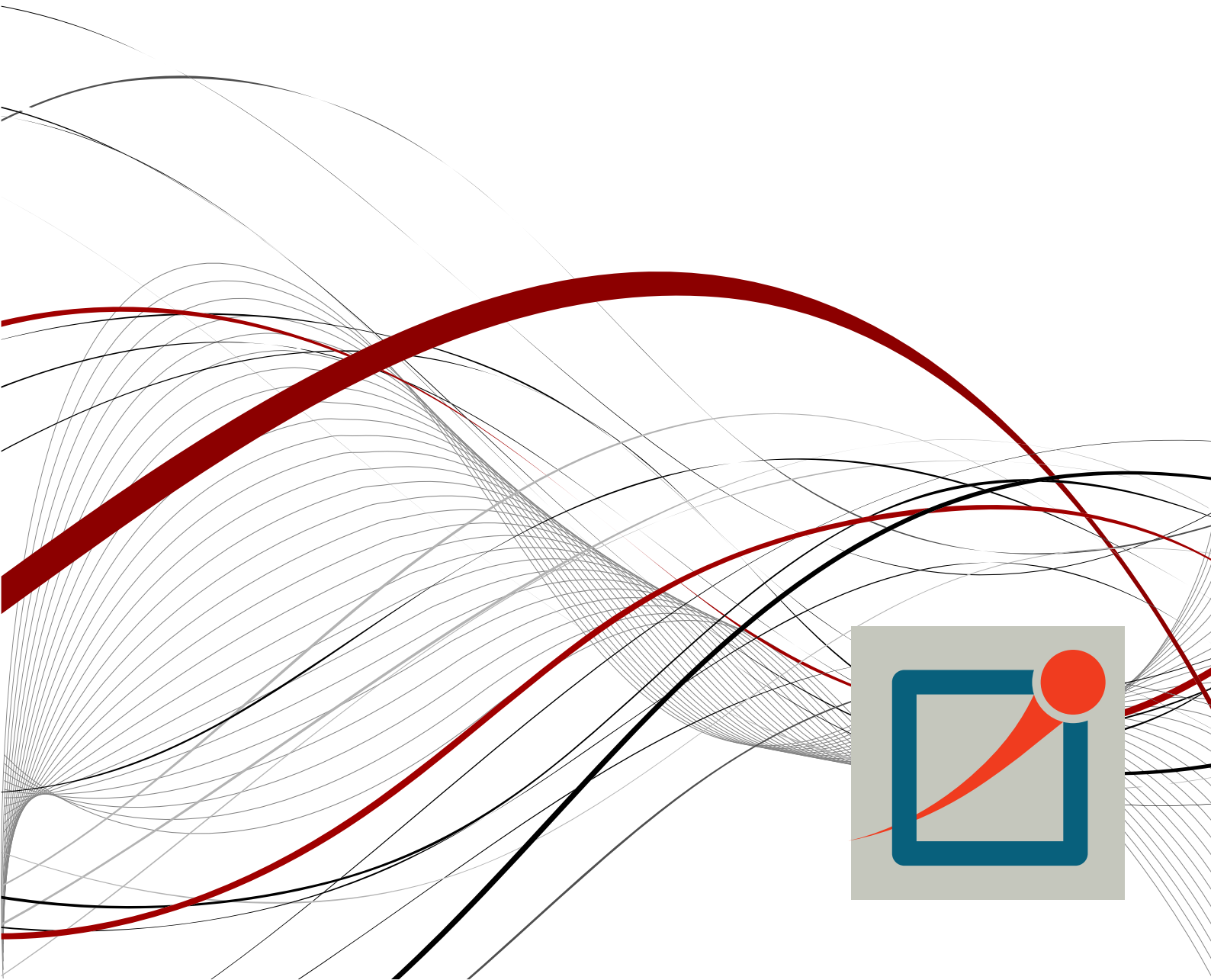
OMNEST

Installation Guide

Version 6.1



Build: 241025-2758eec6d6

CONTENTS

1 General Information **1**
- 1.1 Introduction . . . 1
- 1.2 Supported Platforms . . . 1

2 Windows - Using the Installer **3**
- 2.1 Supported Windows Versions . . . 3
- 2.2 Pre-installation Steps . . . 3
- 2.3 Installing OMNEST . . . 3
- 2.4 Using the IDE . . . 6
- 2.5 Using OMNEST with the MinGW GCC Compiler . . . 7
- 2.6 Using OMNEST with Microsoft Visual Studio Clang compiler . . . 8
- 2.7 Switching Compilers . . . 9
- 2.8 Additional Packages . . . 10

3 macOS **11**
- 3.1 Supported Releases . . . 11
- 3.2 Installing the Prerequisite Packages . . . 11
- 3.3 Enabling Development Mode in Terminal . . . 12
- 3.4 Debugging Unsigned Code . . . 13
- 3.5 Additional Steps Required on macOS to Use the Debugger . . . 13
- 3.6 Downloading and Unpacking OMNEST . . . 13
- 3.7 Environment Variables . . . 13
- 3.8 Configuring and Building OMNEST . . . 14
- 3.9 Verifying the Installation . . . 14
- 3.10 Starting the IDE . . . 15
- 3.11 Using the IDE . . . 15
- 3.12 Reconfiguring the Libraries . . . 15
- 3.13 Additional Packages . . . 16

4 Linux **17**
- 4.1 Supported Linux Distributions . . . 17
- 4.2 Installing the Prerequisite Packages . . . 17
- 4.3 Downloading and Unpacking . . . 18
- 4.4 Environment Variables . . . 18
- 4.5 Configuring and Building OMNEST . . . 18
- 4.6 Verifying the Installation . . . 20
- 4.7 Starting the IDE . . . 20
- 4.8 Using the IDE . . . 21
- 4.9 Reconfiguring the Libraries . . . 21
- 4.10 Additional Packages . . . 22

5 Ubuntu **23**
- 5.1 Supported Releases . . . 23
- 5.2 Opening a Terminal . . . 23
- 5.3 Installing the Prerequisite Packages . . . 23

6 Fedora 33 **27**
6.1 Supported Releases . . . 27
6.2 Opening a Terminal . . . 27
6.3 Installing the Prerequisite Packages . . . 27

7 Red Hat **29**
7.1 Supported Releases . . . 29
7.2 Opening a Terminal . . . 29
7.3 Installing the Prerequisite Packages . . . 29
7.4 SELinux . . . 30

8 OpenSUSE **31**
8.1 Supported Releases . . . 31
8.2 Opening a Terminal . . . 31
8.3 Installing the Prerequisite Packages . . . 31

9 Generic Unix **33**
9.1 Introduction . . . 33
9.2 Dependencies . . . 33
9.3 Determining Package Names . . . 34
9.4 Downloading and Unpacking . . . 34
9.5 Environment Variables . . . 35
9.6 Configuring and Building OMNEST . . . 35
9.7 Verifying the Installation . . . 37
9.8 Starting the IDE . . . 37
9.9 Optional Packages . . . 38

10 Build Options **39**
10.1 Configure.user Options . . . 39
10.2 Moving the Installation . . . 40
10.3 Using Different Compilers . . . 41

CHAPTER

ONE

GENERAL INFORMATION

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 has been tested and is supported on the following operating systems:

- Windows on x86_64 architecture
- macOS 13 and 14 on x86_64/aarch64 architecture
- Linux distributions covered in this Installation Guide

64-bit precompiled binaries are provided for the following platforms:

- Windows with Microsoft Visual C++ 2017 / ClangC2
- Windows with the bundled MinGW-w64 gcc/clang compiler

On other platforms, OMNEST needs to be compiled from source.

The Simulation IDE is supported on the following platforms:

- Linux x86_64/aarch64
- Windows x86_64
- macOS 13 and 14 (x86_64/aarch64)

Note: Simulations can be run practically on any unix-like environment with a decent and fairly up-to-date C++ compiler, for example gcc 8.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.

CHAPTER

TWO

WINDOWS - USING THE INSTALLER

2.1 Supported Windows Versions

OMNEST supports 64-bit versions of Windows.

2.2 Pre-installation Steps

Download `omnest-6.1-win64.exe` from https://omnest.com.

Note: The MinGW GCC and Clang compilers are bundled with OMNEST. You do not need to install any additional compiler to work with OMNEST. These comilers use the industry standard Itanium C++ ABI (http://itanium-cxx-abi.github.io/cxx-abi/. If you would like to use Microsoft ABI compliant compilers from *MS Visual Studio* or *MS Build Tools*, you should install *MSVC v142 - VS 2019 C++ x64/x86 build tools* (from Build Tools 2019 or Visual Studio 2019) along with the *Microsoft Windows 10 SDK* and the *C++ Clang tools for Windows (9.0.0 - x64/x86)* component, before installing OMNEST.

2.3 Installing OMNEST

Find the downloaded installation file using Windows Explorer, and double-click the file. This will start the installation process.

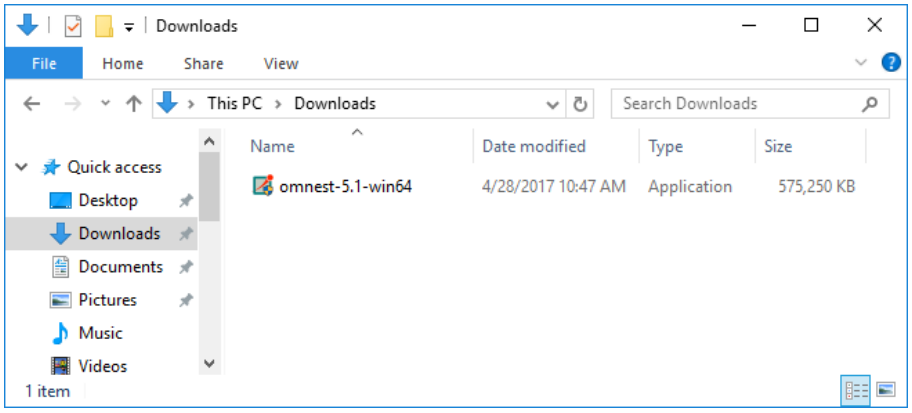


Fig. 2.1: Starting the installer

To start the installation, accept the Licensing agreements:

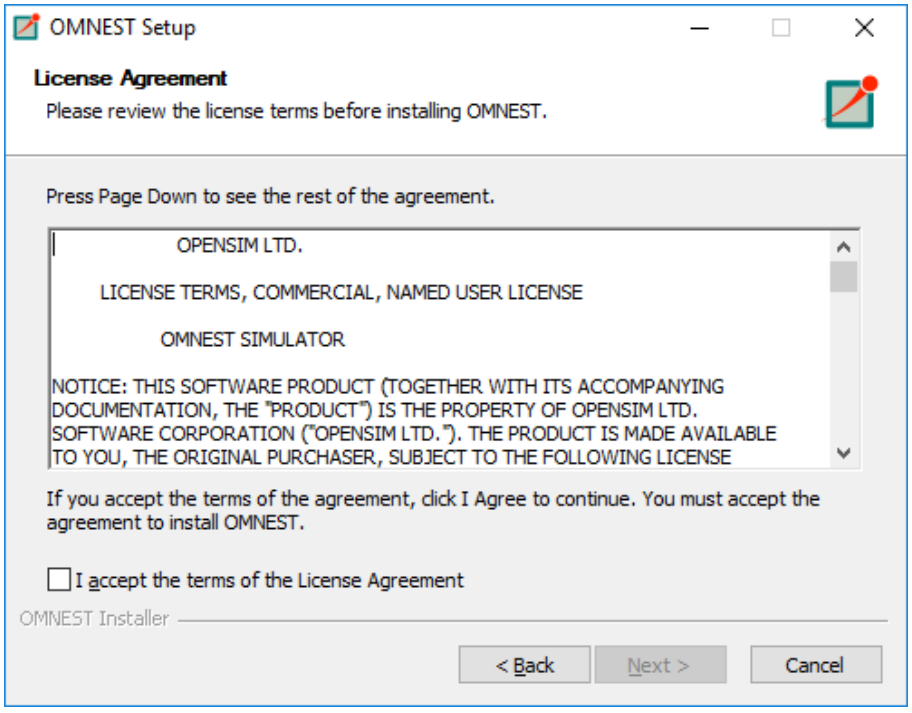


Fig. 2.2: License agreement

Select an installation target directory. Please make sure that the installation path does not contain spaces.

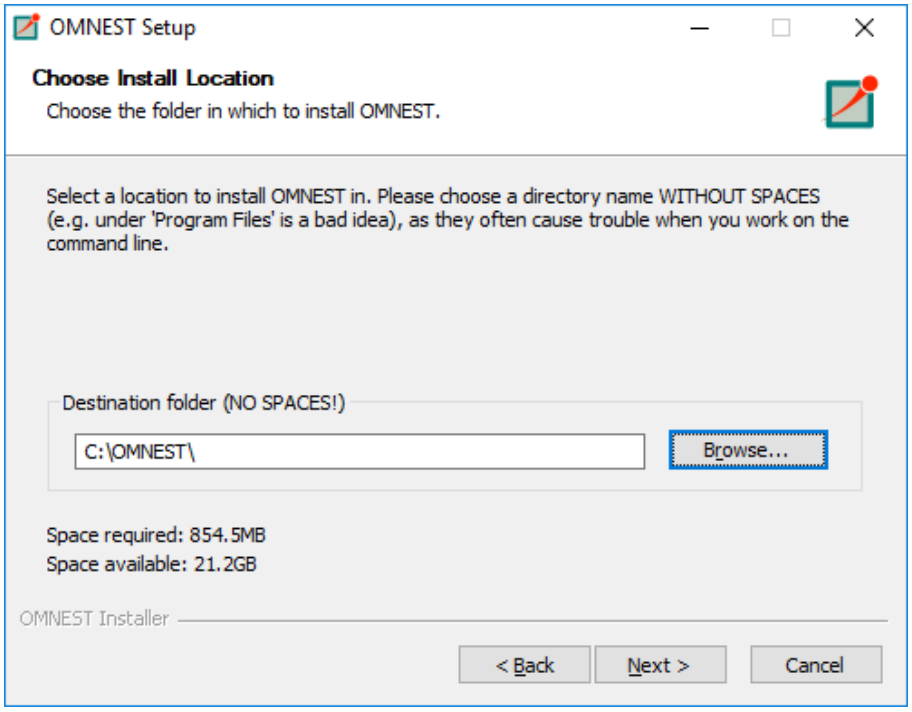


Fig. 2.3: Selecting the installation directory

On the next page you have to specify which compiler you intend to use with OMNEST. This can be the bundled MinGW GCC compiler (recommended), or the Clang compiler from Microsoft Visual Studio/MS Build Tools 2019. If you do not install any pre-built binary packages (none), you have to compile OMNEST manually after the installation has finished.

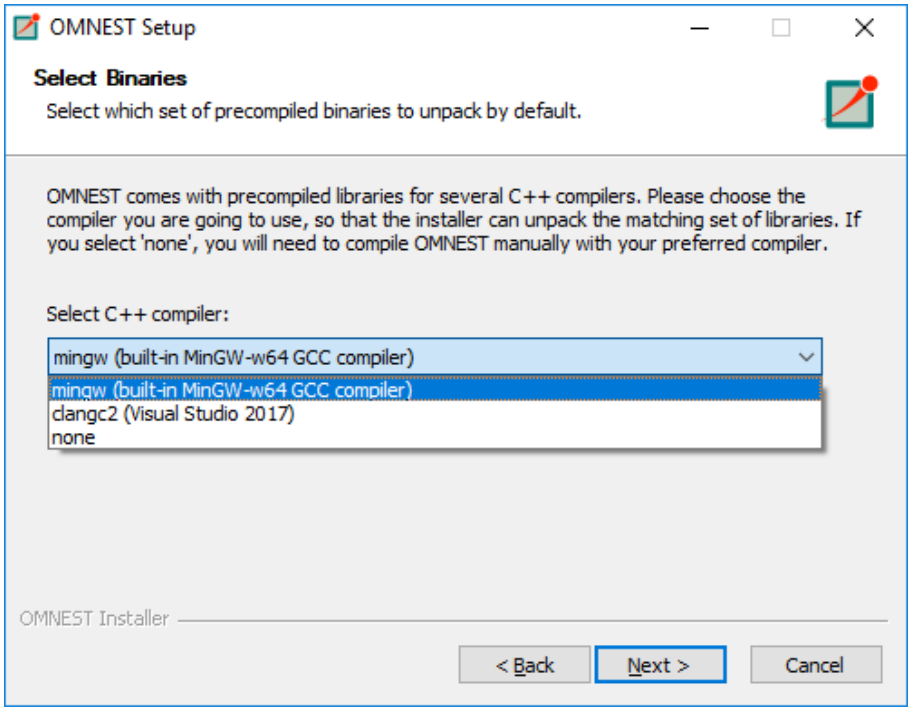


Fig. 2.4: Compiler selection

On the last page, you can optionally create program launcher icons for your desktop, too.

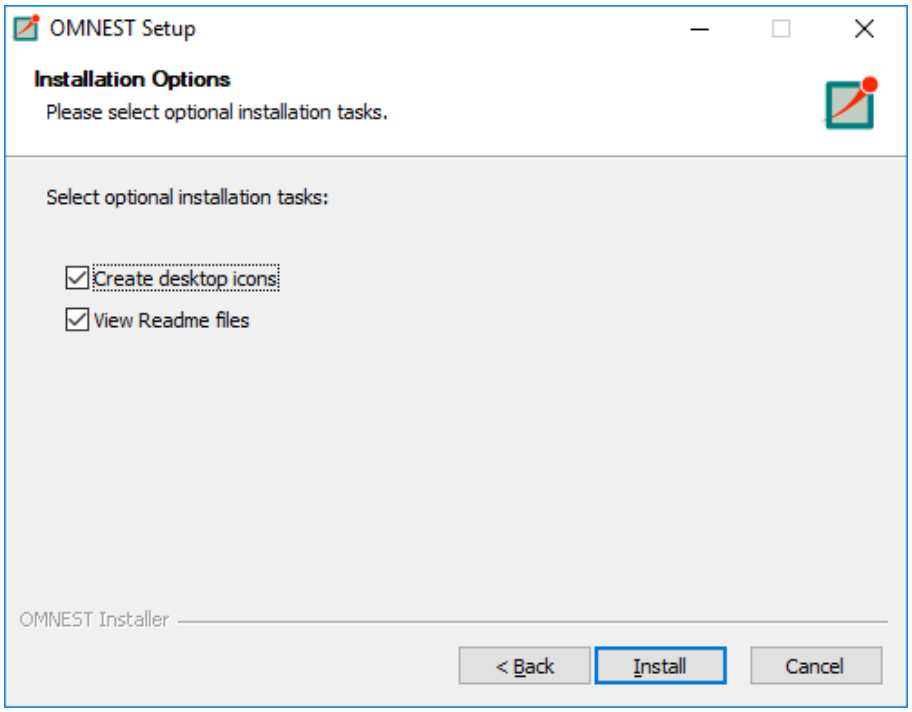


Fig. 2.5: Installation options

After this step the installation process starts, and all files required by OMNEST will be copied to the installation folder.

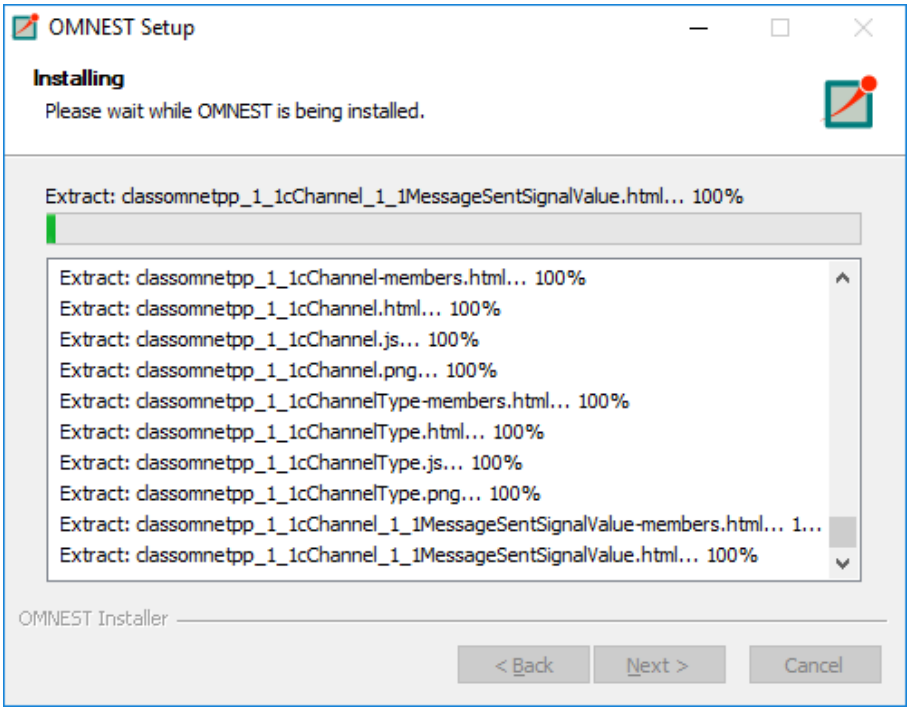


Fig. 2.6: Installation options

Finally, a new Start Menu folder is created (along with desktop shortcuts). You can start the *OMNEST Shell* or the *OMNEST IDE* by clicking on the icons.

Tip: If you want to work from the command line, use the provided *OMNEST Shell*. This shell sets all environment variables and the path necessary to run OMNEST simulations.

2.4 Using the IDE

Once the installation has finished, you can start using OMNEST by launching the IDE. The IDE can be launched with the corresponding Start Menu or desktop shortcut, or by typing `omnest` at the OMNEST Shell prompt.

In the IDE, each simulation example is a separate project. To build an example, open its project using the context menu (right-click, *Open Project*), and click the *Run* button on the toolbar. To rebuild the example, first make sure that the correct configuration is selected (context menu, *Build Configurations* > *Set Active*), then choose *Clean Project* and *Build Project* from the context menu.

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

2.5 Using OMNEST with the MinGW GCC Compiler

The bundled MinGW GCC compiler is preconfigured for OMNEST. Note that only the bundled version of MinGW has been tested and is supported with OMNEST.

If you have installed the pre-compiled binary package for the MinGW compiler, make sure that the `<installdir>/bin` and `<installdir>/lib` directories contain the correct executables and libraries. You should see `libopp*.dll` files in the `<installdir>/bin` directory and similarly named `*.a` files in `<installdir>/lib`. If you do not see them, check the "Switching Compilers" or the "Recompiling OMNEST" section before proceeding.

To test the installation, try to run models from the `<installdir>/samples` directory.

2.5.1 Compiling Simulations on the Command Line

To build simulations from the command line, the following directories have to be included in the path: `<installdir>\bin`, `<installdir>\tools\win64\usr\bin` and `<installdir>\tools\win64\mingw64\bin`.

OMNEST provides a *OMNEST MinGW Shell* window (`mingwenv.cmd`), which sets the path and environment variables.

Before compiling a model, you must generate a `Makefile` for it. Change into the model directory and execute:

```
$ opp_makemake -f --deep
```

This command will generate a `Makefile` that can compile all your .cc files in the model directory.

```
$ make
```

will compile and build your project.

Tip: Be sure to check the *Manual* for all the options and features of `opp_makemake`.

2.5.2 Recompiling OMNEST

Open the *OMNEST MinGW Shell* window and type:

```
$ ./configure
```

This command will detect all required software on your machine, and configure your build environment. The configuration process creates a file called `Makefile.inc` in your installation root. This file will be included in all of your makefiles, and contains all variables, paths and settings for the build process.

If you do not have binary files in your `bin` directory (no pre-compiled binaries were installed), you should compile OMNEST now manually by typing:

```
$ make
```

Tip: The above command creates both debug and release versions of the libraries. If you want to create only one type, use the `make MODE=debug` or `make MODE=release` commands.

Tip: If you have a dual-core machine, you can speed up the compilation by adding the `-j2` option to the `make` command line, which enables parallel build support.

2.6 Using OMNEST with Microsoft Visual Studio Clang compiler

OMNEST comes with pre-built binaries for the Clang compiler. If you have installed the pre-compiled binary package for the Clang compiler, make sure that the `<installdir>/bin` and `<installdir>/lib` directories contain the correct executables and libraries. You should see `opp*.dll` files in the `<installdir>/bin` directory and `opp*.lib` files in the `<installdir>/lib` directory. If you do not see them, check the "Switching Compilers" or the "Recompiling OMNEST" section before proceeding.

To test the installation try to run models from the `<installdir>/samples` directory.

Note: Be sure to modify the path to point to your Visual Studio installation (VS_INSTALL_DIR) and Clang compiler version (ClangToolsInstallDir) in the `<installdir>\vcenv.cmd` file. By default, the file is using the MS Build Tools 2019 installation on your C: drive. If you have installed Visual Stuido instead of the Build Tools package, you must modify this file.

Note: For now, the OMNEST IDE cannot be used for debugging Visual C++ binaries. This is a limitation of the Eclipse CDT component that OMNEST build on. We recommend that you use the Visual Studio IDE for debugging.

2.6.1 Compiling Simulations on the Command Line (MS ABI)

To build simulations from the command line, the following directories have to be included in the path: `<installdir>\bin`, `<installdir>\tools\win64\usr\bin`, `<installdir>\tools\win64\visualc\bin` plus the directories required by Visual Studio Clang itself. To include the required Visual C++ directories, VC provides a batch file called `vcvars64.bat` in its `bin` directory.

OMNEST provides a *OMNEST Visual C++ Shell* window (`vcenv.cmd`), which sets the path and environment variables and properly class also the `vcvars64.bat` file.

Note: You may need to adjust your Visual Studio/MS Build Tools and Clang installation directory in the `vcenv.cmd` file.

Before compiling a model, you must generate a `Makefile` for it. Change into the model directory and execute:

```
> opp_makemake -f --deep
```

This command will generate a `Makefile` file that can compile all your .cc files in the model directory.

```
> make
```

will build your project.

2.6.2 Compiling Simulations from the IDE

Before compiling simulations from the IDE, make sure that the correct build configuration (*debug* or *release*) is selected then choose `Build` in the `Project` menu.

2.6.3 Recompiling OMNEST

Open the *OMNEST Visual C++ Shell* window (`vcenv.cmd`). Make sure that `USE_MS_ABI` is set to *yes* (and uncommented) in the `configure.user` file and configure the installation with `./configure`.

If you do not have binary files in your `bin` directory (no pre-compiled binaries were installed), you should compile OMNEST now manually by typing:

```
> make
```

2.7 Switching Compilers

If you want to switch compilers after you have installed OMNEST, we recommend uninstalling the software and reinstalling it with the selected new compiler.

It is also possible to manually change the compiler used:

First you have to delete all executable files generated by that compiler.

Open the *OMNEST MinGW Shell* window (`mingwenv.cmd`) and clean OMNEST by executing the following command in `<installdir>`

```
$ make cleanall
```

Pre-built binaries are stored in the `<installdir>/store` directory. You must extract their content in the root OMNEST directory. Execute:

```
$ 7za x store/mingw-bin.7z
```

or

```
$ 7za x store/clangc2-bin.7z
```

depending on your compiler. After extracting the executables you will be able to run the sample simulations immediately.

Finally, you may need to modify the shortcut that is used to start the IDE. Open the shortcut properties and change the command to “mingw.env ide” or “vcenv.cmd ide” depending on the compiler you intend to use.

Note: Be sure to modify the path to your Visual Studio installation (VS_INSTALL_DIR) and Clang compiler version (ClangToolsInstallDir) in the `<installdir>\vcenv.cmd` file if you are switching between different versions of Visual Studio or MS Build Tools.

2.8 Additional Packages

Note that Doxygen and GraphViz are already included in the OMNEST package, and will be used by the IDE automatically.

2.8.1 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.

DeinoMPI ships with binaries compiled with MSVC. After installing DeinoMPI, adjust the `MPI_DIR` setting in `configure.user`, and recompile OMNEST with the version of MSVC that matches the DeinoMPI binaries.

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

2.8.2 Akaroa

Akaroa 2.6.7, 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.

2.8.3 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

THREE

MACOS

3.1 Supported Releases

This chapter provides additional information for installing OMNEST on macOS.

The following releases are covered:

- macOS 14.x

3.2 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.

3.2.1 Intel-based Macs

On an Intel-based Mac, OMNEST is bundled with all the required external dependencies in the *tools* directory of the archive, so no additional steps are needed. (You can still opt to install the external dependencies using Homebrew.)

3.2.2 Apple Silicon

On an Apple Silicon-based computer, external dependencies must be installed manually, using a 3rd party package manager like Homebrew. With Homebrew, you can install the packages with the following command:

```
$ brew install bison flex perl python@3 make qt@5 pkg-config \
    doxygen graphviz openscenegraph
```

Make sure that the following lines are sourced in your shell (e.g. add them to your *.zprofile* file or create a new file for them):

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

export PATH="$(brew --prefix qt@5)/bin:$PATH"
export PATH="$(brew --prefix bison)/bin:$PATH"
export PATH="$(brew --prefix flex)/bin:$PATH"
export PATH="$(brew --prefix doxygen)/bin:$PATH"
export PATH="$(brew --prefix graphviz)/bin:$PATH"
```

(continues on next page)

(continued from previous page)

```
export PATH="$(brew --prefix pkg-config)/bin:$PATH"
export PATH="$(brew --prefix make)/libexec/gnubin:$PATH"
export LDFLAGS="-L$(brew --prefix)/lib $LDFLAGS"
export CFLAGS="-I$(brew --prefix)/include $CFLAGS"
```

Restart your shell to activate the above changes and make sure that you are using *python3* from Homebrew (*which python3*) and not the system's Python interpreter provided by macOS (*/usr/lib/python3*) and then install the Python dependencies:

```
$ python3 -m venv .venv && source .venv/bin/activate
$ python3 -m pip install -r python/requirements.txt
```

3.3 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.

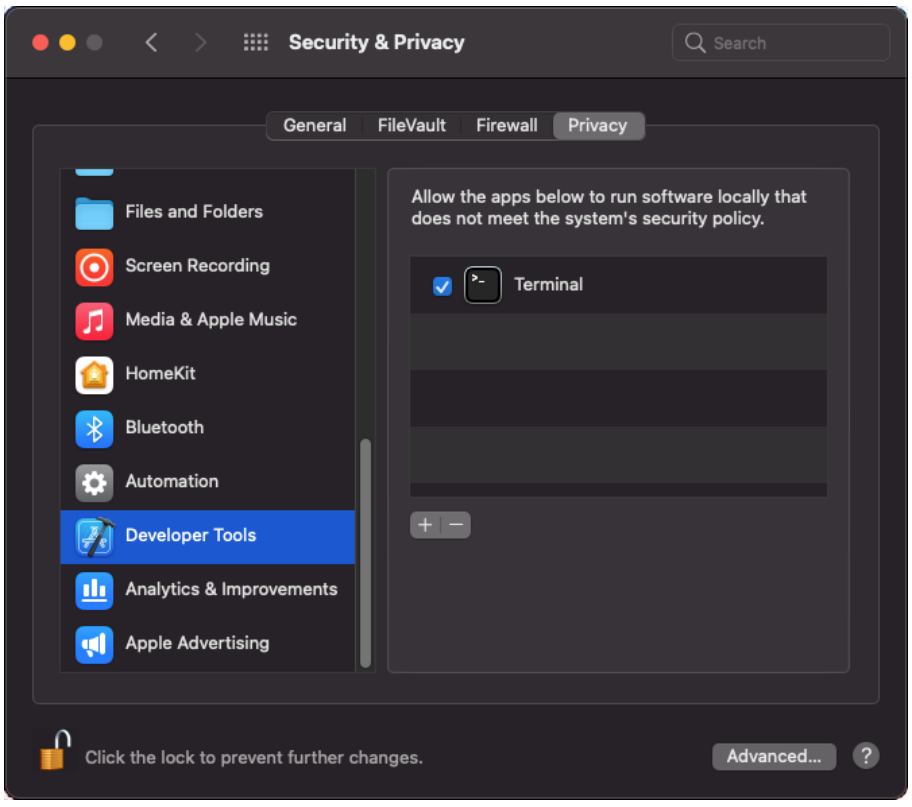


Fig. 3.1: Enable Running Unsigned Code in Terminal

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.

3.4 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.

3.5 Additional Steps Required on macOS to Use the Debugger

The Command Line Developer Tools package contains the `lldb` debugger. OMNEST 6.0 and later contains the necessary driver binary (`lldbmi2`) that allows `lldb` to be used in the OMNEST IDE. 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 `lldbmi2` as the debugger executable.

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

3.6 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-6.1-macos-aarch64.tgz` (for Apple Silicon) or `omnest-6.1-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-6.1-macos-aarch64.tgz
```

A subdirectory called `omnest-6.1` 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.

3.7 Environment Variables

In general OMNEST requires that certain environment variables are set and the `omnest-6.1/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnest-6.1
$ 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 "$HOME/omnest-6.1/setenv" ] && source "$HOME/omnest-6.1/setenv"
```

3.8 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`:

3.9 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.

3.10 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-6.1` 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.

3.11 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*.

3.12 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.

3.13 Additional Packages

3.13.1 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`.

3.13.2 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

FOUR

LINUX

4.1 Supported Linux Distributions

This chapter provides instructions for installing OMNEST on selected Linux distributions:

- Ubuntu 22.04 and 24.4 LTS
- Fedora Workstation 31
- Red Hat Enterprise Linux Desktop Workstation 8.x
- OpenSUSE Leap 15.3

This chapter describes the overall process. Distro-specific information, such as how to install the prerequisite packages, are covered by distro-specific chapters.

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

4.2 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.

4.3 Downloading and Unpacking

Download OMNEST from https://omnest.com. Make sure you select to download the Linux specific archive, `omnest-6.1-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-6.1-linux-x86_64.tgz
```

This will create an `omnest-6.1` 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.

4.4 Environment Variables

In general OMNEST requires that certain environment variables are set and the *omnest-6.1/bin* directory is in the PATH. Source the *setenv* script to set up all these variables.

```
$ cd omnest-6.1
$ source setenv
```

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

```
[ -f "$HOME/omnest-6.1/setenv" ] && source "$HOME/omnest-6.1/setenv"
```

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

4.5 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.

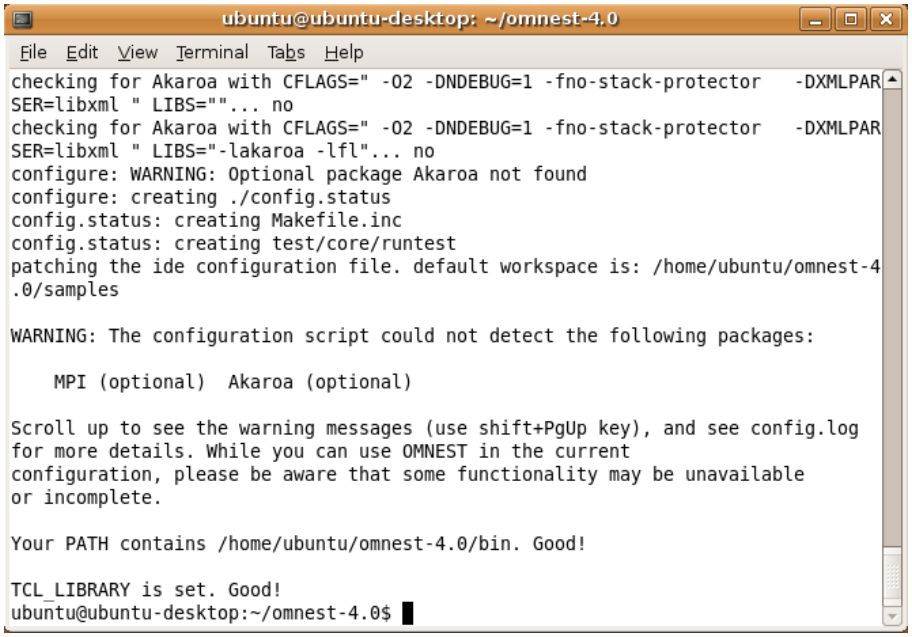


Fig. 4.1: Configuring OMNEST

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
```

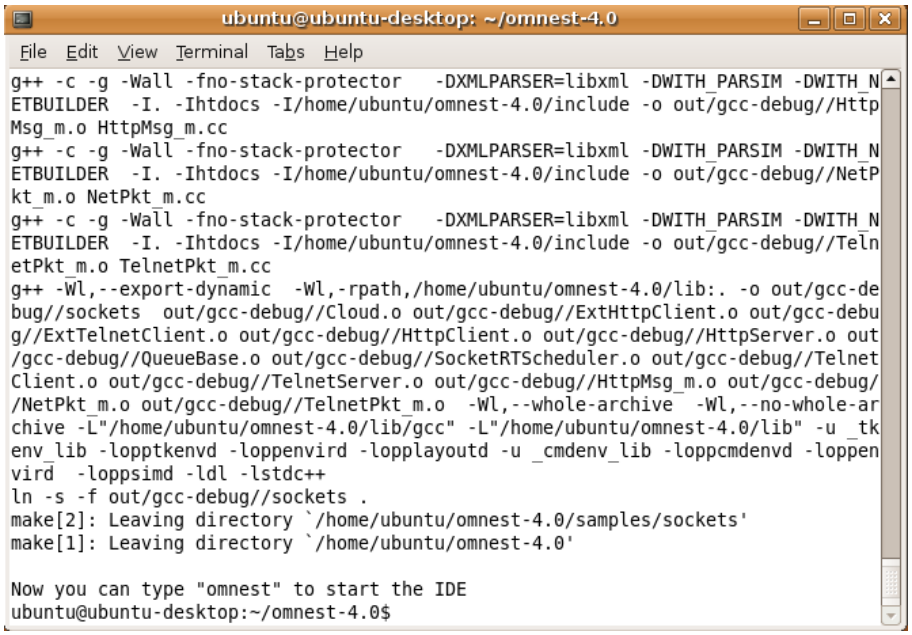


Fig. 4.2: Building OMNEST

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`:

4.6 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.

4.7 Starting the IDE

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

```
$ omnest
```

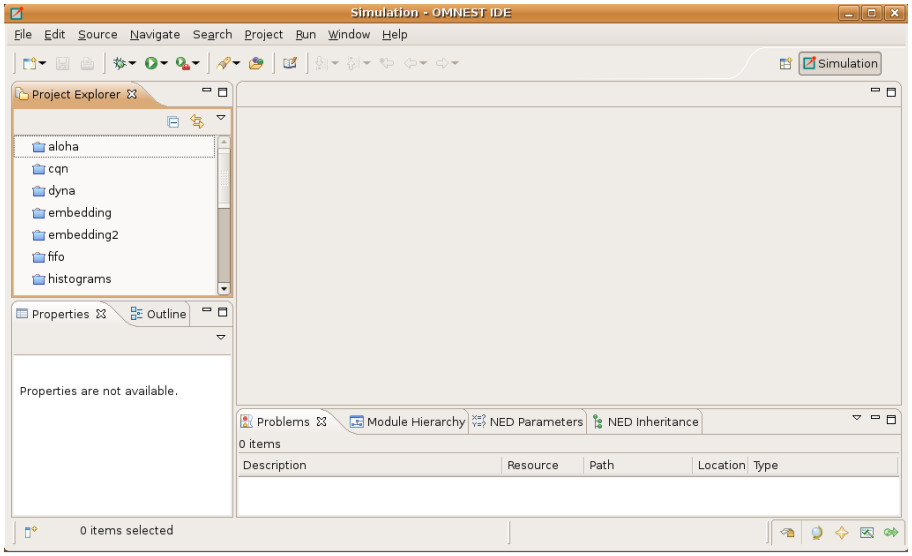


Fig. 4.3: The Simulation IDE

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.

4.8 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*.

4.9 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.

4.10 Additional Packages

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

4.10.1 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`.

4.10.2 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.

4.10.3 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

FIVE

UBUNTU

5.1 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 covered:

- Ubuntu 22.04 LTS or 24.04 LTS

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.

5.2 Opening a Terminal

Type *terminal* in your program launcher and click on the Terminal icon.

5.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

5.3.1 Command-Line Installation

Before starting the installation, refresh the database of available packages. Type in the terminal:

```
$ sudo apt-get update
```

To install the required packages, change into the root of the OMNEST installation and type in the terminal:

```
$ sudo apt-get install build-essential clang lld gdb bison flex perl \
    python3 python3-pip libpython3-dev qtbase5-dev qtchooser qt5-qmake␣
↪qtbase5-dev-tools \
    libqt5opengl5-dev libxml2-dev zlib1g-dev doxygen graphviz \
    libwebkit2gtk-4.1-0 xdg-utils libdw-dev
$ python3 -m venv .venv && source .venv/bin/activate
$ python3 -m pip install -r python/requirements.txt
```

To use Qtenv with 3D visualization support, install the development packages for OpenSceneGraph (3.4 or later) and the osgEarth (2.9 or later) packages. (You may need to enable

the *Universe* software repository in Software Sources. and also enable *WITH_OSGEARTH* in *configure.user*.)

```
$ sudo apt-get install openscenegraph-plugin-osgearth libosgearth-dev
```

Warning: Ubuntu 22.04 no longer provides the *libosgearth* package so osgEarth must be installed from sources. OpenSceneGraph can still be installed using *sudo apt-get install libopenscenegraph-dev*.

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. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

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*.

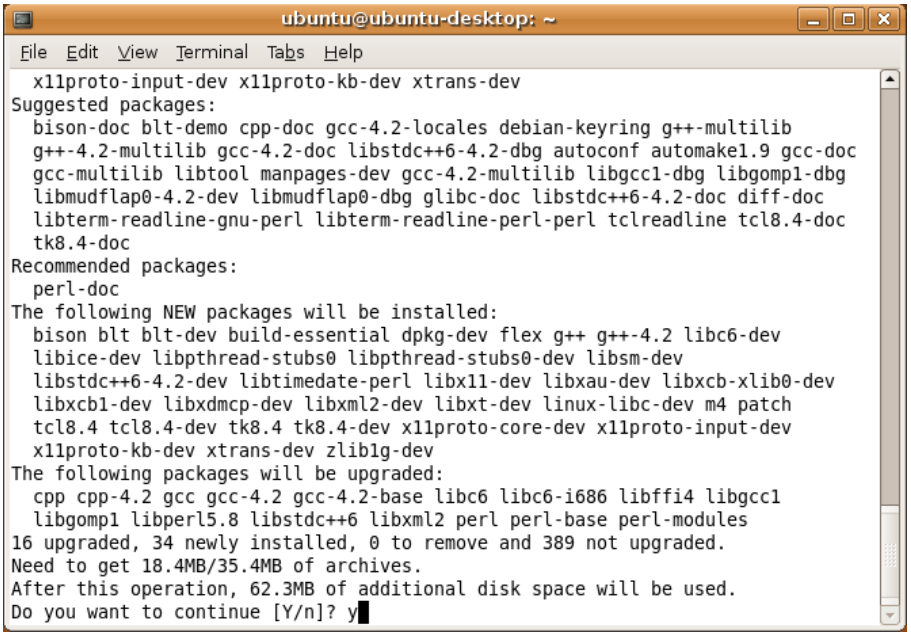


Fig. 5.1: Command-Line Package Installation

5.3.2 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

SIX

FEDORA 33

6.1 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 covered:

- Fedora 40

It was tested on the following architectures:

- Intel 64-bit

6.2 Opening a Terminal

Open the Search bar, and type *Terminal*.

6.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

6.3.1 Command-Line Installation

To install the required packages, type in the terminal:

```
$ sudo dnf install make gcc gcc-c++ clang lld bison flex perl \
    python3-devel python3-pip qt5-qtbase-devel libxml2-devel \
    zlib-devel doxygen graphviz xdg-utils libdwarf-devel webkit2gtk4.1
$ python3 -m venv .venv && source .venv/bin/activate
$ python3 -m pip install -r python/requirements.txt
```

To use 3D visualization support in Qtenv, you should install OpenSceneGraph 3.2 or later and osgEarth 2.7 or later (recommended):

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

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. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

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

SEVEN

RED HAT

7.1 Supported Releases

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

The following Red Hat release is covered:

- Red Hat Enterprise Linux Desktop Workstation 8.x

It was tested on the following architectures:

- Intel 64-bit

7.2 Opening a Terminal

Choose *Applications* > *Accessories* > *Terminal* from the menu.

7.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

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

7.3.1 Command-Line Installation

To install the required packages, change into the root of the OMNEST installation and type in the terminal:

```
$ su -c 'yum install make gcc gcc-c++ clang lld bison flex perl \
    python3-devel python3-pip qt5-qtbase-devel libxml2-devel \
    zlib-devel doxygen graphviz xdg-utils libdwarf-devel'
$ python3 -m venv .venv && source .venv/bin/activate
$ python3 -m pip install -r python/requirements.txt
```

To use 3D visualization support in Qtenv (recommended), you should install the OpenSceneGraph-devel (3.2 or later) and osgEarth-devel (2.7 or later) packages. These packages are not available from the official RedHat repository so you may need to get them from different sources (e.g. rpmfind.net).

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. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

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 `i386` or `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`).`

7.4 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

EIGHT

OPENSUSE

8.1 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 covered:

- openSUSE Leap 15.6

It was tested on the following architectures:

- Intel 64-bit

8.2 Opening a Terminal

Open the Search bar, and type *Terminal*.

8.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

8.3.1 Command-Line Installation

To install the required packages, type in the terminal:

```
$ sudo zypper install make gcc gcc-c++ clang lld bison flex perl \
    python3-devel python3-pip qt5-qtbase-devel libxml2-devel \
    zlib-devel doxygen graphviz xdg-utils libdw-devel libwebkit2gtk-4_1-0
$ python3 -m venv .venv && source .venv/bin/activate
$ python3 -m pip install -r python/requirements.txt
```

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. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

To use 3D visualization support in Qtenv (recommended), you should install the OpenSceneGraph-devel (3.2 or later) and osgEarth-devel (2.7 or later) packages. These packages are not available from the official RedHat repository so you may need to get them from different sources (e.g. rpmfind.net).

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

NINE

GENERIC UNIX

9.1 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.

9.2 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.

9.3 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.

9.3.1 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.

9.3.2 MPI

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

9.4 Downloading and Unpacking

Download OMNEST from https://omnest.com. Make sure you select to download the generic archive, `omnest-6.1-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-6.1-core.tgz
```

This will create an `omnest-6.1` subdirectory with the OMNEST files in it.

9.5 Environment Variables

In general OMNEST requires that certain environment variables are set and the `omnest-6.1/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnest-6.1
$ source setenv
```

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

```
[ -f "$HOME/omnest-6.1/setenv" ] && source "$HOME/omnest-6.1/setenv"
```

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

9.6 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.

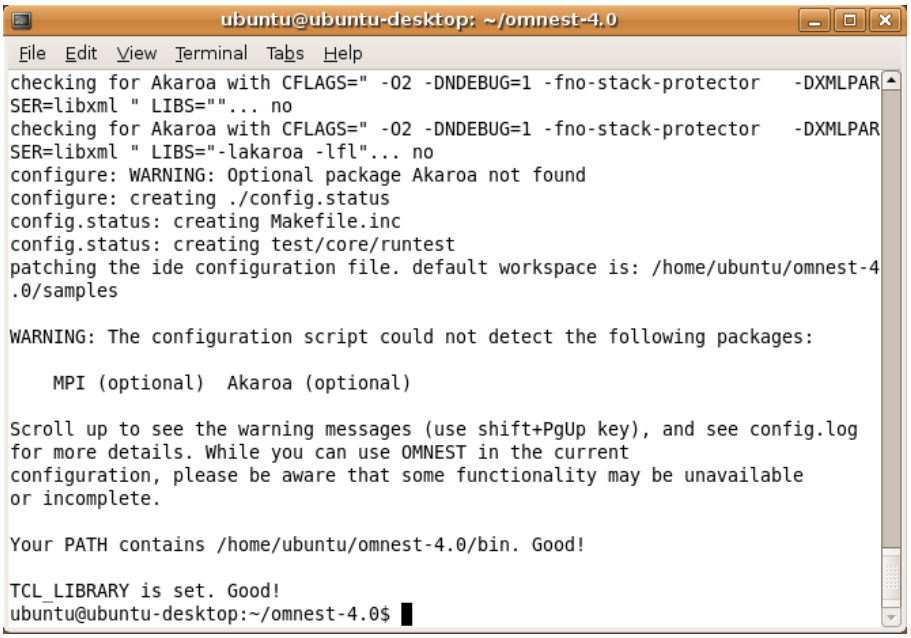


Fig. 9.1: Configuring OMNEST

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. 9.2: 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`:

9.7 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.

9.8 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
```

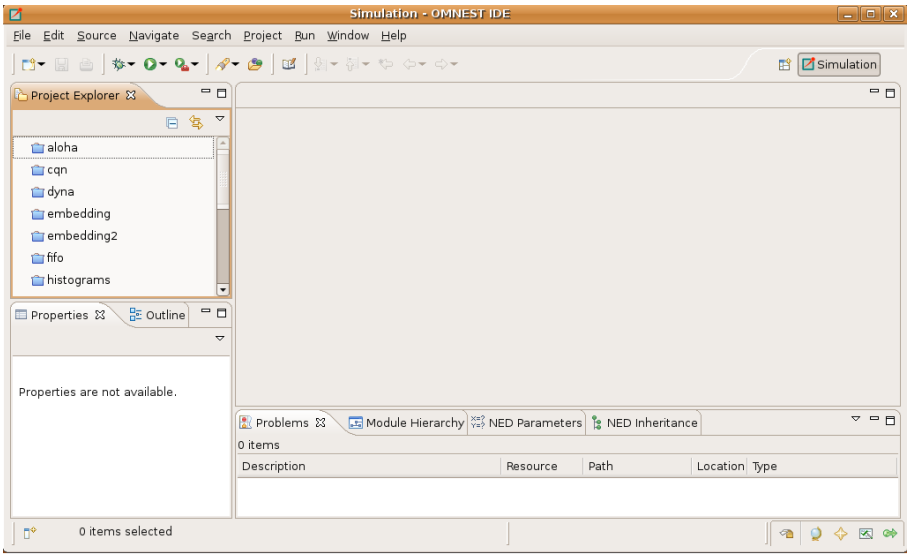


Fig. 9.3: The Simulation IDE

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.

9.9 Optional Packages

9.9.1 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

TEN

BUILD OPTIONS

10.1 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.

10.2 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.

10.3 Using Different Compilers

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

- Intel compiler (icc, icpc)
- GNU C/C++ (gcc, g++)
- Clang (clang, clang++)
- Clang/C2 (from Microsoft Visual Studio)
- Sun Studio (cc, cxx)
- IBM compiler (xlc, xlC)

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.