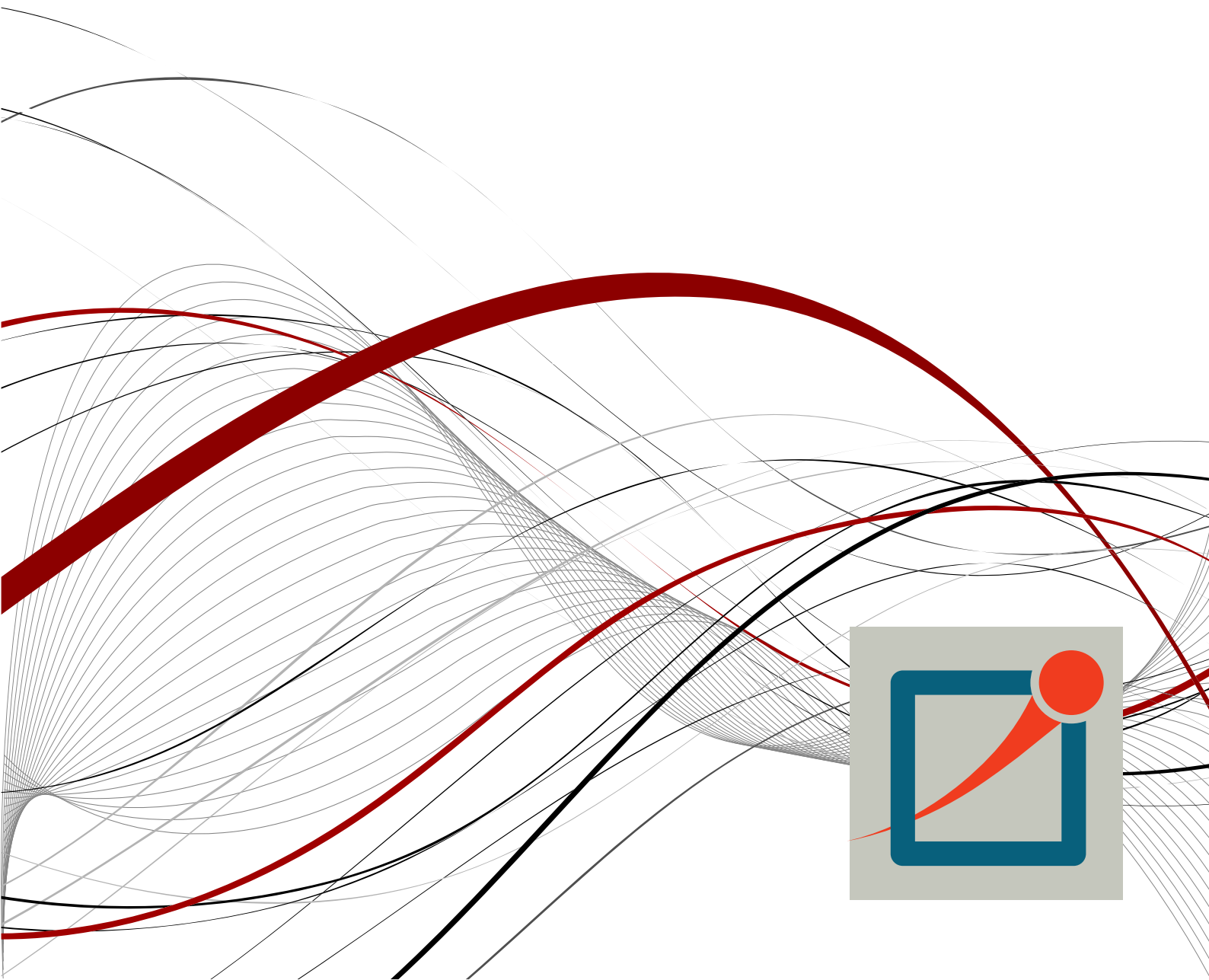
# OMNEST

## Installation Guide

Version 6.2.0



Build: 250731-59c166ccd9

# CONTENTS

CHAPTER

ONE

# QUICK INSTALLATION

## 1.1 Introduction

This document describes how to install OMNEST on various platforms. One chapter is dedicated to each operating system.

## 1.2 Supported Platforms

OMNEST (including the Simulation IDE) has been tested and is supported on the following operating systems:

- Linux x86_64/aarch64 distributions covered in this Installation Guide
- macOS 15 on x86_64/aarch64 architectures
- Windows 11 on x86_64 architecture

---

**Note:** Simulations can be run practically on any unix-like environment with a decent and fairly up-to-date C++ compiler, for example gcc 14.x. Certain OMNEST features (Qtenv, parallel simulation, XML support, etc.) depend on the availability of external libraries (Qt, MPI, LibXML, etc.)

IDE platforms are restricted because the IDE relies on a native shared library, which we compile for the above platforms and distribute in binary form for convenience.

---

## 1.3 Recommended Installation Method (opp_env)

The recommended installation method is to use `opp_env install omnetpp-latest`. `opp_env` is a package manager for OMNEST and its dependencies. You can download it from https://github.com/omnetpp/opp_env. Its main advantage is that it can automate the installation of any version of OMNEST, its dependencies and also various simulation models and tools.

`opp_env` is supported only on Linux and macOS. To install it on Windows, use the WSL (Windows Subsystem for Linux) feature of Windows 11. See further details in the chapter ch-windows-omnetpp.

## 1.4 Installing OMNEST with the System Package Manager

To install OMNEST using the system package manager, start the `install.sh` script in the root directory of the OMNEST installation. The script will detect the system package manager and will guide you through the installation process.

## 1.5 Manual Installation

To manually install OMNEST and its dependencies, visit the appropriate chapter for your platform.

CHAPTER

TWO

# WINDOWS - USING WSL (RECOMMENDED)

Windows Subsystem for Linux (WSL) supports running a full Linux distribution on a Windows machine. Running OMNEST in WSL 2 has several advantages compared to running OMNEST natively on Windows:

Advantages:

- You will probably see **significant** speedup on certain tasks (like compilation) compared to the native Windows (MinGW64) toolchain, because the compiler toolchain and the filesystem (ext4) is much faster in WSL 2 than their Windows equivalents.
- The native MinGW64 toolchain on Windows is basically a mini (Unix-like) system, emulated on top of Windows. Because of the emulation, it may have incompatibilities and limitations compared to the Linux tools. You will have fewer issues and surprises when running OMNEST on Linux.

Disadvantages:

- You will not be able to link against Windows libraries, however this is seldom needed as almost all libraries are available in the Linux environment, too.

## 2.1 Enabling or Upgrading WSL 2 on Windows

Installing OMNEST on WSL is supported on WSL 2.5.7 or later.

Open a PowerShell with Administrator privileges. On newer versions of Windows, you can install the WSL subsystem by typing:

```
wsl --install
```

Or if you have WSL already installed, just upgrade it to the latest version:

```
wsl.exe --upgrade
```

Make sure that it is 2.5.7 or later and continue to install either a Linux distribution from the Microsoft Store or `opp_env` in WSL.

---

**Tip:** We recommend installing and using the Windows Terminal application, which is available at https://www.microsoft.com/store/productId/9N0DX20HK701

---

## 2.2 Installing with opp_env.wsl (RECOMMENDED)

**opp_env.wsl** is a pre-configured Linux environment that can be easily installed on Windows and contains the `opp_env` package manager, maintained by the OMNEST team. Its main advantage is that it can automate the installation of OMNEST and its dependencies. Additionally, it can install a growing list of simulation models and tools with a single, very simple command.

Just download the `opp_env.wsl` file from https://github.com/omnetpp/opp_env/releases/download/wsl/opp_env.wsl and start it from your browser or the File Explorer. Then, follow the on-screen instructions to install OMNEST and its dependencies.

From command line you can use:

```
curl.exe -L https://github.com/omnetpp/opp_env/releases/download/wsl/opp_
↪env.wsl | wsl --import opp_env -
```

For more information, visit: https://github.com/omnetpp/opp_env.

## 2.3 Installing a Linux distribution in WSL

As a next step, you must install a Linux distribution from the Microsoft Store. We recommend using Ubuntu from https://apps.microsoft.com/detail/9pdxgncfsczv.

Once the installation is done, run the distro and finish the setup process by setting up a user name and password. At this point, you could install OMNEST.

## 2.4 Install OMNEST Linux

At this point, you have a fully functional Linux environment that can run GUI apps. You can go on and follow the Ubuntu specific installation steps to finally install OMNEST on your system.

CHAPTER

THREE

# WINDOWS - USING THE MINGW64 COMPILER TOOLCHAIN

## 3.1 Supported Windows Versions

OMNEST is supported on 64-bit versions of Windows 11.

## 3.2 Installing OMNEST

Download the OMNEST source code from https://omnetpp.org. Make sure you select the Windows-specific archive, named `omnest-6.2.0-windows-x86_64.7z`.

The package is self-contained: in addition to OMNEST files it includes a C++ compiler, a command-line build environment, and all libraries and programs required by OMNEST.

Copy the OMNEST archive to the directory where you want to install it. Choose a directory whose full path **does not contain any space**; for example, do not put OMNEST under *Program Files*.

Extract the archive file. To do so, right-click the file in Windows Explorer, and select *Extract All* from the menu.

When you look into the new `omnest-6.2.0` directory, should see directories named `doc`, `images`, `include`, `tools`, etc., and files named `opp_shell.cmd`, `configure`, `Makefile`, and others.

## 3.3 Configuring and Building OMNEST

Start `opp_shell.cmd` in the `omnest-6.2.0` directory by double-clicking it in Windows Explorer. It will bring up a console with the MSYS *bash* shell, where the path is already set to include the `omnest-6.2.0/bin` directory. On the first start of the shell, you may need to wait for the extraction of the `tools` directory.

First, check the contents of the `configure.user` file to make sure it contains the settings you need. In most cases you don't need to change anything.

```
notepad configure.user
```

Then enter the following commands:

```
$ ./configure
$ make -j16
```

The build process will create both debug and release binaries.

**Note:** If you want to install the dependencies manually instead of using the pre-packaged tools archive, delete all `*.7z` files from the `tools` directory **before** starting `opp_shell.cmd` the first time. This will prevent the extraction of the pre-packaged tools. After starting `opp_shell.cmd`, you **must** install the dependencies manually by executing the `./install.sh` script. The script will install all the dependencies and configure, then build OMNEST.

## 3.4 Verifying the Installation

You should now test all samples and check they run correctly. As an example, the *aloha* example is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the graphical Qtenv environment. You should see GUI windows and dialogs.

## 3.5 Starting the IDE

OMNEST comes with an Eclipse-based Simulation IDE. You should be able to start the IDE by typing:

```
$ omnest
```

We recommend that you start the IDE from the command-line. The build process will also create a shortcut for you if you want to use the start menu.

**Warning:** Pinning the OMNEST IDE to the taskbar will **NOT** work.

## 3.6 Environment Variables

In general OMNEST requires that certain environment variables are set. Always use the the provided shell window to start the IDE or your simulations.

## 3.7 Reconfiguring the Libraries

If you need to recompile the OMNEST components with different flags (e.g. different optimization), then change the top-level OMNEST directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make clean
$ make -j16
```

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

---

**Note:** The built libraries and programs are immediately copied to the `lib/` and `bin/` subdirs.

---

## 3.8 Portability Issues

OMNEST has been tested with both the clang compiler from the MinGW-w64 package.

Microsoft Visual C++ is not supported in the Academic Edition.

## 3.9 Additional Packages

### 3.9.1 MPI

MPI is only needed if you would like to run parallel simulations.

There are several MPI implementations for Windows, and OMNEST does not mandate any specific one. We recommend DeinoMPI, which can be downloaded from http://mpi.deino.net.

After installing DeinoMPI, adjust the `MPI_DIR` setting in OMNEST's `configure.user`, and reconfigure and recompile OMNEST:

```
$ ./configure
$ make cleanall
$ make
```

---

**Note:** In general, if you would like to run parallel simulations, we recommend that you use Linux, macOS, or another unix-like platform.

---

### 3.9.2 Akaroa

Akaroa 2.7.9, which is the latest version at the time of writing, does not support Windows. You may try to port it using the porting guide from the Akaroa distribution.

CHAPTER

FOUR

# MACOS

## 4.1 Supported Releases

This chapter provides additional information for installing OMNEST on macOS.

The following release is known to work:

- macOS 15 (and likely newer versions)

## 4.2 Installing the Prerequisite Packages

Install the command line developer tools for macOS (compiler, debugger, etc.)

```
$ xcode-select --install
```

Installing additional packages will enable more functionality in OMNEST; see the *Additional packages* section at the end of this chapter.

### 4.2.1 Using Homebrew

The `install.sh` script relies on Homebrew (https://brew.sh) for installing prerequisite packages on all modern macOS systems (both Intel and Apple-Silicon).

If you don't have Homebrew installed, follow the instructions on its website. Once Homebrew is ready, ensure its environment is set up correctly in your shell. Typically, this involves adding a line to your shell profile (e.g., `.zprofile` or `.bash_profile`):

```
eval "$(/opt/homebrew/bin/brew shellenv)"
```

Restart your terminal or source your profile script for the changes to take effect.

Install the core development tools and libraries using Homebrew:

```
$ brew install bison ccache flex perl python@3 make pkg-config doxygen␣
↪graphviz
```

Next, install packages for the graphical environment (Qtenv and IDE). If you do not need GUI support, you can skip this step and later configure OMNEST with `WITH_QTENV=no` and `WITH_OSG=no`.

```
$ brew install qt@6
```

For 3D visualization support in Qtenv, install the OpenSceneGraph package. If you do not need 3D support, you can skip this step and later configure OMNEST with `WITH_OSG=no`.

```
$ brew install openscenegraph
```

After installing packages with Homebrew, set up a Python virtual environment for OMNEST. In the root directory of your OMNEST download:

```
$ python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"
$ source .venv/bin/activate
```

Then, install the required Python packages into the virtual environment:

```
$ python3 -m pip install -r python/requirements.txt
```

**Note:** Make sure you are using `python3` from Homebrew (check with `which python3`). The system Python provided by macOS should generally not be used for development with OMNEST. If you skip the GUI or 3D packages, remember to disable the corresponding features (`WITH_QTENV=no`, `WITH_OSG=no`) in `configure.user` or during the `./configure` step.

## 4.3 Enabling Development Mode in Terminal

MacOS has a strict default security policy that prevents the execution of unsigned code. This behavior often interferes with the development process so you must explicitly allow running unsigned code from a Terminal. On the *System Preferences* / *Security and Privacy* / *Privacy* tab, select *Development Tools* on the left side, unlock the panel with the lock icon on the bottom left and select the Terminal app on the right side to override the default security policy for the Terminal app.

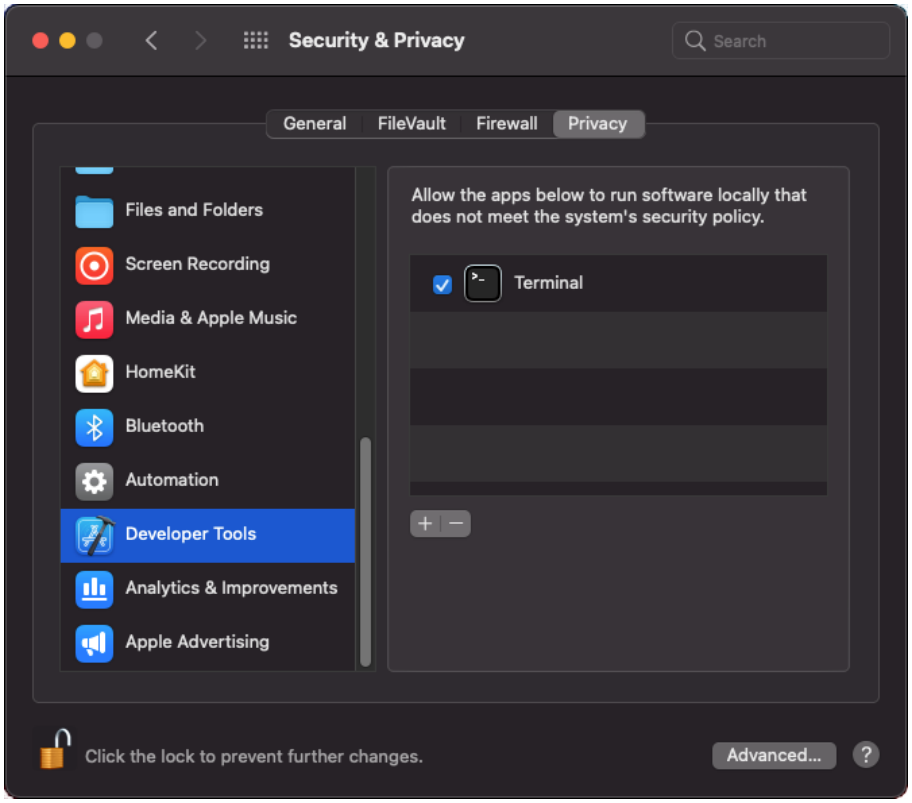



Fig. 4.1: Enable Running Unsigned Code in Terminal

**Note:** If you do not see the *Terminal* item in the *Development Tools* section, you should execute *spctl developer-mode enable-terminal* in the terminal and then restart *System Preferences* applet.

## 4.4 Debugging Unsigned Code

Even if you have enabled development mode in the terminal, missing code signatures will still cause problems during debugging, because the debugged process is started by the IDE, not the terminal. To be able to debug, you must disable code signature checking globally by typing:

```
$ sudo spctl --global-disable
```

After issuing the above command go to *System Preferences* / *Security and Privacy* / *General* and select *Any* at the bottom of the dialog. After restarting your terminal application, you will be able to debug your unsigned simulation models.

## 4.5 Additional Steps Required on macOS to Use the Debugger

The Command Line Developer Tools package contains the `lldb` debugger. If you are upgrading from an earlier version of OMNEST, be sure to delete and recreate all Launch Configurations in the IDE. This is required because older Launch Configurations were using `gdb` as the debugger, but the new IDE uses `lldb-dap` as the debugger executable.

On the first debug session the OS may prompt you to allow debugging with the `lldb` executable.

## 4.6 Downloading and Unpacking OMNEST

Download OMNEST from https://omnest.com. Make sure you select to download the macOS specific archive matching your machine's architecture, `omnest-6.2.0-macos-aarch64.tgz` (for Apple Silicon) or `omnest-6.2.0-macos-x86_64.tgz` (for Intel-based Macs).

Copy the archive to the directory where you want to install it. This is usually your home directory, `/Users/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar zxvf omnest-6.2.0-macos-aarch64.tgz
```

A subdirectory called `omnest-6.2.0` will be created, containing the simulator files.

Alternatively, you can also unpack the archive using Finder.

**Note:** The Terminal can be found in the Applications / Utilities folder.

## 4.7 Environment Variables

In general OMNEST requires that certain environment variables are set and the `omnest-6.2.0/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnest-6.2.0
$ source setenv
```

To set the environment variables permanently, edit `.profile`, `.zprofile` or `.zshenv` in your home directory and add a line something like this:

```
[ -f "$HOME/omnest-6.2.0/setenv" ] && source "$HOME/omnest-6.2.0/setenv"
```

## 4.8 Configuring and Building OMNEST

Check `configure.user` to make sure it contains the settings you need. In most cases you don't need to change anything in it.

In the top-level OMNEST directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.

---

**Note:** If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (You may need to increase the scrollback buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

---

When `./configure` has finished, you can compile OMNEST. Type in the terminal:

```
$ make
```

---

**Tip:** To take advantage of multiple processor cores, add the `-j4` option to the `make` command line.

---

---

**Note:** The build process will not write anything outside its directory, so no special privileges are needed.

---

---

**Tip:** The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

---

## 4.9 Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the aloha simulation is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the Qtenv environment. You should see nice gui windows and dialogs.

## 4.10 Starting the IDE

OMNEST comes with an Eclipse-based simulation IDE.

Start the IDE by typing:

```
$ omnest
```

If you would like to be able to launch the IDE via Applications, the Dock or a desktop shortcut, do the following: open the `omnest-6.2.0` folder in Finder, go into the `ide` subfolder, create an alias for the omnest program there (right-click, *Make Alias*), and drag the new alias into the Applications folder, onto the Dock, or onto the desktop.

Alternatively, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

which will do roughly the same.

## 4.11 Using the IDE

When you try to build a project in the IDE, you may get the following warning message:

> Toolchain "..." is not supported on this platform or installation. Please go to the Project menu, and activate a different build configuration. (You may need to switch to the C/C++ perspective first, so that the required menu items appear in the Project menu.)

If you encounter this message, choose *Project* > *Properties* > *C/C++ Build* > *Tool Chain Editor* > *Current toolchain* > *GCC for OMNEST*.

The IDE is documented in detail in the *User Guide*.

## 4.12 Reconfiguring the Libraries

If you need to recompile the OMNEST components with different flags (e.g. different optimization), then change the top-level OMNEST directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make clean
$ make
```

**Tip:** To take advantage of multiple processor cores, add the `-j4` option to the `make` command line.

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

**Note:** The built libraries and programs are immediately copied to the `lib/` and `bin/` subdirectories.

## 4.13 Additional Packages

### 4.13.1 OpenMPI

MacOS does not come with OpenMPI, so you must install it manually. You can install it from the Homebrew repo (http://brew.sh) by typing `brew install open-mpi`. In this case, you have to manually set the MPI_CFLAGS and MPI_LIBS variables in `configure.user` and re-run `./configure`.

### 4.13.2 Akaroa

Akaroa 2.7.9, which is the latest version at the time of writing, does not support macOS. You may try to port it using the porting guide from the Akaroa distribution.

#### SystemC

To enable SystemC integration, set WITH_SYSTEMC=yes in the configure.user file, run *configure* and then rebuild your project. You can check the systemc examples in the samples/systemc-embedding directory.

CHAPTER

FIVE

# LINUX

## 5.1 Supported Linux Distributions

This guide provides installation instructions for OMNEST on various Linux distributions. The `install.sh` script, included with OMNEST, automates much of this process.

The following distributions and versions are explicitly covered by the `install.sh` script and have dedicated chapters or sections in this guide:

- **Ubuntu**: 22.04 LTS, 24.04 LTS, 25.04 (and derivatives like Linux Mint)
- **Fedora**: 42 (and similar RPM-based distributions)
- **Red Hat Enterprise Linux (RHEL) / AlmaLinux**: 9.x and 10.x (and compatible distributions like Rocky Linux, CentOS Stream)
- **OpenSUSE**: Tumbleweed (rolling release)
- **Arch Linux**: (rolling release)

This chapter describes the general installation process common to these distributions. For distribution-specific details, particularly regarding the installation of prerequisite system packages, please refer to the relevant chapter:

- ch-ubuntu
- ch-fedora
- ch-redhat
- ch-opensuse
- ch-archlinux

If you are using the `install.sh` script, it will attempt to auto-detect your distribution and install the necessary system packages.

---

**Note:** If your Linux distribution is not listed above, you still may be able to use some distro-specific instructions in this Guide.

Ubuntu derivatives (Ubuntu instructions may apply):

- Kubuntu, Xubuntu, Edubuntu, . . .
- Linux Mint

Some Debian-based distros (Ubuntu instructions may apply, as Ubuntu itself is based on Debian):

- Knoppix and derivatives
- Mepis

Some Fedora-based distros (Fedora instructions may apply):

- Simplis
- Eeedora

## 5.2 Installing the Prerequisite Packages

OMNEST requires several packages to be installed on the computer. These packages include the C++ compiler (gcc or clang) and several other libraries and programs. These packages can be installed from the software repositories of your Linux distribution.

**See the chapter specific to your Linux distribution for instructions on installing the packages needed by OMNEST.**

Generally, you will need superuser permissions to install packages.

Not all packages are available from software repositories; some (optional) ones need to be downloaded separately from their web sites, and installed manually. See the section *Additional Packages* later in this chapter.

## 5.3 Downloading and Unpacking

Download OMNEST from https://omnest.com. Make sure you select to download the Linux specific archive, `omnest-6.2.0-linux-x86_64.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/home/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar xvfz omnest-6.2.0-linux-x86_64.tgz
```

This will create an `omnest-6.2.0` subdirectory with the OMNEST files in it.

---

**Note:** On how to open a terminal on your Linux installation, see the chapter specific to your Linux distribution.

---

## 5.4 Setting up the Python Virtual Environment

OMNEST uses Python for various tools and scripts. It is highly recommended to use a Python virtual environment to manage dependencies and avoid conflicts with system-wide Python packages.

The `install.sh` script automates the creation and setup of this virtual environment for most Linux distributions.

If you are installing manually or want to understand the process, the typical steps performed by the script (after system packages, including `python3` and `python3-venv`, are installed) are:

1. Navigate to the OMNEST root directory (e.g., `cd omnest-6.2.0`).
2. Create the virtual environment (this example uses `.venv` as the directory name):

   ```
   $ python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"
   ```

3. Activate the virtual environment:

```
$ source .venv/bin/activate
```

4. Install required Python packages using `pip`. It's also common to upgrade `pip` itself:

```
$ python3 -m pip install --upgrade pip
$ python3 -m pip install -r python/requirements.txt
```

Once the virtual environment is active, your shell prompt will usually change, and calls to `python` and `pip` will use the versions within the `.venv` directory.

**Note:** If you use the `install.sh` script, these steps are generally handled for you.

## 5.5 Environment Variables

In general OMNEST requires that certain environment variables are set and the *omnest-6.2.0/bin* directory is in the PATH. Source the *setenv* script to set up all these variables.

```
$ cd omnest-6.2.0
$ source setenv
```

To set the environment variables permanently, edit `.profile` or `.zprofile` in your home directory and add a line something like this:

```
[ -f "$HOME/omnest-6.2.0/setenv" ] && source "$HOME/omnest-6.2.0/setenv"
```

**Note:** The *setenv* script requires Bash or Zsh.

## 5.6 Configuring and Building OMNEST

In the top-level OMNEST directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.

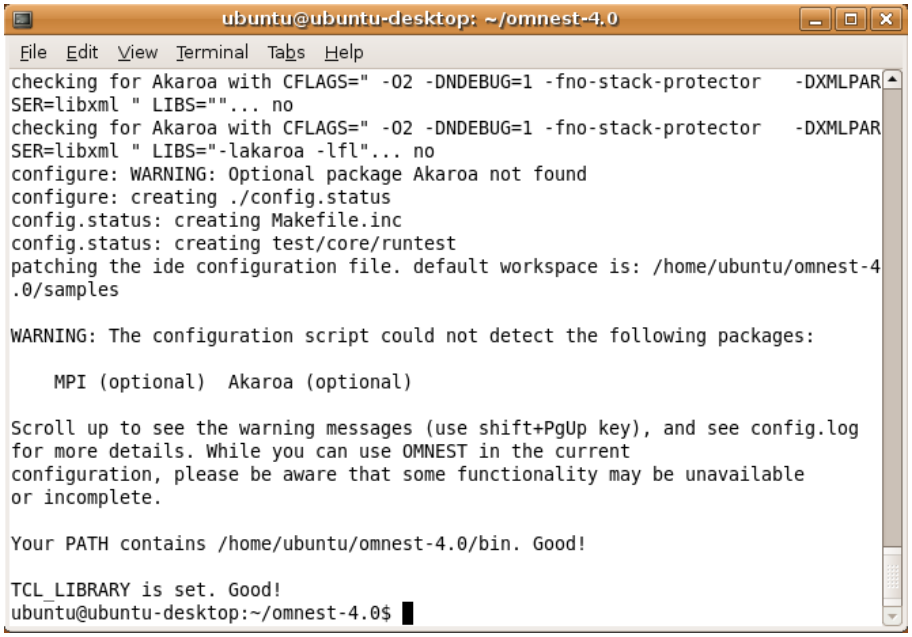



Fig. 5.1: Configuring OMNEST

**Note:** If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (Use Shift+PgUp; you may need to increase the scroll-back buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

When `./configure` has finished, you can compile OMNEST. Type in the terminal:

```
$ make
```

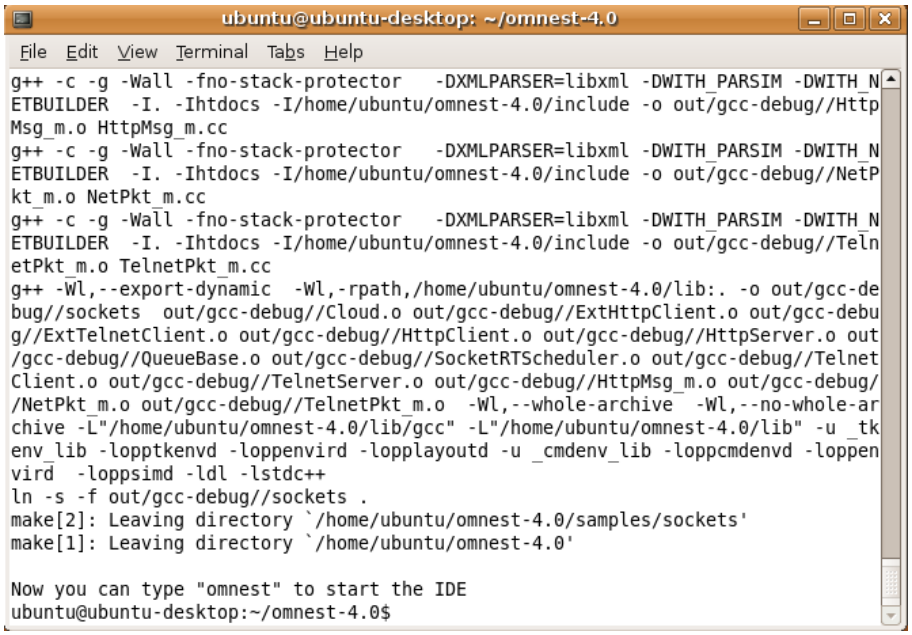



Fig. 5.2: Building OMNEST

**Tip:** To take advantage of multiple processor cores, add the `-j8` option to the `make` command line.

**Note:** The build process will not write anything outside its directory, so no special privileges are needed.

**Tip:** The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

## 5.7 Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the aloha simulation is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the Qtenv environment. You should see nice gui windows and dialogs.

## 5.8 Starting the IDE

You can launch the OMNEST Simulation IDE by typing the following command in the terminal:

```
$ omnest
```

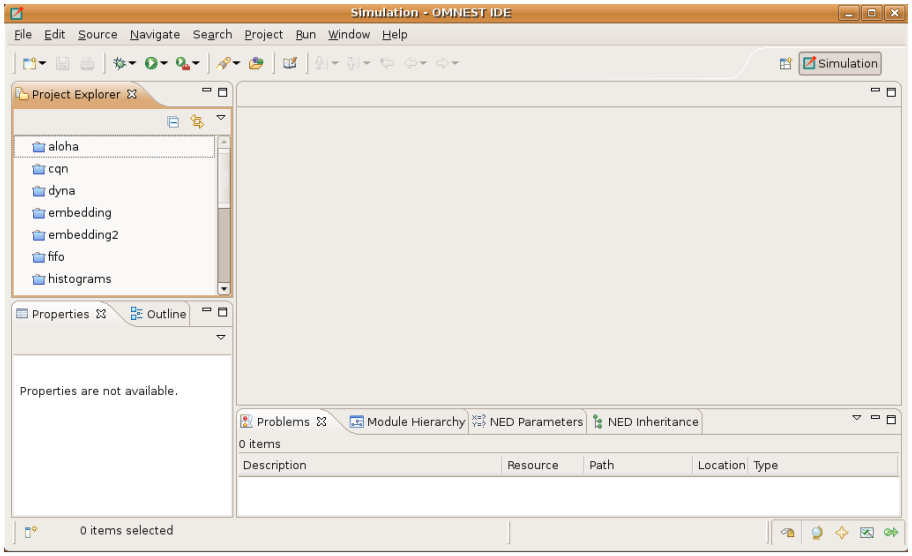



Fig. 5.3: The Simulation IDE

If you would like to be able to access the IDE from the application launcher or via a desktop shortcut, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

Or add a shortcut that points to the `omnest` program in the `ide` subdirectory by other means, for example using the Linux desktop's context menu.

## 5.9 Using the IDE

When you try to build a project in the IDE, you may get the following warning message:

> Toolchain "..." is not supported on this platform or installation. Please go to the Project menu, and activate a different build configuration. (You may need to switch to the C/C++ perspective first, so that the required menu items appear in the Project menu.)

If you encounter this message, choose *Project* > *Properties* > *C/C++ Build* > *Tool Chain Editor* > *Current toolchain* > *GCC for OMNEST*.

The IDE is documented in detail in the *User Guide*.

## 5.10 Reconfiguring the Libraries

If you need to recompile the OMNEST components with different flags (e.g. different optimization), then change the top-level OMNEST directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make cleanall
$ make
```

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

---

**Note:** For detailed description of all options please read the *Build Options* chapter.

---

## 5.11 Additional Packages

Note that at this point, MPI, Doxygen and GraphViz have been installed as part of the prerequisites.

### 5.11.1 Qtenv

OMNEST comes with a Qt based runtime environment that supports also 3D visualization. The new environment can be disabled by the WITH_QTENV=no variable in the configure.user file and then running `./configure`.

### 5.11.2 Akaroa

Linux distributions do not contain the Akaroa package. It must be downloaded, compiled and installed manually before installing OMNEST.

---

**Note:** As of version 2.7.9, Akaroa only supports Linux and Solaris.

---

Download Akaroa 2.7.9 from: http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/download.chtml

Extract it into a temporary directory:

```
$ tar xfz akaroa-2.7.9.tar.gz
```

Configure, build and install the Akaroa library. By default, it will be installed into the `/usr/local/akaroa` directory.

```
$ ./configure
$ make
$ sudo make install
```

Go to the OMNEST directory, and (re-)run the `configure` script. Akaroa will be automatically detected if you installed it to the default location.

#### SystemC

To enable SystemC integration, set WITH_SYSTEMC=yes in the configure.user file, run *configure* and then rebuild your project. You can check the systemc examples in the samples/systemc-embedding directory.

### 5.11.3 Nemiver

Nemiver is the default debugger for the OMNEST just-in-time debugging facility (see the `debugger-attach-on-startup` and `debugger-attach-on-error` configuration options). Nemiver can be installed via the package manager in most Linux distros. For example, on Ubuntu and other Debian-based distros you can install it by the following command:

```
$ sudo apt-get install nemiver
```

CHAPTER

SIX

# UBUNTU

## 6.1 Supported Releases

This chapter provides additional information for installing OMNEST on Ubuntu Linux installations. The overall installation procedure is described in the *Linux* chapter.

The following Ubuntu releases are known to work (based on the `install.sh` script):

- Ubuntu 22.04 LTS
- Ubuntu 24.04 LTS
- Ubuntu 25.04 (and likely newer versions)

The instructions below assume that you use the default desktop and the bash shell. If you use another desktop environment or shell, you may need to adjust the instructions accordingly.

## 6.2 Installing the Prerequisite Packages

Before starting the installation, it's a good practice to refresh the database of available packages. Type in the terminal:

```
$ sudo apt update
```

To install the required packages, ensure you are in the root directory of your OMNEST download. The following commands will install the necessary dependencies.

First, install the core development tools and libraries:

```
$ sudo apt install -y make diffutils pkg-config ccache clang lld gdb lldb \
    bison flex perl sed gawk python3 python3-pip python3-venv python3-dev \
    libxml2-dev zlib1g-dev doxygen graphviz xdg-utils libdw-dev
```

Next, install packages for the graphical environment (Qtenv and IDE). If you do not need GUI support (e.g., for a server installation), you can skip this step and later configure OMNEST with `WITH_QTENV=no` and `WITH_OSG=no`.

```
$ sudo apt install -y qt6-base-dev qt6-base-dev-tools qmake6 libqt6svg6 \
    qt6-wayland libwebkit2gtk-4.1-0
```

For 3D visualization support in Qtenv, install the OpenSceneGraph development package. If you do not need 3D support, you can skip this step and later configure OMNEST with `WITH_OSG=no`.

```
$ sudo apt install -y libopenscenegraph-dev
```

After installing system packages, it's good practice to clean the local repository of retrieved package files:

```
$ sudo apt clean
```

Next, set up a Python virtual environment for OMNEST. In the root directory of your OMNEST download:

```
$ python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"
$ source .venv/bin/activate
```

Then, install the required Python packages into the virtual environment:

```
$ python3 -m pip install -r python/requirements.txt
```

---

**Note:** The commands above install Clang as the C++ compiler and LLD as the linker. If you prefer to use GCC and the system's default linker, you can adjust the package list accordingly (e.g., replace `clang` with `g++` and omit `lld`) and set the `PREFER_CLANG=no` and `PREFER_LLD=no` options in the `configure.user` file or during the `./configure` step. If you skip the GUI or 3D packages, remember to disable the corresponding features (`WITH_QTENV=no`, `WITH_OSG=no`) in `configure.user` or during the `./configure` step.

---

To enable the optional parallel simulation support you will need to install the MPI packages:

```
$ sudo apt-get install mpi-default-dev
```

At the confirmation questions (*Do you want to continue? [Y/N]*), answer *Y*.

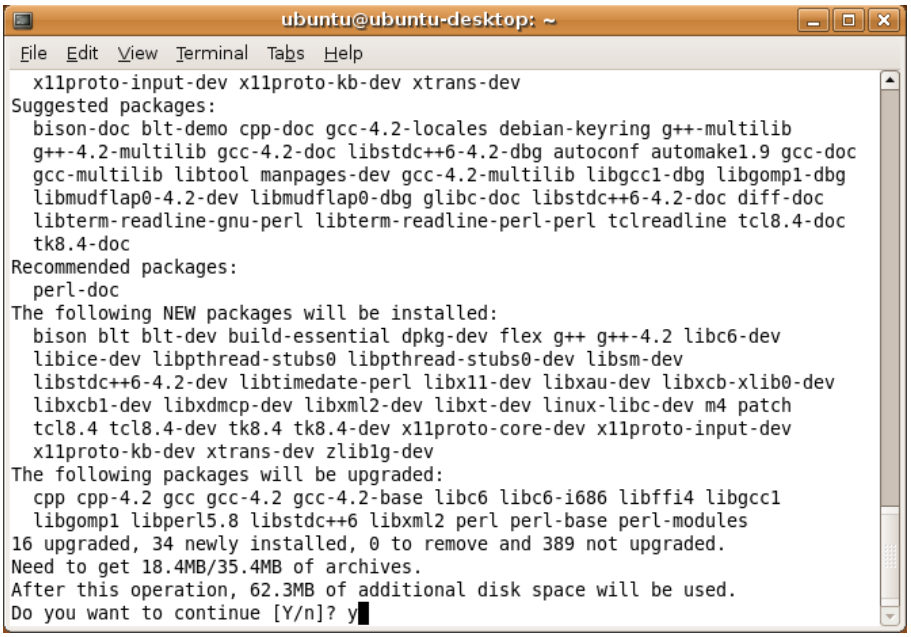



Fig. 6.1: Command-Line Package Installation

### 6.2.1 Post-Installation Steps

#### Setting Up Debugging

By default, Ubuntu does not allow ptracing of non-child processes by non-root users. That is, if you want to be able to debug simulation processes by attaching to them with a debugger, or similar, you want to be able to use OMNEST just-in-time debugging (`debugger-attach-on-startup` and `debugger-attach-on-error` configuration options), you need to explicitly enable them.

To temporarily allow ptracing non-child processes, enter the following command:

```
$ echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope
```

To permanently allow it, edit `/etc/sysctl.d/10-ptrace.conf` and change the line:

```
kernel.yama.ptrace_scope = 1
```

to read

```
kernel.yama.ptrace_scope = 0
```

CHAPTER

SEVEN

# FEDORA

## 7.1 Supported Releases

This chapter provides additional information for installing OMNEST on Fedora installations. The overall installation procedure is described in the *Linux* chapter.

The following Fedora release is known to work:

- Fedora 42 (and likely newer versions)

## 7.2 Installing the Prerequisite Packages

To install the required packages, type in the terminal.

First, install the core development tools and libraries:

```
$ sudo dnf install -y make ccache clang awk lld lldb gdb bison flex perl \
    python3-devel python3-pip libxml2-devel zlib-devel doxygen graphviz \
    xdg-utils libdwarf-devel
```

Next, install packages for the graphical environment (Qtenv and IDE). If you do not need GUI support (e.g., for a server installation), you can skip this step and later configure OMNEST with `WITH_QTENV=no` and `WITH_OSG=no`.

```
$ sudo dnf install -y qt6-qttools-devel qt6-qtbase-devel qt6-qtsvg \
    qt6-qtwayland webkit2gtk4.1
```

For 3D visualization support in Qtenv, install the OpenSceneGraph development package. If you do not need 3D support, you can skip this step and later configure OMNEST with `WITH_OSG=no`.

```
$ sudo dnf install -y OpenSceneGraph-devel
```

After installing system packages, it's good practice to clean the local repository of retrieved package files:

```
$ sudo dnf clean packages
```

Next, set up a Python virtual environment for OMNEST. In the root directory of your OMNEST download:

```
$ python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"
$ source .venv/bin/activate
```

Then, install the required Python packages into the virtual environment:

```
$ python3 -m pip install -r python/requirements.txt
```

**Note:** The commands above install Clang as the C++ compiler and LLD as the linker. If you prefer to use GCC and the system's default linker, you can adjust the package list accordingly (e.g., replace `clang` with `g++` and omit `lld`) and set the `PREFER_CLANG=no` and `PREFER_LLD=no` options in the `configure.user` file or during the `./configure` step. If you skip the GUI or 3D packages, remember to disable the corresponding features (`WITH_QTENV=no`, `WITH_OSG=no`) in `configure.user` or during the `./configure` step.

To enable the optional parallel simulation support you will need to install the MPI package:

```
$ sudo dnf install openmpi-devel
```

Note that *openmpi* will not be available by default, it needs to be activated in every session with the

```
$ module load mpi/openmpi-x86_64
```

command. When in doubt, use `module avail` to display the list of available modules. If you need MPI in every session, you may add the `module load` command to your startup script (`.bashrc`).

CHAPTER

EIGHT

# RED HAT ENTERPRISE LINUX (RHEL) AND ALMALINUX

## 8.1 Supported Releases

This chapter provides additional information for installing OMNEST on Red Hat Enterprise Linux (RHEL) and AlmaLinux distributions.

## 8.2 Installing the Prerequisite Packages

**Note:** You will need Red Hat Enterprise Linux Desktop Workstation for OMNEST. The *Desktop Client* version does not contain development tools.

To install the required packages, type in the terminal. You may need `sudo` privileges for these commands.

First, enable the EPEL (Extra Packages for Enterprise Linux) repository, which provides additional packages:

```
$ sudo dnf install -y epel-release
```

Then, install the core development tools and libraries:

```
$ sudo dnf install -y make ccache clang lld lldb gdb bison flex perl \
    python3-devel python3-pip libxml2-devel zlib-devel graphviz \
    xdg-utils elfutils-devel
```

Next, install packages for the graphical environment (Qtenv and IDE). If you do not need GUI support, you can skip this step and later configure OMNEST with `WITH_QTENV=no` and `WITH_OSG=no`.

```
$ sudo dnf install -y qt6-qttools-devel qt6-qtbase-devel qt6-qtsvg qt6-
↪qtwayland
```

**Warning: 3D Visualization (OpenSceneGraph) Support**

OpenSceneGraph is generally **not available or easily installable** on RHEL/AlmaLinux distributions from standard repositories. Therefore, it is strongly recommended to build OMNEST **without** 3D support on these systems.

You should configure OMNEST with the `WITH_OSG=no` option. If you are using the `install.sh` script, it will prompt you or you can use the `--no-3d` flag.

Next, set up a Python virtual environment for OMNEST. In the root directory of your OMNEST download:

```
$ python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"
$ source .venv/bin/activate
```

Then, install the required Python packages into the virtual environment:

```
$ python3 -m pip install -r python/requirements.txt
```

**Note:** The commands above install Clang as the C++ compiler and LLD as the linker. If you prefer to use GCC and the system's default linker, you can adjust the package list accordingly (e.g., replace `clang` with `g++` and omit `lld`) and set the `PREFER_CLANG=no` and `PREFER_LLD=no` options in the `configure.user` file or during the `./configure` step. If you skip the GUI packages or due to the lack of OpenSceneGraph, remember to disable the corresponding features (`WITH_QTENV=no`, `WITH_OSG=no`) in `configure.user` or during the `./configure` step.

To install additional (optional) packages for parallel simulation, type:

```
$ su -c 'yum install openmpi-devel'
```

Note that *openmpi* will not be available by default, it needs to be activated in every session with the

```
$ module load openmpi_<arch>
```

command, where `<arch>` is your architecture (usually `x86_64`). When in doubt, use `module avail` to display the list of available modules. If you need MPI in every session, you may add the `module load` command to your startup script (`.bashrc`).

## 8.3 SELinux

You may need to turn off SELinux when running certain simulations. To do so, click on *System* > *Administration* > *Security Level* > *Firewall*, go to the *SELinux* tab, and choose *Disabled*.

You can verify the SELinux status by typing the `sestatus` command in a terminal.

**Note:** From OMNEST 4.1 on, makefiles that build shared libraries include the `chcon -t textrel_shlib_t lib<name>.so` command that properly sets the security context for the library. This should prevent the SELinux-related *"cannot restore segment prot after reloc: Permission denied"* error from occurring, unless you have a shared library which was built using an obsolete or hand-crafted makefile that does not contain the `chcon` command.

CHAPTER

NINE

# OPENSUSE

## 9.1 Supported Releases

This chapter provides additional information for installing OMNEST on openSUSE installations. The overall installation procedure is described in the *Linux* chapter.

The following openSUSE release is supported:

- openSUSE Leap 15.4+

It was tested on the following architectures:

- Intel 64-bit

## 9.2 Installing the Prerequisite Packages

First, install the core development tools and libraries:

```
$ sudo zypper install -y make ccache clang lld lldb gdb bison gawk flex␣
↪perl \
    python311-devel python311-pip libxml2-devel zlib-devel doxygen␣
↪graphviz \
    xdg-utils libdw-devel
```

Next, install packages for the graphical environment (Qtenv and IDE). If you do not need GUI support (e.g., for a server installation), you can skip this step and later configure OMNEST with `WITH_QTENV=no` and `WITH_OSG=no`.

```
$ sudo zypper install -y qt6-base-devel qt6-wayland libQt6Svg6␣
↪libwebkit2gtk-4_1-0
```

For 3D visualization support in Qtenv, install the OpenSceneGraph development packages. If you do not need 3D support, you can skip this step and later configure OMNEST with `WITH_OSG=no`.

```
$ sudo zypper install -y libOpenSceneGraph-devel OpenSceneGraph-plugins
```

After installing system packages, it's good practice to clean the local repository of retrieved package files:

```
$ sudo zypper clean
```

Next, set up a Python virtual environment for OMNEST. In the root directory of your OMNEST download:

```
$ python311 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"
$ source .venv/bin/activate
```

Then, install the required Python packages into the virtual environment:

```
$ python311 -m pip install -r python/requirements.txt
```

---

**Note:** The commands above install Clang as the C++ compiler and LLD as the linker. If you prefer to use GCC and the system's default linker, you can adjust the package list accordingly (e.g., replace `clang` with `g++` and omit `lld`) and set the `PREFER_CLANG=no` and `PREFER_LLD=no` options in the `configure.user` file or during the `./configure` step. If you skip the GUI or 3D packages, remember to disable the corresponding features (`WITH_QTENV=no`, `WITH_OSG=no`) in `configure.user` or during the `./configure` step.

---

To enable the optional parallel simulation support you will need to install the MPI package:

```
$ sudo zypper install openmpi-devel
```

Note that *openmpi* will not be available by default, first you need to log out and log in again, or source your `.profile` script:

```
$ . ~/.profile
```

CHAPTER

TEN

# ARCH LINUX

## 10.1 Supported Releases

This chapter provides additional information for installing OMNEST on Arch Linux. The overall installation procedure is described in the *Linux* chapter.

These instructions assume you are using the `pacman` package manager.

## 10.2 Installing the Prerequisite Packages

First, ensure your system's package database is up to date and install the core development tools and libraries:

```
$ sudo pacman -Sy --needed --noconfirm make diffutils ccache clang pkgconf␣
↪lld lldb gdb \
    bison gawk flex perl python python-pip libxml2 zlib doxygen graphviz \
    xdg-utils libdwarf
```

Next, install packages for the graphical environment (Qtenv and IDE). If you do not need GUI support (e.g., for a server installation), you can skip this step and later configure OMNEST with `WITH_QTENV=no` and `WITH_OSG=no`.

```
$ sudo pacman -Sy --needed --noconfirm qt6-base qt6-svg qt6-wayland␣
↪webkit2gtk
```

For 3D visualization support in Qtenv, install the OpenSceneGraph package. If you do not need 3D support, you can skip this step and later configure OMNEST with `WITH_OSG=no`.

```
$ sudo pacman -Sy --needed --noconfirm openscenegraph
```

After installing system packages, it's good practice to clean the package cache:

```
$ sudo pacman -Scc --noconfirm
```

**Note:** The commands above install Clang as the C++ compiler and LLD as the linker. If you prefer to use GCC and the system's default linker, you can adjust the package list accordingly (e.g., replace `clang` with `gcc` and omit `lld`) and set the `PREFER_CLANG=no` and `PREFER_LLD=no` options in the `configure.user` file or during the `./configure` step. If you skip the GUI or 3D packages, remember to disable the corresponding features (`WITH_QTENV=no`, `WITH_OSG=no`) in `configure.user` or during the `./configure` step.

To enable the optional parallel simulation support you will need to install an MPI package (e.g., OpenMPI):

```
$ sudo pacman -Sy --needed --noconfirm openmpi
```

Refer to the Arch Linux documentation for managing MPI environments if needed.

CHAPTER

ELEVEN

# GENERIC UNIX

## 11.1 Introduction

This chapter provides additional information for installing OMNEST on Unix-like operating systems not specifically covered by this Installation Guide. The list includes FreeBSD, Solaris, and Linux distributions not covered in other chapters.

---

**Note:** In addition to Windows and macOS, the Simulation IDE will only work on Linux x86/arm 64-bit platforms. Other operating systems (FreeBSD, Solaris, etc.) and architectures may still be used as simulation platforms, without the IDE.

---

## 11.2 Dependencies

The following packages are required for OMNEST to work:

> build-essential, GNU make, gcc, g++, bison (3.0+), flex, perl, python3-devel, xdg-utils
>
> These packages are needed for compiling OMNEST and simulation models, and also for certain OMNEST tools to work.

It is also recommended to install the *clang* and *lld* package as they provide faster compilation and linking.

---

**Note:** You may opt to use gcc instead of the clang compiler and/or use the system default linker instead of *lld* by setting the `PREFER_CLANG` and `PREFER_LLD` variables in the *configure.user* file. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

---

**Warning:** The IDE requires GLIBC 2.28 version or later, so you will need at least Debian 10, RedHat 8 or Ubuntu 18.10 to run the IDE.

The following packages are strongly recommended, because their absence results in severe feature loss:

**Qt 5.9 or later**
: Required by the Qtenv simulation runtime environment. You need the *devel* packages that include header files as well.

**OpenSceneGraph (3.4+) and osgEarth (2.9+)**
: These packages will enable 3D visualization in Qtenv. You need the *devel* packages that include header files as well.

The following packages are required if you want to take advantage of some advanced OMNEST features:

**LibXML2**
: LibXML2 is needed for OMNEST to be able to DTD validate an XML file. The *devel* packages (that include the header files) are needed.

**GraphViz, Doxygen**
: These packages are used by the NED documentation generation feature of the IDE. When they are missing, documentation will have less content.

**MPI**
: openmpi or some other MPI implementation is required to support parallel simulation execution.

**Akaroa**
: Implements Multiple Replications In Parallel (MRIP). Akaroa can be downloaded from the project's website.

The exact names of these packages may differ across distributions.

## 11.3 Determining Package Names

If you have a distro unrelated to the ones covered in this Installation Guide, you need to figure out what is the established way of installing packages on your system, and what are the names of the packages you need.

### 11.3.1 Qt

If your platform does not have suitable Qt packages, you may still use OMNEST to run simulations from the command line. To disable the Qtenv runtime environment, use:

```
$ ./configure WITH_QTENV=no
```

This will prevent the build system to link with Qt libraries. It is also recommended if you are installing OMNEST from a remote terminal session.

### 11.3.2 MPI

OMNEST is not sensitive to the particular MPI implementation. You may use OpenMPI, or any other standards-compliant MPI package.

## 11.4 Downloading and Unpacking

Download OMNEST from https://omnest.com. Make sure you select to download the generic archive, `omnest-6.2.0-core.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/home/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar xvfz omnest-6.2.0-core.tgz
```

This will create an `omnest-6.2.0` subdirectory with the OMNEST files in it.

## 11.5 Environment Variables

In general OMNEST requires that certain environment variables are set and the `omnest-6.2.0/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnest-6.2.0
$ source setenv
```

To set the environment variables permanently, edit `.profile` or `.zprofile` in your home directory and add a line something like this:

```
[ -f "$HOME/omnest-6.2.0/setenv" ] && source "$HOME/omnest-6.2.0/setenv"
```

**Note:** The *setenv* script requires Bash or Zsh.

## 11.6 Configuring and Building OMNEST

In the top-level OMNEST directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.

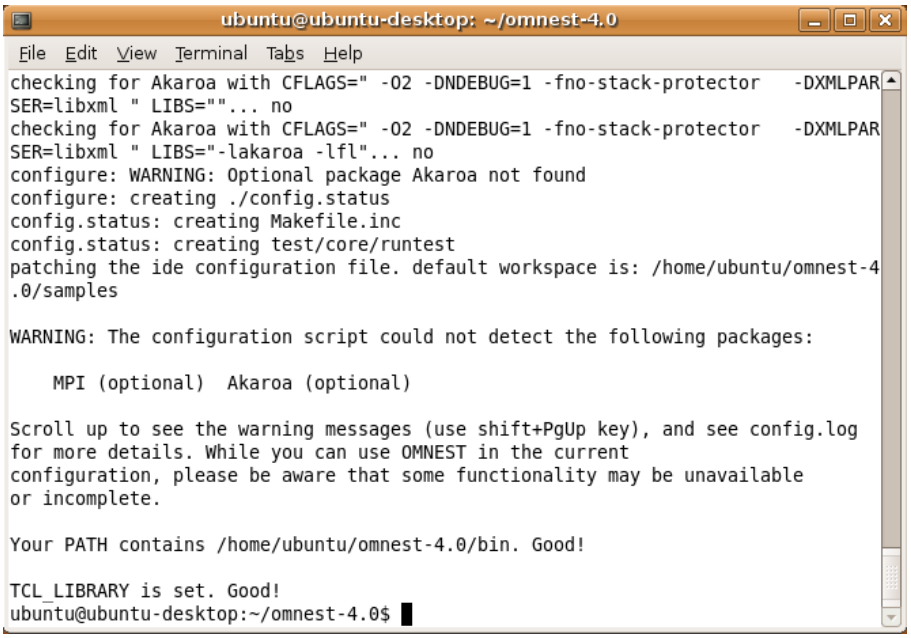



Fig. 11.1: Configuring OMNEST

**Note:** If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (Use Shift+PgUp; you may need to increase the scroll-back buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is

very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

The `configure` script tries to build and run small test programs that are using specific libraries or features of the system. You can check the `config.log` file to see which test program has failed and why. In most cases the problem is that the script cannot figure out the location of a specific library. Specifying the include file or library location in the `configure.user` file and then re-running the `configure` script usually solves the problem.

When `./configure` has finished, you can compile OMNEST. Type in the terminal:

```
$ make
```


Fig. 11.2: Building OMNEST

**Tip:** To take advantage of multiple processor cores, add the `-j8` option (for 8 cores) to the `make` command line.

**Note:** The build process will not write anything outside its directory, so no special privileges are needed.

**Tip:** The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

## 11.7 Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the aloha simulation is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the Qtenv environment. You should see nice gui windows and dialogs.

## 11.8 Starting the IDE

**Note:** The IDE is supported only on 64-bit versions of Windows, macOS and Linux.

You can run the IDE by typing the following command in the terminal:

```
$ omnest
```

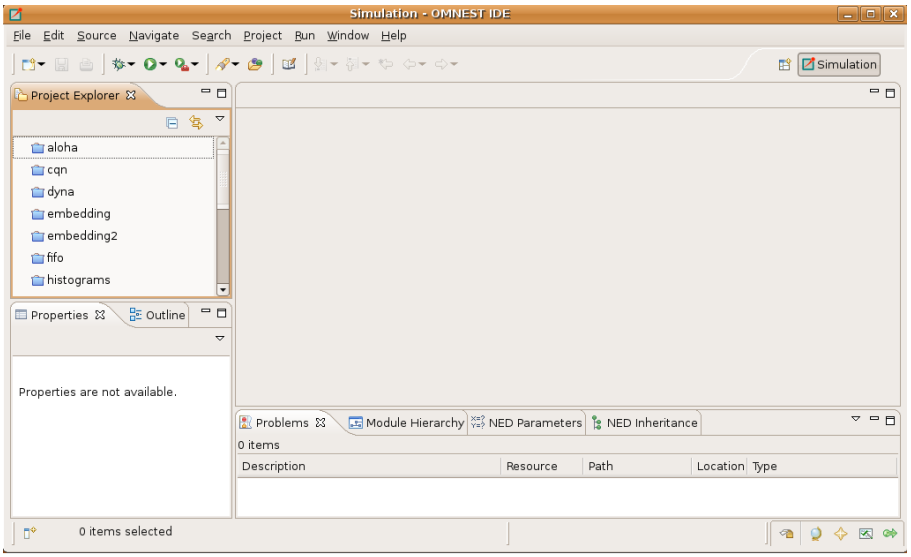



Fig. 11.3: The Simulation IDE

If you would like to be able to access the IDE from the application launcher or via a desktop shortcut, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

**Note:** The above commands assume that your system has the `xdg` commands, which most modern distributions do.

## 11.9 Optional Packages

### 11.9.1 Akaroa

If you wish to use Akaroa, it must be downloaded, compiled, and installed manually before installing OMNEST.

---

**Note:** As of version 2.7.9, Akaroa only supports Linux and Solaris.

---

Download Akaroa 2.7.9 from: http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/download.chtml

Extract it into a temporary directory:

```
$ tar xfz akaroa-2.7.9.tar.gz
```

Configure, build and install the Akaroa library. By default, it will be installed into the `/usr/local/akaroa` directory.

```
$ ./configure
$ make
$ sudo make install
```

Go to the OMNEST directory, and (re-)run the `configure` script. Akaroa will be automatically detected if you installed it to the default location.

#### SystemC

To enable SystemC integration, set WITH_SYSTEMC=yes in the configure.user file, run *configure* and then rebuild your project. You can check the systemc examples in the samples/systemc-embedding directory.

CHAPTER

TWELVE

# BUILD OPTIONS

## 12.1 Configure.user Options

The `configure.user` file contains several options that can be used to fine-tune the simulation libraries.

You always need to re-run the `configure` script in the installation root after changing the `configure.user` file.

```
$ ./configure
```

After this step, you have to remove all previous libraries and recompile OMNEST:

```
$ make cleanall
$ make
```

Options:

**PREFER_CLANG=no**
: If both `gcc` and `clang` are installed on your system, setting this variable to `no` will force the configure script to use `gcc` as C++ compiler.

**WITH_SYSTEMC=yes**
: Use this variable to enable integration whith the bundled SystemC reference implementation.

**<COMPONENTNAME>_CFLAGS, <COMPONENTNAME>_LIBS**
: The `configure.user` file contains variables for defining the compile and link options needed by various external libraries. By default, the `configure` command detects these automatically, but you may override the auto detection by specifying the values by hand. (e.g. `<COMP>_CFLAGS=-I/path/to/comp/includedir` and `<COMP>_LIBS=-L/path/to/comp/libdir -lnameoflib`.)

**WITH_PARSIM=no**
: Use this variable to explicitly disable parallel simulation support in OMNEST.

**WITH_NETBUILDER=no**
: This option allows you to leave out the NED language parser and the network builder. (This is needed only if you are building your network with C++ API calls and you do not use the built-in NED language parser at all.)

**WITH_QTENV=no**
: This will prevent the build system to link with the Qt libraries. Use this option if your platform does not have a suitable Qt package or you will run the simulation only in command line mode. (i.e. You want to run OMNEST in a remote terminal session.)

**WITH_OSG=no**
: This will prevent the build system to use OpenScreenGraph which is used for 3D visualization in Qtenv.

**WITH_OSGEARTH=no**
: This will prevent the build system to use osgEarth which is used for 2D/3D mapping and visualization in Qtenv.

**CFLAGS_[RELEASE/DEBUG]**
: To change the compiler command line options the build process is using, you should specify them in the `CFLAGS_RELEASE` and `CFLAGS_DEBUG` variables. By default, the flags required for debugging or optimization are detected automatically by the `configure` script. If you set them manually, you should specify all options you need. It is recommended to check what options are detected automatically (check the `Makefile.inc` after running `configure` and look for the `CFLAGS_[RELEASE/DEBUG]` variables.) and add/modify those options manually in the `configure.user` file.

**LDFLAGS**
: Linker command line options can be explicitly set using this variable. It is recommended to check what options are detected automatically (check the `Makefile.inc` after running `configure` and look for the `LDFLAGS` variable.) and add/modify those options manually in the `configure.user` file.

**SHARED_LIBS**
: This variable controls whether the OMNEST build process will create static or dynamic libraries. By default, the OMNEST runtime is built as a set of shared libraries. If you want to build a single executable from your simulation, specify `SHARED_LIBS=no` in `configure.user` to create static OMNEST libraries and then reconfigure (`./configure`) and recompile OMNEST (`make cleanall; make`). Once the OMNEST static libraries are correctly built, your own project have to be rebuilt, too. You will get a single, statically linked executable, which requires only the NED and INI files to run.

**Warning:** It is important to completely delete the OMNEST libraries (`make cleanall`) and then rebuild them, otherwise it cannot be guaranteed that the created simulations are linked against the correct libraries.

**Note:** The `USE_DOUBLE_SIMTIME` and `WITHOUT_CPACKET` options are no longer supported. They were introduced in OMNEST 4.0 to help porting model code from OMNEST 3.x, and having fulfilled their role, they were removed in OMNEST 5.0. If you still have old model code to port, use OMNEST 4.x.

## 12.2 Moving the Installation

When you build OMNEST on your machine, several directory names are compiled into the binaries. This makes it easier to set up OMNEST in the first place, but if you rename the installation directory or move it to another location in the file system, the built-in paths become invalid and the correct paths have to be supplied via environment variables.

The following environment variables are affected (in addition to `PATH`, which also needs to be adjusted):

**OMNETPP_IMAGE_PATH**
: This variable contains the list of directories where Qtenv looks for icons. Set it to point to the `images/` subdirectory of your OMNEST installation.

**LD_LIBRARY_PATH**
: This variable contains the list of additional directories where shared libraries are looked for. Initially, `LD_LIBRARY_PATH` is not needed because shared libraries are located via the *rpath* mechanism. When you move the installation, you need to add the `lib/` subdirectory of your OMNEST installation to `LD_LIBRARY_PATH`.

**Note:** On macOS, `DYLD_LIBRARY_PATH` is used instead of `LD_LIBRARY_PATH`. On Windows, the `PATH` variable must contain the directory where shared libraries (DLLs) are present.

## 12.3 Using Different Compilers

By default, the configure script detects the following compilers automatically in the path:

- Clang (clang, clang++)
- GNU C/C++ (gcc, g++)

If you want to use compilers other than the above ones, you should specify the compiler name in the `CC` and `CXX` variables, and re-run the configuration script.

**Note:** Different compilers may have different command line options. If you use a compiler other than the default `gcc`, you may have to revise the `CFLAGS_[RELEASE/DEBUG]` and `LDFLAGS` variables.