

WIND RIVER

Wind River® VxWorks Simulator

USER'S GUIDE

6.6

Copyright 2007 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. The Wind River logo is a trademark of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:

<http://www.windriver.com/company/terms/trademark.html>

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation at the following location:

installDir\product_name\3rd_party_licensor_notice.pdf.

Wind River may refer to third-party documentation by listing publications or providing links to third-party Web sites for informational purposes. Wind River accepts no responsibility for the information provided in such third-party documentation.

Corporate Headquarters

Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

<http://www.windriver.com>

For information on how to contact Customer Support, please visit the following URL:

<http://www.windriver.com/support>

Contents

1	Overview	1
1.1	Introduction	1
1.2	Supported Features and Compatibility	2
1.2.1	VxWorks Feature Support	2
1.2.2	Simulated Hardware Support	3
1.3	Limitations	3
2	Getting Started	5
2.1	Introduction	5
2.2	System Requirements	6
2.3	Configuring and Building a VxWorks Image	6
2.3.1	Default Configuration Components	6
2.3.2	Configuring Optional Components	7
2.3.3	Building Your VxWorks Image	8
2.4	Launching the VxWorks Simulator	8
2.4.1	vxsim Configuration Options	9
2.4.2	Launching a VxWorks Simulator Instance from the Command Line	12

	Modifying the Windows Simulator Console Appearance	13
	Starting Instances to Run on a Simulated Subnet	13
2.4.3	Launching the VxWorks Simulator From Workbench	14
2.4.4	Rebooting and Exiting the VxWorks Simulator	14
2.4.5	Accessing the VxWorks Simulator from a Remote Host	15
3	Introduction to the VxWorks Simulator Environment	17
3.1	Introduction	17
3.2	Understanding the VxWorks Simulator BSP	18
3.3	Building Applications	19
3.3.1	Defining Compiler Options	19
3.3.2	Compiling Modules for Debugging	21
3.4	Interface Variations	21
3.4.1	Memory Management Unit	22
	Simulation	22
	Translation Model	22
	Page Size	22
	Limitations	23
	Running the VxWorks Simulator Without MMU Support	23
3.4.2	RTP Considerations	23
3.4.3	File System Support	23
	Pass-Through File System (passFS)	23
	Virtual Disk Support	24
3.4.4	WDB Back End	24
3.4.5	Connection Timeout	24
3.5	Architecture Considerations	24
3.5.1	Byte Order	25
3.5.2	Hardware Breakpoint	25
3.5.3	Floating-Point Support	25

3.5.4	ISR Stack Protection	26
3.5.5	Interrupts	26
	Solaris and Linux Systems	26
	Windows Systems	28
3.5.6	Memory Layout	29
4	Using the VxWorks Simulator	33
4.1	Introduction	33
4.2	Configuring the VxWorks Simulator	34
4.2.1	Boot Parameters	34
4.2.2	Memory Configuration	34
	Physical Memory Address Space	35
	Virtual Memory Address Space	35
	Memory Protection Level	36
4.2.3	Miscellaneous Configuration	36
4.3	Configuring Hardware Simulation	36
4.3.1	Pass-Through File System (passFS)	36
4.3.2	Virtual Disk Support	37
4.3.3	Non-Volatile RAM Support	38
4.3.4	Standard I/O	39
4.3.5	Timers	39
4.3.6	Timestamp Driver	39
4.3.7	Serial Line Support	40
4.3.8	Shared Memory Network	40
4.4	Using VxWorks SMP with the VxWorks Simulator	43
4.4.1	Creating an SMP Image	43
4.5	Migrating Applications to a Hardware-Based System	45

5	Networking with the VxWorks Simulator	47
5.1	Introduction	47
5.2	Building Network Simulations	48
5.3	Setting Up the Network Daemon	50
5.3.1	Starting the Network Daemon	51
	Starting the Network Daemon as a Service	51
	Starting the Network Daemon From the Command Line	56
5.3.2	Removing the Network Daemon Service	57
5.3.3	Network Daemon Debug Shell	57
5.3.4	Creating a Network Daemon Configuration File	60
	Configuring Multiple External Subnets	65
5.4	Installing the Host Connection Driver	66
5.4.1	Managing the WRTAP Driver on Windows Hosts	68
5.5	Configuring a Simulated Subnet	70
5.5.1	Starting a Simulator Instance With Multiple Network Interfaces	70
5.5.2	Starting a Simulator Instance Without an IPv4 Address	71
6	Networking Tutorials	73
6.1	Introduction	73
6.2	Simple Simulated Network	74
6.2.1	Set Up the Network Daemon	74
6.2.2	Start a VxWorks Simulator Instance	76
6.2.3	Test the Simulated Network	78
6.3	Basic Simulated Network with Multiple Simulators	78
6.3.1	Creating a Static Configuration	79
6.3.2	Creating a Dynamic Configuration Using the vxsimnetd Shell	87

6.4	Running the VxWorks Simulator on the Local Network	90
6.4.1	Default subnet configuration	90
6.4.2	Configuring a Bridge	92
	Windows Setup	92
	Linux bridge setup	93
6.5	IPv6 Tutorial	95
6.5.1	Configure the Network	96
6.5.2	Configuring VxWorks with IPv6 Components	98
	Build Your Projects	99
6.5.3	Testing the IPv6 Connection	99
	Start the VxWorks Simulator Instances	99
A	Accessing Host Resources	107
A.1	Introduction	107
A.2	Accessing Host OS Routines	108
A.3	Loading a Host-Based Application	108
A.4	Host Application Interface (vxsimapi)	109
A.4.1	Defining User Exit Hooks	109
A.4.2	Configuring a Host Device to Generate interrupts (UNIX Only)	109
A.4.3	Simulating interrupts From a User Application (Windows Only) ...	110
A.5	Tutorials and Examples	111
A.5.1	Running Tcl on the VxWorks Simulator	111
	Code Sample	111
	Running The Code	113
A.5.2	Controlling a Host Serial Device	113
	Index	115

1

Overview

1.1	Introduction	1
1.2	Supported Features and Compatibility	2
1.3	Limitations	3

1.1 Introduction

The Wind River VxWorks Simulator is a simulated hardware target for use as a prototyping and test-bed environment for VxWorks. The VxWorks simulator allows you to develop, run, and test VxWorks applications on your host system, reducing the need for target hardware during development. The VxWorks simulator also allows you to set up a simulated target network for developing and testing complex networking applications.

For external applications needing to interact with a VxWorks target, the capabilities of a VxWorks simulator instance are identical to those of a VxWorks system running on target hardware. A VxWorks simulator instance supports a standard set of VxWorks features, such as network applications and target and host VxWorks shells. Building these features into a VxWorks simulator image is no different than building them into any VxWorks cross-development environment using a standard board support package (BSP).

The goal of this document is to quickly familiarize you with the VxWorks simulator. The early chapters discuss basic configuration information, instructions for building a VxWorks image based on the VxWorks simulator BSP, instructions for launching the VxWorks simulator from Wind River Workbench or the command line, and information on building applications. Later chapters provide more detailed usage information as well as instructions and tutorials for setting up a network of VxWorks simulator instances.

This document provides instructions for all supported VxWorks simulator host types including Linux, Solaris, and Windows-based simulators.



NOTE: Many examples in this document are provided generically for all hosts. These examples are typically provided in UNIX syntax. Be sure to modify the generic examples to suit your host platform. For example, on Windows hosts, be sure to change forward slashes to back slashes in file paths.

1.2 Supported Features and Compatibility

The VxWorks simulator supports most VxWorks features available on target hardware and also provides support for a number of simulated hardware devices. In addition, applications developed for the simulator are fully compatible with VxWorks.

1.2.1 VxWorks Feature Support

Applications developed using the VxWorks simulator can take advantage of the following VxWorks features:

- Real-Time Processes (RTPs)
- Error Detection and Reporting
- ISR Stack Protection (Solaris and Linux hosts only)
- Shared Data Regions
- Shared Libraries (Windows and Linux hosts only)
- ROMFS

- VxMP (shared-memory objects)
- VxFusion (distributed message queues)
- Wind River System Viewer

For more information on these features, see the *VxWorks Kernel Programmer's Guide*.

1.2.2 Simulated Hardware Support

To support application development, the VxWorks simulator provides simulated hardware support for the following hardware-related features:

- a VxWorks console
- a system timer
- a memory management unit (MMU)—MMU support is required to take advantage of the VxWorks real-time process (RTP) feature.
- non-volatile RAM (NVRAM)
- virtual disk support—Virtual disk support allows you to simulate a disk block device. The simulated disk block device can then be used with any file system supported by VxWorks.
- a timestamp driver
- a real-time clock
- symmetric multiprocessing (SMP) environment

For information on including support for these simulated hardware features in your VxWorks image, see [2.3 Configuring and Building a VxWorks Image](#), p.6. For more information on hardware simulation, see [4.3 Configuring Hardware Simulation](#), p.36.

1.3 Limitations

The VxWorks simulator is not suitable for all prototyping needs. The VxWorks simulator executes on the host machine and does not attempt the simulation of machine-level instructions for a target architecture. For this reason, the VxWorks

simulator is not a suitable basis on which to develop hardware device drivers. However, the VxWorks simulator includes MMU support and implements the architecture-specific part of a memory management unit in order to provide the same level of feature support as hardware-based targets.

SMP simulation using the VxWorks simulator will be more accurate if the host machine provides multiple CPU cores.

2

Getting Started

- 2.1 Introduction 5
- 2.2 System Requirements 6
- 2.3 Configuring and Building a VxWorks Image 6
- 2.4 Launching the VxWorks Simulator 8

2.1 Introduction

This chapter briefly describes how to set up and configure standard features into a VxWorks image for use with the VxWorks simulator. It also includes instructions for launching the VxWorks simulator from your development environment. For more information on configuring and building VxWorks, see the *VxWorks Kernel Programmer's Guide* and the *Wind River Workbench User's Guide* or *VxWorks Command-Line Tools User's Guide*.

2.2 System Requirements

Host system requirements for the VxWorks simulator are the same as that of a standard VxWorks installation, no additional resources are required. For information on VxWorks host system requirements, see your Platform release notes.

SMP simulations will be more accurate if the host machine provides the same number of CPU cores as the hardware target.

2.3 Configuring and Building a VxWorks Image

The default configuration file included in the VxWorks simulator BSP produces a full-featured VxWorks image. Many standard VxWorks features are included in the image by default. In addition to a list of default configuration components, this section provides configuration information for supported optional components as well as basic instructions for building a VxWorks image.

2.3.1 Default Configuration Components

The VxWorks simulator default configuration includes support for many VxWorks features including:

- the kernel shell (and its C and command interpreters)
- the Wind River System Viewer
- kernel hardening features (such as text segment write protection)
- error detection and reporting features
- the ROM-based file system (ROMFS)
- shared libraries and shared data regions
- POSIX support
- basic memory management support (`INCLUDE_MMU_BASIC`) (See [3.4.1 Memory Management Unit](#), p.22)
- real time process (RTP) support
- the network stack
- virtual disk support (See [4.3.2 Virtual Disk Support](#), p.37)
- non-volatile RAM (See [4.3.3 Non-Volatile RAM Support](#), p.38)

For more information on these features, see the *VxWorks Kernel Programmer's Guide* and the *VxWorks Application Programmer's Guide*. For information on using these features with the VxWorks simulator, see [4. Using the VxWorks Simulator](#).



NOTE: Networking for the VxWorks simulator is configured by default. However, the network is not automatically enabled because no network device is initialized. To initialize a network device, specify the **simnet** device using the **-d** option when launching the VxWorks simulator or specify a network interface using the **-ni** option. For more information on these options, see [2.4.1 vxsim Configuration Options](#), p.9.

2.3.2 Configuring Optional Components

The VxWorks simulator provides optional support for the following VxWorks features. To take advantage of these features, you must configure and build a new VxWorks image for the VxWorks simulator. For more information on building and configuring a VxWorks image, see [2.3.3 Building Your VxWorks Image](#), p.8.

VxMP

The VxWorks simulator optionally supports the multiprocessor capabilities available using the VxWorks VxMP feature. To include support for this feature, you must include the **INCLUDE_SM_COMMON**, **INCLUDE_BOOT_LINE_INIT**, and **INCLUDE_SM_OBJ** components in your VxWorks image.

To tune the shared memory size allocated for VxMP (default: 8 KB), use the **SM_MEM_SIZE** parameter. To modify the shared memory pool size assigned to shared objects, change the **SM_OBJ_MEM_SIZE** parameter.

Shared Memory END Driver

The VxWorks simulator optionally includes shared memory END driver support (**smEnd**). To include **smEnd** driver support, the macro **INCLUDE_SM_NET** must be defined into the BSP configuration. To define the **smEnd** driver IP address, use the following VxWorks simulator command line option

```
% vxsim -b backplaneAddress
```

or:

```
% vxsim -backplane backplaneAddress
```

2.3.3 Building Your VxWorks Image

The process for configuring and building a VxWorks image is the same for the VxWorks simulator as it is for target hardware. The VxWorks simulator BSP is comparable to a standard VxWorks BSP for a hardware target architecture and uses a standard VxWorks **Makefile**. However, the BSP makefile builds only the images **vxWorks** and **vxWorks.st** (standalone VxWorks). Your VxWorks image can be built using either the Wind River Compiler or the Wind River GNU Compiler.

To build a default VxWorks image, you can create a VxWorks image project using Wind River Workbench **New VxWorks Image Project** wizard. To open the wizard, select **File > New > VxWorks Image Project**. Base your project on the appropriate VxWorks simulator BSP for your host type. [Table 2-1](#) lists the available simulator BSPs.

Table 2-1 **VxWorks Simulator BSPs**

Host Type	BSP Name
Windows	simpc
Solaris	solaris
Linux	linux

For more information on building VxWorks image projects, see the *Wind River Workbench User's Guide: Creating VxWorks Image Projects*.

You can also build a VxWorks image project from the command line using the command line project facility, **vxprj**. For more information on building a VxWorks image project using **vxprj**, see the *VxWorks Command-Line Tools User's Guide: Building Kernel and Application Projects*.

2.4 Launching the VxWorks Simulator

This section provides information on launching the VxWorks simulator from your development environment.



NOTE: On Solaris simulators, your path environment variable must include `/usr/openwin/bin` so that your host can locate `xterm`. If `xterm` is not in your path, your VxWorks simulator connection will fail.

2.4.1 vxsim Configuration Options

The `vxsim` executable provides the equivalent functionality of a boot load program. However, you cannot build or customize `vxsim` as you can other boot loaders.

You can use `vxsim` to load an image from the VxWorks simulator BSP directory. The `vxsim` utility supports a set of command-line configuration options that you can use to specify the boot line parameters for the image that will be loaded. The command-line interface also supports additional convenience options that let you handle things such as configuring multiple interfaces for the VxWorks simulator instance. Use `vxsim -help` to see a complete list of supported options.



NOTE: To preserve boot line parameters after a reboot, you must use the `bootChange()` routine. This routine is executed in the kernel shell. For more information, see [4.2.1 Boot Parameters](#), p.34.

Table 2-2 Command-Line Options for the VxWorks Simulator

VxWorks Simulator Option	Description
<code>-backplane</code> <i>inetOnBackplane</i> <code>-b</code> <i>inetOnBackplane</i>	Backplane address of the target system.
<code>-device</code> <i>bootDevice</i> <code>-d</code> <i>bootDevice</i>	Type of device to boot from, select <code>passDev</code> or <code>simnet</code> . Default: <code>passDev</code> ^a
<code>-ethernet</code> <i>inetOnEthernet</i> <code>-e</code> <i>inetOnEthernet</i>	Internet address to assign to the target system (the boot interface).
<code>-exitOnError</code>	Upon error, exit the VxWorks simulator without prompting for user input.
<code>-file</code> <i>fileName</i> <code>-f</code> <i>fileName</i>	File containing the VxWorks image to load. If no file is specified, the <code>vxWorks</code> file, if any, in the current directory is loaded.
<code>-flags</code> <i>flags</i>	Configuration flags. Default: 0.

Table 2-2 **Command-Line Options for the VxWorks Simulator** (cont'd)

VxWorks Simulator Option	Description
-gateway <i>gatewayInet</i> -g <i>gatewayInet</i>	Internet address of the gateway.
-help	Print a help message listing the command-line options.
-hostinet <i>hostInet</i> -h <i>hostInet</i>	Host Internet address.
-hostname <i>hostName</i> -hn <i>hostName</i>	Host machine to boot from. The default value for this option on a Windows system is host . On a UNIX system, the default value is always the actual host name.
-kill <i>processNumber</i> -k <i>processNumber</i>	Kills the VxWorks simulator referenced by <i>processNumber</i> .
-logfile <i>log file</i> -l <i>log file</i>	Enables output logging (a Windows-only option). Default: VxSIMn.log .
-ncpu <i>cpuNumber</i>	Limits the number of CPUs to <i>cpuNumber</i> . Allows a VxWorks simulator image configured with <i>n</i> CPUs to run with 1 to <i>n</i> CPUs.
-n -nvram	VxWorks simulator non-volatile RAM file.
-netif <i>additionalInterface</i> -ni <i>additionalInterface</i>	VxWorks simulator additional network interfaces.
-other <i>other</i> -o <i>other</i>	Unused, available for applications.
-password <i>ftpPassword</i> -pw <i>ftpPassword</i>	User password (for FTP only).
-processor <i>number</i> -p <i>number</i>	Sets the processor number, which is effectively an identifier for this simulator instance. Default value: 0.
-prot-level <i>protectionLevel</i> -pl <i>protectionLevel</i>	Sets the VxWorks simulator memory protection level. Values include min (0), max (1), or integer. Default value is max (1).

Table 2-2 Command-Line Options for the VxWorks Simulator (cont'd)

VxWorks Simulator Option	Description
-size <i>memorySize</i> -memsize <i>memorySize</i>	VxWorks simulator memory size in bytes. Default: 32 MB.
-startup <i>script</i> -s <i>script</i>	Startup script for the kernel shell.
-targetname <i>targetName</i> -tn <i>targetName</i>	Name of the target. Default: VxSim , for the VxSim0 instance, VxSimCpuNum for all other instances.
-tmpdir <i>directory</i>	Directory for temporary VxWorks simulator files. Default: TEMP environment variable value, on Windows system; /tmp on a Solaris or Linux system.
-unit <i>unitNumber</i>	Network device unit number (for the boot interface). Default: 0.
-user <i>user</i> -username <i>user</i>	User name used to access the host.
-vaddr <i>address</i>	Specifies the VxWorks virtual base address.
-vsize <i>virtualSize</i>	Specifies the VxWorks virtual memory size. Default: 1 GB.
-version -v	Print the VxWorks simulator version.
a. For an actual hardware target, this option is used to specify the device that is used to access the VxWorks image. Because the VxWorks simulator always accesses the VxWorks image from the host file system, this field is used to mimic real target behavior. Specifying simnet for this option indicates that a simulated boot device should be initialized. In all other cases, passDev is used as the boot device type. For example, the tutorial in 6.3 Basic Simulated Network with Multiple Simulators , p.78 uses the passDev option for the simulated router and the simnet option for each of the simulated network devices.	



NOTE: When issuing these command options using the **vxsim** command from the command line on a Linux or Solaris host, use double quotes around the option values. For example, use **vxsim -ni "simnet"** in place of **vxsim -ni simnet**. The double quotes should *not* be used when passing options in Workbench or from the command line on a Windows host as the command will fail.

When launching your VxWorks simulator from Workbench (**Target > New Connection > Wind River VxWorks 6.x Simulator Connection**), the command-line options listed in [Table 2-2](#) are configured using the **New Connection** wizard. Certain options (for example, the **-ni** option) are not available as specific options in the **New Connection** wizard dialogs. These options can be passed directly to the VxWorks simulator using the **Other VxWorks Simulator Options** field of the **VxWorks Simulator Miscellaneous Options** dialog.

2.4.2 Launching a VxWorks Simulator Instance from the Command Line

To start a VxWorks simulator instance from the command line, use the **vxsim** command in the VxWorks development shell. Using this command is similar to using a boot program to load an image. As options, **vxsim** lets you specify values for the parameters typically supplied in a boot line (use **vxsim -help** to list the option descriptions).



NOTE: You must use the **bootChange()** routine in order to make boot-parameter changes that survive a reboot. Even if non-volatile RAM support is included, boot parameters will be lost once the simulator is exited because boot parameters are derived from the VxWorks simulator command line.

To start a VxWorks simulator using the default configuration values, type the following in the VxWorks development shell:

```
% vxsim -f pathToVxWorksImage
```

If you run **vxsim** from the directory containing the VxWorks image that you want to load, you can simply type:

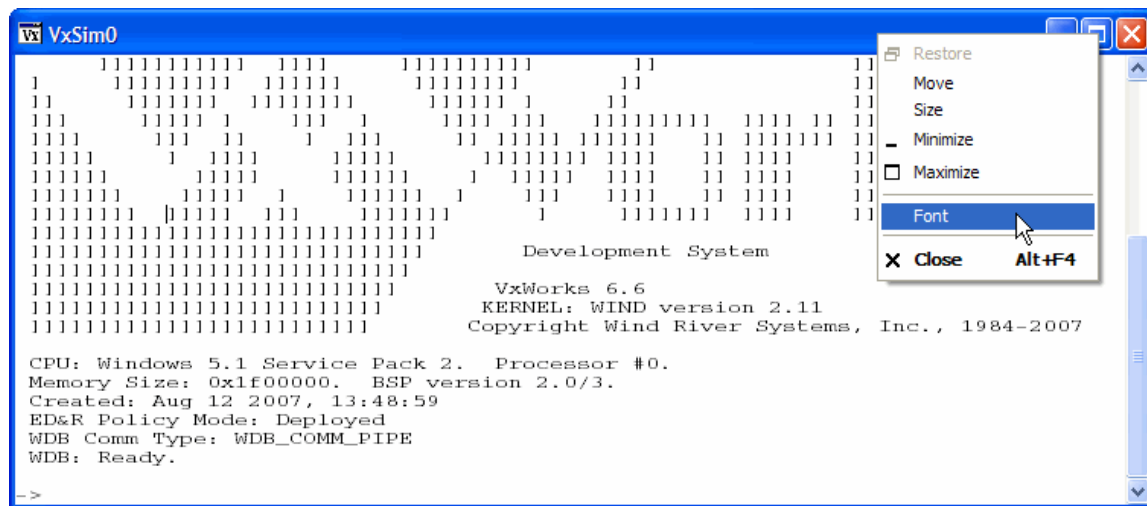
```
% vxsim
```

If your path does not include **vxsim**, ensure that your VxWorks environment is properly set. For more information, see the *VxWorks Command-Line Tools User's Guide: Creating a Development Shell with wrenv*.

Modifying the Windows Simulator Console Appearance

When you launch a VxWorks simulator instance on a Windows host, the target server launches a console connected to the target. To change the font used in the console, right-click the top bar and select **Font** in the resulting menu. For example, [Figure 2-1](#) shows the simulator console in Windows with the right-click menu open.

Figure 2-1 **Simulator Console**



Starting Instances to Run on a Simulated Subnet

If you intend to use network services, do not start a VxWorks simulator instance until you start the VxWorks simulator network daemon is (for more information, see [5.3 Setting Up the Network Daemon](#), p.50). For the most part, you need only concern yourself with those devices related to the network interface, specifically, the **simnet** device, and its address information (IP address and network mask). If a simulator requires only a single interface, you can specify **simnet** as the boot device and its IP address using the **-e** parameter. For example:

```
% vxsim -d simnet -e 192.168.200.1
% vxsim -d simnet -e 192.168.200.2 -p 1
```

Assuming the following default subnet:

```
SUBNET_START default {
    SUBNET_EXTERNAL = yes;
```

```
SUBNET_EXTPROMISC = yes;  
SUBNET_ADDRESS = "192.168.200.0";  
};
```

The two **vxsim** commands shown above start two VxWorks simulator instances that attach to the default subnet 192.168.200.0. In the second **vxsim** command, the **-p** parameter assigns the value **1** as a “processor number,” to the second VxWorks simulator instance. If you do not specify a **-p** number, the default value of **0** is applied.

A VxWorks simulator instance can also be configured with multiple network interfaces (a router configuration) using the **-ni** option. For more information on this configuration, see [5.5.1 Starting a Simulator Instance With Multiple Network Interfaces](#), p.70.

For more information on setting up a simulated network, see [5. Networking with the VxWorks Simulator](#).

2.4.3 Launching the VxWorks Simulator From Workbench

The VxWorks simulator can be launched from Workbench. To do this, you must create a VxWorks simulator target connection using the Target Manager. This is accomplished by launching the Workbench **New Connection** wizard. Select **Target > New Connection...** and select **Wind River VxWorks 6.x Simulator Connection** as your connection type. You can then use the wizard to configure your VxWorks simulator.

For more information on creating a VxWorks simulator connection, refer to the *Wind River Workbench User's Guide: New VxWorks Simulator Connections*. For information on establishing a target connection, refer to the *Wind River Workbench User's Guide: Connecting to Targets*. More detailed instructions are also provided as part of the tutorials in [6. Networking Tutorials](#).

2.4.4 Rebooting and Exiting the VxWorks Simulator

As with other targets, you can reboot the VxWorks simulator by typing **Ctrl+X** in the VxWorks simulator console window. On Windows systems, you can exit the VxWorks simulator by closing the VxWorks simulator window. On Solaris and Linux systems, you must type **Ctrl+** in the VxWorks simulator console window to exit.



NOTE: Issuing the exit command in the kernel shell terminates the shell session only and cannot be used to exit the VxWorks simulator. To programmatically exit the VxWorks simulator (which may be useful for scripting), you can reboot (**BOOT_NO_AUTOBOOT**). If you wish to use this option, you must start the VxWorks simulator with the **-exitOnError** option.

2.4.5 Accessing the VxWorks Simulator from a Remote Host

You can access a VxWorks simulator target from a remote host (a host other than the one running the VxWorks simulator instance) using the following process:

1. Enable IP forwarding on the host that will be running the VxWorks simulator instance.
 - On Windows hosts, start the **Routing and Remote Access** service.
 - On Solaris hosts, run the following with root permissions:


```
# # ndd -set /dev/tcp ip_forwarding 1
```
 - On Linux hosts, run the following with root permissions:


```
$ # echo "1" > /proc/sys/net/ipv4/ip_forward
```

 or, edit **/etc/sysconfig/network** and add the following line:


```
FORWARD_IPV4=yes
```

 Then, restart the host.
2. Start a VxWorks simulator instance on an external subnet.
3. Specify the route to access the remote host on the VxWorks simulator instance. The gateway is the address of the host running the VxWorks simulator instance on the simulated subnet (*subnetAddress.254*).

Note that this can be done using VxWorks simulator boot parameters as follows (this example assumes the IP address of the host running the VxWorks simulator instance is 90.0.0.1):

```
% vxsim -d simnet -e 192.168.200.1 -h 90.0.0.1 -g 192.168.200.254
```

This sets a route that enables the VxWorks simulator instance to send a packet to any host reachable from 90.0.0.1.

4. On the remote host, specify the route to access the VxWorks simulator instance. The gateway is the address of the host running the VxWorks simulator instance on the remote host subnet.

You can now connect to the VxWorks simulator instance from the remote host.

3

Introduction to the VxWorks Simulator Environment

- 3.1 Introduction 17
- 3.2 Understanding the VxWorks Simulator BSP 18
- 3.3 Building Applications 19
- 3.4 Interface Variations 21
- 3.5 Architecture Considerations 24

3.1 Introduction

This chapter discusses the differences between the VxWorks simulator development environment and the standard VxWorks development environment. In general, the VxWorks simulator environment differs very little from the development environment for any other target hardware system. The most notable differences are the addition of the VxWorks simulator network daemon, which is discussed in [5.3 Setting Up the Network Daemon](#), p.50, and variations in the VxWorks simulator BSP and architecture behavior, which are discussed in the following sections.

3.2 Understanding the VxWorks Simulator BSP

Aside from the exceptions noted in this section, the VxWorks simulator BSP is similar to any other VxWorks BSP for a target hardware board and provides similar functionality.

sysLib.c

The **sysLib.c** module contains the same essential routines: **sysModel()**, **sysHwInit()**, and **sysClkConnect()** through **sysNvRamSet()**. However, because there is no bus, **sysBusToLocalAdrs()** and related routines have no effect in the VxWorks simulator environment.

The BSP file **sysLib.c** can also be extended to emulate the eventual target hardware more completely. For more information on **sysLib.c**, see the *VxWorks BSP Developer's Guide*.

Standard I/O

The file **winSio.c** (or **unixSio.c** for Linux and Solaris systems) ultimately calls the host OS **read()** and **write()** routines on the standard input and output for the process. Nevertheless, it supports all of the functionality provided by **tyLib.c**.

config.h

The configuration header file, **config.h**, is minimal:

- It does not reference a *bspname.h* file.
- The boot line has no fixed memory location. Instead, it is stored in the variable **sysBootLine** in **sysLib.c**.

Makefile

The VxWorks simulator **Makefile** is similar to the standard version used for VxWorks target hardware BSPs. However, it does not build boot ROM images (although the makefile rules remain intact); it can only build **vxWorks** and **vxWorks.st** (standalone) images. The final linking does not arrange for the **TEXT** segment to be loaded at a fixed area in RAM, but follows the usual loading model. The makefile macro **MACH_EXTRA** is provided so that users can easily link their application modules into the VxWorks image if they are using manual build methods.

3.3 Building Applications

Wind River highly recommends that you build VxWorks simulator modules using the **vxprj** command-line facility or Wind River Workbench. However, if you wish to customize your build, you may need the information in the following sections.

3.3.1 Defining Compiler Options

The Workbench build mechanism for the VxWorks simulator uses two preprocessor constants, **CPU** and **TOOL**, to define compiler options for a specific build target. The **CPU** variable ensures that VxWorks and your applications are compiled with the appropriate features enabled. The **TOOL** variable defines the toolchain to use for compiling and linking modules.

VxWorks simulator modules can be built with either the Wind River Compiler or the GNU compiler. To build modules using the Wind River Compiler, define **TOOL** as **diab**. To build modules using the GNU compiler, define **TOOL** as **gnu**.



NOTE: Modules built with either **gnu** or **diab** can be linked together in any combination, except for modules that require C++ support. Cross-linking of C++ modules is not supported in this release.

Applications for the VxWorks simulator can be built to run in either the VxWorks kernel or a VxWorks RTP.

[Table 3-1](#) lists the available **CPU** and **TOOL** definitions for the VxWorks simulator. It also provides sample command line options specific to the VxWorks simulator architecture. For more information on the available compiler options, see the compiler documentation for Pentium (Windows and Linux hosts) or SPARC (Solaris hosts).

Table 3-1 VxWorks Simulator Compiler Options

CPU Definition	Host (Run Type)	TOOL	Compiler Command-Line Options
SIMNT	Windows (kernel)	diab	-tX86LH:vxworks65 -DCPU=SIMNT
		gnu	-mcpu=i486 -march=i486 -DCPU=SIMNT

Table 3-1 VxWorks Simulator Compiler Options (cont'd)

CPU Definition	Host (Run Type)	TOOL	Compiler Command-Line Options
SIMLINUX	Linux (kernel)	diab	-tX86LH:vxworks65 -DCPU=SIMLINUX
		gnu	-mcpu=i486 -march=i486 -DCPU=SIMLINUX
SIMSPARCSOLARIS	Solaris (kernel)	diab	-tSPARCFH:vxworks65 -DCPU=SIMSPARCSOLARIS
		gnu	-DCPU=SIMSPARCSOLARIS
	Solaris (RTP)	diab	-tSPARCFH:rtpsim -DCPU=SIMSPARCSOLARIS
		gnu	-mrtp -DCPU=SIMSPARCSOLARIS
SIMPENTIUM	Windows and Linux (RTP)	diab	-tX86LH:rtpsim -DCPU=SIMPENTIUM
		gnu	-mcpu=i486 -march=i486 -DCPU=SIMPENTIUM

For example, to specify the CPU value for an application that will run in an RTP on a Windows (or Linux) host, use the following command line option when you invoke the compiler:

-DCPU=SIMPENTIUM

On all hosts, the VxWorks simulator uses ELF object module format (OMF). Your VxWorks installation also includes a VxWorks simulator binary for each supported host system. You can use this binary as a boot loader for a VxWorks image which you can configure and build using exactly the same compiler options you use to build a VxWorks image for a hardware target architecture. Thus, if you are compiling a VxWorks image for a Linux or Windows VxWorks simulator, you should use the same compiler as the Intel Architecture. If you are compiling for the Solaris VxWorks simulator, you must compile for the SPARC architecture.

If you attempt to load an ELF image for another architecture onto a VxWorks simulator, you will get a load error. For example, if you attempt to load a PPC32 ELF image from the kernel shell:

```
-> ld < host:C:/WindRiver/workspace/PPC_ELF.out
ld(): error loading file (errno = 0x610001).
value = 0 = 0x0
```

For information on available compiler options, see the *Wind River Compiler for x86 User's Guide* or the *Wind River Compiler for SPARC User's Guide*.

3.3.2 Compiling Modules for Debugging

To compile C and C++ modules for debugging, you must use the **-g** flag to generate debug information. An example command line for the Wind River Compiler is as follows:

```
% dcc -tX86LH:vxworks65 -DCPU=SIMNT -IinstallDir/target/h -g test.cpp
```

In this example, *installDir* is the location of your VxWorks tree and **-DCPU** specifies the CPU type.

An equivalent example for the Wind River GNU Compiler is as follows:

```
% ccpentium -mcpu=i486 -march=i486 -DCPU=SIMNT -IinstallDir/target/h -g test.cpp
```



NOTE: Debugging code compiled with optimization is likely to produce unexpected behavior, such as breakpoints that are never hit or an inability to set breakpoints at some locations. This is because the compiler may re-order instructions, expand loops, replace routines with in-line code, and perform other code modifications during optimization, making it difficult to correlate a given source line to a particular point in the object code. Users are advised to be aware of these possibilities when attempting to debug optimized code. Alternatively, users may choose to debug applications without using compiler optimization. To compile without optimization using the Wind River Compiler, compile without the **-XO** option or use the **-Xno-optimized-debug** option. To compile without optimization using the GNU compiler, compile without a **-O** option or use the **-O0** option.

3.4 Interface Variations

This section describes particular functions and tools that are specific to VxWorks simulator targets in any of the following ways:

- available only for VxWorks simulator targets
- parameters specific to VxWorks simulator targets
- special restrictions on, or characteristics of, VxWorks simulator targets

For complete documentation, see the reference entries for the libraries, routines, and tools discussed in the following sections.

3.4.1 Memory Management Unit

This section describes the memory management unit implementation for the VxWorks simulator and how it varies from the standard VxWorks MMU implementation.

Simulation

The VxWorks simulator provides a simulated hardware memory management unit. The simulated MMU provides features comparable to those found on typical hardware MMUs. The simulation uses features provided by the host operating system to map, unmap, and protect pages in memory. MMU simulation is provided for all supported host operating systems.

Translation Model

All VxWorks simulator implementations share a common programming model for mapping memory pages. The data structure **sysPhysMemDesc**], defined in **sysLib.c** describes the physical memory address space. This data structure is made up of configuration constants for each page or group of pages.

Use of the **VM_STATE_CACHEABLE** constant for each page or group of pages, sets the cache to copy-back mode.

In addition to **VM_STATE_CACHEABLE**, the following additional constants are supported:

- **VM_STATE_CACHEABLE_NOT**
- **VM_STATE_WRITEABLE**
- **VM_STATE_WRITEABLE_NOT**
- **VM_STATE_VALID**
- **VM_STATE_VALID_NOT**

For more information on these configuration constants, see the *VxWorks Kernel Programmer's Guide*.

Page Size

The VxWorks simulator uses a page size that is determined by the memory page mapping routines in the host operating system. On Solaris and Linux-based simulators, this page size is 8 KB. On Windows simulators, this page size is 64 KB.

Limitations

The VxWorks simulator MMU implementation does not provide support for supervisor/user mode.

Running the VxWorks Simulator Without MMU Support

You can configure the VxWorks simulator to run without an MMU. For more information on how to configure your VxWorks image for this type of operation, see the *VxWorks Kernel Programmer's Guide: Memory Management*.

To run the VxWorks simulator without an MMU, Wind River recommends that you change the MMU page size (`VM_PAGE_SIZE` parameter) in your VxWorks image to 0x1000 (the default value is 0x2000 on Solaris and Linux simulators and 0x10000 on Windows simulators) in order to limit the amount of physical memory required to run your applications.

3.4.2 RTP Considerations

Because the VxWorks simulator MMU implementation does not support supervisor/user mode, it is not possible to prevent a task running in an RTP from writing in the kernel memory space. Therefore, on the VxWorks simulator architecture, an RTP task can potentially crash a kernel task.

3.4.3 File System Support

This section discusses the file systems supported by the VxWorks simulator. For more information on file systems, see the *VxWorks Kernel Programmer's Guide*.

Pass-Through File System (passFS)

By default, the VxWorks simulator uses a pass-through file system (passFS) to access files directly on the host system. For more information on using passFS with the VxWorks simulator, refer to [4.3.1 Pass-Through File System \(passFS\)](#), p.36.

Virtual Disk Support

The VxWorks simulator provides virtual disk support which allows you to simulate a disk block device. The simulated disk block device can be used to access any file system supported by VxWorks. For more information on virtual disk support for the VxWorks simulator, refer to [4.3.2 Virtual Disk Support](#), p.37.

3.4.4 WDB Back End

The VxWorks simulator supports the WDB pipe and WDB RPC target agent communication back ends; the WDB pipe back end is used by default. If network support is enabled on your VxWorks simulator target, the WDB RPC back end can also be used.

3.4.5 Connection Timeout

Occasionally, VxWorks simulator sessions lose their target server connections when the host CPU becomes overwhelmed by too many requests. If you find that your application is frequently losing its target server connection, you can adjust the back end request timeout (**-Bt**) and back end request resend number (**-Br**) parameters from Workbench using the **Advanced Target Server Options** in your **VxWorks Simulator Connection**. For more information on resolving connection timeouts, refer to the *Wind River Workbench User's Guide: New Target Server Connections*.

3.5 Architecture Considerations

This section describes characteristics of the VxWorks simulator architecture that you should be aware of as you write a VxWorks application. The following topics are addressed:

- byte order
- hardware breakpoint
- floating-point support
- interrupts
- memory layout

3.5.1 Byte Order

The Solaris simulator uses a big-endian environment. The Windows and Linux simulators use a little-endian environment.

3.5.2 Hardware Breakpoint

The VxWorks simulator does not support hardware breakpoints.

3.5.3 Floating-Point Support

The VxWorks simulator does not support hardware floating-point instructions. However, VxWorks provides a floating-point library that emulates the following mathematical routines. All ANSI floating-point routines have been optimized using libraries from U. S. Software.

<code>acos()</code>	<code>asin()</code>	<code>atan()</code>	<code>atan2()</code>
<code>cos()</code>	<code>cosh()</code>	<code>exp()</code>	<code>fabs()</code>
<code>floor()</code>	<code>fmod()</code>	<code>log()</code>	<code>log10()</code>
<code>pow()</code>	<code>sin()</code>	<code>sinh()</code>	<code>sqrt()</code>
<code>tan()</code>	<code>tanh()</code>		

The following floating-point routines are *not* available on the VxWorks simulator:

<code>cbrt()</code>	<code>ciel()</code>	<code>infinity()</code>	<code>rint()</code>
<code>iround()</code>	<code>log2()</code>	<code>round()</code>	<code>sincos()</code>
<code>trunc()</code>	<code>cbrtf()</code>	<code>infinityf()</code>	<code>rintf()</code>
<code>iroundf()</code>	<code>log2f()</code>	<code>roundf()</code>	<code>sincosf()</code>
<code>truncf()</code>			

In addition, the following single-precision routines are *not* available:

<code>acosf()</code>	<code>asinf()</code>	<code>atanf()</code>	<code>atan2f()</code>
<code>cielf()</code>	<code>cosf()</code>	<code>expf()</code>	<code>fabsf()</code>
<code>floorf()</code>	<code>fmodf()</code>	<code>logf()</code>	<code>log10f()</code>
<code>powf()</code>	<code>sinf()</code>	<code>sinhf()</code>	<code>sqrtf()</code>
<code>tanf()</code>	<code>tanhf()</code>		

3.5.4 ISR Stack Protection

ISR stack overflow and underflow protection is supported on Solaris and Linux simulators. The VxWorks simulator does not require ISR stack overflow and underflow protection on Windows simulators because the Windows operating system automatically detects this type of error condition and handles it before VxWorks can take action.

For more information on ISR stack protection, see the *VxWorks Kernel Programmer's Guide: Memory Management*.

3.5.5 Interrupts

This section discusses interrupt simulation on the VxWorks simulator and how interrupts are handled in the simulator environment.

Solaris and Linux Systems

On Solaris and Linux simulators, the hardware interrupt simulation is performed using host signals. For example, the VxWorks simulator uses the **SIGALRM** signal to simulate system clock interrupts.

Furthermore, all host file descriptors (such as standard input) can be put in asynchronous mode, so that the **SIGPOLL** signal is sent to the VxWorks simulator when data becomes available. For more information on how to configure a host device to generate interrupts when data is available, refer to [A. Accessing Host Resources](#).

For the VxWorks simulator, signal handlers provide the equivalent functionality of interrupts available on other target architectures. You can install ISRs in the VxWorks simulator to handle these interrupts.



NOTE: Not all VxWorks routines can be called from ISRs. For more information, see the *VxWorks Kernel Programmer's Guide*.

To run ISR code during a future system clock interrupt, use the watchdog timer facilities. To run ISR code during auxiliary clock interrupts, use the **sysAuxClkxxx()** routines.

Table 3-2 shows how the Linux and Solaris simulator interrupt vector tables are assigned.

Table 3-2 Interrupt Assignments (Linux and Solaris Simulators)

Interrupt Vectors	Description
1	Host signal 1
...	
SIGUSR1	User signal 1
SIGUSR2	User signal 2
...	
32	Host signal 32
33	Interrupt vector for host file descriptor 1 (SIGPOLL)
...	
288	Interrupt vector for host file descriptor 256 (SIGPOLL)

You can create pseudo-drivers to use these interrupts. You must connect your interrupt code with the standard VxWorks **intConnect()** mechanism.

For example, to install an ISR that logs a message whenever the host signal **SIGUSR2** arrives, execute the following commands:

On Solaris:

```
% intConnect (17, logMsg, "Help!\n")
```

On Linux:

```
$ intConnect (12, logMsg, "Help!\n")
```

Next, send the **SIGUSR2** signal to the VxWorks simulator from the host. This can be done using the **kill** command. The ISR (**logMsg()** in this case) runs every time the signal is received.



NOTE: In your VxWorks applications, avoid using the preprocessor constants **SIGUSR1** or **SIGUSR2**. VxWorks defines its own values for these constants and those values differ from the host definitions. Therefore, you must specify the host signal numbers explicitly in your VxWorks application code.

Only **SIGUSR1** and **SIGUSR2** can be used to represent user-defined interrupts (see Table 3-3).

Table 3-3 **User-Defined Interrupts (Linux and Solaris Simulators)**

Signal	Solaris Value	Linux Value
SIGUSR1	16	10
SIGUSR2	17	12

Windows Systems

On Windows the VxWorks simulator uses Windows messages to simulate hardware interrupts. For example, the VxWorks simulator uses messages to simulate interrupts from the network connections, the pipe back end, and so forth.

For the VxWorks simulator, messages provide the equivalent functionality of interrupts available on other target architectures. You can install ISRs in the VxWorks simulator to handle these interrupts.



NOTE: Not all VxWorks routines can be called from ISRs. For more information, see the *VxWorks Kernel Programmer's Guide*.

To run ISR code during a future system clock interrupt, use the watchdog timer facilities. To run ISR code during auxiliary clock interrupts, use the **sysAuxClkxxx()** routines.

Table 3-4 shows how the Windows simulator interrupt vector table is assigned.

Table 3-4 **Interrupt Assignments (Windows Simulators)**

Interrupt Vectors	Description
0x0000	system clock interrupt
0x0001	auxiliary clock interrupt
0x0002	timestamp rollover interrupt
0x0003	back end pipe interrupt
0x0004	SIO driver interrupt
0x0005	bus interrupt
0x0006-0x0009	inter-processor interrupts (IPIs)

Table 3-4 Interrupt Assignments (Windows Simulators) (cont'd)

Interrupt Vectors	Description
0x0009-0x00ef	reserved for internal use
0x00f0-0x00ff	Wind River Media Library interrupt range
0x0100-0x017f	ULIP interrupt range
0x180-0x01ff	simulated network interrupt range
0x0200-0x02ff	user interrupt range

You can create pseudo-drivers to use these interrupts. You must connect your interrupt code using the standard VxWorks **intConnect()** mechanism.

For example, the following code installs an ISR that logs a message whenever an auxiliary clock message arrives. In this example, the auxiliary clock rate is configured to generate two ticks per second using **sysAuxClkRateSet()** so that the message is logged every 500 ms.

```
% sysAuxClkRateSet (2)
value = 0 = 0x0
% sysAuxClkEnable ()
value = 0 = 0x0
% intConnect (0x1, logMsg, "Aux Clock Int!\n")
```

The user interrupt range can be used by the host side user application. For more information on using user interrupts, refer to [A. Accessing Host Resources](#).

3.5.6 Memory Layout

The VxWorks memory layout is the same for all VxWorks simulators. [Figure 3-1](#) shows the memory layout, labeled as follows:

Boot Line

ASCII string of boot parameters.

Exception Message

ASCII string of the fatal exception message.

System Image

The VxWorks system image itself (three sections: text, data, and bss). The entry point for VxWorks is at the start of this region.

Host Memory Pool

Memory allocated by the host tools. The size depends on the **WDB_POOL_SIZE** macro.

System Memory Pool

Size depends on the size of the system image. The **sysMemTop()** routine returns the address of the end of the free memory pool.

Interrupt Stack

Size is defined by **ISR_STACK_SIZE** under **INCLUDE_KERNEL**. The location depends on the system image size.

Interrupt Vector Table

Table of interrupt vectors.

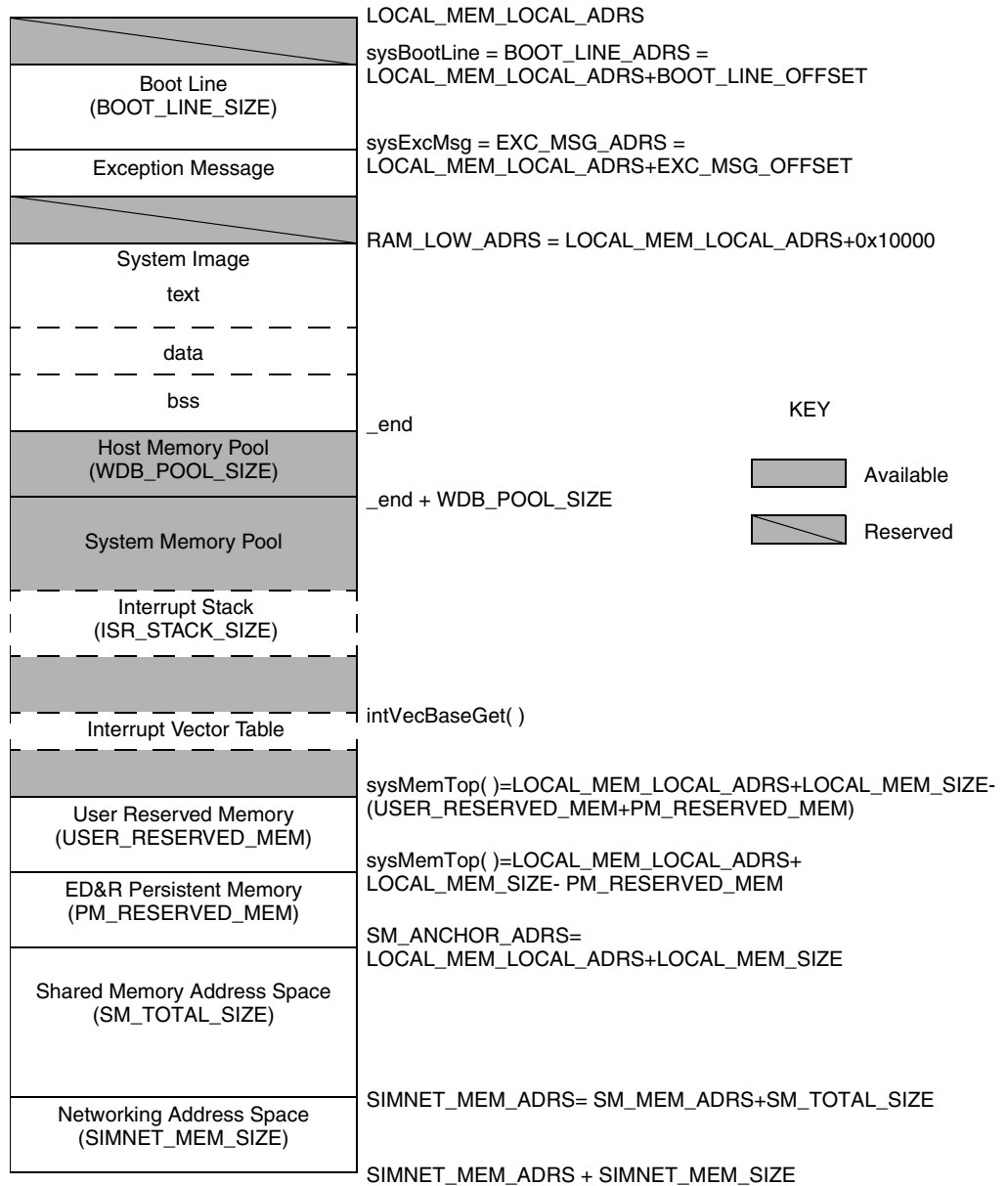
Shared Memory Address Space

Address space reserved for shared memory, which includes the shared memory anchor, the shared memory pool, and the address space for VxMP shared memory objects (if included) or the shared memory TIPC pool (if included).

Networking Address Space

Address space for VxWorks simulator networking (if network support is included).

Figure 3-1 **VxWorks System Memory Layout (VxWorks Simulator)**



4

Using the VxWorks Simulator

- 4.1 Introduction 33
- 4.2 Configuring the VxWorks Simulator 34
- 4.3 Configuring Hardware Simulation 36
- 4.4 Using VxWorks SMP with the VxWorks Simulator 43
- 4.5 Migrating Applications to a Hardware-Based System 45

4.1 Introduction

This chapter discusses how to use the VxWorks simulator for VxWorks development. It includes information on configuration, hardware simulation, and basic guidelines and limitations for migrating your application to a target hardware system.

4.2 Configuring the VxWorks Simulator

This section discusses configuration issues particular to the VxWorks simulator environment, including boot parameter configuration. For more information on general VxWorks configuration, see the *VxWorks Kernel Programmer's Guide*.

4.2.1 Boot Parameters

Using the command-line interface, you can specify all parameters available in the standard VxWorks boot line for a VxWorks simulator target. However, you lose these parameters when you exit the simulator even if you include non-volatile RAM support. This occurs because, even when the specified boot parameters are saved in a file (**nvr_{am}.vxWorkscpuNum**) similar to actual hardware target behavior, they are only used for a target reboot, not when the target is exited or restarted.

Once the VxWorks simulator starts, you can use the VxWorks **bootChange()** routine to modify boot line parameters. The new parameters are preserved and taken into account on the next reboot.



NOTE: The **bootChange()** routine can be used to boot another VxWorks image. However, the new image must be built with the same memory configuration. That is, the **LOCAL_MEM_SIZE** and **LOCAL_MEM_LOCAL_ADRS** macros in the BSP **config.h** file must be identical.

4.2.2 Memory Configuration

The VxWorks simulator displays its memory settings at startup, as shown in the following example:

```
Virtual Base Address: 0x10000000
Virtual Top Address: 0x50000000    Virtual Size: 0x40000000 (1024Mb)
Physical Base Address: 0x10000000
Physical Top Address: 0x12000000    Physical Size: 0x02000000 (32Mb)
```

The following sections discuss the VxWorks simulator memory parameters and describe how the parameters can be modified.

Physical Memory Address Space

The VxWorks simulator physical memory address space is defined by the **LOCAL_MEM_LOCAL_ADRS** and **LOCAL_MEM_SIZE** parameters in the VxWorks simulator BSP.

The VxWorks simulator physical memory size can be dynamically modified using the VxWorks simulator command-line interface (using the **-size** or **-memsize** command line options).



NOTE: The **LOCAL_MEM_ADRS** parameter must be aligned to 1 MB (0x100000) and the **LOCAL_MEM_SIZE** parameter must be a multiple of 1 MB.



NOTE: If you modify **LOCAL_MEM_ADRS**, you may need to use the **-vaddr** command line option to set a virtual address value that is coherent with the physical memory address space.

Virtual Memory Address Space

The VxWorks simulator virtual memory size is limited to 1 GB.

On Windows hosts, the VxWorks simulator virtual base address is 0x10000000 and the VxWorks simulator virtual top address is 0x4FFFFFFF.

On Solaris and Linux hosts, the VxWorks simulator virtual base address is 0x60000000 and the VxWorks simulator virtual top address is 0x9FFFFFFF.



NOTE: Depending on your host configuration, you may obtain less than 1 GB of virtual memory.

The default settings for the virtual memory base address and the virtual memory size should work for most host configurations. However, you may need to modify the virtual memory values in order to avoid a conflict between the VxWorks simulator address space and the host system DLL load addresses. You may also need to decrease the base address in order to get a larger address space. The default values for the virtual memory base address and the virtual memory size can be overridden using the **-vaddr** and **-vsize** command line options.



NOTE: If you decide to modify the virtual memory base address or virtual memory size, you must ensure that the values are coherent with the physical memory address space.

Memory Protection Level

The VxWorks simulator allows you to specify a memory protection level using the **-prot-level** option. This level can be set to **min**, **max**, or an intermediate integer value representing a given protection level. By default, the memory protection is set to the maximum level (**max**).



NOTE: Currently, only one protection level is provided. See [Table 4-1](#).

Table 4-1 Available Memory Protection Levels

Protection Level	Description
0 (min)	No specific protection
1 (max)	Enable stack overflow protection

4.2.3 Miscellaneous Configuration

The VxWorks simulator command-line interface also provides a set of miscellaneous options for scripting, help, version, and so forth. For complete information on all available options, refer to the API reference entry for **vxsim**.

4.3 Configuring Hardware Simulation

This section discusses the available hardware simulation options for the VxWorks simulator.

4.3.1 Pass-Through File System (passFS)

The default file system for the VxWorks simulator is the pass-through file system (passFS). This file system is unique to the VxWorks simulator. The **INCLUDE_PASSFS** component is included by default and mounts this file system on startup. passFS is a file-oriented device driver that provides easy access to the

host file system. To specify the passFS device name (the default is your system host name), use the following command-line option:

```
% vxsim -hn hostname
```

or

```
% vxsim -hostname hostname
```

On Linux and Solaris hosts, the default value for the passFS device name is the name of the host on which the simulator is running. On Windows, for backward compatibility with previous releases, the default value is **host**.

The VxWorks syntax for accessing a host file system is summarized in [Table 4-2](#).

Table 4-2 VxWorks Syntax for Accessing passFS

Host Type	passFS Syntax	Example
Linux or Solaris	<i>passFSDevice:/dir/file</i>	ls myhost:/myDir/myFile (where host name is myHost)
Windows	<i>passFSDevice:/disk/dir/file</i>	ls host:/c/myDir/myFile
Windows (deprecated, syntax preserved for backward compatibility)	<i>passFSDevice:disk:/dir/file</i>	ls host:c:/myDir/myFile



NOTE: passFS uses UNIX-style path separators (/) even on a Windows-based simulator.

4.3.2 Virtual Disk Support

To simulate access to file systems, either the file system supplied with VxWorks or one you have implemented yourself, the VxWorks simulator includes support for virtual disks. A virtual disk converts all read and write accesses to read and write accesses to a file on the host file system. However, to an application running in a VxWorks image, accessing the virtual disk looks no different than any other disk in a VxWorks I/O system.



NOTE: Virtual disk support replaces the UNIX disk driver library (**unixDrv**) included in earlier versions of the VxWorks simulator.

By default, the VxWorks simulator includes support for virtual disks. The relevant configuration component is **INCLUDE_VIRTUAL_DISK**. To initialize the virtual disk system, call **virtualDiskInit()**. After control returns from a successful call to **virtualDiskInit()**, you can create a virtual disk instance by calling **virtualDiskCreate()**:

```
BLK_DEV * virtualDiskCreate
(
    char *   hostFile,          /* name of the host file to use      */
    int      bytesPerBlk,       /* number of bytes per block         */
    int      blksPerTrack,      /* number of blocks per track        */
    int      nBlocks            /* number of blocks on this device   */
)
```

Although a successful call to **virtualDiskCreate()** creates a disk instance, there is not yet any name or file system associated with the instance. To create this association, you must call a file system device initialization routine. Consider the following code fragment:

```
BLK_DEV * vdBlkDev;
vdBlkDev = virtualDiskCreate (hostFile, 512, 32, 32*200);
fsmNameInstall("/Q:0", "/Q");
xbd = xbdBlkDevCreateSync(vdBlkDev, "/Q");
status = dosFsVolFormat("/Q", DOS_OPT_QUIET | DOS_OPT_BLANK, NULL);
```

This code creates **/Q**, a 3 MB DOS disk with 512-byte blocks, and 32 tracks. In support of this virtual disk, the **virtualDiskCreate()** call creates the file **c:/tmp/filesys1** (if the file does not already exist). Do not delete this file while the virtual disk is open (to close a virtual disk, call the **virtualDiskClose()** routine). To check whether this code has successfully created a virtual disk, you can call **devs()**, which should return the following:

```
drv name
0 /null
1 /tyCo/0
5 host:
6 /vio
3 C:
```

4.3.3 Non-Volatile RAM Support

By default, a VxWorks image includes support for non-volatile RAM, which has a default size of 256 bytes. This memory is dedicated to storing boot line information. To store anything else in NVRAM, use the **NV_RAM_SIZE** macro to

increase the size of NVRAM. To access NVRAM, use `sysNvRamSet()` and `sysNvRamGet()`.

To simulate NVRAM, the VxWorks simulator uses a file on the host system. By default, this file resides in the same directory that contains the VxWorks image. To specify another location, use the `-nvram` command-line option:

```
% vxsim -nvram pathToFileForNVRAM
```

4.3.4 Standard I/O

VxWorks simulator BSPs provide a standard I/O (SIO) driver to handle standard input and output. For Windows, Linux, and Solaris simulators, this driver is `target/config/simpc/simSio.c`.



NOTE: On Linux and Solaris simulators, UNIX job control characters are enabled even when the I/O is in raw mode. Trapping of control characters like `^Z` is UNIX-shell specific and does not conform to the usual VxWorks `tyLib` conventions. Trapping of the `^C` character is performed by the kernel shell (when it is included in your image).

4.3.5 Timers

Similar to any VxWorks target, the VxWorks simulator provides a system clock and an auxiliary clock. The macros `SYS_CLK_RATE_MIN`, `SYS_CLK_RATE_MAX`, `AUX_CLK_RATE_MIN`, and `AUX_CLK_RATE_MAX` are defined to provide parameter checking for the `sysClkRateSet()` and `sysAuxClkRateSet()` routines.



NOTE: If the VxWorks simulator process is preempted by another process on the host machine, the VxWorks simulator clock can be impacted. In such cases, the current activity of each VxWorks task is delayed by an interval of time that corresponds to the preempted time of the process.

4.3.6 Timestamp Driver

The VxWorks simulator provides a system-defined timestamp driver. In general, this driver is used to extend the range of information available from VxWorks kernel instrumentation. For example, when a timestamp driver is available, a precise time line can be displayed using the Wind River System Viewer.

The timestamp driver is included in the default VxWorks simulator configuration. The timestamp driver is selected by including the **INCLUDE_TIMESTAMP** and **INCLUDE_SYS_TIMESTAMP** components in your VxWorks image.



NOTE: If the VxWorks simulator process is preempted by another process on the host system, the System Viewer graph can be impacted. In this situation, the current activity of each VxWorks task is delayed by an interval of time that corresponds to the preempted time of the process.

4.3.7 Serial Line Support

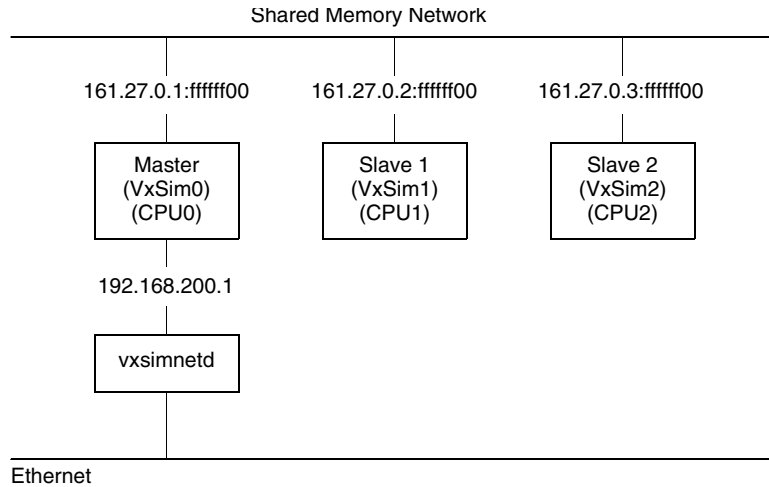
The VxWorks simulator provides a sample host serial I/O driver (**hostSio**) is. This driver provides access to a host serial device from the VxWorks simulator. This feature is not included in the default VxWorks simulator. To add host serial device support, the **INCLUDE_HOST_SIO** component must be defined in the BSP configuration. The macro **HOST_SIO_PORT_NUMBER** can be used to select which host serial device to use. For more information, see [A. Accessing Host Resources](#).

4.3.8 Shared Memory Network

You can configure a VxWorks system where multiple CPU boards are connected using a common backplane (for example, a VMEbus configuration). This allows the target boards to communicate through shared memory. VxWorks provides a standard network driver to access shared memory such that all higher-level network protocols are available over the backplane. In a typical configuration, one of the CPU boards (CPU 0) communicates with the host using Ethernet. The rest of the CPU boards communicate with each other, and the host, using the shared memory network. In this configuration, CPU 0 acts as a gateway to the outside network.

This type of hardware configuration can be emulated using the VxWorks simulator. In this case, multiple VxWorks simulator instances are configured to use a host shared-memory region as the basis for the shared-memory network.

Figure 4-1 VxWorks Simulator Shared-Memory Network



Configuring Your VxWorks Simulator for a Shared-Memory Network

In order to configure the VxWorks simulator for use with a shared-memory network, you must configure your VxWorks image to use the following components:

```
INCLUDE_SM_NET
INCLUDE_SM_COMMON
INCLUDE_SM_OBJ
```

You can reconfigure your image using either the **vxprj** command-line utility or using the Workbench kernel configuration tool. For more information, see the *Wind River Workbench User's Guide* or the *VxWorks Command-Line Tools User's Guide*.

Starting the Master Simulator

Use the master simulator to communicate with the host through the **simnet** device. You can specify the shared-memory network address for the master simulator by starting the VxWorks simulator instance with the **-backplane** (or **-b**) option as follows:

```
% vxsim -p 0 -d simnet -e 192.168.200.1 -b 161.27.0.1:ffffff00
```



NOTE: Because it is responsible for initializing the shared memory region, the master simulator must always be started first. You must also reboot all slave instances each time the master instance is rebooted.

Starting the Slave Simulators

Once you start the master simulator instance, you can start slave simulator instances with a gateway set to the master simulator using the **-gateway** (or **-g**) option as follows:

```
% vxsim -p 1 -b 161.27.0.2:ffffff00 -g 161.27.0.1
% vxsim -p 2 -b 161.27.0.3:ffffff00 -g 161.27.0.1
```

An alternative option would be to start the slave simulator as follows:

```
% vxsim -p 1 -b 161.27.0.2:ffffff00
```

and then add a network route to specify which gateway should be used for communication. This is done from the VxWorks simulator kernel shell as follows:

```
-> routec "add -net 0.0.0.0/24 161.27.0.1"
```



NOTE: If you choose to use the alternative option described above, you must include the **INCLUDE_ROUTECD** component in your VxWorks image.

Configuring the Host System

Before your host system can communicate with the master simulator, you must configure your host routing table with the new subnet information. The routing information can be configured as follows:

For Windows hosts, enter the following command from a Windows command shell:

```
C:\> route add 161.27.0.0 MASK 255.255.255.0 192.168.200.1
```

For Solaris and Linux hosts, enter the following command from your host shell:

```
# route add -net 161.27.0.0 192.168.200.1
```



NOTE: To configure routing information, you must have administrator or root privileges on your host.

4.4 Using VxWorks SMP with the VxWorks Simulator

The VxWorks simulator provides support for VxWorks SMP. This allows you to test SMP applications before SMP hardware is available. Even if you have not purchased the VxWorks SMP option, you can use the precompiled SMP image located in *installDir/target/proj/vxsimBSP_smp/default*. For example, on Windows hosts, the precompiled SMP image is *installDir\target\proj\simpc_diab_smp\default\vxWorks*.

The host OS simulates each CPU with a process that is scheduled by the host OS itself. Thus SMP performance can be achieved when the host has more CPUs than the VxWorks simulator. Otherwise, since simulated CPUs have to share the real CPU power, performance will not match SMP expectations. You specify the number of simulated CPUs using the Workbench kernel configurator before building the VxWorks image.

If you are using the VxWorks simulator to simulate multiple processors in SMP mode, you should be cautious when calling host routines from the VxWorks simulator. Review code that uses `vxsimHostProcAddrGet()` to make sure it is SMP safe. Note that `vxsimHostProcCall()` replaces a direct function pointer dereference that is forbidden in SMP. See [A.2 Accessing Host OS Routines](#), p. 108 for more detailed information. For general background on SMP, see the *VxWorks Kernel Programmer's Guide: VxWorks SMP*.

4.4.1 Creating an SMP Image

If you have purchased the VxWorks SMP option, you can create a custom SMP-enabled VxWorks simulator image. To create a new VxWorks simulator image that includes support for SMP:

1. Create a new VxWorks image project (VIP) in Workbench. In the **Options** dialog of the wizard, check the **SMP support in kernel** option checkbox.
2. If you want to change the number of processor cores (the default is two), use the **Kernel Configuration** facility. Go to **operating system components > kernel components > kernel**. Double-click on the line labeled **Number of CPUs enabled** (which corresponds to the parameter `VX_SMP_NUM_CPUS`). Change the highlighted number (2) to the desired number of CPUs.
3. Build the project.

4. Select **Target > New Connection**, then select the connection type Wind River VxWorks 6.x Simulator Connection.
5. Click **Next** then click on **Custom simulator** and navigate to the project directory for the VIP you just built.

When you connect to the SMP-enabled VxWorks simulator instance, the startup banner should indicate that VxWorks SMP is running. In addition, as shown in [Figure 4-2](#), the startup sequence reports both the number of CPU cores that the VxWorks image expects and the actual number of CPU cores present on the host machine.

Figure 4-2 VxWorks Simulator SMP Startup Screen

At the VxWorks simulator prompt, you should be able to see the idle tasks running on the two different CPU cores by issuing the **i** command:

-> i

NAME	ENTRY	TID	PRI	STATUS	PC	SP	ERRNO	CPU #
tJobTask	10055d20	1040adb0	0	PEND	100e7d3c	1064ff98	0	-
tExcTask	10054f30	101a5130	0	PEND	100e7d3c	101ae360	1c0001	-
tLogTask	logTask	1040ed08	0	PEND	100e590f	1068ff2c	0	-
tNbioLog	100569f0	1040f1a0	0	PEND	100e7d3c	106cff44	0	-
tShell0	shellTask	105439b8	1	READY	100f0d80	108fe394	0	0
tWdbTask	wdbTask	104ad8c8	3	PEND	100e7d3c	108aff3c	0	-
tAioIoTask>	aioIoTask	1042e408	50	PEND	100e83d5	1074ff8c	0	-
tAioIoTask>	aioIoTask	1042e828	50	PEND	100e83d5	1078ff8c	0	-
tNet0	ipcomNetTask	1042f4f8	50	PEND	100e7d3c	107cff64	0	-
ipcom_sysl>	10107950	1044d318	50	PEND	100e83d5	105efea0	0	-
ipnetd	1010bc50	1044e540	50	PEND	100e7d3c	1086ff80	6	-
tAioWait	aioWaitTask	1042aee8	51	PEND	100e7d3c	1070ff0c	0	-
tIdleTask0	idleTaskEntr	1038ad40	287	READY	1002781c	1038abe8	0	-
tIdleTask1	idleTaskEntr	103c6000	287	READY	1002781c	103c5ea8	0	1

value = 0 = 0x0
->

4.5 Migrating Applications to a Hardware-Based System

Kernel and RTP applications developed using the VxWorks simulator are easily transferred to target hardware systems. However, because the VxWorks simulator environment is not a suitable basis for developing hardware device drivers, more work may be required once your application is transferred to the target system.

To migrate your application, change your project build specifications to reflect the new hardware-based system. This involves recompiling your code using the appropriate CPU type for your target hardware.

For more information on building applications for your target architecture, see the *VxWorks Architecture Supplement*. For general application build instructions, see the *Wind River Workbench User's Guide*.

5

Networking with the VxWorks Simulator

- 5.1 Introduction 47
- 5.2 Building Network Simulations 48
- 5.3 Setting Up the Network Daemon 50
- 5.4 Installing the Host Connection Driver 66
- 5.5 Configuring a Simulated Subnet 70

5.1 Introduction

The VxWorks simulator provides support for setting up a subnet using a network of VxWorks simulator instances. This chapter discusses how to configure your system and the required VxWorks simulator instances for use in a simulated subnet. It includes general network simulation information, instructions for setting up the VxWorks simulator network daemon (**vxsimnetd**) and installing the host connection driver, and information on configuring your simulated network. Tutorials for setting up a simulated network are provided in [6. Networking Tutorials](#).



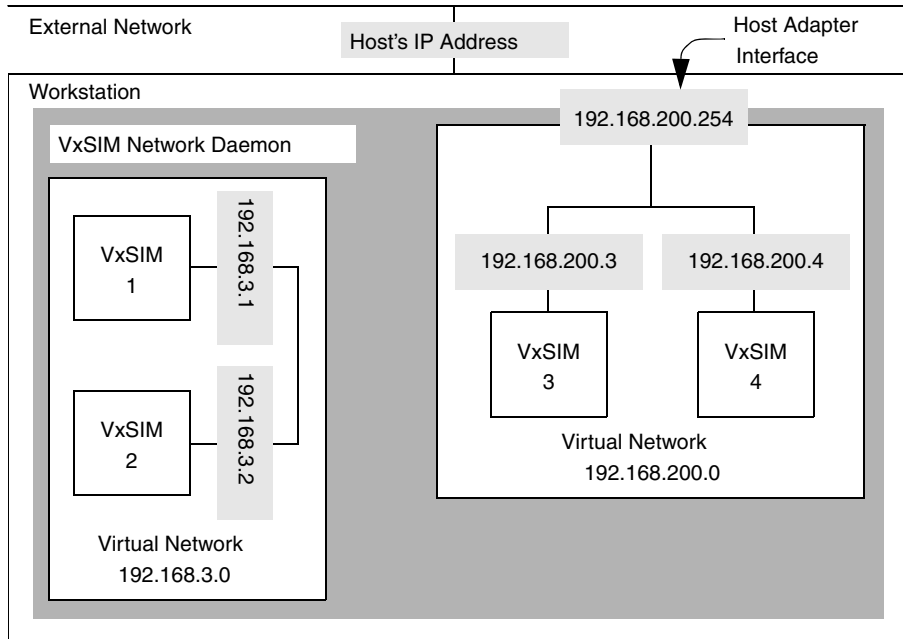
NOTE: The VxWorks simulator supports IPv6. For IPv6 configuration information, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide, Volume 1: Transport and Network Protocols*. For information on using the VxWorks simulator with IPv6, see [6.5 IPv6 Tutorial](#), p.95.

5.2 Building Network Simulations

Using the VxWorks simulator network daemon, you can link same-host VxWorks simulator instances into simulated subnets. By default, these internal subnets do not communicate with the host. However, included with the VxWorks simulator is a simulated network drive—a host adapter interface—that you can use to give the host system an address on the simulated subnet.

Through the host adapter, packet sniffers, such as **tcpdump**, **snoop**, or **ethereal**, can monitor traffic on an internally simulated subnet. In addition, this adapter on the simulated subnet lets you use the host to route packets for the subnet and thus link it with an external network. [Figure 5-1](#) shows two subnets, 192.168.3.0 and 192.168.2.0, simulated on a single host.

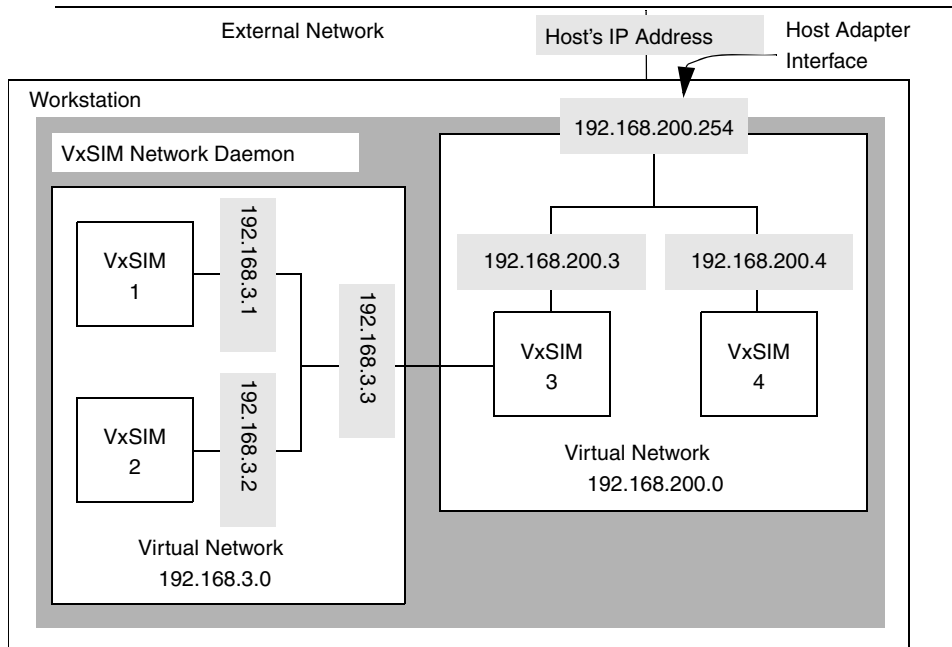
Figure 5-1 VxWorks Simulator Instances on Simulated Subnets



Virtual subnet 192.168.3.0 is an entirely internal subnet. It is isolated from the host, although you can use a kernel shell or the target server (using the WDB pipe back end) to access a target. After you have access to one target on the subnet, you can use the subnet to communicate with other targets on the subnet. Virtual subnet 192.168.200.0 is an “external” subnet. It is networked to the host workstation through the host adapter interface, 192.168.200.254. If you set up the host system route table correctly, the host system can route packets for the 192.168.200.0 subnet.

As shown in [Figure 5-2](#), it is possible to create a multiple interface VxWorks simulator instance. Using such a VxWorks simulator instance, it is possible to route between simulated subnets.

Figure 5-2 VxWorks Simulator Instances Can Support Multiple Network Interfaces



In [Figure 5-2](#), the multi-interface **VxSIM 3** can route packets from the 192.168.3.0 subnet to the greater external network through the host adapter to the host, which can then route packets to the external network through its network interface.

5.3 Setting Up the Network Daemon

The VxWorks simulator includes a network daemon that you can use to link multiple VxWorks simulator instances into one or more subnets. You can also use this network daemon to link these subnets (or even individual VxWorks simulator instances) to the larger Internet. The network daemon can support any protocol over the Ethernet layer (for example, TCP/IP). Thus, you can use the VxWorks simulator instances to test any broadcasting or multicasting features you may have built into an application.



NOTE: Although the VxWorks simulator network daemon allows you to set up complex simulated networks, it is also required for minimal networks—that is, a connection between the host and a single simulator instance.

The remainder of this section tells you how to set up a VxWorks simulator network daemon. Using the VxWorks simulator network daemon, you can link same-host VxWorks simulator instances into simulated subnets. By default, these internal subnets do not communicate with the host. However, included with the VxWorks simulator is a simulated network drive—a host adapter interface—that you can use to give the host system an address on the simulated subnet. For more information on the VxWorks simulator host connection driver, see [5.4 Installing the Host Connection Driver](#), p. 66.



NOTE: If you want to use a VxWorks simulator instance(s) on a simulated network, you must start the VxWorks simulator network daemon before you start the VxWorks simulator instance. Keep in mind that even a connection between the host system and a single instance requires the network daemon.

5.3.1 Starting the Network Daemon

The VxWorks simulator network daemon can be started either as a service (Windows service or root service on Linux and Solaris), which is the recommended method, or from the command line. The following sections describe each of these methods.

Starting the Network Daemon as a Service

Wind River recommends starting **vxsimnetd** as a Windows service (or a root service on Linux and Solaris) because this method provides full network support for the VxWorks simulator even if you are not logged in with administrator or root privileges on the host system.



NOTE: You must remove any previously installed network daemon services before attempting to install a new service. For example, if you installed a network daemon service (**vxsimnetds**) as part of an earlier VxWorks simulator release, you must remove the old service before attempting to install a service from the latest release. (This may be the case even if you uninstalled your previous installation using a Wind River uninstall utility.) For information on removing the network daemon service, see [5.3.2 Removing the Network Daemon Service](#), p.57.

Starting vxsimnetd as a Windows Service

To install **vxsimnetd** as a Windows service:

1. Log in to the Windows host with administrator privileges.
2. If you have an existing **vxsimnetd** service from a previous VxWorks simulator installation, you must remove the service. To uninstall the service, select **Run...** from the Windows **Start** menu and execute the following command:

```
installDir/vxworks-6.x/host/x86-win32/bin/vxsimnetds_inst.exe /u
```

where *installDir* is the name of your VxWorks installation directory.

3. Install the new network daemon. From the Windows **Start** menu, select **Run....** Browse to *installDir/vxworks-6.x/host/x86-win32/bin/vxsimnetds_inst.exe* (where *installDir* is the name of your VxWorks installation directory) and click **OK** to run the file.



NOTE: You must run **vxsimnetds_inst.exe** with administrator privileges.

To start the service:

1. Open **Control Panel > Administrative Tools**.
2. Click **Services**.
3. Select **Wind River Network Daemon for VxWorks Simulator** and start the service by right-clicking and selecting **Start**. (If the service is set to **Disabled**, right-click **Wind River Network Daemon for VxWorks Simulator**, select **Properties**, and set the startup type to **Automatic**. You can then start the service by right-clicking and selecting **Start**.)

By default, the network daemon starts with the default 192.168.200.0 external subnet configuration and with a shell server (**-sv** option). To change these options, right-click **Wind River Network Daemon for VxWorks Simulator**, select **Properties**, and specify the desired options before starting the service.

Once the network daemon service is started, non-administrator users can start VxWorks simulator instances and attach them to any configured subnet.

Special Considerations when Specifying a Configuration File

When the daemon is started as a service, the configuration parameters are saved in the registry. This allows you to start and stop the service without entering the configuration parameters each time. However, if the service is started and a configuration file is specified using the **-f** option, that file is used for configuration indefinitely. (For more information on configuration files, see [5.3.4 Creating a Network Daemon Configuration File](#), p.60.)

To specify new parameters, you must stop the service and specify new settings using one of the following options:

- Specify the **-f** option without a configuration file when starting the service. (The start parameters for the network daemon can be specified in the **Properties** dialog for the service.)
- Manually edit the registry keys using the Windows registry editor. The registry settings for the network daemon are located under **HKEY_LOCAL_MACHINE\Software\Wind River Systems\Wind River VxWorks Simulator Network Daemon**.



NOTE: Specifying a non-existent configuration file prevents the **vxsimnetd** service from starting.

Starting vxsimnetd as a Root Service on Solaris/Linux

You can create scripts for Solaris and Linux systems that start **vxsimnetd** automatically on reboot (use the **-sv** option if you want the ability to modify network configuration).

You can install **vxsimnetd** as a root service using the following steps:

1. Copy **vxsimnetd** to your **/usr/sbin** directory.
2. Create a **vxsimnetd** script as follows in **/etc/init.d**:

On Solaris, use the following script:

```
#!/bin/sh
#
# description: Starts and stops the vxsimnetd daemon
#              used to provide external network access to the
#              VxWorks simulator.
```

```
case "$1" in
    start)
        if [ -x /usr/sbin/vxsimnetd ] ; then
            echo "Starting vxsimnetd ..."
            /usr/sbin/vxsimnetd -sv
        fi
        ;;
    stop)
        echo "Stopping vxsimnetd ..."
        /usr/bin/pkill -x vxsimnetd
        ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1
esac

exit 0
```

On Linux, use the following script:

```
#!/bin/sh
#
# chkconfig: 3 91 02
# description: Starts and stops the vxsimnetd daemon \
#              used to provide VxWorks Simulator network services.
#
# pidfile: /var/run/vxsimnetd.pid

# Source function library.
if [ -f /etc/init.d/functions ] ; then
    . /etc/init.d/functions
elif [ -f /etc/rc.d/init.d/functions ] ; then
    . /etc/rc.d/init.d/functions
else
    exit 0
fi

# Check that vxsimnetd exists.
[ -x /usr/sbin/vxsimnetd ] || exit 0

RETVAL=0

start() {
    echo -n $"Starting vxsimnetd service: "
    daemon vxsimnetd -sv
    RETVAL=$?
    echo
    [ $RETVAL -eq 0 ] && touch /var/lock/subsys/vxsimnetd || \
        RETVAL=1
    return $RETVAL
}
```

```

stop() {
    echo -n $"Shutting down vxsimnetd services: "
    killproc vxsimnetd
    RETVAL=$?
    echo
    [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/vxsimnetd
    echo ""
    return $RETVAL
}

restart() {
    stop
    start
}

rhstatus() {
    status vxsimnetd
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        restart
        ;;
    status)
        rhstatus
        ;;
    condrestart)
        [ -f /var/lock/subsys/vxsimnetd ] && restart || :
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|status|condrestart}"
        exit 1
esac

exit $?

```

3. Create a link.

On Solaris, create a link in **/etc/rc3.d/** as follows:

```
# ln -s /etc/init.d/vxsimnetd S91vxsimnetd
```

On Linux, in **/etc/init.d/**, run:

```
$ /sbin/chkconfig --add vxsimnetd
```

This creates two links on **/etc/init.d/vxsimnetd**, **/etc/rc3.d/S91vxsimnetd** and **/etc/r6.d/K02vxsimnetd**.

4. Start **vxsimnetd** as follows:

```
# /etc/init.d/vxsimnetd start
```

or reboot your host.

Starting the Network Daemon From the Command Line

You can use the **vxsimnetd** command to start the VxWorks simulator network daemon on your host system. You can configure the daemon using a configuration file statically at startup time or you can configure it interactively using the daemon's debug shell. You can also combine these configuration methods which allows you to use a configuration file to supply some defaults and read in additional configuration files as needed.



NOTE: If the daemon must support an externally visible subnet, you must launch the daemon from a task with the appropriate privileges. On a Solaris or Linux host, this means starting the daemon with supervisor or root privileges. On a Windows host, this means starting the daemon with administrator privileges.

The **vxsimnetd** command supports the following options:

-f or -file

This option specifies the configuration file parsed when the network daemon starts. For more information on the format of this file, see [5.3.4 Creating a Network Daemon Configuration File](#), p.60.

-s or -shell

This option starts a debug shell that you can use to control network daemon configuration interactively. For more information on the debug shell options, see [5.3.3 Network Daemon Debug Shell](#), p.57.

-sv or -shellserver

This option starts the network daemon in server/background mode. When in background mode, you can telnet to a debug port to access the debug shell. The **-sv** and **-s** options are mutually exclusive.

-sp or -shellport

This option specifies the debug port used to start a shell on a network daemon in background mode. If not specified, the default port is 7777.

-force

Forces the deletion of IPC objects left after **vxsimnetd** dies. (UNIX only)

To configure the daemon statically, use a command such as the following, where **vxsimnetd.conf** is a file supplying configuration parameters:

```
% vxsimnetd -f vxsimnetd.conf
```

To start the VxWorks simulator network daemon interactively, use a command such as the following, where **vxsimnetd.conf** is a file supplying configuration parameters:

```
% vxsimnetd -f vxsimnetd.conf -s
```

If you use the **-sv** option instead of the **-s** option, the debug shell runs in the background and is accessible using **telnet**. For example:

```
% telnet hostname portNumber
```

The *portNumber* defaults to 7777, but **vxsimnetd** supports the **-sp** parameter, which you can use to specify a different port number.

vxsimnetd can also be started without any configuration options. In this case, the network daemon is started with a default external subnet of 192.168.200.0 and with the host node set in promiscuous mode.

5.3.2 Removing the Network Daemon Service



NOTE: The product uninstaller does not uninstall **vxsimnetd**. If you plan to uninstall the product, be sure to uninstall **vxsimnetd** (as described above) *before* running the uninstaller program.

To remove the network daemon service (**vxsimnetd**), open a VxWorks development shell and enter the following:

```
% vxsimnetds_inst.exe /u
```

This uninstalls the **vxsimnetd** service.

5.3.3 Network Daemon Debug Shell

You can access the network daemon debug shell by starting **vxsimnetd** with the **-s** (or **-shell**) or **-sv** (or **-shellserver**) option. The shell supports command line completion as well as history with two editing modes, emacs (default) or vi.

The available shell commands are:

subnet [*subnetName*]

This command displays subnet information. When no subnet is specified, the command lists a summary for all configured subnets. A detailed summary is provided when a subnet name is specified.

node *subnetName* [*nodeIp*]

node *subnetName* [*nodeNb*]

This command displays information about how many nodes are configured and used. For example:

```
vxsimnetd> node default
NODE INFORMATION:
      CONFIGURED IN-USE      MAX      TOTAL      FAIL
      33          1          1          1          0

Current Nodes of the subnet (default):
#  COMM      STATUS IP      PROMISC RCVQ  PID
0  special(*) UP      192.168.200.254 Yes    0   24076
```

If you specify a node number (*nodeNb*) or node IP address (*nodeIp*), the **node** command displays specifics about the particular node, such as the number of packets sent and received. For example:

```
vxsimnetd> node default 0
#  COMM      STATUS IP      PROMISC RCVQ  PID
0  special(*) UP      192.168.200.254 Yes    0   24076

MAC ADDRESS:
      Mac Address      7a:7a:c0:a8:c8:fe

SEND/RECEIVE STATISTICS:
# of receives      0
# of sends          6
# of send failures  6

RECEIVE QUEUE INFORMATION:
      CONFIGURED CURRENT  MAX
      64          0      0
```

packet *subnetName*

This command displays packet information for a subnet. For example:

```
vxsimnetd> packet default
PACKET INFORMATION:
      CONFIGURED IN-USE      MAX      TOTAL      FAIL
      100          1          1          7          0

HANGING PACKETS:
PKT-#  SEND-NODE      RECV-NODE      LEN      REFCNT      PKTPTR
0      192.168.200.254[E] N/A      1514      0      0xff0e2b14
```

In this example, the default subnet is configured with 100 packets, only one is currently in use, and seven packets were used over all. The hanging packets section displays packets that are allocated but not yet sent.

help *[command]*

This command specifies detailed help for a given command. If no command is specified, a summary of all available commands is provided.

?

Displays a one-line summary for all commands.

quit

This command exits the shell. If you started **vxsimnetd** with the **-s** option, this command destroys all subnets and **vxsimnetd** exits. For more information on **vxsimnetd** options, see [5.3.1 Starting the Network Daemon](#), p.51.

source *configFile*

This command reads subnet configuration information from a file and adds the corresponding subnets. If **vxsimnetd** cannot add all configured subnets, then this command adds no subnets at all.

delete *subnetName*

This command deletes a configured subnet. To delete all subnets, use **delete all**.

extpromisc *subnetName 0*

extpromisc *subnetName 1*

This command sets the promiscuous mode for the host node of an external subnet. The **0** option sets promiscuous mode to off, **1** sets promiscuous mode to on. When the external node is in promiscuous mode (**1**), it receives every packet sent on the subnet. While this heavily impacts network performance, it allows you to analyze network traffic by connecting a packet sniffer on the external host node interface.

erate *subnetName*

This command sets the error rate for a given subnet. The error rate is the percentage of packets that will *not* be sent without giving error notification to the sender. Thus, if the error rate is set at five percent, five randomly chosen packets per 100 will be purposely lost. This feature is provided to simulate packet loss on an actual subnet.

timeout *subnetName*

This command sets the subnet timeout value. If a node does not read any packets for the length of time specified as the timeout, the packets are picked up by garbage collection.

mode vi
mode emacs

This command sets the shell editing mode to vi or emacs.

5.3.4 Creating a Network Daemon Configuration File

As an option, the **vxsimnetd** command (the command used to start the network daemon) lets you specify a file containing network daemon configuration parameter values. To assign a value to a parameter, enter a semicolon (;) terminated line with the following general format:

```
PARAMETER = value;
```

Where *PARAMETER* is either a parameter name in capital letters or an alias.

For parameters related to a subnet, group those parameters using the following syntax:

```
SUBNET_START subnetName {  
    SUBNET_PARAM = value;  
};
```

For example, consider the following default configuration file:

```
SUBNET_START default {  
    SUBNET_EXTERNAL = yes;  
    SUBNET_EXTPROMISC = yes;  
    SUBNET_ADDRESS = "192.168.200.0";  
};
```

This configures the VxWorks simulator network daemon to support a subnet with external access. The network address for the subnet is 192.168.200.0 and, because the network mask is not specified, the pre-CIDR¹ default mask applies. For 192.168.200.0, that would be the mask for a class C address, which is 0xfffff00.

To add another subnet, you could add the lines:

```
SUBNET_START user1 {  
    SUBNET_UID = 323;  
    SUBNET_GID = 100;  
    SUBNET_ACCESSMODE = "0600";  
    SUBNET_ADDRESS = "192.168.201.0";  
};
```

The parameters supported in a VxWorks simulator network daemon configuration file are described in [Table 5-1](#).

1. CIDR refers to classless inter-domain routing. See RFCs 1518 and 1519.

Table 5-1 VxWorks Simulator Network Daemon Configuration Parameters

Parameter	Description
Default Parameters:	
DEFAULT_GARBAGE	Alias: dgarbage Default Value: 30 Specifies the number of seconds in the garbage collection interval. For each subnet, the garbage collection thread runs every DEFAULT_GARBAGE seconds.
DEFAULT_MACPREFIX	Alias: dmacprefix Default Value: 7a:7a Specifies the first bytes of simulator Ethernet addresses.
DEFAULT_UID	Alias: duid Default Value: user ID of the user that started the network daemon Defines the user ID (UNIX only).
DEFAULT_GID	Alias: dgid Default Value: group ID of the user that started the network daemon Defines the group ID (UNIX only).
DEFAULT_ACCESSMODE	Alias: daccessmode Default Value: "0666" Defines access mode (UNIX only). You can use the three parameters (duid , dgid , and daccessmode) to restrict access to subnets to a given user or group of users when the network daemon is shared between users on the same host.
DEFAULT_EXTERNAL	Alias: dexternal Default Value: no Defines the default subnet type.

Table 5-1 VxWorks Simulator Network Daemon Configuration Parameters (cont'd)

Parameter	Description
DEFAULT_EXTPROMISC	Alias: dextpromisc Default Value: yes Defines whether the external subnet host node is set in promiscuous mode.
DEFAULT_ERATE	Alias: derate Default Value: 0 Defines the default subnet error rate.
DEFAULT_TIMEOUT	Alias: dtimeout Default Value: -1 Defines how long packets queued to a VxWorks simulator instance are left in the queue. The default is forever.
Subnet-Specific Default-Override Parameters:	
SUBNET_MACPREFIX	Alias: macprefix Default: DEFAULT_MACPREFIX Specifies the first two bytes of the Ethernet address on this subnet. Overrides DEFAULT_MACPREFIX.
SUBNET_UID	Alias: uid Default: DEFAULT_UID Specifies the user IP for this subnet. Overrides DEFAULT_UID.
SUBNET_GID	Alias: gid Default: DEFAULT_GID Specifies the group ID for this subnet. Overrides DEFAULT_GID.
SUBNET_ACCESSMODE	Alias: accessmode Default: DEFAULT_ACCESSMODE Specifies the access mode for this subnet. Overrides DEFAULT_ACCESSMODE.

Table 5-1 VxWorks Simulator Network Daemon Configuration Parameters (cont'd)

Parameter	Description
Topology Parameters:	
SUBNET_ADDRESS	Alias: address Default: "0.0.0.0" Specifies the network address for this subnet.
SUBNET_MASK	Alias: mask Default: Pre-CIDR mask associated with the address in SUBNET_ADDRESS. Specifies the subnet mask for this subnet.
SUBNET_EXTERNAL	Alias: external Default: DEFAULT_EXTERNAL Specifies whether this subnet can communicate with the host on which it runs. This communication requires you to create a VxWorks simulator target with a network interface on the host system's network, and to start the VxWorks simulator network daemon with administrator privileges.
SUBNET_EXTPROMISC	Alias: extpromisc Default: DEFAULT_EXTPROMISC Specifies whether the host sees every packet sent on this subnet. It allows you to attach a sniffer on the host interface to monitor traffic. However, it has a dramatically negative impact on network performance.
SUBNET_EXTDEVNUM	Alias: extdevice Default: 0 Specifies the host device number to use. This parameter is required when using more than one external subnet.

Table 5-1 VxWorks Simulator Network Daemon Configuration Parameters (cont'd)

Parameter	Description
Resource-Related Parameters:	
SUBNET_MAXBUFFERS	Alias: maxbuffers Default: 100 Specifies the maximum number of packet buffers available.
SUBNET_MAXNODES	Alias: maxnodes Default: 32 Specifies the maximum number of simulators that can attach to this subnet.
SUBNET_RECQLEN	Alias: recvqlen Default: 64 Specifies how many packets can be queued to a simulator.
SUBNET_SHMKEY	Alias: shmkey Default: IP address Specifies the shared memory key.
Option Parameters:	
SUBNET_BROADCAST	Alias: broadcast Default: yes Specifies whether to allow MAC broadcast packets.
SUBNET_MULTICAST	Alias: multicast Default: yes Specifies whether to allow multicast packets.
SUBNET_ERATE	Alias: errorrate Default Value: DEFAULT_ERATE Defines the subnet error rate (the percentage of packet loss on this subnet)

Table 5-1 VxWorks Simulator Network Daemon Configuration Parameters (cont'd)

Parameter	Description
SUBNET_TIMEOUT	Alias: timeout Default Value: DEFAULT_TIMEOUT Defines how long packets that are queued are left in the queue before garbage collection removes them.
SUBNET_MTU	Alias: mtu Default Value: 1500 Defines the MTU value that a VxWorks simulator instance is configured to use when it attaches to a subnet.
SUBNET_EXTCONNNAME	Alias: extconnname Default: Specifies the network interface name to use for this subnet as set in Control Panel > Network Connections (Windows only).

Configuring Multiple External Subnets

The VxWorks simulator network daemon can be configured with multiple external subnets. However, the following caveats should be observed:

On Windows hosts:

You must install a VxWorks simulator host connection driver (WRTAP) for each external subnet. (For information on installing and configuring the WRTAP driver, see [5.4 Installing the Host Connection Driver](#), p.66.) You may also want to specify a name for the WRTAP device driver used by a given subnet through the SUBNET_EXTCONNNAME configuration parameter.

On Solaris and Linux hosts:

You must specify the device number of all but the first external subnet using the SUBNET_EXTDEVNUM parameter.

5.4 Installing the Host Connection Driver

This section provides instructions for installing the optional VxWorks simulator host connection driver. You need install this driver only if you want to set up an externally visible subnet (able to communicate with or through the host) of VxWorks simulator instances. After this host driver is installed, **vxsimnetd** automatically configures its IP address and mask according to the configuration file. Packet sniffers such as **tcpdump**, **snoop**, or **ethereal** can then be attached to the host interface to monitor traffic on the internal simulated subnet.



NOTE: If you are working on a Windows host, installing the host connection driver (the WRTAP driver) can have an impact on your host system performance. Before installing this driver, review the information in [5.4.1 Managing the WRTAP Driver on Windows Hosts](#), p.68.

Windows Hosts

Use the following instructions to install the WRTAP driver on Windows XP or Windows Vista hosts.

To Install the Driver on a Windows XP Host:

1. Open the **Control Panel**.
2. Double-click **Add Hardware** to open the **Add Hardware Wizard**, click **Next**.
3. Answer **Yes, I have already connected the hardware**, click **Next**.
4. Select **Add a new hardware device** (you may need to scroll down to see this option), click **Next**.
5. Select **Install the hardware that I manually select from a list (Advanced)**.
6. Click **Next**.
7. Select **Network Adapters**, click **Next**.
8. Click the **Have Disk...** button.
9. **Browse** to *installDir\vxworks-6.x\host\x86-win32\bin* (*installDir* is the name of your VxWorks installation directory).
10. Select **wrtap.inf** and click **Open**.
11. Click **OK** to select the directory.
12. Select **WindRiver WRTAP**, click **Next**.

13. Click **Next** to start installing the driver.
14. Click **Continue Anyway** in the **Hardware installation** pop-up window.
15. Click **Finish** to close the wizard.

To Install the Driver on a Windows Vista Host:

1. Open the **Control Panel** and select the classic view.
2. Double-click **Add Hardware** to open the **Add Hardware Wizard**, click **Next**.
3. Select **Install the hardware that I manually select from a list (Advanced)**.
4. Click **Next**.
5. Select **Network Adapters**, click **Next**.
6. Click the **Have Disk...** button.
7. **Browse** to *installDir\vxworks-6.x\host\x86-win32\bin* (*installDir* is the name of your VxWorks installation directory).
8. Select **wrtap.inf** and click **Open**.
9. Click **OK** to select the directory.
10. Select **WindRiver WRTAP**, click **Next**.
11. Click **Next** to start installing the driver.
12. Click **Install this driver software anyway** in the **Hardware installation** pop-up window.
13. Click **Finish** to close the wizard.



NOTE: If you intend to use more than one external subnet, repeat the above steps for each subnet. You must install and configure a **WindRiver WRTAP** driver individually for each subnet that is marked as external. (Windows hosts only).

For more information on using the WRTAP driver on Windows hosts, see [5.4.1 Managing the WRTAP Driver on Windows Hosts](#), p.68.

Solaris Hosts

To install the VxWorks simulator host connection driver on a Solaris host:

1. Copy *installDir/vxworks-6.x/host/sun4-solaris2/bin/tap* to a directory accessible by root.
2. Become the administrator.

3. Go to the directory to which you copied the **tap** package.
4. Install the **tap** package as follows:

```
# pkgadd -d tap
```
5. Select the Universal TAP device driver and answer “yes” to run the install scripts.

Linux Hosts

The **tun** module required by the TAP driver must be available in your Linux distribution.



NOTE: The **tun** driver is available by default as part of the core kernel package for Red Hat Enterprise Linux Workstation 4.0 and later versions. It is also available as part of the default distribution for SuSE Linux 9.2. However, the driver is *not* available in the core kernel package of Red Hat Workstation 3.0, update 4 or earlier.

If you are using an earlier release of Red Hat (prior to Linux kernel version 2.4.21-20), the **tun** module is part of the kernel-unsupported RPM package. To use the **tun** module with Red Hat Workstation 3.0, update 4 or earlier, you must update your Linux kernel to version 2.4.21-20 and install the kernel-unsupported RPM package.

The **tun** module should be loaded automatically when **vxsimnetd** is started. However, some OS versions require you to load the module into the kernel. To do this, first check that the module is present:

```
$ modinfo tun
filename:      /lib/modules/2.4.21-20.EL/unsupported/drivers/net/tun.o
description:   <none>
author:        <none>
license:       "GPL"
```

To load the module into the kernel, type:

```
$ modprobe tun
```

5.4.1 Managing the WRTAP Driver on Windows Hosts

The following information may be useful when installing and using the host connection (WRTAP) driver on a Windows host. For instructions on installing the WRTAP driver, see the [5.4 Installing the Host Connection Driver](#), p.66.

Migrating from the ULIP Driver

The WRTAP driver replaces the ULIP driver used in earlier VxWorks simulator releases. You can use the WRTAP driver even if the ULIP driver is installed.

Handling Networking Problems on Your Host System

If you encounter networking problems with the VxWorks simulator or your host system after installing the WRTAP driver, you may need to make certain changes to your Windows network connection settings.



NOTE: You will need administrator privileges on the Windows host to make the following changes.

To access Windows network connection settings, select **Start > Control Panel > Network Connections** (on Windows XP hosts) or **Start > Control Panel > Network and Sharing Center > Manage network connections** (on Windows Vista hosts).

Certain communication protocols (particularly those which alter the maximum transmission unit (MTU) setting of the interface) can cause problems when the WRTAP driver is in use. In some instances, you may need to remove all protocols except TCP/IP. To do this, right click on each network connection and select **Properties**. Under the **General** tab, uncheck all items and components except TCP/IP (**Internet Protocol TCP/IP**).



NOTE: Removing all protocols except TCP/IP has no impact on the general host system. The change only applies to the WRTAP interface that is used for communication between the host system and the VxWorks simulator.

When you install the WRTAP driver, it becomes the primary network connection type on your host system. This can cause other applications to run slowly and can cause failures on the host system. To replace WRTAP as your main network connection, do the following:

- In the **Network Connections** (or **Network and Sharing Center > Manage network connections**) window, select **Advanced > Advanced Settings...** (On Vista hosts, you need to press the **Alt** key to access the **Advanced** menu.)
- Under the **Adapters and Bindings** tab, move your main Ethernet interface to the top of the **Connections** list.

Disabling the WRTAP Driver

IP address configuration for the WRTAP network connection is handled automatically by the VxWorks simulator network daemon. However, by default, the WRTAP network connection is turned on immediately following installation and uses DHCP to configure its IP address. To avoid the DHCP configuration, you can disable the WRTAP network connection after installation and allow it to be restarted and configured by the VxWorks simulator network daemon when necessary.

To disable the WRTAP network connection, access the Windows network connection settings by selecting **Start > Control Panel > Network Connections** (on Windows XP hosts) or **Start > Control Panel > Network and Sharing Center** (on Windows Vista hosts). In the **Network Connections** (or **Network and Sharing Center**) window, right-click the WRTAP network interface you want to disable and select **Disable**. (You can see that an interface is connected using the WRTAP driver by right-clicking and selecting **Properties**. The device the interface is using is listed in the **Connect using** field under the **General** tab.)

5.5 Configuring a Simulated Subnet

This section describes how to configure simulated subnets for use with the VxWorks simulator.

5.5.1 Starting a Simulator Instance With Multiple Network Interfaces

If you need to configure a VxWorks simulator instance with multiple network interfaces (a router configuration), **vxsim** includes the **-ni** option. The syntax for this options is as follows:

deviceNameDeviceNumber:subnet=IP_address:IP_netmask

This describes one interface. You can chain these descriptions together using a semi-colon (;) as a delimiter. For example:

```
% vxsim -ni simnet2=192.168.2.1:0xfffffffff0;simnet3=192.168.3.1:0xfffffffff0;  
simnet4=192.168.4.1:0xfffffffff0
```



NOTE: When launching the **vxsim** command from the command line on Linux and Solaris hosts, you must use double quotes (") around the **-ni** option parameter values to prevent the UNIX shell from interpreting the semi-colon (;). For example:

```
% vxsim -ni "simnet2=192.168.2.1:0xffffffff0;simnet3=192.168.3.1:0xffffffff0;  
simnet4=192.168.4.1:0xffffffff0"
```

Double quotes should *not* be used when passing this option to the command line using Workbench or when using the command line on a Windows host.

This command starts a VxWorks simulator instance configured with three simulated network interfaces that link the target with three very small subnets.



NOTE: When using the Wind River Workbench **New Connection** wizard to launch your VxWorks simulator, the **-ni** option can be passed to the simulator using the **Other VxWorks simulator options** field of the **VxWorks Simulator Miscellaneous Options** dialog.

5.5.2 Starting a Simulator Instance Without an IPv4 Address

You can start a VxWorks simulator instance and attach to a subnet through its name. For example, you can use the following commands:

```
% vxsim -d simnet  
% vxsim -ni simnet
```

These commands start a simulator that attaches to the first configured subnet (neither IPv4 address is specified). In this example, a MAC address can no longer be deduced from the IP address so a node number is used instead. The first attaching simulator instance gets 7a:7a:0:0:0:1, and the second instance gets 7a:7a:0:0:0:2 where 7a:7a is the subnet MAC prefix of the first configured subnet.



NOTE: In this example, the MAC address is no longer fixed and can change if the simulator instance is rebooted. This may cause a problem with ARP tables.

You can also use the following command:

```
% vxsim -ni simnet0:default
```

Use this command to start a VxWorks simulator instance that attaches to a subnet named **default**. The MAC address is determined as described in the earlier example.

It is also possible to get a fixed MAC address by specifying an IPv4 address that is not used to configure the **simnet** interface if the component

`INCLUDE_NET_BOOT_CONFIG` is not defined. Thus, you can use the following commands:

```
% vxsim -d simnet -e 192.168.3.1
% vxsim -ni simnet1=192.168.3.1
```

This command sequence starts a VxWorks simulator instance with a simulated subnet interface with a MAC address of 7a:7a:c0:a8:03:01 and an IP address (or addresses) that can be configured later using the **ifconfig()** command.

6

Networking Tutorials

- 6.1 Introduction 73
- 6.2 Simple Simulated Network 74
- 6.3 Basic Simulated Network with Multiple Simulators 78
- 6.4 Running the VxWorks Simulator on the Local Network 90
- 6.5 IPv6 Tutorial 95

6.1 Introduction

This chapter presents tutorials that provide step-by-step instruction for setting up a simulated network of VxWorks simulator instances. The network simulation is demonstrated using the ping function. This chapter also includes an IPv6 tutorial that describes how to set up your host system and VxWorks simulator instances to communicate using IPv6 protocol.



NOTE: When launching the **vxsim** command from the command line on Linux and Solaris hosts (as instructed in the tutorials throughout this chapter), you must use double quotes (") around the **-ni** option parameter values to prevent the UNIX shell from interpreting the semi-colon (;). For example, the following line:

```
% vxsim -ni simnet2=192.168.200.1;simnet3=192.168.3.1;simnet4=192.168.4.1
```

should be executed as follows when using the Linux or Solaris command line interface:

```
% vxsim -ni "simnet2=192.168.200.1;simnet3=192.168.3.1;simnet4=192.168.4.1"
```

Double quotes should *not* be used when passing options to the command line using Workbench or when using the command line on a Windows host.

6.2 Simple Simulated Network

The most basic (and common) network used by the VxWorks simulator is a network set up between the host and a single VxWorks simulator instance. You can set up this simple network using the default VxWorks simulator configuration. Therefore, the following tutorial does not require you to reconfigure or rebuild the default VxWorks image provided with the VxWorks simulator BSP.

This tutorial describes how to:

1. Set up and start the VxWorks simulator network daemon (**vxsimnetd**).
2. Start a single VxWorks simulator instance.
3. Test the simulator network by pinging the VxWorks simulator instance from the host.

This tutorial can be performed with any supported host using the command-line utility, **vxprj**, or Wind River Workbench.

6.2.1 Set Up the Network Daemon

The first step in setting up a VxWorks simulator network is to start the network daemon. For this tutorial, you start the network daemon is started on the host before you configure any VxWorks simulator instances.

Installing the VxWorks Simulator Host Connection Driver

Before configuring and starting the VxWorks simulator network daemon, you must install the VxWorks simulator host connection driver (WRTAP driver). If you have not already installed the host connection driver, do so now. Instructions for all supported hosts are provided in [5.4 Installing the Host Connection Driver](#), p.66.

Configuring the Network Daemon

This tutorial uses the default configuration for the VxWorks simulator network daemon. Therefore, **vxsimnetd** can be started without any options and no custom configuration file is required.



NOTE: The default configuration uses a default subnet of 192.168.200.0. If this subnet already exists on your host, you must change the VxWorks simulator network daemon configuration file. For more information on the default configuration options as well as other network daemon configuration file options, see [5.3.4 Creating a Network Daemon Configuration File](#), p.60.

Starting the Network Daemon on the Host System



NOTE: Wind River recommends that you start **vxsimnetd** as a service, see [Starting the Network Daemon as a Service](#), p.51 for complete instructions. If you have already started the default **vxsimnetd** as a service or choose to do so now, you can skip the instructions in this section.

To start the VxWorks simulator network daemon in the default configuration:

On Windows hosts:

1. Log in as administrator or be sure you have administrator privileges.
2. Open a Windows command shell (**Start > Run...**, then type **cmd**)
3. In the command shell, type the following:

```
C:\> installDir\vxworks-6.x\host\x86-win32\bin\vxsimnetd
```

where *installDir* is the name of your VxWorks installation directory.

On Linux hosts:

1. Open a host shell and log in as root.

2. In the host shell, type the following:

```
$ installDir/vxworks-6.x/host/x86-linux2/bin/vxsimnetd
```

where *installDir* is the name of your VxWorks installation directory.

On Solaris hosts:

1. Open a host shell and log in as root.
2. In the host shell, type the following:

```
# installDir/vxworks-6.x/host/sun4-solaris2/bin/vxsimnetd
```

where *installDir* is the name of your VxWorks installation directory.

6.2.2 Start a VxWorks Simulator Instance

Next, you need to start a VxWorks simulator instance from the command line in the VxWorks development shell or from the Workbench New Target Connection wizard. Again, the default VxWorks configuration is used for this tutorial therefore you do not need to reconfigure or rebuild the default VxWorks image for the simulator.

Start the Simulator Instance from the Command Line

To start a VxWorks simulator instance from the command line, do the following:

1. Open the VxWorks development shell.

On Windows hosts, select **Start > Wind River > VxWorks 6.x and General Purpose Technologies > VxWorks Development Shell**.

On Solaris and Linux hosts, run the **wrenv** utility program to open a development shell as follows:

```
% wrenv.sh -p vxworks-6.x
```

2. In the VxWorks development shell, type the following:

On Windows hosts:

```
C:\> vxsim -d simnet -e 192.168.200.1 -f  
installDir\vxworks-6.x\target\config\simpc\vxWorks
```

where *installDir* is the name of your VxWorks installation directory.

On Linux or Solaris hosts:

```
% vxsim -d "simnet" -e "192.168.200.1" -f  
"installDir/vxworks-6.x/target/config/bsp/vxWorks"
```

where *installDir* is the name of your VxWorks installation directory and *bsp* is the name of the BSP directory for the VxWorks simulator on your host (**linux** for Linux hosts or **solaris** for Solaris hosts).

The above command starts a VxWorks simulator instance and attaches it to the default subnet. The **VxSim0** console windows appears.

6

Start the Simulator Instance from Workbench

To start the VxWorks simulator instance from Workbench, complete the following steps:

1. Select **Target > New Connection....** This launches the **New Connection** wizard.
2. In the **Connection Type** dialog box, select **Wind River VxWorks 6.x Simulator Connection** from the selection list. Click **Next**.
3. In the **Select boot file name** field of the **VxWorks Boot parameters** dialog, click the **Standard simulator (Default)** radio button (this option should be selected by default). This selects the default VxWorks image for the VxWorks simulator.
4. Enter **0** in the **Processor Number** field (this is the default value).
5. Click the **Advanced Boot Parameters...** button. In the **boot device** field, select **simnet** from the drop down list. In the **Inet on ethernet (e)** field, enter **192.168.200.1** (this is the default value). Leave all other fields with their default settings. Click **OK**. This returns you to the **VxWorks Boot parameters** dialog of the **New Connection** wizard, click **Next**.
6. The **VxSim Memory Options** dialog appears. Click **Next** to accept the default options.
7. The **VxWorks Simulator Miscellaneous Options** dialog appears. Click **Next** to accept the default options (most options are blank by default).
8. The **Target Server Options** dialog appears. Click **Next** to accept the default options.
9. The **Object Path Mappings** dialog appears. Click **Next** to accept the default options.

10. The **Target State Refresh** dialog appears. Click **Next** to accept the default options.
11. The **Default Breakpoint Options** dialog appears. Click **Next** to accept the default options.
12. The **Connection Summary** dialog appears. Click **Finish**.

This boots VxWorks on the simulator target and launches the **VxSim0** console window. If your target connection fails or you encounter problems during configuration, see the *Wind River Workbench User's Guide* for more information.

6.2.3 Test the Simulated Network

To test that the simulated network is working, ping the simulator instance from your host system. From a host command window or shell, type:

```
% ping 192.168.200.1
```

6.3 Basic Simulated Network with Multiple Simulators

The following tutorials present the steps required to set up a simulated network with multiple VxWorks simulator instances. Two configuration options are described. The first tutorial creates a subnet with a static configuration. The second tutorial launches the VxWorks simulator network daemon (**vxsimnetd**) in interactive mode. The interactive mode starts a **vxsimnetd** shell that allows you to dynamically configure and monitor the subnet. (For more information on the VxWorks simulator network daemon, see [5.3 Setting Up the Network Daemon](#), p.50.)

The following tutorials require you to configure and build a new VxWorks image for the VxWorks simulator. The steps required to build and configure the image are included in this tutorial. However, if you require more information on building and configuring VxWorks, see the *VxWorks Application Programmer's Guide* or the Wind River Workbench documentation.

6.3.1 Creating a Static Configuration

The following tutorial takes you through the steps of creating a simulated network with a static configuration. The following steps are performed:

1. Configure and launch the VxWorks simulator network daemon (**vxsimnetd**).
2. Configure and build a VxWorks image for use with the VxWorks simulator instances.
3. Launch the required simulator instances.
4. Run the ping application.

Step 1: Configure and Launch vxsimnetd

1. The first step in setting up your simulated network is to set up and start **vxsimnetd**. Before completing the following steps, be sure that you have:
 - Installed the VxWorks simulator host connection driver (WRTAP driver). (Instructions for installing the driver are available in [5.4 Installing the Host Connection Driver](#), p.66.)
 - Stopped any previously started VxWorks simulator network daemons (including those started as a service, see [Starting the Network Daemon as a Service](#), p.51).
2. Now, create a **vxsimnetd** configuration file. For the purposes of this tutorial, this file is used to configure the network for use with a simulated router (see [Configure and Start the Simulated Router](#), p.83).

Create the following file and save it as **vxsimTest.conf**:

```
SUBNET_START sub2 {  
    SUBNET_ADDRESS = "192.168.200.0";  
    SUBNET_EXTERNAL = yes;  
    SUBNET_EXTPROMISC = yes;  
};  
SUBNET_START sub3 {  
    SUBNET_ADDRESS = "192.168.3.0";  
    SUBNET_EXTERNAL = no;  
};  
SUBNET_START sub4 {  
    SUBNET_ADDRESS = "192.168.4.0";  
    SUBNET_EXTERNAL = no;  
};
```

3. Next, launch **vxsimnetd** using the configuration file you just created.

On Windows hosts:

Log in with administrator privileges and start **vxsimnetd** from a command window as follows:

```
C:\> installDir\vxworks-6.x\host\x86-win32\bin\vxsimnetd -f  
vxsimTest.conf
```

Be sure to provide the full path to your **vxsimTest.conf** file if it is not in your current directory.

On Linux and Solaris hosts:

Log in as root and start **vxsimnetd** from a host shell as follows:

```
# installDir/vxworks-6.x/host/myHost/bin/vxsimnetd -f vxsimTest.conf
```

In this command, *myHost* is your host type: **x86-linux2** for Linux hosts or **sun4-solaris2** for Solaris hosts.

Be sure to provide the full path to your **vxsimTest.conf** file if it is not in your current directory.



NOTE: You can also start **vxsimnetd** as a service. For instructions, see [Starting the Network Daemon as a Service](#), p.51. If you choose to start **vxsimnetd** as a service, you will need to configure the daemon as directed in this section (by passing the **-f** option to the service) before proceeding with the tutorial.



NOTE: If you do not start **vxsimnetd** with administrator (or root) privileges, **vxsimnetd** prints a warning. In this case, you will not be able to connect the host system to the simulated network. However, all other simulated network functionality is available.

Step 2: Prepare a VxWorks Image for Use with the Simulated Network

In this tutorial, the **VxSim0** simulator instance acts as a router. This tutorial uses ping to communicate between simulator instances therefore ping functionality must be enabled. Because this functionality is not included in the default VxWorks image, you must configure and build a new VxWorks image for use with the simulated network.

To properly configure the VxWorks image, you must include the **INCLUDE_ROUTECMD** and **INCLUDE_PING** components in your custom VxWorks image. Once this configuration is in place, you must rebuild the VxWorks image for the simulator.

Prepare Your VxWorks Image Using the vxprj Command-Line Utility

To reconfigure the VxWorks image with the necessary components using the command-line project facility, **vxprj**, complete the following steps:

1. Generate a project. This can be done from the command line as follows:

In the VxWorks development shell, type the following:

On Windows hosts:

```
C:\> vxprj create simpc TOOL network_demo
```

where *TOOL* is your chosen compiler (**diab** for the Wind River Compiler or **gnu** for the GNU compiler).

On Linux hosts:

```
$ vxprj create linux TOOL network_demo
```

where *TOOL* is your chosen compiler (**diab** for the Wind River Compiler or **gnu** for the GNU compiler).

On Solaris hosts:

```
% vxprj create solaris TOOL network_demo
```

where *TOOL* is your chosen compiler (**diab** for the Wind River Compiler or **gnu** for the GNU compiler).

This creates a project directory under *installDir* with the name, **network_demo**.

2. Add the **INCLUDE_ROUTECD** and **INCLUDE_PING** components to the image:

```
% cd network_demo  
% vxprj component add INCLUDE_ROUTECD INCLUDE_PING
```

3. Now, rebuild VxWorks. In the project directory (*installDir/network_demo*), execute **make**.

Prepare Your VxWorks Image Using Workbench

1. Generate a project.
 - a. In Workbench, select **File > New > VxWorks Image Project**. This launches the **New VxWorks Image Project** wizard.
 - b. In the **Project** dialog, enter **network_demo** in the **Project name** field. Select the location for your project in the **Location** field (**Create Project in Workspace** is selected by default). Click **Next**.

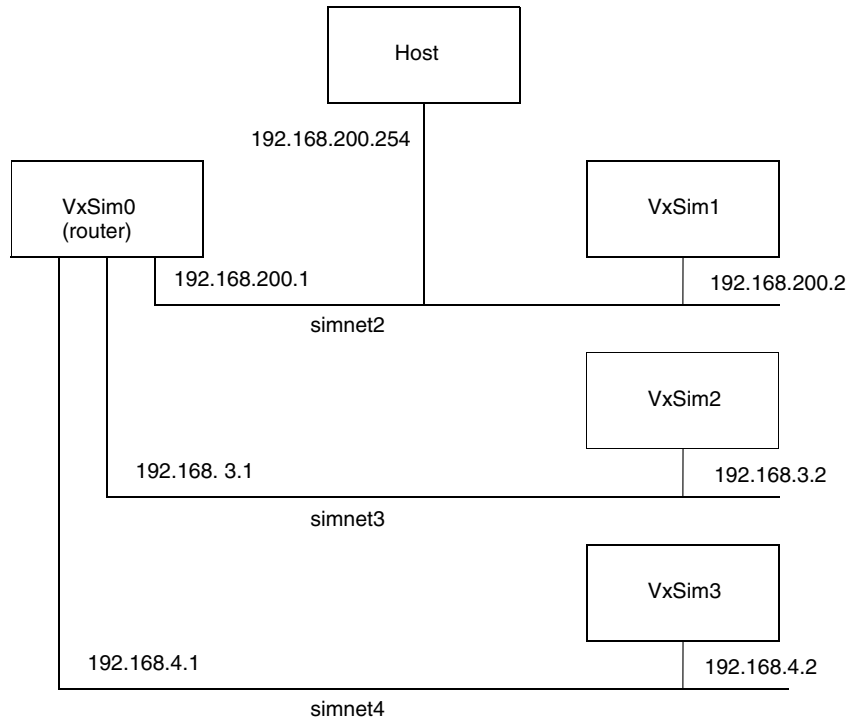
- c. In the **Project Setup** dialog, select the option to set up your project based on a board support package (this option is selected by default). And select the appropriate VxWorks simulator BSP for your host (for example, **simpc** on Windows hosts) and your desired tool chain (for example, **diab**). Click **Next**.
- d. In the **Options** dialog, click **Next** to accept the default options (no options are selected by default).
- e. In the **Configuration Profile** dialog, select **(no profile)** from the **Profile** drop-down list (this is the default). Click **Finish**.

This creates a project directory with the name, **network_demo**. The new project appears in the **Project Navigator** pane on the left.

2. Configure a custom VxWorks kernel to include the appropriate networking components.
 - a. Expand the new **network_demo** project and right-click **Kernel Configuration**. Select **Edit Kernel Configuration**. This opens the component configuration tool in the center pane of the **Application Development** window.
 - b. Select the **Components** tab at the bottom of the kernel configuration pane (if it is not already selected). Right-click in the component configuration field and select **Find**. Use the **Find** tool to locate the **INCLUDE_ROUTECMD** and **INCLUDE_PING** components. To find a component, type the component name (for example, **INCLUDE_PING**) in the **Pattern** field. When the component appears in the **Matching** list, select the component and click the **Find** button. This returns you to the component configuration tool. The selected component is highlighted in the component tree.
 - c. Right-click the selected component and select **Include (quick include)**.
 - d. Right-click in the component tree and select **Save**.
3. Build your project.
 - a. Now, right-click on the **network_demo** project in the **Project Navigator** (upper left pane) and select **Build Project** (this executes the **make** command in the project directory). The build output appears in the **Build Console** (lower right pane).

Step 3: Launch the VxWorks Simulator Instances

Now, start the simulator instances to attach to the configured subnets. This results in a simulated network with the following topology:



You can launch the simulator instances from the VxWorks development shell using the command line, or from Workbench.

Start the VxWorks Simulator Instances from the Command Line

If you have not already done so, change to the directory where you built your VxWorks image (for example, *installDir/network_demo/default*).

Configure and Start the Simulated Router

To configure the VxWorks simulator instance for the simulated router, type the following in the VxWorks development shell:

```
% vxsim -ni simnet2=192.168.200.1;simnet3=192.168.3.1;simnet4=192.168.4.1
```

This command sets up a network interface for the router on each subnet so that it can properly forward packets.

Once you execute this command, the **VxSim0** console window appears. (The **VxSim0** instance acts as the simulated router.)

Configure and Start the Simulated Network Instances

To configure three VxWorks simulator instances on the simulated network, type the following in the VxWorks development shell:

```
% vxsim -d simnet -e 192.168.200.2 -p 1
% vxsim -d simnet -e 192.168.3.2 -p 2
% vxsim -d simnet -e 192.168.4.2 -p 3
```

This command creates three instances on the simulated network; **VxSim1**, **VxSim2**, and **VxSim3**. One node is created on each of the three subnets you set up when you launched **vxsimnetd** (see [Step 1: Configure and Launch vxsimnetd](#), p.79).

Start the VxWorks Simulator Instances from Workbench

This section provides instructions for launching the simulated router and networked VxWorks simulator using Workbench.

Configure and Start the Simulated Router

First, start the VxWorks simulator instance that will act as the router in the simulated network. To start this instance from Workbench, complete the following steps:

1. Select **Target > New Connection....** This launches the **New Connection** wizard.
2. In the **Connection Type** dialog box, select **Wind River VxWorks 6.x Simulator Connection** from the selection list. Click **Next**.
3. In the **Select boot file name** field of the **VxWorks Boot parameters** dialog, click the **Custom simulator** radio button. Browse to your project location and click **Open** to select your VxWorks image (for example, *installDir/workspace/network_demo/default/vxWorks*).
4. Enter **0** in the **Processor Number** field. Click **Next**.
5. The **VxSim Memory Options** dialog appears. Click **Next** to accept the default options.

6. The **VxWorks Simulator Miscellaneous Options** dialog appears. In the **Other VxWorks simulator options** field, enter:

```
-ni simnet2=192.168.200.1;simnet3=192.168.3.1;simnet4=192.168.4.1
```

This option sets up a network interface for the router on each subnet so that it can properly forward packets.

Click **Next**.

7. The **Target Server Options** dialog appears. Click **Next** to accept the default options.
8. The **Object Path Mappings** dialog appears. Click **Next** to accept the default options.
9. The **Target State Refresh** dialog appears. Click **Next** to accept the default options.
10. The **Default Breakpoint Options** dialog appears. Click **Next** to accept the default options.
11. The **Connection Summary** dialog appears. Click **Finish**.

This boots VxWorks on a simulator target and launches the **VxSim0** console window. **VxSim0** acts as a router on the simulated network.

Configure and Start the Simulated Network Instances

Now, configure three VxWorks simulator instances, one instance per subnet (repeat the following process for each of the three instances). To configure each of the devices, complete the following steps:

1. Select **Target > New Connection....** This launches the **New Connection** wizard.
2. In the **Connection Type** dialog box, select **Wind River VxWorks 6.x Simulator Connection** from the selection list. Click **Next**.
3. In the **Select boot file name** field of the **VxWorks Boot parameters** dialog, click the **Custom simulator** radio button. Browse to your project location and click **Open** to select your VxWorks image (for example, *installDir/workspace/network_demo/default/vxWorks*).
4. Enter **1** in the **Processor Number** field. (Enter **2** for this option when starting the second instance and **3** when starting the third instance)
5. Click the **Advanced Boot Parameters...** button. In the **boot device** field, select **simnet** from the drop down list. In the **Inet on ethernet (e)** field, enter

192.168.200.2. (Enter **192.168.3.2** when starting the second instance and **192.168.4.2** when starting the third instance). Leave all other fields with their default settings. Click **OK**. This returns you to the **VxWorks Boot parameters** dialog of the **New Connection** wizard, click **Next**.

6. The **VxSim Memory Options** dialog appears. Click **Next** to accept the default options.
7. The **VxWorks Simulator Miscellaneous Options** dialog appears. Click **Next** to accept the default options (most options are blank by default).
8. The **Target Server Options** dialog appears. Click **Next** to accept the default options.
9. The **Object Path Mappings** dialog appears. Click **Next** to accept the default options.
10. The **Target State Refresh** dialog appears. Click **Next** to accept the default options.
11. The **Default Breakpoint Options** dialog appears. Click **Next** to accept the default options.
12. The **Connection Summary** dialog appears. Click **Finish**.

Repeat this process three times to create three VxWorks simulator instances, **VxSim1**, **VxSim2**, and **VxSim3**. This results in a node being configured on each of the three subnets you set up when you launched **vxsimnetd** (see [Step 1: Configure and Launch vxsimnetd](#), p.79).

Step 4: Set Up the Routing Table

Before pinging between simulator instances, you must set up the routing table for the simulated network.

In the **VxSim1** shell, type:

```
-> routec ("add -net 192.168.3.0/24 192.168.200.1");  
-> routec ("add -net 192.168.4.0/24 192.168.200.1");
```

In the **VxSim2** shell, type:

```
-> routec ("add -net 192.168.200.0/24 192.168.3.1");  
-> routec ("add -net 192.168.4.0/24 192.168.3.1");
```

In the **VxSim3** shell, type:

```
-> routec ("add -net 192.168.200.0/24 192.168.4.1");  
-> routec ("add -net 192.168.3.0/24 192.168.4.1");
```



NOTE: These route settings can be saved in files and run automatically. To do this, specify the saved file as a startup script when invoking **vxsim** from the command line as follows:

```
% vxsim -d simnet -e 192.168.200.2 -p 1 -s filename
```

where *filename* is the name of your startup script.

You can also specify the startup script when launching the simulator from Workbench by adding the startup script to the **startup script (s)** field in **Advanced Boot Parameter Options**.

6

Step 5: Run the Ping Application

To verify the network connections, ping **VxSim3** and **VxSim2** from **VxSim1** as follows:

```
-> ping "192.168.3.2", 5  
-> ping "192.168.4.2", 5
```

6.3.2 Creating a Dynamic Configuration Using the vxsimnetd Shell

The following steps demonstrate how to dynamically configure the VxWorks simulator using the network daemon shell.



NOTE: This tutorial is an extension of the static configuration tutorial presented in [6.3.1 Creating a Static Configuration](#), p.79. You must use the VxWorks image you created in the earlier tutorial to launch the VxWorks simulator instances in this tutorial.

Step 1: Launch the vxsimnetd Shell Server

Before launching a the **vxsimnetd** shell server, be sure to kill all previously started VxWorks simulator instances and then kill the previously started network daemon.

1. Start the **vxsimnetd** shell server. From the command shell on Windows or the host shell on Linux or Solaris, start **vxsimnetd** with the **-sv** option as follows:

```
# installDir\vxworks-6.x\host\myHost\bin\vxsimnetd -sv
```



NOTE: You must start **vxsimnetd** with administrator (or root) privileges. Once **vxsimnetd** is started, administrator privileges are no longer required.

Step 2: Configure vxsimnetd Dynamically Using the Shell

1. To configure **vxsimnetd** using the shell, you must create an additional subnet configuration file. Create and save the following file:

```
SUBNET_START sub3 {  
    SUBNET_ADDRESS = "192.168.3.0";  
    SUBNET_EXTERNAL = no;  
};  
SUBNET_START sub4 {  
    SUBNET_ADDRESS = "192.168.4.0";  
    SUBNET_EXTERNAL = no;  
};
```

2. Save this file as **sub_3_4.conf**.
3. Connect to the **vxsimnetd** shell. From the host shell, type the following:

```
% telnet yourHostName 7777
```

where *yourHostName* is the name of your host machine.

The **vxsimnetd** debug shell appears.

4. Source the new configuration file as follows:

```
vxsimnetd> source /myDir/sub_3_4.conf
```

```
Subnet(s) <sub3, sub4> added.
```

Step 3: Prepare a VxWorks Image

If you have not already done so, prepare a VxWorks image according to the instructions in [Step 2: Prepare a VxWorks Image for Use with the Simulated Network](#), p.80.

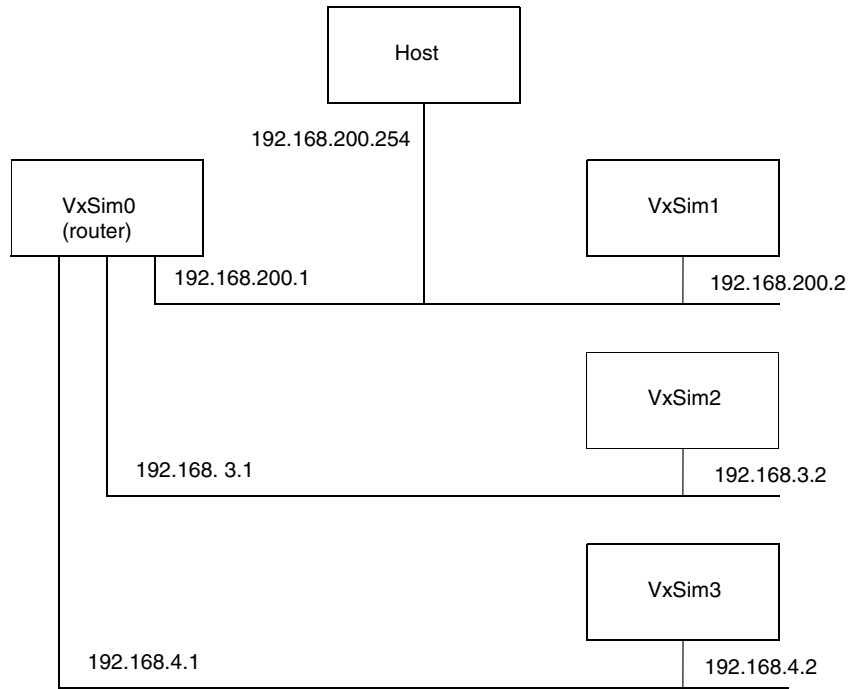


NOTE: If you have already completed the tutorial in [6.3.1 Creating a Static Configuration](#), p.79, you can use the VxWorks image you prepared in the **network_demo** project.

Step 4: Launch the VxWorks Simulator Instances

Launch the VxWorks simulator instances as described in [Step 3: Launch the VxWorks Simulator Instances](#), p.83.

Again, this sets up a simulated network with the following topology:



Step 5: Set Up the Routing Table

In the current configuration, **VxSim2** is configured to be externally available so it can be pinged from the host. However, before pinging the host, you must add the appropriate route information.

First, provide the appropriate routing information on your host system.



NOTE: To run the following route commands, you must have administrator privileges on Windows and supervisor privileges on UNIX.

For Windows hosts:

Type the following from the Windows command shell:

```
C:\> route add 192.168.3.0 MASK 255.255.255.0 192.168.200.1  
C:\> route add 192.168.4.0 MASK 255.255.255.0 192.168.200.1
```

For Solaris and Linux hosts:

Type the following from the host shell:

```
% route add -net 192.168.3.0 192.168.200.1
% route add -net 192.168.4.0 192.168.200.1
```

Next, add the appropriate routing information to the **VxSim2** instance. In the **VxSim2** console, type the following:

```
-> routec ("add -net 192.168.200.0/24 192.168.3.1");
```

Now, add the appropriate routing information to the **VxSim3** instance. In the **VxSim3** console, type the following:

```
-> routec ("add -net 192.168.200.0/24 192.168.4.1");
```

Step 6: Run the Ping Application

Now, you can verify the network connection by pinging **VxSim2** and **VxSim3** from the host shell as follows:

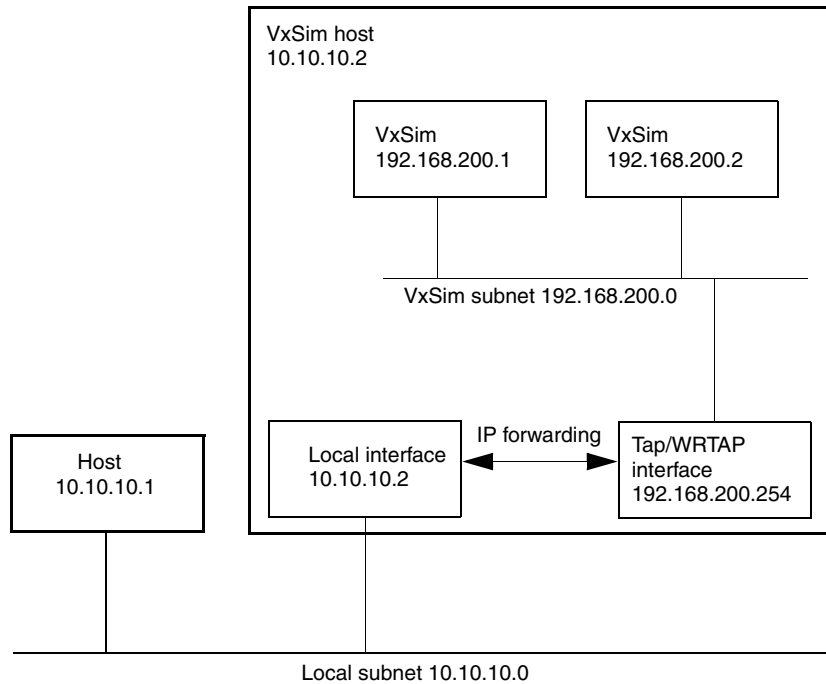
```
% ping 192.168.3.2
% ping 192.168.4.2
```

6.4 Running the VxWorks Simulator on the Local Network

This tutorial demonstrates how to plug the VxWorks simulator directly into your local network through a bridge configured on your host.

6.4.1 Default subnet configuration

By default, **vxsimnetd** configures the **wrtap/tap** interface with an IP address of 192.168.200.254. In this configuration, the host machine appears as a gateway between two subnets: the local one (for example, 10.10.10.0) and the VxSim subnet (192.168.200.0). See the diagram below for an illustration of the VxWorks simulator host machine and another host machine on the same local subnetwork.



in this case, if you want to access host 10.10.10.1 from the VxWorks simulator you need to set up routes as follows:

1. Indicate that you can reach the 10.10.10.0 subnet through the host interface 192.168.200.254:
2. on *every* remote host, indicate that host 10.10.10.2 is the gateway to reach the 192.168.200.0 network:

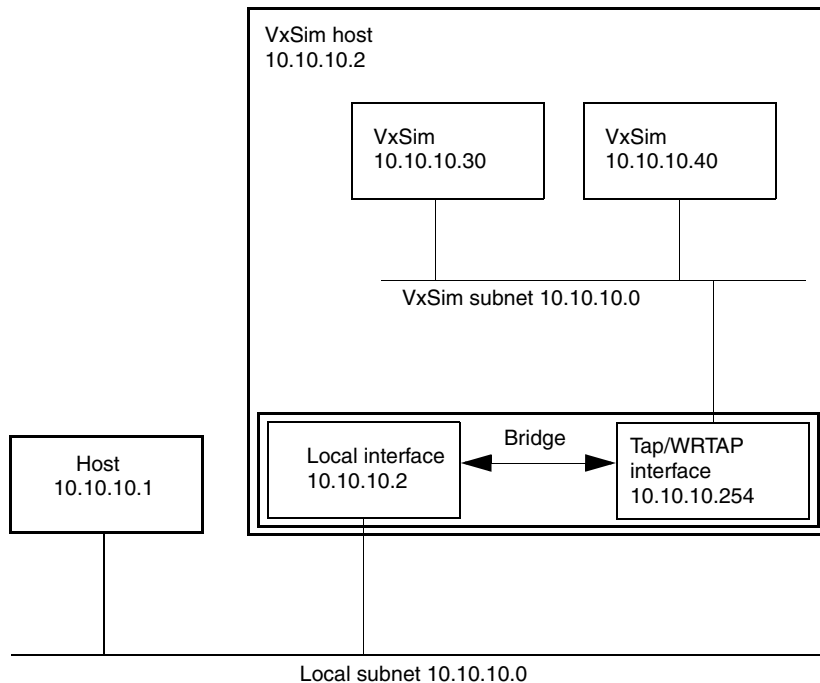
```
-> route add -net 10.10.10.0/24 192.168.200.254"
```

```
# on linux  
route add -net 192.168.200.0 gw 10.10.10.2
```

```
# on windows  
route add 192.168.200.0 mask 255.255.255.0 10.10.10.2
```

6.4.2 Configuring a Bridge

To simplify route settings, you can connect vxsim directly to your local network through a bridge configured on your host machine between the local network interface and the wrtap interface. The goal of setting up such a bridge is to allow a VxSim started on 10.10.10.2 to ping any host on the local subnet (without having to set up additional routes) as shown below:



Windows Setup



NOTE: This feature is supported on Windows XP and Windows Vista.

1. Create a vxsimnetd configuration file for your local subnet (this example assumes that your local subnet is 10.10.10.0):

```
subnet_start local_subnet {  
    subnet_address = "10.10.10.0";  
    subnet_mask = "255.255.255.0";  
    subnet_external = yes;  
    subnet_extpromisc = no;  
};
```

2. Start up **vxsimnetd** using the configuration file you just created. see [5.3.1 Starting the Network Daemon](#), p.51 for details.
3. Once you have started **vxsimnetd**, check the network interfaces in **control panel->network connections**: you should be able to see the wrtap interface configured on the same subnetwork as your local network interface (for example, local interface of 10.10.10.2, and wrtap interface of 10.10.10.254).
4. To set up the bridge, control-click to select both interfaces then right click and select **bridge** from the popup menu. select the newly created bridge then right click and select **properties**. configure the bridge as the local interface is configured (either dhcp or fixed address and gateway).
5. Start vxsim:

```
vxsim -d simnet -e 10.10.10.50
```

You should now be able to access any other host without modifying routing tables.
6. If you started **vxsimnetd** as a service with your local subnet configuration, the bridge should be available after a reboot of your host.

Linux bridge setup

To use the VxWorks simulator on a local bridge in Linux, you must ensure that the bridge module is installed on your Linux host, using the **modinfo bridge** command. If necessary, install the package that does contain this module. You also need the **brctl** command, available in the **bridge-utils** package (<http://bridge.sourceforge.net/>).

1. To make sure the bridge module is available and loaded on your linux host, run the following commands:

```
# modinfo bridge  
# modprobe bridge
```

2. Create a **vxsimnetd** configuration file for your local subnet (10.10.10.0 in this example):

```
subnet_start local_subnet {  
    subnet_address = "10.10.10.0";  
    subnet_mask = "255.255.255.0";  
    subnet_external = yes;  
    subnet_extpromisc = no;  
    subnet_extdevnum = 1;  
};
```

3. Start your **vxsimnetd** with this configuration file or source it through an existing **vxsimnetd** shell.

```
# as root  
> vxsimnetd -f local_subnet.conf -sv  
  
# or as normal user (if vxsimnetd was previously started as root with a  
shell server)  
> cp local_subnet.conf /tmp/local_subnet.conf  
> telnet localhost 7777  
vxsimnet> source /tmp/local_subnet.conf  
subnet(s) <local_subnet> added.
```

4. Set up the bridge (as root):

```
>brctl addbr bridge0  
>brctl addif bridge0 eth0  
>brctl addif bridge0 tap1  
>ifconfig tap1 0.0.0.0  
>ifconfig eth0 0.0.0.0  
>ifconfig bridge0 10.10.10.2 netmask 255.255.255.0 up
```

5. Start the VxWorks simulator:

```
>vxsim -d simnet -e 10.10.10.50
```

6. Check your routes and reset if necessary.

Note that when **vxsimnetd** is stopped, it removes the wrtap interface from the bridge. When **vxsimnetd** is restarted, re-run the following commands to install the tap interface into the bridge:

```
>brctl addif bridge0 tap1  
>ifconfig tap1 0.0.0.0
```

6.5 IPv6 Tutorial

This tutorial illustrates how to configure your host system and your target simulators to communicate using IPv6 protocol. For more information on IPv6, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide, Volume 1: Transport and Network Protocols*.

This tutorial describes how to:

- Enable IPv6 support on your host system.
- Configure the VxWorks simulator network daemon.
- Configure a VxWorks image for use as an IPv6-enabled simulator.
- Start your IPv6 VxWorks simulator network and test your connections.

6.5.1 Configure the Network

This section describes how to set up your host system and the VxWorks simulator network daemon for use with an IPv6 network.

Step 1: Configure Your Host System

In order to receive IPv6 packets from the VxWorks simulator subnet, you must configure your host system with IPv6 support.

On Windows hosts, configure IPv6 support by issuing the following command from a Windows command shell:

```
C:\> ipv6 install
```

On Linux hosts, issue the following command in a host shell:

```
$ modprobe ipv6
```

On Solaris hosts, IPv6 support is configured during setup. To confirm IPv6 support on your host, log in with root privileges and issue the following command in the host shell:

```
# ifconfig -a6
```

If IPv6 support is present the command will be successful. If the command is unsuccessful, see your host system documentation for information on enabling IPv6 support or consult your system administrator.

Step 2: Configure and Start the Network Daemon

This tutorial uses a network configuration that includes two subnets (**default** and **sub3**) that are configured with IPv4 addresses. The IPv4 addresses are used only to identify the subnets and to assign MAC addresses (see [5.5.2 Starting a Simulator Instance Without an IPv4 Address](#), p.71).



NOTE: Before launching **vxsimnetd**, be sure to kill all previously started VxWorks simulator instances and then kill the previously started network daemon.

You can configure the VxWorks simulator network daemon (**vxsimnetd**) using one of the following methods:

Start vxsimnetd Using a Static Configuration File

Follow the instructions as provided in [Step 1:Configure and Launch vxsimnetd](#), p.79 using the following file in place of the **vxsimTest.conf** file (save the file as **ipv6_tutorial_static.conf**):

```
SUBNET_START default {  
    SUBNET_ADDRESS = "192.168.200.0";  
    SUBNET_EXTERNAL = yes;  
    SUBNET_EXTPROMISC = yes;  
};  
SUBNET_START sub3 {  
    SUBNET_ADDRESS = "192.168.3.0";  
    SUBNET_EXTERNAL = no;  
};
```

Configure vxsimnetd Dynamically Using the Shell

You can also configure the VxWorks simulator network daemon dynamically. First, launch the **vxsimnetd** shell server as directed in [Step 1:Launch the vxsimnetd Shell Server](#), p.87. Then, complete the following steps:

1. Create and save the following file as **ipv6_tutorial_dynamic.conf**:

```
SUBNET_START sub3 {  
    SUBNET_ADDRESS = "192.168.3.0";  
    SUBNET_EXTERNAL = no;  
};
```

2. Now, connect to the **vxsimnetd** shell. From your host shell, type the following:

```
% telnet yourHostName 7777
```

where *yourHostName* is the name of your host machine.

3. In the **vxsimnetd** debug shell, source the new configuration file as follows:

```
vxsimnetd> source /myDir/ipv6_tutorial_dynamic.conf
```

```
Subnet <sub3> added.
```

4. Quit the **vxsimnetd** shell.

```
vxsimnetd> quit
```

6.5.2 Configuring VxWorks with IPv6 Components



NOTE: Before configuring your VxWorks image, make sure that your network stack is enabled with IPv6 support. For information on building your network stack with IPv6 support, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide, Volume 1: Transport and Network Protocols*.

This tutorial uses a VxWorks simulator configuration to demonstrate router advertisement and solicitation. The tutorial requires that you configure your VxWorks with the following components:

```
INCLUDE_IPD_CMD
INCLUDE_IPCOM_SYSVAR_CMD
INCLUDE_IPRADVD_CMD
INCLUDE_PING6
```

Set these components using the **vxprj** command-line utility using the following steps:

1. Create a project called **demo_router**. For example:

```
% vxprj -inet6 create bsp TOOL demo_router
```

where *bsp* is the BSP for your host system type (**smpc**, **linux**, or **solaris**) and *TOOL* is your desired toolchain (**diab** or **gnu**).



NOTE: Wind River VxWorks Platforms users must omit the **-inet6** option (or, when using Workbench, do *not* to check the **Use IPv6 enabled kernel libraries** option in the **Options** dialog of the **New VxWorks Image Project** wizard). Also, be sure that you have built your Platform with IPv6 support. For more information, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide, Volume 1: Transport and Network Protocols* and your Platform getting started guide.

2. Change to the **demo_router** directory:

```
% cd demo_router
```

3. Add the **INCLUDE_IPD_CMD**, **INCLUDE_IPCOM_SYSVAR_CMD**, **INCLUDE_IPRADVD_CMD**, and **INCLUDE_PING6** components:

```
% vxprj component add INCLUDE_IPD_CMD INCLUDE_IPCOM_SYSVAR_CMD
INCLUDE_IPRADVD_CMD INCLUDE_PING6
```



NOTE: You can also create your project and configure your VxWorks image from Workbench using the kernel configuration tool. For more information on configuring components using Workbench, see [Prepare Your VxWorks Image Using Workbench](#), p.81 or the *Wind River Workbench User's Guide*.

Build Your Projects

Now, build the **demo_router** project as follows:

```
% vxprj build
```

6.5.3 Testing the IPv6 Connection

This section describes how to test the IPv6 connections in your simulated network.

Start the VxWorks Simulator Instances

Before you can test the IPv6 connection, you must start your VxWorks simulator instances using the VxWorks images you created.

Step 1: Start the Simulator Instances on the Default Subnet

1. Create a startup script (**default_startup**) that specifies the prefix to advertise:

```
# switch interpreter
cmd
# add IPv6 address
ifconfig simnet0 inet6 add 2002:A01:201::787A:C0FF:FEA8:C801
# configure advertisement
radvdconfig simnet0 add test 2002:0a01:0201::/64 valid 600 preferred 200
L A
radvdconfig simnet0 enable test
# start advertising prefix
sysvar set -o ipnet.inet6.radvd.interfaces simnet0
ipd start ipnet_radvd
```

2. Start one advertiser VxWorks simulator instance on the default subnet. You can start the simulator instance from the VxWorks development shell using the script you just created as follows:

```
% vxsim -p 0 -d simnet -e 192.168.200.1 -f
installDir/demo_router/default/vxworks -s pathToStartupScript/default_startup
```

Where *pathToStartupScript* is the full path to your startup script location.

This starts a VxWorks simulator instance (**VxSim0**) with an advertiser configuration on the default subnet.

3. Start a solicitor VxWorks simulator instance on the default subnet as follows:

```
% vxsim -p 1 -d simnet -e 192.168.200.2 -f  
installDir/demo_router/default/vxworks
```

This starts a VxWorks simulator instance (**VxSim1**) with the solicitor configuration on the default subnet.

Step 2: Start the Simulator Instances on Subnet 3

1. Start one advertiser and one solicitor instance on subnet 3. First, create a startup script (**sub3_startup**) that specifies the prefix to advertise:

```
# switch interpreter  
cmd  
# add IPv6 address  
ifconfig simnet0 inet6 add 2002:A01:202::787A:C0FF:FEA8:C801  
# configure advertisement  
radvdconfig simnet0 add test 2002:0a01:0202::/64 valid 600 preferred 200  
L A  
radvdconfig simnet0 enable test  
# start advertising prefix  
sysvar set -o ipnet.inet6.radvd.interfaces simnet0  
ipd start ipnet_radvd
```

2. Start the advertiser VxWorks simulator instance. You can start the simulator instance from the VxWorks development shell using the script you just created:

```
% vxsim -p 2 -ni simnet0=192.168.3.1;simnet1=192.168.200.3 -f  
installDir/demo_router/default/vxworks -s pathToStartupScript/sub3_startup
```

Where *pathToStartupScript* is the full path to your startup script location.

This starts a VxWorks simulator instance (**VxSim2**) with the advertiser configuration on subnet 3.

3. Start the solicitor VxWorks simulator instance as follows:

```
% vxsim -p 3 -d simnet -e 192.168.3.2 -f  
installDir/demo_router/default/vxworks
```

This starts a VxWorks simulator instance (**VxSim3**) with the solicitor configuration on subnet 3.

Step 3: Check Your Connections

You can now check to see if the VxWorks simulator instances are correctly configured.



NOTE: You may need to wait 10-30 seconds for the network autoconfiguration to complete.

1. In the **VxSim0** console, type:

```
-> ifconfig "simnet0"
simnet0 Link type:Ethernet HWaddr 7a:7a:c0:a8:c8:01 Queue:none
inet 192.168.200.1 mask 255.255.255.0 broadcast 192.168.200.255
inet 224.0.0.1 mask 240.0.0.0
inet6 unicast 2002:A01:201::787A:C0FF:FEA8:C801 prefixlen 64
inet6 unicast FE80::787A:C0FF:FEA8:C801%simnet0 prefixlen 64
automatic
inet6 unicast 3FFE:1:2:3::4 prefixlen 64
inet6 unicast FE80::%simnet0 prefixlen 64 anycast
inet6 unicast 2002:A01:201:: prefixlen 64 anycast
inet6 unicast 3FFE:1:2:3:: prefixlen 64 anycast
inet6 multicast FF02::1:FF00:4%simnet0 prefixlen 16
inet6 multicast FF02::1:FF00:0%simnet0 prefixlen 16
inet6 multicast FF02::1%simnet0 prefixlen 16 automatic
inet6 multicast FF02::1:FFA8:C801%simnet0 prefixlen 16
inet6 multicast FF02::2%simnet0 prefixlen 16
UP RUNNING SIMPLEX BROADCAST MULTICAST
MTU:1500 metric:1 VR:0
RX packets:0 mcast:0 errors:0 dropped:0
TX packets:11 mcast:10 errors:0
collisions:0 unsupported proto:0
RX bytes:0 TX bytes:966

value = 0 = 0x0
```

2. In the **VxSim1** console, type:

```
-> ifconfig "simnet0"
simnet0 Link type:Ethernet HWaddr 7a:7a:c0:a8:c8:02 Queue:none
inet 192.168.200.2 mask 255.255.255.0 broadcast 192.168.200.255
inet 224.0.0.1 mask 240.0.0.0
inet6 unicast 2002:A01:201::787A:C0FF:FEA8:C802 prefixlen 64
autonomous
inet6 unicast FE80::787A:C0FF:FEA8:C802%simnet0 prefixlen 64
automatic
inet6 unicast 3FFE:1:2:3::4 prefixlen 64
inet6 unicast 2002:A01:201:: prefixlen 64 anycast
inet6 unicast FE80::%simnet0 prefixlen 64 anycast
inet6 unicast 3FFE:1:2:3:: prefixlen 64 anycast
inet6 multicast FF02::1:FF00:4%simnet0 prefixlen 16
inet6 multicast FF02::1:FF00:0%simnet0 prefixlen 16
inet6 multicast FF02::1%simnet0 prefixlen 16 automatic
inet6 multicast FF02::1:FFA8:C802%simnet0 prefixlen 16
UP RUNNING SIMPLEX BROADCAST MULTICAST
MTU:1500 metric:1 VR:0
RX packets:3 mcast:2 errors:0 dropped:0
TX packets:9 mcast:8 errors:0
collisions:0 unsupported proto:0
RX bytes:264 TX bytes:730
```

value = 0 = 0x0

3. In the VxSim2 console, type:

```
-> ifconfig
lo0      Link type:Local loopback Queue:none
         inet 127.0.0.1 mask 255.255.255.255
         inet 224.0.0.1 mask 240.0.0.0
         inet6 unicast FE80::1%lo0 prefixlen 64 automatic
         inet6 unicast ::1 prefixlen 128
         inet6 multicast FF02::1:FF00:1%lo0 prefixlen 16
         inet6 multicast FF01::1 prefixlen 16
         inet6 multicast FF02::1%lo0 prefixlen 16 automatic
         UP RUNNING LOOPBACK MULTICAST
         MTU:1500 metric:1 VR:0
         RX packets:9 mcast:0 errors:0 dropped:5
         TX packets:9 mcast:3 errors:0
         collisions:0 unsupported proto:0
         RX bytes:440 TX bytes:440

simnet0 Link type:Ethernet HWaddr 7a:7a:c0:a8:03:01 Queue:none
         inet 192.168.3.1 mask 255.255.255.0 broadcast 192.168.3.255
         inet 224.0.0.1 mask 240.0.0.0
         inet6 unicast 2002:A01:202::787A:C0FF:FEA8:C801 prefixlen 64
         inet6 unicast FE80::787A:C0FF:FEA8:301%simnet0 prefixlen 64
         automatic
         inet6 unicast FE80::%simnet0 prefixlen 64 anycast
         inet6 unicast 2002:A01:202:: prefixlen 64 anycast
         inet6 multicast FF02::1%simnet0 prefixlen 16 automatic
         inet6 multicast FF02::1:FFA8:301%simnet0 prefixlen 16
         inet6 multicast FF02::1:FFA8:C801%simnet0 prefixlen 16
         inet6 multicast FF02::1:FF00:0%simnet0 prefixlen 16
         inet6 multicast FF02::2%simnet0 prefixlen 16
         UP RUNNING SIMPLEX BROADCAST MULTICAST
         MTU:1500 metric:1 VR:0
         RX packets:2 mcast:1 errors:0 dropped:0
         TX packets:11 mcast:10 errors:0
         collisions:0 unsupported proto:0
         RX bytes:130 TX bytes:966

simnet1 Link type:Ethernet HWaddr 7a:7a:c0:a8:c8:03 Queue:none
         inet 192.168.200.3 mask 255.255.255.0 broadcast 192.168.200.255
         inet 224.0.0.1 mask 240.0.0.0
         inet6 unicast 2002:A01:201::787A:C0FF:FEA8:C803 prefixlen 64
         autonomous
         inet6 unicast FE80::787A:C0FF:FEA8:C803%simnet1 prefixlen 64
         automatic
         inet6 unicast 2002:A01:201:: prefixlen 64 anycast
         inet6 unicast FE80::%simnet1 prefixlen 64 anycast
         inet6 multicast FF02::1%simnet1 prefixlen 16 automatic
         inet6 multicast FF02::1:FFA8:C803%simnet1 prefixlen 16
         inet6 multicast FF02::1:FF00:0%simnet1 prefixlen 16
         UP RUNNING SIMPLEX BROADCAST MULTICAST
         MTU:1500 metric:1 VR:0
         RX packets:1 mcast:1 errors:0 dropped:0
```

```
TX packets:7 mcast:6 errors:0
collisions:0 unsupported proto:0
RX bytes:102 TX bytes:550
```

```
value = 0 = 0x0
```

4. In the VxSim3 console, type:

```
-> ifconfig "simnet0"
simnet0 Link type:Ethernet HWaddr 7a:7a:c0:a8:03:02 Queue:none
        inet 192.168.3.2 mask 255.255.255.0 broadcast 192.168.3.255
        inet 224.0.0.1 mask 240.0.0.0
        inet6 unicast 2002:A01:202::787A:C0FF:FEA8:302 prefixlen 64
autonomous
        inet6 unicast FE80::787A:C0FF:FEA8:302%simnet0 prefixlen 64
automatic
        inet6 unicast 3FFE:1:2:3::4 prefixlen 64
        inet6 unicast 2002:A01:202:: prefixlen 64 anycast
        inet6 unicast FE80::%simnet0 prefixlen 64 anycast
        inet6 unicast 3FFE:1:2:3:: prefixlen 64 anycast
        inet6 multicast FF02::1:FF00:4%simnet0 prefixlen 16
        inet6 multicast FF02::1:FF00:0%simnet0 prefixlen 16
        inet6 multicast FF02::1%simnet0 prefixlen 16 automatic
        inet6 multicast FF02::1:FFA8:302%simnet0 prefixlen 16
UP RUNNING SIMPLEX BROADCAST MULTICAST
MTU:1500 metric:1 VR:0
RX packets:3 mcast:3 errors:0 dropped:0
TX packets:9 mcast:8 errors:0
collisions:0 unsupported proto:0
RX bytes:306 TX bytes:730

value = 0 = 0x0
```

Step 4: Ping your VxSim Instances

You can now ping the VxWorks simulator instances on the same subnet. The default subnet (**default**) includes **VxSim0**, **VxSim1**, and the host system. Subnet 3 (**sub3**) includes **VxSim2** and **VxSim3**.

1. Ping VxSim1 from VxSim0. In the VxSim0 console, enter the following:

```
-> cmd
[vxWorks *]# ping6 2002:A01:201::787A:C0FF:FEA8:C802

Pinging 2002:A01:201::787A:C0FF:FEA8:C802
(2002:A01:201::787A:C0FF:FEA8:C802) with 64 bytes of data:
Reply from 2002:A01:201::787A:C0FF:FEA8:C802 bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C802 bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C802 bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C802 bytes=64 time=0ms hlim=64
```

```
--- 2002:A01:201::787A:C0FF:FEA8:C802 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 4066 ms
rtt min/avg/max = 0/0/0 ms
```

2. In the VxSim1 console, you can ping VxSim0 with the local address as follows:

```
-> cmd
[vxWorks *]# ping6 FE80::787A:C0FF:FEA8:C801%simnet0

Pinging FE80::787A:C0FF:FEA8:C801%simnet0
(FE80::787A:C0FF:FEA8:C801%simnet0) with 64 bytes of data:
Reply from FE80::787A:C0FF:FEA8:C801%simnet0 bytes=64 time=0ms hlim=64
Reply from FE80::787A:C0FF:FEA8:C801%simnet0 bytes=64 time=0ms hlim=64
Reply from FE80::787A:C0FF:FEA8:C801%simnet0 bytes=64 time=0ms hlim=64
Reply from FE80::787A:C0FF:FEA8:C801%simnet0 bytes=64 time=0ms hlim=64

--- FE80::787A:C0FF:FEA8:C801%simnet0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 4066 ms
rtt min/avg/max = 0/0/0 ms
```

3. From VxSim1, you can also ping VxSim0 at the automatically configured address as follows:

```
-> cmd
[vxWorks *]# ping6 2002:A01:201::787A:C0FF:FEA8:C801

Pinging 2002:A01:201::787A:C0FF:FEA8:C801
(2002:A01:201::787A:C0FF:FEA8:C801) with 64 bytes of data:
Reply from 2002:A01:201::787A:C0FF:FEA8:C801 bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C801 bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C801 bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C801 bytes=64 time=0ms hlim=64

--- 2002:A01:201::787A:C0FF:FEA8:C801 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 4066 ms
rtt min/avg/max = 0/0/0 ms
```

4. From VxSim0 or VxSim1, you can also ping the host:

```
-> cmd
[vxWorks *]# ping6 2002:A01:201::787A:C0FF:FEA8:C8FE

Pinging 2002:A01:201::787A:C0FF:FEA8:C8FE
(2002:A01:201::787A:C0FF:FEA8:C8FE) with 64 bytes of data:
Reply from 2002:A01:201::787A:C0FF:FEA8:C8FE bytes=64 time=16ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C8FE bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C8FE bytes=64 time=0ms hlim=64
Reply from 2002:A01:201::787A:C0FF:FEA8:C8FE bytes=64 time=0ms hlim=64

--- 2002:A01:201::787A:C0FF:FEA8:C8FE ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 4067 ms
rtt min/avg/max = 0/4/16 ms
```


In addition, you can set the routing such that you can ping between the default subnet and subnet 3 through **VxSim2**. For an example of how to set up the routing information, see the tutorials in [6.3 Basic Simulated Network with Multiple Simulators](#), p.78.

A

Accessing Host Resources

- [A.1 Introduction 107](#)
- [A.2 Accessing Host OS Routines 108](#)
- [A.3 Loading a Host-Based Application 108](#)
- [A.4 Host Application Interface \(vxsimapi\) 109](#)
- [A.5 Tutorials and Examples 111](#)

A.1 Introduction

The VxWorks simulator provides support to access the underlying host OS routines from a VxWorks application and to call host code stored in a dynamic-link library (DLL). That is, you can write a generic DLL (on any host) to control a hardware device connected to the host. The DLL can then be loaded by, and accessed through, the VxWorks simulator.

For information on the available host access routines, see the reference entry for **vxsimHostArchLib**. **vxsimHostArchLib** also provides a host library (**vxsimapi**) for VxWorks simulator host application development. For more information, see the reference entry for **vxsimapi**.

A.2 Accessing Host OS Routines

The `vxsimHostProcAddrGet()` routine allows you to retrieve the address of a host routine. For example, the following code retrieves the address of the underlying host OS `malloc()` routine:

```
/* Get underlying host OS malloc() address */
pHostMalloc = vxsimHostProcAddrGet ("malloc");

/* Allocate a buffer on host side of VxWorks Simulator */

lvl = intLock ();          /* lock interrupts */
pHostBuf = (*pHostMalloc) (0x1000);
intUnlock (lvl);          /* unlock interrupts */
```

You should observe the following guidelines when making a call to host code:

- When you call a host routine from VxWorks code, you must always lock interrupts before calling the routine. Failure to do so can result in unexpected VxWorks simulator behavior.
- When you call a host routine and the routine is system blocking, the routines will not only block the VxWorks task from which it was called, but will also block the entire VxWorks simulator.

To avoid blocking on a Windows simulator, create a specific thread that is responsible for calling the potentially blocking host code and set up a simple communication mechanism between VxWorks and the thread you have created. For an example of this, see [A.5.2 Controlling a Host Serial Device](#), p.113.

The method described for Windows simulators cannot be used on Linux or Solaris simulators because these simulators do not support multithreading. On these hosts, if the blocking system call is a device access, the solution is to configure the host device to generate an interrupt when data becomes available. For more information on using this method, refer to [A.4.2 Configuring a Host Device to Generate interrupts \(UNIX Only\)](#), p.109.

A.3 Loading a Host-Based Application

The `vxsimHostDllLoad()` routine provides the ability to load a DLL in the VxWorks simulator process. The exported symbols of the DLL can then be

retrieved using the **vxsimHostProcAddrGet()** routine described in [A.2 Accessing Host OS Routines](#), p.108.

A.4 Host Application Interface (vxsimapi)

The **vxsimapi** library provides the ability to extend VxWorks simulator capabilities with native OS code to perform operations that cannot be done directly using VxWorks code. For example, this facility can be used to add code to control peripherals connected to the host machine, to add code for graphic applications, or to add any functionality that requires host-specific code. For more information, see the reference entry for **vxsimapi**.

A.4.1 Defining User Exit Hooks

Applications often need to perform specific actions on exit. This includes items such as releasing resources, re-initializing peripherals, and other cleanup operations. The VxWorks simulator exit hook facility provides the **vxsimExitHookAdd()** routine. This routine gives you the ability to specify a routine to perform any necessary cleanup when the VxWorks simulator exits or reboots.

A.4.2 Configuring a Host Device to Generate interrupts (UNIX Only)

You can put the host file descriptors in asynchronous mode, such that VxWorks sends a **SIGPOLL** signal is sent to the VxWorks simulator when data becomes available. If a VxWorks task reads from a host device, the task normally requires a blocking read. Because Linux and Solaris simulators are mono-threaded, this action stops the VxWorks simulator process entirely until data is ready. As an alternative, you can open the file in non-blocking mode and then put the device into asynchronous mode. This causes a **SIGPOLL** signal to be sent whenever data becomes available. In this case, an input ISR reads the data, puts it in a buffer, and unblocks the waiting task.

To install an ISR that runs whenever data is ready on some underlying host device, you must first open the host device in non-blocking mode. Then, put the file descriptor in asynchronous mode using the **vxsimFdIntEnable()** routine. This

ensures that the host will send a **SIGPOLL** signal when data is available. On the target side, an interrupt service routine (ISR) is connected using **intConnect()**.

The following code example shows how to do this on a host serial port.

Host side (DLL code linked with the **vxsimapi** library):

```
/* open host device in non-blocking mode */
fd = open ("/dev/ttyb", O_NONBLOCK);
/* Enable interrupts on file descriptor */
vxsimFdIntEnable (fd);
```

Target side:

```
/* connect the interrupt service routine */
intConnect (FD_TO_IVEC (fd), ISRfunc, 0);
```

Interrupts can also be disabled using **vxsimFdIntDisable()**, and the ISR can be disconnected using **intDisconnect()**. For example:

Host side (DLL code linked with the **vxsimapi** library):

```
/* Disable interrupts on file descriptor */
vxsimFdIntDisable (fd);
```

Target side:

```
/* disconnect the interrupt service routine from file descriptor */
intDisconnect (FD_TO_IVEC (fd), ISRfunc, 0);
```

A.4.3 Simulating interrupts From a User Application (Windows Only)

The **vxsimIntRaise()** routine provides a host side application with the ability to notify VxWorks of a given event, allowing VxWorks to take the appropriate action. For example, if you have an application collecting data from a device, you can raise an interrupt to VxWorks when data has been read from the device. On the target side of the application, an ISR can be connected to the interrupt vector using **intConnect()**. Now, each time **vxsimIntRaise()** is called, the ISR is called to handle the read data.

When an interrupt needs to be acknowledged, the **vxsimIntAckRtnAdd()** routine can be used to connect an acknowledgement routine for a given interrupt vector. This routine is called immediately after the interrupt handling.

A range of interrupt vectors are available on Windows simulators. This range is defined in the **config.h** file of the **simpc** BSP:

USER_INT_RANGE_BASE	User interrupts range base
USER_INT_RANGE_END	User interrupts range end

The routines **vxsimIntToMsg()** and **vxsimMsgToInt()** allow you to convert an interrupt vector number to a Windows message number, and conversely, allow you to convert a message number to a vector number.

The **vxsimWindowsHandleGet()** routine can be used with the VxWorks simulator to get a windows handle for sending messages.

For more information on Windows simulator interrupt assignments, refer to [Table 3-4](#) in [3.5.5 Interrupts](#), p.26.

A

A.5 Tutorials and Examples

The following sections provide simple tutorials and examples illustrating host resource accessing.

A.5.1 Running Tcl on the VxWorks Simulator

This section provides a simple tutorial that illustrates how to load a standard Tcl DLL on the VxWorks simulator, and start a Tcl interpreter.

Code Sample

The following code sample can be built as a downloadable kernel module for all simulator types.

```
#include "vxWorks.h"
#include "vxsimHostLib.h"
```

```
#if (CPU==SIMNT)
#define TCL_DLL "tcl84.dll"      /* Windows Tcl Dll name */
#else
#define TCL_DLL "libtcl8.4.so"  /* Unix Tcl Dll name */
#endif

BOOL tclLoaded = FALSE;

STATUS tclStart (void)
{
    /* Function pointers for Tcl Dll routines */

    FUNCPTR    pTcl_CreateInterp;
    FUNCPTR    pTcl_Init;
    FUNCPTR    pTcl_Eval;
    FUNCPTR    pTcl_GetStringResult;
    FUNCPTR    pTcl_DeleteInterp;

    char        tclCommand[400];    /* buffer for Tcl command */
    int          evalResult;        /* Tcl command evaluation result */
    int          lvl;               /* interrupt lock level */
    void *       pInterp;           /* Tcl interpreter Id */

    /* load Tcl Dll */

    if (tclLoaded == FALSE)
    {
        if (vxsimHostDllLoad (TCL_DLL) != OK)
        {
            printf ("Error: Failed to load %s\n", TCL_DLL);
            return (ERROR);
        }

        tclLoaded = TRUE;
    }

    /* retrieve some Tcl routine address from the loaded Dll */

    pTcl_CreateInterp    = vxsimHostProcAddrGet ("Tcl_CreateInterp");
    pTcl_Init             = vxsimHostProcAddrGet ("Tcl_Init");
    pTcl_Eval             = vxsimHostProcAddrGet ("Tcl_Eval");
    pTcl_GetStringResult = vxsimHostProcAddrGet ("Tcl_GetStringResult");
    pTcl_DeleteInterp    = vxsimHostProcAddrGet ("Tcl_DeleteInterp");

    /* Create and Initialize Tcl interpreter */

    lvl = intLock ();                /* lock interrupts */
    pInterp = (void *)(*pTcl_CreateInterp) (); /* Create interpreter */
    (*pTcl_Init) (pInterp);          /* Initialize interpreter */
    intUnlock (lvl);                 /* unlock interrupts */

    printf ("Tcl Ready (Type CTRL+D to exit interpreter)\n\ntcl> ");

    while (gets (tclCommand) != NULL)
    {
        lvl = intLock ();            /* lock interrupts */

```



```
evalResult = (*pTcl_Eval) (pInterp, tclCommand);

if (evalResult != 0)
{
    printf ("Tcl Error: ");
}

if (strlen ((*pTcl_GetStringResult)(pInterp)) != 0)
    printf ("%s\n", (*pTcl_GetStringResult)(pInterp));

intUnlock (lvl);          /* unlock interrupts */

printf ("tcl> ");
}

/* Delete Tcl interpreter */

lvl = intLock ();        /* lock interrupts */
(*pTcl_DeleteInterp) (pInterp);
intUnlock (lvl);         /* unlock interrupts */

return (OK);
}
```

A

Running The Code

The sample code can be executed directly from the VxWorks kernel shell or using the host shell. A sample host shell session is as follows:

```
-> ld < tclInterp.o
value = 1634769168 = 0x61709910
-> tclStart
Loading libtcl8.4.so ... succeeded.
Tcl Ready (Type CTRL+D to exit interpreter)

tcl> glob *
tclInterp.c tclInterp.o hello.tcl
tcl> source hello.tcl
Hello !
tcl> ^Dvalue = 0 = 0x0
->
```

A.5.2 Controlling a Host Serial Device

Controlling a host serial device is a more complex application that allows you to control a host serial device from the VxWorks simulator. For an example of this application type, see the reference entry for **commSio** (Windows simulators) or

ttySio (Linux or Solaris simulators). The examples provided for **commSio** and **ttySio** exercise most of the features described in this chapter.

Index

A

- accessing
 - host OS routines 108
 - the VxWorks Simulator from a remote host 15
- application compatibility 2, 45
- assigning a processor number 14
- AUX_CLK_RATE_MAX 39
- AUX_CLK_RATE_MIN 39
- auxiliary clock 39
- auxiliary clock interrupts 26, 28

B

- b 7, 9
- backplane 7, 9
- boot parameters 9, 34
- BOOT_NO_AUTOBOOT 15
- bootChange() 9, 12, 34
- Br 24
- bspname.h 18
- BSPs 8, 18
 - linux 8
 - Makefile 18
 - simpc 8
 - solaris 8
- Bt 24
- building a VxWorks image 8, 43, 80

- with vxprj 8
- with Workbench 8
- building applications 19
- byte order 25

C

- C++ modules 19
- commSio 113
- compiler options 19
 - g 21
 - O 21
 - O0 21
 - Xno-optimized-debug 21
 - XO 21
- config.h 18, 34
- configuring
 - a simulated subnet 70
 - IPv6 support on your host system 96
 - multiple external subnets 65
 - multiple network interfaces 14
- connection timeout 24
- console
 - SMP 44
 - uniprocessor 13
- CPU 19

D

- d 9
- debugging 21
- DEFAULT_ACCESSMODE 61
- DEFAULT_ERATE 62
- DEFAULT_EXTERNAL 61
- DEFAULT_EXTPROMISC 62
- DEFAULT_GARBAGE 61
- DEFAULT_GID 61
- DEFAULT_MACPREFIX 61
- DEFAULT_TIMEOUT 62
- DEFAULT_UID 61
- development limitations 3, 45
- device 9
- devs() 38
- diab 19
- DLL 107
- dynamic-link library
 - see DLL

E

- e 9, 13
- ELF object module format 20
- emacs 57
- ethernet 9
- exit hook facility 109
- exiting the VxWorks Simulator 14
- exitOnError 9, 15

F

- f 9, 12, 56
- file 9, 56
- file system support 3, 23
 - pass-through file system 23, 36
 - virtual disk 24, 37
- flags 9
- floating-point
 - routines 25
 - support 25

- force 56

G

- g 10
- gateway 10
- gnu 19

H

- h 10
- hardware breakpoint support 25
- hardware simulation 36
- help 10
- hn 10, 37
- host application interface 109
- host connection driver 66
 - Linux 68
 - Solaris 67
 - Windows 66
- host system requirements 6
- HOST_SIO_PORT_NUMBER 40
- hostinet 10
- hostname 10, 37
- hostSio 40

I

- ifconfig() 72
- INCLUDE_BOOT_LINE_INIT 7
- INCLUDE_HOST_SIO 40
- INCLUDE_IPCOM_SYSVAR_CMD 98
- INCLUDE_IPD_CMD 98
- INCLUDE_IPRADVD_CMD 98
- INCLUDE_KERNEL 30
- INCLUDE_NET_BOOT_CONFIG 72
- INCLUDE_PASSFS 36
- INCLUDE_PING 80
- INCLUDE_PING6 98
- INCLUDE_ROUTECD 80
- INCLUDE_SM_COMMON 7, 41

INCLUDE_SM_NET 7, 41
 INCLUDE_SM_OBJ 7, 41
 INCLUDE_SYS_TIMESTAMP 40
 INCLUDE_TIMESTAMP 40
 INCLUDE_VIRTUAL_DISK 38
 intConnect() 27, 29, 110
 intDisconnect() 110
 interrupt assignments
 Windows simulator 28
 interrupt simulation 26
 host signals 26
 on Linux and Solaris 26
 on Windows 28
 Windows messages 28
 IPv6
 configuring IPv6 support on your host system 96
 support 48
 tutorial 95
 ISR_STACK_SIZE 30

K

-k 10
 -kill 10

L

-l 10
 linux 8
 loading a VxWorks image 9
 LOCAL_MEM_LOCAL_ADRS 34, 35
 LOCAL_MEM_SIZE 34, 35
 -logfile 10

M

MACH_EXTRA 18
 macros
 AUX_CLK_RATE_MAX 39
 AUX_CLK_RATE_MIN 39

HOST_SIO_PORT_NUMBER 40
 INCLUDE_SM_NET 7
 LOCAL_MEM_LOCAL_ADRS 34, 35
 LOCAL_MEM_SIZE 34, 35
 MACH_EXTRA 18
 NV_RAM_SIZE 38
 SYS_CLK_RATE_MAX 39
 SYS_CLK_RATE_MIN 39
 Makefile 8, 18
 memory
 configuration 34
 layout 29
 memory management support
 running without MMU 23
 memory management unit 3, 22
 limitations 23
 MMU simulation 22
 page size 22
 translation Model 22
 memory protection level 36
 -memsize 11, 35
 migrating applications 45
 MMU
 see memory management unit
 MMU simulation 22

N

-n- nvram 10
 -netif 10
 network daemon
 configuration file parameters 60
 configuration files 60
 network simulation 48
 networking address space 30
 -ni 10, 14
 non-volatile RAM support 38
 NV_RAM_SIZE 38
 -nvram 39

O

-o 10
-other 10

P

-p 10, 14
packet loss 59
packet sniffers 48, 59, 66
passDev 9
passFS
 see pass-through file system
pass-through file system 23, 36
-password 10
physical memory address space 35
-pl 10
-processor 10
-prot-level 10, 36
-pw 10

R

remote host 15
router configuration 14
routines
 bootChange() 9, 12, 34
 devs() 38
 ifconfig() 72
 intConnect() 110
 intDisconnect() 110
 sysAuxClkRateSet() 39
 sysBusToLocalAdrs() 18
 sysClkRateSet() 39
 sysMemTop() 30
 sysNvRamGet() 39
 sysNvRamSet() 39
 virtualDiskClose() 38
 virtualDiskCreate() 38
 virtualDiskInit() 38
 vxsimExitHookAdd() 109
 vxsimFdIntDisable() 110

vxsimFdIntEnable() 109
vxsimHostDllLoad() 108
vxsimHostProcAddrGet() 108, 109
vxsimIntAckRtnAdd() 110
vxsimIntRaise() 110
vxsimIntToMsg() 111
vxsimMsgToInt() 111
vxsimWindowsHandleGet() 111

RTP support 3, 23

S

-s 11, 56, 57
serial I/O driver 40
serial line support 40
shared memory
 address space 30
 END driver 7
 pool size 7
 size 7
shared memory network 40
-shell 56, 57
-shellport 56
-shellserver 56, 57
SIGALRM 26
SIGPOLL 26, 109
SIGUSR1 27
SIGUSR2 27
SIMLINUX 20
SIMNT 19
simpc 8
SIMPENTIUM 20
SIMSPARCSOLARIS 20
simulated hardware support 3
simulating packet loss 59
SIO driver 39
-size 11, 35
SM_MEM_SIZE 7
SM_OBJ_MEM_SIZE 7
smEnd 7
SMP support 4
 creating an image 43
 system requirements 6
solaris 8

- sp 56
- specifying the passFS device name 37
- standard I/O 18, 39
- starting
 - a standalone VxWorks Simulator instance 12
 - simulator instances for use with network services 13
 - the VxWorks Simulator from Workbench 14
- startup 11
- SUBNET_ACCESSMODE 62
- SUBNET_ADDRESS 63
- SUBNET_BROADCAST 64
- SUBNET_ERATE 64
- SUBNET_EXTCONNNAME 65
- SUBNET_EXTDEVNUM 63, 65
- SUBNET_EXTERNAL 63
- SUBNET_EXTPROMISC 63
- SUBNET_GID 62
- SUBNET_MACPREFIX 62
- SUBNET_MASK 63
- SUBNET_MAXBUFFERS 64
- SUBNET_MAXNODES 64
- SUBNET_MTU 65
- SUBNET_MULTICAST 64
- SUBNET_RECQLEN 64
- SUBNET_SHMKEY 64
- SUBNET_TIMEOUT 65
- SUBNET_UID 62
- supported VxWorks features 2, 6
- sv 56, 57
- SYS_CLK_RATE_MAX 39
- SYS_CLK_RATE_MIN 39
- sysAuxClkRateSet() 39
- sysBootLine 18
- sysBusToLocalAdrs() 18
- sysClkRateSet() 39
- sysLib.c 18
- sysMemTop() 30
- sysNvRamGet() 39
- sysNvRamSet() 39
- system clock 39

T

- targetname 11
- timers 39
- timestamp driver 39
- tmpdir 11
- tn 11
- TOOL 19
- ttySio 114
- tun module 68
- tutorial
 - IPv6 95
 - running VxSim on the local network 90
 - simple simulated network 74
 - simulated network with multiple simulators 78
- tyLib.c 18

U

- unit 11
- UNIX disk driver library 38
- unixDrv 38
- unixSio.c 18, 39
- user 11
- USER_INT_RANGE_BASE 111
- USER_INT_RANGE_END 111
- username 11

V

- v 11
- vaddr 11, 35
- version 11
- vi 57
- virtual disk support 3, 24, 37
- virtual memory address space 35
- virtualDiskClose() 38
- virtualDiskCreate() 38
- virtualDiskInit() 38
- VM_PAGE_SIZE 23
- VM_STATE_CACHEABLE 22

VM_STATE_CACHEABLE_NOT 22
VM_STATE_VALID 22
VM_STATE_VALID_NOT 22
VM_STATE_WRITEABLE 22
VM_STATE_WRITEABLE_NOT 22
VMEbus 40
-vsize 11, 35
VxMP 7
vxprj 81
vxsim 9, 12
 configuration options 9
vxsimapi 107, 109
vxsimExitHookAdd() 109
vxsimFdIntDisable() 110
vxsimFdIntEnable() 109
vxsimHostArchLib 107
vxsimHostDllLoad() 108
vxsimHostProcAddrGet() 108, 109
vxsimIntAckRtnAdd() 110
vxsimIntRaise() 110
vxsimIntToMsg() 111
vxsimMsgToInt() 111
vxsimnetd 50
 command line options 56
 debug shell 57
 removing the Windows service 57
 shell commands 57
 ? 59
 delete 59
 erate 59
 extpromisc 59
 help 59
 mode emacs 60
 mode vi 60
 node 58
 packet 58
 quit 59
 source 59
 subnet 58
 timeout 59
 starting as a root service 53
 starting as a Windows service 52
vxsimnetds_inst.exe 52
vxsimWindowsHandleGet() 111
vxWorks 8

VxWorks components
 INCLUDE_BOOT_LINE_INIT 7
 INCLUDE_HOST_SIO 40
 INCLUDE_IPCOM_SYSVAR_CMD 98
 INCLUDE_IPD_CMD 98
 INCLUDE_IPRADVD_CMD 98
 INCLUDE_NET_BOOT_CONFIG 72
 INCLUDE_PASSFS 36
 INCLUDE_PING 80
 INCLUDE_PING6 98
 INCLUDE_ROUTECD 80
 INCLUDE_SM_COMMON 7, 41
 INCLUDE_SM_NET 41
 INCLUDE_SM_OBJ 7, 41
 INCLUDE_SYS_TIMESTAMP 40
 INCLUDE_TIMESTAMP 40
 INCLUDE_VIRTUAL_DISK 38
VxWorks image
 vxWorks 8
 vxWorks.st 8
VxWorks image projects 8
VxWorks simulator network daemon
 see vxsimnetd
vxWorks.st 8

W

watchdog timer facilities 26, 28
WDB
 back end 24
WDB_POOL_SIZE 30
Wind River System Viewer 39
winSio.c 18, 39
WRTAP 65, 66

X

xterm 9