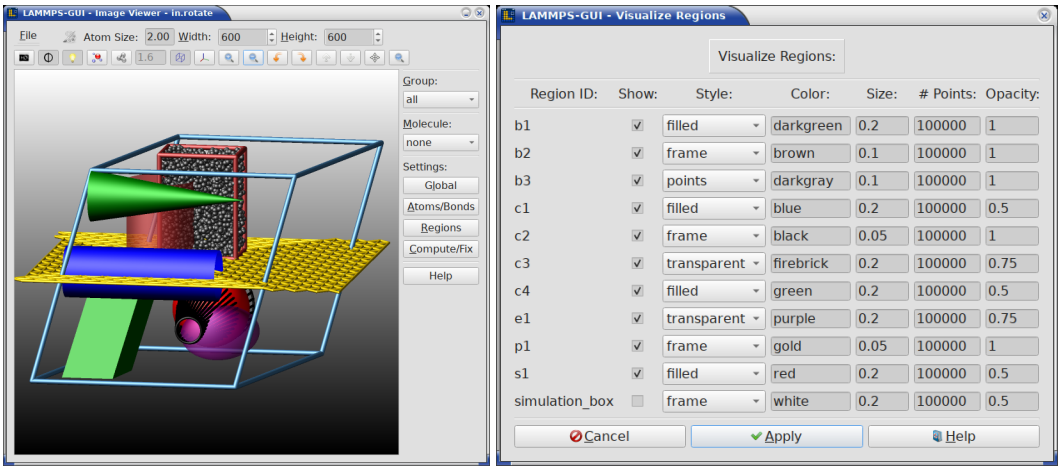
Press **Apply** to apply the settings and re-render the image, or **Cancel** to discard changes. The **Help** button opens the LAMMPS dump image documentation.

Note

Some options are only available when a sufficiently recent LAMMPS version is used and particles or the corresponding atom style are present. These fields are grayed out and disabled otherwise.

Region settings

This dialog allows enabling and configuring the visualization of regions defined in the LAMMPS input script. It is opened by pressing the "Regions" button in the settings panel or by using the *Alt-R* keyboard shortcut. The dialog only appears when at least one region is defined in the current simulation.



For each region, the following settings can be adjusted:

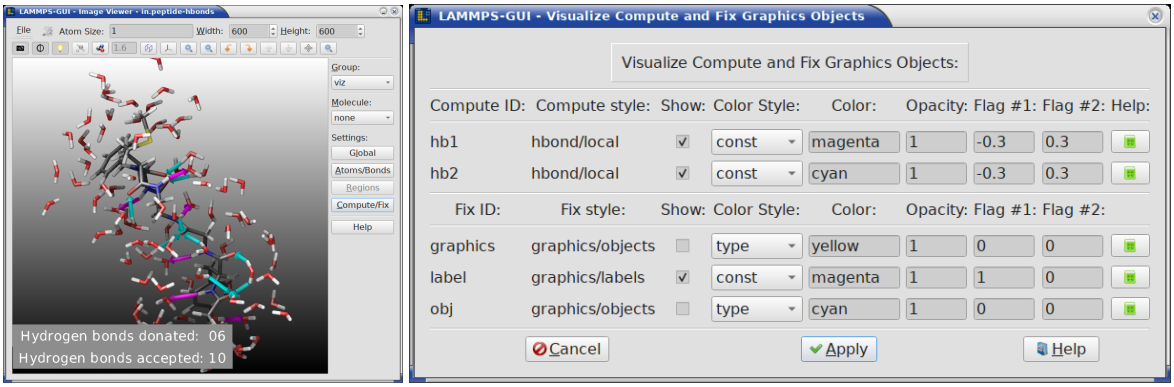
- **Region ID**: The identifier of the region (read-only).
- **Show** (checkbox): Enable or disable visualization of this region.
- **Style**: The rendering style for the region surface – *frame* (wireframe), *filled* (solid), *transparent* (see-through solid), or *points* (point cloud).
- **Color**: The color used to render the region. Accepts named colors or hex color values.
- **Size**: The diameter of the lines (for frame style) or points (for points style).
- **# Points**: The number of points used to approximate the region volume (range: 100 – 1,000,000). Higher values reproduce the volume better, but may obscure other details of the image.
- **Opacity**: The transparency of the region rendering (range: 0.0 – 1.0, where 1.0 is fully opaque and 0.0 fully transparent).

Press **Apply** to apply the settings and re-render the image, or **Cancel** to discard changes. The **Help** button opens the LAMMPS visualization howto documentation and jumps to the section discussing visualizing regions.

Graphics from computes and fixes

Starting with the 11 Feb 2026 version of LAMMPS, it is possible for compute and fix styles to prepare lists of graphics objects for inclusion into visualizations generated by the dump image command. This command is used internally by LAMMPS-GUI to create the snapshot image.

The “Visualize Compute and Fix Graphics Objects” dialog allows enabling these graphics objects and adjusting their settings. The dialog is opened by pressing the “Compute/Fix” button in the settings panel or by using the *Alt-C* keyboard shortcut. The dialog only appears when at least one compute or fix with graphics capabilities is defined.



For each compute or fix that supports graphics output, the following settings can be adjusted:

- **Compute/Fix ID**: The identifier of the compute or fix (read-only).
- **Style**: The compute or fix style name (read-only).
- **Show** (checkbox): Enable or disable visualization of the graphics objects from this compute or fix.
- **Color Style**: Select the coloring mode – *type*, *element*, or *const* (constant color).
- **Color**: The color to use when *const* color style is selected. Accepts named colors or hex color values.
- **Opacity**: The opacity of the graphics objects (range: 0.0 – 1.0).
- **Flag #1 / Flag #2**: Style-specific numeric flags that control additional rendering options. Their meaning depends on the specific compute or fix style.

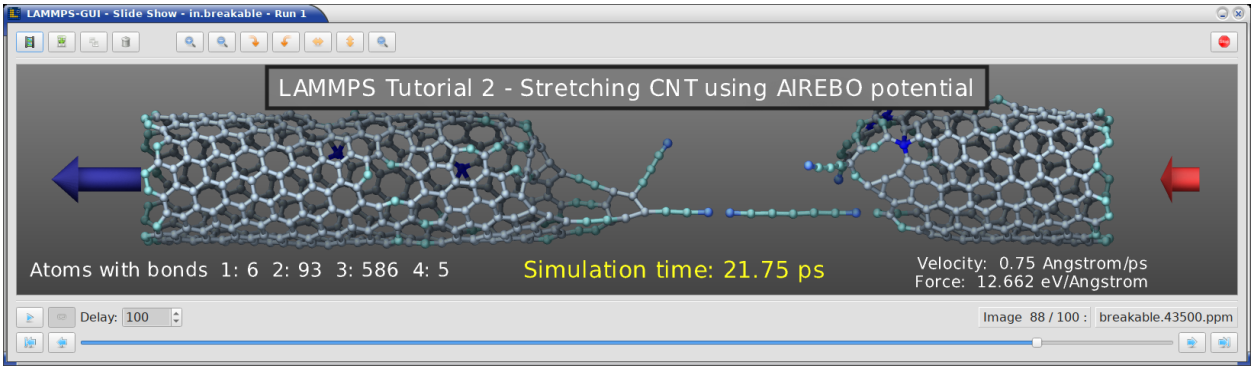
Each row also has a **Help** button that opens the LAMMPS documentation page for the corresponding compute or fix style, jumping directly to the “Dump image info” section that describes the available graphics objects and flag settings.

Press **Apply** to apply the settings and re-render the image, or **Cancel** to discard changes. The general **Help** button at the bottom opens the LAMMPS dump image documentation.

Image Slide Show

When running a LAMMPS input containing a dump image command with LAMMPS-GUI, the “Slide Show” window opens to load and display the images created by LAMMPS as they are written. This is a convenient way to visually monitor the progress of the simulation. It also can be used as an effective way to refine visualizations created with the *Snapshot Image Viewer*.

From the slide show window the following global keyboard shortcuts are supported: *Ctrl-W*: close window, *Ctrl-Q*: quit application, *Ctrl-/*: stop running simulation. Other keyboard shortcuts are connected to some of the controls and listed in their documentation below.



Slide show controls

There are controls and displays above and below the image. If you are uncertain about the function of a specific button, you can place the cursor on top of it and a descriptive tooltip will appear.

The **toolbar** at the top of the Slide Show window provides the following controls, organized from left to right:

- **Export to movie** (*Ctrl-E*): Export the entire image sequence to a movie file or animated GIF file. This requires that either the FFmpeg program or the ImageMagick software are installed. Supported output formats include MP4, MKV, AVI, MPG, MPEG, WEBM, and animated GIF. The file format is determined by the file name extension. Any active image transformations (rotation, mirroring, see below) are applied to the exported movie.
- **Save current image** (*Ctrl-S*): Save the currently displayed image to a file, including any applied transformations (rotation, mirroring, see below). The file format is inferred from the file name extension. When the ImageMagick software is installed, additional file formats beyond those natively supported by the Qt library become available.
- **Copy to clipboard** (*Ctrl-C*): Copy the current image to the system clipboard for pasting it into another application. This requires support from the receiving applications, but many applications like document editors or web browsers are.
- **Delete all images**: Remove all image files associated with the slide show. Since the number of image files can be large for long simulations, this provides a safe way to clean up the working directory without risk of accidentally deleting other files. This will, however, only delete the images of the last run. If that was stopped before completion or the output filename has changed, older images created by previous runs will not be deleted.
- **Zoom in**: Increase the displayed image size by scaling it up. Every click on the button increases the zoom factor by 10 percent.
- **Zoom out**: Decrease the displayed image size by scaling it down. Every click on the button decreases the zoom factor by 10 percent until a minimum zoom factor of 0.1.
- **Rotate clockwise**: Rotate the displayed image 90 degrees clockwise.
- **Rotate counter-clockwise**: Rotate the displayed image 90 degrees counter-clockwise.
- **Mirror horizontally**: Flip the displayed image along the vertical axis.
- **Mirror vertically**: Flip the displayed image along the horizontal axis.
- **Reset image**: Reset the displayed image to the original image. This reverts all zoom, rotate, and mirror operations.
- **Stop Simulation** (*Ctrl-/*): Stop a running simulation.

These image transformations are useful when the simulation images need to be adjusted for presentation purposes. The same transformations are also applied when exporting images or movies.

The **playback controls** below the image allow to select the displayed image and control the slideshow settings:

- **Play**: Start playing the animation from the current frame to the last frame.
- **Loop**: Toggle continuous looping of the animation. When enabled, playback wraps around from the last image back to the first.
- **Delay**: Set the delay in milliseconds between frames during animation playback.
- **First**: Jump to the first image in the sequence.
- **Previous**: Step back to the previous image.
- The **slider control** allows selecting a frame in the image sequence by by moving the slider position.
- **Next**: Step forward to the next image.
- **Last**: Jump to the last image in the sequence.

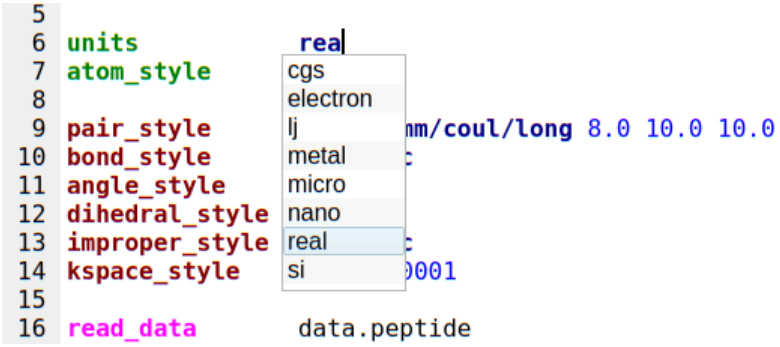
4.6 Editor Window

The *Editor* window of LAMMPS-GUI has most of the usual functionality that similar programs have: text selection via mouse or with cursor moves while holding the Shift key, Cut (*Ctrl-X*), Copy (*Ctrl-C*), Paste (*Ctrl-V*), Undo (*Ctrl-Z*), Redo (*Ctrl-Shift-Z*), Select All (*Ctrl-A*). When trying to exit the editor with a modified buffer, a dialog will pop up asking whether to cancel the exit operation, or to save or not save the buffer contents to a file.

The editor has an auto-save mode that can be enabled or disabled in the *Preferences* dialog. In auto-save mode, the editor buffer is automatically saved before running LAMMPS or before exiting LAMMPS-GUI.

Context Specific Word Completion

By default, LAMMPS-GUI displays a small pop-up frame with possible choices for LAMMPS input script commands or styles after 2 characters of a word have been typed.



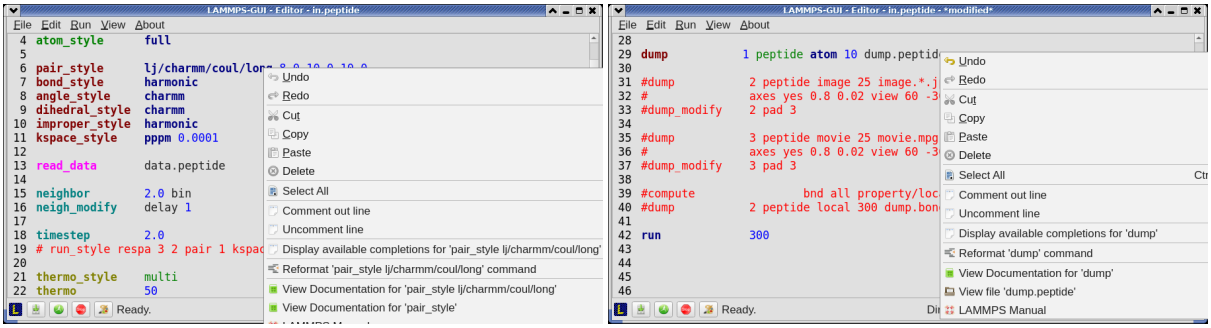
The word can then be completed through selecting an entry by scrolling up and down with the cursor keys and selecting with the 'Enter' key or by clicking on the entry with the mouse. The automatic completion pop-up can be disabled in the *Preferences* dialog, but the completion can still be requested manually by either hitting the 'Shift-TAB' key or by right-clicking with the mouse and selecting the option from the context menu. Most of the completion information is retrieved from the active LAMMPS instance and thus it shows only available options that have been enabled when compiling LAMMPS. That list, however, excludes accelerated styles and commands; for improved clarity, only the non-suffix version of styles are shown.

Line Reformatting

The editor supports reformatting lines according to the syntax in order to have consistently aligned lines. This primarily means adding whitespace padding to commands, type specifiers, IDs and names. This reformatting is performed manually by hitting the 'Tab' key. It is also possible to have this done automatically when hitting the 'Enter' key to start a new line. This feature can be turned on or off in the *Preferences* dialog for *Editor Settings* with the "Reformat with 'Enter'" checkbox. The amount of padding for multiple categories can be adjusted in the same dialog.

Internally this functionality is achieved by splitting the line into "words" and then putting it back together with padding added where the context can be detected; otherwise a single space is used between words.

Context Specific Help



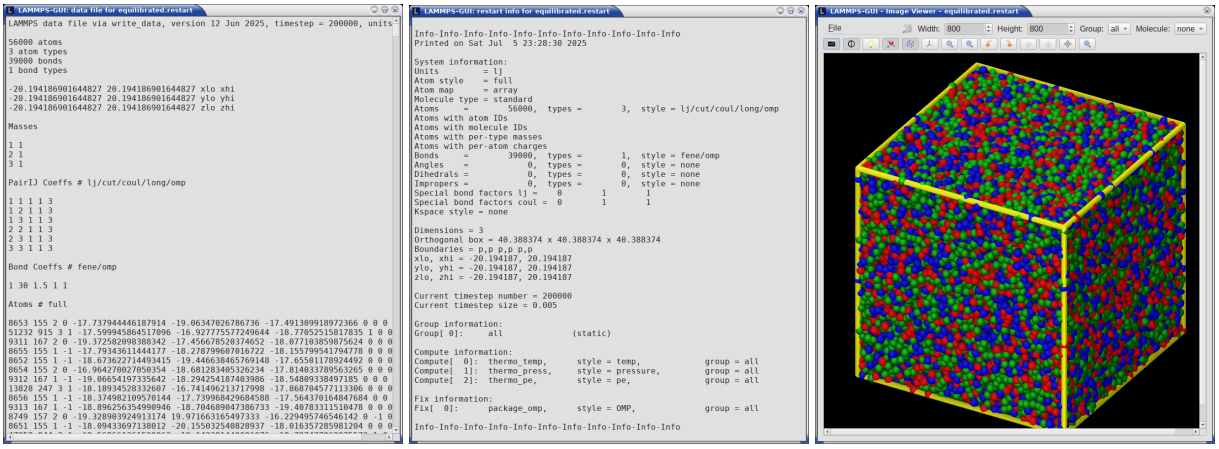
A unique feature of LAMMPS-GUI is the option to look up the LAMMPS documentation for the command in the current line. This can be done by either clicking the right mouse button or by using the *Ctrl-?* keyboard shortcut. When using the mouse, there are additional entries in the context menu that open the corresponding documentation page in the online LAMMPS documentation in a web browser window. When using the keyboard, the first of those entries is chosen.

If the word under the cursor is a file, then additionally the context menu has an entry to open the file in a read-only text viewer window. If the file is a LAMMPS restart file, instead the menu entry offers to *inspect the restart*.

The text viewer is a convenient way to view the contents of files that are referenced in the input. The file viewer also supports on-the-fly decompression based on the file name suffix in a similar fashion as available with LAMMPS. If the necessary decompression program is missing or the file cannot be decompressed, the viewer window will contain a corresponding message.

Inspecting a Restart file

When LAMMPS-GUI is asked to "Inspect a Restart", it will read the restart file into a LAMMPS instance and then open three different windows. The first window is a text viewer with the output of an info command with system information stored in the restart. The second window is text viewer containing a data file generated with a write_data command. The third window is a *Snapshot Image Viewer* containing a visualization of the system in the restart.



Large Restart Files

If the restart file is larger than 250 MBytes, a dialog will ask for confirmation before continuing, since large restart

files may require large amounts of RAM since the entire system must be read into RAM. Thus restart file for large simulations that have been run on an HPC cluster may overload a laptop or local workstation. The *Show Details...* button will display a rough estimate of the additional memory required.

4.7 Menus

The menu bar has entries *File*, *Edit*, *Run*, *View*, and *About*. Instead of using the mouse to click on them, the individual menus can also be activated by hitting the *Alt* key together with the corresponding underlined letter, that is *Alt-F* activates the *File* menu. For the corresponding activated sub-menus, the key corresponding the underlined letters can be used to select entries instead of using the mouse.

File

The *File* menu offers the usual options

- *New* clears the current buffer and resets the file name to `*unknown*`
- *Open* opens a dialog to select a new file for editing in the *Editor*
- *View* opens a dialog to select a file for viewing in a *separate* window (read-only) with support for on-the-fly decompression as explained above.
- *Inspect restart* opens a dialog to select a file. If that file is a LAMMPS restart three windows with *information about the file are opened*.
- *Save* saves the current file; if the file name is `*unknown*` a dialog will open to select a new file name
- *Save As* opens a dialog to select and new file name (and folder, if desired) and saves the buffer to it. Writing the buffer to a different folder will also switch the current working directory to that folder.
- *Quit* exits LAMMPS-GUI. If there are unsaved changes, a dialog will appear to either cancel the operation, or to save, or to not save the modified buffer.

In addition, up to 5 recent file names will be listed after the *Open* entry that allows re-opening recently opened files. This list is stored when quitting and recovered when starting again.

Edit

The *Edit* menu offers the usual editor functions like *Undo*, *Redo*, *Cut*, *Copy*, *Paste*, and a *Find and Replace* dialog (keyboard shortcut *Ctrl-F*). It can also open a *Preferences* dialog (keyboard shortcut *Ctrl-P*) and allows deleting all stored preferences and settings, so they are reset to their default values.

Run

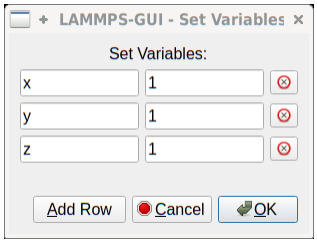
The *Run* menu has options to start and stop a LAMMPS process. Rather than calling the LAMMPS executable as a separate executable, the LAMMPS-GUI is linked to the LAMMPS library and thus can run LAMMPS internally through the LAMMPS C-library interface in a separate thread.

Specifically, a LAMMPS instance will be created by calling lammps_open_no_mpi. The buffer contents are then executed by calling lammps_commands_string. Certain commands and features are only available after a LAMMPS instance is created. Its presence is indicated by a small LAMMPS L logo in the status bar at the bottom left of the main window. As an alternative, it is also possible to run LAMMPS using the contents of the edited file by reading the file. This is mainly provided as a fallback option in case the input uses some feature that is not available when running from a string buffer.

The LAMMPS calculations are run in a concurrent thread so that the GUI can stay responsive and be updated during the run. The GUI can retrieve data from the running LAMMPS instance and tell it to stop at the next timestep. The *Stop LAMMPS* entry will do this by calling the lammps_force_timeout library function, which is equivalent to a timer timeout 0 command.

The *Relaunch LAMMPS Instance* will destroy the current LAMMPS thread and free its data and then create a new thread with a new LAMMPS instance. This is usually not needed, since LAMMPS-GUI tries to detect when this is needed and does it automatically. This is available in case it missed something and LAMMPS behaves in unexpected ways.

The *Set Variables...* entry opens a dialog box where index style variables can be set. Those variables are passed to the LAMMPS instance when it is created and are thus set *before* a run is started.



The *Set Variables* dialog will be pre-populated with entries that are set as index variables in the input and any variables that are used but not defined, if the built-in parser can detect them. New rows for additional variables can be added through the *Add Row* button and existing rows can be deleted by clicking on the *X* icons on the right.

The *Create Image* entry will send a dump image command to the LAMMPS instance, read the resulting file, and show it in an *Image Viewer* window.

The *View in OVITO* entry will launch OVITO with a data file containing the current state of the system. This option is only available if LAMMPS-GUI can find the OVITO executable in the system path.

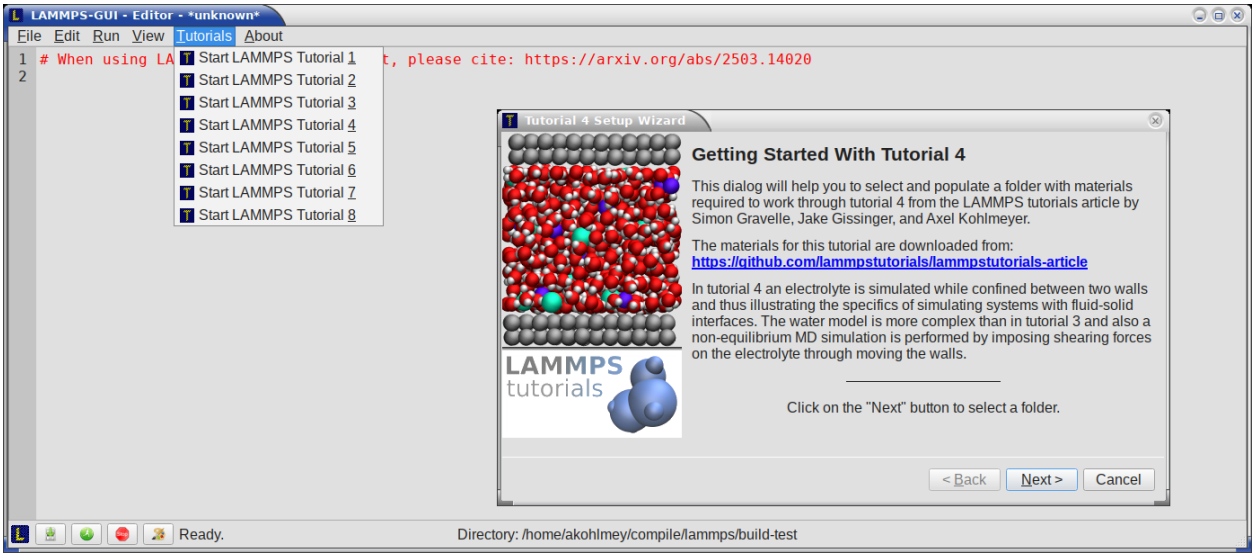
The *View in VMD* entry will launch VMD with a data file containing the current state of the system. This option is only available if LAMMPS-GUI can find the VMD executable in the system path.

View

The *View* menu offers to show or hide additional windows with log output, charts, slide show, variables, or snapshot images. The default settings for their visibility can be changed in the *Preferences* dialog.

Tutorials

The *Tutorials* menu is to support the set of LAMMPS tutorials for beginners and intermediate LAMMPS users documented in (*Gravelle1*). From the drop down menu you can select which of the eight currently available tutorial sessions you want to begin. This opens a 'wizard' dialog where you can choose in which folder you want to work, whether you want that folder to be wiped from *any* files, whether you want to download the solution files (which can be large) to a `solution` sub-folder, and whether you want the corresponding tutorial's online version opened in your web browser. The dialog will then start downloading the files requested (download progress is reported in the status line) and load the first input file for the selected session into LAMMPS-GUI.



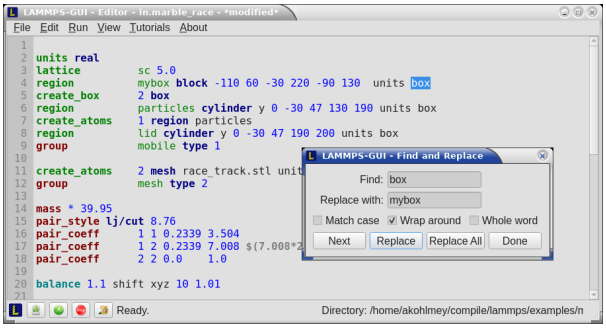
About

The *About* menu finally offers a couple of dialog windows and an option to launch the LAMMPS online documentation in a web browser. The *About LAMMPS-GUI* entry displays a dialog with a summary of the configuration settings of the LAMMPS library in use and the version number of LAMMPS-GUI itself. The *Quick Help* displays a dialog with a minimal description of LAMMPS-GUI. The *LAMMPS-GUI Documentation* entry will open the LAMMPS-GUI online documentation website https://lammps-gui.lammps.org in a web browser window. The *LAMMPS Manual* entry will open the main page of the LAMMPS online documentation in a web browser window. The *LAMMPS Tutorial* entry will open the main page of the set of LAMMPS tutorials authored and maintained by Simon Gravelle at https://lammpstutorials.github.io/ in a web browser window.

(Gravelle1) Gravelle, Alvares, Gissinger, Kohlmeyer, Living Journal of Computational Molecular Science, 6(1), 3027. https://doi.org/10.33011/livecoms.6.1.3037 (2025)

4.8 Dialogs

Find and Replace



The *Find and Replace* dialog allows searching for and replacing text in the *Editor* window.

The dialog can be opened either from the *Edit* menu or with the keyboard shortcut *Ctrl-F*. You can enter the text to search for.

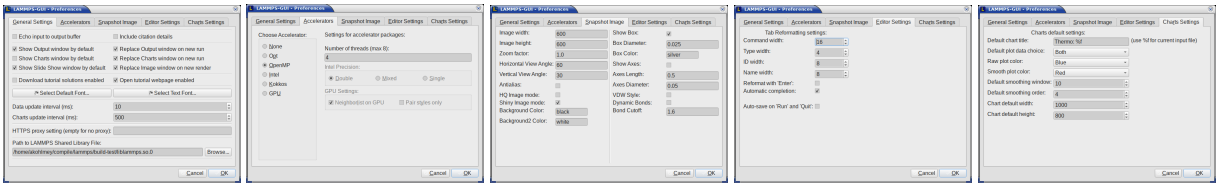
Through three check-boxes the search behavior can be adjusted

- If checked, “Match case” does a case sensitive search; otherwise the search is case insensitive.
- If checked, “Wrap around” starts searching from the start of the document, if there is no match found from the current cursor position until the end of the document; otherwise the search will stop.
- If checked, the “Whole word” setting only finds full word matches (white space and special characters are word boundaries).

Clicking on the *Next* button will search for the next occurrence of the search text and select / highlight it. Clicking on the *Replace* button will replace an already highlighted search text and find the next one. If no text is selected, or the selected text does not match the selection string, then the first click on the *Replace* button will only search and highlight the next occurrence of the search string. Clicking on the *Replace All* button will replace all occurrences from the cursor position to the end of the file; if the *Wrap around* box is checked, then it will replace **all** occurrences in the **entire** document. Clicking on the *Done* button will dismiss the dialog.

Preferences

The *Preferences* dialog allows customization of the behavior and look of LAMMPS-GUI. The settings are grouped and each group is displayed within a tab.



General Settings:

The following settings are available in this tab

- **Echo input to log:** when checked, all input commands, including variable expansions, are echoed to the *Output window*. This is equivalent to using *-echo screen* at the command-line. There is no log *file* produced by default, since LAMMPS-GUI uses *-log none*.
- **Include citation details:** when checked full citation info will be included to the log window. This is equivalent to using *-cite screen* on the command-line.
- **Show log window by default:** when checked, the screen output of a LAMMPS run will be collected in a log window during the run
- **Show chart window by default:** when checked, the thermodynamic output of a LAMMPS run will be collected and displayed in a chart window as line graphs.
- **Show slide show window by default:** when checked, a slide show window will be shown with images from a dump image command, if present, in the LAMMPS input.
- **Replace log window on new run:** when checked, an existing log window will be replaced on a new LAMMPS run, otherwise each run will create a new log window.
- **Replace chart window on new run:** when checked, an existing chart window will be replaced on a new LAMMPS run, otherwise each run will create a new chart window.

- **Replace image window on new render:** when checked, an existing chart window will be replaced when a new snapshot image is requested, otherwise each command will create a new image window.
- **Download tutorial solutions enabled** this controls whether the "Download solutions" option is enabled by default when setting up a tutorial.
- **Open tutorial webpage enabled** this controls whether the "Open tutorial webpage in web browser" option is enabled by default when setting up a tutorial.
- **Select Default Font:** Opens a font selection dialog where the type and size for the default font (used for everything but the editor and log) of the application can be set.
- **Select Text Font:** Opens a font selection dialog where the type and size for the text editor and log font of the application can be set.
- **Data update interval:** Allows to set the time interval between data updates during a LAMMPS run in milliseconds. The default is to update the data (for charts and output window) every 10 milliseconds. This is good for many cases. Set this to 100 milliseconds or more if LAMMPS-GUI consumes too many resources during a run. For LAMMPS runs that run *very* fast (for example in tutorial examples), however, data may be missed and through lowering this interval, this can be corrected. However, this will make the GUI use more resources. This setting may be changed to a value between 1 and 1000 milliseconds.
- **Charts update interval:** Allows to set the time interval between redrawing the plots in the *Charts window* in milliseconds. The default is to redraw the plots every 500 milliseconds. This is just for the drawing, data collection is managed with the previous setting.
- **HTTPS proxy setting:** Allows to enter a URL for an HTTPS proxy. This may be needed when the LAMMPS input contains geturl commands or for downloading tutorial files from the *Tutorials* menu. If the `https_proxy` environment variable was set externally, its value is displayed but cannot be changed.
- **Path to LAMMPS Shared Library File:** this option is only visible when LAMMPS-GUI was compiled to load the LAMMPS library at run time instead of being linked to it directly. With the *Browse..* button or by changing the text, a different shared library file with a different compilation of LAMMPS with different settings or from a different version can be loaded. After this setting was changed, LAMMPS-GUI needs to be re-launched.

Accelerators:

This tab enables selection of an accelerator package and modification of some of its settings for use when running LAMMPS. This is equivalent to using the -sf and -pk flags on the command-line. Only settings supported by the LAMMPS library and local hardware are available. The *Number of threads* field allows setting the number of threads for the accelerator packages that support using threads (OPENMP, INTEL, KOKKOS, and GPU). Furthermore, the choice of precision mode (double, mixed, or single) for the INTEL package can be selected, and for the GPU package, whether the neighbor lists are built on the GPU or the host (required for pair style hybrid) and whether only pair styles should be accelerated (i.e., run PPPM entirely on the CPU, which sometimes leads to better overall performance). Whether settings can be changed depends on which accelerator package is chosen (or "None").

Snapshot Image:

This tab allows setting defaults for the snapshot images displayed in the *Image Viewer window*, such as its dimensions, the zoom factor, and view angles. The **Antialias** switch will render images with double the number of pixels for width and height and then smoothly scale the image back to the requested size. This produces higher quality images with smoother edges at the expense of requiring more CPU time to render a four times size initial image. The **HQ Image mode** option turns on "Screen Space Ambient Occlusion (SSAO)" mode when rendering images. This is also more time consuming, but produces a more 'spatial' representation of the system with shading of atoms by their depth. The **Shiny Image mode** option will render objects with a shiny surface when enabled. Otherwise, the surfaces will be matte. The **Show Box** option selects whether the system box is drawn as a colored set of sticks. Furthermore, the diameter

of the sticks and their color can be set. Similarly, the **Show Axes** option selects whether a representation of the three system axes will be drawn or not (as colored and labeled arrows). In addition, the axes length and diameter can be set in fractions of the image size. The **VDW Style** checkbox selects whether atoms are represented by space filling spheres when checked or by smaller spheres and sticks. The **Dynamic Bonds** checkbox selects whether bonds between atoms shall be automatically determined from the atom distances (for instance to visualize simulations using force fields with implicit bonds), and the corresponding "Bond Cutoff" text field allow to set the cutoff used for that feature. Finally, there are a couple of text fields to select the two **Background Colors**. If the two colors differ, there will be a vertical background gradient starting with the "Background" color at the bottom and ending with the "Background2" color at the top.

These settings correspond to the available settings for the LAMMPS dump image and corresponding dump_modify commands.

Editor Settings:

This tab allows tweaking settings of the *editor window*. Specifically, the amount of padding to be added to LAMMPS commands, types or type ranges, IDs (e.g., for fixes), and names (e.g., for groups). The value set is the minimum width for the text element and it can be chosen in the range between 1 and 32.

The three settings which follow enable or disable the automatic reformatting when hitting the 'Enter' key, the automatic display of the completion pop-up window, and whether auto-save mode is enabled. In auto-save mode the editor buffer is saved before a run or before exiting LAMMPS-GUI.

Charts Settings:

This tab allows tweaking settings of the *Charts window*. Specifically, one can set the default chart title (if the title contains '%f' it will be replaced with the name of the current input file), one can select whether by default the raw data, the smoothed data, or both will be plotted, one can set the colors for the two lines, the default smoothing parameters, the default size of the chart graph in pixels, and whether you want to display major and minor grid lines.

4.9 Keyboard Shortcuts

Almost all functionality is accessible from the menu of the editor window or through keyboard shortcuts. The following shortcuts are available (On macOS use the Command key instead of Ctrl/Control).

Shortcut	Function	Shortcut	Function	Shortcut	Function
Ctrl+N	New File	Ctrl+Z	Undo edit	Ctrl+Enter	Run Input
Ctrl+O	Open File	Ctrl+Shift+Z	Redo edit	Ctrl+/	Stop Active Run
Ctrl+Shift+F	View File	Ctrl+C	Copy text	Ctrl+Shift+V	Set Variables
Ctrl+S	Save File	Ctrl+X	Cut text	Ctrl+I	Snapshot Image
Ctrl+Shift+S	Save File As	Ctrl+V	Paste text	Ctrl+L	Slide Show
Ctrl+Q	Quit Application	Ctrl+A	Select All	Ctrl+F	Find and Replace
Ctrl+W	Close Window	TAB	Reformat line	Shift+TAB	Show Completions
Ctrl+Shift+Enter	Run File	Ctrl+Shift+W	Show Variables	Ctrl+P	Preferences
Ctrl+Shift+A	About LAMMPS	Ctrl+Shift+H	Quick Help	Ctrl+Shift+G	LAMMPS-GUI Docs
Ctrl+Shift+R	Inspect Restart	Ctrl+Shift+O	View in OVITO	Ctrl+Shift+D	View in VMD
Ctrl+Shift+L	Output Window	Ctrl+Shift+C	Charts Window	Ctrl+Shift+I	Image Window
Ctrl+Shift+M	LAMMPS Manual	Ctrl+?	Context Help	Ctrl+Shift+T	LAMMPS Tutorial

Further keybindings of the editor window are documented with the Qt documentation. In case of conflicts the list above takes precedence.

All other windows only support a subset of keyboard shortcuts listed above. Typically, the shortcuts *Ctrl-/* (Stop Run), *Ctrl-W* (Close Window), and *Ctrl-Q* (Quit Application) are supported.

5 Programmer's Guide

This guide provides documentation for developers who want to understand the internals of LAMMPS-GUI or contribute to its development.

AI Generated Content

The initial version of the Programmer's Guide section was created by the GitHub Copilot Coding Agent and not everything has been carefully checked. It is therefore possible that it contains errors where the LLM has misinterpreted the LAMMPS-GUI source code. If you spot any such errors or inconsistencies, please submit a bug report issue to point them out or - even better - submit a pull request with the necessary corrections.

5.1 Overview

LAMMPS-GUI is built using C++17 and the Qt framework (Qt 5.15+ or Qt 6.2+). The application follows object-oriented design principles with separation of concerns between different components:

- **Editor Components**: Handle text editing, syntax highlighting, and auto-completion
- **LAMMPS Interface**: Wraps the LAMMPS C library API
- **Visualization**: Displays images, charts, and simulation output
- **GUI Framework**: Main window, dialogs, and preferences

LAMMPS Interface

LAMMPS-GUI can operate in two modes: **Plugin Mode** and **Linked Mode**. The mode is controlled by the `-D LAMMPS_GUI_USE_PLUGIN=(ON|OFF)` CMake configuration option.

Plugin Mode (default)
: LAMMPS is loaded dynamically at runtime from a shared library file (.so, .dll, .dylib). This allows using different LAMMPS builds with different compilation setting and different LAMMPS versions without recompiling the GUI. The library loading is handled in *`LammpsWrapper`* using platform-specific dynamic loading functions (`dlopen()` on Unix/Linux/macOS, `LoadLibrary()` on Windows). The path to the shared library file is auto-detected or configured via command line or preferences.

Linked Mode LAMMPS library is linked at compile time. Used by
: default when building LAMMPS-GUI as part of a LAMMPS CMake build with `-D BUILD_LAMMPS_GUI=on`. For standalone builds also the `-D LAMMPS_SOURCE_DIR=<path to LAMMPS' src folder>` and `-D LAMMPS_LIBRARY=<path to LAMMPS shared or static library file>` settings are required when configuring with CMake. It may be needed to adjust the environment variable to find shared libraries (`LD_LIBRARY_PATH` on Linux, `DYLD_LIBRARY_PATH` on macOS, or `PATH` on Windows) when linked to a shared library.

Qt Integration

LAMMPS-GUI makes extensive use of Qt features:

Signals and Slots
: Used for inter-component communication, especially between GUI components and background threads.

Qt Designer Forms
: The main window layout uses a `.ui` file edited in Qt Designer. Dialogs are created programmatically in C++.

Qt Resource System
: Icons and resources embedded via `resources/lammpsgui.qrc`.

Qt Models
: Used for data display in various viewers and inspectors.

LAMMPS-GUI is compatible with Qt version 5.15 or Qt 6.2 and later. If a Qt 6.x version is available it is preferred unless the CMake setting `-D LAMMPS_GUI_USE_QT5=yes` is used. For Qt versions 6.10 and later the QtGraphs module is used for creating charts instead of the QtCharts module. Using QtCharts can be enforced by setting `-D LAMMPS_GUI_USE_QTCHARTS=yes`.

For more details on Qt in general, see the Qt Documentation.

5.2 Architecture

Main Components

The application architecture consists of several key components organized into functional groups:

Main Window

LammpsGui (lammpsgui.h/.cpp)
: The main window class that coordinates all other components. It manages the editor, handles file operations, controls LAMMPS execution, and manages the overall application state. This is the central hub of the application that integrates all other components. See *LammpsGui*

Editor Components

CodeEditor (codeeditor.h/.cpp)
: Custom text editor widget based on QPlainTextEdit, providing LAMMPS-specific features including syntax highlighting, auto-completion, line numbers, and context-sensitive help. The main editing surface for LAMMPS input scripts. See *CodeEditor*

LineNumberArea (linenumberarea.h)
: Widget that displays line numbers in the left margin of the CodeEditor. Updates dynamically as text is added or removed. See *LineNumberArea*

Highlighter (highlighter.h/.cpp)
: Syntax highlighter for LAMMPS input scripts. Categorizes and colors different types of commands, keywords, variables, and comments using Qt's QSyntaxHighlighter framework. See *Highlighter*

FindAndReplace (findandreplace.h/.cpp)
: Dialog for searching and replacing text in the editor. Supports case sensitivity and whole word matching options. See *FindAndReplace*

LAMMPS Interface

LammpsWrapper (lammpswrapper.h/.cpp)
: C++ wrapper around the LAMMPS C library interface. Provides a clean C++ API and handles dynamic library loading in plugin mode. Manages LAMMPS initialization, command execution, and error handling. See *LammpsWrapper*

LammpsRunner (lammpsrunner.h)
: Worker thread for executing LAMMPS simulations without blocking the GUI. Uses Qt's threading facilities to run simulations in the background, allowing the UI to remain responsive during long calculations. See *LammpsRunner*

Visualization Components

ImageViewer (imageviewer.h/.cpp)
Dialog for viewing and manipulating LAMMPS snapshot images created by the `dump image` command. Supports interactive control of visualization parameters such as zoom, rotation, atom size, coloring, and rendering options. Changes can be applied to regenerate the image using the LAMMPS library interface. See *ImageViewer*. This uses two internal helper classes:

- **ImageInfo** - Stores settings for displaying graphics from a LAMMPS compute or fix in snapshot images.
- **RegionInfo** - Stores settings for displaying a region in snapshot images.

ChartWindow (chartviewer.h/.cpp)
Window for displaying thermodynamic data as charts using Qt Charts. Supports line plots and multiple data series. See *ChartWindow*

ChartViewer (chartviewer.h/.cpp) Custom chart view widget that
provides interactive features like zooming, smoothing, and panning for data visualization. See `ChartViewer`.

Two implementations exist: The older is based on the QChartView widget from the QtCharts module and the newer implementation is based on QQuickWidget and the QtGraphs module. For Qt versions 6.10 and later the QtGraphs module variant is used and for older versions the QtGraphs module variant. Using the QtCharts can be enforced by setting `-D LAMMPS_GUI_USE_QTCHARTS=yes`.

SlideShow (slideshow.h/.cpp)
Dialog for viewing multiple images as a slideshow or animation with navigation controls. Supports converting an animation to a movie file when FFmpeg or ImageMagick is available. See *SlideShow*

RangeSlider (rangeslider.h/.cpp)
Custom slider widget with two handles for selecting a range of values. This is code written by Hoyoung Lee and distributed under the CeCILL-A license as circulated by CEA, CNRS and INRIA at the following URL: "http://www.cecill.info". Used in *ChartWindow* for selecting x- and y-direction plot ranges. See *RangeSlider*

Dialog and Utility Components

LogWindow (logwindow.h/.cpp)
Window displaying captured output from LAMMPS simulations. Updates in real-time as the simulation progresses and highlights warning and error messages. Provides navigation to jump between warnings. See *LogWindow*

Preferences (preferences.h/.cpp)
Dialog for configuring application settings including accelerator packages, editor appearance, snapshot settings, and chart preferences. Settings are made persistent across LAMMPS-GUI sessions using the QSettings class. See *Preferences*. The dialog is organized into five tabs, each implemented as a separate widget class:

- *GeneralTab* - General settings (LAMMPS library path, fonts, etc.)
- *AcceleratorTab* - LAMMPS accelerator package configuration
- *SnapshotTab* - Snapshot image viewer defaults
- *EditorTab* - Editor appearance and behavior settings
- *ChartsTab* - Chart viewer display settings

SetVariables (setvariables.h/.cpp)
Dialog for editing LAMMPS index-style variable definitions. Allows users to define name-value pairs that are substituted in input scripts using `${varname}` syntax. See *SetVariables*

FileViewer (fileviewer.h/.cpp)
Read-only text viewer dialog for displaying file contents. Used for viewing auxiliary files without allowing modifications. See *FileViewer*

TutorialWizard (lammpsgui.h/.cpp)
: Wizard dialog for interactive LAMMPS tutorials. Guides users through setting up tutorial directories and files, providing a structured learning experience. See *TutorialWizard*

AboutDialog (aboutdialog.h/.cpp)
: Custom About dialog that displays version information, LAMMPS configuration details, and available styles in two scrollable text areas. The dialog automatically scrolls down when the content exceeds the visible area, pauses at the bottom, and then returns back to the top. See *AboutDialog*

Support Components

StdCapture (stdcapture.h/.cpp)
: Utility class that captures stdout and stderr output from LAMMPS. Redirects C-level file descriptors to allow capturing output from the LAMMPS library. See *StdCapture*

FlagWarnings (flagwarnings.h/.cpp)
: Syntax highlighter for LAMMPS warning and error messages in log output. Detects and highlights WARNING/ERROR lines and URLs for documentation links. Maintains a count of warnings and updates a summary label. See *FlagWarnings*

QHline (qaddon.h/.cpp)
: Simple horizontal line widget for visual separation in dialogs and forms. See *QHline*

QColorCompleter (qaddon.h/.cpp)
: Auto-completer for color name inputs, suggesting valid Qt color names as the user types. See *QColorCompleter*

QColorValidator (qaddon.h/.cpp)
: Validator for color input fields, ensuring they contain valid color names or hex color codes. See *QColorValidator*

Helper Functions

The *helpers module* provides utility functions used throughout the application:

- String manipulation (mystrdup variants for different string types)
- Date comparison (date_compare for version comparisons)
- Command-line parsing (split_line with quote handling)
- System utilities (has_exe for executable detection)
- UI utilities (is_light_theme for theme detection)

Data Flow

1. **User Input**: User edits LAMMPS input in CodeEditor with syntax highlighting
2. **Execution Request**: User triggers execution via menu or button
3. **Preparation**: LammpsGui creates/configures LammpsWrapper and prepares variables
4. **Threading**: Commands sent to LammpsRunner thread to avoid UI blocking
5. **Execution**: LammpsRunner executes commands via LammpsWrapper
6. **Output Capture**: Output captured via StdCapture for display
7. **Visualization**: Results displayed in LogWindow, ImageViewer, or ChartWindow
8. **Completion**: UI updated when execution completes, progress indicators cleared

Settings and State Management

The application uses Qt's QSettings mechanism to persist:

- Recent files list
- Window geometry and state
- Editor preferences (font, colors)
- Accelerator settings
- LAMMPS plugin path
- Tutorial preferences

Settings are stored in platform-specific locations (the application name includes the Qt major version, e.g. `LAMMPS-GUI (QT6)`):

- Linux: `~/.config/The LAMMPS Developers/LAMMPS-GUI (QT6).conf`
- macOS: `~/Library/Preferences/org.lammps.LAMMPS-GUI (QT6).plist`
- Windows: Registry under `HKEY_CURRENT_USER\Software\The LAMMPS Developers\LAMMPS-GUI (QT6)`

Threading Model

The application uses Qt's event-driven architecture with threading:

- **Main Thread**: Handles all UI operations and user interactions
- **LAMMPS Thread**: LammpsRunner executes LAMMPS in a separate QThread
- **Communication**: Signals/slots for thread-safe communication between threads

This design keeps the UI responsive even during long-running simulations.

5.3 API Reference

The following sections provide detailed API documentation for the main classes in LAMMPS-GUI. Documentation is generated from Doxygen comments in the source code.

Main Window

LammpsGui Class

class **LammpsGui** : public QMainWindow

Main application window for LAMMPS-GUI.

LammpsGui is the central component of the LAMMPS-GUI application, serving as the main window that coordinates all other components. It manages:

- The code editor for LAMMPS input scripts with syntax highlighting
- File operations (open, save, recent files)
- LAMMPS simulation execution and control
- Visualization windows (images, charts, log output)
- Application preferences and settings
- Tutorial wizard for interactive LAMMPS learning

The class integrates with Qt's main window framework and provides menu actions, toolbars, and status bar components. It uses a *LammpsWrapper* to interface with the LAMMPS library and *LammpsRunner* to execute simulations in a separate thread.

See also

CodeEditor for the text editor component

See also

ChartWindow for the charts window component

See also

LogWindow for the log output window component

See also

ImageViewer for the snapshot image window component

See also

SlideShow for the slide show viewer window component

See also

Preferences for the preferences window component

See also

LammpsRunner for simulation execution in a separate thread

See also

LammpsWrapper for LAMMPS library interface

Public Functions

LammpsGui(QWidget *parent = nullptr, const QString &filename = QString(), int width = 0, int height = 0)

Construct the main application window.

Initializes the main window, sets up the UI components, loads preferences, initializes the LAMMPS library, and optionally opens a file if provided.

Parameters

- **parent** – Parent widget (typically nullptr for main window)
- **filename** – Optional file to open on startup
- **width** – Optional main editor window width override
- **height** – Optional main editor window height override

~LammpsGui() override

Destructor.

Cleans up resources including dynamically created widgets and LAMMPS instances.

LammpsGui() = delete

LammpsGui(const *LammpsGui*&) = delete

LammpsGui(*LammpsGui*&&) = delete

LammpsGui &**operator=**(const *LammpsGui*&) = delete

LammpsGui &**operator=**(*LammpsGui*&&) = delete

Public Slots

void **quit**()

Quit the application.

void **stop_run**()

Stop a running LAMMPS simulation.

Protected Functions

void **open_file**(const QString &filename)

Open a file in the editor.

void **view_file**(const QString &filename)

Open a file in a read-only viewer dialog.

void **inspect_file**(const QString &filename)

Open a file for inspection (data files, etc.)

void **write_file**(const QString &filename)

Write current editor content to a file.

void **update_recents**(const QString &filename = "")

Update the recent files list.

void **clear_variables**()

Clear the list of index-style variables.

void **update_variables**()

Update variables in LAMMPS from the variables list.

void **do_run**(bool use_buffer)

Execute a LAMMPS simulation.

Parameters
use_buffer – If true, runs from editor buffer; if false, saves and runs from file

void **start_lammps**()

Initialize and start a new LAMMPS instance.

void **run_done**()

Handle completion of a LAMMPS run.

void **setDocver**()

Set the documentation version string for help links.

void **autoSave**()

Perform an auto-save of the current file.

void **setFont**(const QFont &newfont)

Update the editor font.

Parameters
newfont – The font to apply to the editor

QWizardPage ***tutorial_intro**(const int ntutorial, const QString &infotext)

Create an introduction page for a tutorial.

Parameters

- **ntutorial** – Tutorial number
- **infotext** – Information text to display

Returns
Wizard page with tutorial introduction

QWizardPage ***tutorial_directory**(const int ntutorial)

Create a directory selection page for a tutorial.

Parameters
ntutorial – Tutorial number

Returns
Wizard page for tutorial directory selection

void **setup_tutorial**(int tutno, const QString &dir, bool purgedir, bool getsolution, bool openwebpage)

Set up files and resources for a tutorial.

Parameters

- **tutno** – Tutorial number
- **dir** – Directory to create files in
- **purgedir** – Whether to clean the directory first
- **getsolution** – Whether to include solution files
- **openwebpage** – Whether to open the tutorial web page

void **purge_inspect_list**()

Clean up the inspect file dialog list.

bool **eventFilter**(QObject *watched, QEvent *event) override

Event filter for handling special events.

Parameters

- **watched** – Object being watched
- **event** – Event to filter

Returns

true if event was handled

Protected Attributes

Ui::LammpsGui ***ui**

UI components generated from .ui file.

int **nthreads**

Number of threads for parallel execution.

int **mainx**

Override value for main editor window width or 0.

int **mainy**

Override value for main editor window height or 0.

TutorialWizard Class

class **TutorialWizard** : public QWizard

Wizard dialog for interactive LAMMPS tutorials.

TutorialWizard provides a step-by-step wizard interface for setting up and running LAMMPS tutorials. It guides users through directory selection, file preparation, and launching tutorial exercises.

Public Functions

TutorialWizard(int ntutorial, QWidget *parent = nullptr)

Construct a tutorial wizard.

Parameters

- **ntutorial** – Tutorial number (1-8)

- **parent** – Parent widget

void **accept**() override

Accept the wizard and set up the tutorial.

Called when the user completes the wizard. Sets up tutorial files and opens the tutorial in the main window.

Editor Components

CodeEditor Class

class **CodeEditor** : public QPlainTextEdit

Custom text editor with LAMMPS syntax support and auto-completion.

The *CodeEditor* class extends QPlainTextEdit to provide specialized features for editing LAMMPS input scripts, including:

- Line numbers in a margin area
- LAMMPS syntax highlighting via *Highlighter*
- Context-aware auto-completion for LAMMPS commands
- Automatic indentation and formatting
- Context menu with LAMMPS-specific help
- Line highlighting for errors

Public Functions

CodeEditor(QWidget *parent = nullptr)

Constructor.

Parameters
parent – Parent widget (typically the main window)

~CodeEditor() override

Destructor.

CodeEditor() = delete

CodeEditor(const *CodeEditor*&) = delete

CodeEditor(*CodeEditor*&&) = delete

CodeEditor &**operator=**(const *CodeEditor*&) = delete

CodeEditor &**operator=**(*CodeEditor*&&) = delete

void **lineNumberAreaPaintEvent**(QPaintEvent *event)

Paint line numbers in the line number area.

Parameters
event – Paint event to handle

int **lineNumberAreaWidth**()

Calculate width needed for line number area.

Returns
Width in pixels

void **setFont**(const QFont &newfont)

Set editor font.

Parameters
newfont – Font to use for editor text

void **setCursor**(int block)

Set cursor to specific text block.

Parameters
block – Block number (line number) to position cursor

void **setHighlight**(int block, bool error)

Highlight a specific line (used for error indication)

Parameters

- **block** – Block number to highlight
- **error** – true for error highlight, false to clear

inline void **setReformatOnReturn**(bool flag)

Enable/disable automatic reformatting on Enter key.

Parameters
flag – true to enable, false to disable

inline void **setAutoComplete**(bool flag)

Enable/disable automatic completion popup.

Parameters
flag – true to enable, false to disable

QString **reformatLine**(const QString &line)

Reformat a line with proper indentation.

Parameters
line – Line to reformat

Returns
Reformatted line

void **setCommandList**(const QStringList &words)

Set word list for LAMMPS command completion.

Parameters
words – List of command names

void **setFixList**(const QStringList &words)

Set word list for fix style completion.

Parameters
words – List of fix style names

void **setComputeList**(const QStringList &words)

Set word list for compute style completion.

Parameters
words – List of compute style names

void **setDumpList**(const QStringList &words)

Set word list for dump style completion.

Parameters
words – List of dump style names

void **setAtomList**(const QStringList &words)

Set word list for atom style completion.

Parameters
words – List of atom style names

void **setPairList**(const QStringList &words)

Set word list for pair style completion.

Parameters
words – List of pair style names

void **setBondList**(const QStringList &words)

Set word list for bond style completion.

Parameters
words – List of bond style names

void **setAngleList**(const QStringList &words)

Set word list for angle style completion.

Parameters
words – List of angle style names

void **setDihedralList**(const QStringList &words)

Set word list for dihedral style completion.

Parameters
words – List of dihedral style names

void **setImproperList**(const QStringList &words)

Set word list for improper style completion.

Parameters
words – List of improper style names

void **setKspaceList**(const QStringList &words)

Set word list for kspace style completion.

Parameters
words – List of kspace style names

void **setRegionList**(const QStringList &words)

Set word list for region style completion.

Parameters
words – List of region style names

void **setIntegrateList**(const QStringList &words)

Set word list for integration style completion.

Parameters
words – List of integration style names

void **setMinimizeList**(const QStringList &words)

Set word list for minimization style completion.

Parameters
words – List of minimization style names

void **setVariableList**(const QStringList &words)

Set word list for variable style completion.

Parameters
words – List of variable style names

void **setUnitsList**(const QStringList &words)

Set word list for units style completion.

Parameters
words – List of units style names

void **setExtraList**(const QStringList &words)

Set extra word list for completion.

Parameters
words – List of extra words

void **setGroupList**()

Update group ID list from current LAMMPS instance.

void **setVarNameList**()

Update variable name list from current LAMMPS instance.

void **setComputeIDList**()

Update compute ID list from current LAMMPS instance.

void **setFixIDList**()

Update fix ID list from current LAMMPS instance.

void **setFileList**()

Update file list from current directory.

Public Static Attributes

static int **NO_HIGHLIGHT** = 1 << 30

Constant for disabled highlighting.

Protected Functions

void **resizeEvent**(QResizeEvent *event) override

Handle resize events to update line number area.

Parameters
event – The resize event

bool **canInsertFromMimeData**(const QMimeData *source) const override

Check if MIME data can be inserted (for drag-and-drop)

Parameters
source – The MIME data to check

Returns
true if data can be inserted

void **dragEnterEvent**(QDragEnterEvent *event) override

Handle drag enter events.

Parameters
event – The drag enter event

void **dragLeaveEvent**(QDragLeaveEvent *event) override

Handle drag leave events.

Parameters
event – The drag leave event

void **dropEvent**(QDropEvent *event) override

Handle drop events.

Parameters
event – The drop event

void **contextMenuEvent**(QContextMenuEvent *event) override

Handle context menu events.

Parameters
event – The context menu event

void **keyPressEvent**(QKeyEvent *event) override

Handle key press events (for auto-completion and formatting)

Parameters
event – The key event

void **setDocver**()

Set LAMMPS documentation version for help links.

LineNumberArea Class

class **LineNumberArea** : public QWidget

Widget for displaying line numbers alongside the code editor.

This class provides a custom widget that displays line numbers in the margin of the *CodeEditor*. It delegates painting to the *CodeEditor* class.

Public Functions

inline explicit **LineNumberArea**(*CodeEditor* *editor)

Constructor.

Parameters
editor – Pointer to the associated *CodeEditor*

~LineNumberArea() override = default

Destructor.

LineNumberArea() = delete

LineNumberArea(const *LineNumberArea*&) = delete

LineNumberArea(*LineNumberArea*&&) = delete

LineNumberArea &**operator=**(const *LineNumberArea*&) = delete

LineNumberArea &**operator=**(*LineNumberArea*&&) = delete

inline QSize **sizeHint**() const override

Get the ideal size for the line number area.

Returns

QSize with width from editor, height 0 (fills available height)

Protected Functions

inline void **paintEvent**(QPaintEvent *event) override

Paint event handler - delegates to *CodeEditor*.

Parameters

event – The paint event

Highlighter Class

class **Highlighter** : public QSyntaxHighlighter

Syntax highlighter for LAMMPS input scripts.

This class provides syntax highlighting for LAMMPS input files in the *CodeEditor*. It categorizes and colors different types of LAMMPS commands, keywords, variables, numbers, strings, and comments.

Public Functions

explicit **Highlighter**(QTextDocument *parent = nullptr)

Constructor.

Parameters

parent – Parent text document to highlight

~Highlighter() override = default

Destructor.

Highlighter() = delete

Highlighter(const *Highlighter*&) = delete

Highlighter(*Highlighter*&&) = delete

Highlighter &**operator=**(const *Highlighter*&) = delete

Highlighter &**operator=**(*Highlighter*&&) = delete

Protected Functions

void **highlightBlock**(const QString &text) override

Highlight a single block (line) of text.

Parameters

text – The text to highlight

LAMMPS Interface

LammpsWrapper Class

class **LammpsWrapper**

C++ wrapper for the LAMMPS C library interface.

This class provides a C++ interface to the LAMMPS library. It wraps the C library API functions and handles dynamic loading of the LAMMPS library when built in plugin mode. All LAMMPS library function calls are routed through this wrapper class.

Public Types

enum **StyleConst**

! Constants for variable styles

Values:

enumerator **EQUAL_STYLE**

enumerator **ATOM_STYLE**

enumerator **VECTOR_STYLE**

enumerator **STRING_STYLE**

enum **ScopeConst**

! Constants for data scopes

Values:

enumerator **GLOBAL_STYLE**

enumerator **DUMMY**

enumerator **LOCAL_STYLE**

enum **TypeConst**

! Constants for data types

Values:

enumerator **SCALAR_TYPE**

enumerator **VECTOR_TYPE**

enumerator **ARRAY_TYPE**

enumerator **NUM_ROWS**

enumerator **NUM_COLS**

Public Functions

LammpsWrapper()

Constructor - initializes wrapper.

~LammpsWrapper() = default

Destructor - cleans up LAMMPS instance if open.

LammpsWrapper(const *LammpsWrapper*&) = delete

LammpsWrapper(*LammpsWrapper*&&) = delete

LammpsWrapper &**operator=**(const *LammpsWrapper*&) = delete

LammpsWrapper &**operator=**(*LammpsWrapper*&&) = delete

void **open**(int nargs, char **args)

Create a new LAMMPS instance.

Parameters

- **nargs** – Number of command-line arguments
- **args** – Command-line arguments array

void **close**()

Close the LAMMPS instance.

void **finalize**()

Finalize MPI (if used) and close LAMMPS.

void **file**(const char *fname)

Process commands from a LAMMPS input file.

Parameters

fname – Filename as C-style string

inline void **file**(const QString &fname)

Process commands from a LAMMPS input file This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

fname – Filename as Qt-style QString

inline void **file**(const std::string &fname)

Process commands from a LAMMPS input file This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters
fname – Filename as C++-style std::string

void **command**(const char *cmd)

Execute a single LAMMPS command.

Parameters
cmd – Command string as C-style string

inline void **command**(const QString &cmd)

Execute a single LAMMPS command This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters
cmd – Command string as Qt-style QString

inline void **command**(const std::string &cmd)

Execute a single LAMMPS command This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters
cmd – Command string as C++-style std::string

void **commands_string**(const char *cmd)

Execute multiple LAMMPS commands from a string.

Parameters
cmd – Commands string with newlines as C-style string

inline void **commands_string**(const QString &cmd)

Execute multiple LAMMPS commands from a string This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters
cmd – Commands string with newlines as Qt-style QString

inline void **commands_string**(const std::string &cmd)

Execute multiple LAMMPS commands from a string This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters
cmd – Commands string with newlines as C++-style std::string

void **force_timeout**()

Force a timeout condition in LAMMPS.

int **version**()

Get LAMMPS version number.

Returns
Version number as integer (YYYYMMDD format)

int **extract_setting**(const char *keyword)

Extract a global setting from LAMMPS.

Parameters
keyword – Setting name to extract

Returns
Integer value of the setting

void *`extract_global`(const char *keyword)

Extract a pointer to global data from LAMMPS.

Parameters
`keyword` – Name of global data to extract

Returns
Pointer to the data

void *`extract_pair`(const char *keyword)

Extract pair style data from LAMMPS.

Parameters
`keyword` – Name of pair data to extract

Returns
Pointer to the pair data cast to a void pointer

void *`extract_atom`(const char *keyword)

Extract atom data from LAMMPS.

Parameters
`keyword` – Name of atom data to extract

Returns
Pointer to the atom data cast to void pointer

void *`extract_compute`(const char *id, int style, int type)

Extract data from a compute from LAMMPS.

Parameters

- `id` – compute id to extract
- `style` – style of data to extract
- `type` – type of data to extract

Returns
data cast to a void pointer.

void *`extract_fix`(const char *id, int style, int type, int nrow, int ncol)

Extract data from a fix from LAMMPS.

Parameters

- `id` – fix id to extract
- `style` – style of data to extract
- `type` – type of data to extract
- `nrow` – row index (only for global)
- `ncol` – column index (only for global)

Returns
data cast to a void pointer. Must be freed for global elements

double **extract_variable**(const char *keyword)

Extract a variable value from LAMMPS.

Parameters
keyword – Variable name to extract

Returns
Value of the variable as double

int **extract_variable_datatype**(const char *keyword)

Extract style of a variable from LAMMPS.

Parameters
keyword – Variable name to extract

Returns
Value type of variable as integer

int **has_id**(const char *idtype, const char *id)

Check if a compute/fix/variable ID exists.

Parameters

- **idtype** – Type of ID (“compute”, “fix”, “variable”)
- **id** – The ID to check

Returns
1 if exists, 0 otherwise

int **id_count**(const char *idtype)

Get count of IDs of a specific type.

Parameters
idtype – Type of ID (“compute”, “fix”, “variable”, “group”)

Returns
Number of IDs of that type

int **id_name**(const char *idtype, int idx, char *buf, int buflen)

Get name of an ID by index.

Parameters

- **idtype** – Type of ID
- **idx** – Index of the ID
- **buf** – Buffer to store the name
- **buflen** – Length of buffer

Returns
0 on success, -1 on error

int **style_count**(const char *keyword)

Get count of styles of a specific type.

Parameters
keyword – Type of style (“compute”, “fix”, “pair”, etc.)

Returns
Number of available styles

int **style_name**(const char *keyword, int idx, char *buf, int buflen)

Get name of a style by index.

Parameters

- **keyword** – Type of style
- **idx** – Index of the style
- **buf** – Buffer to store the name
- **buflen** – Length of buffer

Returns

0 on success, -1 on error

int **variable_info**(int idx, char *buf, int buflen)

Get information about a variable by index.

Parameters

- **idx** – Variable index
- **buf** – Buffer to store variable info
- **buflen** – Length of buffer

Returns

Variable type code

double **get_thermo**(const char *keyword)

Get current value of a thermodynamic quantity.

Parameters

keyword – Thermo keyword

Returns

Value of the thermo quantity

void ***last_thermo**(const char *keyword, int idx)

Get a specific value from last thermo output.

Parameters

- **keyword** – Thermo keyword
- **idx** – Index for vector quantities

Returns

Pointer to the value

inline bool **is_open**() const

Check if LAMMPS instance is open.

Returns

true if LAMMPS is initialized, false otherwise

bool **is_running**()

Check if LAMMPS is currently executing a run.

Returns

true if running, false otherwise

bool **has_error**() const

Check if LAMMPS has encountered an error.

Returns
true if error occurred, false otherwise

int **get_last_error_message**(char *errorbuf, int buflen)

Get the last error message from LAMMPS.

Parameters

- **errorbuf** – Buffer to store error message
- **buflen** – Length of buffer

Returns
Error type code

bool **config_accelerator**(const char *package, const char *category, const char *setting) const

Check if an accelerator package is available.

Parameters

- **package** – Package name
- **category** – Category name
- **setting** – Setting name

Returns
true if available, false otherwise

bool **config_has_package**(const char *pkg) const

Check if a package is included in LAMMPS build.

Parameters
pkg – Package name

Returns
true if included, false otherwise

bool **config_has_curl_support**() const

Check if LAMMPS was built with CURL support.

Returns
true if CURL is available, false otherwise

bool **config_has_omp_support**() const

Check if LAMMPS was built with OpenMP support.

Returns
true if OpenMP is available, false otherwise

bool **config_has_png_support**() const

Check if LAMMPS was compiled with PNG format image support.

Returns
true if PNG image format support is available, false if not

bool **config_has_jpeg_support**() const

Check if LAMMPS was compiled with JPEG format image support.

Returns
true if JPEG image format support is available, false if not

bool **has_gpu_device**() const

Check if GPU device is available for GPU package.

Returns
true if GPU device found, false otherwise

inline bool **load_lib**(const QString &fname)

Load LAMMPS shared library (plugin mode)

Parameters
fname – Library filename (QString version)

Returns
true on success, false on failure

bool **load_lib**(const char *lammpslib)

Load LAMMPS shared library (plugin mode)

Parameters
lammpslib – Library filename (C-string version)

Returns
true on success, false on failure

bool **has_plugin**() const

Check if running in plugin mode.

Returns
true if plugin mode enabled, false if linked mode

LammpsRunner Class

class **LammpsRunner** : public QThread

Worker thread for executing LAMMPS simulations.

This class provides a separate thread for running LAMMPS simulations so that the GUI remains responsive during long-running calculations. It executes LAMMPS commands or input files in the background and emits a signal when complete.

Public Functions

inline **LammpsRunner**(QObject *parent = nullptr)

Constructor.

Parameters
parent – Parent QObject

~LammpsRunner() override = default

Destructor.

LammpsRunner() = delete

LammpsRunner(const *LammpsRunner*&) = delete

LammpsRunner(*LammpsRunner*&&) = delete

LammpsRunner &**operator=**(const *LammpsRunner*&) = delete

LammpsRunner &**operator=**(*LammpsRunner*&&) = delete

inline void **run**() override

Thread execution function - runs LAMMPS commands or input file.

This function executes in the worker thread. It processes either a string of LAMMPS commands or an input file, then signals completion.

inline void **setup_run**(*LammpsWrapper* *_lammps, const char *_input, const char *_file = nullptr)

Prepare the runner thread with LAMMPS instance and commands.

Sets up the runner with the LAMMPS instance and input. Clears any previous LAMMPS state with the "clear" command. Either input or file should be provided, not both.

Parameters

- **_lammps** – Pointer to *LammpsWrapper* instance
- **_input** – String of LAMMPS commands to execute (can be nullptr)
- **_file** – Input file path to execute (can be nullptr)

Signals

void **resultReady**()

Signal emitted when LAMMPS execution completes.

Visualization Components

ChartWindow Class

class **ChartWindow** : public QWidget

Window for displaying and managing multiple time-series charts.

ChartWindow provides a GUI for displaying and manipulating multiple charts showing time-series data from LAMMPS simulations (thermodynamic output). It supports features like data smoothing, normalization, zoom/pan, and export to various formats (PNG, CSV, YAML, DAT).

Public Functions

ChartWindow(const QString &filename, QWidget *parent = nullptr)

Constructor.

Parameters

- **filename** – Path to the log file containing the data
- **parent** – Parent widget

inline int **num_charts**() const

Get the number of charts currently displayed.

Returns

Number of charts

inline bool **has_title**(const QString &title, int index) const

Check if a chart at given index has the specified title.

Parameters

- **title** – Title to check
- **index** – Chart index

Returns
true if chart has the title, false otherwise

int **get_step**() const

Get the current simulation step number.

Returns
Current step

void **reset_charts**()

Reset all charts to initial state.

void **reset_zoom**()

Manually update chart display zoom status.

This is needed at an end of a run when the run finishes too quickly and the regular chart update has not yet triggered.

void **add_chart**(const QString &title, int index)

Add a new chart to the window.

Parameters

- **title** – Chart title (thermodynamic property name)
- **index** – Chart index

void **add_data**(int step, double data, int index)

Add a data point to a chart.

Parameters

- **step** – Simulation step number
- **data** – Data value
- **index** – Chart index

void **set_units**(const QString &_units)

Set the units displayed for thermodynamic quantities.

Parameters
_units – Units string (e.g., “real”, “metal”, “lj”)

void **set_norm**(bool norm)

Enable/disable data normalization.

Parameters
norm – true to normalize data, false otherwise

Protected Functions

void **closeEvent**(QCloseEvent *event) override

Handle window close event.

Parameters
event – Close event

bool **eventFilter**(QObject *watched, QEvent *event) override

Event filter for keyboard shortcuts.

Parameters

- **watched** – Object being watched
- **event** – Event to filter

Returns
true if event handled, false otherwise

ChartViewer Class

class **ChartViewer** : public QChartView

Individual chart viewer for displaying a single time-series.

ChartViewer extends QChartView (or wraps a QGraphsView with Qt 6.10+) to display a single thermodynamic property as a function of simulation time. It supports both raw and smoothed data display, interactive zoom/pan, and provides accessors for data export.

Public Functions

explicit **ChartViewer**(const QString &title, int index, QWidget *parent = nullptr)

Constructor.

Parameters

- **title** – Chart title (property name)
- **index** – Chart index
- **parent** – Parent widget

~ChartViewer() override

Destructor.

ChartViewer() = delete

ChartViewer(const *ChartViewer*&) = delete

ChartViewer(*ChartViewer*&&) = delete

ChartViewer &**operator=**(const *ChartViewer*&) = delete

ChartViewer &**operator=**(*ChartViewer*&&) = delete

void **add_data**(int step, double data)

Add a data point to the chart.

Parameters

- **step** – Simulation step number
- **data** – Data value for this step

QRectF **get_minmax**() const

Get the min/max bounds of the data.

Returns
Rectangle containing data bounds

inline QList<QAbstractAxis*> **get_axes**() const

Get list of chart axes.

Returns
List of axes (X and Y)

void **reset_zoom**()

Reset zoom to show all data.

void **smooth_param**(bool _do_raw, bool _do_smooth, int _window, int _order)

Set smoothing parameters.

Parameters

- **_do_raw** – Show raw data series
- **_do_smooth** – Show smoothed data series
- **_window** – Smoothing window size
- **_order** – Polynomial order for Savitzky-Golay

void **update_smooth**()

Recalculate and update smoothed data.

inline int **get_index**() const

Get chart index.

Returns
Index of this chart

inline int **get_count**() const

Get number of data points.

Returns
Number of points in series

inline QString **get_title**() const

Get chart title.

Returns
Title string

inline double **get_step**(int index) const

Get step number at given index.

Parameters
index – Data point index

Returns
Step number (X value)

inline double **get_data**(int index) const

Get data value at given index.

Parameters
index – Data point index

Returns
Data value (Y value)

void **set_tlabel**(const QString &tlabel)

Set chart title.

Parameters
tlabel – New title

void **set_ylabel**(const QString &ylabel)

Set Y-axis label.

Parameters
ylabel – New Y-axis label

inline QString **get_tlabel**() const

Get current chart title.

Returns
Chart title

inline QString **get_xlabel**() const

Get X-axis label.

Returns
X-axis label

inline QString **get_ylabel**() const

Get Y-axis label.

Returns
Y-axis label

ImageViewer Class

class **ImageViewer** : public QDialog

Dialog for viewing and manipulating LAMMPS snapshot images.

This class provides an image viewer dialog for displaying LAMMPS snapshots created by the `dump image` command. It allows interactive manipulation of visualization parameters such as zoom, rotation, atom size, coloring, and rendering options. Changes can be applied to regenerate the image using the LAMMPS library interface.

Public Functions

explicit **ImageViewer**(const QString &fileName, *LammpsWrapper* *_lammps, QWidget *parent = nullptr)

Constructor.

Parameters

- **fileName** – Path to the image file to display
- **_lammps** – Pointer to *LammpsWrapper* for regenerating images
- **parent** – Parent widget

~ImageViewer() override

Destructor.

ImageViewer() = delete

ImageViewer(const *ImageViewer*&) = delete

ImageViewer(*ImageViewer*&&) = delete

ImageViewer &**operator=**(const *ImageViewer*&) = delete

ImageViewer &**operator=**(*ImageViewer*&&) = delete

void **createImage**()

Generate image using current settings.

Constructs and executes a LAMMPS dump image command with current visualization parameters and updates the displayed image.

Protected Functions

bool **eventFilter**(QObject *watched, QEvent *event) override

Intercept Alt-keystrokes.

SlideShow Class

class **SlideShow** : public QDialog

Slideshow viewer for displaying sequences of images.

SlideShow provides a dialog for viewing and navigating through sequences of images, typically from LAMMPS dump image commands. It supports manual navigation (first/prev/next/last), automatic playback with configurable timing, looping, and zoom controls. Images can be exported as a movie file.

Public Functions

explicit **SlideShow**(const QString &fileName, QWidget *parent = nullptr)

Constructor.

Parameters

- **fileName** – Path to first image file
- **parent** – Parent widget

~SlideShow() override = default

Destructor.

SlideShow() = delete

SlideShow(const *SlideShow*&) = delete

SlideShow(*SlideShow*&&) = delete

SlideShow &**operator=**(const *SlideShow*&) = delete

SlideShow &**operator=**(*SlideShow*&&) = delete

void **add_image**(const QString &filename)

Add an image to the slideshow sequence.

Parameters

filename – Path to image file to add

void **clear**()

Clear all images from slideshow.

Dialog Components

FindAndReplace Class

class **FindAndReplace** : public QDialog

Find and Replace dialog for the code editor.

FindAndReplace provides a dialog for searching and replacing text in the *CodeEditor*. It supports case-sensitive/insensitive search, whole word matching, text wrapping, and batch replace operations. The dialog is non-modal so users can continue editing while searching.

Public Functions

explicit **FindAndReplace**(*CodeEditor* *_editor, QWidget *parent = nullptr)

Constructor.

Parameters

- **_editor** – Pointer to the *CodeEditor* to search in
- **parent** – Parent widget

~FindAndReplace() override = default

Destructor.

FindAndReplace() = delete

FindAndReplace(const *FindAndReplace*&) = delete

FindAndReplace(*FindAndReplace*&&) = delete

FindAndReplace &**operator=**(const *FindAndReplace*&) = delete

FindAndReplace &**operator=**(*FindAndReplace*&&) = delete

SetVariables Class

class **SetVariables** : public QDialog

Dialog for editing LAMMPS index-style variable definitions.

SetVariables provides a dialog for managing name-value pairs that will be used as index-style variables in LAMMPS input scripts. Users can add, delete, and edit variable definitions. The dialog supports variable substitution using ${varname} syntax.

Public Functions

explicit **SetVariables**(QList<QPair<QString, QString>> &vars, QWidget *parent = nullptr)

Constructor.

Parameters

- **vars** – Reference to list of variable name-value pairs (modified in place)
- **parent** – Parent widget

~SetVariables() override = default

Destructor.

SetVariables() = delete

SetVariables(const *SetVariables*&) = delete

SetVariables(*SetVariables*&&) = delete

SetVariables &**operator=**(const *SetVariables*&) = delete

SetVariables &**operator=**(*SetVariables*&&) = delete

Preferences Class

class **Preferences** : public QDialog

Preferences/Settings dialog for LAMMPS-GUI.

This dialog provides a tabbed interface for configuring various aspects of LAMMPS-GUI including:

- General settings (LAMMPS library path, plugins, etc.)
- Accelerator package settings
- Image viewer defaults
- Editor appearance and behavior
- Chart viewer settings

Settings are persisted using QSettings and loaded on startup.

Public Functions

explicit **Preferences**(*LammpsWrapper* *lammps, QWidget *parent = nullptr)

Constructor.

Parameters

- **lammps** – Pointer to *LammpsWrapper* for querying LAMMPS configuration
- **parent** – Parent widget

~Preferences() override

Destructor - saves settings on close.

Preferences() = delete

Preferences(const *Preferences*&) = delete

Preferences(*Preferences*&&) = delete

Preferences &**operator=**(const *Preferences*&) = delete

Preferences &**operator=**(*Preferences*&&) = delete

inline void **set_relaunch**(bool val)

Set flag indicating application needs restart.

Some settings require restarting the application to take effect.

Parameters

val – true if restart needed, false otherwise

GeneralTab Class

class **GeneralTab** : public QWidget

Preferences Tab for General LAMMPS-GUI Settings.

Public Functions

explicit **GeneralTab**(QSettings *settings, *LammpsWrapper* *lammps, QWidget *parent = nullptr)

Constructor.

Parameters

- **settings** – Pointer to QSettings for storing preferences
- **lammps** – Pointer to *LammpsWrapper* for querying LAMMPS configuration
- **parent** – Parent widget

AcceleratorTab Class

class **AcceleratorTab** : public QWidget

Preferences Tab for LAMMPS Accelerator settings.

Public Types

enum **AccelType**

Values:

enumerator **None**

enumerator **Opt**

enumerator **OpenMP**

enumerator **Intel**

enumerator **Kokkos**

enumerator **Gpu**

enum **AccelPrec**

Values:

enumerator **Double**

enumerator **Mixed**

enumerator **Single**

Public Functions

explicit **AcceleratorTab**(QSettings *settings, *LammpsWrapper* *lammps, QWidget *parent = nullptr)

Constructor.

Parameters

- **settings** – Pointer to QSettings for storing preferences
- **lammps** – Pointer to *LammpsWrapper* for querying available accelerator packages
- **parent** – Parent widget

SnapshotTab Class

class **SnapshotTab** : public QWidget

Preferences Tab for Snapshot Viewer Settings.

Public Functions

explicit **SnapshotTab**(QSettings *settings, QWidget *parent = nullptr)

Constructor.

Parameters

- **settings** – Pointer to QSettings for storing preferences
- **parent** – Parent widget

EditorTab Class

class **EditorTab** : public QWidget

Preferences Tab for LAMMPS-GUI Editor Settings.

Public Functions

explicit **EditorTab**(QSettings *settings, QWidget *parent = nullptr)

Constructor.

Parameters

- **settings** – Pointer to QSettings for storing preferences
- **parent** – Parent widget

ChartsTab Class

class **ChartsTab** : public QWidget

Preferences Tab for LAMMPS-GUI Charts Viewer Settings.

Public Functions

explicit **ChartsTab**(QSettings *settings, QWidget *parent = nullptr)

Constructor.

Parameters

- **settings** – Pointer to QSettings for storing preferences
- **parent** – Parent widget

AboutDialog Class

class **AboutDialog** : public QDialog

Custom About dialog for LAMMPS-GUI.

AboutDialog displays version information, LAMMPS configuration details, and available styles in scrollable text areas. The dialog automatically scrolls down when the content exceeds the visible area, pauses at the bottom, and then returns back to the top.

When available style information is available, the dialog allocates 2/3rd of the combined scroll area space to the configuration information text and 1/3rd to the style information text. The style information text uses the configured fixed-width font from the QSettings keys "monofamily" and "monosize" while the rest uses the application's default (variable width) font.

Public Functions

AboutDialog(const QString &version, const QString &info, const QString &details, int minwidth, QWidget *parent = nullptr)

Constructor.

Parameters

- **version** – Version information text displayed at the top
- **info** – LAMMPS configuration info displayed in a scroll area
- **details** – Style information displayed in a scroll area with fixed-width font
- **minwidth** – minimum width of dialog
- **parent** – Parent widget

~AboutDialog() override = default

AboutDialog() = delete

AboutDialog(const *AboutDialog*&) = delete

AboutDialog(*AboutDialog*&&) = delete

AboutDialog &**operator=**(const *AboutDialog*&) = delete

AboutDialog &**operator=**(*AboutDialog*&&) = delete

Protected Functions

void **showEvent**(QShowEvent *event) override

Utility Components

FileViewer Class

class **FileViewer** : public QPlainTextEdit

Read-only text viewer for displaying file contents.

FileViewer provides a simple read-only text window for viewing file contents. It's used in the context menu of the code editor to view files referenced in LAMMPS input scripts (data files, potential files, etc.). The viewer supports keyboard shortcuts for closing and stopping the simulation.

Public Functions

FileViewer(const QString &filename, const QString &title = "", QWidget *parent = nullptr)

Constructor.

Parameters

- **filename** – Path to file to display

- **title** – Window title (defaults to filename if empty)
- **parent** – Parent widget

~FileViewer() override = default

Destructor.

FileViewer() = delete

FileViewer(const *FileViewer*&) = delete

FileViewer(*FileViewer*&&) = delete

FileViewer &**operator=**(const *FileViewer*&) = delete

FileViewer &**operator=**(*FileViewer*&&) = delete

Protected Functions

bool **eventFilter**(QObject *watched, QEvent *event) override

Event filter for keyboard shortcuts.

Parameters

- **watched** – Object being watched
- **event** – Event to filter

Returns

true if event handled, false otherwise

LogWindow Class

class **LogWindow** : public QPlainTextEdit

Window for displaying LAMMPS log output with warning detection.

LogWindow provides a specialized text viewer for LAMMPS log files. It highlights warnings and errors using *FlagWarnings*, supports extraction of embedded YAML data, navigation to warnings, and opening error documentation URLs. The window shows a summary of detected warnings in the title bar.

Public Functions

LogWindow(const QString &filename, QWidget *parent = nullptr)

Constructor.

Parameters

- **filename** – Path to log file to display
- **parent** – Parent widget

~LogWindow() override

Destructor.

LogWindow() = delete

LogWindow(const *LogWindow*&) = delete

LogWindow(*LogWindow*&&) = delete

LogWindow &**operator=**(const *LogWindow*&) = delete

LogWindow &**operator=**(*LogWindow*&&) = delete

Protected Functions

void **closeEvent**(QCloseEvent *event) override

Handle window close event.

Parameters
event – Close event

void **mouseDoubleClickEvent**(QMouseEvent *event) override

Handle double-click to open URLs.

Parameters
event – Mouse event

void **contextMenuEvent**(QContextMenuEvent *event) override

Show context menu with log-specific actions.

Parameters
event – Context menu event

bool **eventFilter**(QObject *watched, QEvent *event) override

Event filter for keyboard shortcuts.

Parameters

- **watched** – Object being watched
- **event** – Event to filter

Returns
true if event handled, false otherwise

bool **check_yaml**()

Check if log contains embedded YAML data.

Returns
true if YAML data detected, false otherwise

FlagWarnings Class

class **FlagWarnings** : public QSyntaxHighlighter

Syntax highlighter for LAMMPS warning and error messages.

FlagWarnings extends QSyntaxHighlighter to detect and highlight warning and error messages in LAMMPS log output. It also detects and highlights URLs for documentation links. The class maintains a count of warnings and updates a summary label.

Public Functions

explicit **FlagWarnings**(QLabel *label = nullptr, QTextDocument *parent = nullptr)

Constructor.

Parameters

- **label** – Optional label to display warning count summary
- **parent** – Text document to apply highlighting to

~FlagWarnings() override = default

Destructor.

FlagWarnings() = delete

FlagWarnings(const *FlagWarnings*&) = delete

FlagWarnings(*FlagWarnings*&&) = delete

FlagWarnings &**operator=**(const *FlagWarnings*&) = delete

FlagWarnings &**operator=**(*FlagWarnings*&&) = delete

inline int **get_nwarnings**() const

Get the current number of warnings detected.

Returns

Number of warnings found in the document

Protected Functions

void **highlightBlock**(const QString &text) override

Highlight a single block (line) of text.

Searches for warning/error patterns and URLs, applies formatting, and updates warning count.

Parameters

text – Text to highlight

ImageInfo Class

class **ImageInfo**

Store settings for displaying graphics from a fix or compute in a LAMMPS snapshot image.

Public Functions

ImageInfo() = delete

inline **ImageInfo**(bool _enabled, const QString &_style, int _colorstyle, const std::string &_color, double _opacity, double _flag1, double _flag2)

Custom constructor

Public Members

bool **enabled**

display graphics if true

QString **style**

name of style

int **colorstyle**

color style for graphics: TYPE, ELEMENT, CONSTANT

std::string **color**

custom color of graphics objects for style == CONSTANT

double **opacity**

opacity of graphics objects for style == CONSTANT

double **flag1**

Flag #1 for graphics.

double **flag2**

Flag #2 for graphics.

RegionInfo Class

class **RegionInfo**

Store settings for displaying a region in a LAMMPS snapshot image.

Public Functions

RegionInfo() = delete

inline **RegionInfo**(bool _enabled, int _style, const std::string &_color, double _diameter, double _opacity, int _npoints)

Custom constructor

Public Members

bool **enabled**

display region if true

int **style**

style of region object: FRAME, FILLED, TRANSPARENT, or POINTS

std::string **color**

color of region display

double **diameter**

diameter value for POINTS and FRAME

double **opacity**

opacity for TRANSPARENT

int **npoints**

number of points to be used for POINTS style region display

StdCapture Class

class **StdCapture**

Capture stdout output to a string buffer.

This class provides functionality to redirect and capture standard output (stdout) into a string buffer. Used to capture output from LAMMPS library calls for display in the GUI.

Public Functions

StdCapture()

Constructor - initializes capture buffers.

StdCapture(const *StdCapture*&) = delete

StdCapture(*StdCapture*&&) = delete

StdCapture &**operator=**(const *StdCapture*&) = delete

StdCapture &**operator=**(*StdCapture*&&) = delete

virtual **~StdCapture**()

Destructor - restores stdout and frees buffers.

void **BeginCapture**()

Start capturing stdout.

Redirects stdout to an internal pipe for capture

bool **EndCapture**()

Stop capturing stdout and restore original stdout.

Returns
true if capture was active, false otherwise

std::string **GetCapture**()

Get all captured output and clear the buffer.

Returns
String containing all captured output

std::string **GetChunk**()

Get a chunk of captured output without clearing.

Returns
String containing new output since last GetChunk call

double **get_bufferuse**() const

Get the buffer usage as a fraction of max buffer size.

Returns
Value between 0.0 and 1.0 indicating buffer fullness

Qt Helper Widgets

class **QHline** : public QFrame

Horizontal line widget for visual separation in dialogs.

QHline provides a simple horizontal line widget that can be used to visually separate sections in forms and dialogs. It's essentially a styled QFrame with a horizontal line shape.

Public Functions

QHline(QWidget *parent = nullptr)

Constructor.

Parameters
parent – Parent widget

---

class **QColorCompleter** : public QCompleter

Auto-completer for color name inputs.

QColorCompleter provides auto-completion for color names in text input fields. It suggests valid color names from Qt's color name list as the user types.

Public Functions

QColorCompleter(QWidget *parent = nullptr)

Constructor.

Parameters
parent – Parent widget

---

class **QColorValidator** : public QValidator

Validator for color name inputs.

QColorValidator validates color input fields to ensure they contain valid color names or hex color codes. It can also fix up partially entered color names to valid values.

Public Functions

QColorValidator(QWidget *parent = nullptr)

Constructor.

Parameters
parent – Parent widget

void **fixup**(QString &input) const override

Attempt to fix invalid color input.

Parameters
input – String to fix (modified in place)

QValidator::State **validate**(QString &input, int &pos) const override

Validate color input string.

Parameters

- **input** – String to validate
- **pos** – Cursor position (unused)

Returns
Validation state (Invalid, Intermediate, Acceptable)

---

class **RangeSlider** : public QSlider

Custom Qt Widget implementing a variant of QSlider that has two instead of one handle.

Public Functions

RangeSlider(Qt::Orientation ot = Qt::Horizontal, QWidget *parent = nullptr)

Constructor.

Parameters

- **ot** – Slider orientation (default: Horizontal)
- **parent** – Parent widget

inline int **low**() const

Return the rangeslider's current low handle value.

inline void **setLow**(int low_limit)

Set the rangeslider's current low handle value.

inline int **high**() const

Return the rangeslider's current high handle value.

inline void **setHigh**(int high_limit)

Set the rangeslider's current high handle value.

Signals

void **sliderMoved**(int, int)

This signal is emitted when sliderDown is true and one of the two sliders moves.

Protected Functions

void **paintEvent**(QPaintEvent *ev) override

handle paint event

void **mousePressEvent**(QMouseEvent *ev) override

handle mouse press event

void **mouseMoveEvent**(QMouseEvent *ev) override

handle mouse move event

inline int **pick**(QPoint const &pt) const

Extract the relevant coordinate from a point based on slider orientation.

int **pixelPosToRangeValue**(int pos)

Convert a pixel position along the slider to a range value.

Protected Attributes

int **lowLimit**

position of rangeslider's lower handle

int **highLimit**

position of rangeslider's upper handle

QStyle::SubControl **pressed_control**

currently pressed sub-control (handle)

int **tick_interval**

interval between tick marks

QSlider::TickPosition **tick_position**

position of tick marks relative to slider

QStyle::SubControl **hover_control**

sub-control currently under the mouse cursor

int **click_offset**

offset from handle center to click position

int **active_slider**

index of the currently active slider handle

---

class **VerticalLabel** : public QWidget

Widget that displays text rotated 90 degrees counter-clockwise.

VerticalLabel renders text vertically, suitable for use as a y-axis title in chart views where the built-in axis title positioning may cause overlap with tick labels.

Public Functions

explicit **VerticalLabel**(const QString &text, QWidget *parent = nullptr)

Constructor.

Parameters

- **text** – Text to display
- **parent** – Parent widget

void **setText**(const QString &text)

Update the displayed text.

Parameters

text – New text to display

QString **text**() const

Get the displayed text.

Returns

Current text

Helper Functions

Functions

char ***mystrdup**(const std::string &text)

Duplicate a string from std::string.

Parameters

text – The string to duplicate

Returns

Pointer to newly allocated C-string (caller must free)

char *mystrdup(const char *text)

Duplicate a string from C-string.

Parameters
text – The C-string to duplicate

Returns
Pointer to newly allocated C-string (caller must free)

char *mystrdup(const QString &text)

Duplicate a string from QString.

Parameters
text – The QString to duplicate

Returns
Pointer to newly allocated C-string (caller must free)

int date_compare(const QString &one, const QString &two)

Compare two date strings in "YYYY-MM-DD" format.

Parameters

- **one** – First date string
- **two** – Second date string

Returns
-1 if one < two, 0 if equal, 1 if one > two

std::vector<std::string> split_line(const std::string &text)

Split a string into words while respecting quotes.

Parameters
text – The string to split

Returns
Vector of words extracted from the string

class QWidget *get_main_widget()

Get pointer to the main LAMMPS-GUI widget.

Returns
Pointer to the main widget (used for dialogs)

void critical(QWidget *parent, const QString &title, const QString &text1, const QString &text2 = QString())

Provide standardized critical error dialog.

Parameters

- **parent** – Pointer to parent widget
- **title** – Error dialog title
- **text1** – Error message summary
- **text2** – Detailed error message (optional)

void warning(QWidget *parent, const QString &title, const QString &text1, const QString &text2 = QString())

Provide standardized custom warning dialog.

Parameters

- **parent** – Pointer to parent widget

- **title** – Warning dialog title
- **text1** – Warning message summary
- **text2** – Detailed warning message (optional)

void **export_image**(QWidget *parent, QImage *image, const QString &title)

Save image directly or convert with ImageMagick.

Parameters

- **parent** – Pointer to parent widget
- **image** – Pointer to image class
- **title** – Warning dialog title if failed

bool **has_exe**(const QString &exe)

Check if an executable is in the system PATH.

Parameters

exe – The executable name to search for

Returns

true if executable is found in PATH, false otherwise

void **purge_directory**(const QString &dir)

Recursively delete all files in a directory.

Parameters

dir – The directory to purge

bool **is_light_theme**()

Determine if the current Qt theme is light or dark.

Returns

true if light theme, false if dark theme

void **silence_stdout**()

Silence stdout by redirecting it to the null device.

Redirects stdout to /dev/null (Unix) or NUL: (Windows) to suppress all output. Does nothing if *StdCapture* is currently active or if stdout is already silenced.

Note

Not thread-safe. Must only be called from the main thread.

void **restore_stdout**()

Restore stdout after it was silenced.

Restores the original stdout file descriptor that was saved by silence_stdout(). Does nothing if stdout is not currently silenced.

Note

Not thread-safe. Must only be called from the main thread.

bool **is_stdout_silenced**()

Check if stdout is currently silenced.

Returns
true if silence_stdout() is active, false otherwise

void **notify_capture_state**(bool active)

Notify the silence/restore system about *StdCapture* state changes.

Called by *StdCapture* to indicate whether it is actively capturing output. While capture is active, silence_stdout() becomes a no-op to avoid interfering with the capture pipe.

Parameters
active – true when *StdCapture* starts capturing, false when it stops

5.4 Development Guidelines

Code Style

The project follows these coding conventions:

- **Indentation**: 4 spaces (no tabs)
- **Line length**: Maximum 100 characters
- **Formatting**: Enforced by `.clang-format` configuration (LLVM-based)
- **Comments**: Use Doxygen-style documentation comments
- **Naming**: - Classes: CamelCase (e.g., `CodeEditor`, `ChartWindow`) - Qt overrides: camelCase (e.g., `setFont`, `eventFilter`) - Custom methods/slots: snake_case (e.g., `stop_run`, `save_as`) - Members: snake_case (e.g., `reformat_on_return`)

Documentation

Added functionality should be documented both in the User's Guide and the Programmer's Guide section of the documentation. The documentation should have suitable `.. index::` directives to populate the index with suitable keywords. The documentation is written in reStructuredText and imports documentation of the source code from Doxygen using the Breathe Sphinx plugin. The documentation should translate cleanly to HTML and PDF using the `html` or `pdf` build targets. Additionally the build targets `spelling` and `linkcheck` are available to run a spell checker on the documentation and validate external links.

All public classes and functions should have Doxygen documentation as demonstrated below:

```
/**
 * @brief Brief description
 *
 * Detailed description if needed
 *
 * @param param1 Description of parameter
 * @return Description of return value
 */
int myFunction(int param1);
```

Building

See *Installation* for detailed build instructions. For development:

```
# Debug build with Qt6
cmake -S . -B build -DCMAKE_BUILD_TYPE=Debug \\
      -DLAMMPS_GUI_USE_PLUGIN=yes -DBUILD_DOC=no
cmake --build build --parallel 2
```

Contributing

To contribute to LAMMPS-GUI:

1. Fork the repository on GitHub
2. Create a feature branch
3. Make your changes with proper documentation
4. Ensure code compiles with both Qt5 and Qt6 (if possible)
5. Test your changes thoroughly
6. Submit a pull request

All contributions must:

- Follow the existing code style
- Include Doxygen documentation for new public APIs
- Not break existing functionality
- Have GPG-signed commits

5.5 Testing

The `test` directory contains some tests for the LAMMPS-GUI project using either the GoogleTest framework or the Python unittest framework

Overview

The test suite uses CMake's CTest front end to select and run the tests. Tests implemented with GoogleTest are automatically discovered and can be run individually or as a complete suite for each test program. Test running LAMMPS-GUI itself use the "virtual frame buffer" X server `Xvfb` and are written in Python using the `unittest` Python module and the PyAutoGUI module

Building the Tests

Tests are built as part of the main project build when using `-D ENABLE_TESTING=ON` during CMake configuration (default setting is *OFF*). Due to technical requirements, testing is currently only enabled for native Linux builds of LAMMPS-GUI. In any other build environments the `-D ENABLE_TESTING=ON` setting is ignored.

Quick Build

For running the tests, it is not necessary to build the documentation, so its build can be skipped during configuration.

```
cmake -S . -B build -D LAMMPS_GUI_USE_PLUGIN=yes -D BUILD_DOC=no -D ENABLE_TESTING=ON
cmake --build build --parallel 2
```

Disable Tests

Tests are disabled by default. If they have been enabled during CMake configuration they can be disabled at a later point with:

```
cmake -S . -B build -D ENABLE_TESTING=OFF
```

Running Tests

Below are some frequently used command line examples for running tests. These examples assume that LAMMPS-GUI was compiled in the folder `build`

Run All Tests

```
ctest --test-dir build/test
```

Run Tests with Verbose Output

```
ctest --test-dir build/test -V
```

List Available Tests

The list of the names of all available tests can be obtained with:

```
ctest --test-dir build/test -N
```

Run Specific Tests

Individual tests can be selected in different ways. Most common is the use of regular expressions to select (`-R`) or exclude (`-E`) tests. It is also possible to select tests by a range of Test numbers (`-I`) from the -N test list output. Examples:

```
ctest --test-dir build/test -R MyStrdup
ctest --test-dir build/test -E Frame
ctest --test-dir build/test -I 20,25
```

Current Test Coverage

The test infrastructure is under active development. Currently, the test suite focuses on utility functions and command-line interface validation. Future expansion will include GUI component testing and integration tests.

Test Organization

Tests are organized into three main categories:

1. **Unit Tests**: Using GoogleTest framework to test individual functions
2. **Command-Line Tests**: Using command-line to validate basic executable behavior
3. **GUI Tests**: Tests using the Python unittest framework and PyAutoGUI to run LAMMPS-GUI inside a virtual frame buffer in a "remote controlled fashion".

Unit Tests

test_helpers.cpp

Comprehensive tests for functions in `src/helpers.h` and `src/helpers.cpp`. This module contains 28 test cases covering utility functions used throughout the application.

String Duplication (mystrdup)

Tests for the three overloaded mystrdup functions that create heap-allocated C-style strings from different input types:

- `mystrdup(const std::string&)` - From std::string
- `mystrdup(const char*)` - From C string (handles nullptr)
- `mystrdup(const QString&)` - From Qt QString

Coverage includes:

- Normal strings with content
- Empty strings
- Null pointers (C string variant)
- UTF-8 and special characters
- Long strings

Date Comparison (date_compare)

Tests for the date_compare function that compares version date strings in LAMMPS date format (e.g., "22 Jul 2025"):

- Same dates (returns 0)
- Different years (returns positive/negative)
- Different months (returns positive/negative)
- Different days (returns positive/negative)
- Full month names vs. abbreviations
- Invalid date formats
- Edge cases (year boundaries, month boundaries)

Line Splitting (split_line)

Tests for the split_line function that parses command-line style input with proper quote handling:

- Simple whitespace-separated tokens
- Single-quoted strings
- Double-quoted strings
- Escaped quotes within strings
- Mixed quoting styles
- Triple-nested quotes
- Multiple consecutive whitespace characters
- Empty input
- Quotes at string boundaries

Executable Detection (has_exe)
Tests for the has_exe function that checks if an executable exists in PATH:

- Common system commands (sh, ls on Unix; cmd on Windows)
- Non-existent commands
- Commands with spaces in paths
- Platform-specific behavior (conditional compilation)

Theme Detection (is_light_theme)
Tests for the is_light_theme function that determines if the current Qt theme is light or dark:

- Boolean return value validation
- Consistency across calls
- No crashes on theme query

Command-Line Tests

These tests validate the `lammps-gui` executable behavior without starting the full GUI. They run quickly and are useful for CI/CD pipelines.

CommandLine.GetVersion

Purpose: Verify version reporting consistency

This test runs:

```
lammps-gui --platform offscreen -v
```

and validates that:

- The executable launches successfully
- Version output includes "LAMMPS-GUI (QT5)" or "LAMMPS-GUI (QT6)"
- Version number matches the `PROJECT_VERSION` CMake variable
- Process exits cleanly with status 0

Environment: `OMP_NUM_THREADS=1` to ensure consistent behavior

CommandLine.HasPlugin

Purpose: Verify build configuration is reflected in help text

This test runs:

```
lammps-gui --platform offscreen -h
```

and validates that help text is consistent with CMake configuration:

- **Plugin Mode** (`LAMMPS_GUI_USE_PLUGIN=ON`): Help text includes "-p, –pluginpath <path>" option
- **Linked Mode** (`LAMMPS_GUI_USE_PLUGIN=OFF`): Help text omits plugin path option

Environment: `OMP_NUM_THREADS=1` to ensure consistent behavior

GUI Tests

These tests validate LAMMPS-GUI functionality using PyAutoGUI and Xvfb (virtual frame buffer). They run the actual GUI application in a headless X server environment, allowing automated interaction and screenshot capture.

Important Note: The argument for the screen number flag `-n` for `xvfb-run` *must* be different for each test, so that the tests may run in parallel.

Framebuffer.CreateScreenshot (test_shooter.py)

Purpose: Test the screenshot wrapper utility that abstracts different screenshooter applications

Test File: `test/test_shooter.py`

This test validates the `shooter` wrapper script that provides a unified interface to various Linux screenshot utilities (ImageMagick's `import`, `magick import`, `xfce4-screenshooter`, `gnome-screenshooter`).

The test runs:

```
xvfb-run -n 11 -s "-screen 0 1024x768x24" -w 1 python test_shooter.py
```

within a virtual frame buffer and validates:

ScreenshotChecks.testCreateImage

- The `shooter` command executes without errors
- A PNG file is created at the specified path
- The image dimensions match the virtual frame buffer size (1024x768)
- The image format is PNG
- The screenshot captures an all-black screen (expected for empty Xvfb)
- Specific pixel values at multiple locations are (0,0,0) RGB

Dependencies:

- PyAutoGUI - for screen size detection
- Pillow (PIL) - for image file validation
- One of: ImageMagick (`import` or `magick`), `xfce4-screenshooter`, or `gnome-screenshooter`

Setup/Teardown:

- `setUp()`: Removes leftover `shot.png` from previous runs
- `tearDown()`: Cleans up `shot.png` after test completion

Environment: Virtual frame buffer at 1024x768x24, `PYTHONUNBUFFERED=1`, `PYTHONDONTWRITEBYTECODE=1`, `OMP_NUM_THREADS=1`

Framebuffer.CheckSize (test_xvfbsize.py)

Purpose: Verify PyAutoGUI functionality and Xvfb screen size configuration

Test File: `test/test_xvfbsize.py`

This test validates that PyAutoGUI can properly interact with the virtual frame buffer created by Xvfb, which is essential for GUI automation tests.

The test runs:

```
xvfb-run -n 12 -s "-screen 0 1024x768x24" -w 1 python test_xvfbsize.py
```

within a virtual frame buffer and validates:

PyAutoGUIChecks.testScreenSize

- PyAutoGUI correctly detects the screen dimensions
- Screen width is 1024 pixels
- Screen height is 768 pixels

PyAutoGUIChecks.testMousePosition

- PyAutoGUI can detect the mouse cursor position
- Initial mouse position is at screen center (512, 384)
- `pyautogui.moveTo()` can move cursor to absolute positions
- `pyautogui.moveRel()` can move cursor by relative offsets
- Position queries return expected coordinates after moves

Dependencies:

- PyAutoGUI - for screen size detection and mouse control

Environment: Virtual frame buffer at 1024x768x24, `PYTHONUNBUFFERED=1`, `PYTHONDONTWRITEBYTECODE=1`, `OMP_NUM_THREADS=1`

Framebuffer.GUIEditorChecks (test_gui_edit.py)

Purpose: Test basic LAMMPS-GUI editor functionality using automated GUI interactions

Test File: `test/test_gui_edit.py`

This test validates fundamental editor operations in LAMMPS-GUI by launching the application inside a virtual frame buffer and automating user interactions with PyAutoGUI. The test uses screenshot comparison to verify visual state.

The test runs:

```
xvfb-run -n 13 -s "-screen 0 1024x768x24" -w 1 python test_gui_edit.py
```

within a virtual frame buffer and validates:

GUIEditorChecks.testExitShortcut

- LAMMPS-GUI launches and displays a white editor background
- The `Ctrl-Q` keyboard shortcut exits the application cleanly
- The process exits with status 0
- Screenshots confirm the window was displayed and then closed

GUIEditorChecks.testExitMenu

- The `Alt-F` menu shortcut opens the File menu
- The `Q` key selects the Quit entry
- The application exits cleanly with status 0
- Screenshots confirm expected visual state

GUIEditorChecks.testExitModCancelNo

- Text can be typed into the editor buffer
- Exiting with a modified buffer shows a confirmation dialog
- The `Cancel` option returns to the editor without exiting
- The `No` option exits without saving

Dependencies:

- PyAutoGUI - for keyboard and mouse automation
- Pillow (PIL) - for screenshot validation
- A supported screenshooter application

Setup/Teardown:

- `setUp()`: Launches LAMMPS-GUI, cleans up leftover test files
- `tearDown()`: Terminates the LAMMPS-GUI process

Environment: Virtual frame buffer at 1024x768x24, `PYTHONUNBUFFERED=1`, `PYTHONDONTWRITEBYTECODE=1`, `OMP_NUM_THREADS=1`, `LAMMPS_GUI=<path to executable>`

Test Fixtures and Utilities

HelpersTest Fixture
Base test fixture that creates a `QCoreApplication` instance for tests that require Qt functionality. The application is created once per test suite and reused across tests for efficiency.

Platform-Specific Testing
Tests use conditional compilation (`#ifdef _WIN32`) to adapt to platform differences in:

- Path separators
- Line endings
- Available system executables
- Default shell commands

Future Test Expansion

Planned additions to the test suite include:

GUI Component Tests

- CodeEditor text manipulation
- Syntax highlighter accuracy
- Find/replace functionality
- Auto-completion behavior

LAMMPS Integration Tests

- LammpsWrapper command execution
- Variable substitution
- Error handling
- Output capture

File I/O Tests

- File opening/saving
- Recent files management
- Auto-save functionality
- Data file inspection

Preferences Tests

- Settings persistence
- Default value initialization
- Migration between versions

Tutorial Tests

- Tutorial file generation
- Directory setup
- Resource extraction

Adding Tests

Create a New Test File

1. Create a new test file in the *test/* directory (e.g., *test_newfile.cpp*)
2. Add the test executable to *test/CMakeLists.txt*:

```
add_executable(test_newfile
  test_newfile.cpp
  ${CMAKE_SOURCE_DIR}/src/newfile.cpp
)

target_include_directories(test_newfile PRIVATE
  ${CMAKE_SOURCE_DIR}/src
)

target_link_libraries(test_newfile
  GTest::gtest_main
  Qt${QT_VERSION_MAJOR}::Widgets
)

gtest_discover_tests(test_newfile)
```

Add Tests to Existing File

Add new test cases using GoogleTest macros:

```
TEST_F(HelpersTest, NewTestName)
{
    // Arrange
    std::string input = "test data";

    // Act
    auto result = function_to_test(input);
```

(continues on next page)

(continued from previous page)

```
    // Assert
    EXPECT_EQ(result, expected_value);
}
```

Dependencies

- **GoogleTest**: Automatically fetched via CMake FetchContent (v1.17.0)
- **Qt6**: Required for Qt-dependent functions (Widgets component)
- **CTest**: Part of CMake, used for test execution

Notes

- Tests that require a Qt application context use a *HelpersTest* fixture that creates a *QCoreApplication* instance.
- Platform-specific tests (e.g., *has_exe*) use conditional compilation to test appropriate commands on different operating systems.
- The test suite is designed to be easily extended with additional test files and test cases.
- GoogleTest is fetched automatically during CMake configuration, so no manual installation is required.

CI Integration

The test suite integrates with existing CI workflows: - Tests run as part of the standard build process when *ENABLE_TESTING=ON* - CTest provides standard output for CI systems - Tests can be disabled for documentation-only builds

---

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

V

W