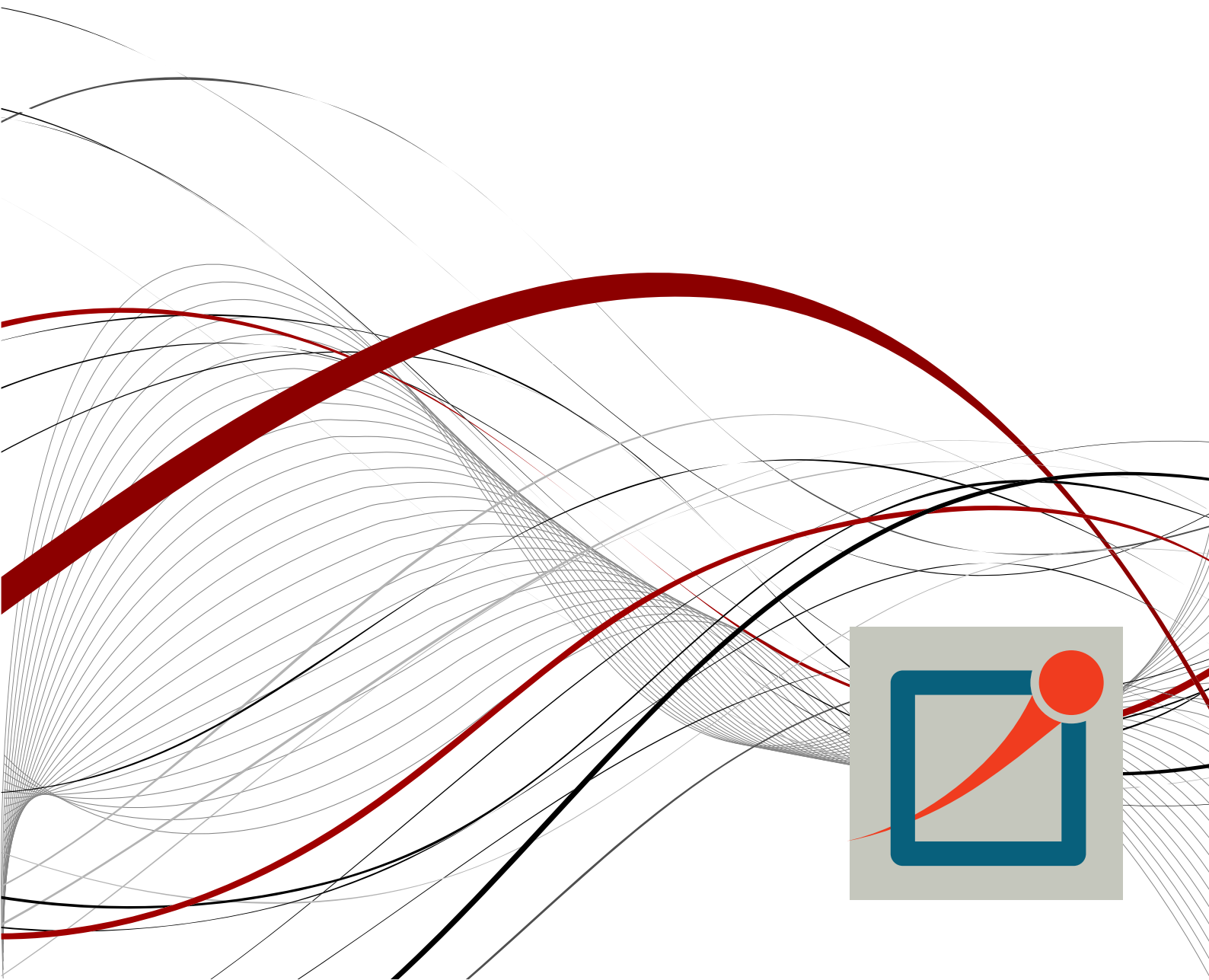
OMNeT++

Installation Guide

Version 6.5



Copyright © 1992-2024, András Varga and OpenSim Ltd
Build: a2a6c68a91

CONTENTS

1 Quick Installation **1**
- 1.1 Introduction . . . 1
- 1.2 Supported Platforms . . . 1
- 1.3 Recommended Installation Method (opp_env) . . . 1
- 1.4 Installing OMNeT++ with the System Package Manager . . . 2
- 1.5 Manual Installation . . . 2

2 Windows - Using WSL **3**
- 2.1 Enabling or Upgrading WSL 2 on Windows . . . 3
- 2.2 Installing with opp_env.wsl . . . 3
- 2.3 Installing a Linux distribution in WSL . . . 4
- 2.4 Install OMNeT++ Linux . . . 4

3 Windows - Using the MinGW64 Compiler Toolchain **5**
- 3.1 Supported Windows Versions . . . 5
- 3.2 Installing OMNeT++ . . . 5
- 3.3 Configuring and Building OMNeT++ . . . 5
- 3.4 Verifying the Installation . . . 6
- 3.5 Starting the IDE . . . 6
- 3.6 Environment Variables . . . 6
- 3.7 Reconfiguring the Libraries . . . 6
- 3.8 Portability Issues . . . 7
- 3.9 Additional Packages . . . 7

4 macOS **9**
- 4.1 Supported Releases . . . 9
- 4.2 Installing the Prerequisite Packages . . . 9
- 4.3 Enabling Development Mode in Terminal . . . 10
- 4.4 Debugging Unsigned Code . . . 11
- 4.5 Additional Steps Required on macOS to Use the Debugger . . . 11
- 4.6 Downloading and Unpacking OMNeT++ . . . 12
- 4.7 Environment Variables . . . 12
- 4.8 Configuring and Building OMNeT++ . . . 12
- 4.9 Verifying the Installation . . . 13
- 4.10 Starting the IDE . . . 13
- 4.11 Using the IDE . . . 14
- 4.12 Reconfiguring the Libraries . . . 14
- 4.13 Additional Packages . . . 15

5 Linux **17**
- 5.1 Supported Linux Distributions . . . 17
- 5.2 Installing the Prerequisite Packages . . . 18
- 5.3 Downloading and Unpacking . . . 18
- 5.4 Setting up the Python Virtual Environment . . . 18
- 5.5 Environment Variables . . . 19

5.6 Configuring and Building OMNeT++ . . . 19
5.7 Verifying the Installation . . . 21
5.8 Starting the IDE . . . 22
5.9 Using the IDE . . . 22
5.10 Reconfiguring the Libraries . . . 23
5.11 Additional Packages . . . 23

6 Ubuntu **25**
6.1 Supported Releases . . . 25
6.2 Installing the Prerequisite Packages . . . 25

7 Fedora **29**
7.1 Supported Releases . . . 29
7.2 Installing the Prerequisite Packages . . . 29

8 Red Hat Enterprise Linux (RHEL) and AlmaLinux **31**
8.1 Supported Releases . . . 31
8.2 Installing the Prerequisite Packages . . . 31
8.3 SELinux . . . 33

9 OpenSUSE **35**
9.1 Supported Releases . . . 35
9.2 Installing the Prerequisite Packages . . . 35

10 Arch Linux **37**
10.1 Supported Releases . . . 37
10.2 Installing the Prerequisite Packages . . . 37

11 NixOS **39**
11.1 Supported Releases . . . 39
11.2 Installation Prerequisites . . . 39

12 Generic Unix **41**
12.1 Introduction . . . 41
12.2 Dependencies . . . 41
12.3 Determining Package Names . . . 42
12.4 Downloading and Unpacking . . . 42
12.5 Environment Variables . . . 43
12.6 Configuring and Building OMNeT++ . . . 43
12.7 Verifying the Installation . . . 45
12.8 Starting the IDE . . . 45
12.9 Optional Packages . . . 46

13 Build Options **47**
13.1 Configure.user Options . . . 47
13.2 Moving the Installation . . . 48
13.3 Using Different Compilers . . . 49

CHAPTER

ONE

QUICK INSTALLATION

1.1 Introduction

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

1.2 Supported Platforms

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

- Linux x86_64 distributions covered in this Installation Guide
- macOS 27 on Apple Silicon (aarch64)
- Windows 11 on x86_64 architecture

Builds for Linux aarch64 and for Intel-based Macs are also provided, and macOS 15 or later is expected to work, but these were not tested for this release.

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 OMNeT++ 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 OMNeT++ 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 OMNeT++, 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 OMNeT++ with the System Package Manager

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

1.5 Manual Installation

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

CHAPTER

TWO

WINDOWS - USING WSL

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

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 OMNeT++ team. Its main

advantage is that it can automate the installation of OMNeT++ 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 OMNeT++ 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 OMNeT++.

2.4 Install OMNeT++ 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 OMNeT++ on your system.

CHAPTER

THREE

WINDOWS - USING THE MINGW64 COMPILER TOOLCHAIN

3.1 Supported Windows Versions

OMNeT++ is supported on 64-bit versions of Windows 11.

3.2 Installing OMNeT++

Download the OMNeT++ source code from https://omnetpp.org. Make sure you select the Windows-specific archive, named `omnetpp-6.5-windows-x86_64.7z`.

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

Copy the OMNeT++ 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 OMNeT++ 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 `omnetpp-6.5` 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 OMNeT++

Start `opp_shell.cmd` in the `omnetpp-6.5` 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 `omnetpp-6.5/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 OMNeT++.

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

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

```
$ omnetpp
```

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 OMNeT++ IDE to the taskbar will **NOT** work.

3.6 Environment Variables

In general OMNeT++ 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 OMNeT++ components with different flags (e.g. different optimization), then change the top-level OMNeT++ 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

OMNeT++ 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 OMNeT++ 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 OMNeT++'s `configure.user`, and reconfigure and recompile OMNeT++:

```
$ ./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 OMNeT++ on macOS.

The following release was tested:

- macOS 27 on Apple Silicon

macOS 15 or later and Intel-based Macs are expected to work, but were not tested for this release.

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 OMNeT++; 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 llvm flex perl python@3 uv 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 OMNeT++ with `WITH_QTENV=no` and `WITH_OSG=no`.

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

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 OMNeT++ with `WITH_OSG=no`.

```
$ brew install openscenegraph
```

After installing packages with Homebrew, set up a Python virtual environment for OMNeT++ using `uv`. In the root directory of your OMNeT++ download:

```
$ UV_PYTHON_DOWNLOADS=never uv venv --python python3 --seed --prompt
↪"omnetpp/.venv" .venv
$ source .venv/bin/activate
```

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

```
$ uv pip install -r python/requirements.txt
```

Note

If you prefer not to use `uv`, create the virtual environment with `python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"` and install the packages with `python3 -m pip install -r python/requirements.txt` instead.

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 OMNeT++. 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.

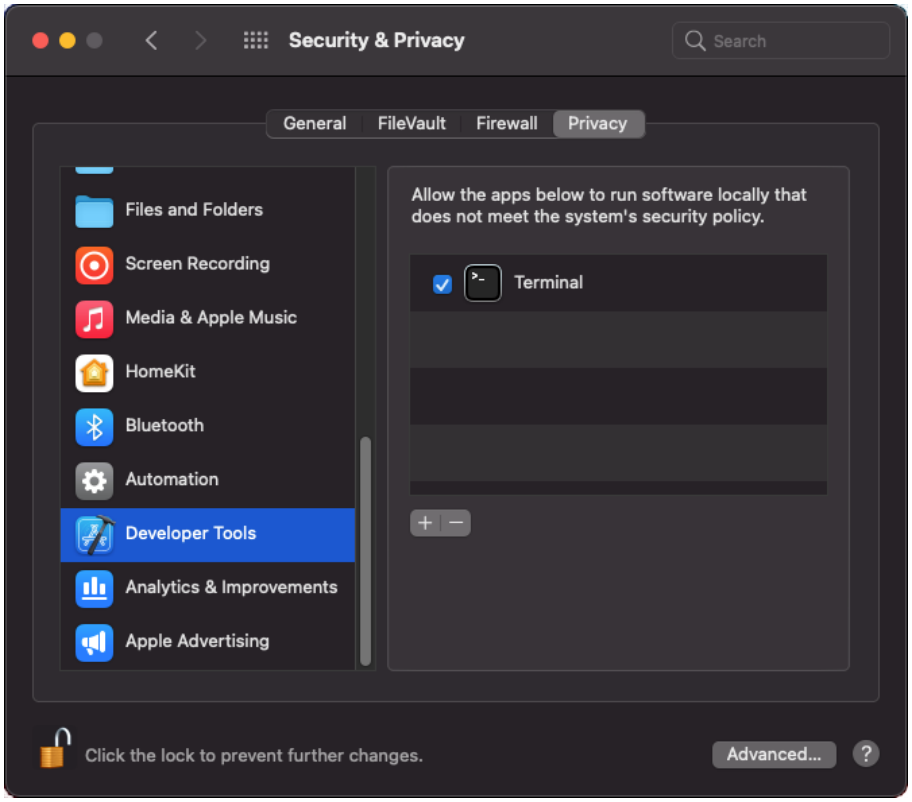


Fig. 4.1: Enable Running Unsigned Code in Terminal

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 OMNeT++, 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 OMNeT++

Download OMNeT++ from https://omnetpp.org. Make sure you select to download the macOS specific archive matching your machine's architecture, `omnetpp-6.5-macos-aarch64.tgz` (for Apple Silicon) or `omnetpp-6.5-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 omnetpp-6.5-macos-aarch64.tgz
```

A subdirectory called `omnetpp-6.5` 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 OMNeT++ requires that certain environment variables are set and the `omnetpp-6.5/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnetpp-6.5
$ 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 "$HOME/omnetpp-6.5/setenv" ] && source "$HOME/omnetpp-6.5/setenv"
```

4.8 Configuring and Building OMNeT++

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 OMNeT++ 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 OMNeT++. 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

OMNeT++ comes with an Eclipse-based simulation IDE.

Start the IDE by typing:

```
$ omnetpp
```

If you would like to be able to launch the IDE via Applications, the Dock or a desktop shortcut, do the following: open the `omnetpp-6.5` folder in Finder, go into the `ide` subfolder, create an alias for the omnetpp 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 OMNeT++*.

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

4.12 Reconfiguring the Libraries

If you need to recompile the OMNeT++ components with different flags (e.g. different optimization), then change the top-level OMNeT++ 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.

CHAPTER

FIVE

LINUX

5.1 Supported Linux Distributions

This guide provides installation instructions for OMNeT++ on various Linux distributions. The `install.sh` script, included with OMNeT++, 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**: 24.04 LTS, 26.04 LTS, 26.10 (and derivatives like Linux Mint)
- **Debian**: 13 (follow the Ubuntu chapter)
- **Fedora**: 45 (and similar RPM-based distributions)
- **Red Hat Enterprise Linux (RHEL), AlmaLinux**: 9.x and 10.x (and compatible distributions like RockyLinux, CentOS Stream)
- **OpenSUSE**: Leap 16.0, Tumbleweed (rolling release)
- **Arch Linux**: (rolling release)
- **NixOS**: (25.11 or later)

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
- ch-nixos

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

OMNeT++ 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 OMNeT++.

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 OMNeT++ from https://omnetpp.org. Make sure you select to download the Linux specific archive, `omnetpp-6.5-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 omnetpp-6.5-linux-x86_64.tgz
```

This will create an `omnetpp-6.5` subdirectory with the OMNeT++ 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

OMNeT++ 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 `uv`, are installed) are:

1. Navigate to the OMNeT++ root directory (e.g., `cd omnetpp-6.5`).
2. Create the virtual environment (this example uses `.venv` as the directory name):

```
$ UV_PYTHON_DOWNLOADS=never uv venv --python python3 --seed --prompt
↪"omnetpp/.venv" .venv
```

3. Activate the virtual environment:

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

4. Install required Python packages using `uv`:

```
$ uv pip install -r python/requirements.txt
```

`UV_PYTHON_DOWNLOADS=never` makes `uv` use the system's Python instead of downloading its own.

If `uv` is not available on your system, `install.sh` installs it for the current user with `pipx` or `pip install --user`. You can also do without it: create the virtual environment with `python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"` and install the packages with `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 OMNeT++ requires that certain environment variables are set and the *omnetpp-6.5/bin* directory is in the PATH. Source the *setenv* script to set up all these variables.

```
$ cd omnetpp-6.5
$ source setenv
```

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

```
[ -f "$HOME/omnetpp-6.5/setenv" ] && source "$HOME/omnetpp-6.5/setenv"
```

Note

The *setenv* script requires Bash or Zsh.

5.6 Configuring and Building OMNeT++

In the top-level OMNeT++ 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.

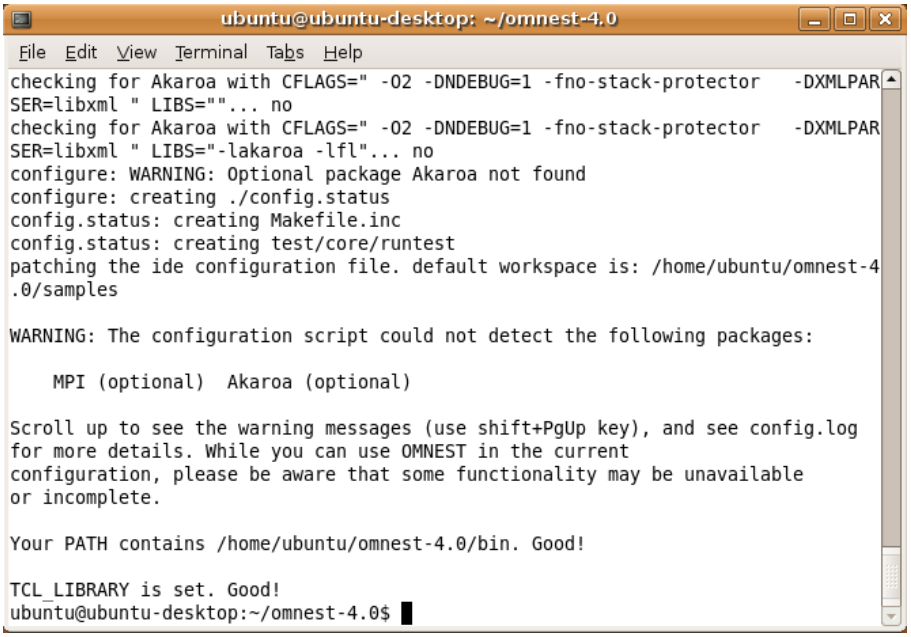


Fig. 5.1: Configuring OMNeT++

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 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 OMNeT++. Type in the terminal:

```
$ make
```



Fig. 5.2: Building OMNeT++

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 OMNeT++ Simulation IDE by typing the following command in the terminal:

```
$ omnetpp
```

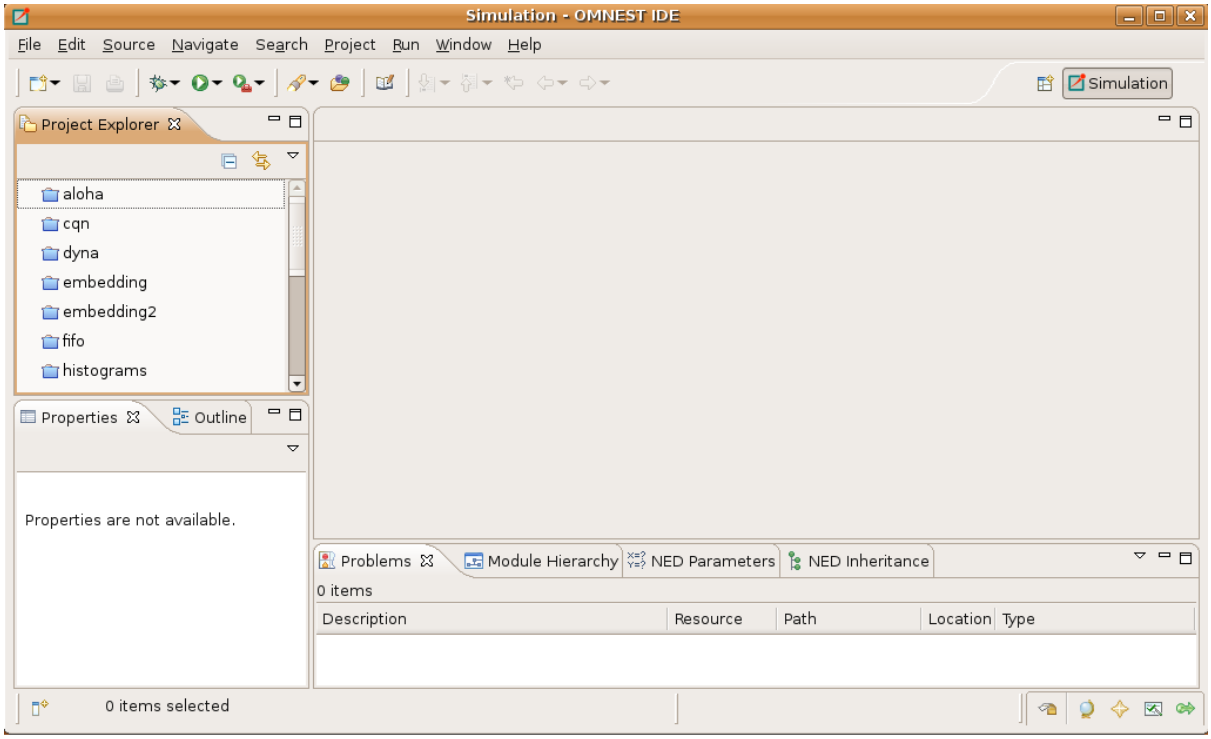


Fig. 5.3: The Simulation IDE

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 `omnetpp` 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 OMNeT++*.

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

5.10 Reconfiguring the Libraries

If you need to recompile the OMNeT++ components with different flags (e.g. different optimization), then change the top-level OMNeT++ 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, Doxygen and GraphViz have been installed as part of the prerequisites (except Doxygen on RHEL-family distributions, see their chapter). MPI is optional; see your distribution's chapter for installing it.

5.11.1 Qtenv

OMNeT++ 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 OMNeT++.

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

5.11.3 Nemiver

Nemiver is the default debugger for the OMNeT++ 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 OMNeT++ on Ubuntu and Debian installations. The overall installation procedure is described in the *Linux* chapter.

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

- Ubuntu 24.04 LTS
- Ubuntu 26.04 LTS
- Ubuntu 26.10 (and likely newer versions)
- Debian 13

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 OMNeT++ 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 OMNeT++ 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 OMNeT++ 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
```

Install `uv`, which is used to set up the Python environment. It is not packaged for Ubuntu, so install it with `pipx` (`uv` will be placed in `~/.local/bin`; `pipx ensurepath` adds it to your `PATH` in new terminals):

```
$ sudo apt install -y pipx
$ pipx install uv
$ pipx ensurepath
```

Next, set up a Python virtual environment for OMNeT++ using `uv`. In the root directory of your OMNeT++ download:

```
$ UV_PYTHON_DOWNLOADS=never uv venv --python python3 --seed --prompt
↪"omnetpp/.venv" .venv
$ source .venv/bin/activate
```

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

```
$ uv pip install -r python/requirements.txt
```

Note

If you prefer not to use `uv`, create the virtual environment with `python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"` and install the packages with `python3 -m pip install -r python/requirements.txt` instead.

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*.

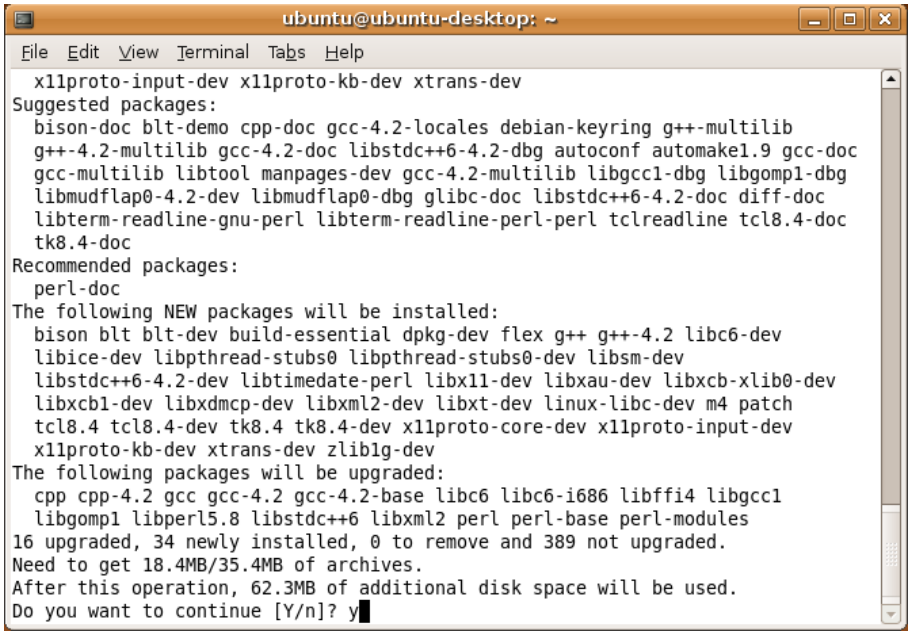


Fig. 6.1: Command-Line Package Installation

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 OMNeT++ 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 OMNeT++ on Fedora installations. The overall installation procedure is described in the *Linux* chapter.

The following Fedora release is known to work:

- Fedora 45 (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 uv libxml2-devel zlib-devel doxygen graphviz␣
↪\
    xdg-utils elfutils-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 OMNeT++ 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 OMNeT++ 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 OMNeT++ using `uv`. In the root directory of your OMNeT++ download:

```
$ UV_PYTHON_DOWNLOADS=never uv venv --python python3 --seed --prompt
↪"omnetpp/.venv" .venv
$ source .venv/bin/activate
```

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

```
$ uv pip install -r python/requirements.txt
```

Note

If you prefer not to use `uv`, create the virtual environment with `python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"` and install the packages with `python3 -m pip install -r python/requirements.txt` instead.

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 OMNeT++ on Red Hat Enterprise Linux (RHEL), RockyLinux and AlmaLinux distributions.

The following releases were tested, without 3D support:

- AlmaLinux 9.x
- AlmaLinux 10.x

RHEL, RockyLinux and CentOS Stream of the same major versions are expected to work.

8.2 Installing the Prerequisite Packages

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
```

Note

Doxygen is not in this list, and `install.sh` does not install it either: on these distributions it is available only from the CodeReady Builder (CRB) repository, which is usually not enabled. Doxygen is needed only for generating the documentation of simulation models. If you need it, enable CRB and install it: `sudo dnf config-manager --set-enabled crb && sudo dnf install -y doxygen` (on RHEL itself, enable `codeready-builder-for-rhel-<version>-<arch>-rpms` with `subscription-manager` instead).

Note

On RHEL/AlmaLinux/RockyLinux 9, the default Python 3.9 is too old. Install `python3.12-devel python3.12-pip` instead of `python3-devel python3-pip`, and use `python3.12` in place of `python3` in the commands below.

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 OMNeT++ 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 OMNeT++ **without** 3D support on these systems.

You should configure OMNeT++ with the `WITH_OSG=no` option. If you are using the `install.sh` script, it asks whether to continue without 3D support (with `-y`, it continues without asking); use the `--no-3d` flag to skip the question.

Install `uv`, which is used to set up the Python environment. It is not packaged for this distribution, so install it for the current user (`uv` will be placed in `~/.local/bin`, make sure it is on your `PATH`):

```
$ python3 -m pip install --user uv
```

Next, set up a Python virtual environment for OMNeT++ using `uv`. In the root directory of your OMNeT++ download:

```
$ UV_PYTHON_DOWNLOADS=never uv venv --python python3 --seed --prompt
↪"omnetpp/.venv" .venv
$ source .venv/bin/activate
```

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

```
$ uv pip install -r python/requirements.txt
```

Note

If you prefer not to use `uv`, create the virtual environment with `python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"` and install the packages with `python3 -m pip install -r python/requirements.txt` instead.

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:

```
$ sudo dnf install -y 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 OMNeT++ 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 OMNeT++ on openSUSE installations. The overall installation procedure is described in the *Linux* chapter.

The following openSUSE releases are supported:

- openSUSE Tumbleweed
- openSUSE Leap 16.0

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 \
    python3-devel python3-pip 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 OMNeT++ 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 OMNeT++ 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
```

Install `uv`, which is used to set up the Python environment. On Tumbleweed, install the `uv` package:

```
$ sudo zypper install -y uv
```

On Leap, install it for the current user (`uv` will be placed in `~/.local/bin`, make sure it is on your `PATH`):

```
$ python3 -m pip install --user uv
```

Next, set up a Python virtual environment for OMNeT++ using `uv`. In the root directory of your OMNeT++ download:

```
$ UV_PYTHON_DOWNLOADS=never uv venv --python python3 --seed --prompt
↪"omnetpp/.venv" .venv
$ source .venv/bin/activate
```

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

```
$ uv pip install -r python/requirements.txt
```

Note

If you prefer not to use `uv`, create the virtual environment with `python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"` and install the packages with `python3 -m pip install -r python/requirements.txt` instead.

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 OMNeT++ on Arch Linux. The overall installation procedure is described in the *Linux* chapter.

Arch Linux is a rolling release; OMNeT++ was tested with the packages current at the time of the release.

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 -Syu --needed --noconfirm make diffutils ccache clang␣
↪pkgconf lld lldb gdb \
    bison gawk flex perl python python-pip uv libxml2 zlib doxygen␣
↪graphviz \
    xdg-utils elfutils
```

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 OMNeT++ with `WITH_QTENV=no` and `WITH_OSG=no`.

```
$ sudo pacman -S --needed --noconfirm qt6-base qt6-svg qt6-wayland␣
↪webkit2gtk-4.1
```

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 OMNeT++ with `WITH_OSG=no`.

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

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

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

Next, set up a Python virtual environment for OMNeT++ using `uv`. In the root directory of your OMNeT++ download:

```
$ UV_PYTHON_DOWNLOADS=never uv venv --python python3 --seed --prompt
↪"omnetpp/.venv" .venv
$ source .venv/bin/activate
```

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

```
$ uv pip install -r python/requirements.txt
```

Note

If you prefer not to use `uv`, create the virtual environment with `python3 -m venv .venv --upgrade-deps --clear --prompt "omnetpp/.venv"` and install the packages with `python3 -m pip install -r python/requirements.txt` instead.

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 -S --needed --noconfirm openmpi
```

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

CHAPTER

ELEVEN

NIXOS

11.1 Supported Releases

This chapter provides additional information for installing OMNeT++ on NixOS. The overall installation procedure is described in the *Linux* chapter.

The following NixOS releases are known to work:

- NixOS 25.11 or later

11.2 Installation Prerequisites

On NixOS, the dependencies come from the Nix environment that omnetpp provides, so `install.sh` installs no system packages; it only sets up the Python virtual environment. omnetpp assumes that certain features are enabled in your NixOS configuration. Make sure that `nix.settings.experimental-features = [ "nix-command" "flakes" ];` is present in your NixOS configuration (`/etc/nixos/configuration.nix`).

Start the `./install.sh` script in the root directory of the omnetpp installation and follow the instructions:

```
$ ./install.sh
```

After the script has finished, you can start an omnetpp session any time by typing:

```
# set up in the current shell
$ source setenv

# or within a new shell
$ ./setenv bash
```

11.2.1 Post-Installation Steps

Setting Up Debugging

By default, NixOS 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 similarly, you want to be able to use OMNeT++ 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, add

```
boot.kernel.sysctl."kernel.yama.ptrace_scope" = 0;
```

to the `/etc/nixos/configuration.nix` file and then rebuild your configuration.

CHAPTER

TWELVE

GENERIC UNIX

12.1 Introduction

This chapter provides additional information for installing OMNeT++ 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.

12.2 Dependencies

The following packages are required for OMNeT++ to work:

build-essential, GNU make, gcc, g++, bison (3.0+), flex, perl, python3-devel, xdg-utils

These packages are needed for compiling OMNeT++ and simulation models, and also for certain OMNeT++ 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 OMNeT++ features:

LibXML2
: LibXML2 is needed for OMNeT++ 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.

12.3 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.

12.3.1 Qt

If your platform does not have suitable Qt packages, you may still use OMNeT++ 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 OMNeT++ from a remote terminal session.

12.3.2 MPI

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

12.4 Downloading and Unpacking

Download OMNeT++ from https://omnetpp.org. Make sure you select to download the generic archive, `omnetpp-6.5-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 omnetpp-6.5-core.tgz
```

This will create an `omnetpp-6.5` subdirectory with the OMNeT++ files in it.

12.5 Environment Variables

In general OMNeT++ requires that certain environment variables are set and the `omnetpp-6.5/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnetpp-6.5
$ source setenv
```

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

```
[ -f "$HOME/omnetpp-6.5/setenv" ] && source "$HOME/omnetpp-6.5/setenv"
```

Note

The *setenv* script requires Bash or Zsh.

12.6 Configuring and Building OMNeT++

In the top-level OMNeT++ 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.

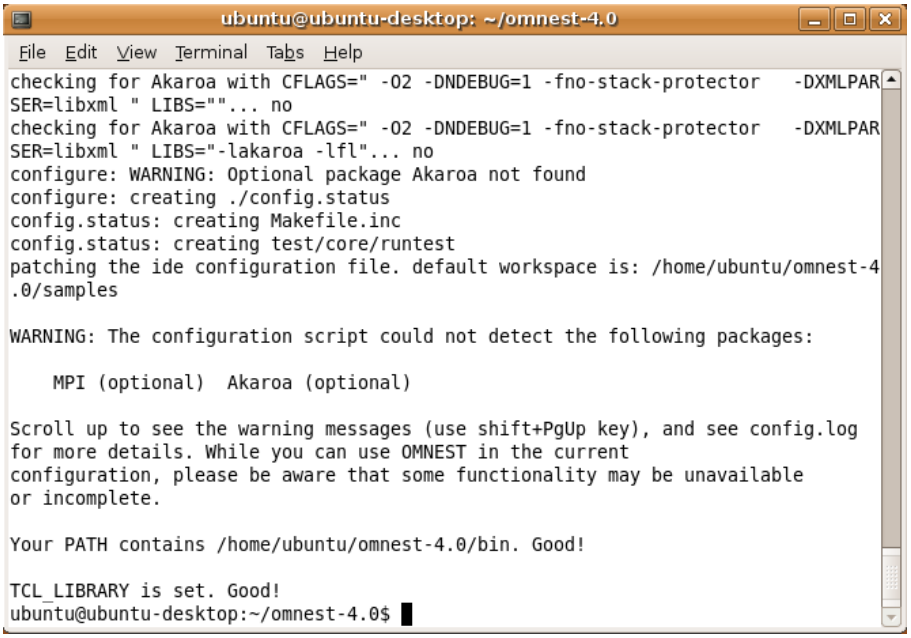


Fig. 12.1: Configuring OMNeT++

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 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.

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 OMNeT++. Type in the terminal:

```
$ make
```

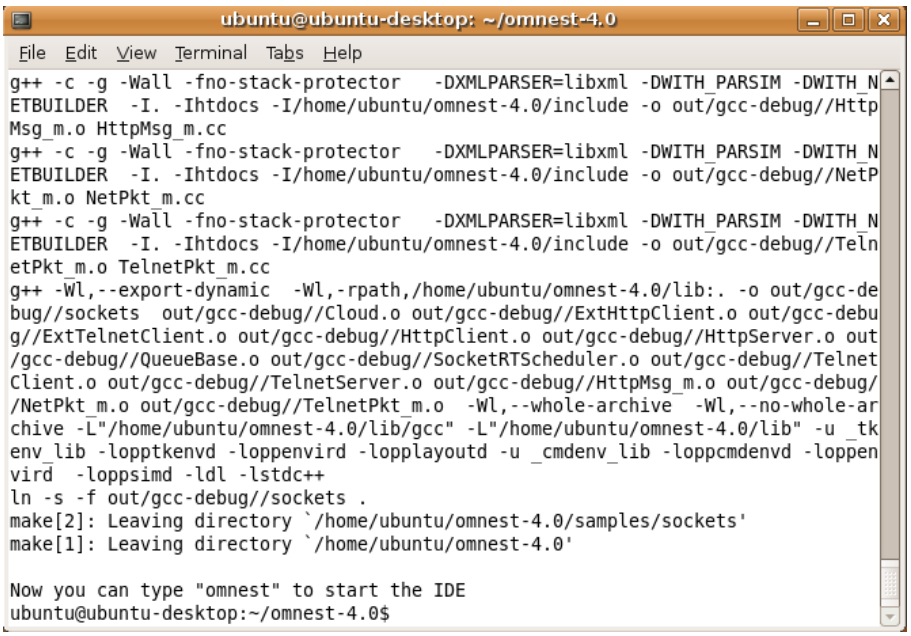


Fig. 12.2: Building OMNeT++

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`:

12.7 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.

12.8 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:

```
$ omnetpp
```

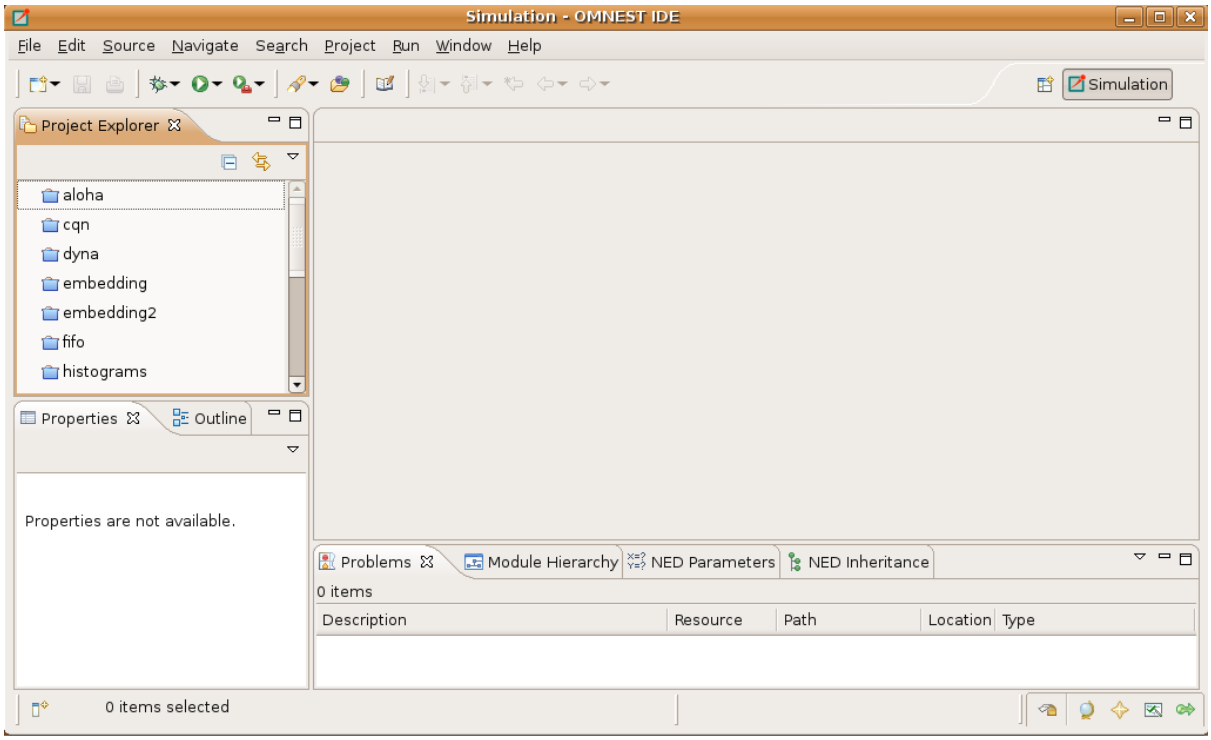


Fig. 12.3: The Simulation IDE

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.

12.9 Optional Packages

12.9.1 Akaroa

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

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

CHAPTER

THIRTEEN

BUILD OPTIONS

13.1 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 OMNeT++:

```
$ 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.

<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 OMNeT++.

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 OMNeT++ 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 OMNeT++ build process will create static or dynamic libraries. By default, the OMNeT++ 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 OMNeT++ libraries and then reconfigure (`./configure`) and recompile OMNeT++ (`make cleanall; make`). Once the OMNeT++ 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 OMNeT++ 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 OMNeT++ 4.0 to help porting model code from OMNeT++ 3.x, and having fulfilled their role, they were removed in OMNeT++ 5.0. If you still have old model code to port, use OMNeT++ 4.x.

13.2 Moving the Installation

When you build OMNeT++ on your machine, several directory names are compiled into the binaries. This makes it easier to set up OMNeT++ 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 OMNeT++ 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 OMNeT++ 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.

13.3 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.