# Manual & User Guide

Version 2.05

# Open Visual Environment

Version 1.3

# OpenComRTOS Host Servers

Version 0.2

# Table of contents

# 1. Chapter 1 - Introduction

## 1.1. Introduction

The Open Visual Environment (OpenVE) was designed to help software engineers to implement applications for Open Communication Real Time Operating System (OpenComRTOS). OpenComRTOS is a network-centric real time operating system for embedded systems.

OpenComRTOS supports systems from single processors with little available memory (5—10 kB) to systems consisting of multiple processors of different types with large amounts of memory, so called heterogeneous systems. Due to OpenComROTS's Virtual Single Processor programming model the application logic does not need to be changed when the underlying system changes.

This manual describes how to use the Win32 based version of OpenVE. By default the Win32 OpenVE installation contains the Win32 version of OpenComRTOS, which allows to create OpenComRTOS application executable on MS-Windows systems. OpenComRTOS for other processor types can easily be integrated into this system.

This Manual & User Guide describes the Open Visual Environment of version **1.3.3.5**

## 1.2. System Requirements

**Software:**

- Microsoft Windows ® XP or later.
- MinGW tool-chain, version 5.1.6 or later.
- CMake cross-platform system for build automation, version 2.6.0 or later.

**Hardware:**

- Pentium 1 GHz or higher.
- 128 MB RAM or higher.
- 100 MB free hard disk space.

## 1.3. Open License Agreement

OpenVE is licensed under a so-called "Open License" besides a standard "binary" license. A binary license is a standard "right to use" license of the software on a given host station where the applications are developed for a given target system or processor type.

A full Open License also includes all source code, design documents, formal and informal models, test suites, etc. Such an Open License can be used to resell or provide binary licenses of the OpenVE. No runtime royalties are due for the use of the software in applications.

More information see at: www.OpenLicenseSociety.org and www.Altreonic.com

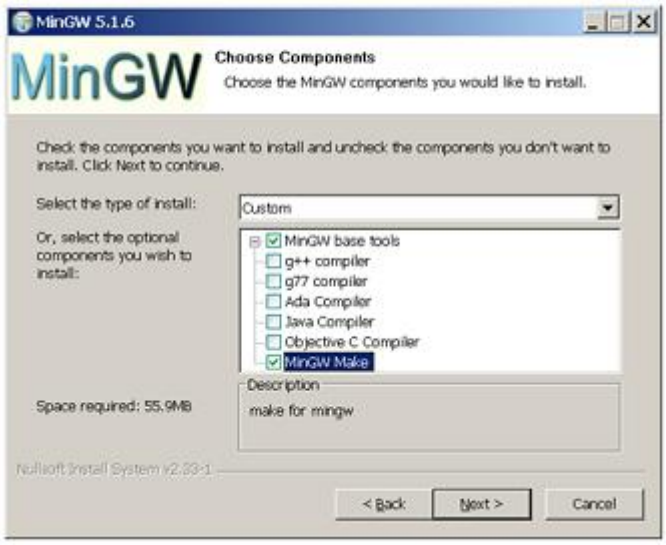
## 1.4. Tools used for the OpenVE development

| **IDE** |
|---|
| Microsoft Visual Studio 2008 |
| **Programming language** |
| Microsoft Visual C++ |
| **Libraries** |
| Qt 4.5.2 (http://qt.nokia.com/) |

## 1.5. Installation procedure

This procedure describes the installation of the Windows version of OpenVE.

1. Install MinGW (http://mingw.org) into its default folder "C:\MinGW".
2. In the component selection screen (Figure 1-1) select to install "MinGW Make". This component is an essential part of the OpenComRTOS build system.



MinGW 5.1.6

MinGW

**Choose Components**

Choose the MinGW components you would like to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Custom

Or, select the optional components you wish to install:

- [✓] MinGW base tools
- [ ] g++ compiler
- [ ] g77 compiler
- [ ] Ada Compiler
- [ ] Java Compiler
- [ ] Objective C Compiler
- [✓] MinGW Make

Space required: 55.9MB

Description

make for mingw

Nullsoft Install System v2.33-1

**Figure 1-1. The MinGW installer component selection**

3. Add the path of MinGW binaries to the System Search Path of MS-Windows:

- Open the System Properties (by right click on “My Computer" and select “Properties") – see Figure 1-2.

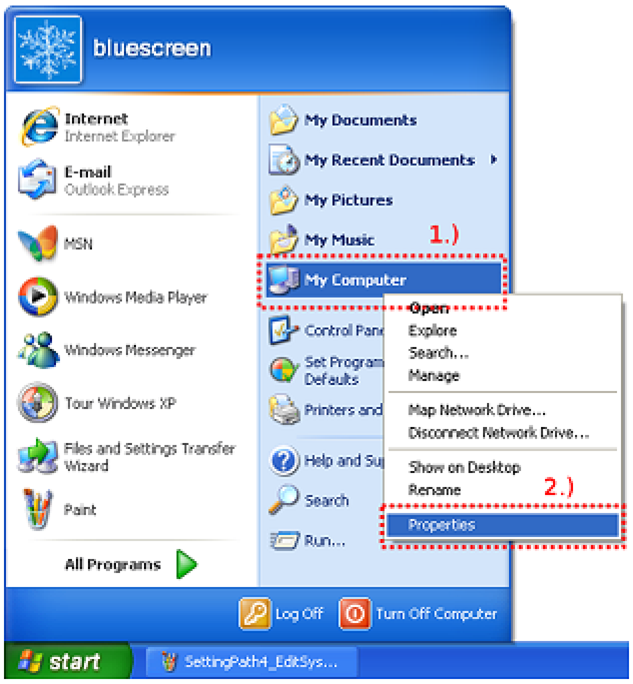
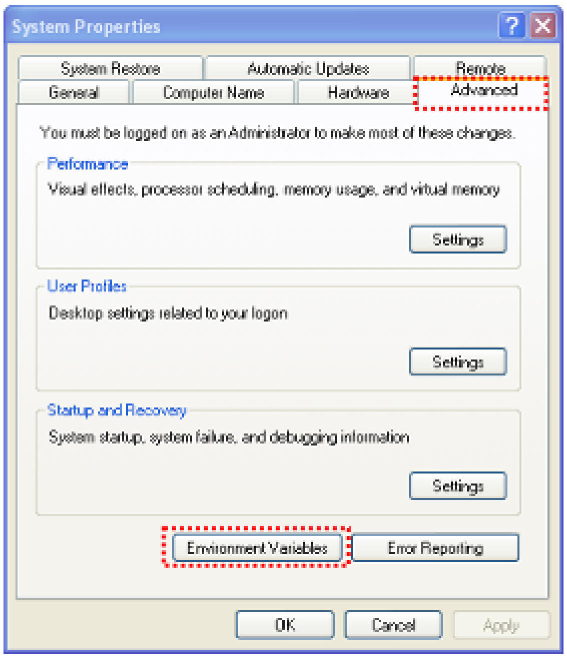


**Figure 1-2. Opening the System Properties Dialogue**

- In opened window select the tab “Advanced", in which click on the “Environment Variables" button, see Figure 1-3.



You must be logged on as an Administrator to make most of these changes.

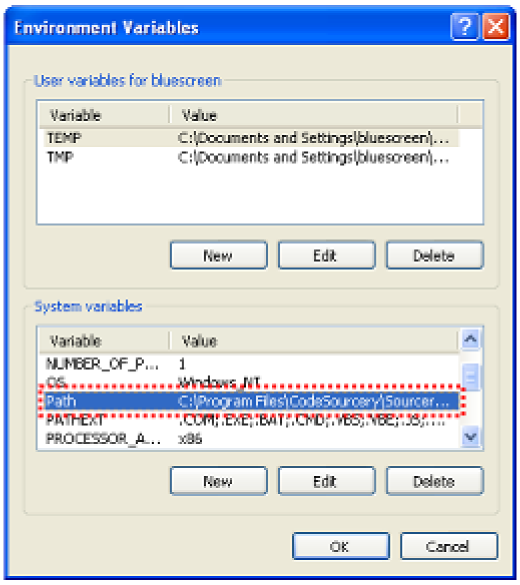
Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your logon

System startup, system failure, and debugging information

**Figure 1-3. Opening the Environment Variables Dialogue**

- In the list box “System Variables" select the variable “Path" and click on the button labelled “Edit" (you can also double click on the list entry), see Figure 1-4.


Environment Variables

User variables for bluescreen

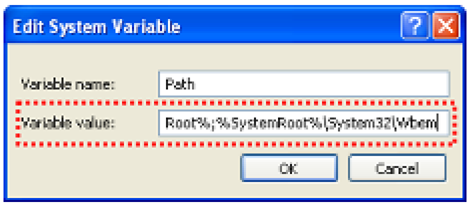
| Variable | Value |
|---|---|
| TEMP | C:\Documents and Settings\bluescreen\... |
| TMP | C:\Documents and Settings\bluescreen\... |

System variables

| Variable | Value |
|---|---|
| NUMBER_OF_P... | 1 |
| OS | Windows_NT |
| Path | C:\Program Files\CodeSourcery\Sourcer... |
| PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;... |
| PROCESSOR_A... | x86 |

Figure 1-4. Opening the Dialogue to modify the variable Path

- In the dialogue "Edit System Variable" (see Figure 1-5) add the following to the end of the "Variable value" Edit Field: "**;c:\MinGW\bin**".


Edit System Variable

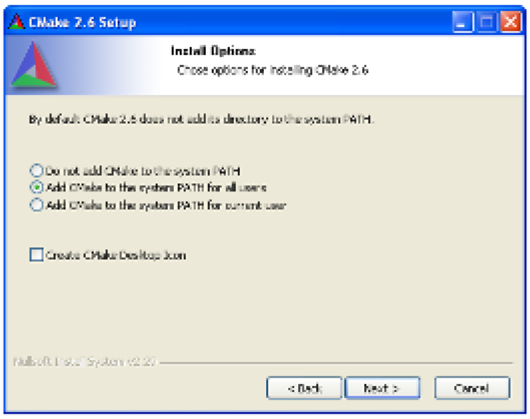
Variable name: Path

Variable value: Root%;%SystemRoot%\System32\Wbem

Figure 1-5. Modifying the value of the path variable

4.Install CMake system (http://www.cmake.org).

5. In the screen "Install Options" select "Add CMake to the system PATH for all users" (see Figure 1-6). This adds the CMake binary directory to the System Search Path, which is necessary in order for the OpenComRTOS build system to be able to use CMake.


CMake 2.6 Setup

**Install Options**
Chose options for installing CMake 2.6

By default CMake 2.6 does not add its directory to the system PATH.

- ○ Do not add CMake to the system PATH
- ● Add CMake to the system PATH for all users
- ○ Add CMake to the system PATH for current user

☐ Create CMake Desktop Icon

Nullsoft Install System v2.20

Figure 1-6. Adding CMake to the System Binary Search Path

3. Run the OpenVE-<version number>.msi installer.

4. In the OpenVE installer window you have to accept the license agreement, select features to be installed and press "Install".

| **Notes** |
|---|
| By default the program will be installed into directory “C:\OpenVE-<version_number>”<br><br>The license agreement is included in the \share\doc folder of OpenVE installation. (OpenComRTOS Open License - Binary only WIN32_ Linux OEM-01 Sept 2008.pdf) |

After installation you will find the following folders structure:

| Folders | Contains |
|---|---|
| **Bin** | OpenVE and OpenTracer executable files |
| **Examples** | OpenComRTOS applications samples |
| **Share** | OpenVE entity icons (in the icons and the svg folder), documentation - help and license files |
| **Targets** | Supported by OpenComRTOS targets, in default installation - Win32 OpenComRTOS Metamodel, include files, libraries, and code generating utilities |

# 2. Chapter 2 – the OpenVE User Interface

## 2.1. Introduction

This chapter introduces the general structure and principles for using the User Interface of Open Visual Environment.

## 2.2. Main Window at start

After installation the Open Visual Environment starts with an empty workspace (see Figure 2-1).
You can create here a new project or open an existing OpenVE project.

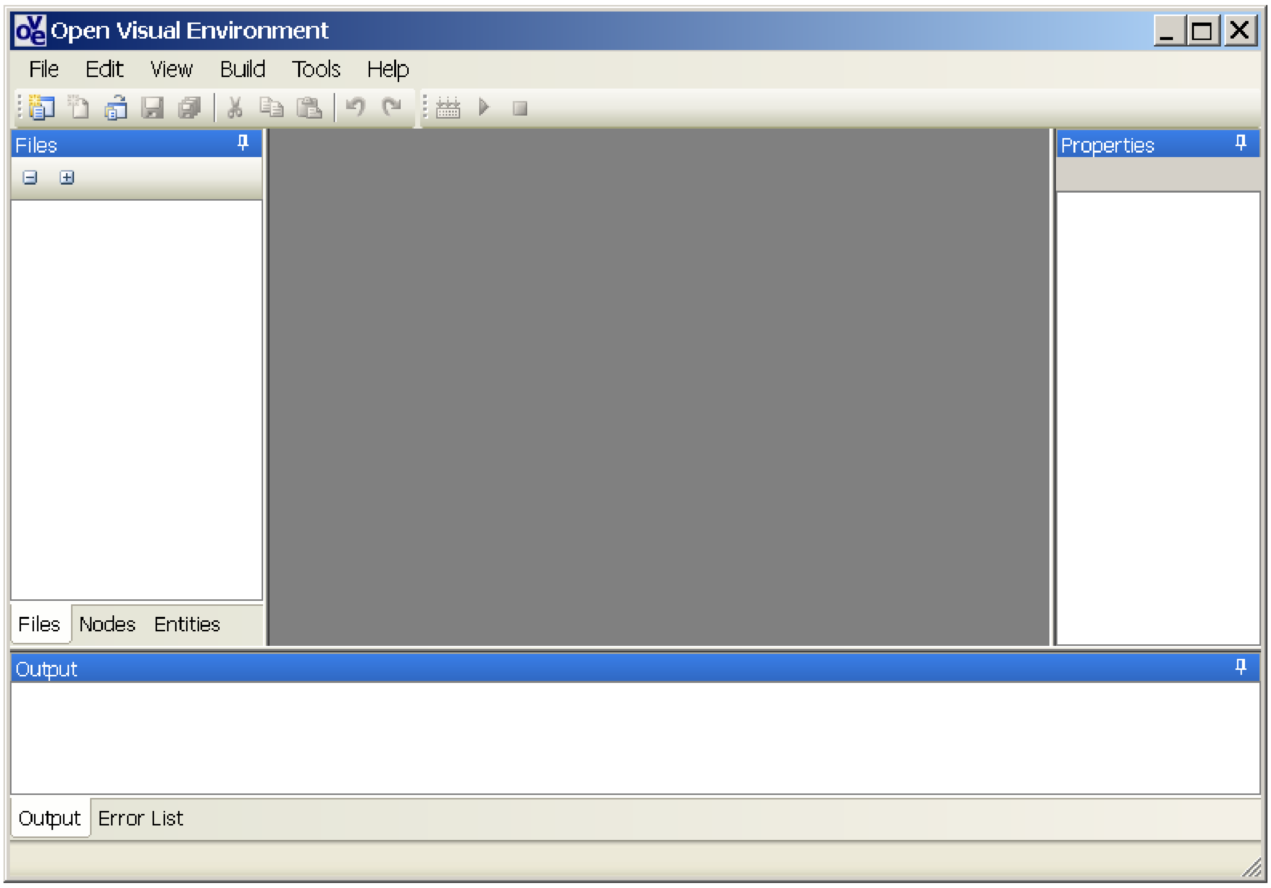
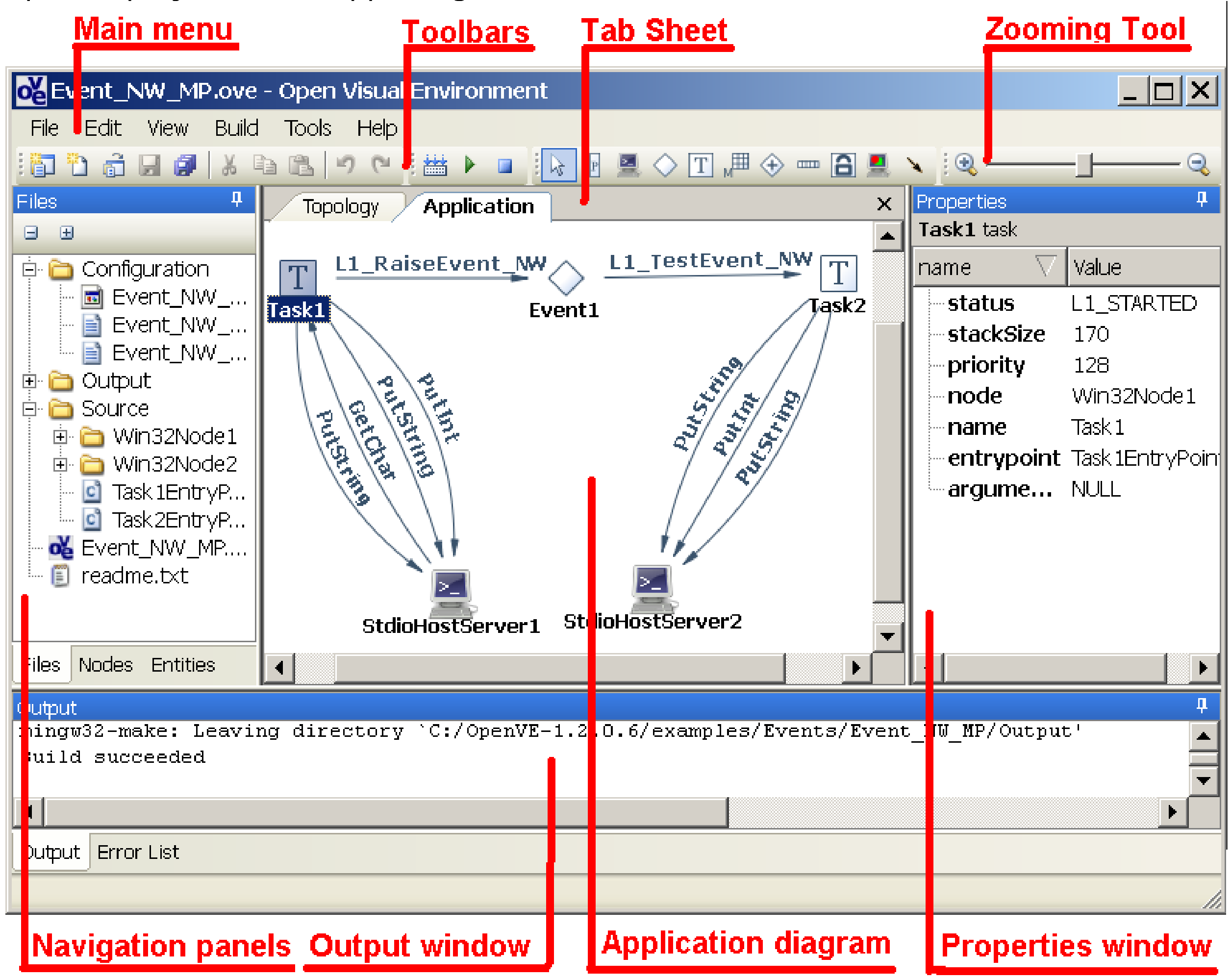


**Figure 2-1. The main window of OpenVE at start**

| **Note** |
|---|
| The visual style of OpenVE depends on the current Windows theme of your PC. |

## 2.3. Workspace structure

The OpenVE workspace (see Figure 2-2) enables creating new and editing existing OpenVE projects and supporting documents.



Task1 task

| name | Value |
|---|---|
| status | L1_STARTED |
| stackSize | 170 |
| priority | 128 |
| node | Win32Node1 |
| name | Task1 |
| entrypoint | Task1EntryPoin |
| argume... | NULL |

```
ingw32-make: Leaving directory `C:/OpenVE-1.2.0.6/examples/Events/Event_NW_MP/Output'
uild succeeded
```

**Figure 2-2. The workspace of OpenVE**

The **OpenVE** workspace consists of the following elements:

- the **Main Menu** gives you access to all available commands;
- the **Toolbars** give you access to common commands;
- the **Zoom-Bar** scales the diagram;
- the **Tab Sheet** allows to switch between multiple different types of documents:
- **Nodes** diagram --- to visually define the topology of the Processor Network.
- **Application** diagram --- to visually define the RTOS based application structure.
- **Source Code Editor** --- to manipulate source code.
- the **Navigation Panels**:
    - **Files tree** window --- displays all OpenVE project folders and files;
    - **Nodes tree** window --- displays nodes and the mapped entities;
    - **Entities tree** window --- displays all entities grouped by type.
- the **Properties window** --- shows and allows to edit the attributes of a selected entity, the attributes can be sorted according to their Name or their Value.

• the **Output window** and **Error list** --- the Output window displays system messages (warning and errors of compilation, information of generation tools). It includes a dedicated Error list which only displays error and warning messages in a tabular format.

## 2.4. Main Menu and Toolbars

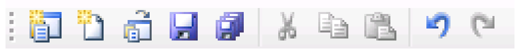
The Main Menu gives you access to all OpenVE commands.
The Toolbars gives you access to common OpenVE commands.

OpenVE has the following toolbars:

- standard;
- build;
- topology;
- application.

| **Notes** |
|---|
| 1. All toolbars are dockable. To dock or undock a toolbar, click on its delimiter and drag it. |
| 2. To show or hide Standard or Build toolbars select the corresponding sub-item in the "View -> Toolbars" main menu item. |
| 3. Topology and Application toolbars are case sensitive and appear when the corresponding diagram becomes active. |

**Standard Toolbar and the corresponding Main Menu Items**



| Command | Meaning |
|---|---|
| New Project | Open the wizard for creating a new project |
| New File | Open a dialog for creating header (*.h) or c source code (*.c) file on the base of predefined in Metamodel templates |
| Open Project | Show a standard Windows dialog to open an existing OpenVE project |
| Save | Save current document. Active if only one of OpenVE project files was changed |
| Save All | Save all opened and changed documents |
| Close | Close current document (the same as click on close button on tab sheet) |
| Close Project | Close all opened documents and the project |
| Recent projects | List of recently opened projects |
| Exit | Exit OpenVE |
| Undo | Undo the last command. Note, some commands like a file deletion cannot be undone |
| Redo | Redo the last command |
| Cut | Cut the selected text |
| Copy | Copy the selected text |
| Paste | Paste the selected text |

| | |
|---|---|
| Delete ✕ | Delete the selected text or entity on a diagram |
| Select All | Select the all text in the opened text editor |
| Options | Open the Options window (see Figure 2-11) |
| Project Settings | Open the Project Settings window (see Figure 2-12) |
| Show Node Diagram | Shows the Node Diagram when hidden |
| Show Application Diagram | Shows the Application Diagram when hidden |
| Toolbars | Shows the Standard or Build diagram when hidden |
| Find | Find a substring in document |
| Replace | Replace a substring in document |
| Help | Show this manual |
| About | Show information about OpenVE |

**Build Toolbar and the corresponding Main Menu Items**

| | |
|---|---|
| Build | Build the current OpenComRTOS Project |
| Run ▶ | Build and run OpenComRTOS executables |
| Stop ■ | Stop all launched from OpenVE processes |
| Clean | Delete all files and folders generated during the build process |

**Topology toolbar**

| | |
|---|---|
| Arrow | Sets the Topology or Application Diagram into the editing mode |
| Node ● | Creates a node of chosen type on the Topology Diagram |
| Links ↘ or ⤡ | Creates a link (of unidirectional or bidirectional kind) |

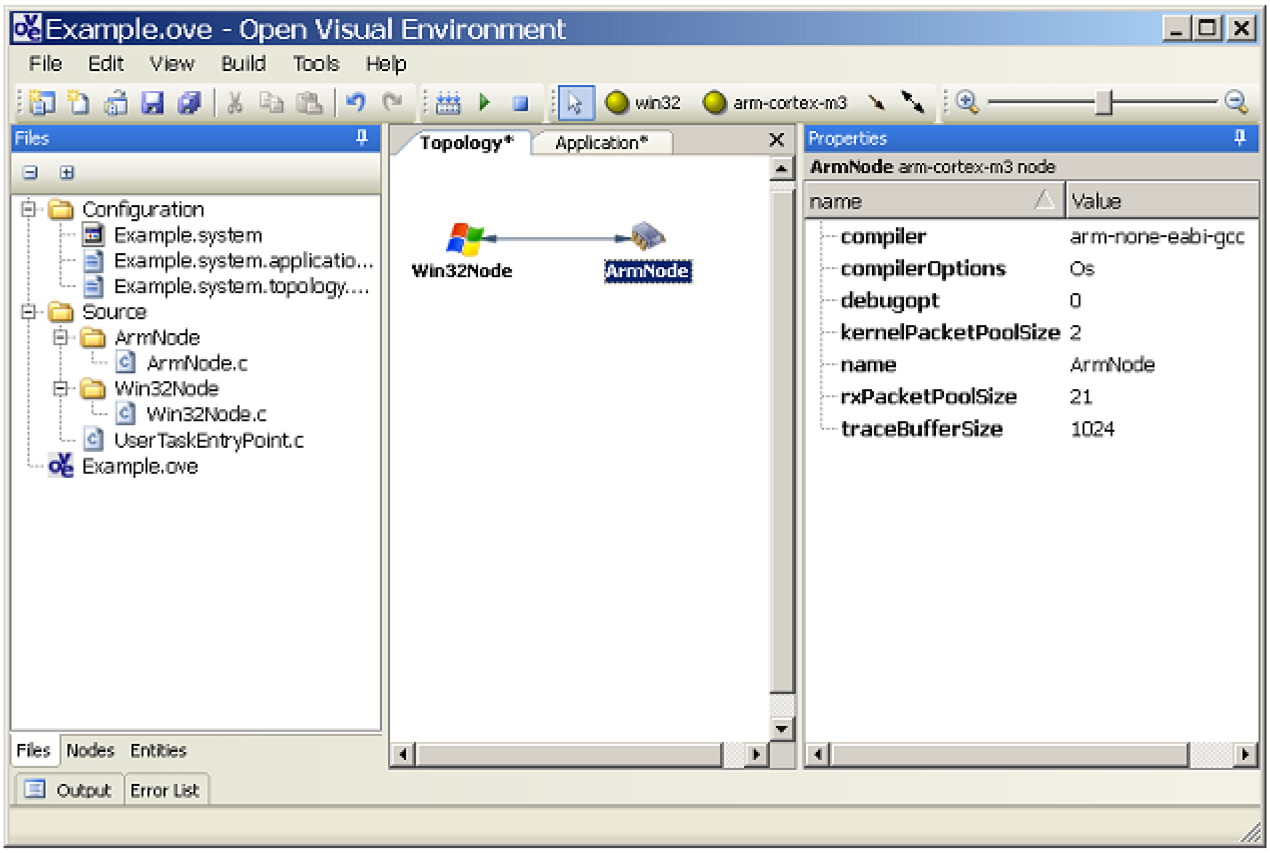
**Application Toolbar**
The structure of the application toolbar depends on chosen RTOS Metamodel

| | |
|---|---|
| T | Creates a Task |
| ◇ | Creates an Event Hub |
| ◈ | Creates a Semaphore Hub |
| 🔒 | Creates a Resource Hub |
| ▭ | Creates a FIFO Hub |
| ▦ | Creates a Memory Pool Hub |
| P | Creates a Port Hub |
| 🖥 | Create a Stdio Host Server |
| 🖥 | Create a Graphics Host Server |
| ↘ | Creates an Interaction between entities |

## 2.5. Zoom Bar

| |
|---|
| Allows scaling the Application or Topology diagram for a more detailed view. |

## 2.6. Topology diagram



ArmNode arm-cortex-m3 node

| name | Value |
|---|---|
| compiler | arm-none-eabi-gcc |
| compilerOptions | Os |
| debugopt | 0 |
| kernelPacketPoolSize | 2 |
| name | ArmNode |
| rxPacketPoolSize | 21 |
| traceBufferSize | 1024 |

**Figure 2-3. Using Topology Diagram**

A topology diagram defines the network topology of a processor network. It is consists of **Nodes** (Processing Entities) and **Links** (Communication links between individual Nodes). There can be more than one link between individual nodes. Links can be either unidirectional or bidirectional.

Thus the **base graphical Entities** of the Topology diagram are:

1. **Nodes**, graphically represented as an icon together with the name of the node e.g. Win32Node1.
2. **Links**, represented by a line between two nodes, with one ➘ (unidirectional link) or two arrow heads ⤡ (bidirectional link) connecting two nodes.

## 2.7. Application diagram

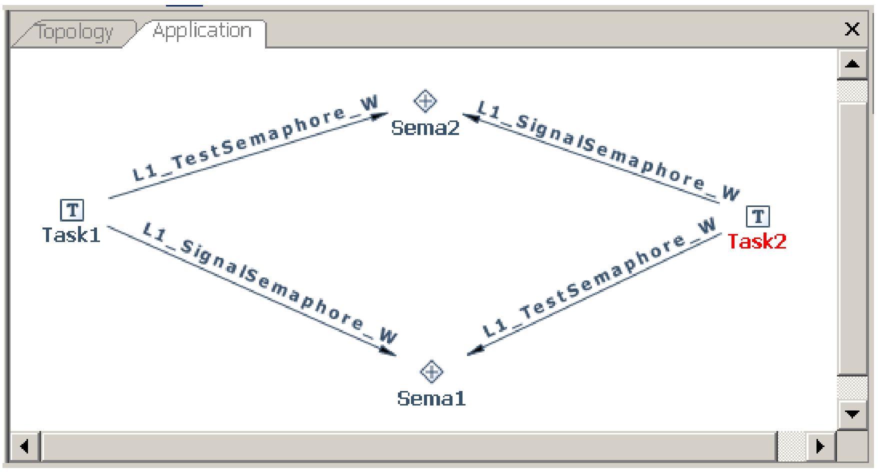


**Figure 2-4. Using Application Diagram**

An OpenComRTOS application consists of Task and Hubs (i.e. Ports, Events, FIFOs etc.). Tasks do not communicate directly but always utilize an intermediate Hub. In the application diagram the developer models these interactions graphically. The diagram consists of the following elements:

Thus the **base graphical entities** of the Application diagram are:

1. **Entities** (Task, Port, Event, FIFO, Packet Pool, Memory Pool, Resource, Semaphore) which are graphically represented by the nodes of the graph (e.g. Task1).
2. **Interactions**. The edges of the graph represent calls of the tasks to the kernel services. The type of service is indicated by a service name placed over the edge (e.g. L1_SignalSemaphore_W).

**Using the Application diagram to build an application**

- To create an **Entity** (e.g. Task, Port etc.) click on the appropriate button on the Application toolbar.

- To create an **Interaction** between entities (e.g. **L1_SignalSemaphore_W**) draw a link by clicking with the mouse first on a source entity and then release it on the target entity.

| **Note** |
|---|
| The interaction type depends on the start point of the interaction: starting on a task entity and release on a hub entity will generate the put type of interaction (e.g. L1_PutPacketToPort_W), starting on the hub entity – the get type (e.g. L1_GetPacketToPort_W) will be generated. |

- Select the required interaction from the drop down menu, press Ok.

- Add needed function parameters, declarations and operators in the opened text editor.

- To change the scale of a diagram use the **Zooming Tool**.

**Relation of the Application diagram with other modules of OpenVE**

Changing the Application diagram results in the following changes in the project:

- Upon adding a Task to the application diagram the Task Entry Point gets added to the list of tasks of the Node this Task is mapped.
- When adding an Interaction to a Task, the corresponding C function call is automatically added to the Task Entry Point source code.
- The arrangement of the Entities on the Application diagram is saved in the project map file.
- All Entities that are mapped on a Node are shown in the Nodes tree view.

## 2.8. Source Code Editor

The Application Diagram describes an OpenComRTOS application as a set of Interacting Entities. The behavior of a task between two interactions are defined by the using the Source Code Editor, which supports the C programming language.

Some parts of the source code are inserted generated automatically from the graphical representation in the Application diagram:

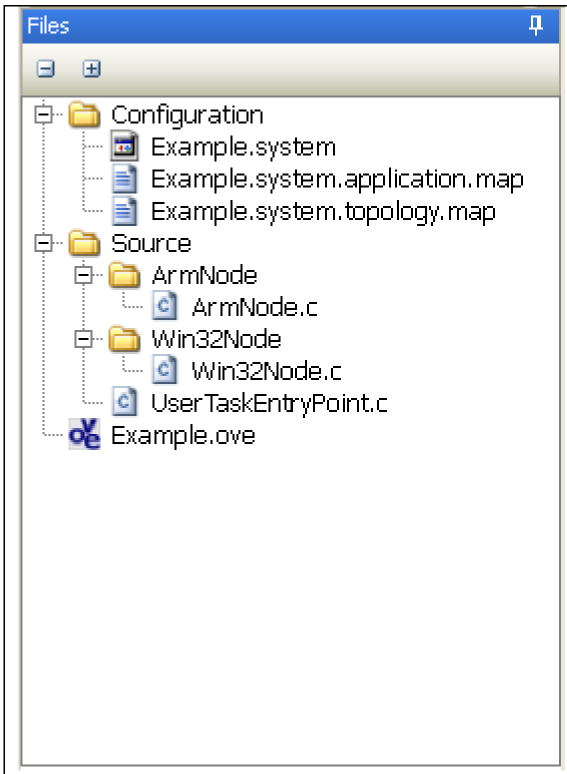
- For each new Task Entry Point, created in the Application Diagram, the corresponding C file will be created.
- Each added Interaction in the Application diagram results in the corresponding C function call to be inserted into the Task Entry Point.

The text editor has its own set of supported actions:

- Lines numbering.
- Vertical and horizontal text scrolling.
- Copy and Paste
- Undo/Redo text function.
- Syntax highlighting for the C programming language.

## 2.9. Navigation panels and Properties window

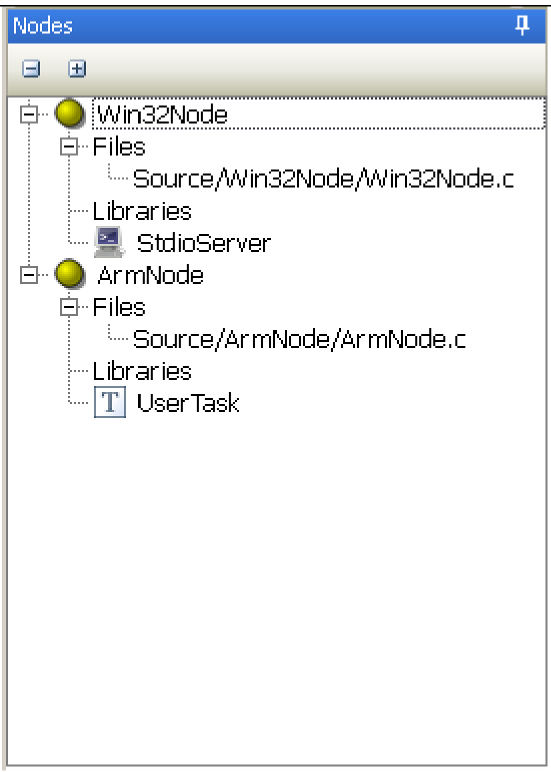
The Navigation panel group includes: Files, Nodes and Entities tree views.



```
Files
Configuration
    Example.system
    Example.system.application.map
    Example.system.topology.map
Source
    ArmNode
        ArmNode.c
    Win32Node
        Win32Node.c
    UserTaskEntryPoint.c
Example.ove
```

Figure 2-5. Files tree view

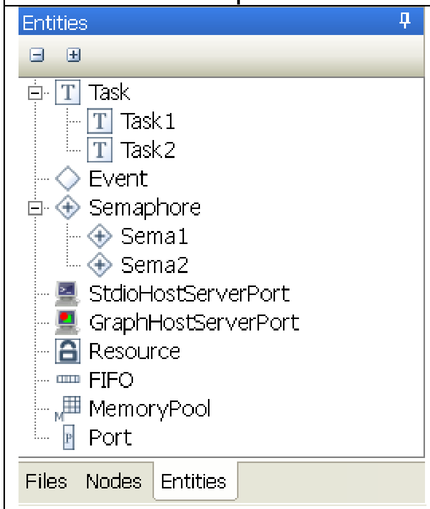
This window shows the files tree and gives you access to the project files and folders. Double-clicking on a *c, h, makefile, and project* file in the tree-view window opens it in the tab sheet.



```
Nodes
Win32Node
    Files
        Source/Win32Node/Win32Node.c
    Libraries
    StdioServer
ArmNode
    Files
        Source/ArmNode/ArmNode.c
    Libraries
    UserTask
```

Figure 2-6. Nodes tree view

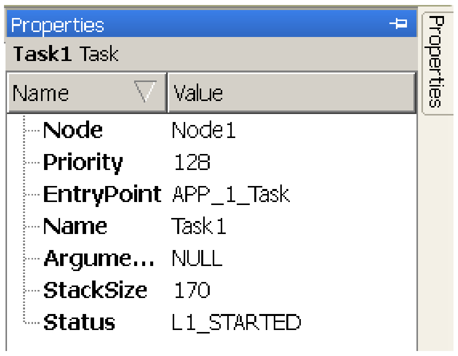
This window shows the nodes tree with all mapped kernel entities. Double-clicking on a node or a kernel entity shows its attributes in the Properties window.



```
Entities
Task
    Task1
    Task2
Event
Semaphore
    Sema1
    Sema2
StdioHostServerPort
GraphHostServerPort
Resource
FIFO
MemoryPool
Port
```

Figure 2-7. Entities tree view

This window shows all kernel entities grouped by type. Inside each group entities are sorted by name. Double-clicking on a kernel entity shows its attributes in the Properties window.



Properties

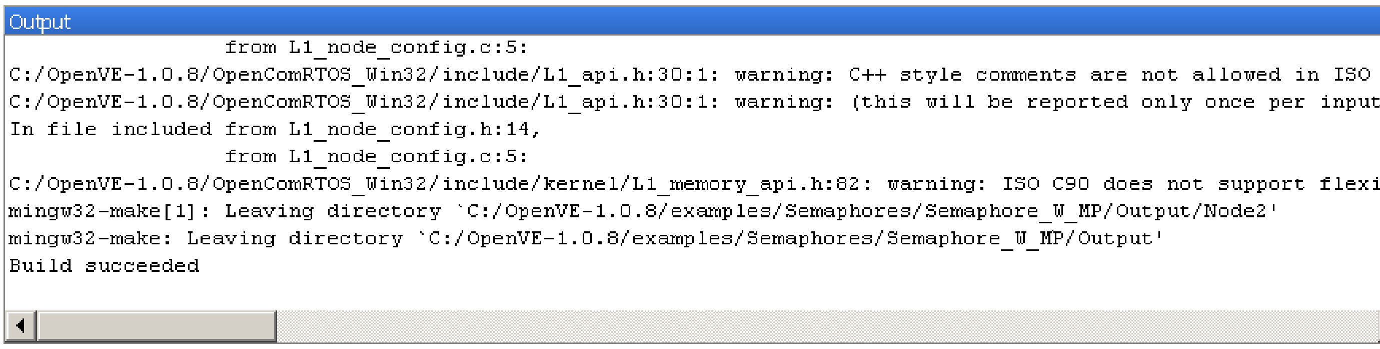
**Task1** Task

| Name | Value |
|---|---|
| **Node** | Node1 |
| **Priority** | 128 |
| **EntryPoint** | APP_1_Task |
| **Name** | Task1 |
| **Argume...** | NULL |
| **StackSize** | 170 |
| **Status** | L1_STARTED |

Figure 2-8. Properties window

This window shows the entity properties and allows a user edit the attributes of a selected kernel entity, node or link. Properties window allows attributes sorting by name and value

## 2.10. Output window

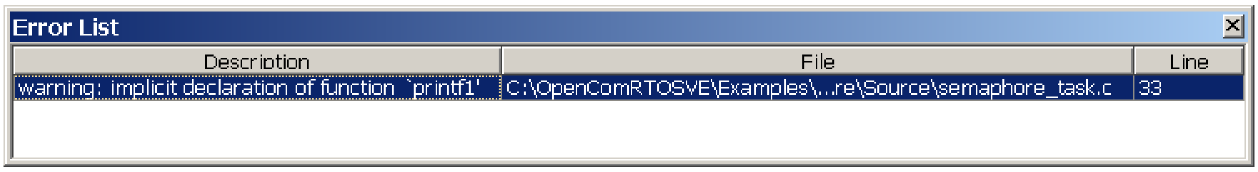


```
Output
                  from L1_node_config.c:5:
C:/OpenVE-1.0.8/OpenComRTOS_Win32/include/L1_api.h:30:1: warning: C++ style comments are not allowed in ISO
C:/OpenVE-1.0.8/OpenComRTOS_Win32/include/L1_api.h:30:1: warning: (this will be reported only once per input
In file included from L1_node_config.h:14,
                  from L1_node_config.c:5:
C:/OpenVE-1.0.8/OpenComRTOS_Win32/include/kernel/L1_memory_api.h:82: warning: ISO C90 does not support flexi
mingw32-make[1]: Leaving directory `C:/OpenVE-1.0.8/examples/Semaphores/Semaphore_W_MP/Output/Node2'
mingw32-make: Leaving directory `C:/OpenVE-1.0.8/examples/Semaphores/Semaphore_W_MP/Output'
Build succeeded
```

**Figure 2-9. Output window**

The output window shows the output generated by the tools used while building the project, among other messages it shows: warnings and errors generated by the compiler and linkers.

The error and warning messages are also shown in the Error list, which is part of the Output window. Double clicking on a message in the Error list will bring you to the source code location reported by the tool as the origin of the error (see Figure 2-10).



Error List

| Description | File | Line |
|---|---|---|
| warning: implicit declaration of function `printf1' | C:\OpenComRTOSVE\Examples\...re\Source\semaphore_task.c | 33 |

**Figure 2-10. Warning of OpenVE project compilation**

## 2.11. Options Dialogue

To change the default options of OpenVE use the Options dialogue, it is available in the Menu bar under Tools -> Options. The following options can be changed here (see Figure 2-11):

**User Interface:**

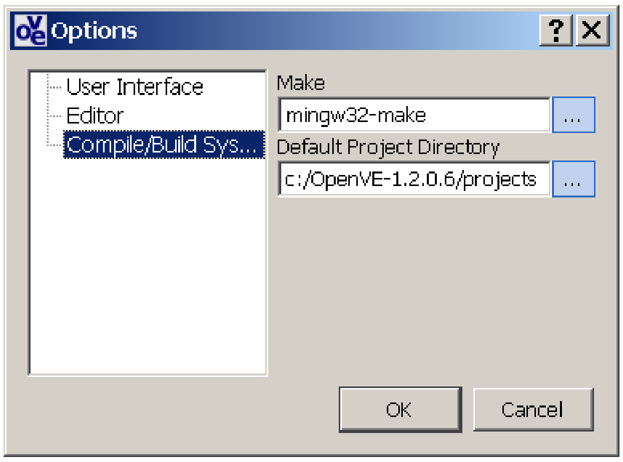
- Startup Behaviour--- Load last Project or start with an empty workspace;
- Automatically save all files before building.

**Editor:**

- Font to use;
- Highlight detected Errors in the Source code.

**Compile / Build System:**

- Make program to use;
- Default location where new projects should be created.



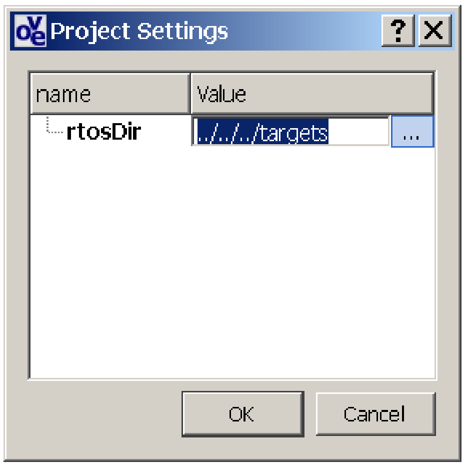
Make: mingw32-make

Default Project Directory: c:/OpenVE-1.2.0.6/projects

Figure 2-11. Options window

## 2.12. Project Settings Dialogue

The Project Settings Dialogue is used to define which OpenComRTOS Kernel image to use for the project (rtosDir). The dialogue is available in the menu bar under **Edit -> Project Settings** (see Figure 2-12).



| name | Value |
|---|---|
| rtosDir | ../../../targets |

Figure 2-12. Project settings window

# 3. Chapter 3 - Working with OpenVE Projects

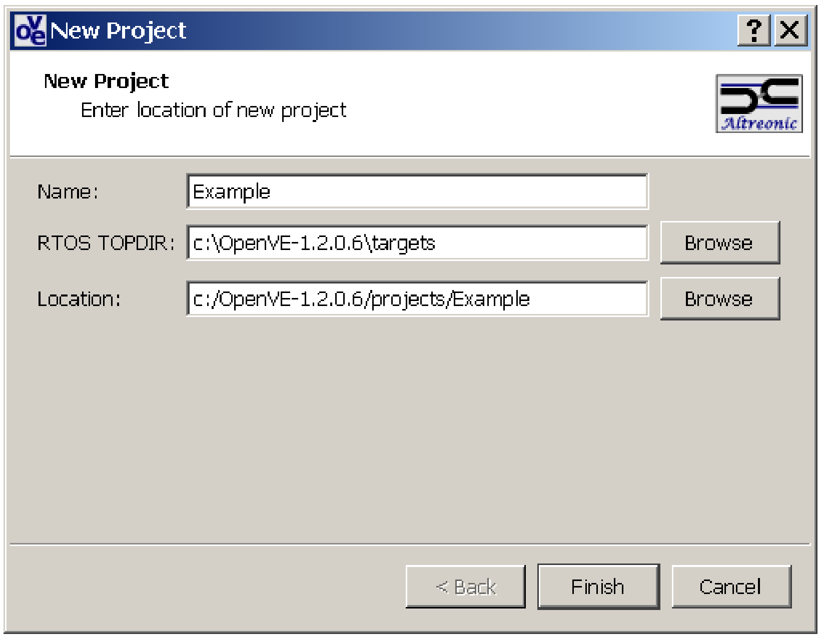
## 3.1. Introduction

This chapter explain the possible operations on OpenVE projects. It starts by explaining how to create a new project in 3.2. Section 3.3 then details how to open an existing project. How to add a new source file to a project is the subject of Section 3.4. Section 3.5 explains how to integrate already existing C files and libraries into a project. Finally an explanation of the general structure of an OpenVE project on the file system is given in 3.6.

## 3.2. Creating a new OpenVE Project

**To create a new OpenVE Project:**

- Click on **New Project** in either the menu or the toolbar.
- Enter the **Name** of the project (note, you cannot write spaces in the project name).
- Choose the OpenComRTOS kernel directory to use for the project (by default it's the **\targets** folder inside your OpenVE installation).
- Select the **Location** where the new project will be created in (note, by default OpenVE uses a project name as a name for the project folder).
- Click **Finish**.


New Project

New Project
Enter location of new project

Name: Example
RTOS TOPDIR: c:\OpenVE-1.2.0.6\targets
Location: c:/OpenVE-1.2.0.6/projects/Example

Figure 3-1. Dialog window of a new project creation

## 3.3. Opening an existing Project

To open an already existing project you have two possibilities:

- In the **File** menu click **Open Project** (or use shortcut: Ctrl+O). Browse your local folders for the project you wish to open and click on **OK.**
- You can also use the **Recent Projects** list in the Main menu.

## 3.4. How to add a new source file to a project

To create and add a new file to the current project follows these steps:

1. In the **File** menu (**File -> New -> New File**) or toolbar click **New File** (or use shortcut: Ctrl+N).
2. In the dialog window (see Figure 3-2) select the file type (c or h file) and enter a file name (e.g. *source.c*).

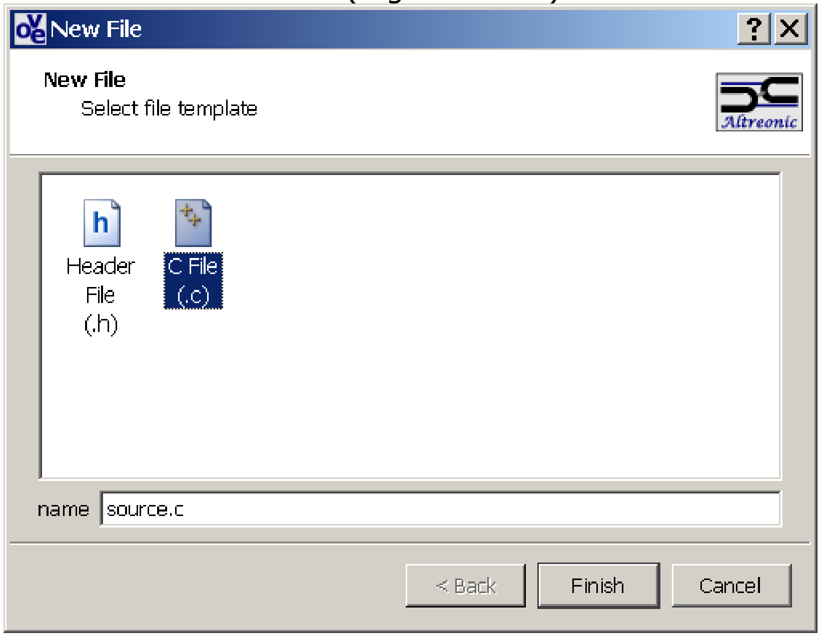


**Figure 3-2. Dialog window of a new file creation**

3. Click on **Finish.** A new file will be added to your project in the Source folder. Next you can manage the file by the Files tree-view.

Please note that this file has not yet been assigned to any Node, thus it will not be compiled. How to associate this file to a Node for compilation is explained in the next section.

**Note**

For creation of the h file an empty template is used.

For creation of the c file is used template given at Appendix A.

The Node file with definition of the main function is created automatically when we place a Node on topology diagram (the template given at Appendix B).

## 3.5. How to add an existing source file or lib to a project

To add an existing source file or lib follows these steps:

1. Do the right mouse click on a node on the Topology diagram to open a node context menu.
2. Choose from the context menu the Add File or the Add Lib option (see Figure 3-3).

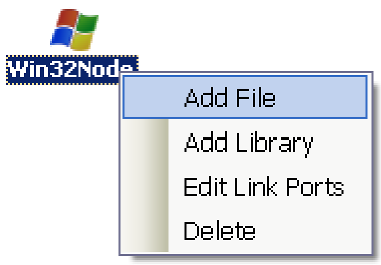


Figure 3-3. A node context menu

3. Specify in the opened dialog the file or the lib name.

## 3.6. Structure of the Project Data Folders

This is the folder where all current project files are stored.

| **Files and folders** | **Meaning** |
|---|---|
| **Configuration** folder | This folder stores all files of the current project configuration (with map and system extensions) |
| **Output** folder | This folder stores the generated files and compiled executables |
| **Source** folder | In this folder the source files of the project are stored |
| Project file | The OpenVE project file with the .ove extension. |

# 4. Chapter 4 - Developing a simple OpenComRTOS application

## 4.1. Introduction

This chapter explains all necessary steps to create a simple OpenComRTOS application. Starting from creating a new project in the Section 4.2 over the creation of a topology for the system in the Section 4.3, towards the application logic itself in the Section 4.4. This chapter closes by explaining how to build and run the application in Section 4.5.

## 4.2. Creating a new Project

To create a new project with OpenVE do the following:

1. Start OpenVE, either by using the shortcut or launch the executable file directly.

2. Create new project, following the instructions given in the Section 3.2 (for instance with the *Example* name).

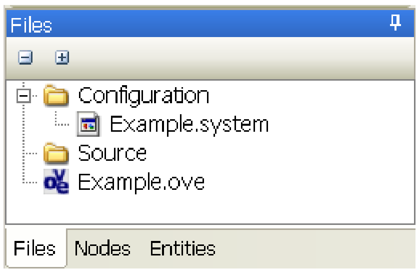


Figure 4-1. Folders structure of an Example project

Together with project file (here *Example.ove*) the application configuration file (here *Example.system)* file will be created.

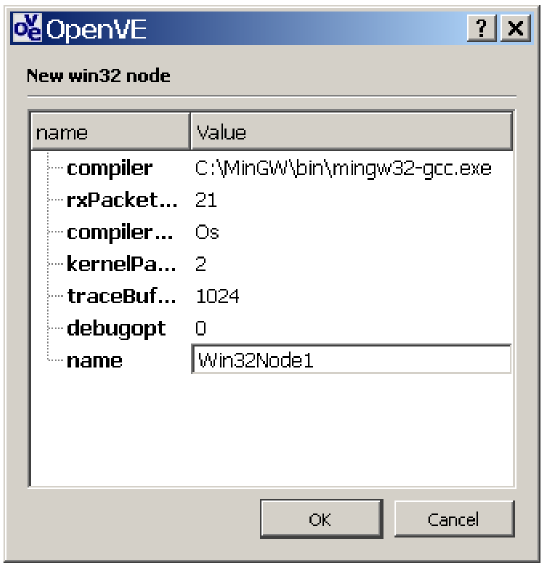
## 4.3. Defining the Topology of the System

Any OpenComRTOS application requires at least one Node[1], to create a node follow these instructions:

1. Activate the node button ● on the Topology Toolbar by pressing it and click with mouse pointer on the Topology Diagram to add a new node.

2. In the New node dialogue that opens (see Figure 4-2), specify the node name (e.g. Win32Node1), compiler (e.g. mingw32-gcc), and check the other node attributes. KernelPacketPoolSize and RxPacketPoolSize properties are needed for MP type of project (with default value set in 2 and 21 correspondingly).

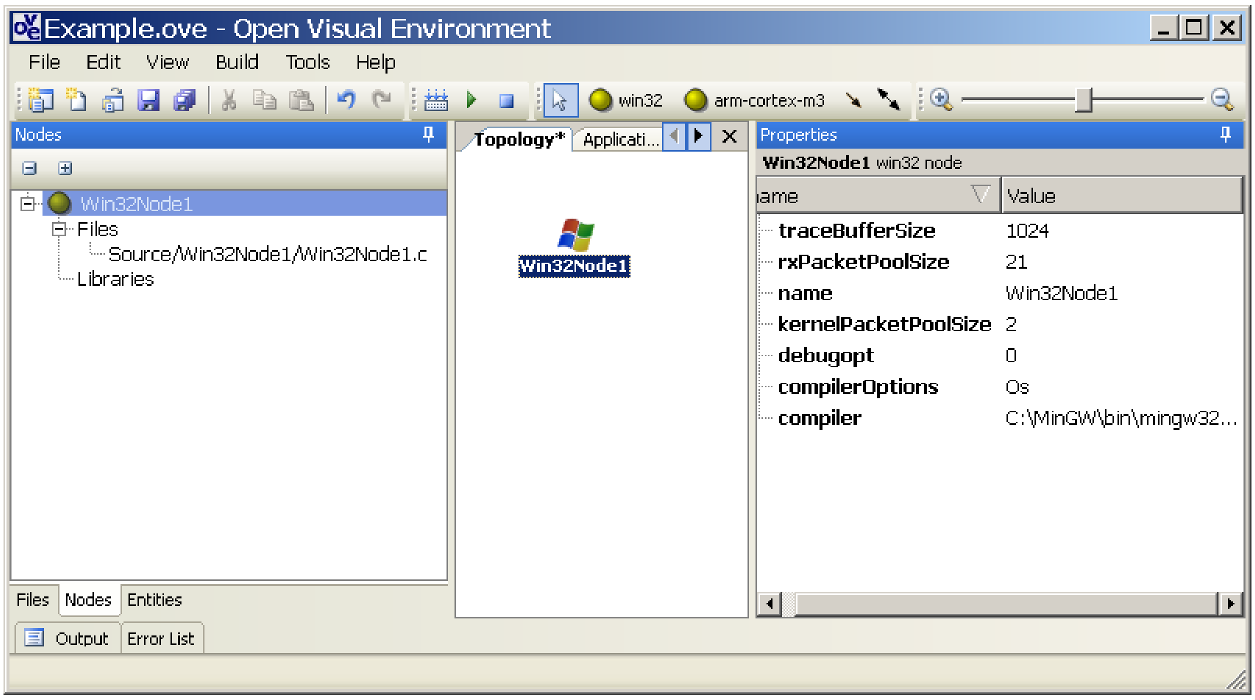
[1] **Node** – a processing device in a network containing CPU and with local memory and periphery



| name | Value |
|---|---|
| compiler | C:\MinGW\bin\mingw32-gcc.exe |
| rxPacket... | 21 |
| compiler... | Os |
| kernelPa... | 2 |
| traceBuf... | 1024 |
| debugopt | 0 |
| name | Win32Node1 |

**Figure 4-2. New node window**

| **Notes** |
|---|
| You may need to check the full path to compiler by the browse button, e.g. for the Windows type of node C:\MinGW\bin\mingw32-gcc.exe.<br><br>After a Node creation it can be found in the Nodes tree view (see the left panel on Figure 4-3)<br><br>All nodes attributes can be further modified in the Properties inspector (see the right panel on Figure 4-3) |



| name | Value |
|---|---|
| traceBufferSize | 1024 |
| rxPacketPoolSize | 21 |
| name | Win32Node1 |
| kernelPacketPoolSize | 2 |
| debugopt | 0 |
| compilerOptions | Os |
| compiler | C:\MinGW\bin\mingw32... |

**Figure 4-3. Specification of attributes of a node in the properties inspector**

| **Notes** |
|---|
| Setting debugopt into 1 (scheduling and Hub interactions) or 2 (scheduling, Hub interactions, task service requests) will enable the node to collect the execution trace. This execution trace can be written onto a disk using the StdioHostService function DumpTraceBuffer() and will be stored in a file with the name: |

opencomrtosNode<Num>.trace These files can be opened using Open Tracer (see the Open Tracer manual).

You can change a size of generated trace file by setting the traceBufferSize value for Node in the Properties Window.

## 4.4. The Minimal OpenComRTOS application

Your OpenComRTOS project should have at least the **main** function calling the L1_runOpenComRTOS function (the definition of this function is given in the L1_api.h):

```
int L1_runOpenComRTOS (int NodeNumberOfTasks, int NodeNumberOfHubs);
```

In the L1_node_config.h the number of tasks and hubs[2] per node are defined as constants, e.g.:

```
#define L1_NODE_NUMBER_OF_TASKS 2
#define L1_NODE_NUMBER_OF_HUBS 0
```

**Note**

The minimal L1_NODE_NUMBER_OF_TASKS is 2: i.e. for the kernel and idle task.

At the moment you press Ok in the New node window (see Figure 4-2) OpenVE automatically create the Node folder and file with the Node name (Win32Node1.c in our example):

```
/* Created <Date> <Time> <Year> */

#include <L1_api.h>
#include <L1node_config.h>

int main(void)

{
    return L1_runOpenComRTOS(L1_NODE_NUMBER_OF_TASKS,
          L1_NODE_NUMBER_OF_HUBS);
}
```

You can already build and run your minimal OpenComRTOS application.

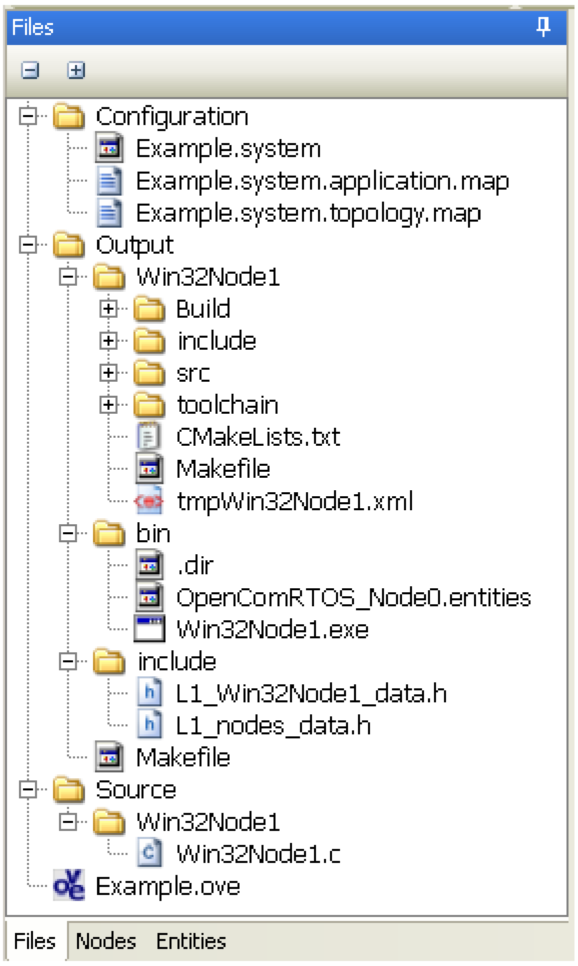
You can already build and run your minimal OpenComRTOS application. Press the Run ▶ to build and start the OpenComRTOS on the node, which was defined on the Topology Diagram. Note, however that because there are no application tasks the resulting program will only idle.

**Note**

Before doing a Build OpenVE automatically save all files in the project. At a first project compilation OpenVE will show notification window

Explore the file structure of your Example project using the Files Tree View (see Figure 4-4).

[2] **Hub** is the generalized concept of OpenComRTOS kernel entity.



```
Files
Configuration
    Example.system
    Example.system.application.map
    Example.system.topology.map
Output
    Win32Node1
        Build
        include
        src
        toolchain
        CMakeLists.txt
        Makefile
        tmpWin32Node1.xml
    bin
        .dir
        OpenComRTOS_Node0.entities
        Win32Node1.exe
    include
        L1_Win32Node1_data.h
        L1_nodes_data.h
    Makefile
Source
    Win32Node1
        Win32Node1.c
Example.ove
Files  Nodes  Entities
```

Figure 4-4. Project files structure after compilation

- In the Source folder there is a Node folder for each node, it has the same name as the Node (here Win32Node1).
- The Node folder contains the Node Entry Point, which is the C file with the main() function. This file has the same name as the Node (here Win32Node1.c).
- Within the Output directory the code generator have generated additional files which are used to configure and build the OpenComRTOS application.

## 4.5. Adding an Application Task

An Application Task is an active entity in OpenComRTOS. To create and add a new Application Task to the project follows these instructions.

1. Click on the Task T button in the Application toolbar and then click on the Application diagram.
2. In the New Task creation dialogue (see Figure 4-5) you have to specify the **Node** which executes this task.

| **Note** |
|---|
| You cannot create any kernel entity (e.g. Task or Port) if you have not created at least one node before. |

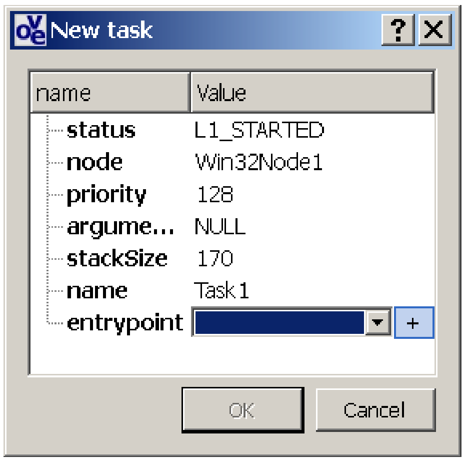
3. Specify the name of the Task (for example *Task1*).
4. You should also specify (or change set by default) other Task attributes.

| **Note** |
|---|
| Priority 1 is the highest and reserved for the Kernel Task, 255 is the lowest and |

reserved for the Idle Task. Higher priorities (e.g. 2, 3 etc.) are used for driver Tasks. Application tasks priorities should have a lower priority than the driver tasks.

For Nodes of type Win32 a stack size of 170 is acceptable. For nodes of other type please refer to the documentation provided with the Kernel Image.

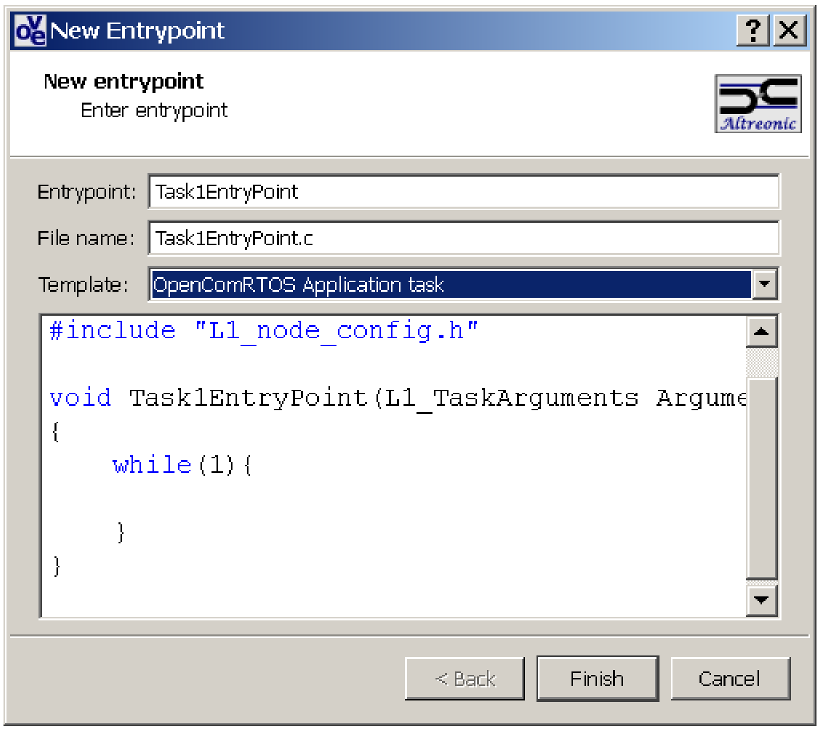
5. Choose an Entry Point of a Task from the EntryPoint drop-down list (*if the EntryPoint was already defined in the source code*) or press the plus button + to create a new Entry Point.



| name | Value |
|---|---|
| status | L1_STARTED |
| node | Win32Node1 |
| priority | 128 |
| argume... | NULL |
| stackSize | 170 |
| name | Task1 |
| entrypoint | |

Figure 4-5. New Task creation wizard

6. Enter the Entry point of a c function (in our example – *Task1EntryPoint* see Figure 4-6) and click the Next button (note, for Entry Point creation will be used the template from the Metamodel.template file – see Appendix C).

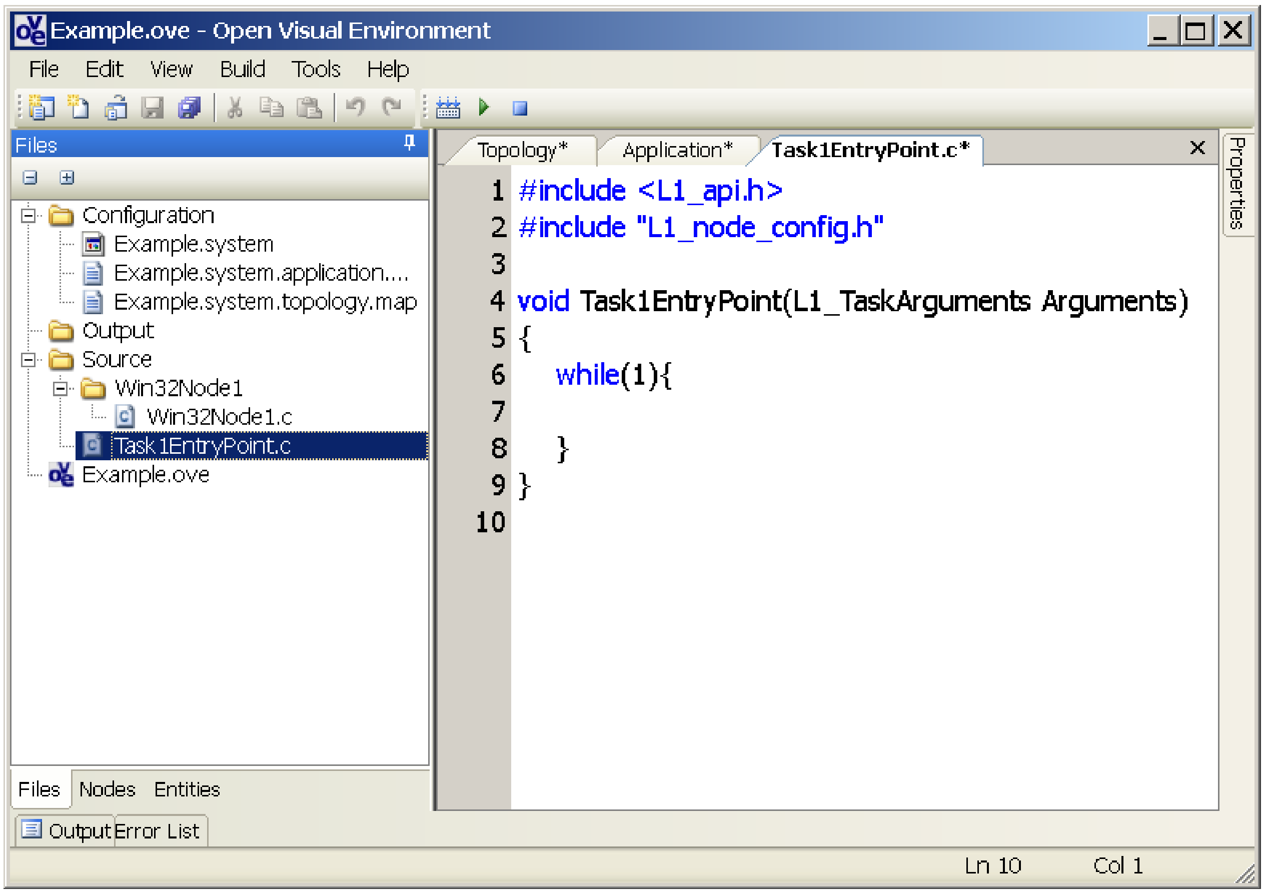


Entrypoint: Task1EntryPoint
File name: Task1EntryPoint.c
Template: OpenComRTOS Application task

```
#include "L1_node_config.h"

void Task1EntryPoint(L1_TaskArguments Argume
{
    while(1){

    }
}
```

Figure 4-6. New Entry Point creation wizard

7. Click on Finish and a new file with the name entered in the New Entry Point creation window will be added in the source folder of your project (see Figure 4-7).


Example.ove - Open Visual Environment
File Edit View Build Tools Help
Files
Configuration
Example.system
Example.system.application....
Example.system.topology.map
Output
Source
Win32Node1
Win32Node1.c
Task1EntryPoint.c
Example.ove
Files Nodes Entities
Output Error List
Topology* Application* Task1EntryPoint.c*
Properties
Ln 10 Col 1

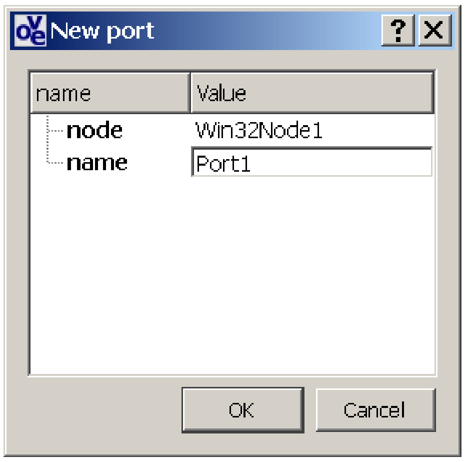
```
#include <L1_api.h>
#include "L1_node_config.h"

void Task1EntryPoint(L1_TaskArguments Arguments)
{
    while(1){

    }
}

```

Figure 4-7. Generation of the entry point from template

8. The template of an Entry Point (here Task1EntryPoint) is inserted in the source file (here *Task1EntryPoint.c*).

## 4.6. Creating a Port Entity.

Change to the Application diagram and click on the Port button P , then click on the Application diagram to add the Port. In the dialogue you have to specify the name of the Port and the Node where the port will be located (see Figure 4-8). To create the port click on OK.



| name | Value |
|---|---|
| node | Win32Node1 |
| name | Port1 |

Figure 4-8. New Port Creation Window

**Note**
You can change the attributes of any OpenComRTOS entity later using the Properties window.

## 4.7. Specifying Interactions

To specify an interaction follow these steps:

1. Press the Interaction button on Application diagram toolbar.
2. Connect using the mouse Task and Port icons on the Application diagram (see Figure 4-9).
3. In the drop-down list choose the required interaction between the selected entities (for example **L1_PutPacketToPort_W** - see Figure 4-9). Note, the allowable for the Entity interactions depend on Metamodel.

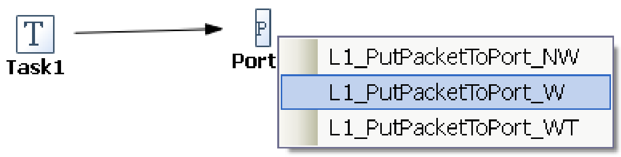
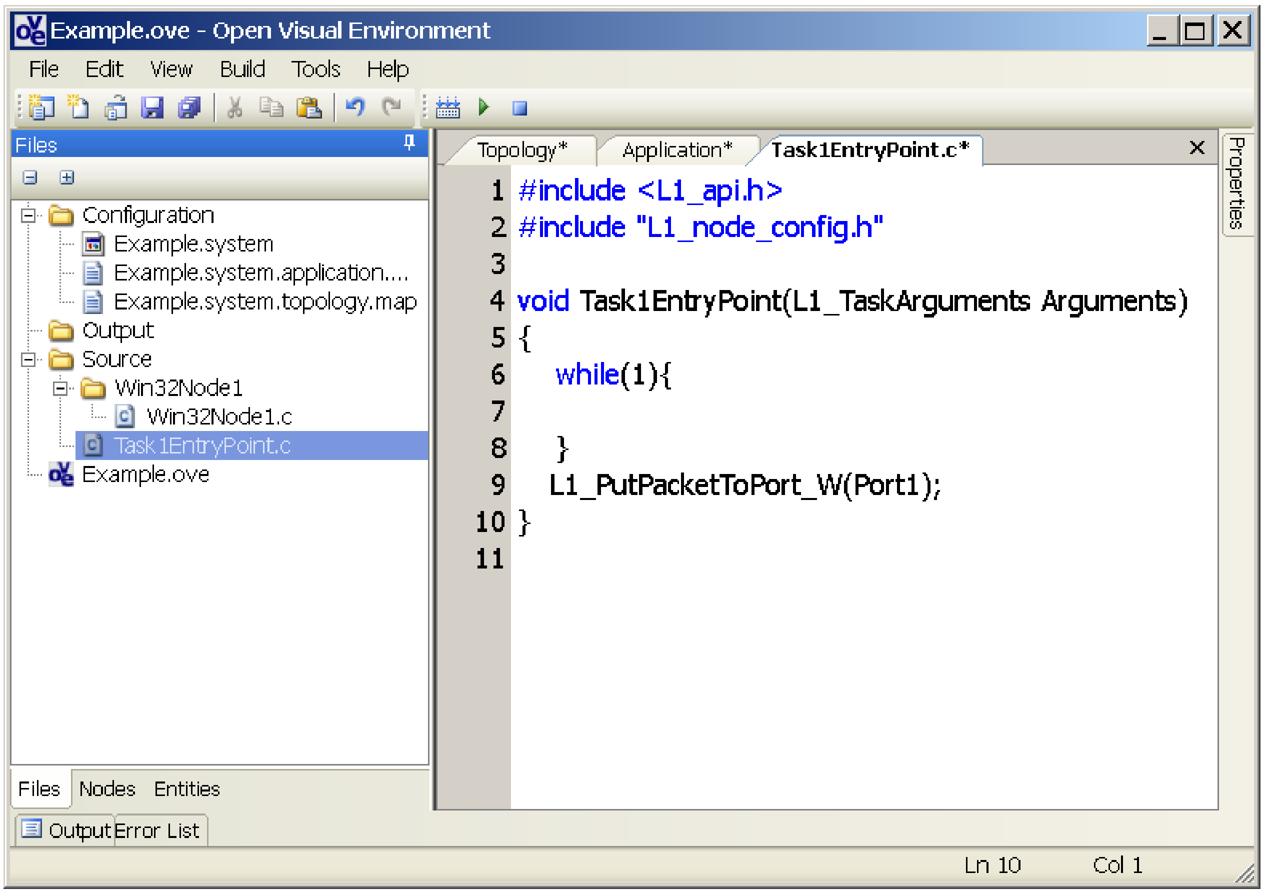


**Figure 4-9. Specifying Interaction between Entities**

4. The selected interaction will be placed at the end of the EntryPoint (here *Task1EntryPoint*) of the corresponding Task (here *Task1*) - see Figure 4-10.


Example.ove - Open Visual Environment
File Edit View Build Tools Help
Files
Configuration
Example.system
Example.system.application....
Example.system.topology.map
Output
Source
Win32Node1
Win32Node1.c
Task1EntryPoint.c
Example.ove
Files Nodes Entities
Output Error List
Topology* Application* Task1EntryPoint.c*
Properties
Ln 10 Col 1

```
#include <L1_api.h>
#include "L1_node_config.h"

void Task1EntryPoint(L1_TaskArguments Arguments)
{
    while(1){

    }
   L1_PutPacketToPort_W(Port1);
}
```

**Figure 4-10. Generation of interaction**

5. Use the text editor to correct the automatically generated text (here e.g. you should place **L1_PutPacketToPort_W** inside the **while(1)** operator). Some functions (like _WT) require additional parameters that you have to add manually, refer to the API manual for this information.

| **Note** |
|---|
| The text of an Interaction and its visual representation (i.e. the connecting Tasks and Hubs arrow) is automatically synchronized.<br><br>If you've made an error in the source code OpenVE cannot understand it and thus not include it in the Application Diagram. |

6. To construct the minimal *correct* OpenComRTOS application you should add a Task2 (with e.g. Task2EntryPoint entry point) following all described above steps and specify **L1_GetPacketFromPort_W** interaction. *Correct* OpenComRTOS application in our case means corresponding to the interaction symmetry principle: if you *put* a packet to a port you should also *get* it from there.

| **Note** |
|---|
| All tasks in OpenComRTOS interact only through special synchronization entities – Hubs (see Figure 4-11). |

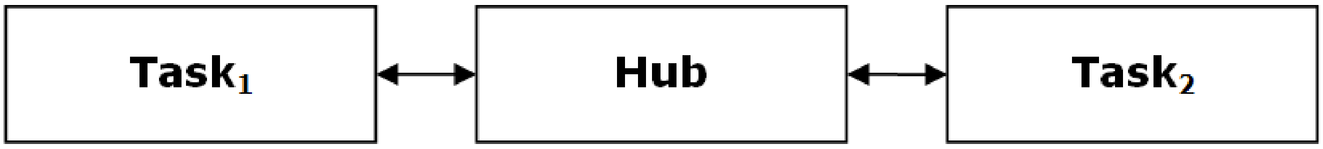


**Figure 4-11. Interaction via intermediate entity - Hub**

## 4.8. Building, Running and Stopping a Project

To build the project and then run it follow these steps:

1. Click on **Build** in the Main menu. The application configuration c and h files will be generated in the Output folder. OpenVE will generate the Makefile in the project folder and call MinGW make. MinGW compiles all source files and links them with libraries in executable images with the node names.

2. Press **Run** button or the menu item to compile and run the executable files of your project.

3. After exploring your first OpenComRTOS application don't forget close all applications lunched by OpenVE. For this purpose you can use the stop button.

## 4.9. Using a Port Entity to transfer data between tasks

A packet in OpenComRTOS consists of two parts: the header and the payload.

Section 4.8 explained how to utilize a port to synchronies two tasks. A data transfer can be considered a side effect of synchronizing on a Port.

To transfer data using a Port all that has to be done is to write the data into the payload of the RequestPacket of the task (L1_CurrentTaskCR-> RequestPacket -> Data) and then set the datasize (L1_CurrentTaskCR -> RequestPacket -> DataSize) of the packet to the number of bytes that should be transferred (maximum datasize is 32 byte).

The following guides you through the steps to extend the existing example to utilise the Port to exchange data between two tasks:

1. To simplify the access to the RequestPacket of the task, add the following line to the Task Entry Points: Task1EntryPoint and Task2EntryPoint.

   **L1_Packet *Packet = L1_CurrentTaskCR->RequestPacket;**

2. In this example we want to transfer only a single byte thus we set the data size of the Packet to 1, using the following line:

   **Packet->DataSize = sizeof(L1_BYTE);**

3. To transfer the letter 'a' we have to insert it into the first byte of the Packet payload:

   **Packet->Data[0] = 'a';**

4. Upon synchronization with Task1, the payload will be in RequestPacket of Task2. Thus the following snippet can be used access it:

   **Packet->Data[0];**

5. In OpenComRTOS Applications the use of the stdio functions to access the console (printf, scanf) is discouraged. Instead we provide a special StdioHostService which provides similar functionality (see Chapter 5 - for more information regarding this). For now simply add a StdioHostServer to the application diagram, and call it: StdioHostServer1.

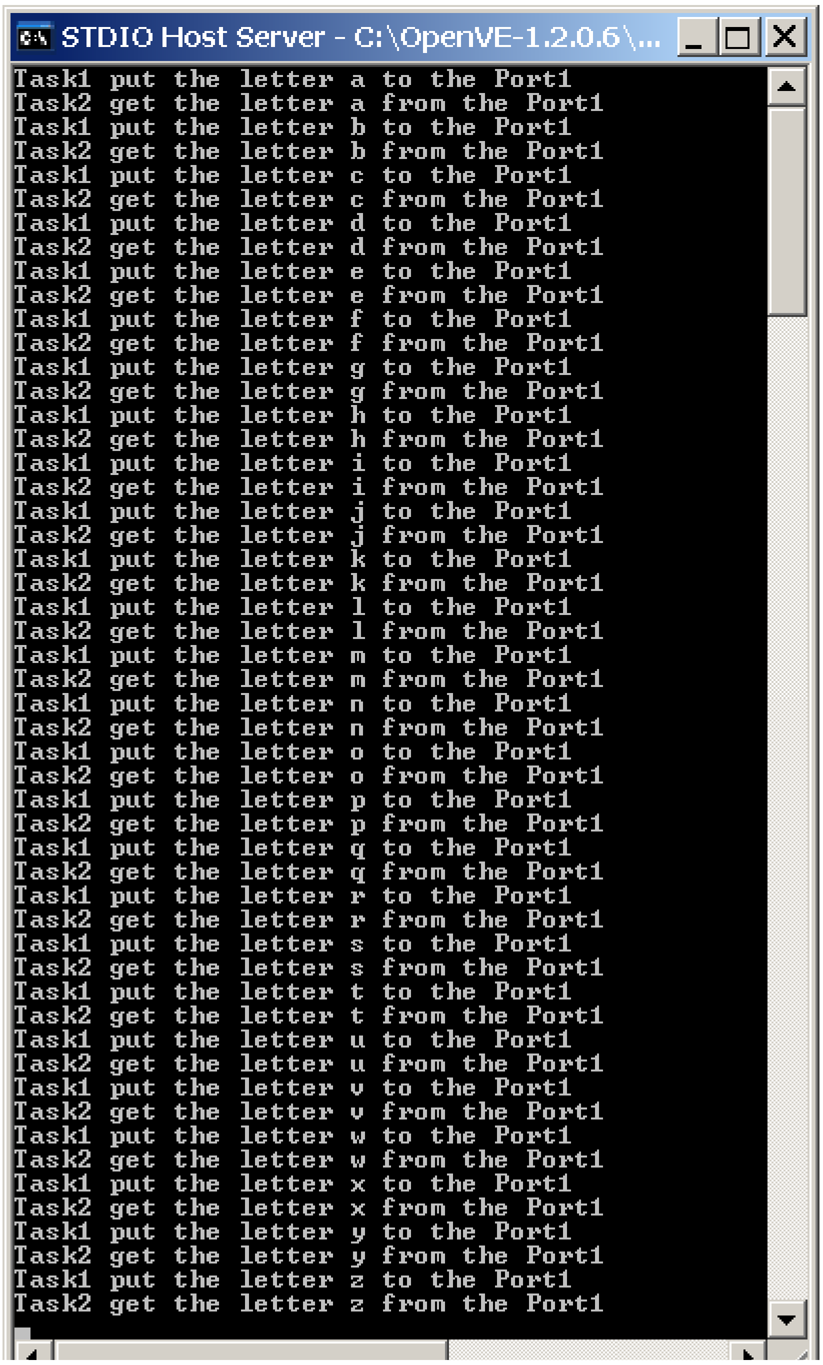
6. With the Stdio Host Server being a shared resource between two tasks, it is necessary to guard it against concurrent access. For this purpose add a Resource to the Application diagram and call it: Resource1.

7. Modify the two Task Entry Points: Task1EntryPoint and Task2EntryPoint according to the following listings:

```
void Task1EntryPoint(L1_TaskArguments Arguments)
{
  L1_BYTE ch;
  L1_Packet*Packet = L1_CurrentTaskCR->RequestPacket;
  for(ch='a'; ch<='z'; ch++)
  {
   Packet->DataSize= sizeof(L1_BYTE);
   Packet->Data[0] = ch;
   L1_PutPacketToPort_W(Port1);
   L1_LockResource_W(Resource1);
   Shs_putString(StdioHostServer1, "Task1 put the letter ");
   Shs_putChar(StdioHostServer1, ch);
   Shs_putString(StdioHostServer1, " to the Port1\n");
   L1_UnlockResource_W(Resource1);
}

}
```

```
void Task2EntryPoint(L1_TaskArguments Arguments)
{
  L1_BYTE i, ch;
  L1_Packet*Packet = L1_CurrentTaskCR->RequestPacket;
  for(i='a'; i<='z'; i++)
  {
    Packet->DataSize= sizeof(L1_BYTE);
    L1_GetPacketFromPort_W(Port1);
    ch = Packet->Data[0];
    L1_LockResource_W(Resource1);
    Shs_putString(StdioHostServer1, "Task2 get the letter ");
    Shs_putChar(StdioHostServer1, ch);
    Shs_putString(StdioHostServer1, " from the Port1\n");
    L1_UnlockResource_W(Resource1);
  }

}
```

8. After example compilation and starting you will see the next console window output (Figure 4-12):



```
Task1 put the letter a to the Port1
Task2 get the letter a from the Port1
Task1 put the letter b to the Port1
Task2 get the letter b from the Port1
Task1 put the letter c to the Port1
Task2 get the letter c from the Port1
Task1 put the letter d to the Port1
Task2 get the letter d from the Port1
Task1 put the letter e to the Port1
Task2 get the letter e from the Port1
Task1 put the letter f to the Port1
Task2 get the letter f from the Port1
Task1 put the letter g to the Port1
Task2 get the letter g from the Port1
Task1 put the letter h to the Port1
Task2 get the letter h from the Port1
Task1 put the letter i to the Port1
Task2 get the letter i from the Port1
Task1 put the letter j to the Port1
Task2 get the letter j from the Port1
Task1 put the letter k to the Port1
Task2 get the letter k from the Port1
Task1 put the letter l to the Port1
Task2 get the letter l from the Port1
Task1 put the letter m to the Port1
Task2 get the letter m from the Port1
Task1 put the letter n to the Port1
Task2 get the letter n from the Port1
Task1 put the letter o to the Port1
Task2 get the letter o from the Port1
Task1 put the letter p to the Port1
Task2 get the letter p from the Port1
Task1 put the letter q to the Port1
Task2 get the letter q from the Port1
Task1 put the letter r to the Port1
Task2 get the letter r from the Port1
Task1 put the letter s to the Port1
Task2 get the letter s from the Port1
Task1 put the letter t to the Port1
Task2 get the letter t from the Port1
Task1 put the letter u to the Port1
Task2 get the letter u from the Port1
Task1 put the letter v to the Port1
Task2 get the letter v from the Port1
Task1 put the letter w to the Port1
Task2 get the letter w from the Port1
Task1 put the letter x to the Port1
Task2 get the letter x from the Port1
Task1 put the letter y to the Port1
Task2 get the letter y from the Port1
Task1 put the letter z to the Port1
Task2 get the letter z from the Port1
```

**Figure 4-12. Send-receive packet with via Port (SP)**

## 4.10. The Return Values of OpenComRTOS Services

All functions of the OpenComRTOS API have a return value which should be checked to guarantee the correct functioning of the application.

There are three different return values possible, but not all functions do return all different types:

- RC_OK: The service completed successfully;
- RC_FAIL: The service failed to complete;
- RC_TO: The timeout expired. This only occurs when using services which can timeout, the postfix `_WT' indicates such a service.

Modify call to **L1_PutPacketToPort_W** and **L1_GetPacketFromPort_W** in example in such a way:

```
if (RC_OK == L1_PutPacketToPort_W(Port1))
{
  …
}

if(RC_OK == L1_GetPacketFromPort_W(Port1))
 {
  …
 }
```

In between { } brackets put the code, was defined in previous section.

## 4.11. Extending the Example to run on multiple Nodes

To develop Multiple Processers project you must extend the topology with a second Node and set a link between the two nodes:

1. Add an additional node to the topology with Win32Node2 Name (as was described in the section 4.3).

2. Add to each node a link port:

- Right click on the Win32Node1 and select from the context menu the "Edit Link Ports" item.
- Choose from drop-down menu the tcp type of link and press the plus button (see Figure 4-13).
- Specify attributes of the newly created link port. The attributes you can modify are the name and the port number of the socket. In this example it is only necessary to modify the name of the port, set it to *socket1* (see Figure 4-14).

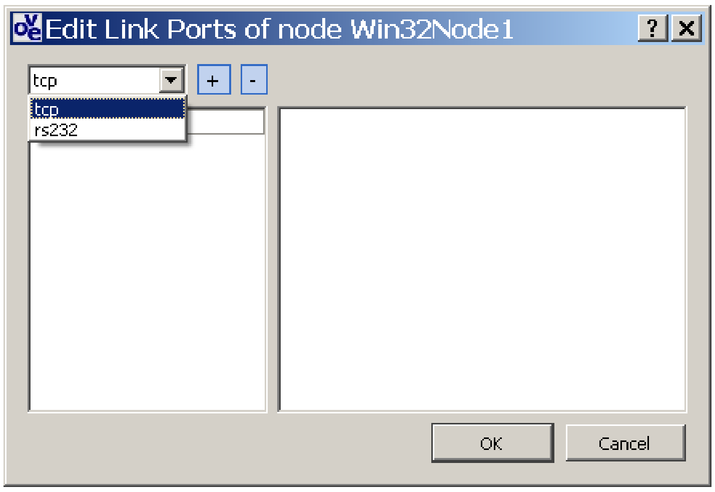
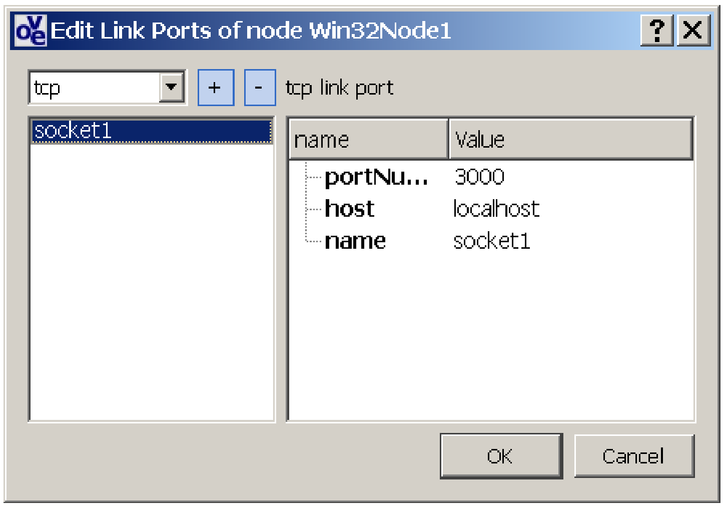


**Figure 4-13. Create a link port of a node**



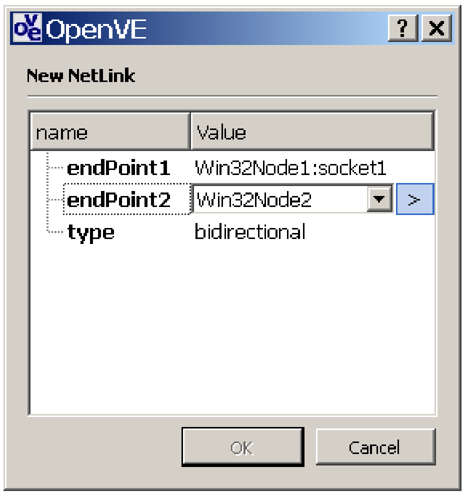
| name | Value |
|---|---|
| portNu... | 3000 |
| host | localhost |
| name | socket1 |

**Figure 4-14. Specification of a link port attributes**

3. Repeat the step 2 to define link port of the Win32Node2 node.

4. Connect the two Nodes using a bidirectional link.

- Activate the button on the Topology Toolbar to select the bidirectional link type.
- Connect two nodes by clicking with the mouse first on a source node and then release it on the target node.
- In the opened dialog specify endPoint1 and endPoint2 (see Figure 4-15).

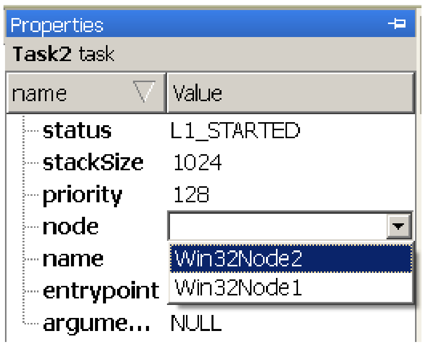


| name | Value |
|---|---|
| endPoint1 | Win32Node1:socket1 |
| endPoint2 | Win32Node2 |
| type | bidirectional |

**Figure 4-15. Specification of NetLink between nodes**

| Note |
|---|
| Any Node should have an incoming and an outcoming link. If a Node can't reach another node the build process will fail. |

5. To execute Task2 on Win32Node2, modify the Node it is mapped to using the properties inspector (see Figure 4-16).


Properties
Task2 task

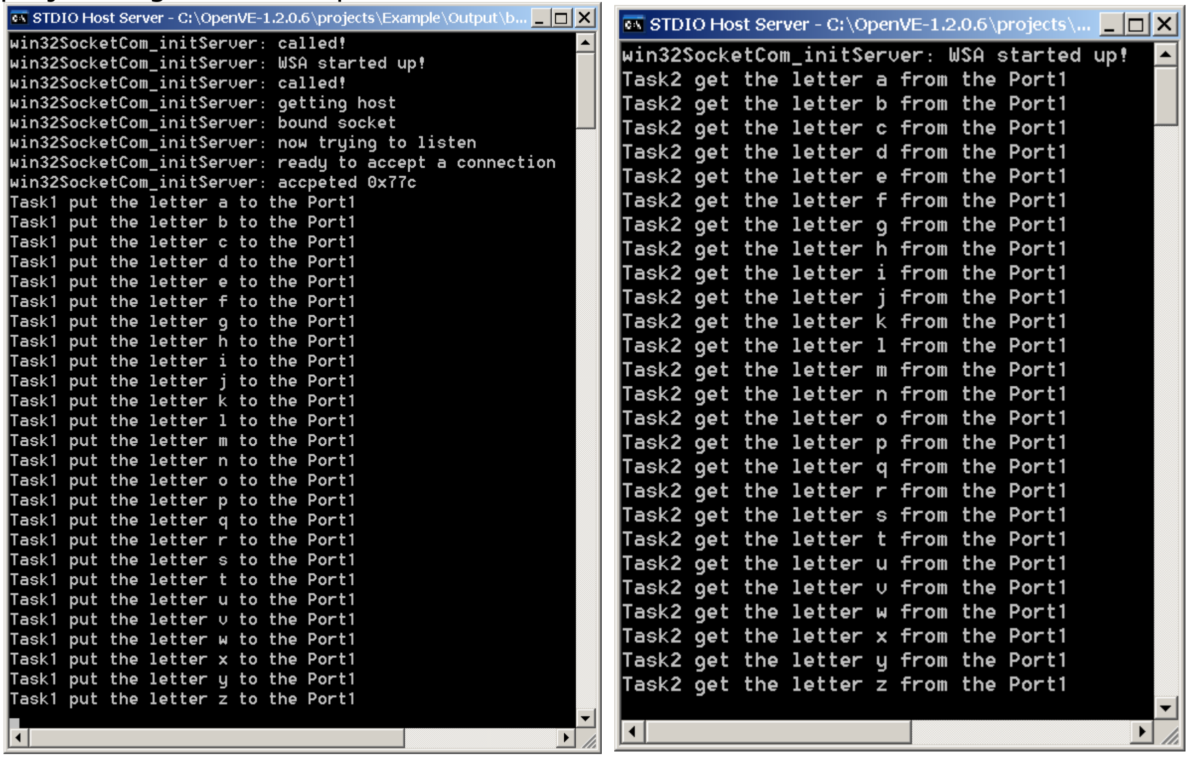
| name | Value |
|---|---|
| status | L1_STARTED |
| stackSize | 1024 |
| priority | 128 |
| node | |
| name | Win32Node2 |
| entrypoint | Win32Node1 |
| argume... | NULL |

Figure 4-16. Setting the Node property for a Task

6. For best performance of your MP application you can specify for RxPacketPoolSize and KernelPacketPoolSize bigger values then given by default.
7. Add on application diagram additional Stdio Host Server and set the name as StdioHostServer2. Map StdioHostServer2 on the Win32Node2.
8. Modify the entrypoint definition of Task2 in order the output goes into StdioHostServer2.

```
Shs_putString(StdioHostServer2, "Task2 get the letter ");
Shs_putChar(StdioHostServer2, ch);
Shs_putString(StdioHostServer2, " from the Port1\n");
```

9. Press **Run** button or the menu item to run the executable files of your project. Figure 4-17 represents the results.


STDIO Host Server - C:\OpenVE-1.2.0.6\projects\Example\Output\b...

```
win32SocketCom_initServer: called!
win32SocketCom_initServer: WSA started up!
win32SocketCom_initServer: called!
win32SocketCom_initServer: getting host
win32SocketCom_initServer: bound socket
win32SocketCom_initServer: now trying to listen
win32SocketCom_initServer: ready to accept a connection
win32SocketCom_initServer: accpeted 0x77c
Task1 put the letter a to the Port1
Task1 put the letter b to the Port1
Task1 put the letter c to the Port1
Task1 put the letter d to the Port1
Task1 put the letter e to the Port1
Task1 put the letter f to the Port1
Task1 put the letter g to the Port1
Task1 put the letter h to the Port1
Task1 put the letter i to the Port1
Task1 put the letter j to the Port1
Task1 put the letter k to the Port1
Task1 put the letter l to the Port1
Task1 put the letter m to the Port1
Task1 put the letter n to the Port1
Task1 put the letter o to the Port1
Task1 put the letter p to the Port1
Task1 put the letter q to the Port1
Task1 put the letter r to the Port1
Task1 put the letter s to the Port1
Task1 put the letter t to the Port1
Task1 put the letter u to the Port1
Task1 put the letter v to the Port1
Task1 put the letter w to the Port1
Task1 put the letter x to the Port1
Task1 put the letter y to the Port1
Task1 put the letter z to the Port1
```

STDIO Host Server - C:\OpenVE-1.2.0.6\projects\...

```
win32SocketCom_initServer: WSA started up!
Task2 get the letter a from the Port1
Task2 get the letter b from the Port1
Task2 get the letter c from the Port1
Task2 get the letter d from the Port1
Task2 get the letter e from the Port1
Task2 get the letter f from the Port1
Task2 get the letter g from the Port1
Task2 get the letter h from the Port1
Task2 get the letter i from the Port1
Task2 get the letter j from the Port1
Task2 get the letter k from the Port1
Task2 get the letter l from the Port1
Task2 get the letter m from the Port1
Task2 get the letter n from the Port1
Task2 get the letter o from the Port1
Task2 get the letter p from the Port1
Task2 get the letter q from the Port1
Task2 get the letter r from the Port1
Task2 get the letter s from the Port1
Task2 get the letter t from the Port1
Task2 get the letter u from the Port1
Task2 get the letter v from the Port1
Task2 get the letter w from the Port1
Task2 get the letter x from the Port1
Task2 get the letter y from the Port1
Task2 get the letter z from the Port1
```

Figure 4-17. Send-receive packet with via the Port (MP)

# 5. Chapter 5 - Host Services in OpenComRTOS

## 5.1. Introduction

This chapter explains how to use the Host Servers provided with OpenComRTOS. The Host Servers are a component, which allow OpenComRTOS applications to use services provided by the host operating system. For instance the Stdio Host Server provides access to the console of a MS-Windows or GNU Linux system, as well as access to the file system.

Due to the use of the Host Servers these services become available also embedded targets which very often do not even provide a form of user interface. Thus the Host Servers make it for instance possible for an embedded ARM node to access the screen, keyboard, and file system of the Win32 executing the Stdio Host Server.

## 5.2. General Principles of OpenComRTOS Host Services

- A Host Service consists of a Host Server and a Host Client component, for application developer only the Host Server component has to be added to the Application diagram. The build system ensures that nodes accessing this Host Server include the Host Client component.
- A Host Server component consists of a task and multiple hubs.
- Stdio Host Server and Graphical Host Server components have their own interactions, which are fully integrated in OpenVE.
- The first parameter of Host Service function is always the Host Server Port of type L1_HubID, this identifies the Host Server.
- All interactions between a Host Service Client and a Host Service Server use waiting semantics. This means once a task has committed to interact with the Host Service Server this task will wait until this interaction has completed.

## 5.3. Stdio Host Server

The Stdio Host Server provides standard input and output functionality to OpenComRTOS tasks. This includes file operations.

## 5.4. Graph Host Server

Graph Host Server supports drawing on Host Server the basic graphics primitives like line, circle, and string with possibility of color and style settings by an OpenComRTOS application task.

# 6. The list of Figures

# Appendix A

C file and Node source files text

```
/* Created <Date> <Time> <Year> */
#include <L1_api.h>

#include "L1_node_config.h"
```

# Appendix B

Node **c** source files text

```
/* Created <Date> <Time> <Year> */
#include <L1_api.h>
#include <L1_node_config.h>
int main(void)
{
return L1_runOpenComRTOS(L1_NODE_NUMBER_OF_TASKS,L1_NODE_NUMBER_OF_HUBS);
}
```

# Appendix C

Task Entry Point template

```
#include <L1_api.h>
#include "L1_node_config.h"
void Task1EntryPoint(L1_TaskArguments Arguments)
{
while(1){

  }
}
```