OMNeT++

User Guide

Version 4.1

Chapter 1. Introduction

The OMNeT++ simulation IDE is based on the Eclipse platform and extends it with new editors, views, wizards, and other functionality. OMNeT++ adds functionality for creating and configuring models (NED and INI files), performing batch executions and analyzing the simulation results, while Eclipse provides C++ editing, SVN/GIT integration and other optional features (UML modeling, bug-tracker integration, database access, etc.) via various open-source and commercial plug-ins. The environment will be instantly recognizable to those at home with the Eclipse platform.

1.1. The Workbench

The Eclipse main window consists of various Views and Editors. These are collected into Perspectives that define which Views and Editors are visible and how they are sized and positioned.

Eclipse is a very flexible system. You can move, resize, hide and show various panels, editors and navigators. This allows you to customize the IDE to your liking, but it also makes it more difficult to describe. First, we need to make sure that we are looking at the same thing.

The OMNeT++ IDE provides a "Simulation perspective" to work with simulation-related NED, INI and MSG files. To switch to the simulation perspective, select *Window | Open Perspective | Simulation*.

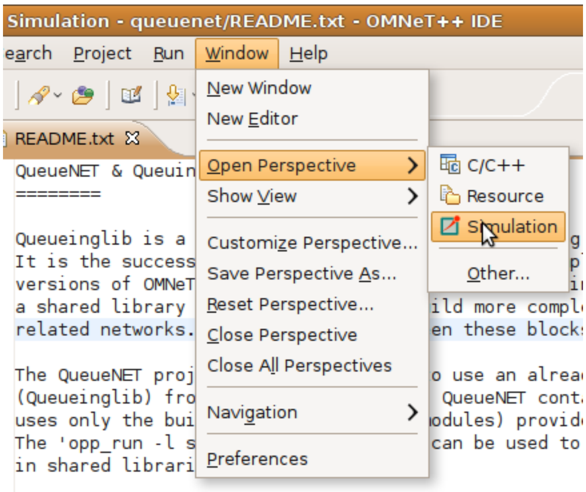


Figure 1.1. Selecting the "Simulation Perspective" in Eclipse

Most interface elements within Eclipse are can be moved or docked freely so you can construct your own workbench to fit your needs.

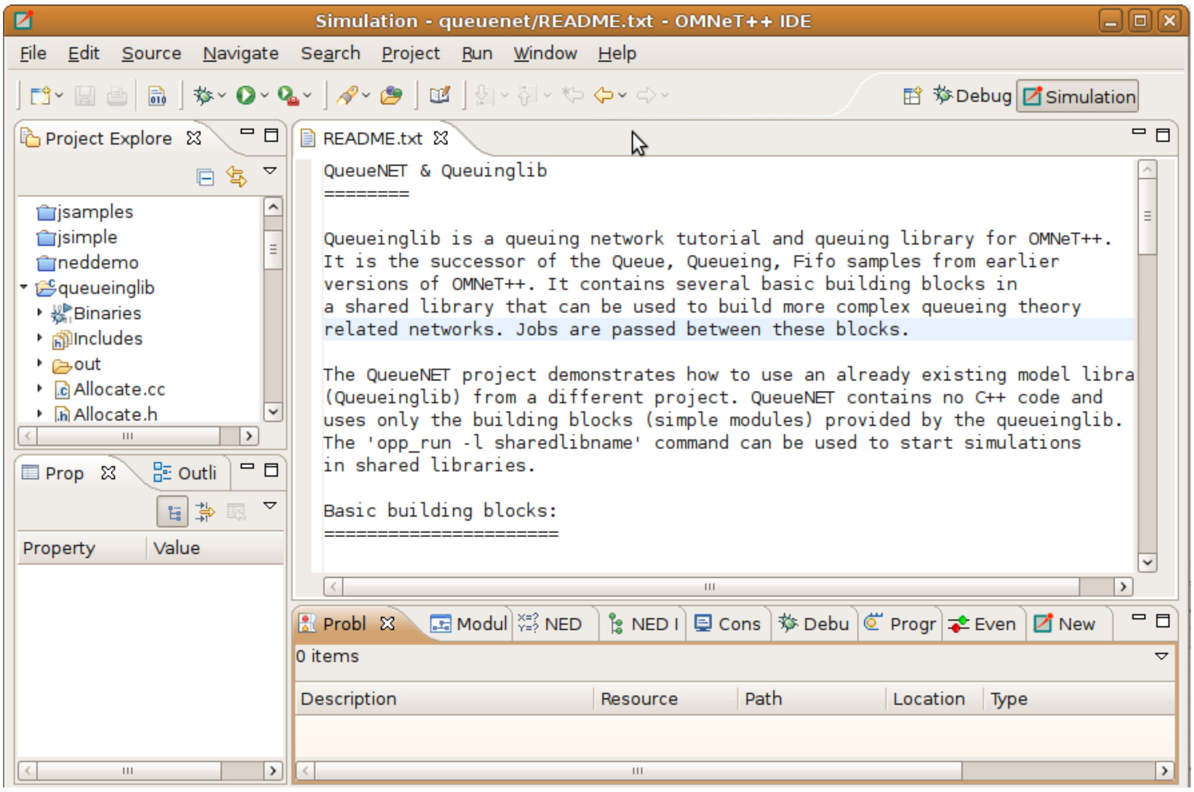


Figure 1.2. Default layout of the OMNeT++ IDE

The *Project Explorer* on the top left part of the screen shows the projects and their content in your workspace. In the example above, the *queueinglib* demo project is open. You can see the various `.ned`, `.ini` and other files inside. A number of views are docked at the bottom of the window.

The screenshot shows the open README.txt file in the editor area. When a user double-clicks on a file, Eclipse automatically launches the editor associated with that particular file type.

The *Properties View* contains information on the particular object that is selected in the editor area, or one of the other views that serve as a selection provider. The *Problems View* references code lines where Eclipse encountered problems.

Several OMNeT++-specific views exist that can be used during development. We will discuss how you can use them effectively in a later chapter. You can open any View by selecting *Window* | *Show View* from the menu.

1.2. Workspaces

A workspace is basically a directory where all your projects are located. You may create and use several workspaces and switch between them as needed. During the first run, the OMNeT++ IDE offers to open the samples directory as the workspace, so you will be able to experiment with the available examples immediately. Once you start working on your own projects, we recommend that you create your own workspace by selecting *File* | *Switch Workspace* | *Other*. You can switch between workspaces, as necessary. Please be aware that the OMNeT++ IDE restarts with each switch in workspaces. This is normal. You can browse workspace content in the *Project Explorer*, *Navigator*, *C/C++ Projects* and similar views. We recommend using *Project Explorer*.

1.3. The Simulation Perspective

The OMNeT++ IDE defines the *Simulation Perspective* so that it is specifically geared towards the design of simulations. The *Simulation Perspective* is simply a set of conveniently selected views, arranged to make the creation of NED, INI and MSG files easier. If you are working with INI and NED files a lot, we recommend selecting this perspective. Other perspectives are optimized for different tasks like C ++ development or debugging.

1.4. Configuring OMNeT++ Preferences

The OMNeT++ IDE preferences dialog is available through the standard preferences menu, which is under the main Window menu item. These settings are global and shared between all projects. The OMNeT++ install locations are automatically filled in for you after installation. The default settings for the NED documentation generation assume that the PATH environment variable is already set, so that third party tools can be found. The license configuration settings specify the preferred license type or a custom license text. The IDE will copy the license into new files and projects. The license will also be shown in the generated NED documentation.

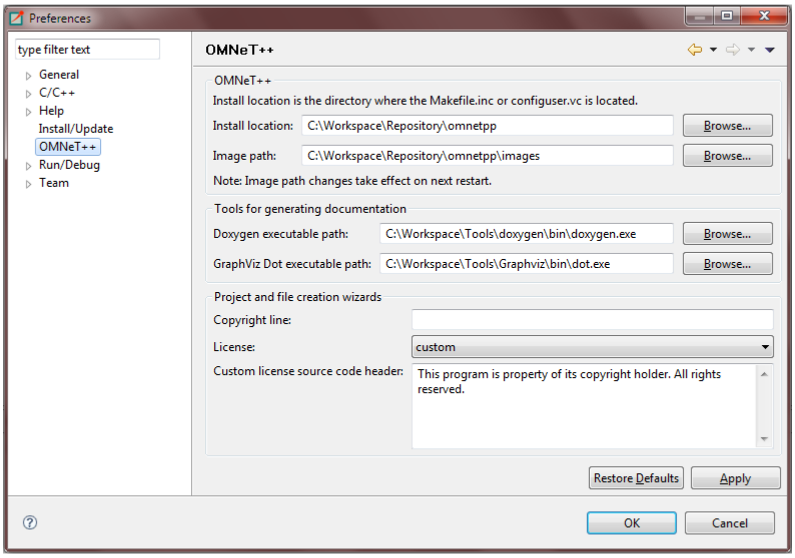


Figure 1.3. Configuring OMNeT++ preferences

Use the Browse buttons to find files or folders easily. Specify full path for executables if you do not want to extend the PATH environment variable.

1.5. Creating OMNeT++ Projects

In Eclipse, all files are within projects, so you will need a suitable project first. The project needs to be one designated as an OMNeT++ Project (in Eclipse lingo, it should have the OMNeT++ Nature). The easiest way to create such a project is to use a wizard. Choose *File|New|OMNeT++ Project...* from the menu, specify a project name, and click the *Finish* button. If you do not plan to write simple modules, you may unselect the *C++ Support* checkbox which will disable all C++ related features for the project.

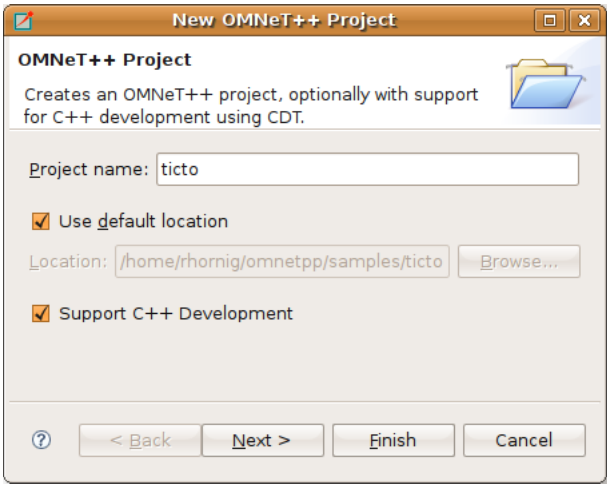


Figure 1.4. Creating a new OMNeT++ project

1.6. Getting Help

You may access the online help system from the *Help* | *Help Contents* menu item. The OMNeT++ IDE is built on top of Eclipse, so if you are not familiar with Eclipse, we recommend reading the `Workbench User Guide` and the `C/C++ Development User Guide` before starting to use OMNeT++-specific features.

Chapter 2. Editing NED Files

2.1. Overview

When you double-click a `.ned` file in the IDE, it will open in the NED editor. The new NED editor is a dual-mode editor. In the editor's graphical mode, you can edit the network using the mouse. The textual mode allows you to work directly on the NED source.

When the IDE detects errors in a NED file, the problem will be flagged with an error marker in the *Project Explorer* and the *Problems View* will be updated to show the description and location of the problem. In addition, error markers will appear in the text window or on the graphical representation of the problematic component. Opening a NED file which contains an error will open the file in text mode. Switching to graphical mode is possible only if the NED file is syntactically correct.

> As a side effect, if there are two modules with the same name and package in related projects, they will collide and both will be marked with an error. Furthermore, the name will be treated as undefined and any other modules depending on it will also generate an error (thus, a "no such module type" error may mean that there are actually multiple definitions which nullify each other).

2.2. Opening Older NED Files

The syntax of NED files has changed significantly from the 3.x version. The NED editor primarily supports the new syntax. However, it is still possible to read and display NED files with the old syntax. It is important to note that many of the advanced features (syntax highlighting, content assistance, etc.) will not work with the old syntax. There is automatic conversion from the old syntax to the new, available both from the NED editor and as an external utility program (**nedtool**).

The **gned** program from OMNeT++ 3.x viewed NED files in isolation. In contrast, the OMNeT++ IDE gathers information from all `.ned` files in all open OMNeT++ projects and makes this information available to the NED editor. This is necessary because OMNeT++ 4.x modules may inherit parameters, visual appearance or even submodules and connections from other modules, so it is only possible to display a compound module correctly if all related NED definitions are available.

2.3. Creating New NED Files

Once you have an empty OMNeT++ project, you can create new NED files. Choosing *File|New|Network Description File* from the menu will bring up a wizard where you can specify the target directory and the file/module name. You may choose to create an empty NED file, a simple/compound module, or a network. Once you press the *Finish* button, a new NED file will be created with the requested content.

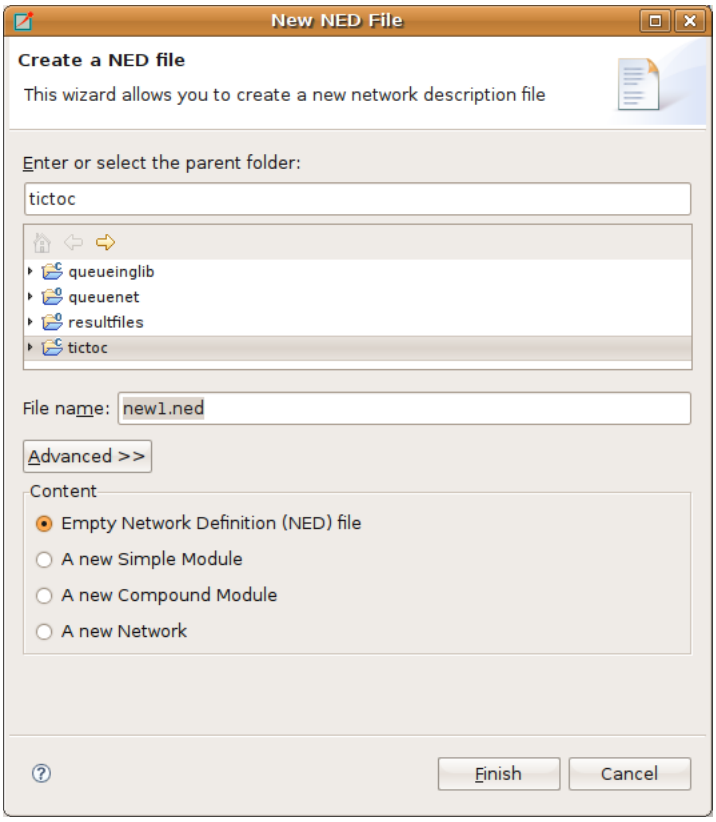


Figure 2.1. Creating a new NED file

> Make sure that the NED file and the contained module have the same name. For example, a compound module named `Wireless42` should be defined within its own `Wireless42.ned` file.

2.3.1. NED Source Folders

It is possible to specify which folders the IDE should scan for NED files and that the IDE will use as the base directory for your NED package hierarchy. The IDE will not use any NED files outside the specified NED Source Folders and those files will be opened in a standard text editor. To specify the directory where the NED files will be stored, right-click on the project in the *Project Explorer* and choose *Properties*. Select the *OMNeT++ | NED Source Folders* page and click on the folders where you store your NED files. The default value is the project root.

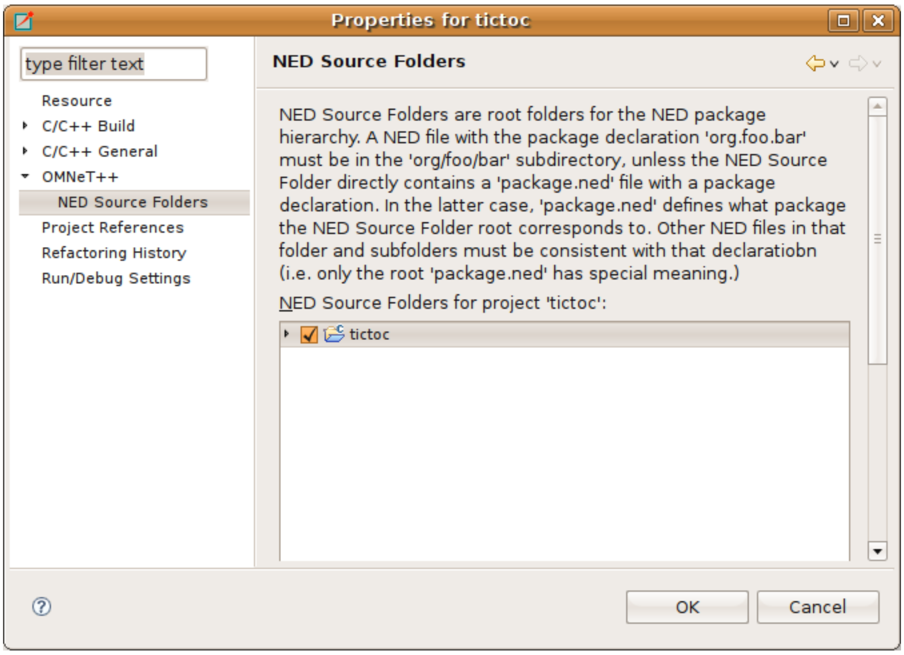


Figure 2.2. Specifying which folder will hold your NED files

2.4. Using the NED Editor

If you want to open an NED file, just double-click its icon in the *Project Explorer*. If the NED file can be parsed without an error, the graphical representation of the file will be opened; otherwise, the text view will be opened and the text will be annotated with error markers.

> Only files located in NED Source Folders will be opened with the graphical editor. If a NED file is not in the NED Source Folders, it will be opened in a standard text editor.

You can switch between graphical and source editing mode by clicking the tabs at the bottom of the editor, or by using the **Ctrl+PGUP/PGDN** key combinations. The editor will try to keep the selection during the switch. Selecting an element in a graphical view and then switching to text view will move the cursor to the related element in the NED file. When switching back to graphical view, the graphical editor will try to select the element that corresponds to the cursor location in the NED source. This allows you to keep the context, even when switching back and forth.

2.4.1. Editing in Graphical Mode

The graphical editor displays the visible elements of the loaded NED file. Simple modules, compound modules and networks are represented by figures or icons. Each NED file can contain more than one module or network. If it does, the corresponding figures will appear in the same order as they are found in the NED file.

> Place only a single module or network into an NED file, and name the file according to the module name.

Simple modules and submodules are represented as icons while compound modules and networks are displayed as rectangles where other submodules can be dropped. Connections between submodules are represented either by lines or arrows depending on whether the connection was uni- or bi-directional. Submodules can be dragged or resized using the mouse and connected by using the Connection Tool in the palette.

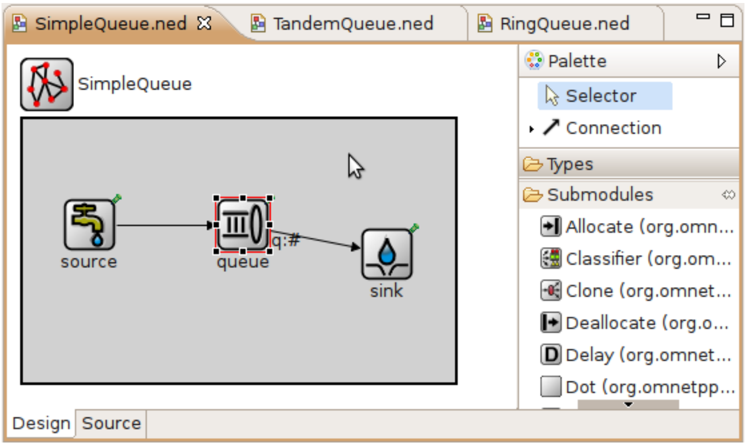


Figure 2.3. Graphical NED Editor

The palette is normally to the right of the editor area. The upper part of the palette contains the basic tools: selector, connection selector, and the connection creator tool. To use a palette item, simply click on it. Then, click in the module where you want to place/activate it. The mouse pointer will give you feedback as to whether the requested operation is allowed. The middle part of the toolbox contains the basic elements that can be placed at the top level in a NED file (simple module, compound module, interface, channel, etc.) and a "generic" submodule. Click on any of these and then click into the editor area to create an instance. The bottom part of the palette contains all module types that can be instantiated as a submodule. They are shortcuts for creating a generic submodule and then modifying its type. They will display the default icon (if any) and a short description if you hover the mouse over them. You may configure the palette by right-clicking on a button and selecting *Settings...* or filter its content by selecting *Select Packages...*

Right-clicking any element in the edited NED file will bring up a context menu that allows several actions like changing the icon, pinning/unpinning a submodule, re-layouting a compound module, or deleting/renaming the element. There are also items to activate various views. For example, the *Properties View* allows you to edit properties of the element.

Hovering over an element will display its documentation (the comment in the NED source above the definition) as a tooltip. Pressing **F2** will make the tooltip window persistent, so it can be resized and scrolled for more convenient reading.

Creating Modules

To create a module or a submodule, click on the appropriate palette item and then click where you want to place the new element. Submodules can be placed only inside compound modules or networks, while other types can be dropped directly into the editor window.

Creating and Changing Connections

Select the *connection tool* (if there are channels defined in the project, you can use the dropdown to select the connection channel type). First, click the source module and then, the destination. A popup menu will appear, asking which gates should be connected on the two selected modules. The tool will offer only valid connections (e.g. it will not offer to connect two output gates).

Reconnecting Modules

Clicking and dragging a connection end point to another module will reconnect it (optionally, asking which gate should be connected). If you want to change only the gate, drag the connection end point and drop it over the original module. A popup will appear asking for the source or destination gate.

Selecting Elements

Selecting an element is done by clicking on it or by dragging a rectangle over the target modules. A compound module can be selected by clicking on its border or title. If you want to select only connections within a selection rectangle, use the *connection selector* tool in the dropdown menu of the *connection tool*. The **CTRL** and **SHIFT** key can be used to add/remove to/from the current selection. Note that the keyboard (arrow keys) can also be used to navigate between submodules. You can also select using a selection rectangle by dragging the mouse around the modules.

Undo, Redo, Deleting Elements

Use **Ctrl+Z** and **Ctrl+Y** for undo and redo respectively, and the DEL key for deletion. These functions are also available in the *Edit* menu and in the context menu of the selected element.

Moving and Resizing Elements

You can move/resize the selected elements with the mouse. Holding down **SHIFT** during move will perform a constrained (horizontal, diagonal or vertical) move operation. **CTRL** + resize will resize around the object's center. **SHIFT** + resize will keep the aspect ratio of the element.

If you turn on *Snap to Geometry* in the *View* menu, helper lines will appear to help you align with other modules. Selecting more than one submodule activates the *Alignment* menu (found both in the *View* menu and in the context menu).

Copying Elements

Holding down **CTRL** while dragging will clone the module(s). Copy/Paste can also be used both on single modules and with group selection.

Zooming

Zooming in and out is possible from the *View* menu, or using **Ctrl+-** , **Ctrl+=,** or holding down **CTRL** and using the mouse wheel.

Pinning, Unpinning, Re-Layouting

A submodule display string may or may not contain explicit coordinates for the submodule; if it does not, then the location of the submodule will be determined by the layouting algorithm. A submodule with explicit coordinates is pinned; one without is unpinned. The Pin action inserts the current coordinates into the display string and the Unpin action removes them. Moving a submodule also automatically pins it. The position of an unpinned module is undetermined and may change every time the layouting algorithm runs. For convenience, the layouter does not run when a submodule gets unpinned (so that the submodule does not jump away on unpinning), but this also means that unpinned submodules may appear at different locations next time the same NED file is opened.

Choosing an Icon for a Module

To choose an icon for a module, right-click on it and select the *Choose an Icon...* menu item from the context menu or select the module and modify the icon property in the *Properties View*.

Navigation

Double-clicking a submodule will open the corresponding module type in a NED editor. Selecting an element in the graphical editor and then switching to text mode will place the cursor near the previously selected element in the text editor.

Navigating inside a longer NED file is easier if you open the *Outline View* to see the structure of the file. Selecting an element in the outline will select the same element in the graphical editor.

If you want to see the selected element in a different view, select the element and right-click on it. Choose *Show In* from the context menu, and select the desired view.

Opening a NED Type

If you know only the name of a module type or other NED element, you can use the *Open NED Type* dialog by pressing **Ctrl+SHIFT+N**. Type the name, or search with wildcards. The requested type will be opened in an editor. This feature is not tied to the graphical editor: the Open NED Type dialog is available from anywhere in the IDE.

Setting Properties

Elements of the display string and other properties associated with the selected elements can be edited in the *Properties View*. The Property View is grouped and hierarchically organized; however, you can switch off this behavior on the view toolbar. Most properties can be edited directly in the *Properties View*, but some also have specific editors that can be activated by pressing the ellipsis button at the end of the field. Fields marked with a small light bulb support content assist. Use the **CTRL+SPACE** key combination to get a list of possible values.

> NOTE
>
> The following functions are available only in source editing mode:
>
> - Creating or modifying gates
> - Creating or modifying inner types in compound modules
> - Creating grouped and conditional connections
> - Adding or editing properties
> - Adding or editing imports
> - Creating submodule vectors
> - Editing submodule vector size

2.4.2. Editing in Source Mode

The NED source editor supports all functionality that one can expect from an Eclipse-based text editor, such as syntax highlighting (for the new NED syntax), clipboard cut/copy/paste and unlimited undo/redo. Here is a summary of these features:

- Undo (**Ctrl+Z**), Redo (**Ctrl+Y**)
- Indenting/unindenting code blocks with **TAB/Shift+TAB**
- Correct indentation (NED syntax aware) (**Ctrl+I**)
- Find (**Ctrl+F**), incremental search (**Ctrl+J**)
- Moving lines with **ALT+UP/DOWN**

- Folding regions (NED syntax aware)

> NOTE: **Ctrl+Shift+L** pops up a window with all keyboard bindings listed.

The NED source is continually parsed as you type, and errors and warnings are displayed as markers on the editor rulers. At times when the NED text is syntactically correct, the editor has full knowledge of "what is what" in the text buffer.

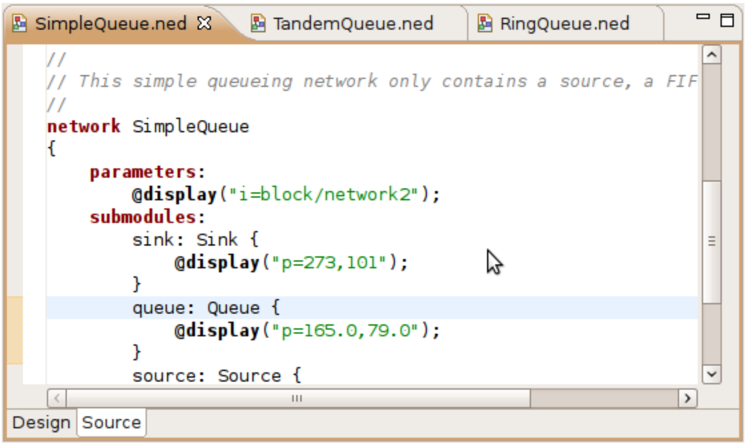


Figure 2.4. NED Source Editor

Hovering the mouse over a NED type name will display the documentation in a "tooltip" window, which can be made persistent by hitting **F2**.

Converting to the New NED Syntax

If you have an NED file with older syntax, you can still open it. A context menu item allows you to convert it to the new syntax. If the NED file is already using the new syntax, the *Convert to 4.x Format* menu item is disabled.

Content Assist

If you need help, just press **Ctrl+SPACE**. The editor will offer possible words or templates. This is context sensitive, so it will offer only valid suggestions. Content assist is also a good way of exploring the new NED syntax and features.

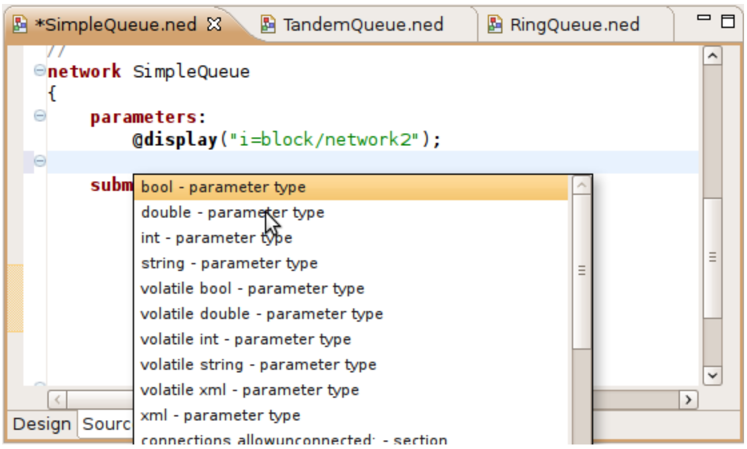


Figure 2.5. NED Source Editor with content assist activated

Searching in NED Files

Selecting a text or moving the cursor over a word and pressing **CTRL+SHIFT+G** searches for the selection in all NED files in the open projects. This function lets you quickly find references to the word or type currently under the cursor. The results are shown in the standard *Search View*.

Organizing Imports

Sometimes, it is very inconvenient to add the necessary import statements to the beginning of the NED file by hand. The IDE can do it for you (almost) automatically. Pressing **CTRL+SHIFT+O** will cause the IDE to try to insert all necessary import statements. You will be prompted to specify the used packages in case of ambiguity.

Cleaning Up NED Files

This function does a general repair on all selected NED files by throwing out or adding import statements as needed, checking (and fixing) the file's package declaration, and reformatting the source code. It can be activated by clicking on the *Project | Clean Up NED Files* menu item from the main menu.

Formatting the Source Code

It is possible to reformat the whole NED file according to the recommended coding guidelines by activating the *Format Source* context menu item or by pressing the **CTRL+SHIFT+F** key combination.

> NOTE: Using the graphical editor and switching to source mode automatically re-formats the NED source code, as well.

Navigation

Holding the **CTRL** key and clicking on an URL or on a module type will jump to the target page or type definition.

If you switch to graphical mode from text mode, the editor will try to locate the NED element under the cursor and select it in the graphical editor.

The Eclipse platform's bookmarking and navigation history facilities also work in the NED editor.

2.5. Associated Views

There are several views related to the NED editor. These views can be displayed (if not already open) by choosing *Window | Show View* in the menu or by selecting a NED element in the graphical editor and selecting *Show In* from the context menu.

2.5.1. Outline View

The *Outline View* allows an overview of the current NED file. Clicking on an element will select the corresponding element in the text or graphical view. It has limited editing functionality; you can copy/cut/paste and delete an object.

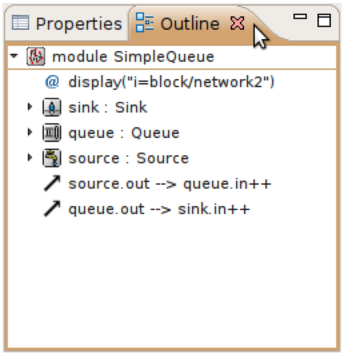


Figure 2.6. Outline View

2.5.2. Property View

The *Property View* contains all properties of the selected graphical element. Visual appearance, name, type and other properties can be changed in this view. Some fields have specialized editors that can be activated by clicking on the ellipsis button in the field editor. Fields marked with a small light bulb icon have content assist support. Pressing **CTRL+SPACE** will display the possible values the field can hold.

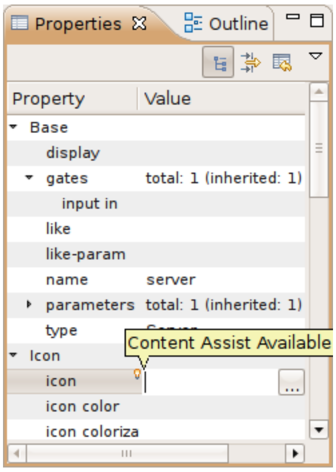


Figure 2.7. Properties View

2.5.3. Palette View

The Palette is normally displayed on the left or right side of the editor area and contains tools to create various NED elements. It is possible to hide the Palette by clicking on the little arrow in the corner. You can also detach it from the editor and display it as a normal Eclipse View (*Window* | *Show View* | *Other...* | *General* | *Palette*).

2.5.4. Problems View

The *Problems View* contains error and warning messages generated by the parser. Double-clicking a line will open the problematic file and move to the appropriate marker.

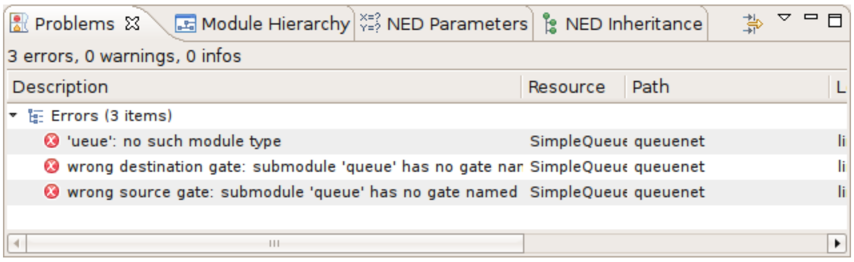


Figure 2.8. Problems View

2.5.5. NED Inheritance View

The *Inheritance View* displays the relationship between different NED types. Select a NED element in the graphical editor or move the cursor into a NED definition and the *Inheritance View* will display the ancestors of this type. If you do not want the view to follow the selection in the editor, click the Pin icon on the view toolbar. This will fix the displayed type to the currently selected one.

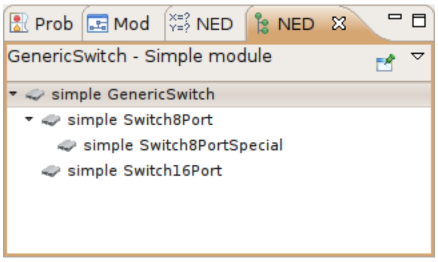


Figure 2.9. NED Inheritance View

2.5.6. Module Hierarchy View

The *Module Hierarchy View* shows the contained submodules and their parameters, several levels deep. It also displays the parameters and other contained features.

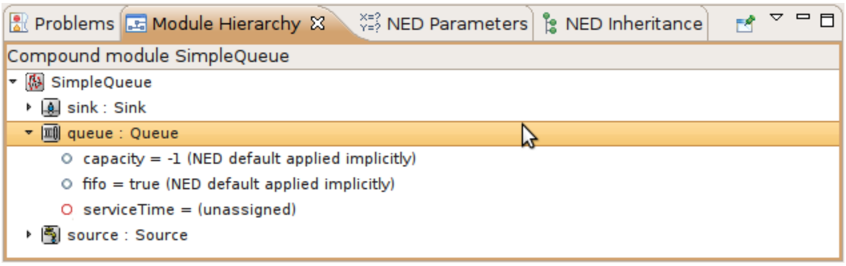


Figure 2.10. Module Hierarchy View

2.5.7. Parameters View

The *Parameters View* shows the parameters of the selected module including inherited parameters.

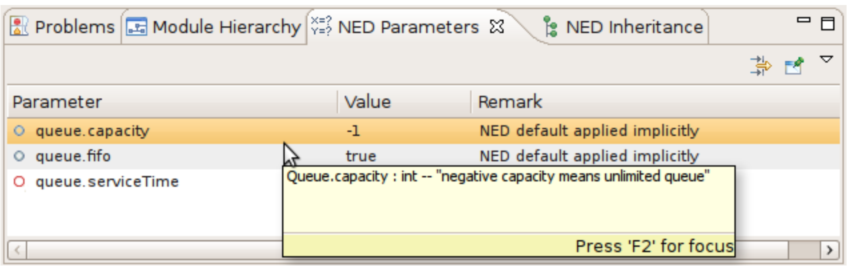


Figure 2.11. Outline View

The latter two views are used mainly with the INI File Editor.

Chapter 3. Editing INI Files

3.1. Overview

In OMNeT++, simulation models are parameterized and configured for execution using configuration files with the `.ini` extension, called INI files. Ini files are text files, which can be edited using any text editor. However, OMNeT++ 4.x introduces a tool expressly designed for editing INI files. The INI File Editor is part of the OMNeT++ IDE and is very effective in assisting the user to author INI files. It is a very useful feature because it has detailed knowledge of the simulation model, the INI file syntax, and the available configuration options.

> NOTE
> The syntax and features of INI files have changed since OMNeT++ 3.x. These changes are summarized in the "Configuring Simulations" chapter of the "OMNeT++ 4.x. User Manual".

The INI File Editor is a dual-mode editor. The configuration can be edited using forms and dialogs, or as plain text. Forms are organized around topics like general setup, Cmdenv, Tkenv, output files, extensions and so on. The text editor provides syntax highlighting and auto completion. Several views can display information, which is useful when editing INI files. For example you can see the errors in the current INI file or all the available module parameters in one view. You can easily navigate from the module parameters to their declaration in the NED file.

3.2. Creating INI Files

To create a new INI file, choose *File* | *New* | *Initialization File* from the menu. It opens a wizard where you can enter the name of the new file and select the name of the network to be configured.

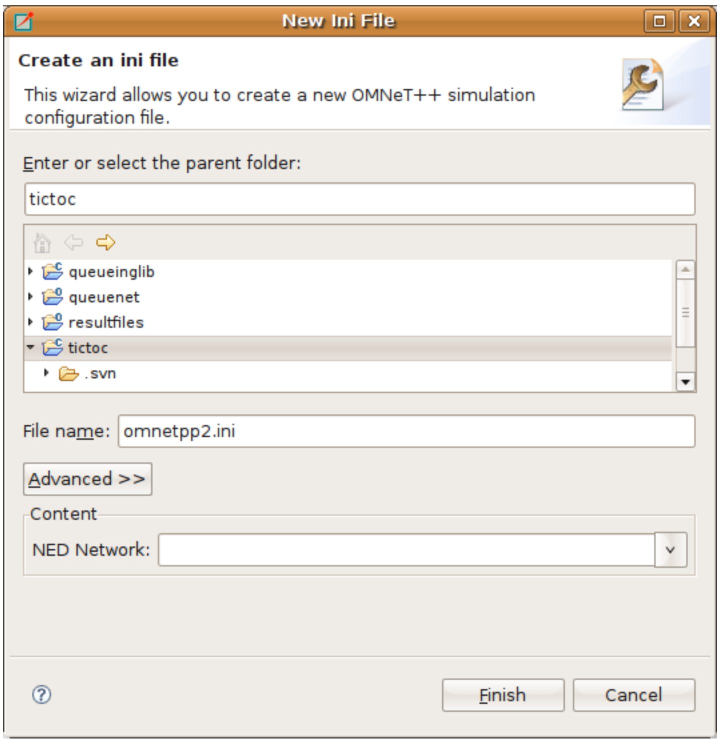


Figure 3.1. New Initialization File dialog

3.3. Using the INI File Editor

The INI File Editor has two modes. The *Source* mode provides a text editor with syntax highlighting and auto completion of names. In the *Form* mode, you can edit the configuration by entering the values in a form. You can switch between the modes by selecting the tabs at the bottom of the editor.

3.3.1. Editing in Form Mode

The INI file contains the configuration of simulation runs. The content of the INI file is divided into sections. In the simplest case, all parameters are set in the General section. If you want to create several configurations in the same INI file, you can create named Configuration (Config) sections and refer to them with the `-c` option when starting the simulation. The Config sections inherit the settings from the General section or from other Config sections. This way you can factor out the common settings into a "base" configuration.

On the first page of the form editor, you can edit the sections. The sections are displayed as a tree; the nodes inherit settings from their parents. The icon before the section name shows how many runs are configured in that section (see Table 3.1, “Legend of Icons Before Sections”). You can use drag and drop to reorganize the sections. You can delete, edit, or add a new child to the selected section.

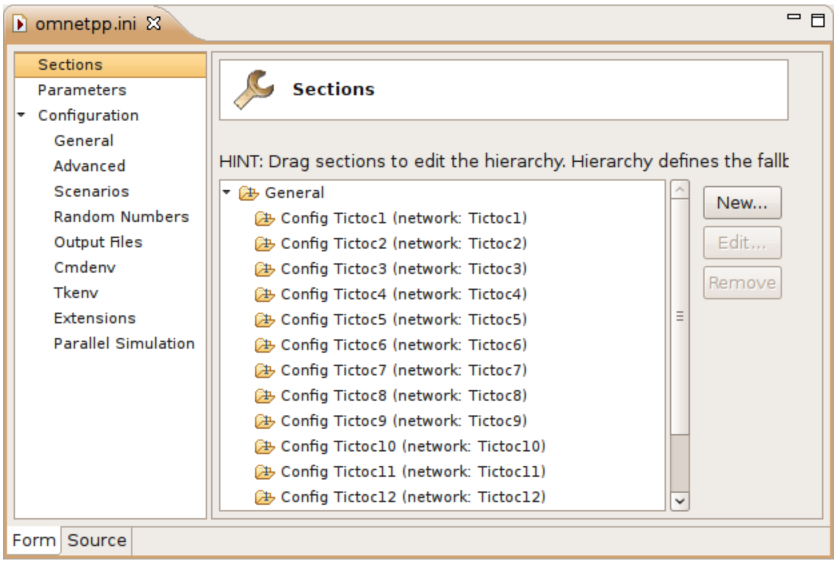


Figure 3.2. Editing INI file sections

	contains a single run
	contains multiple replications (specified by 'repeat=...')
	contains iteration variables
	contains multiple replications for each iteration

Table 3.1. Legend of Icons Before Sections

The Config sections have a name and an optional description. You can specify a fallback section other than General. If the network name is not inherited, it can be specified, as well.

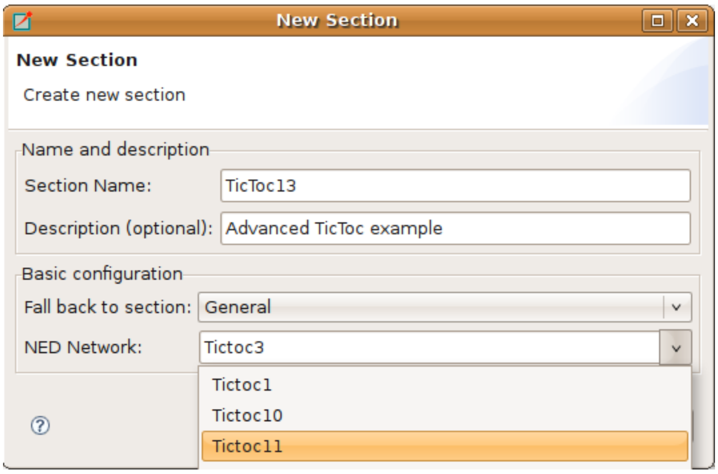


Figure 3.3. Creating a new INI file section

On the *Parameters* page of the form editor, you can set module parameters. First, you have to select the section where the parameters are stored. After selecting the section from the list, the form shows the name of the edited network and the fallback section. The table below the list box shows current settings of the section and all other sections from which it has inherited settings. You can move parameters by dragging them. If you click a table cell, you can edit the parameter name (or pattern), its value and the comment attached to it. **Ctrl+Space** brings up a content assist. If you hover over a table row, the parameter is described in the tooltip that appears.

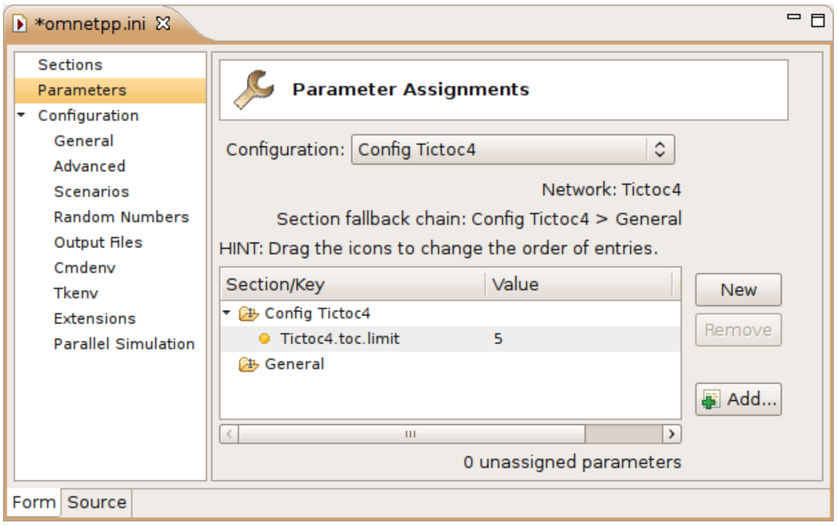


Figure 3.4. Editing module parameters

New parameters can be added one by one by pressing the *New* button and filling the new table row. The selected parameters can be removed with the *Remove* button. If you press the *Add...* button, you can add any missing parameters.

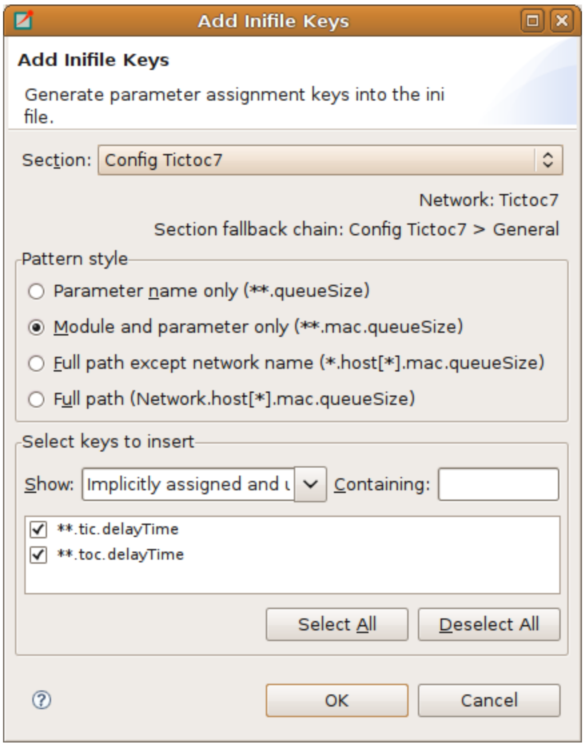


Figure 3.5. Add missing module parameters dialog

The rest of the settings do not belong to modules (e.g. configuration of random number generators, output vectors, simulation time limit). These settings can be edited from the forms listed under the Configuration node. If the field has a default value and it is not set, the default value is displayed in gray. If its value is set, you can reset the default value by pressing the *Reset* button. These fields are usually set in the General section. If you want to specify them in a Config section, press the button and add a section-specific value to the opening table. If the table contains the Generic section only, then it can be collapsed again by pressing the button. Some fields can be specified in the General section only, so they do not have a button next to them.

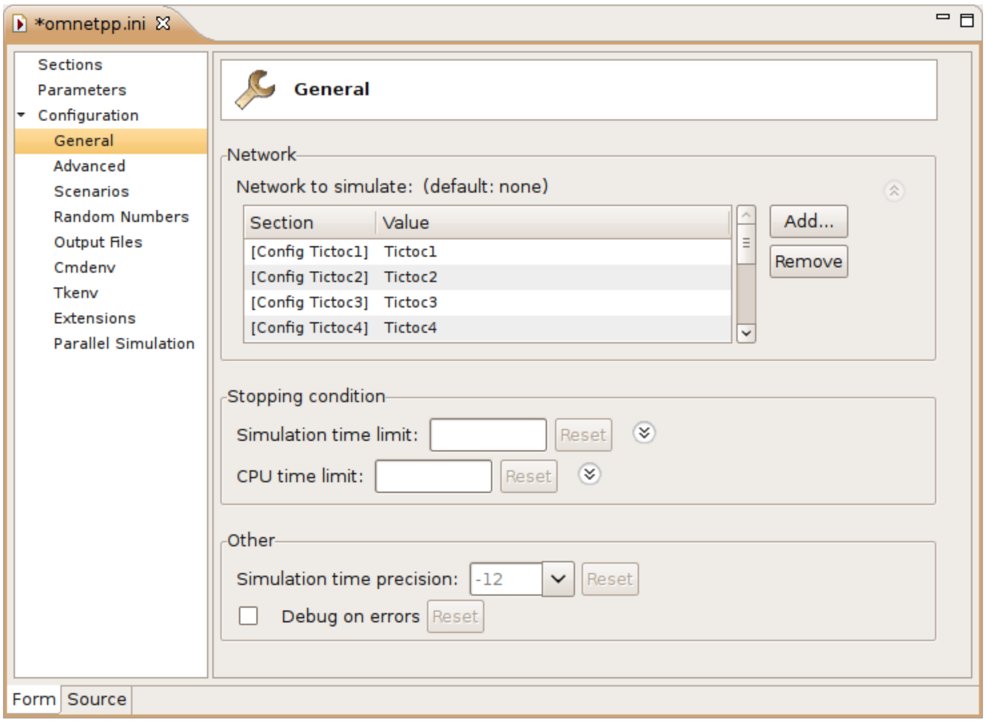


Figure 3.6. Editing general configuration

3.3.2. Editing in Text Mode

If you want to edit the INI file as plain text, switch to the Source mode. The editor provides several features in addition to the usual text editor functions like copy/paste, undo/redo and text search.

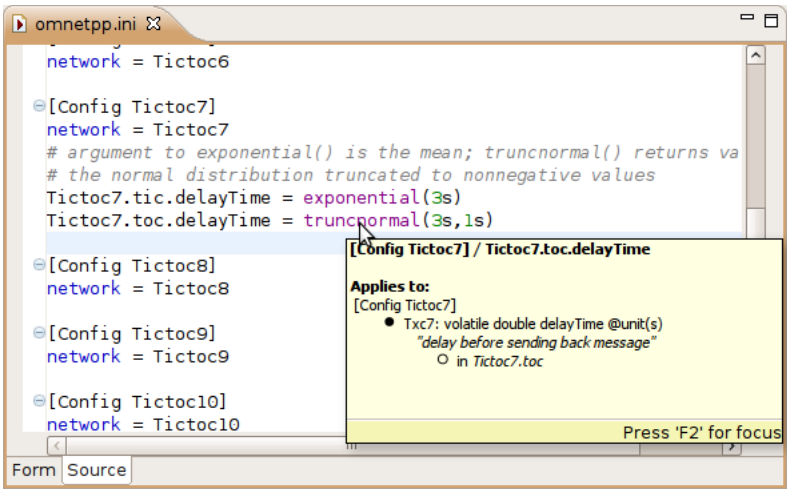


Figure 3.7. Editing the INI file in text mode

Opening Old INI Files

When you open an INI file with the old format, the editor offers to convert it to the new format. It creates Config sections from Run sections and renames old parameters.

Content Assist

If you press **Ctrl+Space**, you will get a list of proposals valid at the insertion point. The list may contain section names, general options, and parameter names and values of the modules of the configured network.

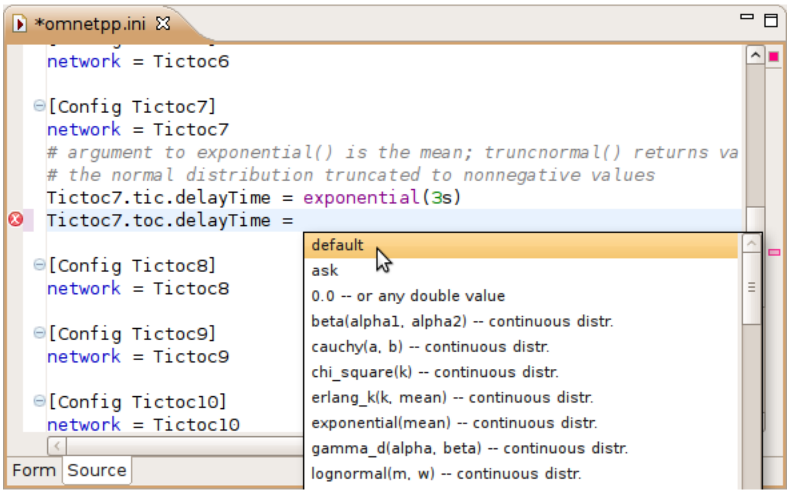


Figure 3.8. Content assist in source mode

Tooltip

If you hover over a section or parameter, a tooltip appears showing the properties of the section or parameter. The tooltip for sections displays the inheritance chain, the network name, number of errors and warnings and the yet unassigned parameters. For parameters, the definition, description and the module name are displayed.

Add Unassigned Parameters

You can add the names of unassigned module parameters to a Config section by choosing *Add Missing keys...* from the context menu.

Comments

To comment out the selected lines, press **Ctrl**+/. To remove the comment, press **Ctrl**+/ again.

Navigation

If you press the **Ctrl** key and click on a module parameter name, then the declaration of the parameter will be shown in the NED editor. You can navigate from a network name to its definition, too.

Error Markers

Errors are marked on the left/right side of the editor. You can move to the next/previous error by pressing **Ctrl**+**.** and **Ctrl**+**,** respectively. You can get the error message in a tooltip if you hover over the marker.

3.4. Associated Views

There are several views related to the INI editor. These views can be displayed (if not already open) by choosing the view from the *Window* | *Show View* submenu.

3.4.1. Outline View

The *Outline View* allows an overview of the sections in the current INI file. Clicking on a section will highlight the corresponding element in the text or form view.

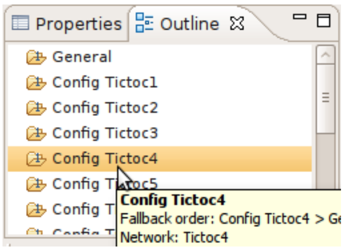


Figure 3.9. Outline View showing the content of an INI file

3.4.2. Problems View

The *Problems View* contains error and warning messages generated by the parser. Double-clicking on a row will open the problematic file and move to the location of the problem.

3.4.3. Parameters View

The *Parameters View* shows parameters of the selected section including inherited parameters. It also displays the parameters that are unassigned in the configuration. When the toggle button on the toolbar is on, then all parameters are displayed; otherwise, only the unassigned ones are visible.

If you want to fix the content of the view, press the button. After pinning, the content of this view will not follow the selection made by the user in other editors or views.

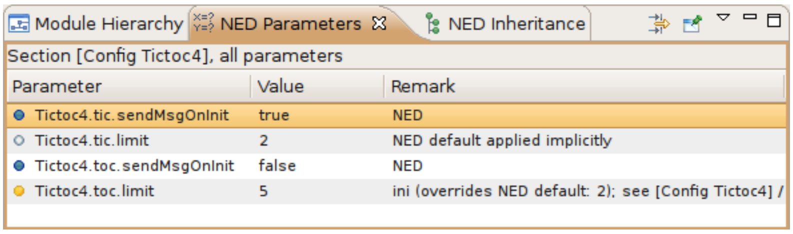


Figure 3.10. Parameters View

	value is set in the NED file
	default from the NED file is explicitly set in the INI file (`**.paramname=default`)
	default from the NED file is automatically applied, because no value is specified in the INI file
	value is set in the INI file (may override the value from the NED file)
	value is set in the INI file to the same value as the NED default
	will ask the user at runtime (`**.paramname=ask`)
	unassigned -- has no values specified in the NED or INI files

Table 3.2. Legend of icons before module parameters

Right-clicking on any line will show a context menu that allows you to navigate to the definition of that parameter or module.

3.4.4. Module Hierarchy View

The *Module Hierarchy View* shows the contained submodules, several levels deep. It also display the module parameters, and where its value comes from (INI file, NED file or unassigned).

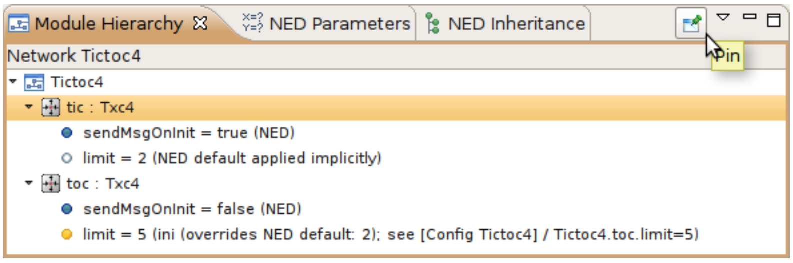


Figure 3.11. Module Hierarchy View

> Before you use the context menu to navigate to the NED definition, pin down the hierarchy view. This way you will not lose the current context and content if the view will not follow the selection.

3.4.5. NED Inheritance View

The *NED Inheritance View* shows the inheritance tree of the network configured in the selected section.

Chapter 4. Editing Message Files

4.1. Creating Message Files

Choosing *File|New|Message Definition (msg)* from the menu will bring up a wizard where you can specify the target directory and the file name for your message definition. You may choose to create an empty MSG file, or choose from the predefined templates. Once you press the *Finish* button, a new MSG file will be created with the requested content.

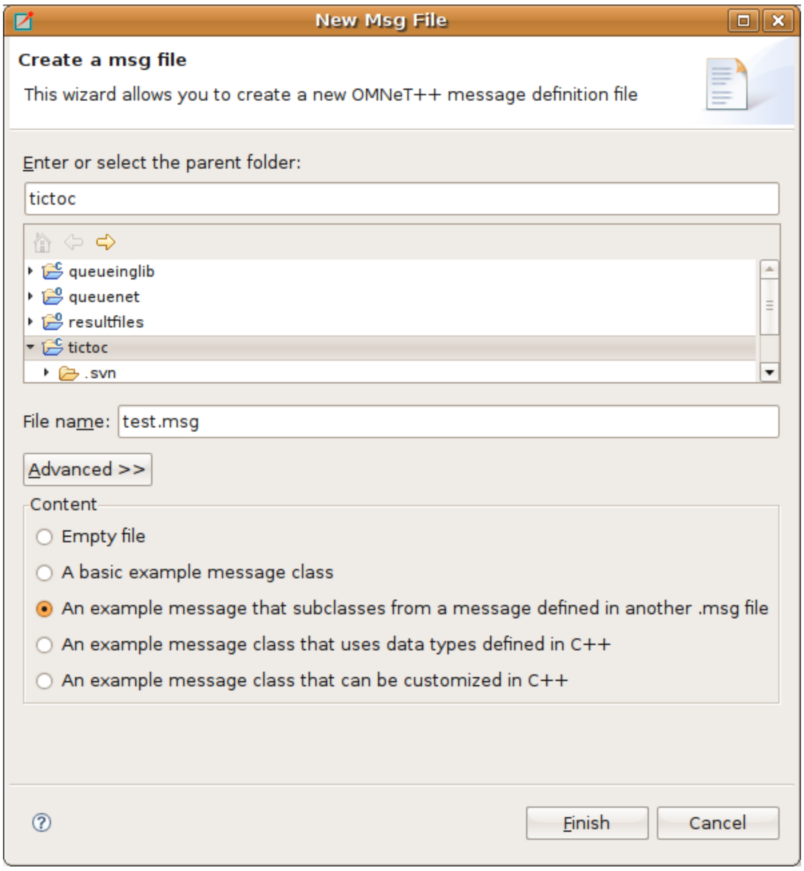


Figure 4.1. Creating a new MSG file

4.2. The Message File Editor

The message file editor is a basic text editor with syntax highlight support.

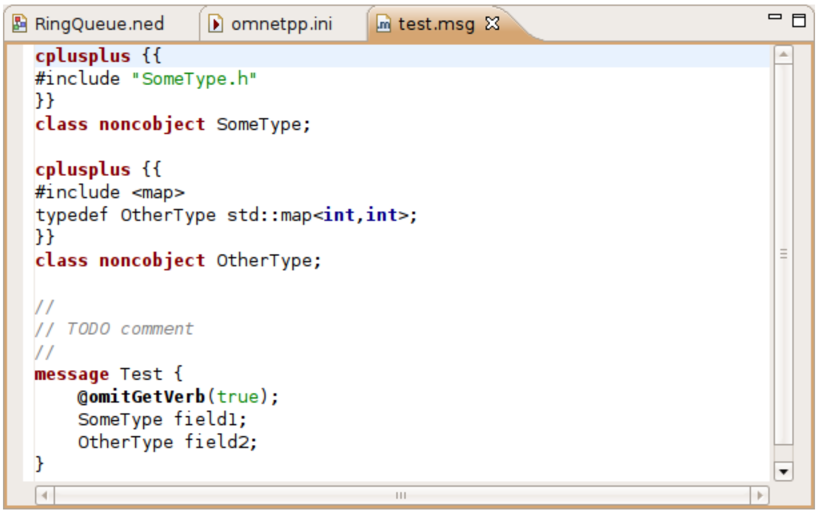


Figure 4.2. Message File Editor

> **Note:** Currently the editor does not support advanced features like content assistance or syntax aware folding.

Chapter 5. C++ Development

5.1. Introduction

The OMNeT++ IDE contains CDT (C/C++ Development Tooling) to help you develop the C++ part of your simulation. We recommend reading the CDT documentation in the IDE help system (Help/Help Content) to get a better general understanding of how CDT works. The OMNeT++ IDE adds several extensions to the standard CDT features for ease of use:

- Customized project creation dialog
- Extended C++ toolchain for OMNeT++ (including MSG compiler)
- Automatic makefile generation
- OMNeT++-specific build configuration
- Special launch configuration type for launching and debugging OMNeT++ Simulations

CDT already supports "standard make" builds. It can invoke "make" and when you select "Build" in the IDE, we generate a makefile (via a built-in Java version of opp_makemake) and then let CDT do a "standard make" build.

> NOTE: The OMNeT++ IDE does not use the "managed make" feature of CDT.

Eclipse, CDT and the OMNeT++ IDE uses several files in the project to store settings. The following files should be present in the project root directory (these files are normally hidden by the IDE in the *Project Explorer View*):

- `.project` : Eclipse stores the general project properties in this file including project name, dependencies from other projects, and project type (i.e. whether OMNeT++-specific features are supported at all).
- `.cproject` : This file contains settings specific to C++ development including toolchain definition, configuration templates (i.e. debug / release), error parsers, and debugger settings.
- `.oppbuildspec` : Contains all settings specific to OMNeT++. This file contains the settings passed to the makefile generator. You can configure all OMNeT++-specific settings in the *Project Properties* dialog on the *OMNeT++/Makemakepage*. Everything you set on this page will be stored in the `.oppbuildspec` file.

> NOTE: If you are creating a project where no C++ support is needed (i.e. you are using an existing precompiled simulation library and you edit only NED and Ini files), the `.cproject` and `.oppbuildspec` files will not be present in your project.

5.2. Prerequisites

The OMNeT++ IDE (and the OMNeT++ simulation framework itself) requires a preinstalled compiler toolchain to function properly.

- On Windows: The OMNeT++ distribution comes with a preconfigured MinGW compiler toolchain. You do not have to install or download anything. The IDE should be able to use the MinGW compiler without any problem.

- On Linux: The default GCC toolchain can be used. OMNeT++ 4.1 was tested on GCC 4.4, although other versions should work, too. You have to install the GCC toolchain on your distribution before trying to compile a simulation with OMNeT++.
- On Mac OS X: Version 10.5 (i386) is supported. You should install XCode 3.x tools to get compiler support before trying to compile a simulation with OMNeT++.

5.3. Creating a C++ Project

To create an OMNeT++ project that supports C++ development, select *File* | *New* | *OMNeT++ Project*.

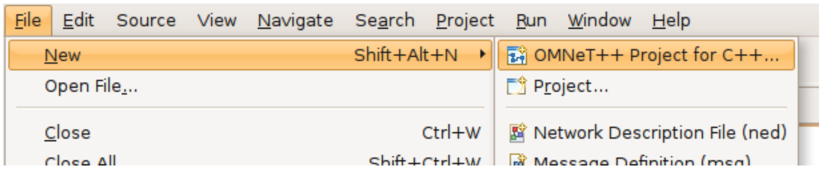


Figure 5.1. Creating an OMNeT++ project

This will show you the *New OMNeT++ Project* wizard that lets you create a project that supports NED, MSG and INI file editing as well as C++ development for simple modules.

On the first page of the wizard, you should specify the project name and ensure that the *Support C++ Development* checkbox is selected.

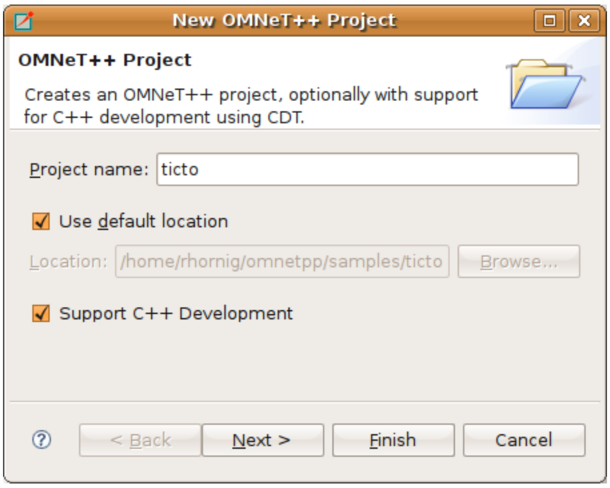


Figure 5.2. Setting project name and enabling C++ support

Select a project template.

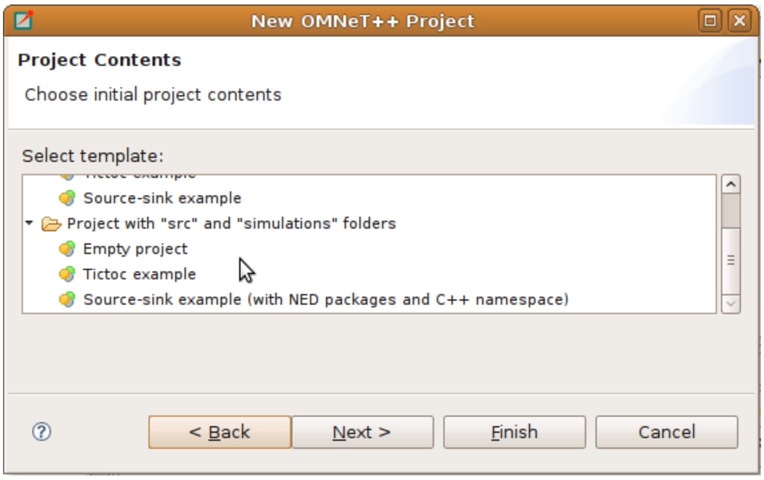


Figure 5.3. Selecting a project template

Select a toolchain which is supported on your platform. Usually you will see only a single toolchain supported, so you do not need to select anything.

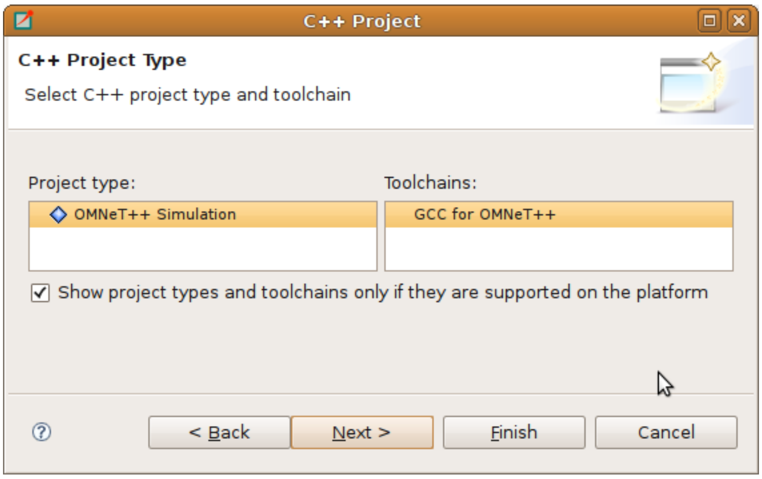


Figure 5.4. Selecting a toolchain

Finally, choose from the preset configurations. By default, a debug and release configuration can be created. A configuration is a set of options that are associated with the build process. It is used mainly to create debug and release builds. Pressing the *Finish* button will create the project.

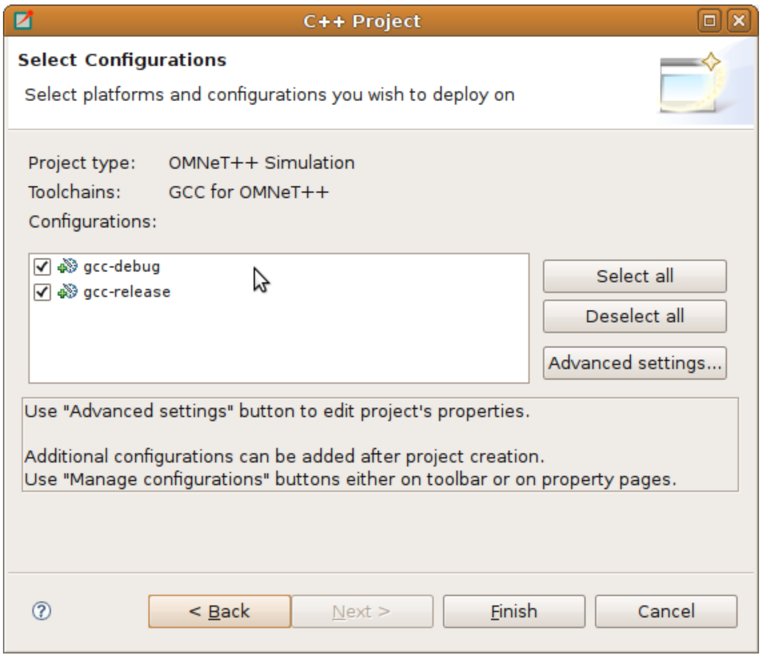


Figure 5.5. Selecting configurations

5.4. Configuring the Project

The OMNeT++ IDE adds several extensions to the standard CDT to make the building of simulations easier. The project properties dialog is extended with a special page - *OMNeT++ | Makemake* - where you can configure the options used for automatically generating the makefile(s). A separate makefile will be created according to the settings you specify in the dialog. On the *C/C++ Build* page you can specify which makefile will be called by the IDE build process when you initiate a Build action. By default, the Build directory is specified as `${ProjDirPath}`, which is the root directory of your project. You can set any directory for the build directory, but keep in mind that it should contain an automatically generated makefile or you will have to create a makefile manually there (not recommended). If you have several source directories, make sure that the primary makefile calls the sub-makefiles correctly.

> The build configuration is done on the *Makemake* page. Usually, you do not have to change anything on other C++ related property pages.

> Use the project templates in the *New OMNeT++ Project* wizard to start a new project. The templates are especially helpful if you have a project where you need multiple source trees that have their own makefiles.

You can open the property dialog by right-clicking on a folder or project and selecting the *Properties* menu. Choose the *OMNeT++ | Makemake* page.

Before you start modifying the settings, make sure that you select the folder you want to configure. If you want to convert the current folder to a source folder, simply click on *Source Location*. It is also possible to selectively include or exclude a whole directory tree if it is already inside a source tree.

> Creating a source folder here will affect all configurations.

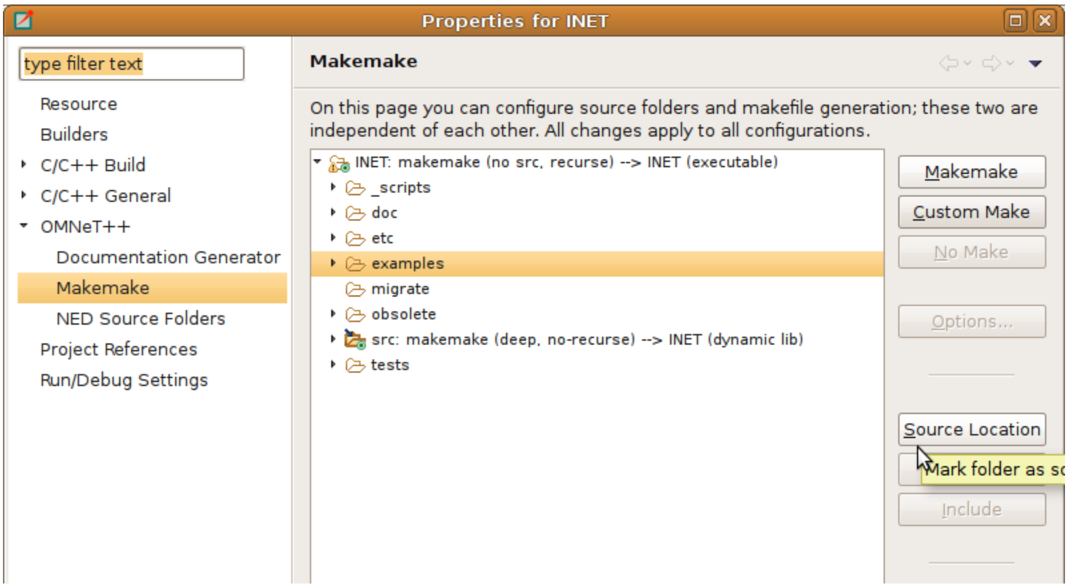


Figure 5.6. Setting a source folder

> NOTE
>
> To regenerate your makefile on the command line, you can export the settings by pressing the *Export* button. This action will create a file with the name `makemakefiles`. After exporting, execute `make -f makemakefiles` from the command line.

Press the *Makemake* button to turn on the automatic makefile generation in the selected folder. Manually written makefiles can be used by selecting *Custom Make*. This expects that a makefile will be already present in the selected directory (it must be called `Makefile`).

To configure how the makefile will be generated, press the *Options* button.

On the first tab of the options dialog, you can specify how the final target of the makefile is created.

- The target type can be either executable or a shared/static library. Libraries can be exported to be used by other dependent projects that can link to them automatically. It is possible to compile all the sources without linking them.
- You may set the target name. The default value is derived from the project name.
- The output directory can specify where the object files and the final target will be created (relative to the project root)

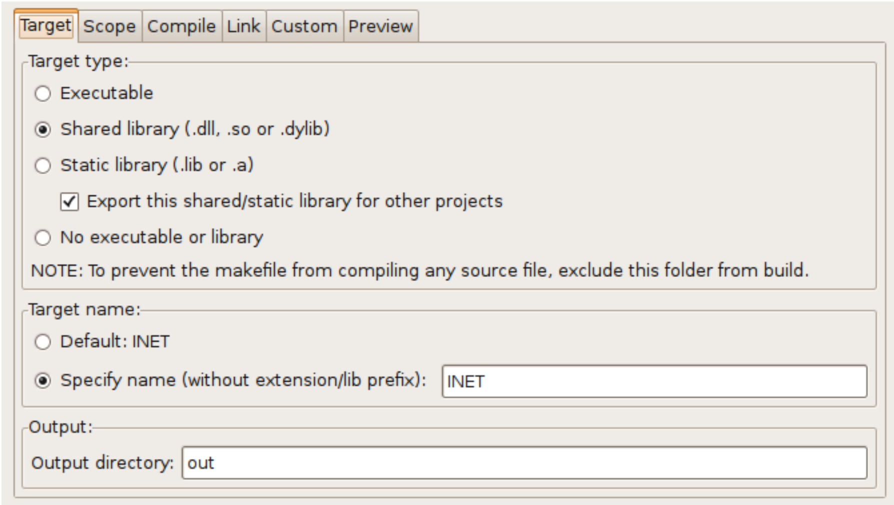


Figure 5.7. Target definition

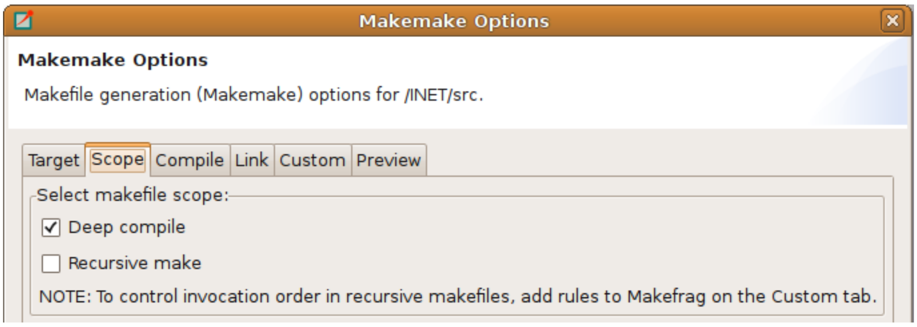


Figure 5.8. Scope of makefile

The *Scope* tab allows you to configure the scope of the makefile and what source files will be included.

- Deep - will use all source files recursively in all subdirectories.
- Recursive - will use source files only in the current directory. Invokes make in all subdirectories (i.e. makefiles must exist in those directories).

If you want to invoke additional makefiles from this makefile, click on the *More >>* button and specify which directories should be visited (relative to this makefile).

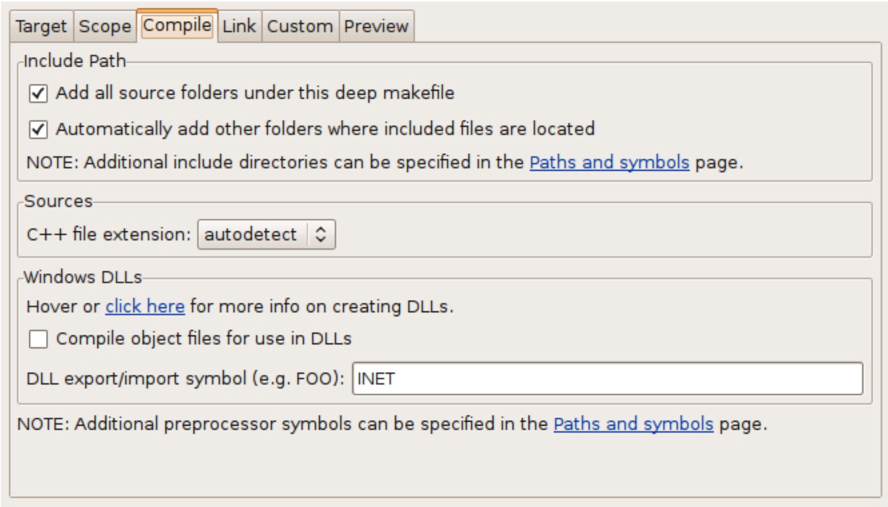


Figure 5.9. Compiler options

The *Compile* tab allows you to adjust the parameters passed to the compiler during the build process. You can specify where the system should look for input files. You can scan all subfolders in the source directory or even allow scanning of all directories in the project (and all other projects it depends on).

As a rule of thumb, the include file names must be unique (i.e. include files with the same name in different directories are not allowed) otherwise the builder cannot determine the correct location of the file.

Include files can be qualified with the parent directories to avoid this problem. If you have two include files with the same name (e.g. `inc.h`) in different directories, add the directory names before the filenames to make the include statements unique. Use `#include "dir1/inc.h"` and `#include "dir2/inc.h"` in your C++ files. If the problematic file is located in the project root, use the form `#include "../projectname/inc.h"` to make it unique.

If needed, you can set the source file extensions (`.cc` or `.cpp`). In most cases, auto-detection works correctly, so you do not have to change this parameter.

Link options allows the user to fine-tune the linking steps at the end of the build process.

- Link with libraries exported from referenced projects - if you are depending on an other project's target (i.e. static or dynamic library), you can instruct the linker to automatically link with those libraries. You must explicitly set your project dependencies (See Dependent Projects section).
- Additional object files and libraries can be added by clicking on the *More>>* button.

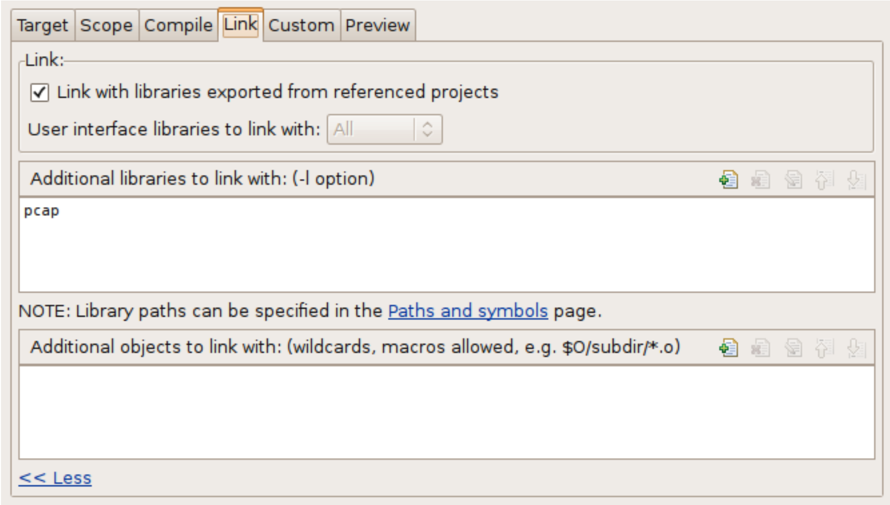


Figure 5.10. Linker options

The *Custom* tab allows the customization of the makefiles by inserting handwritten makefile fragments inside the automatically generated makefile. It is possible to add additional variables, targets, and rules. The fragment code entered here will be written into the Makefrag file.

The *Preview* tab displays the final command line with all options that will be passed to `opp_makemake` to generate the makefile.

5.5. Dependent Projects

If your project uses code from other projects (i.e. you are using a library provided by an other project like the INET Framework or the queuing library provided in the samples directory), you should make the project dependent on that code. This step ensures that your code is linked with the dependent project's output and that the NED path can be correctly set. Include files are also automatically used from dependent projects.

To set the project dependence, right-click on your project and select *Properties*. In the *Properties* dialog, select the *Project References* page and click on the projects on which you are depending. If those projects provide shared or static libraries as their output, the linker will automatically include those libraries during link phase. It is also possible to use executable files from other projects. In this case, your project should not have any code so that compilation or linking is unnecessary. Your project should contain only INI and NED files and use the prebuilt simulation executable from the other projects.

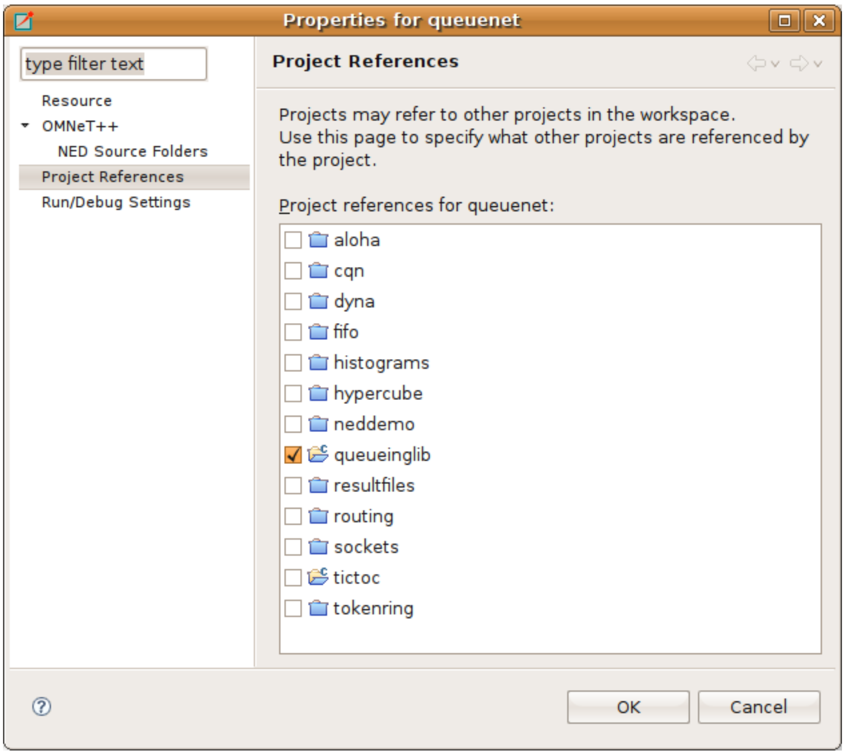


Figure 5.11. Setting project dependencies

> **NOTE**
>
> To see an example of how project dependency works, check the queuenet and queueinglib examples in the samples' directory. Queuinglib provides a prebuilt executable file containing all simple modules while queuenet contains only network definitions and initialization files as samples.

5.6. Editing C++ Code

The OMNeT++ IDE comes with a C/C++ editor provided by the CDT component. In addition to the standard editor features provided by the Eclipse environment, the C/C++ editor provides syntax highlighting and content assistance on the source code. Use **CTRL+SPACE** to activate the content assist window anywhere in your source code. An other useful key combination is **CTRL+TAB**, which switches between your C++ and header file. Press **CTRL+SHIFT+L** to get a list of currently active key bindings. For further information about editing C++ files, please visit the CDT section in the online help system.

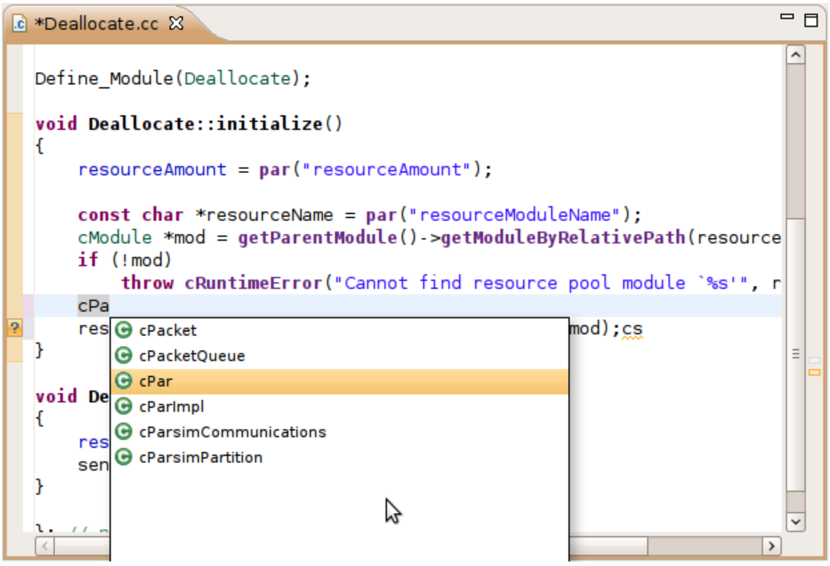


Figure 5.12. C++ source editor

Content assistance works only if the IDE can scan your header files in the background.

5.7. Building the Project

Once you have created your source files and configured your project settings, you can build your executable. Before doing this, check if the active build configuration is correctly set for your project. Select *Build Configurations* | *Set Active* from the project context menu and choose which configuration should be built.

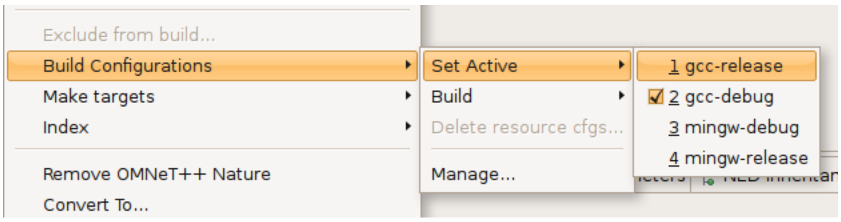


Figure 5.13. Activating a build configuration

Once the correct configuration is active, select *Build Project* from the *Project* menu or from the project context menu. This should start the build process. First, a makefile will be created in each source folder (according to the project properties *Makemake* page) and then the primary makefile will be called to start the build process. You should switch to the console view to see the actual progress of the build.

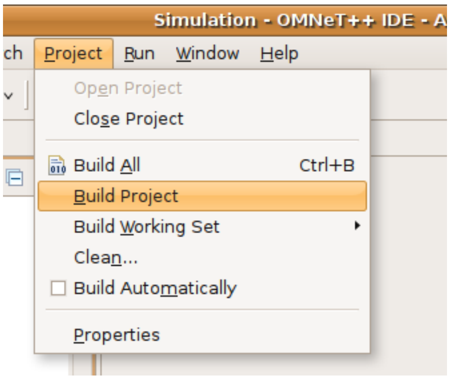


Figure 5.14. Building a project

> If you have a multi-core system, you may try to configure OMNeT++ to support parallel build. Open the Project properties dialog (Properties from the project context menu), and select the *C/C++ Build* page and the *Behavior* tab. Click the *Use parallel build* checkbox and specify the number of parallel jobs. Do not set this higher than the number of CPUs available in your system (Warning: Never use the "Optimal jobs number" choice as it will start too many jobs at once and you will run out of physical memory very quickly).

5.8. Running or Debugging the Project

If you want to run or debug your simulation, open the *Run* menu and choose the *Run/Debug Configurations...* dialog. For a more detailed discussion, please visit the Launching and Debugging chapter.

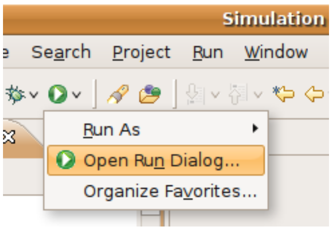


Figure 5.15. Running a project

5.9. Include Browser View

Dropping a C++ file into the *Include Browser View* displays the include files used by the C++ file (either directly or indirectly).

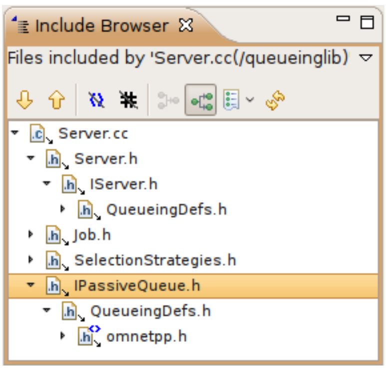


Figure 5.16. Include Browser

5.10. Outline View

During source editing, the Outline View gives you an overview of the structure of your source file and can be used to quickly navigate inside the file.

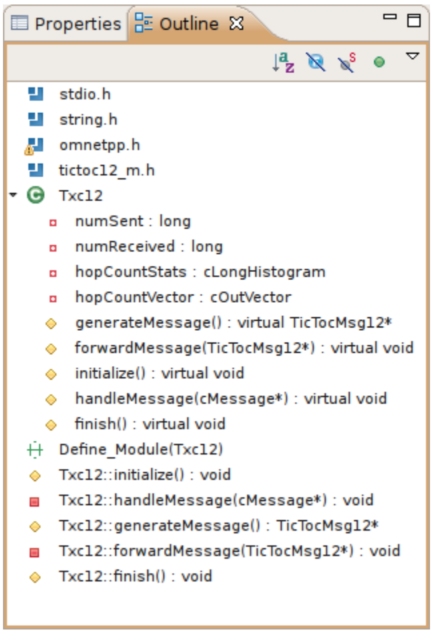


Figure 5.17. Navigating with Outline View

5.11. Type Hierarchy View

Displaying the C++ type hierarchy may be a great help in understanding the code and the inheritance relationship between your classes (and between OMNeT++ classes).

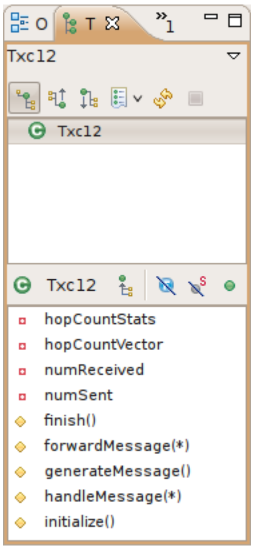


Figure 5.18. C++ Type hierarchy

5.12. Problems View

The *Problems View* contains the errors and warnings generated by the build process. You can browse the problem list and double-click any message to go to the problem location in the source file. NED file and INI file problems are also reported in this view along with C++ problems. The editors are annotated with these markers, too. Hover over an error marker in the editor window to get the corresponding message as a tooltip.

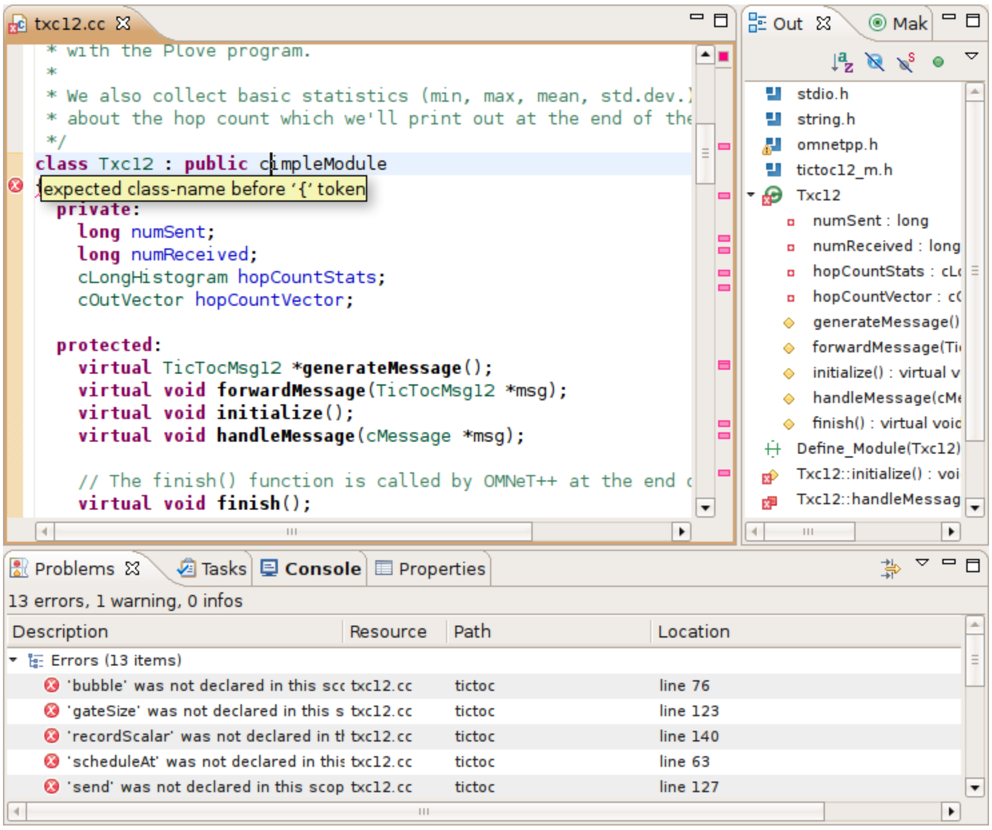


Figure 5.19. C++ problems

5.13. Console View

You may use the *Console View* to see the results of a build process or an actual simulation run.

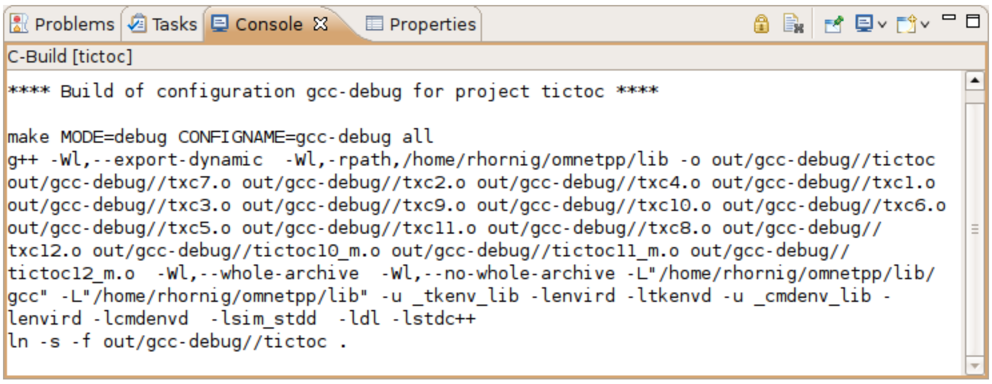


Figure 5.20. Build output in a console

Chapter 6. Launching and Debugging

6.1. Running a Simulation

In previous versions of OMNeT++, executing a simulation or carrying out experiments was done by manually launching the executable file from the windowing environment or from a command line shell. Experiments that needed several runs with different parameters required the writing and execution of external scripts. The new version of the OMNeT++ IDE allows for running simulations and simulation batches directly from the IDE.

6.1.1. Creating a Launch Configuration

OMNeT++ IDE adds a new Eclipse launch configuration type that supports launching simulation executables. This launch type, OMNeT++ Simulation, is available from the *Run Configurations* dialog. To create a new run configuration, select *Run Configurations...* from the *Run* menu. A dialog appears with the possible launch types on the left. Select *OMNeT++ Simulation* and click the *New launch configuration* icon. Then, enter a configuration name at the top of the dialog form that appears.

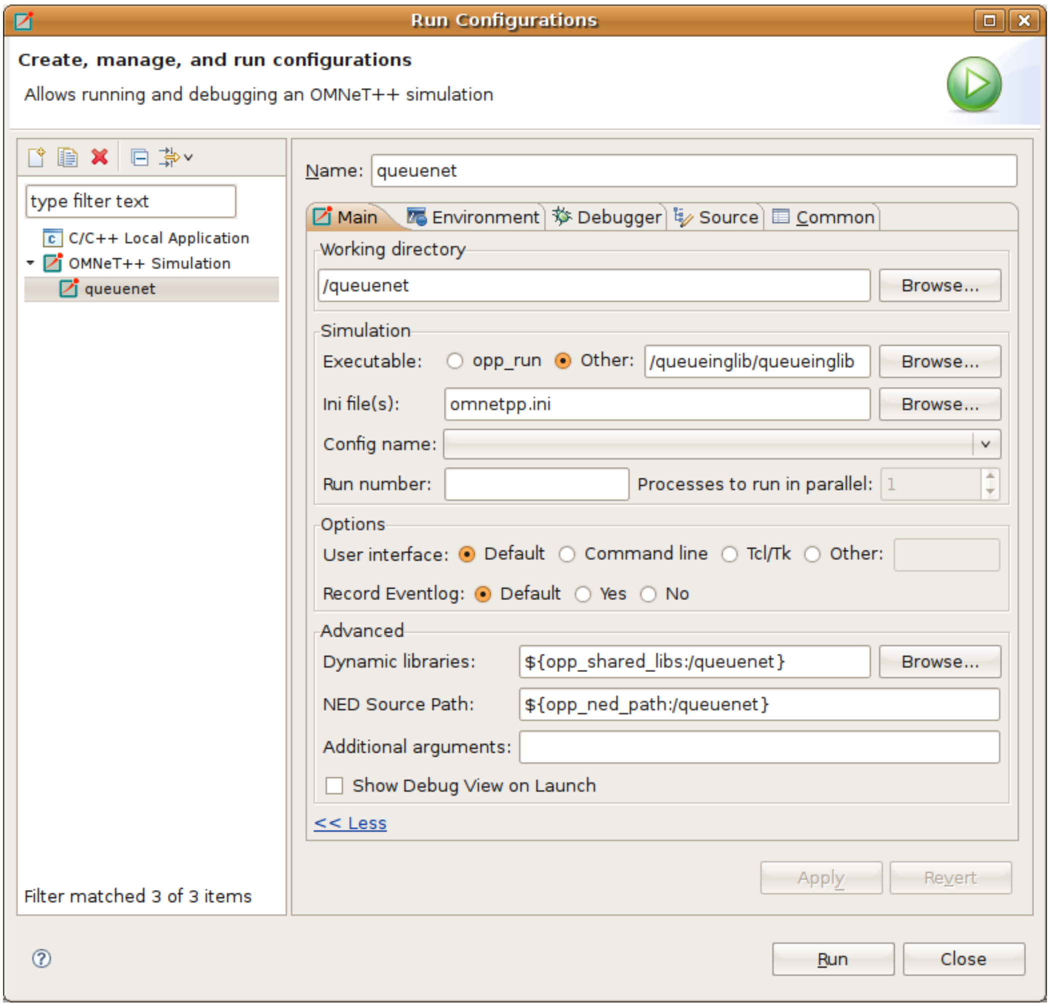


Figure 6.1. The Simulation Launcher

The *Main* tab of the configuration dialog was designed to make the launching of simulations as easy as possible. The only required field is the Working directory; all others have defaults. If you only select the working directory and the simulation program, it will start "Run 0" of the first configuration from the `omnetpp.ini` file in the specified working directory.

> Hover your mouse above the controls in this dialog and you will receive tooltip help for the selected control.

> NOTE
> The *Launch* dialog will try to figure out your initial settings automatically. If you select an INI file in the *Project Explorer View*, or the active editor contains an INI file before launching the *Run* dialog, the INI file and working directory field will be automatically populated for you. The dialog will try to guess the executable name based on the settings of your current open projects.

- *Working directory*: Set it to your working directory. Several of the below entries are treated as relative to this directory. Changing the working directory may invalidate previously selected entries in other fields of the dialog.

- *Simulation program*: You must set the name of the simulation executable here. This path is relative to the workspace root. You may use the *Browse...* button to select the executable directly. If your project output is a shared library, select "opp_run". The launcher dialog will use the "opp_run" helper executable to launch your simulation in the shared library. Check whether the "Dynamic Libraries" field in the advanced section contains the libraries you want to load.

> If you want to debug your simulation, be sure to select the 'debug' version of your executable here.

- *Initialization file(s)*: You should specify one or more INI files that will be used to launch the simulation. Specifying more than one file simply loads all those files in the specified order.

- *Configuration name*: Once you specify a legal INI file, the box will present all of the Config sections in that file. In addition, it will display the description of that section and the information regarding which Config section is extended by this section. You may select which Configuration should be launched.

> NOTE
> The working directory and the INI file must contain valid entries before trying to set this option.

- *Run number*: It is possible to specify which run number(s) must be executed for the simulation. If the executable name and the INI file were already selected, it is possible to hover above the field to get more information about the possible run numbers. You can use comma and .. to separate the run numbers; for example, 1,2,5..9,20 corresponds to run numbers 1,2,5,6,7,8,9,20. Entering a single asterisk (*) corresponds to all run numbers. Running several simulations in this manner is called batch execution. If you do not specify anything, Run 0 will be executed.

> NOTE
> Batch execution is possible only if you run your program in the command line environment. The simulation was built to support command line environment (Cmdenv) and the User interface selection is 'Command line' (see User interface selection).

- *Processes to run in parallel*: With batch execution, it is possible to tell the launcher to keep two or more simulations running at a time. This way you can take advantage of multiple CPUs or CPU cores.

> This is usually an easier and more efficient way to exploit multiprocessing power than parallel simulation (PDES).

> Use this option only if your simulation is CPU-limited and you have enough physical RAM to support all of the processes at the same time. Do not set it higher than the number of physical processors or cores you have in your machine.

- *User interface*: You can specify which UI environment should be used during execution; currently command line (Cmdenv) and Tcl/Tk (Tkenv) is supported. The executable must be linked with the correct UI library or the UI should be loaded dynamically.

> Batch execution and progress feedback during execution is supported only for the command line environment.

- *Record Eventlog*: Choosing "Default" uses the settings specified in the INI file, however you may explicitly enable or disable simulation event logging. This is useful if you want to temporarily switch the logging on or off.

- *Dynamic libraries*: A simulation may load additional DLLs or shared libraries before execution or your entire simulation may be built as a shared library. The *Browse* button is available to select one or more files (use **CTRL** click for multiple selection). This option can be used to load simulation code (i.e. simple modules), user interface libraries, or other extension libraries (scheduler, output file managers, etc.). The special macro `${opp_shared_libs:/workingdir}` expands to all shared libraries provided by the current project or any other project on which you currently depend.

> If your simulation is built as a shared library, you must use the "opp_run" stub executable to start it. opp_run is basically an empty OMNeT++ executable which understands all command line options, but does not contain any simulation code.

> If you use external shared libraries (i.e. libraries other than the ones provided by the current open projects or OMNeT++ itself), you must ensure that the executable part has access to the shared library. On Windows, you must set the `PATH`, while on Linux and Mac you must set the `LD_LIBRARY_PATH` to point to the directory where the DLLs or shared libraries are located. You can set these variables either globally or in the *Environment* tab in the *Launcher Configuration Dialog*.

- *NED Source Path*: The directories where the NED files are read from (relative to the first selected INI file.)

> The variable `${opp_ned_path:/workingdir}` refers to an automatically calculated path (derived from project settings). If you want to add additional NED folders to the automatically calculated list, use the `${opp_ned_path:/workingdir}:/my/additional/path` syntax.

- *Additional arguments*: Other command line argument can be specified here and will be passed to the simulation process.

- *Show Debug View on Launch*: This is a convenience function that allows the user to open the Debug View on simulation execution. *Debug View* allows you to see and terminate the processes you have launched and gives you the option to switch between their outputs in the console view.

Related Command-Line Arguments

Most settings in the dialog simply translate to command-line options to the simulation executable. This is summarized in the following list:

- Initialization files: maps to multiple -f <inifile> options
- Configuration name: adds a -c <configname> option
- Run number: adds a -r <runnumber> option
- User interface: adds a -u <userinterface> option
- Dynamically loaded libraries: maps to multiple -l <library> options
- NED Source Path : adds a -n <ned_source_path> option

6.2. Batch Execution

In previous versions of OMNeT++, to run a simulation several times with different parameters, you had to create an external script that created an INI file for each run and started the simulation with the specified file. In OMNeT++ 4.x, it is no longer needed. The improved INI file syntax allows you to specify loops for specific parameters so you no longer need separate INI files for each run. In addition, the IDE itself supports starting your simulation several times.

```
[Config PureAlohaExperiment]
description = "Channel utilization in the function of packet generation frequency"
repeat = 2
sim-time-limit = 300min
**.vector-recording = false
Aloha.host[*].iaTime = exponential(${mean=1,1.5,2,3,4,5..21 step 2}s)
```

Figure 6.2. Iteration variable in the INI file

> NOTE: Batch running is supported only in the command line environment.

If you create a Configuration with one or more iteration variables, you will be able to run your simulations to explore the parameter space defined by those variables. Practically, the IDE creates the Cartesian product from these variables and assigns a run number to each product. It is possible to execute one, several or all runs of the simulation by specifying the *Run number* field in the *Run Dialog*. You can specify a single number (e.g. 3), a combination of several numbers (e.g. 2,3,6,7..11) or all run numbers (using *).

> If you already have specified your executable, chosen the configuration which should be run and selected the command line environment, you may try to hover over the *Run Number* field. This will give you a description of the possible runs and how they are associated with the iteration variable values (the tooltip is calculated by executing the simulation program with the -x Congiguration -G options in command line mode).

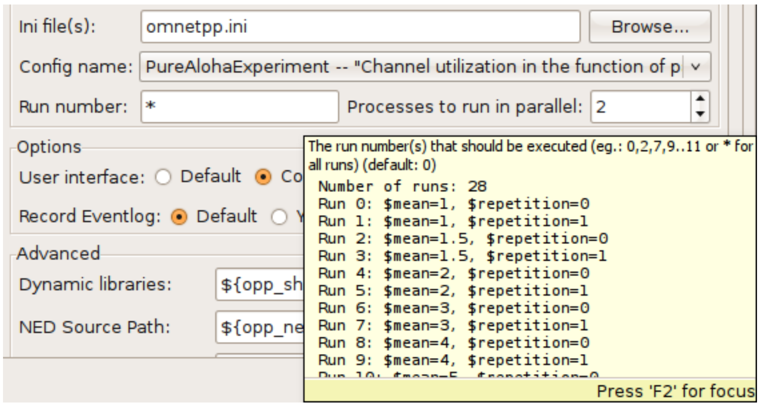


Figure 6.3. Iteration loop expansion in a tooltip

If you have a multi-core or multi-processor system and have ample memory, you may try to set the *Processes to run parallel* field to a higher number. This will allow the IDE to start more simulation processes in parallel, resulting in a much lower overall simulation time for the whole batch.

Be aware that you need enough memory to run all these processes in parallel. We recommend using this feature only if your simulation is CPU-bound. If you do not have enough memory, your operating system may start to use virtual memory, dramatically decreasing the overall performance.

6.3. Debugging a Simulation

The OMNeT++ IDE integrates with the CDT (C/C++ Development Tooling) of Eclipse. If you want to debug your application, you have to do it by starting it from the *Run|Debug Configurations*. You should launch your debugging session exactly the same way you were running it.

If you have problems with starting the debug session, check whether:

- your executable is built with debug information,
- you can run the same executable without problem (using the same launch configuration), and
- the Debugger type is set properly on the *Debugger* tab of the *Launch* dialog.

Batch (and parallel) execution is not possible in this launch type, so you may specify only a single run number.

6.4. Launching Shortcuts

While the *Launch* dialog is very powerful, in most cases you will want to run your simulation as quickly as possible. Based on the current selection, the IDE provides several quick methods to create a launch configuration. You can select a file by either clicking on it in the *Project Explorer*, or by opening it for editing. After selection, just click on the *Run* or *Debug* icon on the toolbar, or right-click on the file and choose the *Run As...* or *Debug As...* menu.

- If a directory is selected and contains a single INI file, the IDE will use this file to start the simulation.

- If an INI file is selected, it will be used during the launch as the main INI file for the simulation.
- If a NED file is selected which contains a network definition, the IDE will scan for INI files in the active projects and will try to find a configuration that allows this network to start.

> NOTE: If the information to launch the simulation is insufficient or ambiguous, the IDE will ask you whether it should create a configuration.

6.5. Controlling the Execution and Progress Reporting

After starting a simulation process or simulation batch you can keep track of the started processes in the *Debug View*. To open the *Debug View* automatically during launch, check the Show Debug View on Launch in the run configuration dialog, or select *Window* | *Show View...* | *Other...* | *Debug* | *Debug*. Select a process and click the terminate button to stop a specific simulation run or use the context menu for more options to control the process execution.

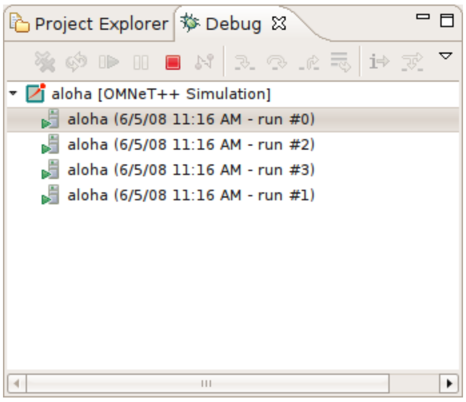


Figure 6.4. Debug View

> Place the Debug View in a different tab group than the console so you will be able to switch between the process outputs and see the process list at the same time.

> NOTE: You can terminate all currently running processes by selecting the root of the launch. This will not cancel the whole batch; only the currently active processes. If you want to cancel the whole batch, open the *Progress View* and cancel the simulation batch there.

Clicking on the process in the *Debug View* switches to the output of that process in the *Console View*. The process may ask for user input via the console, too. Switch to the appropriate console and enter the requested parameters.

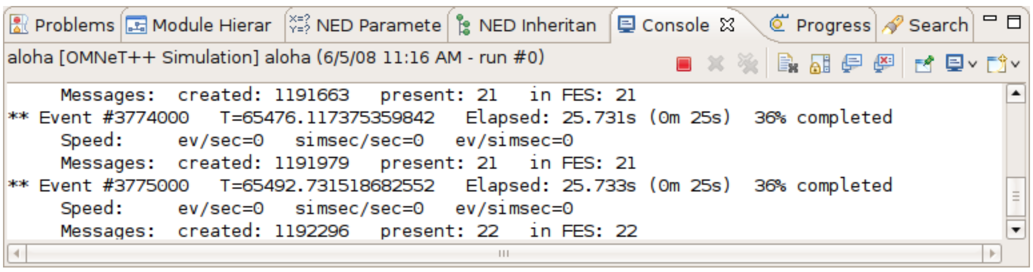


Figure 6.5. Displaying the output of a simulation process in Console View

By default, the *Console View* automatically activates when a process is writing to it. If you are running several parallel processes, this might be an annoying behavior and might prevent you from switching to the *Progress View*. You can switch off the auto-activation by disabling the *Show Console When Standard Out/Error Changes* in the *Console View* toolbar.

Progress Reporting

If you have executed the simulation in the command line environment, you can monitor the progress of the simulation in the *Progress View*. See the status line for the overall progress indicator and click on it to open the detailed progress view. It is possible to terminate the whole batch by clicking on the cancel button in the *Progress View*.

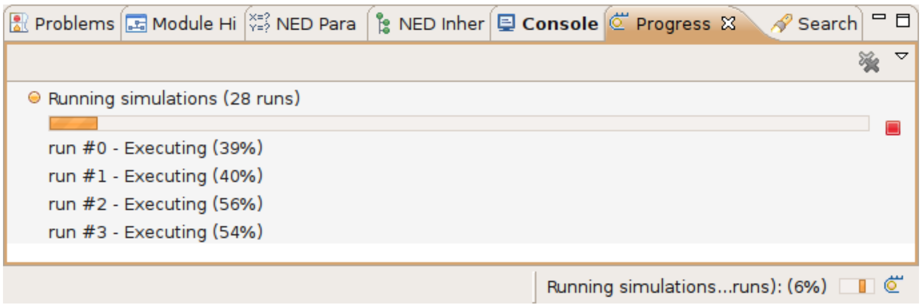


Figure 6.6. Progress report on four parallel processes

When *Progress View* displays "Waiting for user input", the simulation is waiting for the user. Switch to the appropriate console and provide the requested input for the simulation.

By default, cmdenv reports progress only on every 100,000th event. If you want more frequent progress reports, set the `cmdenv-status-frequency` option in your INI file to a lower value.

Chapter 7. The Tkenv Graphical Runtime Environment

7.1. Features

Tkenv is a graphical runtime interface for simulations. Tkenv supports interactive simulation execution, animation, tracing and debugging. Tkenv is recommended in the development stage of a simulation and for presentation and educational purposes, since it allows the user to get a detailed picture of the state of the simulation at any point in its execution, and to follow what happens inside the network. The most important feaures are:

- message flow animation
- graphical display of statistics (histograms, etc.) and output vectors during simulation execution
- separate window for each module's text output
- scheduled messages can be watched in a window as simulation progresses
- event-by-event, normal and fast execution
- labeled breakpoints
- inspector windows to examine and alter objects and variables in the model
- simulation can be restarted
- snapshots (detailed report about the model: objects, variables, etc.)

Tkenv makes it possible to view simulation results (output vectors, etc.) during execution. Results can be displayed as histograms and time-series diagrams. This can speed up the process of verifying the correct operation of the simulation program and provides a good environment for experimenting with the model during execution. When used together with a C++ source-level debugger, Tkenv can significantly speed up development time.

7.2. Starting Tkenv

A simulation program built with Tkenv accepts the following command line switches:

- `-h` : The program prints a help message and exits.
- `-f fileName` : Specifies the name of the configuration file. The default is `omnetpp.ini`. Multiple `-f` switches can be given; this allows you to partition your configuration file. For example, one file can contain your general settings, another one most of the module parameters, and a third one the module parameters you change frequently. The `-f` switch is optional and can be omitted.
- `-l fileName` : Loads a shared library (`.so` file on Unix, `.dll` on Windows, and `.dylib` on Mac OS X). Multiple `-l` switches are accepted. Shared libraries may contain simple modules and other, arbitrary code. File names may be specified without the file extension and the `lib` name prefix (i.e. `foo` instead of `libfoo.so`).
- `-n filePath` : When present, overrides the `NEDPATH` environment variable and sets the source locations for simulation NED files.

- `-c configname` : Selects an INI configuration for execution.
- `-r run-number` : It has the same effect as (but takes priority over) the *tkenv-default-run=* INI file configuration option.

7.3. Configuration Options

Tkenv accepts the following configuration options in the INI file.

- `tkenv-extra-stack` : Specifies the extra amount of stack (in kilobytes) that is reserved for each `activity()` simple module when the simulation is run under Tkenv. This value is significantly higher than the similar one for Cmdenv (handling GUI events requires a large amount of stack space).
- `tkenv-default-config` : Specifies which INI file configuration Tkenv should set up automatically after startup. If there is no such option, Tkenv will ask which configuration to set up.
- `tkenv-default-run` : Specifies which run of the selected configuration Tkenv should set up after startup. If there is no such option, Tkenv will ask.
- `tkenv-image-path` : Specifies which directories (in addition to `OMNETPP_IMAGE_PATH` or its built-in default value) Tkenv will load images from. This is a semicolon-separated list of directories. The directories will be scanned recursively, and subdirectory names become part of the icon name; for example, if an `images/` directory is listed, `images/misc/foo.png` will be registered as icon `misc/foo`. PNG and GIF files are accepted.
- `tkenv-plugin-path` : Specifies which directories (in addition to `OMNETPP_PLUGIN_PATH` or its built-in default value) will be scanned for Tkenv plugins written in Tcl.

The rest of the Tkenv options are saved into the `.tkenvrc` file in the user's home directory.

7.4. Environment Variables

In case of nonstandard installation, it may be necessary to set the `OMNETPP_TKENV_DIR` environment variable so that Tkenv can find its parts written in Tcl script.

The default path from where the icons are loaded can be changed with the `OMNETPP_IMAGE_PATH` variable, which is a semicolon-separated list of directories that defaults to `omnetpp-dir/images;./images`.

The path from where Tkenv plugins written in Tcl are loaded can be changed with the `OMNETPP_PLUGIN_PATH` variable, which is a semicolon-separated list of directories that defaults to `./plugins`.

7.5. The Main Window

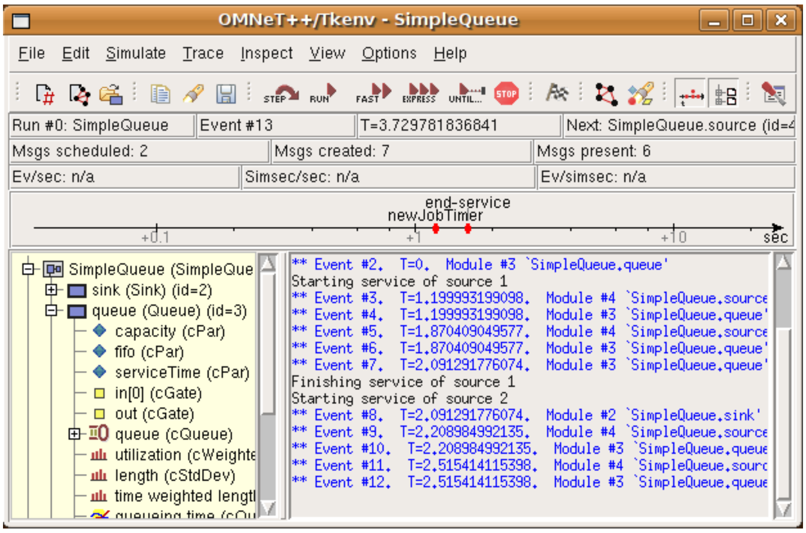


Figure 7.1. The main window of Tkenv

The main window of the simulation environment contains:

- *Toolbar*: You may access the main functions of the runtime environment. The toolbar is displayed directly below the main window's menu. It is possible to step, run, stop, restart or finish the simulation and configure the visual appearance and logging behavior.
- *Status bar*: Contains information about the current state of the simulation. On the upper line, it displays the current run number and network name, the current event number, the simulation time and the module name to where the next simulation event will be delivered. The middle line displays the number of messages in the FES, the total number of messages created during the simulation and finally the number of messages currently present in the whole simulation model. The last line contains performance data: How many events are processed in a real and in a simulated second along with the relative speed between the simulation-time and real-time.
- *Timeline*: Displays the contents of the Future Events Set on a logarithmic time scale. Each dot represents an event. Double-clicking on a dot will open an inspector window for the event. The contents of the Timeline can be configured on the *Timeline* tab of the *Simulation Options* dialog by specifying message filters. (Tip: the Timeline context menu provides a shortcut to that dialog).
- *Object Tree*: Displays all inspectable objects currently present in the memory. You may double-click on nodes to further inspect the given object in a separate window.
- *Log Area*: Contains the output of the simulation. You can filter the content of the window to only include messages from specific modules. Open the log window's context menu and select *Filter window contents*. Note that the whole content of the log window will be filtered including all previous messages.

Both the Object tree and the Timeline can be easily switched off on the toolbar if you need more space for the log area.

7.6. Inspecting the Simulation

7.6.1. Networks, Modules

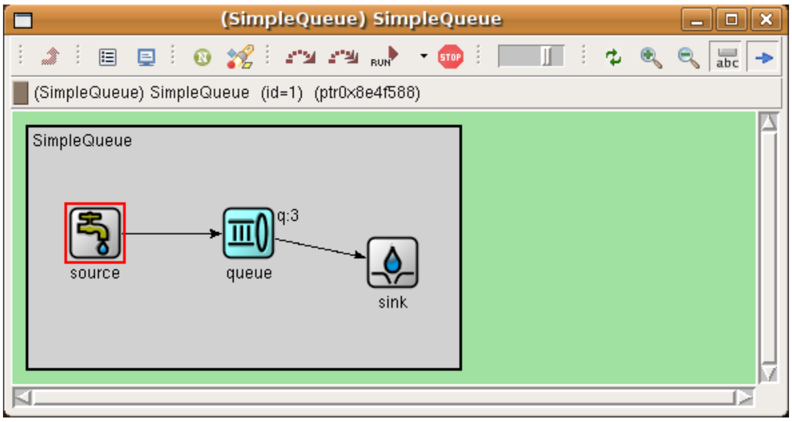


Figure 7.2. Top level network

You can start exploring the component hierarchy by browsing the Object Tree and double-clicking on a node or by clicking the *Inspect network* icon on the toolbar. Once you open a node, an inspector window will be opened. Networks and compound modules are represented by graphical inspectors that display their internal structures. Simple modules are represented by object inspectors that display their content as lists. Items in these lists can also be opened and some of them in the tree indicated by [...] can be changed during runtime. Double-click on the item and edit it in place.

If you intend to watch some variables in your modules, add the `WATCH(varName)` macro to your module's source code, in the `initialize()` method. The watched variables will be displayed on the object inspector's content page.

The toolbar on the module inspector window contains several useful commands that can be used related to this module. It is possible to directly inspect the parent module, or the type definition for this module, show the current module's log output or search for objects inside the module. It is also possible to step or run a simulation until the next event occurs in the module.

NOTE: Tkenv uses reflection information provided by the `Register_ClassDescriptor(class)` macro to display the object tree. If you want to include your own classes in the tree, you need to register them with the above macro.

7.6.2. Future Event Set (FES)

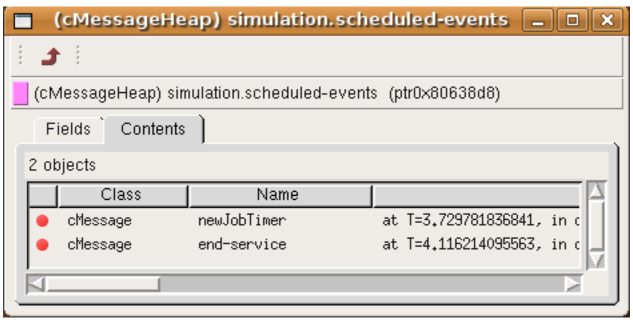


Figure 7.3. The content of the Future Event Set

You can peek into the Future Event Set by clicking on *Inspect | Scheduled events*. The FES contains all messages already scheduled. Double-clicking on any message will show the message object's inspector window for further investigation.

> NOTE
>
> The future event set is also displayed on the main window's timeline.

7.6.3. Output Vectors, Histograms, Queues

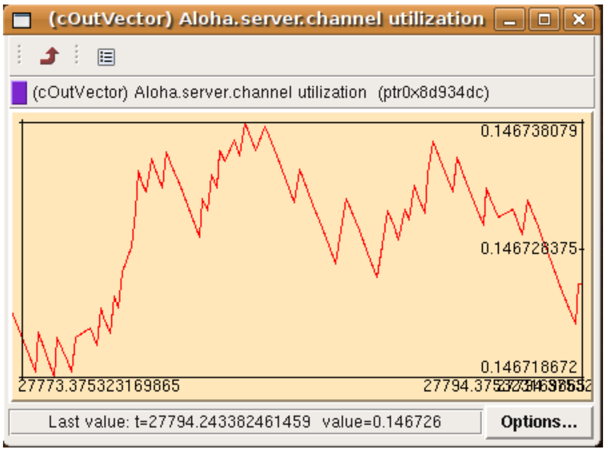


Figure 7.4. An output vector displayed in a window

You can view graphical representations of output vectors. Double-clicking an output vector object displays the content graphically as a line chart. You may set the display properties by clicking on the *Options...* button.

> NOTE
>
> To conserve memory, output vector windows only start gathering data when you open them.

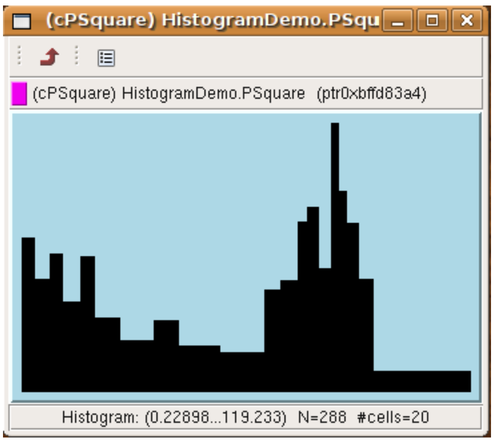


Figure 7.5. A real-time histogram window

Histogram inspectors provide a real-time display of histogram objects.

> NOTE
>
> Histogram graphics may not be available immediately, because a histogram object must precollect some data before it can determine the range of the data and set up the cells.

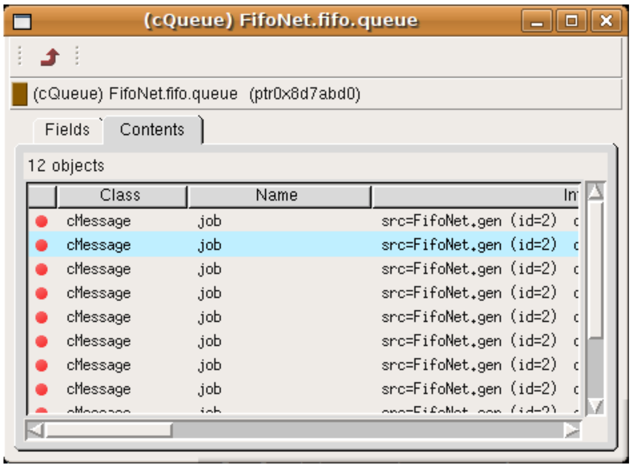


Figure 7.6. Peeking into a queue

If your simulation contains queue objects, you can peek into them. All messages currently waiting in the queue are displayed in the queue inspector window. Double-click on any message to inspect it further.

7.7. Browsing the Registered Components

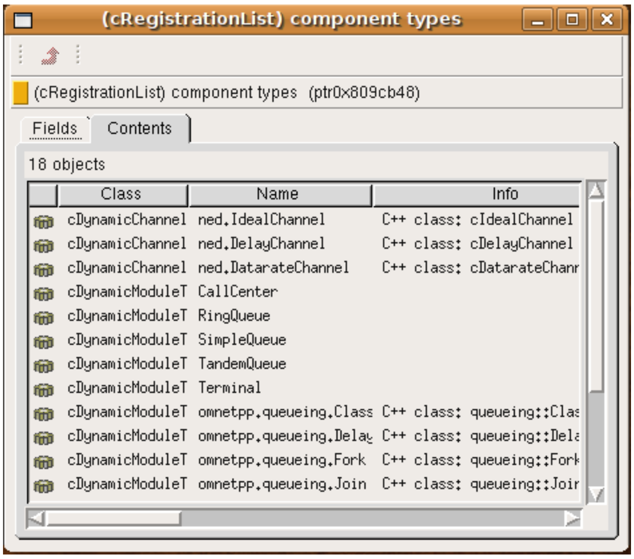


Figure 7.7. Display registered functions, classes, etc.

Registered components (NED Types, classes, functions, enums) can be displayed with the *Inspect | Available components*. If an error message reports missing types or classes, you can check here whether everything is loaded and registered correctly.

7.8. Running and Controlling the Simulation

Tkenv has several modes for running the simulation. These running modes have their corresponding buttons on Tkenv's toolbar:

7.8.1. Step Mode

In Step mode, you can execute the simulation event-by-event. The next event is always shown on the status bar. The module where the next event will be delivered is highlighted with a red rectangle on the graphical display.

7.8.2. Run Mode

In Run mode, the simulation runs with all tracing aids on. Message animation is active and inspector windows are updated after each event. Output messages are displayed in the main window and module output windows. You can stop the simulation with the *Stop* button on the toolbar. You can fully interact with the user interface while the simulation is running (e.g. you can open inspectors, etc.).

> NOTE: If you find this mode too slow or distracting, you may switch off animation features in the Tkenv configuration dialog.

7.8.3. Fast Mode

In Fast mode, animation is turned off. The inspectors and the message output windows are updated every 500 milliseconds (the actual number can be set in *Options|Simulation options...* and also in the INI file). Fast mode is several times faster than the Run mode; the speed can increase by up to 10 times (or up to the configured event count).

7.8.4. Express Mode

In Express mode, the simulation runs at about the same speed as with Cmdenv, all tracing disabled. Module output is not recorded in the output windows. You can interact with the simulation only once in a while, thus the run-time overhead of the user interface is minimal. You have to explicitly push the *Update inspectors* button if you want an update.

7.8.5. Run Until

You can run the simulation until a specified simulation time, event number or until a specific message has been delivered or canceled. This is a valuable tool during debugging sessions (select *Simulate | Run until...*). It is also possible to right-click on an event in the simulation timeline and choose the *Run until this event* menu item.

7.8.6. Run Until Next Event

It is also possible to run until an event occurs in a specified module. Browse for the module and choose *Run until next event in this module*. Simulation will stop once an event occurs in the selected module.

Tkenv has a status bar which is regularly updated while the simulation is running. Global simulation statistics like simulation speed, event density, current event number and simulation time are displayed on the status bar.

Simulations are created when Tkenv is started and the user has selected a network to simulate (or the user clicks the *New Network* button to select a new one). First, the simulation kernel loads all necessary NED files and registers the loaded modules with the kernel. Object constructors are called at this point. The next phase is initialization when all module's `initialize()` methods are called. Once initialization is complete, Tkenv returns the control to the user and allows stepping or running the simulation. If you choose to quit Tkenv before the simulation finishes (or try to restart the simulation), Tkenv will ask you whether to call the `finish()` method for all modules. This will allow the modules to correctly free up resources and save their statistics. Before the simulation program exits, the modules are deleted and their destructors are called.

7.9. Finding Objects

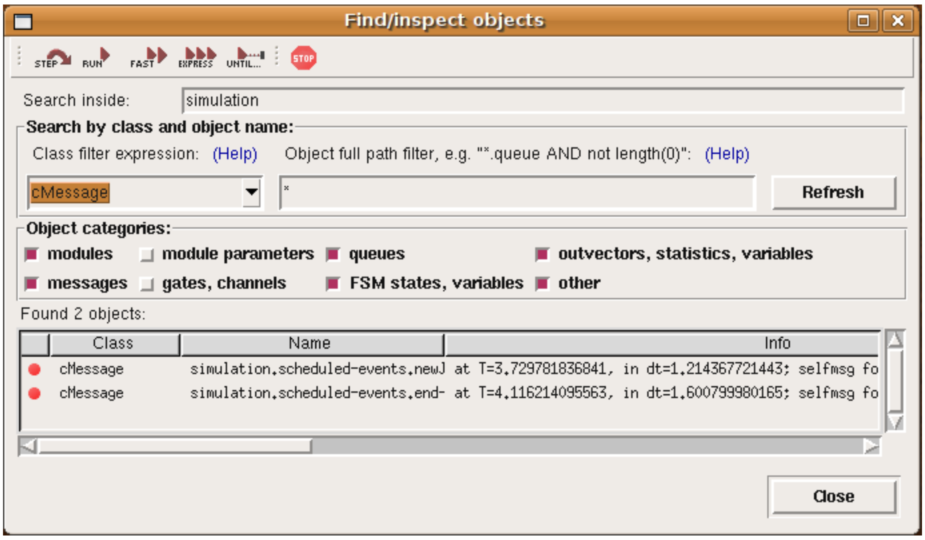


Figure 7.8. Finding a specific object

There are times when you do not know the exact location of the object for which you are looking. For example, you may want to know the locations for all your messages. Invoke the *Find/inspect objects* window and search for your object. The window allows you to set the start of the search along with the type, class and name of the object. The results will be presented as a clickable list.

> NOTE: The checkboxes can be used to select the object category that interests you. If you select a category, all objects with that type (and any type derived from it) will be included in the search. Alternatively, if you specify object class as a class filter expression, the search dialog will try to match the object's class name with the given string. This means that only those types of objects will be included in the search (derived types will not be included as they have different non-matching classnames).

You can provide a generic filter expression, which matches the object's full path by default. Wildcards (`"?"`, `"*"`) are allowed. `"{a-exz}"` matches any character in the range `"a"`..`"e"` plus `"x"` and `"z"`. You can match numbers: `"*.job{128..191}"` will match objects named `"job128"`, `"job129"`, ..., "job191". "job{128..}" and "job{..191}" are also understood. You can combine patterns with `AND`, `OR` and `NOT` and parentheses (lowercase and, or, not are also OK). You can match other object fields such as queue length, message kind, etc., with the syntax `"fieldname(pattern)"`. Put quotation marks around a pattern if it contains parentheses (HINT: You will want to start the pattern with `"*."` in most cases to match objects anywhere in the network!). `*.subnet2.*.destAddr"destAddr""subnet2"`

- `*.destAddr` : Matches all objects with the name `"destAddr"` (likely module parameters).
- `*.node[8..10].*` : Matches anything inside module `node[8]`, `node[9]` and `node[10]` .
- `className(cQueue) and not length(0)` : Matches non-empty queue objects.
- `className(cQueue) and length({10..})` : Matches queue objects with length>=10.
- `kind(3) or kind({7..9})` : Matches messages with message kind equal to 3, 7, 8 or 9 (only messages have a `"kind"` attribute).

- `className(IP*) and *.data-*` : Matches objects whose class name begins with `"IP"` and name begins with `"data-."`
- `not className(cMessage) and byteLength({1500..})` : Matches messages whose class is not cMessage and byteLength is at least 1500 (only messages have a `"byteLength"` attribute).
- `"*(*" or "*.msg(ACK)"` : Quotation marks needed when pattern is a reserved word or contains parentheses (note: *.msg(ACK) without parentheses would be interpreted as some object having a "*.msg" attribute with the value "ACK"!).

> **NOTE** Tkenv uses the `cObject::forEachChild` method to collect all objects from a tree recursively. If you have your own objects derived from `cObject`, you should redefine the `cObject::forEachChild` to function correctly with an object search.

> **NOTE** If you are debugging the simulation with a source level debugger, you may also use the *Inspect by pointer* menu item. Let the debugger display the address of the object to be inspected, and paste it into the dialog. Please note that entering an invalid pointer will crash the simulation.

7.10. Logging and Module Output

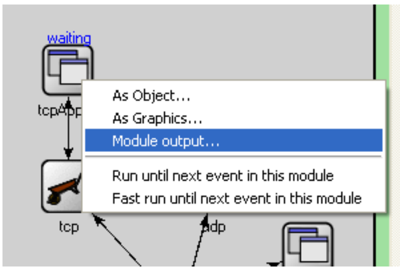


Figure 7.9. Opening the log of a single module

By default, the main window displays the output of the whole simulation. It is possible to filter the content of the output windows by opening the log window's context menu and selecting the *Filter window contents* menu item. Individual modules can be selected and deselected.

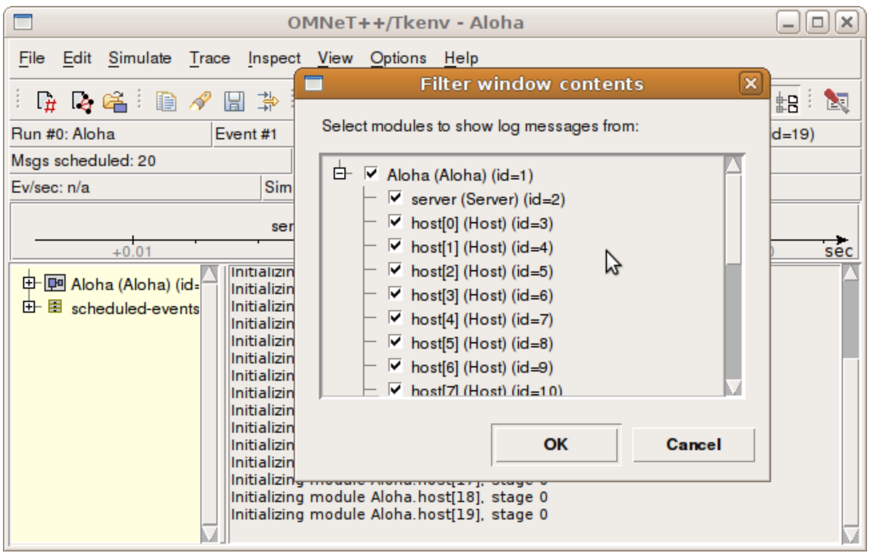


Figure 7.10. Filtering the output of a compound module

General logging behavior can be controlled in the *Simulation Options* dialog.

It is also possible to open separate log windows for individual modules. A log window for a compound module displays the log from all of its submodule tree. To open a log window, find the module in the module tree or the network display, right-click it and choose *Module output* from the context menu.

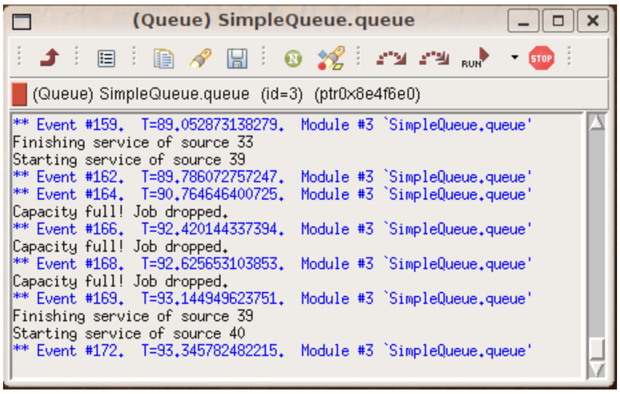


Figure 7.11. Module output

The module output window only displays messages generated by the selected module and its submodules. You can open several module output windows and place them side by side to follow the workings of several modules at the same time.

NOTE

All message lines in the log window are filtered according to the filter settings (older ones, too).

7.11. Simulation Options

Select *Options | Simulation Options...* from the menu to display the runtime environment's preferences dialog. It is possible to adjust various display, network layouting and animation options in the dialog. For example, it is possible to disable various animation effects.

7.11.1. General

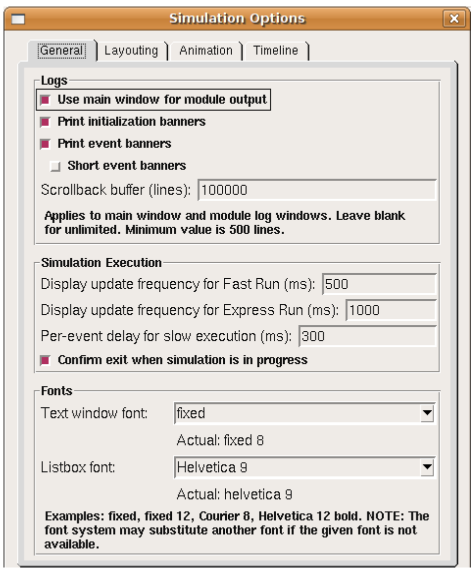


Figure 7.12. General Tkenv options

The *General* tab can be used to set the default display and logging behavior. It is possible to set how often the user interface will be updated during the simulation run. The user can also configure default font size and family here.

7.11.2. Configuring the Layouting Algorithm

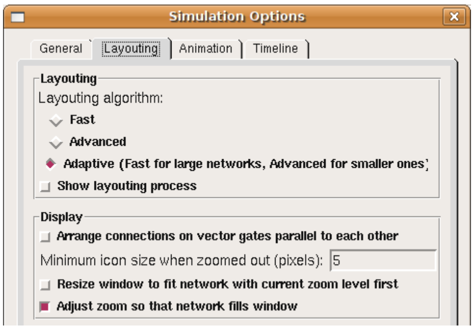


Figure 7.13. Layouting options

Tkenv provides automatic layouting for submodules that do not have their locations specified in the NED files. The layouting algorithm can be fine-tuned on the *Layouting* page of the configuration dialog.

7.11.3. Configuring Animation

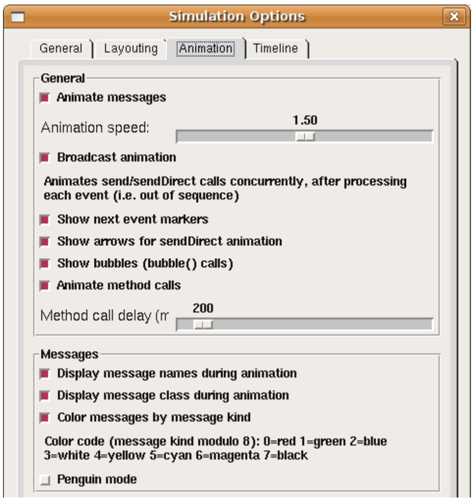


Figure 7.14. Animation options

Tkenv provides automatic animation when you run the simulation. You can fine-tune the animation settings using the *Animation* tab on the configuration dialog. If you do not need all visual feedback Tkenv provides, you can selectively turn off some of the features:

- Animate messages : Turns on/off the visualization of messages passing between modules.
- Broadcast animation : Handles message broadcasts in a special way (messages sent within the same event will be animated concurrently).
- Show next event maker : Highlights the module which will receive the next event.
- Show a flashing arrow when a `sendDirect()` method call is executed.
- Show a flashing arrow when a method call occurs from one module to another. The call is only animated if the called method contains the `Enter_Method()` macro.

- The display of message names and classes can also be turned off.

7.11.4. Configuring the Timeline

On the *Timeline* tab, you can control which messages and events will be displayed on the Timeline View in the main window. Self/non-self messages can be disabled and filter expressions can be specified based on message class or name.

For object names, wildcards (`"?"`, `"*"`) are allowed. `"{a-exz}"` matches any character in the range `"a".."e"` plus `"x"` and `"z"`. You can match numbers: `"job{128..191}"` will match `"job128"`, `"job129"`, ..., "job191". "job{128..}" and "job{..191}" are also understood. You can combine patterns with `AND`, `OR` and `NOT` and parentheses (lowercase and, or, not are also acceptable). You can match against other object fields such as message length, message kind, etc. with the syntax `"fieldname(pattern)"`. Put quotation marks around a pattern if it contains parentheses.

- `m*` : matches any object whose name begins with "m"
- `m* AND *-{0..250}` : matches any object whose name begins with "m" and ends with a dash and a number between 0 and 250
- `not *timer*` : matches any object whose name does not contain the substring "timer"
- `not (*timer* or *timeout*)` : matches any object whose name does not contain either "timer" or "timeout"
- `kind(3) or kind({7..9})` : matches messages with message kind equal to 3, 7, 8 or 9
- `className(IP*) and data-*` : matches objects whose class name begins with "IP" and name begins with "data-"
- `not className(cMessage) and byteLength({1500..})` : matches objects whose class is not cMessage and whose byteLength is at least 1500
- `"or" or "and" or "not" or "*(*" or "msg(ACK)"` : quotation marks needed when pattern is a reserved word or contains parentheses (note: msg(ACK) without parentheses would be interpreted as an object having an "msg" attribute with the value "ACK"!).

7.11.5. The .tkenvrc File

Options in the Tkenv runtime are stored in `.tkenvrc` files. There are two `.tkenvrc` files. One is stored in the current directory and contains project-specific settings like the open inspector's position, size, etc. The other `.tkenvrc` can be found in the user's home directory and contains global settings like font size and family.

> **NOTE**
>
> Inspectors are identified by their object names. If you have several components that share the same name (this is especially common for messages), you may end up with a lot of inspector windows when you start the simulation. In such cases, you may simply delete the `.tkenvrc` file.

Chapter 8. Sequence Charts

8.1. Introduction

This chapter describes the Sequence Chart and the Eventlog Table tools. Both of them display an eventlog file recorded by the OMNeT++ simulation kernel.

An eventlog file contains a log of messages sent during the simulation and the details of events that prompted their sending or reception. This includes both messages sent between modules and self-messages (timers). The user can control the amount of data recorded from messages, start/stop time, which modules to include in the log, and so on. The file also contains the topology of the model (i.e. the modules and their interconnections).

> Please refer to the OMNeT++ Manual for further details on eventlog files and their exact format.

The Sequence Chart displays eventlog files in a graphical form, focusing on the causes and consequences of events and message sends. They help the user understand complex simulation models and help with the correct implementation of the desired component behaviors. The Eventlog Table displays an eventlog file in a more detailed and direct way. It is in a tabular format, so that it can show the exact data. Both tools can display filtered eventlogs created via the Eventlog Tool filter command as described in the OMNeT++ Manual, by a third party custom filter tool, or by the IDE's in-memory filtering.

Using these tools, you will be able to easily examine every detail of your simulation back and forth in terms of simulation time or events. You will be able to focus on the behavior instead of the statistical results of your model.

8.2. Creating an Eventlog File

The INI File Editor in the OMNeT++ IDE provides a group of widgets in the *Output Files* section to configure automatic eventlog recording. To enable it, simply put a checkmark next to its checkbox, or insert the line

```
record-eventlog = true
```

into the INI file.

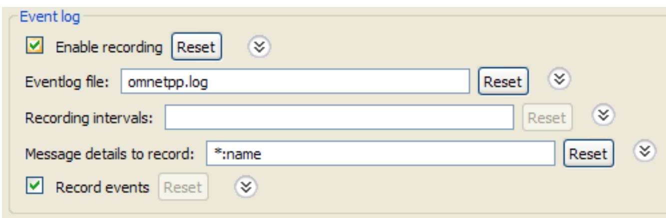


Figure 8.1. INI file eventlog configuration

By default, the recorded eventlog file will be put in the project's `results` directory, with the name `${configname}-${runnumber}.elog`.

> If you override the default file name, please make sure that the file extension is `elog`, so that the OMNeT++ IDE tools will be able to recognize it automatically.

The 'recording intervals' and 'record events' configuration keys control which events will be recorded based on their simulation time and on the module where they occur. The 'message details' configuration key specifies what will be recorded from a message's content. Message content will be recorded each time a message gets sent.

The amount of data recorded will affect the eventlog file size, as well as the execution speed of the simulation. Therefore, it is often a good idea to tailor these settings to get a reasonable tradeoff between performance and details.

> NOTE
>
> Please refer to the OMNeT++ Manual for a complete description of eventlog recording settings.

8.3. Sequence Chart

This section describes the Sequence Chart in detail, focusing on its features without a particular example.

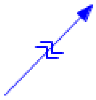
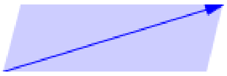
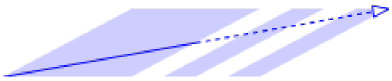
The Sequence Chart is divided into three parts: the top gutter, the bottom gutter and the main area. The gutters show the simulation time while the main area displays module axes, events and message sends. The chart grows horizontally with simulation time and vertically with the number of modules. Module axes can optionally display enumerated or numerical vector data.

There are various options, which control how and what the Sequence Chart displays. Some of these are available on the toolbar, while others are accessible only from the context menu.

8.3.1. Legend

Graphical elements on the Sequence Chart represent modules, events and messages, as listed in the following table.

	simple module axis
	compound module axis
IDLE	axis with attached vector data
Net10.rte[7].queue[3]	module full path as axis label
(hollow circle)	initialization event
(green disc)	self-message processing event
(red disc)	message processing event
#5078	event number
(blue arrow, arched)	self-message
(blue arrow)	message send
(green dotted arrow)	message reuse
(brown dotted arrow)	method call

 (arrow with a dashed segment)	message send that goes far away; split arrow
 (arrow with zigzag)	virtual message send; zigzag arrow
 (blue parallelogram)	transmission duration; reception at start
 (blue parallelogram)	transmission duration; reception at end
 (blue strips)	split transmission duration; reception at start
 (blue strips)	split transmission duration; reception at end
pk-9-to-1-#58 (blue letters)	message name
requestPacket() (brown letters)	method name
(gray background)	zero simulation time region
(dashed gray line)	simulation time hairline

8.3.2. Timeline

Simulation time may be mapped onto the horizontal axis in various ways; linear mapping is only one of the ways. The reason for having multiple mapping modes is that intervals between interesting events are often of different magnitudes (e.g. microsecond timings in a MAC protocol versus multi-second timeouts in higher layers), which is impossible to visualize using a linear scale.

The available timeline modes are:

- Linear -- the simulation time is proportional to the distance measured in pixels.
- Event number -- the event number is proportional to the distance measured in pixels.

- Step -- the distance between subsequent events, even if they have non-subsequent event numbers, is the same.
- Nonlinear -- the distance between subsequent events is a nonlinear function of the simulation time between them. This makes the figure compact even if there are several magnitudes difference between simulation time intervals. On the other hand, it is still possible to decide which interval is longer and which one is shorter.
- Custom nonlinear -- like nonlinear. This is useful in those rare cases when the automatic nonlinear mode does not work well. The best practice is to switch to *Nonlinear* mode first and then to *Custom nonlinear*, so that the chart will continuously refresh as the parameters change. At the extreme, you can set the parameters so that the nonlinear mode becomes equivalent to linear mode or step mode.

You can switch between timeline modes using the button on the toolbar or from the context menu.

8.3.3. Zero Simulation Time Regions

It is quite common in simulation models for multiple events to occur at the same simulation time, possibly in different modules. A region with a gray background indicates that the simulation time does not change along the horizontal axis within the area, thus all events inside it have the same simulation time associated with them.

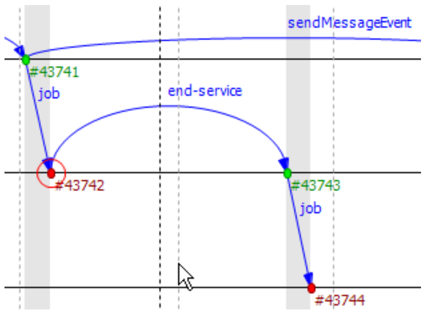


Figure 8.2. Nonlinear simulation time

8.3.4. Module Axes

The Sequence Chart's vertical axis corresponds to modules in the simulation. By default, each simple module is displayed on a separate horizontal axis and events that occurred in that module are shown as circles on it. A compound module is represented with a double line and it will display events from all contained simple modules, except internal events and those that have their own axes displayed. An event is internal to a compound module if it only processes a message from, and sends out messages to, other modules inside.

It is not uncommon for some axes to not have any events at all. These axes would waste space by occupying some place on the screen, so by default they are omitted from the chart unless the *Show Axes Without Events* option is turned on. The discovery process is done lazily as you navigate through the chart, and it may add new axes dynamically as soon as it turns out that they actually have events.

Module axes can be reordered with the option *Axis Ordering Mode*. Ordering can be manual, or sorted by module name, by module id or by minimizing the total number of axes that arrows cross.

> **NOTE**
> The algorithm that minimizes crossings works by taking a random sample from the file and determines the order of axes from that (which means that the resulting order will only be an approximation). A more precise algorithm, which takes all arrows into account would not be practical because of the typically large size of eventlog files.

8.3.5. Gutter

The upper and lower edges of the Sequence Chart show a gutter that displays the simulation time. The left side of the top gutter displays a *time prefix* value, which should be added to each individual simulation time shown at the vertical hairlines. This reduces the number of characters on the gutter and allows easier recognition of simulation time changes in the significant digits. The right side of the figure displays the simulation time range that is currently visible within the window.

> To see the simulation time at a specific point on the chart, move the mouse to the desired place and read the value in the blue box horizontally aligned with the mouse on the gutter.

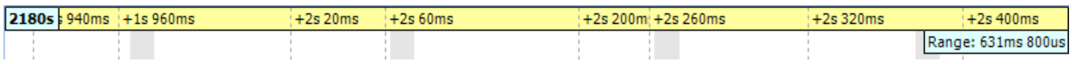


Figure 8.3. Gutter and range

8.3.6. Events

Events are displayed as filled circles along the module axes. A green circle represents the processing of a self-message, while a red circle is an event caused by receiving a message from another module. The event with event number zero represents the module initialization phase and may spread across multiple module axes because the simulation kernel calls each module during initialization. This event is displayed with a white background.

Event numbers are displayed below and to the right of their corresponding events and are prefixed with '#'. Their color changes according to their events' colors.

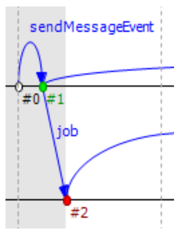


Figure 8.4. Various event kinds

8.3.7. Messages

The Sequence Chart represents message sends with blue arrows. Vertically, the arrow starts at the module which sent the message and ends at the module which processed the message. Horizontally, the start and end points of the arrow correspond to the sender and receiver events. The message name is displayed near the middle of the arrow, but not exactly in the middle to avoid overlapping with other names between the same modules.

Sometimes, when a message arrives at a module, it simply stores it and later sends the very same message out. The events, where the message arrived, and where the message was actually sent, are in a so-called "message reuse" relationship. This is represented by a green dotted arrow between the two events. These arrows are not shown by default because timer self-messages are usually reused continuously. This would add unnecessary clutter to the chart and would make it hard to understand. To show and hide these arrows, use the *Show Reuse Messages* button on the toolbar.

Sometimes, depending on the zoom factor, a message send goes far away on the chart. In this case, the line is split into two smaller parts that are displayed at the two ends pointing towards each other, but without a continuous line connecting them. At one end of both arrow pieces is a dotted line while at the other end is a solid line. The one which is solid always points exactly to, or from, the event to which it is connected. The other one, which is dotted, either specifies the module where the arrow starts, or ends, or in the case of a self-message, it points toward the other arrow horizontally.

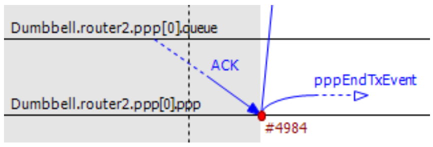


Figure 8.5. Split arrows

8.3.8. Attaching Vectors

It is possible to attach Vector data to individual axes from the context menu by right-clicking on the desired axis and selecting *Attach Vector to Axis* from the corresponding submenu. In this case, the solid line of the axis will be turned into a colored thick bar. If the vector is of type enum and its element names have been registered with the C++ macro Register_Enum, then the chart will display those names inside the bar. Otherwise, it will display the value as a number. The background coloring for the thick bar is automatic; it is either based on the enumeration member, or alternates for other types.

8.3.9. Zooming

To zoom in or out horizontally along the timeline, use the *Zoom In* and *Zoom Out* buttons on the toolbar. To decrease or increase the distance between the axes, use the *Increase/Decrease Spacing* commands.

> When you zoom out, more events and messages become visible on the chart, making it slower. When you zoom in, message lines start break, making it less informative. Try to keep a reasonable zoom level.

8.3.10. Navigation

To scroll through the Sequence Chart, use either the scroll bars, drag with the left mouse button or scroll with the mouse wheel using the **SHIFT** modifier key for horizontal scroll.

There are also navigation options to go to the previous (**SHIFT+LEFT**) or next (**SHIFT+RIGHT**) event in the same module.

Similar to navigating in the Eventlog Table, to go to the cause event, press **CTRL+LEFT**. To go to the arrival of a message send, press **CTRL+RIGHT** while an event is selected.

8.3.11. Tooltips

The Sequence Chart displays tooltips for axes, events, message sends and reuses. When a tooltip is shown for any of the above, the chart will highlight the corresponding parts. Sometimes, when the chart is zoomed out it might show a complex tooltip immediately because there are multiple items under the mouse.

> To measure the simulation time difference between two events, select one of them while staying at the other to display the tooltip.

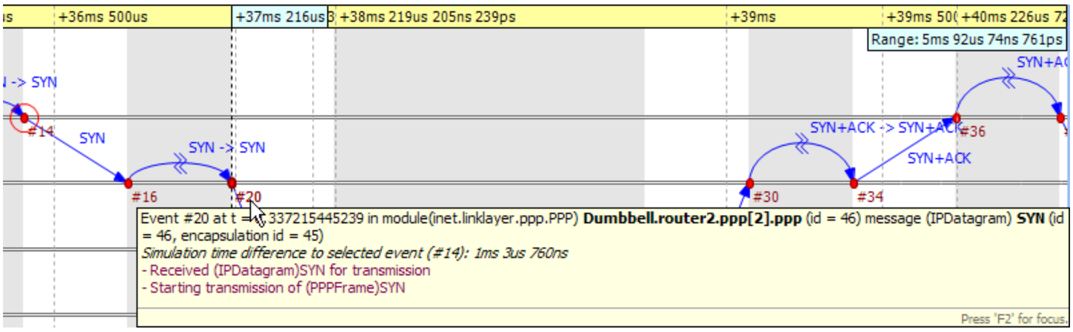


Figure 8.6. Event tooltip

8.3.12. Bookmarks

Just like the Eventlog Table, the Sequence Chart also supports bookmarks to make navigation easier. Bookmarks are saved for the files rather than the various editors, therefore they are shared between them. The chart highlights bookmarked events with a circle around them similar to primary selection but with a different color.

8.3.13. Associated Views

When you open an eventlog file in the Sequence Chart editor, it will automatically open the *Eventlog Table View* with the same file. If you select an event on the Sequence Chart editor, then the *Eventlog Table View* will jump to the same event and vice versa. This interconnection makes navigation easier and you can immediately see the details of the selected event's raw data.

8.3.14. Filtering

You can also filter the contents of the Sequence Chart. This actually means that some of the events are not displayed on the chart so that the user can focus on the relevant parts. When filtering is turned on (displayed in the status line), some of the message arrows might have a filter sign (a double zigzag crossing the arrow line's center). Such a message arrow means that there is a message going out from the source module, which after processing in some other filtered out modules, reaches the target module. The message name of the arrow in this case corresponds to the first and the last message in the chain that was filtered out.

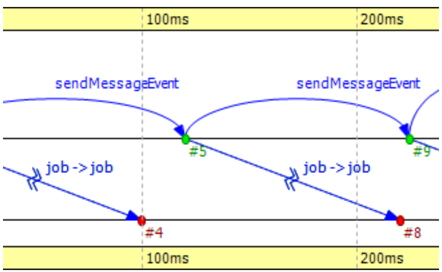


Figure 8.7. Zigzag arrows

When a module filter is used, it will determine which modules will have axes. If the events that occurred in a module are completely filtered out, then the Sequence

Chart will not display the superfluous axis belonging to that module. This reduces the number of axes and makes it easier to understand the figure.

Events may not have subsequent event numbers, which means that the events in between have been filtered out. At the extreme, the chart may even be empty, meaning that there are no matching events at all.

To filter the Sequence Chart, open the *Filter Dialog* using the filter button on the toolbar. You can also filter from the context menu using the shortcuts provided for events and message sends currently under the mouse.

8.4. Eventlog Table

This section describes the Eventlog Table in details focusing on its features without a particular example.

The Eventlog Table has one row per line in the eventlog file. It has three columns. The first two are called event number and simulation time respectively. They show the values corresponding to the simulation event where the line was recorded. The third column, called details, contains the actual data, which varies for each line kind. The different kinds of lines can be easily recognized by their icons. Some lines, such as sending a message through a sequence of gates, relate to each other and are indented so that the user can recognize them more easily.

There are various options, which control how and what the Eventlog Table displays. Some of these are available on the toolbar, while others are accessible only from the context menu.

8.4.1. Display Mode

The eventlog file content may be displayed in two different notations. The *Raw* data notation shows exactly what is present in the file.

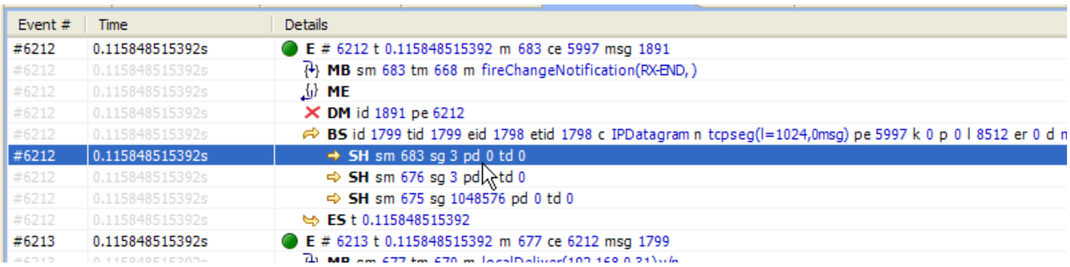


Figure 8.8. *Raw* notation

The *Descriptive* notation, after some preprocessing, displays the log file in a readable format. It also resolves references and types, so that less navigation is required to understand what is going on. To switch between the two, use the *Display Mode* button on the toolbar or the context menu.

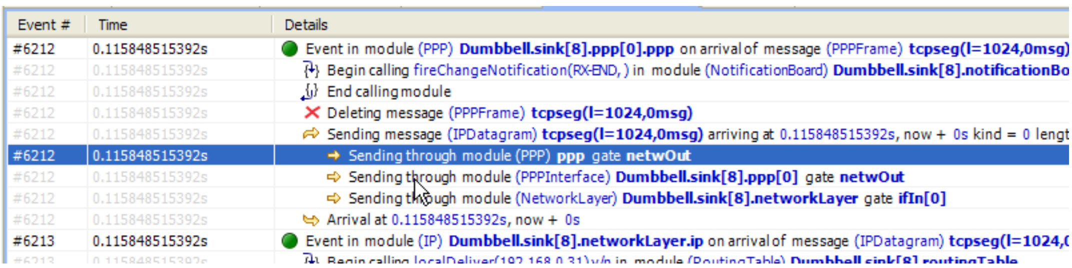


Figure 8.9. *Descriptive* notation

8.4.2. Name Mode

There are three different ways to display names in the Eventlog Table; it is configurable with the *Name Mode* option. Full path and full name shows what you would expect. The smart mode uses the context of the line to decide whether a full path or a full name should be displayed. For each event line, this mode always displays the full path. For all other lines, if the name is the same as the enclosing event's module name, then it shows the full name only. This choice makes lines shorter and allows for faster reading.

8.4.3. Type Mode

The option called *Type Mode* can be used to switch between displaying the C++ class name or the NED type name in parenthesis before module names. This is rarely used, so it is only available from the context menu.

8.4.4. Line Filter

The Eventlog Table may be filtered by using the *Line Filter* button on the toolbar. This option allows filtering for lines with specific kinds. There are some predefined filters.

You can also provide a custom filter pattern, referring to fields present in *Raw* mode, using a match expression. The following example is a custom filter, which will show message sends where the message's class is AirFrame.

```
BS and c(AirFrame)
```

Please refer to the OMNeT++ Manual for more details on match expressions.

> NOTE: To avoid confusion, event lines marked with green circles are always shown in the Eventlog Table and are independent of the line filter.

8.4.5. Navigation

You can navigate using your keyboard and mouse just like in any other table. There are a couple of non-standard navigation options in the context menu, which can also be used with the keyboard.

The simplest are the *Goto Event* and the *Goto Simulation Time*, both of which simply jump to the designated location.

There are navigation options for going to the previous (**ALT+UP**) or next (**ALT+DOWN**) event in general, and to go to the previous (**SHIFT+UP**) or next (**SHIFT+DOWN**) event in the same module.

Some of the navigation options focus on the causes of events and consequences of message sends. To go to the cause event, press **CTRL+UP**. To go to the arrival of a message send, press **CTRL+DOWN**, after selecting the message being sent.

Finally, there are navigation options for message reuse relationships. You can go to the original event of a message from the line where it was being reused. In the other direction, you can go to the reused event of a message from the event where it was received. These options are enabled only if they actually make sense for the current selection.

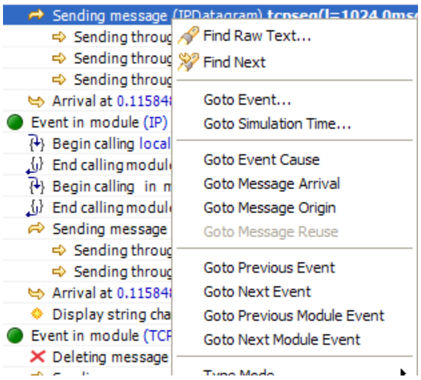


Figure 8.10. Navigation context menu

8.4.6. Selection

The Eventlog Table uses multiple selection even though most of the user commands require single selection.

8.4.7. Searching

For performance reasons, the search function works directly on the eventlog file and not the text displayed in the Eventlog Table. It means that some static text present in *Descriptive* mode cannot be found. Usually, it is easier to figure out what to search for in *Raw* mode, where the eventlog file's content is directly displayed. The search can work in both directions, starting from the current selection, and may be case insensitive. To repeat the last search, use the *Find Next* command.

8.4.8. Bookmarks

For easier navigation, the Eventlog Table supports navigation history. This is accessible from the standard IDE toolbar just like for other kinds of editors. It works by remembering each position where the user stayed more than 3 seconds. The navigation history is temporary and thus it is not saved when the file is closed.

Persistent bookmarks are also supported and they can be added from the context menu. A Bookmarked event is highlighted with a different background color.

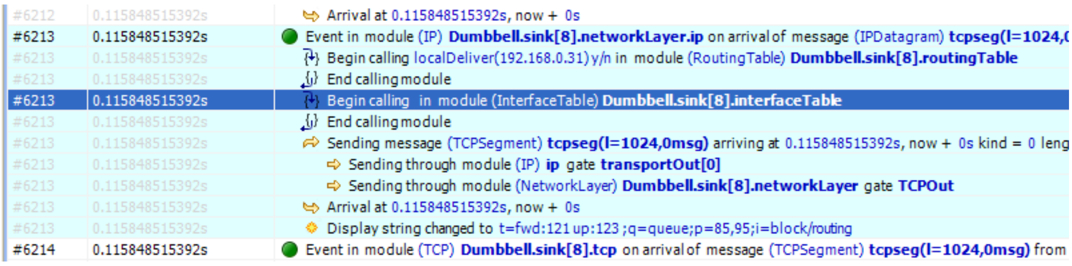


Figure 8.11. A bookmark

To jump to a bookmark, use the standard Bookmark View (this is possible even after restarting the IDE).

8.4.9. Tooltips

Currently, only the message send lines have tooltips. If message detail recording was configured for the simulation, then a tooltip will show the recorded content of a message send over the corresponding line.

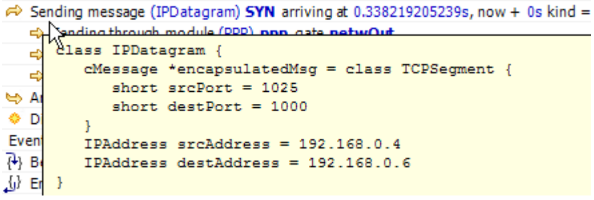


Figure 8.12. A message send tooltip

8.4.10. Associated Views

When you open an eventlog file in the Eventlog Table editor, it will automatically open the *Sequence Chart View* with the same file. If you select an event on the Eventlog Table editor, then the *Sequence Chart View* will jump to the same event and vice versa. This interconnection makes navigation easier, and you can immediately see the cause and effect relationships of the selected event.

8.4.11. Filtering

If the Eventlog Table displays a filtered eventlog, then subsequent events may not have subsequent event numbers. This means that the events in between have been filtered out. At the extreme, the table may even be empty, which means that there are no matching events at all.

8.5. Filter Dialog

The content of an eventlog can be filtered within the OMNeT++ IDE. This is on-the-fly filtering as opposed to the file content filtering provided by the `Eventlog` tool. To use on the fly filtering, open the filter configuration dialog with the button on the toolbar, enable some of the range, module, message, or trace filters, set the various filter parameters, and apply the settings. The result is another eventlog, resident in memory, that filters out some events.

> **NOTE:** Similar to the command line `Eventlog` tool described in the OMNeT++ Manual, the in-memory filtering can only filter out whole events.

In-memory, on-the-fly filtering means that the filter's result is not saved into an eventlog file, but it is computed and stored within memory. This allows rapid switching between different views of the same eventlog within both the Sequence Chart and the Eventlog Table.

The filter configuration dialog shown in Figure 8.13, " Filter Dialog "has many options. They are organized into a tree with each part restricting the eventlog's content. The individual filter components may be turned on and off independent of each other. This allows remembering the filter settings even if some of them are temporarily unused.

The combination of various filter options might be complicated and hard to understand. To make it easier, the *Filter Dialog* automatically displays the current filter in a human readable form at the bottom of the dialog.

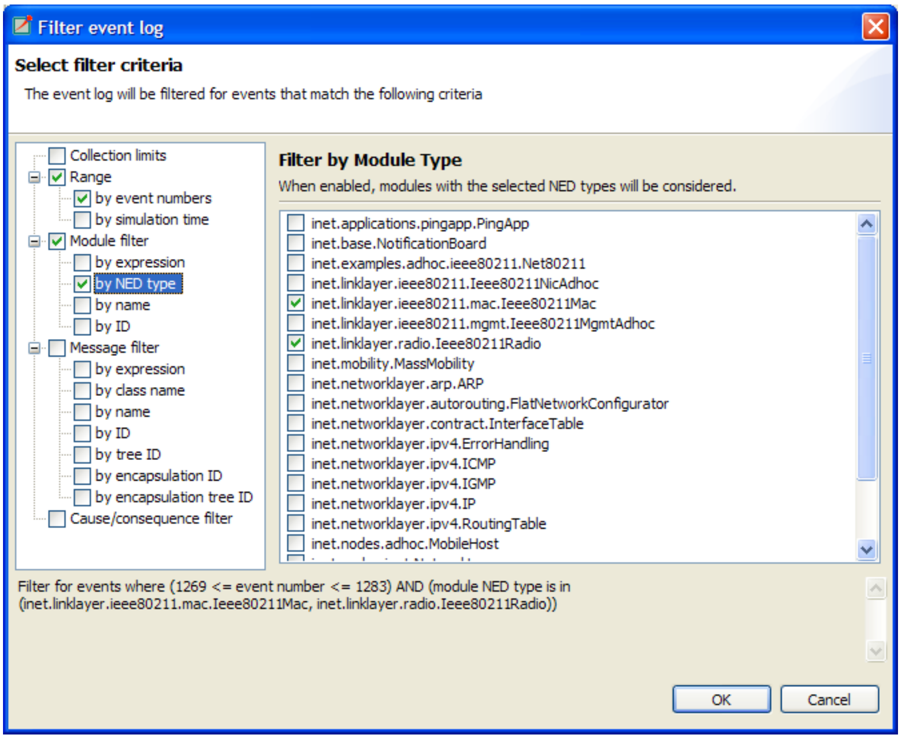


Figure 8.13. Filter Dialog

8.5.1. Range Filter

This is the simplest filter, which filters out events from the beginning and end of the eventlog. It might help to reduce the computation time dramatically when defining filters, which otherwise would be very expensive to compute for the whole eventlog file.

8.5.2. Module Filter

With this kind of filter, you can filter out events that did not occur in any of the specified modules. The modules which will be included in the result can be selected by their NED type, full path, module id, or by a match expression. The expression may refer to the raw data present in the lines marked with 'MC' in the eventlog file.

8.5.3. Message Filter

This filter is the most complicated one. It allows filtering for events, which either process or send specific messages. The messages can be selected based on their C ++ class name, message name, various message ids, and a match expression. The expression may refer to the raw data present in the lines marked with 'BS' in the eventlog file.

There are four different message ids to filter, each with different characteristics. The most basic one is the id, which is unique for each constructed message independent of how it was created. The tree id is special because it gets copied over when a message is created by copying (duplicating) another. The encapsulation id is different in that it gives the id of the innermost encapsulated message. Finally, the encapsulation tree id combines the two by providing the innermost encapsulated message's tree id.

8.5.4. Tracing Causes/Consequences

The trace filter allows filtering for causes and consequence of a particular event specified by its event number. The cause/consequence relation between two events means that there is a message send/reuse path from the cause event to the consequence event. If there was a message reuse in the path, then the whole path is considered to be a message reuse itself.

Since computing the causes and consequences in an eventlog file that is far away from the traced event might be a time consuming task, the user can set extra range limits around the traced event. These limits are separate from the range filter due to being relative to the traced event. This means that if you change the traced event, there is no need to change the range parameters. It is strongly recommended that users provide these limits when tracing events to avoid long running operations.

8.5.5. Collection Limits

When an in-memory filter is applied to an eventlog, it does not only filter out events, but it also provides automatic discovery for virtual message sends. It means that two events far away, and not directly related to each other, might have a virtual message send (or reuse) between them. Recall that there is a virtual message send (or reuse) between two events if and only if there is a path of message sends (or reuses) connecting the two.

The process of collecting these virtual message dependencies is time consuming and thus has to be limited. There are two options. The first one limits the number of virtual message sends collected per event. The other one limits the depth of cause/consequence chains during collection.

8.5.6. Long-Running Operations

Sometimes, computing the filter's result takes a lot of time, especially when tracing causes/consequences without specifying proper range limits in terms of event numbers or simulation times. If you cancel a long running operation, you can go back to the *Filter Dialog* to modify the filter parameters, or simply turn the filter off. To restart drawing, use the refresh button on the toolbar.

Providing a proper range filter is always a good idea to speed up computing the filter's result.

8.6. Other Features

Both the Sequence Chart and the Eventlog Table tools can be used as an editor and also as a view. The difference between an editor or a view is quite important because there is only at most one instance of a view of the same kind. It means that even if multiple eventlog files are open in Sequence Chart editors, there is no more than one *Eventlog Table* view shared between them. This single view will automatically display the eventlog file of the active editor. It will also remember its position and state when it switches among editors. For more details on editors and views, and their differences, please refer to the Eclipse documentation.

Despite the name "editor", which is a concept of the Eclipse platform, neither the Sequence Chart, nor the Eventlog Table can be used to actually change the contents of an eventlog file.

It is possible to open the same eventlog file in multiple editors and to navigate to different locations, or use different display modes or filters in a location. Once an eventlog is open in an editor, you can use the *Window|New Editor* to open it again.

Dragging one of the editors from the tabbed pane to the side of the editor's area allows you to interact with the two simultaneously.

8.6.1. Settings

There are various settings for both tools which affect the display, such as display modes, content position, filter parameters, etc. These user-specified settings are automatically saved for each file and they are reused whenever the file is revisited. The per file settings are stored under the OMNeT++ workspace, in the directory `.metadata\.plugins\org.eclipse.core.resources\.projects\<project-name>`.

8.6.2. Large File Support

Since an eventlog file might be several Gigabytes, both tools are designed in a way that allows for efficient displaying of such a file without requiring large amounts of physical memory to load it at once. As you navigate through the file, physical memory is filled up with the content lazily. Since it is difficult to reliably identify when the system is getting low on physical memory, it is up to the user to release the allocated memory when needed. This operation, although usually not required, is available from the context menu as *Release Memory*. It does not affect the user interface in any way.

The fact that the eventlog file is loaded lazily and optionally filtered also means that the exact number of lines and events it contains cannot be easily determined. This affects the way scrollbars work in the lazy directions: horizontal for the Sequence Chart and vertical for the Eventlog Table. These scrollbars act as a non-linear approximation in that direction. For the most, the user will be unaware of these approximations unless the file is really small.

8.6.3. Viewing a Running Simulation's Results

Even though the simulation kernel keeps the eventlog file open for writing while the simulation is running, it may be open in the OMNeT++ IDE simultaneously. Both tools can be guided by pressing the **END** key to follow the eventlog's end as new content is appended to it. If you pause the simulation in the runtime environment, then after a few seconds the tools will refresh their content and jump to the very end. This process makes it possible to follow the simulation step-by-step on the Sequence Chart.

8.6.4. Caveats

Sometimes, drawing the Sequence Chart may take a lot of time. Zooming out too much, for example, might result in slow response times. A dialog might pop up telling the user that a long running eventlog operation is in progress. You can safely cancel these operations at any time you like, or just wait until they finish. To restart the rendering process, simply press the refresh button on the toolbar. Before refreshing, it is a good idea to revert to some defaults (e.g. default zoom level) or revert the last changes (e.g. navigate back, turn filter off, etc.).

> An operation which runs for an unreasonably long time might be a sign of a problem that should be reported.

8.7. Examples

This section will guide you through the use of the Sequence Chart and Eventlog Table tools, using example simulations from OMNeT++ and the INET Framework. Before running any of the simulations, make sure that eventlog recording is enabled by adding the line

```
record-eventlog = true
```

in the `omnetpp.ini` file in the simulation's directory. To open the generated eventlog in the OMNeT++ IDE, go to the example's `results` directory in the *Resource Navigator* View, and double-click the log file. By default, the file will open in the Sequence Chart.

To open the file in the Eventlog Table as editor, right-click the file, and choose the corresponding item from the context menu's *Open With* submenu.

8.7.1. Tictoc

The Tictoc example is available in the OMNeT++ installation under the directory `samples/tictoc`. Tictoc is the most basic example in this chapter and it provides a quick overview on how to use and understand the Sequence Chart.

Start the simulation and choose the simplest configuration, 'Tictoc1', which specifies only two nodes called 'tic' and 'toc.' During initialization, one of the nodes will send a message to the other. From then on, every time a node receives the message, it will simply send it back. This process continues until you stop the simulation. In Figure 8.14, “ Tictoc with two nodes ”you can see how this is represented on a Sequence Chart. The two horizontal black lines correspond to the two nodes and are labeled 'tic' and 'toc.' The red circles represent events and the blue arrows represent message sends. It is easy to see that all message sends take 100 milliseconds and that the first sender is the node 'tic.'

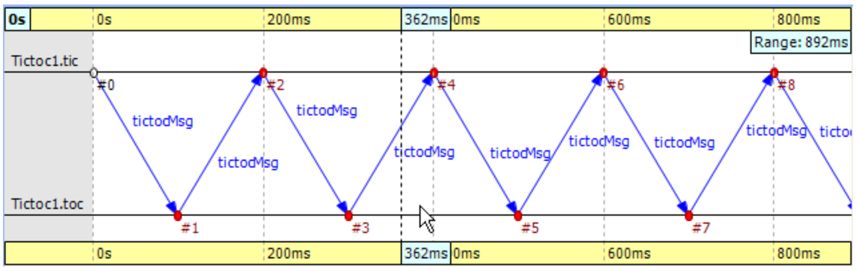


Figure 8.14. Tictoc with two nodes

In the next Tictoc example, there are six nodes tossing a message around until it reaches its destination. To generate the eventlog file, restart the simulation and choose the configuration 'Tictoc9'. In Figure 8.15, “ Tictoc with six nodes ”you can see how the message goes from one node to another, starting from node '0' and passing through it twice more, until it finally reaches its destination, node '3.' The chart also shows that this example, unlike the previous one, starts with a self-message instead of immediately sending a message from initialize to another node.

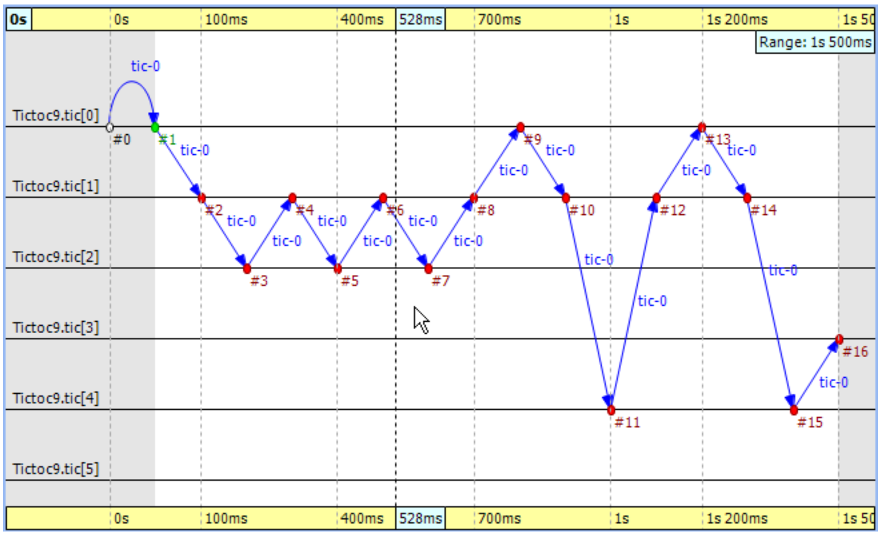


Figure 8.15. Tictoc with six nodes

Let us demonstrate with this simple example how filtering works with the Sequence Chart. Open the *Filter Dialog* with the toolbar button and put a checkmark for node '0' and '3' on the *Module filter|by name* panel, and apply it. The chart now displays only two axes that correspond to the two selected nodes. Note that the arrows on this figure are decorated with zigzags, meaning that they represent a sequence of message sends. Such arrows will be called virtual message sends in the rest of this chapter. The first two arrows show the message returning to node '0' at event #9 and event #13, and the third shows that it reaches the destination at event #16. The events where the message was in between are filtered out.

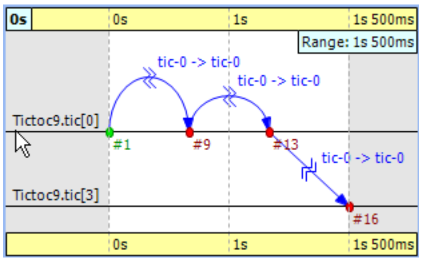


Figure 8.16. Filtering for node '0' and '3'

8.7.2. FIFO

The FIFO example is available in the OMNeT++ installation under the directory `samples/fifo`. The FIFO is an important example because it uses a queue, which is an essential part of discrete event simulations and introduces the notion of message reuses.

When you start the simulation, choose the configuration 'low job arrival rate' and let it run for a while. In Figure 8.17, “ The FIFO example ”you can see three modules: a 'source', a 'queue', and a 'sink.' The simulation starts with a self-message and then the generator sends the first message to the queue at event #1. It is immediately obvious that the message stays in the queue for a certain period of time, between event #2 and event #3.

> When you select one event and hover with the mouse above the other, the Sequence Chart will show the length of this time period in a tooltip.

Finally, the message is sent to the 'sink' where it is deleted at event #4.

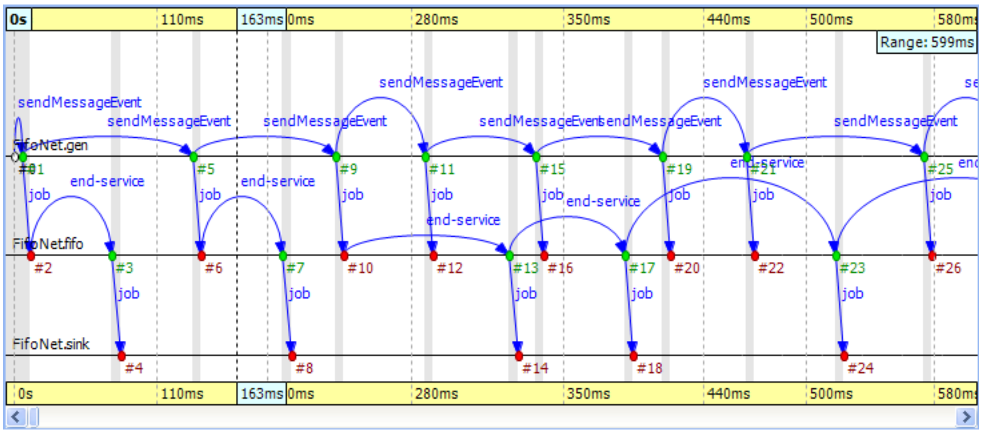


Figure 8.17. The FIFO example

Something interesting happens at event #12 where the incoming message suddenly disappears. It seems like the queue does not send the message out. Actually, what happens is that the queue enqueues the job because it is busy serving the message received at event #10. Since this queue is a FIFO, it will send out the first message at event #13. To see how this happens, turn on *Show Reuse Messages* from the context menu; the result is shown in Figure 8.18, " Showing reuse messages ". It displays a couple of green dotted arrows, one of which starts at event #12 and arrives at event #17. This is a reuse arrow; it means that the message sent out from the queue at event #17 is the same as the one received and enqueued at event #12. Note that the service of this message actually begins at event #13, which is the moment that the queue becomes free after it completes the job received at event #10.

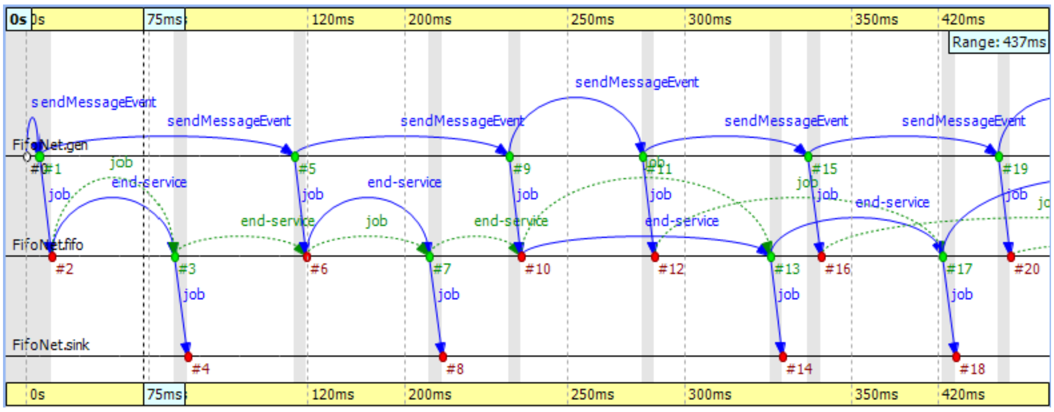


Figure 8.18. Showing reuse messages

Another type of message reuse is portrayed with the arrow from event #3 to event #6. The arrow shows that the queue reuses the same timer message instead of creating a new one each time.

NOTE

Whenever you see a reuse arrow, it means that the underlying implementation remembers the message between the two events. It might be stored in a pointer variable, a queue, or some other data structure.

The last part of this example is about filtering out the queue from the chart. Open the *Filter Dialog*, select 'sink' and 'source' on the *Module filter|by NED type* panel, and apply the change in settings. If you look at the result in Figure 8.19, “ Filtering the queue ”, you will see zigzag arrows going from the 'source' to the 'sink.' These arrows show that a message is being sent through the queue from 'source' to 'sink.' The first two arrows do not overlap in simulation time, which means the queue did not have more than one message during that time. The third and fourth arrows do overlap because the fourth job reached the queue while it was busy with the third one. Scrolling forward you can find other places where the queue becomes empty and the arrows do not overlap.

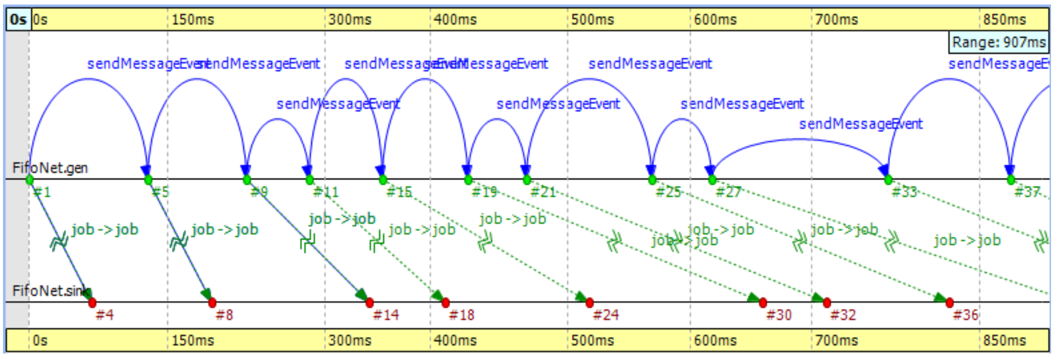


Figure 8.19. Filtering the queue

8.7.3. Routing

The Routing example is available in the OMNeT++ installation under the directory `samples/routing`. The predefined configuration called 'Net10' specifies a network with 10 nodes with each node having an application, a few queues and a routing module inside. Three preselected nodes, namely the node '1,' '6,' and '8' are destinations, while all nodes are message sources. The routing module uses the shortest path algorithm to find the route to the destination. The goal in this example is to create a sequence chart that shows messages which travel simultaneously from multiple sources to their destinations.

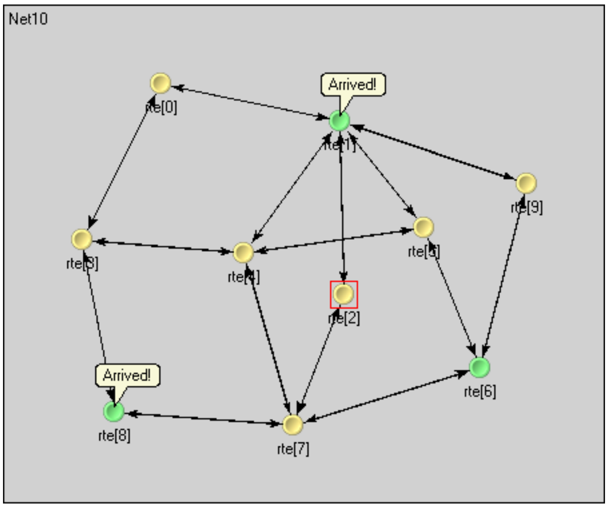


Figure 8.20. Network with 10 nodes

Since we do not care about the details regarding what happens within nodes, we can simply turn on filtering for the NED type 'node.Node.' The chart will have 10 axes with each axis drawn as two parallel solid black lines close to each other. These are the compound modules that represent the nodes in the network. So far events could be directly drawn on the simple module's axis where they occurred, but now they will be drawn on the compound module's axis of their ancestor.

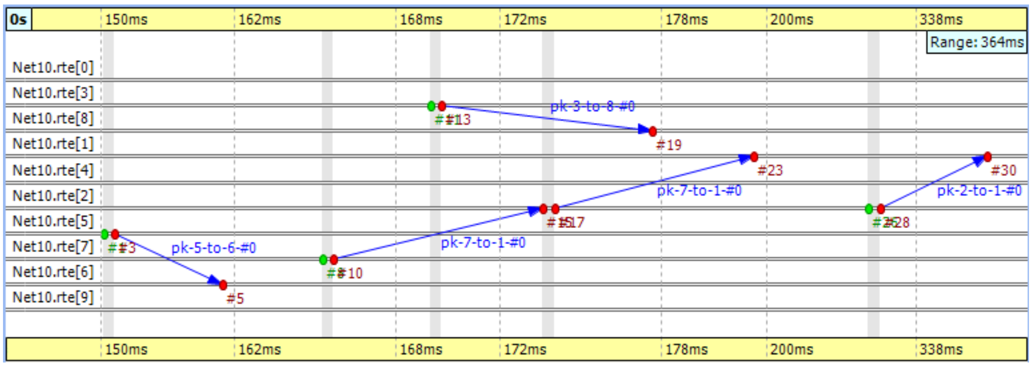


Figure 8.21. Filtering for nodes

To reduce clutter, the chart will automatically omit events which are internal to a compound module. An event is internal to a compound module if it only processes a message from, and sends out messages to, other modules inside the compound module.

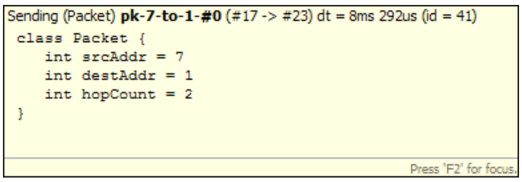
If you look at Figure 8.21, " Filtering for nodes "you will see a message going from node '7' at event #10 to node '1' at event #23. This message stays in node '2' between event #15 and event #17. The gray background area between them means that zero simulation time has elapsed (i.e. the model does not account for processing time inside the network nodes).

> NOTE: This model contains both finite propagation delay and transmission time; arrows in the sequence chart correspond to the interval between the start of the transmission and the end of the reception.

This example also demonstrates message detail recording configured by

```
eventlog-message-detail-pattern = Packet:declaredOn(Packet)
```

in the INI file. The example in Figure 8.22, " Message detail tooltip "shows the tooltip presented for the second message send between event #17 and event #23.


Sending (Packet) **pk-7-to-1-#0** (#17 -> #23) dt = 8ms 292us (id = 41)

```
class Packet {
    int srcAddr = 7
    int destAddr = 1
    int hopCount = 2
}
```

Press 'F2' for focus.

Figure 8.22. Message detail tooltip

It is very easy to find another message on the chart that goes through the network parallel in simulation time. The one sent from node '3' at event #13 to node '8' arriving at event #19 is such a message.

8.7.4. Wireless

The Wireless example is available in the INET Framework under the directory `examples/adhoc/ieee80211`. The predefined configuration called 'Config1' specifies two mobile hosts moving around on the playground and communicating via the IEEE 802.11 wireless protocol. The network devices are configured for ad-hoc mode and the transmitter power is set so that hosts can move out of range. One of the hosts is continuously pinging the other.

In this section, we will explore the protocol's MAC layer, using two sequence charts. The first chart will show a successful ping message being sent through the wireless channel. The second chart will show ping messages getting lost and being continuously re-sent.

We also would like to record some message details during the simulation. To perform that function, comment out the following line from `omnetpp.ini`:

```
eventlog-message-detail-pattern = *:(not declaredOn(cMessage) and not
  declaredOn(cNamedObject) and not declaredOn(cObject))
```

To generate the eventlog file, start the simulation environment and choose the configuration 'host1 pinging host0.' Run the simulation in fast mode until about event #5000.

Preparing the Result

When you open the Sequence Chart, it will show a couple of self-messages named 'move' being scheduled regularly. These are self-messages that control the movement of the hosts on the playground. There is an axis labeled 'pingApp,' which starts with a 'sendPing' message that is processed in an event far away on the chart. This is indicated by a split arrow.

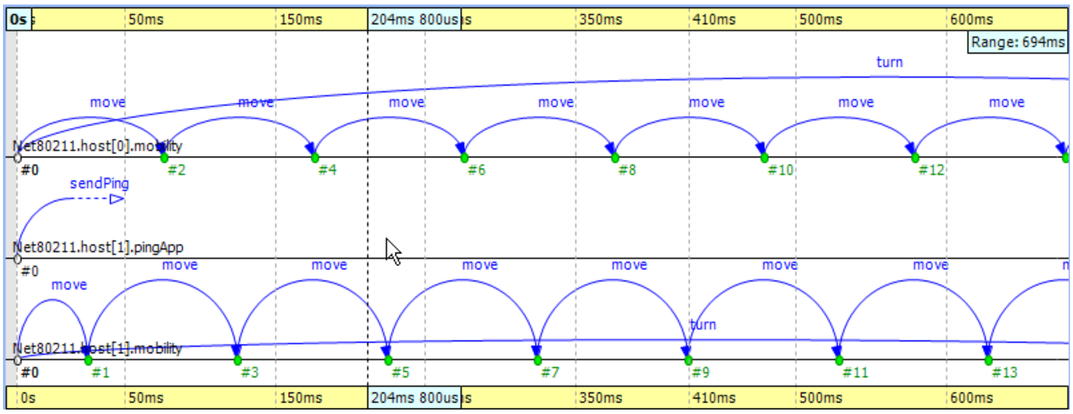


Figure 8.23. The beginning

You might notice that there are only three axes in Figure 8.23, “ The beginning ”even though the simulation model clearly contains more simple modules. This is because the Sequence Chart displays the first few events by default and in this scenario, they all happen to be within those modules. If you scroll forward or zoom out, new axes will be added automatically as needed.

For this example, ignore the 'move' messages and focus on the MAC layer instead. To begin with, open the *Filter Dialog*, select 'Ieee80211Mac' and 'Ieee80211Radio' on the *Module filter|by NED type* panel, and apply the selected changes. The chart will have four axes, two for the MAC and two for the radio simple modules.

The next step is to attach vector data to these axes. Open the context menu for each axis by clicking on them one by one and select the *Attach Vector to Axis* submenu. Accept the vector file offered by default. Then, choose the vector 'mac:State' for the MAC modules and 'mac:RadioState' for the radio modules. You will have to edit the filter in the vector selection dialog (i.e. delete the last segment) for the radio modules because at the moment the radio state is recorded by the MAC module, so the default filter will not be right. When this step is completed, the chart should display four thick colored bars as module axes. The colors and labels on the bars specify the state of the corresponding state machine at the given simulation time.

To aid comprehension, you might want to manually reorder the axis, so that the radio modules are put next to each other. Use the button  on the toolbar to switch to manual ordering. With a little zooming and scrolling, you should be able to fit the first message exchange between the two hosts into the window.

Successful Ping

The first message sent by 'host1' is not a ping request but an ARP request. The processing of this message in 'host0' generates the corresponding ARP reply. This is shown by the zigzag arrow between event #85 and event #90. The reply goes back to 'host1,' which then sends a WLAN acknowledge in return. In this process, 'host1' discovers the MAC address of 'host0' based on its IP address.

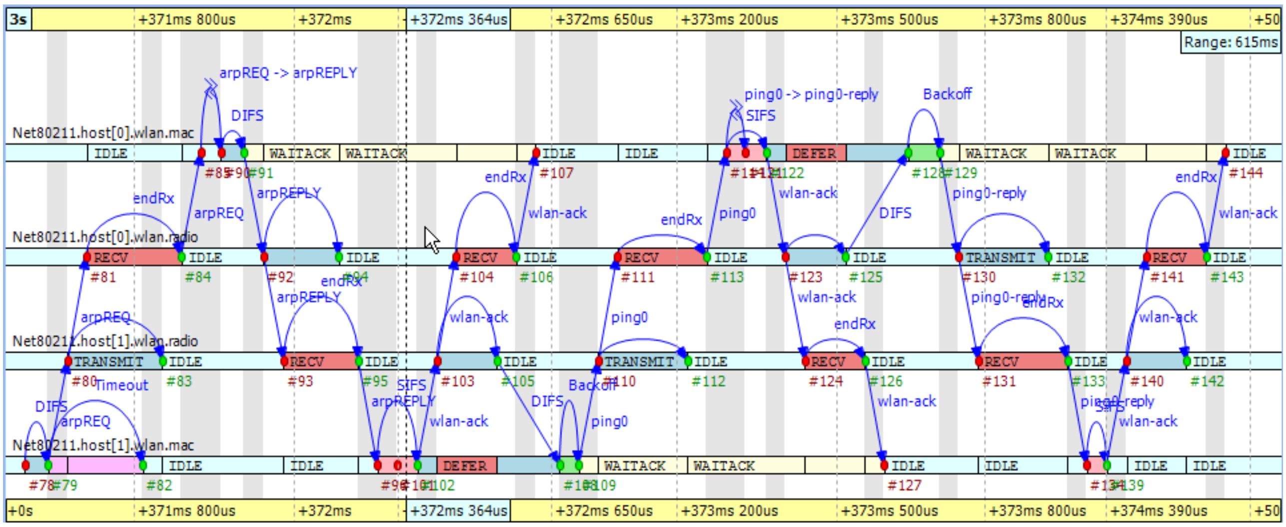


Figure 8.24. Discovering the MAC address

The send procedure for the first ping message starts at event #105 in 'host1' and finishes by receiving the acknowledge at event #127. The ping reply send procedure starts at event #125 in 'host0' and finishes by receiving the WLAN acknowledge at event #144. If you scroll forward, you can see as in Figure 8.25, “ The second ping procedure ” the second complete successful ping procedure between event #170 and event #206. To focus on the second successful ping message exchange, open the *Filter Dialog* and enter these numbers in the range filter.

Timing is critical in a protocol implementation, so we will take a look at it using the Sequence Chart. The first self message represents the fact that the MAC module listens to the radio for a DIFS period before sending the message out. The message send from event #171 to event #172 occurs in zero simulation time as indicated by the gray background. It represents the moment when the MAC module decides to send the ping request down to its radio module. The back-off procedure was skipped for this message because there was no transmission during the DIFS period. If you look at event #172 and event #173, you will see how the message propagates through the air from 'radio1' to 'radio0.' This finite amount of time is calculated from the physical distance of the two modules and the speed of light. In addition, by looking at event #172 and event #174, you will notice that the transmission time is not zero. This time interval is calculated from the message's length and the radio module's bitrate.

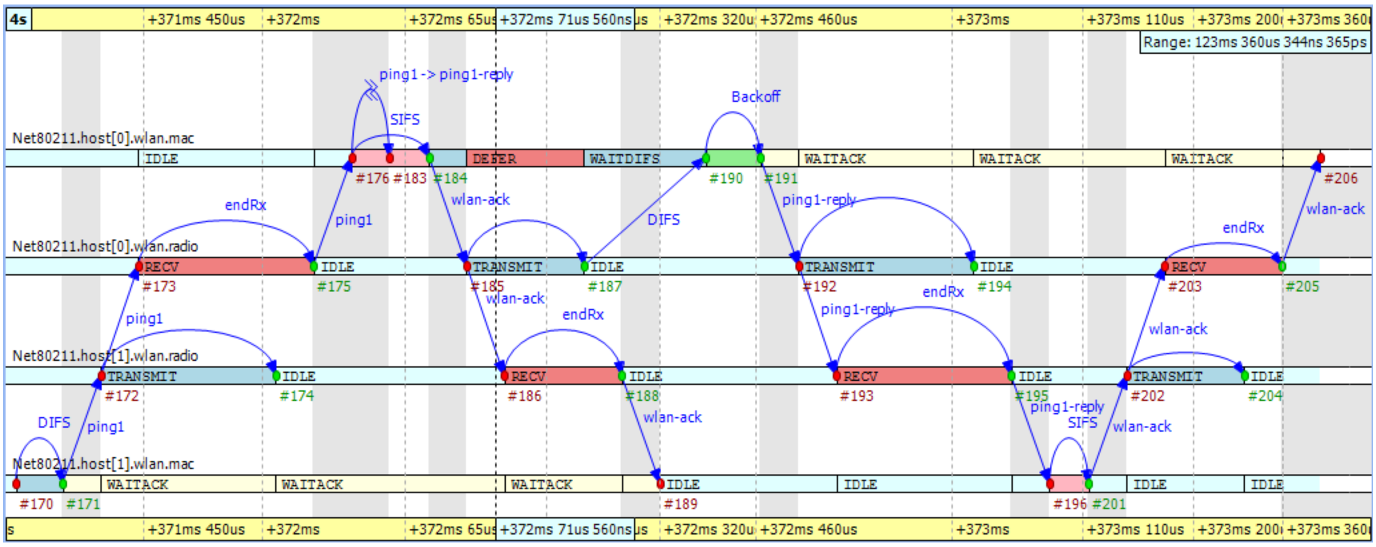


Figure 8.25. The second ping procedure

Another interesting fact seen in the figure is that the higher level protocol layers do not add delay for generating the ping reply message in 'host0' between event #176 and event #183. The MAC layer procedure ends with sending back a WLAN acknowledge after waiting a SIFS period.

Finally, you can get a quick overview of the relative timings of the IEEE 802.11 protocol by switching to linear timeline mode. Use the button  on the toolbar and notice how the figure changes dramatically. You might need to scroll and zoom in or out to see the details. This shows the usefulness of the nonlinear timeline mode.

Unsuccessful Ping

To see how the chart looks when the ping messages get lost in the air, first turn off range filtering. Then, go to event #1269 by selecting the *Goto Event* option from the *Eventlog Table* View's context menu. In Figure 8.26, “ Ping messages get lost ”you can see how the receiver radio does not send up the incoming message to its MAC layer due to the signal level being too low. This actually happens at event #1274 in 'host0.' Shortly thereafter, the transmitter MAC layer in 'host1' receives the timeout message at event #1275, and starts the backoff procedure before resending the very same ping message. This process goes on with statistically increasing backoff time intervals until event #1317. Finally, the maximum number of retries is reached and the message is dropped.

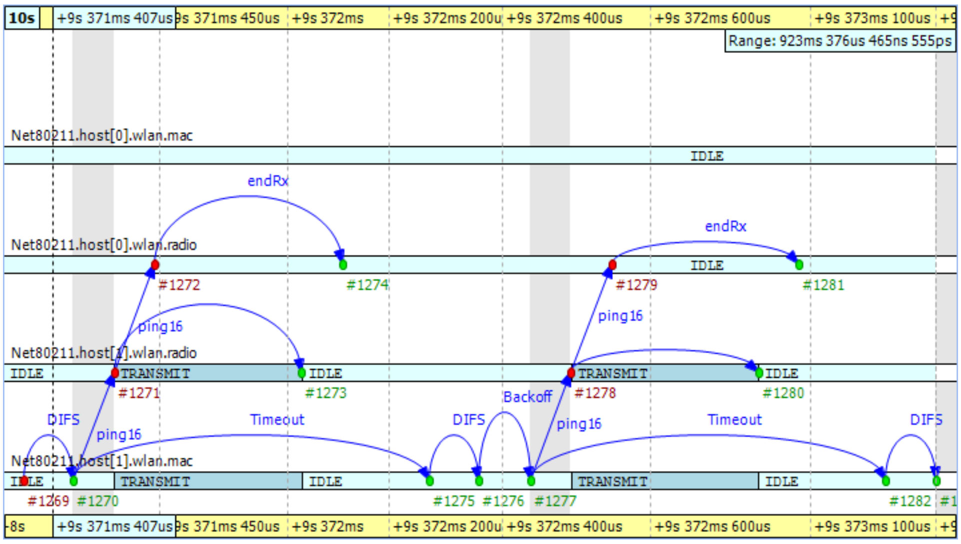


Figure 8.26. Ping messages get lost

The chart also shows that during the unsuccessful ping period, there are no events occurring in the MAC layer of 'host0' and it is continuously in 'IDLE' state.

Chapter 9. Analyzing the Results

9.1. Overview

Analyzing the simulation result is a lengthy and time consuming process. The result of the simulation is recorded as scalar values, vector values and histograms. The user then applies statistical methods to extract the relevant information and to draw a conclusion. This process may include several steps. Usually you need to filter and transform the data, and chart the result. Automation is very important here. You do not want to repeat the steps of recreating charts every time you rerun simulations.

In OMNeT++ 4.x, the statistical analysis tool is integrated into the Eclipse environment. Your settings (i.e. your recipe for finding results from the raw data) will be recorded in analysis files (.anf) and will become instantly reproducible. This means that all processing and charts are stored as datasets; for example, if simulations need to be rerun due to a model bug or misconfiguration, existing charts need not be recreated all over again. Simply replacing the old result files with the new ones will result in the charts being automatically displayed with the new data.

When creating an analysis, the user first selects the input of the analysis by specifying file names or file name patterns (e.g. "adhoc-*.vec"). Data of interest can be selected into datasets using additional pattern rules. The user can define datasets by adding various processing, filtering and charting steps; all using the GUI. Data in result files are tagged with meta information. Experiment, measurement and replication labels are added to the result files to make the filtering process easy. It is possible to create very sophisticated filtering rules (e.g. all 802.11 retry counts of host[5..10] in experiment X, averaged over replications). In addition, datasets can use other datasets as their input so datasets can build on each other.

9.2. Creating Analysis Files

To create a new analysis file, choose *File* | *New* | *Analysis File* from the menu. Select the folder for the new file and enter the file name. Press *Finish* to create and open an empty analysis file.

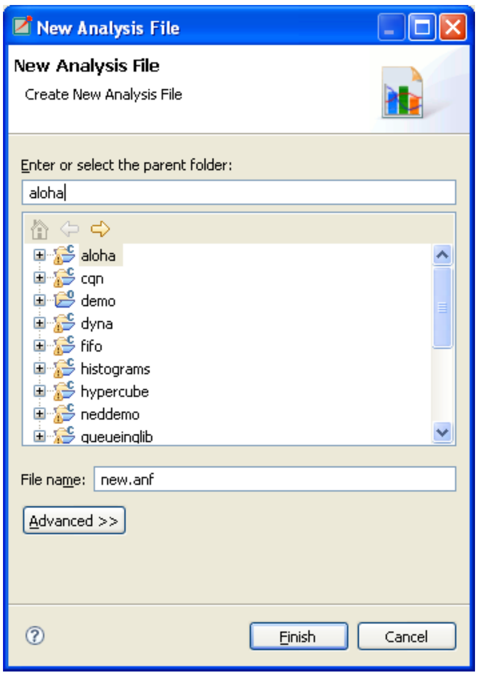


Figure 9.1. New Analysis File dialog

There is a quick way to create an analysis file for a result file. Just double-click on the result file in the *Project Explorer View* to open the *New Analysis File* dialog. The folder and file name is prefilled according to the location and name of the result file. Press *Finish* to create a new analysis file containing the vector and scalar files whose names correspond to the result file. If the name of the result file contains a numeric suffix (e.g. aloha-10.vec), then all files with the same prefix will be added to the analysis file (i.e. aloha-*.vec and aloha-*.sca).

If the analysis file already exists, double-clicking on the result file will open it.

9.3. Using the Analysis Editor

The Analysis Editor is implemented as a multi-page editor. What the editor edits is the "recipe": what result files to take as inputs, what data to select from them, what (optional) processing steps to apply, and what kind of charts to create from them. The pages (tabs) of the editor roughly correspond to these steps.

9.3.1. Input Files

Selecting input files

The first page displays the result files that serve as input for the analysis. The upper half specifies what files to select using explicit filenames or wildcards. New input files can be added to the analysis by dragging vector and scalar files from the *Project Explorer View*, or by opening dialogs with the *Add File...* or *Wildcard...* buttons. If the file name starts with '/,' it is interpreted relative to the workspace root; otherwise, it is relative to the folder of the analysis file.

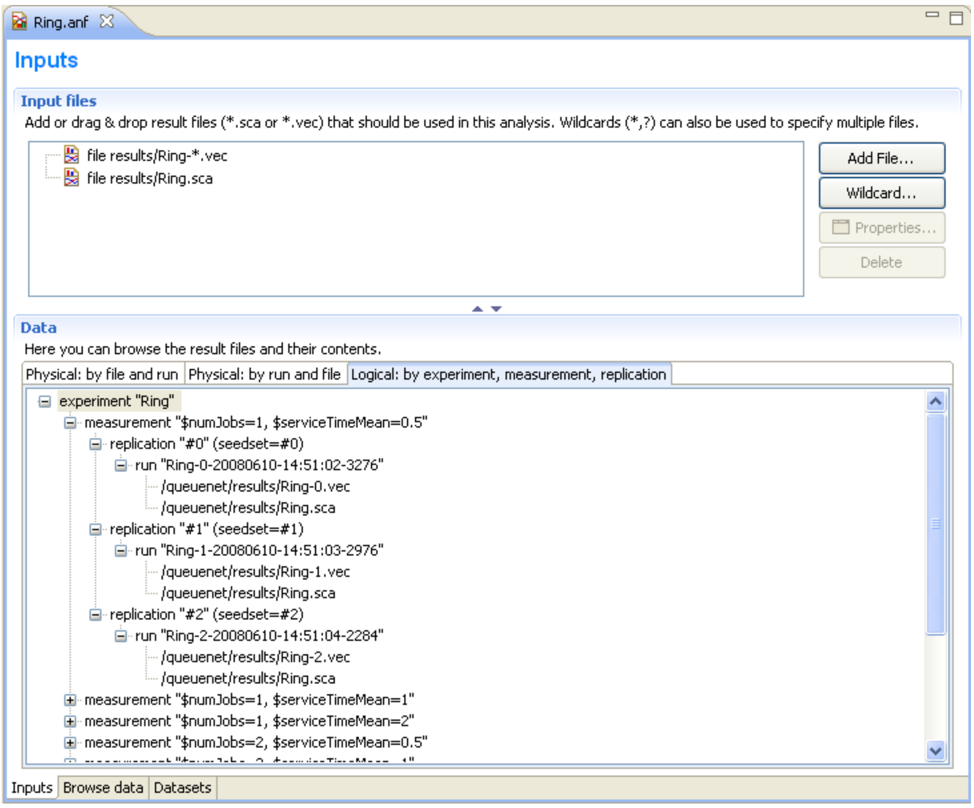


Figure 9.2. Specifying input files for data analysis

The input files are loaded when the analysis file is opened. When the file changes on the disk, it is reloaded automatically when the workspace is refreshed (Eclipse refreshes the workspace automatically if the *General|Workspace|Refresh automatically* option is turned on in the Preferences). Vector files are not loaded directly; instead, an index file is created and the vector attributes (name, module, run, statistics, etc.) are loaded from the index file. The index files are generated during the simulation, but can be safely deleted without loss of information. If the index file is missing or the vector file was modified, the IDE rebuilds the index in the background.

> The *Progress View* displays the progress of the indexing process.

The lower half shows what files matched the input specification and what runs they contain. Note that OMNeT++ 4.x result files contain a unique run ID and several metadata annotations in addition to the actual recorded data. The third tree organizes simulation runs according to their experiment-measurement-replication labels.

The underlying assumption is that users will organize their simulation-based research into various *experiments*. An experiment will consist of several *measurements*, which are typically (but not necessarily) simulations done with the same model but with different parameter settings (i.e. the user will explore the parameter space with several simulation runs). To gain statistical confidence in the results, each measurement may be repeated several times with different random number seeds. It is easy to set up such scenarios with the improved INI files of OMNeT++ 4.x. Then, the experiment-measurement-replication labels will be assigned automatically (Note: please refer to the chapter "Configuring and Running Simulations" in the manual for more discussion).

Browsing input

The second page of the Analysis editor displays results (vectors, scalars and histograms) from all files in tables and lets the user browse them. Results can be sorted and filtered. Simple filtering is possible with combo boxes, or when that is not enough, the user can write arbitrarily complex filters using a generic pattern-matching expression language. Selected or filtered data can be immediately plotted, or remembered in named datasets for further processing.

Pressing the *Advanced* button switches to advanced filter mode. In the text field, you can enter a complex filter pattern.

> You can easily display the data of a selected file, run, experiment, measurement or replication if you double-click on the required tree node in the lower part of the *Inputs* page. It sets the appropriate filter and shows the data on the *Browse Data* page.
>
> If you right-click on a table cell and select the *Set filter: ...* action from the menu, you can set the value of that cell as the filter expression.

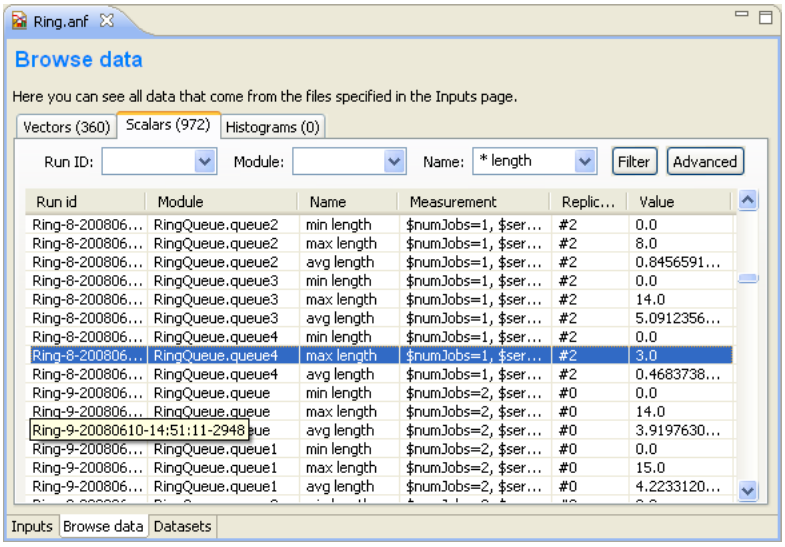


Figure 9.3. Browsing vector and scalar data generated by the simulation

To hide or show table columns, open *Choose table columns...* from the context menu and select the columns to be displayed. The settings are persistent and applied in each subsequently opened editor. The table rows can be sorted by clicking on the column name.

You can display the selected data items on a chart. To open the chart, choose *Plot* from the context menu (double-click also works for single data lines). The opened chart is not added automatically to the analysis file, so you can explore the data by opening the chart this way and closing the chart page without making the editor "dirty."

The selected vector's data can also be displayed in a table. Make sure that the *Output Vector View* is opened. If it is not open, you can open it from the context menu (*Show Output Vector View*). If you select a vector in the editor, the view will display its content.

You can create a dataset from the selected result items. Select *Add Filter Expression to Dataset...* if you want to add all items displayed in the table. Select *Add Filter Selected Data to Dataset...* if you want to add the selected items only. You can add the items to an existing dataset, or you can create a new dataset in the opening dialog.

Filter expressions

A filter expression is composed of atomic patterns with the AND, OR, NOT operators. An atomic pattern filters for the value of a field of the result item and has the form <field_name>(<pattern>). The following table shows the valid field names. You can omit the name field and simply use the name pattern as a filter expression. It must be quoted if it contains whitespace or parentheses.

Field	**Description**
name	the name of the scalar, vector or histogram
module	the name of the module
file	the file of the result item
run	the run identifier
attr: *name*	the value of the run attribute named *name* , e.g. attr:experiment
param: *name*	the value of the module parameter named *name*

In the pattern specifying the field value, you can use the following shortcuts:

Pattern	Description
?	matches any character except '.'
*	matches zero or more characters except '.'
**	matches zero or more characters (any character)
{a-z}	matches a character in range a-z
{^a-z}	matches a character not in range a-z
{32..255}	any number (i.e. sequence of digits) in range 32..255 (e.g. "99")
[32..255]	any number in square brackets in range 32..255 (e.g. "[99]")
\	takes away the special meaning of the subsequent character

Content Assist is available in the text fields where you can enter a filter expression. Press **Ctrl+Space** to get a list of appropriate suggestions related to the expression at the cursor position.

Examples

```
"queuing time"
```

filters for result items named *queuing time*.

```
module(**.sink) AND (name("queuing time") OR
                     name("transmission time"))
```

results in the *queuing times* and *transmission times* that are written by modules named *sink*.

9.3.2. Datasets

Overview

The third page displays the datasets and charts created during the analysis. Datasets describe a set of input data, the processing applied to them and the charts. The dataset is displayed as a tree of processing steps and charts. There are nodes for adding and discarding data, applying processing to vectors, selecting the operands of the operations and content of charts, and for creating charts.

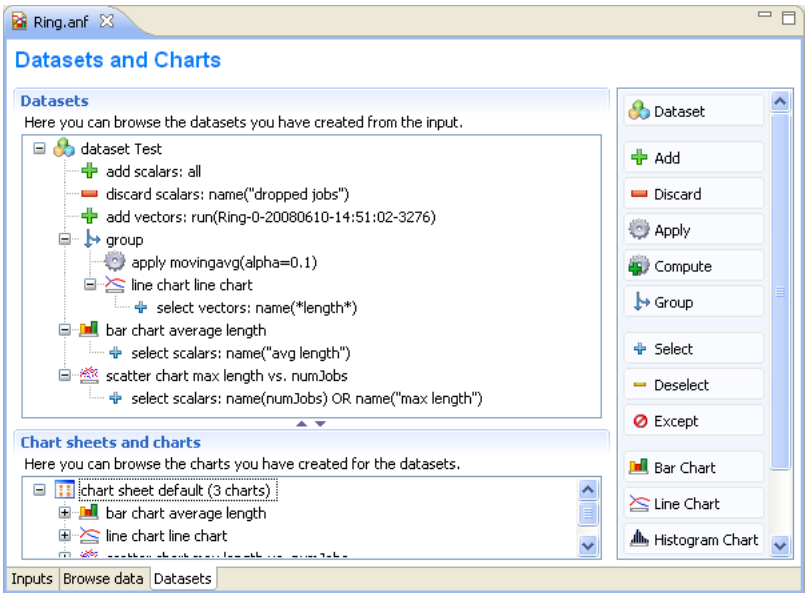


Figure 9.4. Defining datasets to be analyzed

You can browse the dataset's content in the *Dataset View*. Open the view by selecting *Show Dataset View* from the context menu. Select a chart to display its content or another node to display the content of the dataset after processing is applied.

Editing datasets

The usual tree editing operations work on the Dataset tree. New elements can be added by dragging elements from the palette on the right to an appropriate place on the tree. Alternatively, you can select the parent node and press the button on the toolbar. An existing element can be edited by selecting the element and editing its properties on the property sheet, or by opening an item-specific edit dialog by choosing *Properties...* from the context menu.

Datasets can be opened on a separate page by double-clicking on them. It is easier to edit the tree on this page. Double-clicking a chart node will display the chart.

Computations can be applied to the data by adding Apply/Compute nodes to the dataset. The data items to which the computation should be applied can be selected by adding Select/Deselect children to the processing node. By default, the computation is applied to each data item. The difference between Apply and Compute is that Apply will remove its input from the dataset, while Compute keeps the original data, too. Table 9.1, “Processing operations”contains the list of available operations.

Name	Description
add	Adds a constant to the input: $y^{out}_k = y_k + c$
compare	Compares value against a threshold, and optionally replaces it with a constant
crop	Discards values outside the [t1, t2] interval
difference	Substracts the previous value from every value: $y^{out}_k = y_k - y_{k-1}$
diffquot	Calculates the difference quotient of every value and the subsequent one: $y^{out}_k = (y_{k+1}-y_k) / (t_{k+1}-t_k)$
divide-by	Divides input by a constant: $y^{out}_k = y_k / a$
divtime	Divides input by the current time: $y^{out}_k = y_k / t_k$
expression	Evaluates an arbitrary expression. Use t for time, y for value, and tprev, yprev for the previous values.
integrate	Integrates the input as a step function (sample-hold or backward-sample-hold) or with linear interpolation
lineartrend	Adds linear component to input series: $y^{out}_k = y_k + a * t_k$
mean	Calculates mean on (0,t)
modulo	Computes input modulo a constant: $y^{out}_k = y_k \% a$
movingavg	Applies the exponentially weighted moving average filter: $y^{out}_k = y^{out}_{k-1} + alpha*(y_k-y^{out}_{k-1})$
multiply-by	Multiplies input by a constant: $y^{out}_k = a * y_k$
nop	Does nothing
removerepeats	Removes repeated y values
slidingwinavg	Replaces every value with the mean of values in the window: $y^{out}_k = SUM(y_i, i=k-winsize+1..k)/winsize$
sum	Sums up values: $y^{out}_k = SUM(y_i,\ i=0..k)$
timeavg	Calculates the time average of the input (integral divided by time)
timediff	Returns the difference in time between this and the previous value: $y^{out}_k = t_k - t_{k-1}$
timeshift	Shifts the input series in time by a constant: $t^{out}_k = t_k + dt$
timetoserial	Replaces time values with their index: $t^{out}_k = k$
winavg	Calculates batched average: replaces every `winsize' input values with their mean. Time is the time of the first value in the batch.

Table 9.1. Processing operations

Processing steps within a Group node only affect the group. This way, you can create branches in the dataset. To group a range of sibling nodes, select them and choose *Group* from the context menu. To remove the grouping, select the Group node and choose *Ungroup*.

Charts can be inserted to display data. The data items to be displayed can be selected by adding Select/Deselect children to the chart node. By default, the chart displays all data in the dataset at its position. You can modify the content of the chart by adding Select and Deselect children to it. Charts can be fully customized including setting titles, colors, adding legends, grid lines, etc. See the Charts section for details.

Export

You can export the content of the dataset into text files. Three formats are supported: comma separated values (CSV), Octave text files and Matlab script. Right-click on the processing node or chart, and select the format from the *Export to File* submenu. The file will contain the data of the chart, or the dataset after the selected processing is applied. Enter the name of the file and press *Finish*.

Vectors are always written as two columns into the CSV files, but the shape of the scalars table can be changed by selecting the grouping attributes in the dialog. For example, assume we have two scalars (named s1 and s2) written by two modules (m1 and m2) in two runs (r1 and r2), resulting in a total of 8 scalar values. If none of the checkboxes is selected in the *Scalars grouped by* group, then the data is written as:

r1 m1 s1	r1 m1 s2	r1 m2 s1	r1 m2 s2	r2 m1 s1	r2 m1 s2	r2 m2 s1	r2 m2 s2
1	2	3	4	5	6	7	8

Grouping the scalars by module name and scalar name would have the following result:

Module	Name	r1	r2
m1	s1	1	5
m1	s2	2	6
m2	s1	3	7
m2	s2	4	8

The settings specific to the file format are:

CSV. You can select the separator, line ends and quoting character. The default setting corresponds to RFC4180. The precision of the numeric values can also be set. The CSV file contains an optional header followed by the vector's data or groups of scalars. If multiple vectors are exported, each vector is written into a separate file.

Octave. The data is exported as an Octave text file. This format can be loaded into the R [http:://www.r-project.org] statistical data analysis tool, as well. The tables are saved as structures containing an array for each column.

Matlab. The data is exported as a Matlab script file. It can be loaded into Matlab/ Octave with the source() function.

NOTE

Histograms can not be exported yet.

Chart sheets

Sometimes, it is useful to display several charts on one page. When you create a chart, it is automatically added to a default chart sheet. Chart sheets and the their charts are displayed on the lower pane of the *Datasets* page. To create a new chart sheet, use the *Chart Sheet* button on the palette. You can add charts to it either by using the opening dialog or by dragging charts. To move charts between chart sheets, use drag and drop or Cut/Paste. You can display the charts by double-clicking on the chart sheet node.

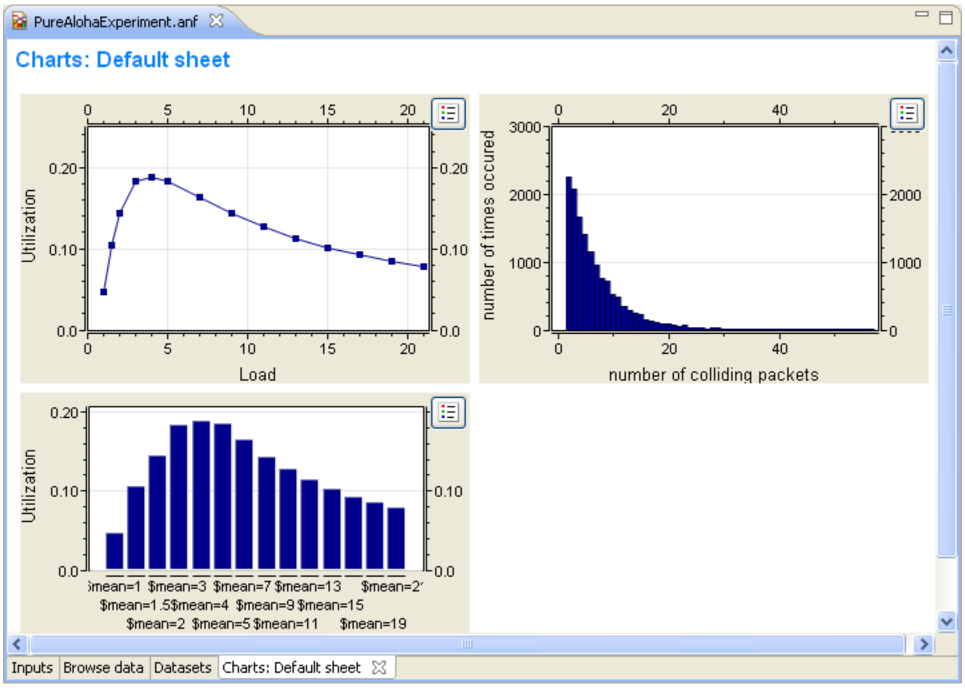


Figure 9.5. Chart Sheet page with three charts

9.3.3. Charts

Overview

You typically want to display the recorded data in charts. In OMNeT++ 4.x, you can open charts for scalar, vector or histogram data with one click. Charts can be saved into the analysis file, too. The Analysis Editor supports bar charts, line charts, histogram charts and scatter charts. Charts are interactive; users can zoom, scroll, and access tooltips that give information about the data items.

Charts can be customized. Some of the customizable options include titles, fonts, legends, grid lines, colors, line styles, and symbols.

Creating charts

To create a chart, use the palette on the *Dataset* page. Drag the chart button and drop it to the dataset at the position you want it to appear. If you press the chart button then it opens a dialog where you can edit the properties of the new chart. In this case the new chart will be added at the end of the selected dataset or after the selected dataset item.

Temporary charts can be created on the *Browse Data* page for quick view. Select the scalars, vectors or histograms and choose *Plot* from the context menu. If you want to save such a temporary chart in the analysis, then choose *Convert to dataset...* from the context menu of the chart or from the toolbar.

Editing charts

You can open a dialog for editing charts from the context menu. The dialog is divided into several pages. The pages can be opened directly from the context menu. When you select a line and choose *Lines...* from the menu, you can edit the properties of the selected line.

You can also use the *Properties View* to edit the chart. It is recommended that users display the properties grouped according to their category (on the toolbar of the *Properties View*).

Main	
antialias	Enables antialising.
caching	Enables caching. Caching makes scrolling faster, but sometimes the plot might not be correct.
background color	Background color of the chart.
Titles	
graph title	Main title of the chart.
graph title font	Font used to draw the title.
x axis title	Title of the horizontal axis.
y axis title	Title of the vertical axis.
axis title font	Font used to draw the axes titles.
labels font	Font used to draw the tick labels.
x labels rotated by	Rotates the tick labels of the horizontal axis by the given angle (in degrees).
Axes	
y axis min	Crops the input below this y value.
y axis max	Crops the input above this y value.
y axis logarithmic	Applies a logarithmic transformation to the y values.
grid	Add grid lines to the plot.
Legend	
display	Displays the legend.
border	Add border around the legend.
font	Font used to draw the legend items.
position	Position of the legend.
anchor point	Anchor point of the legend.

Table 9.2. Common chart properties

Zooming and panning

Charts have two mouse modes. In Pan mode, you can scroll with the mouse wheel and drag the chart. In Zoom mode, the user can zoom in on the chart by left-clicking and zoom out by doing a **Shift**+left click, or using the mouse wheel. Dragging selects a rectangular area for zooming. The toolbar icons and switch between Pan and Zoom modes. You can also find toolbar buttons to zoom in , zoom out () and zoom to fit (). Zooming and moving actions are remembered in the navigation history.

Tooltip

When the user hovers the mouse over a data point, the appearing tooltip shows line labels and the values of the points close to the cursor. The names of all lines can be displayed by hovering over the button at the top right corner of the chart.

Copy to clipboard

You can copy the chart to the clipboard by selecting *Copy to Clipboard* from the context menu. The chart is copied as a bitmap image and is the same size as the chart on the screen.

Bar charts

Bar charts display scalars as groups of vertical bars. The bars can be positioned within a group next to, above or in front of each other. The baseline of the bars can be changed. Optionally, a logarithmic transformation can be applied to the values.

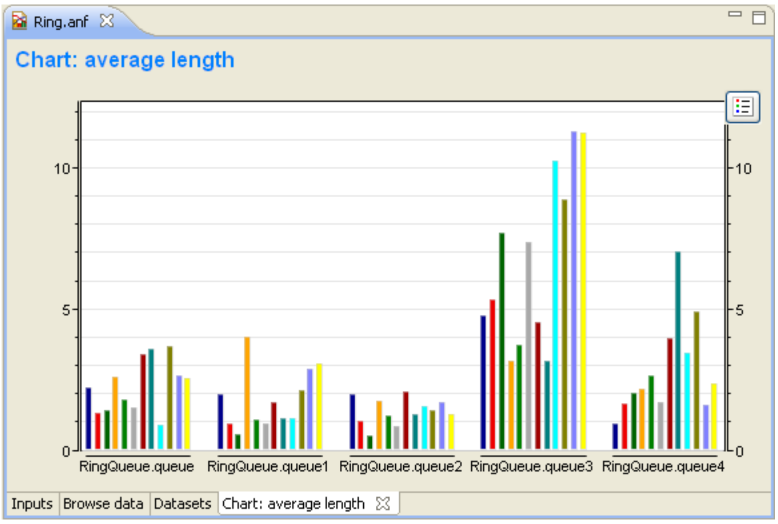


Figure 9.6. Bar chart

The scalar chart's content can be specified on the *Content* tab of their *Properties* dialog. Attributes in the "Groups" list box determine the groups so that within a group each attribute has the same value. Attributes in the "Bars" list box determine the bars; the bar height is the average of scalars that have the same values as the "Bar" attributes. You can classify the attributes by dragging them from the upper list boxes to the lower list boxes. You will normally want to group the scalars by modules and label the bars with the scalar name. This is the default setting, if you leave each attribute in the upper list box.

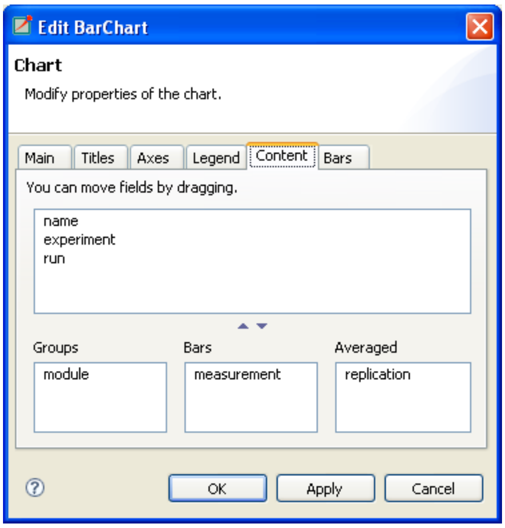


Figure 9.7. Dialog page for bar chart content

In addition to the common chart properties, the properties of bar charts include:

Titles	
wrap labels	If true labels are wrapped, otherwise aligned vertically.
Plot	
bar baseline	Baseline of the bars.
bar placement	Arrangement of the bars within a group.
Bars	
color	Color of the bar. Color name or #RRGGBB. Press Ctrl+Space for a list of color names.

Table 9.3. Bar chart properties

Line charts

Line charts can be used to display output vectors. Each vector in the dataset gives a line on the chart. You can specify the symbols drawn at the data points (cross, diamond, dot, plus, square triangle or none), how the points are connected (linear, step-wise, pins or none) and the color of the lines. Individual lines can be hidden.

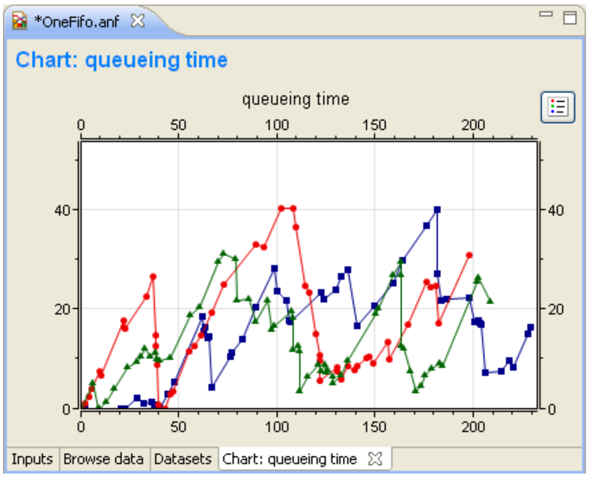


Figure 9.8. Line chart

Line names identify lines on the legend, property sheets and edit dialogs. They are formed automatically from attributes of the vector (like file, run, module, vector name, etc.). If you want to name the lines yourself, you can enter a name pattern in the *Line names* field of the *Properties* dialog (*Main* tab). You can use "{file}", "{run}", "{module}", or "{name}" to refer to an attribute value. Press **Ctrl**+**Space** for the complete list.

Processing operations can be applied to the dataset of the chart by selecting *Apply* or *Compute* from the context menu. If you want to remove an existing operation, you can do it from the context menu, too.

Line charts are synchronized with *Output Vector* and *Dataset* views. Select a data point and you will see that the data point and the vector are selected in the Output Vector and *Dataset View*, as well.

Axes	
x axis min	Crops the input below this x value.
x axis max	Crops the input above this x value.
Lines	
display line	Displays the line.
symbol type	The symbol drawn at the data points.
symbol size	The size of the symbol drawn at the data points.
line type	Line drawing method. One of Linear, Pins, Dots, Points, Sample-Hold or Backward Sample-Hold.
line color	Color of the line. Color name or #RRGGBB. Press Ctrl+Space for a list of color names.

Table 9.4. Line chart properties

Histogram charts

Histogram charts can display data of histograms. They support three view modes:

Count
: The chart shows the recorded counts of data points in each cell.

Probability density
: The chart shows the probability density function computed from the histogram data.

Cumulative density
: The chart shows the cumulative density function computed from the histogram data.

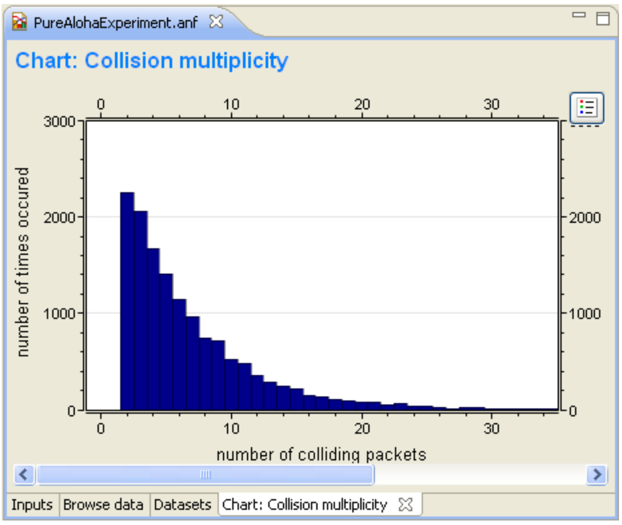


Figure 9.9. Histogram chart

> When drawing several histograms on one chart, set the "Bar type" property to Outline. This way the histograms will not cover each other.

Plot	
bar type	Histogram drawing method.
bar baseline	Baseline of the bars.
histogram data type	Histogram data. Counts, probability density and cumulative density can be displayed.
Histograms	
hist color	Color of the bar. Color name or #RRGGBB. Press Ctrl+Space for a list of color names.

Table 9.5. Histogram chart properties

Scatter charts

Scatter charts can be created from both scalar and vector data. You have to select one statistic for the x coordinates; other data items give the y coordinates. How the x and y values are paired differs for scalars and vectors.

Scalars. For each value of the x scalar, the y values are selected from scalars in the same run.

Vectors. For each value of the x vector, the y values are selected from the same run and with the same simulation time.

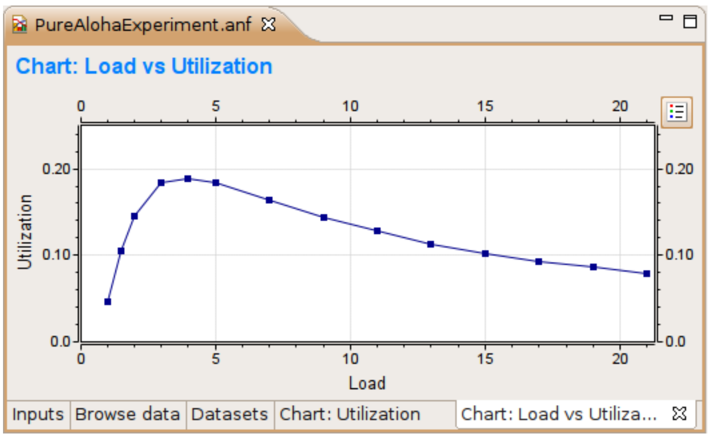


Figure 9.10. A scatter chart

By default, each data point that comes from the same y scalar belongs to the same line. This is not always what you want because these values may have been generated in runs having different parameter settings; therefore, they are not homogenous. You can specify scalars to determine the "iso" lines of the scatter chart. Only those points that have the same values of these "iso" attributes are connected by lines.

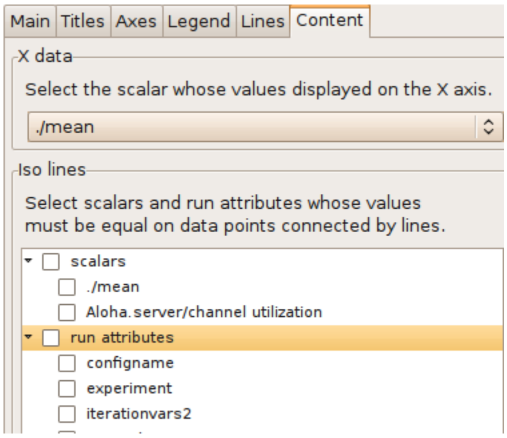


Figure 9.11. A scatter chart

If you want to use a module parameter as an iso attribute, you can record it as a scalar by setting "<module>.<parameter_name>.param-record-as-scalar=true" in the INI file.

Axes	
x axis min	Crops the input below this x value.
x axis max	Crops the input above this x value.
Lines	
display line	Displays the line.
symbol type	The symbol drawn at the data points.
symbol size	The size of the symbol drawn at the data points.
line type	Line drawing method. One of Linear, Pins, Dots, Points, Sample-Hold or Backward Sample-Hold.
line color	Color of the line. Color name or #RRGGBB. Press Ctrl+Space for a list of color names.

Table 9.6. Scatter chart properties

9.4. Associated Views

9.4.1. Outline View

The *Outline View* shows an overview of the current analysis. Clicking on an element will select the corresponding element in the current editor. Tree editing operations also work in this view.

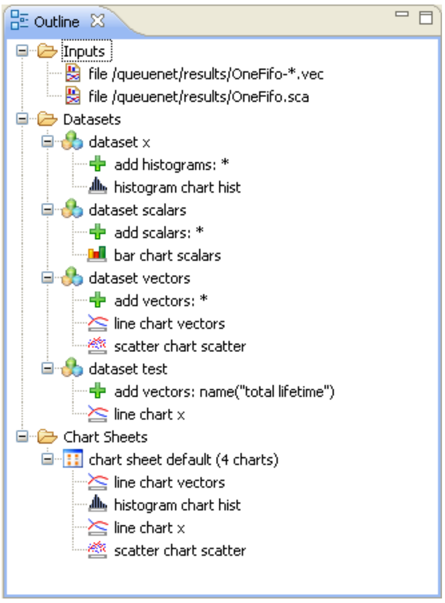


Figure 9.12. Outline View of the analysis

9.4.2. Properties View

The Properties View displays the properties of the selected dataset, processing node and chart. Font and color properties can be edited as text or by opening dialogs. Text fields that have a bulb on the left side have a content assist; press **Ctrl**+**Space** to activate it.

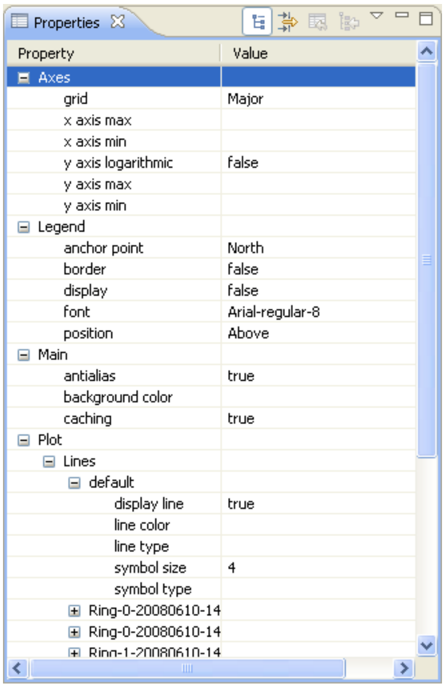


Figure 9.13. Properties View showing chart properties

9.4.3. Output Vector View

The *Output Vector View* shows the content of the selected vector. It displays the serial number, simulation time and value of the data points. When event numbers are recorded during the simulation, they are also displayed. Large output files are handled efficiently; only the visible part of the vector is read from the disk. Vectors that are the result of a computation are saved in temporary files.

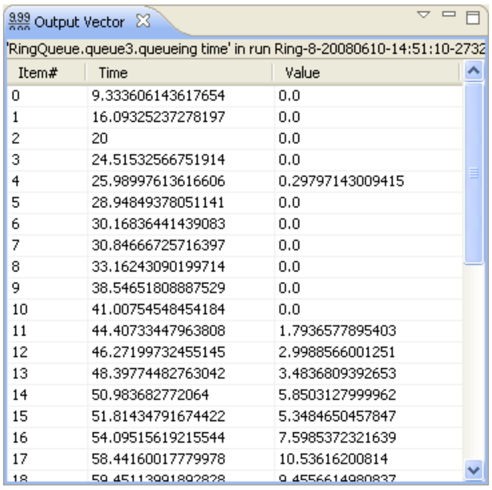


Figure 9.14. Output Vector View

To navigate to a specific line, use the scroll bar or the menu of the view:

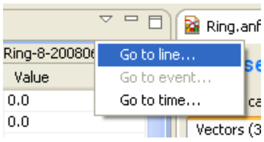


Figure 9.15.

9.4.4. Dataset View

The *Dataset View* displays the dataset's content of the selected dataset item. It is divided into three tables, similar to the ones on the *Browse Data* page. The tables can be selected by the  icons. There is also a tool button () to show or hide the filter.

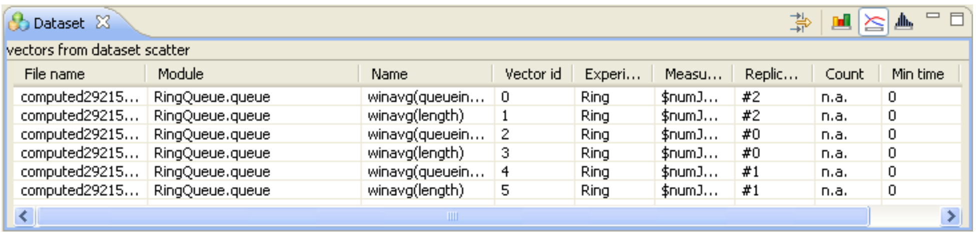


Figure 9.16. Dataset View

Chapter 10. NED Documentation Generator

10.1. Overview

This chapter describes how to use the NED Documentation Generator from the IDE.

Please refer to the OMNeT++ Manual for a complete description of the documentation generation features and the available syntax in NED and MSG file comments.

The generator has several project-specific settings that can be set from the project context menu through the Properties menu item. The output folders for both NED documentation and C++ documentation can be set separately. The doxygen-specific configuration is read from the text file doxy.cfg by default. The IDE provides a sensible default configuration for doxygen in case you do not want to go through all the available options. The generated HTML uses CSS to make its style customizable. You can provide your own style sheet if the default does not meet your needs. In general, all project-specific settings have good defaults that work well with the documentation generator.

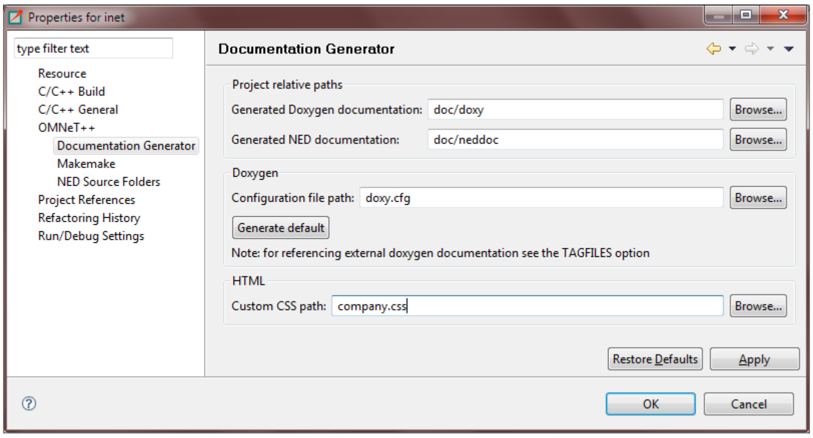


Figure 10.1. Configuring project-specific settings

To generate NED documentation, you need to select one or more projects. Then, either go to the main Project menu or to the project context menu and select the Generate NED Documentation menu item. This will bring up the configuration dialog where you can set various settings for the current generation before starting it.

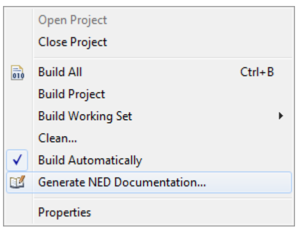


Figure 10.2. Opening the NED documentation generator

The IDE can generate documentation for multiple projects at the same time. Other options control the content of the documentation including what kind of diagrams will be generated and whether NED sources should be included. You can enable doxygen to generate C++ documentation that will be cross-linked from the NED documentation. The tool can generate the output into each project as configured in the project-specific settings, or into a separate directory. The latter is useful for exporting standalone documentation for several complex projects at once.

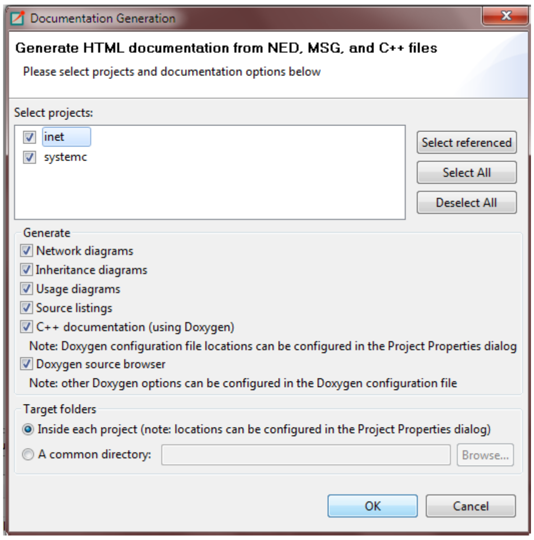


Figure 10.3. Configuring the NED documentation generator

The NED generation process might take a while for big projects, so please be patient. For example, the INET project's complete documentation including the C++ doxygen documentation takes a few minutes to build. You can follow the process in the IDE's progress monitor.

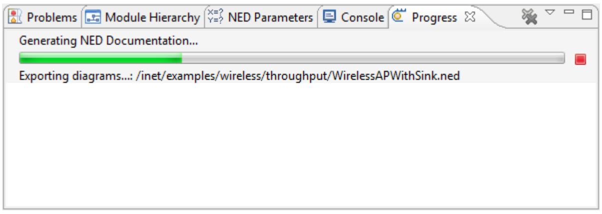


Figure 10.4. Generating NED documentation in progress

The result is a number of cross-linked HTML pages that can be opened by double-clicking the generated `index.html`. On the left side, you will see a navigation tree, while on the right side there will be an overview of the project. If you have not yet added a @titlepage directive into your NED comments, then the overview page will display a default content.

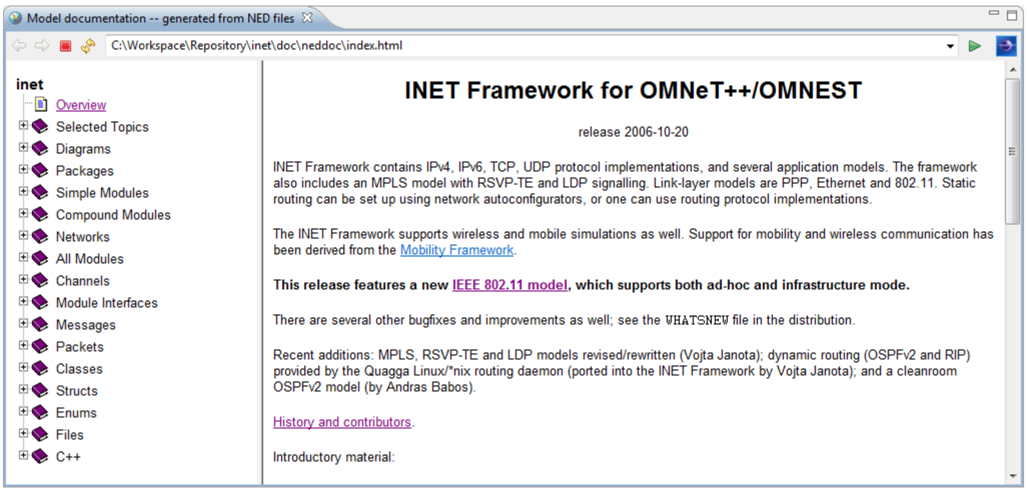


Figure 10.5. The resulting NED documentation

The documentation contains various inheritance and usage diagrams that make it easier to understand complex models. The diagrams are also cross-linked, so that when you click on a box, the corresponding model element's documentation will be opened. The NED model elements are also exported graphically from the NED Editor. These static images provide cross-referencing navigation for submodules.

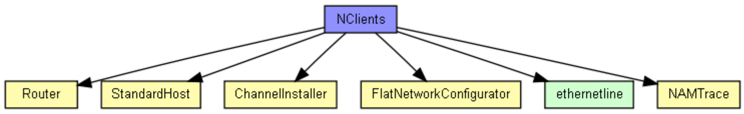
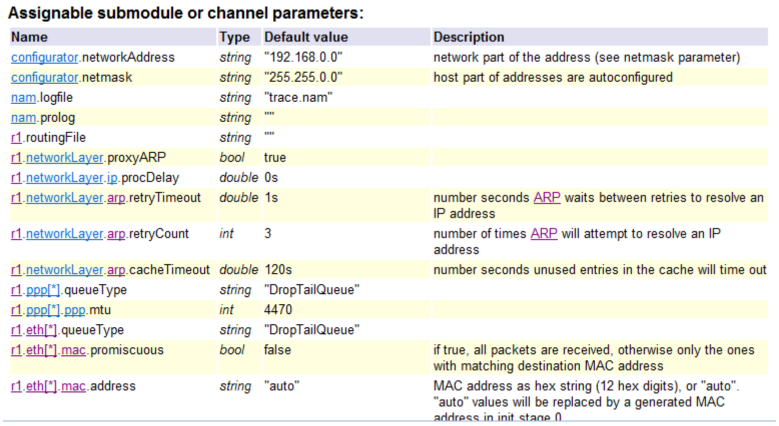


Figure 10.6. NED usage diagram

There are also a number of tables that summarize various aspects of modules, networks, messages, packets, etc. The most interesting is the list of assignable parameters. It shows all parameters from all submodules down the hierarchy that do not have fixed values. These can be set either through inheritance, encapsulation or from the INI file as experiments.



Assignable submodule or channel parameters:

Name	Type	Default value	Description
configurator.networkAddress	*string*	"192.168.0.0"	network part of the address (see netmask parameter)
configurator.netmask	*string*	"255.255.0.0"	host part of addresses are autoconfigured
nam.logfile	*string*	"trace.nam"	
nam.prolog	*string*	""	
r1.routingFile	*string*	""	
r1.networkLayer.proxyARP	*bool*	true	
r1.networkLayer.ip.procDelay	*double*	0s	
r1.networkLayer.arp.retryTimeout	*double*	1s	number seconds ARP waits between retries to resolve an IP address
r1.networkLayer.arp.retryCount	*int*	3	number of times ARP will attempt to resolve an IP address
r1.networkLayer.arp.cacheTimeout	*double*	120s	number seconds unused entries in the cache will time out
r1.ppp[*].queueType	*string*	"DropTailQueue"	
r1.ppp[*].ppp.mtu	*int*	4470	
r1.eth[*].queueType	*string*	"DropTailQueue"	
r1.eth[*].mac.promiscuous	*bool*	false	if true, all packets are received, otherwise only the ones with matching destination MAC address
r1.eth[*].mac.address	*string*	"auto"	MAC address as hex string (12 hex digits), or "auto". "auto" values will be replaced by a generated MAC address in init stage 0

Figure 10.7. NED assignable parameters

There are other tables that list parameters, properties, gates, using modules or networks, and various other data along with the corresponding descriptions. In general, all text might contain cross-links to other modules, messages, classes, etc. to make navigation easier.

Chapter 11. Extending the IDE

There are several ways to extend the functionality of the OMNeT++ IDE. The Simulation IDE is based on the Eclipse platform, but extends it with new editors, views, wizards, and other functionality.

11.1. Installing New Features

Because the IDE is based on the Eclipse platform, it is possible to add additional features that are available for Eclipse. The installation procedure is exactly the same as with a standard Eclipse distribution. Choose the *Help | Install New Software...* menu item and select an existing Update Site to work with or add a new Site (using the site URL) to the Available Software Sites. After the selection, you can browse and install the packages the site offers.

To read about installing new software into your IDE, please visit the *Updating and installing software* topic in the *Workbench User Guide*. You can find the online help system in the *Help | Help Contents* menu.

> There are thousands of useful components and extensions for Eclipse. The best places to start looking for extensions are the Eclipse Marketplace (http://marketplace.eclipse.org/) and the Eclipse Plugins info site (http://www.eclipse-plugins.info).

11.2. Adding New Wizards

The Simulation IDE makes it possible to contribute new wizards into the wizard dialogs under the *File | New* menu without writing Java code or requiring any knowledge of Eclipse internals. Wizards can create new simulation projects, new simulations, new NED files or other files by using templates, or perform export/import functions. Wizard code is placed under the *templates* folder of an OMNeT++ project, which makes it easy to distribute wizards with the model. When the user imports and opens a project which contains wizards, the wizards will automatically become available.

> The way to create wizards is documented in the *IDE Customization Guide*.

11.3. Project-Specific Extensions

It is possible to install an Eclipse plug-in by creating a *plugins* folder in an OMNeT++ project and copying the plug-in JAR file to that folder (this mechanism is implemented as part of the Simulation IDE and does not work in generic Eclipse installations or with non-OMNeT++ projects). This extension mechanism allows the distribution of model-specific IDE extensions together with a simulation project without requiring the end user to do extra deployment steps to install the plug-in. Plugins and wizards that are distributed with a project are automatically activated when the host project is opened.

Eclipse plug-in JAR files can be created using the *Plug-in Development Environment*. The OMNeT++ IDE does not contain the *PDE* by default; however, it can be easily installed, if necessary.

> Read the *OMNeT++ IDE Development Guide* for more information about how to install the PDE and how to develop plug-in extensions for the IDE.