<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Overview</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Supported Features and Compatibility</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 VxWorks Feature Support</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2 Simulated Hardware Support</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Limitations</td>
<td>3</td>
</tr>
<tr>
<td><strong>2 Getting Started</strong></td>
<td></td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2.2 System Requirements</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Configuring and Building a VxWorks Image</td>
<td>6</td>
</tr>
<tr>
<td>2.3.1 Default Configuration Components</td>
<td>6</td>
</tr>
<tr>
<td>2.3.2 Optional Components</td>
<td>7</td>
</tr>
<tr>
<td>2.3.3 Building Your VxWorks Image</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Launching the VxWorks Simulator</td>
<td>8</td>
</tr>
<tr>
<td>2.4.1 vxsim Configuration Options</td>
<td>9</td>
</tr>
<tr>
<td>Starting a Standalone VxWorks Simulator Instance</td>
<td>11</td>
</tr>
</tbody>
</table>
Starting Instances to Run on a Simulated Subnet ...................... 12
2.4.2 Launching the VxWorks Simulator From Workbench .......... 13
2.4.3 Rebooting and Exiting the VxWorks Simulator .................. 13
2.4.4 Accessing the VxWorks Simulator from a Remote Host ........ 14
2.5 Where to Go Next ........................................................................ 15

3 Introduction to the VxWorks Simulator Environment .................. 17
3.1 Introduction .................................................................................... 17
3.2 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
    MMU Simulation ........................................................................... 22
    MMU Translation Model ............................................................. 22
    MMU Page Size ........................................................................... 23
    MMU 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) ............................................. 24
    Virtual Disk Support ................................................................. 24
  3.4.4 WDB Back End ..................................................................... 24
  3.4.5 Connection Timeout ............................................................... 24
3.5 Architecture Considerations ......................................................... 25
  3.5.1 Byte Order ............................................................................ 25
  3.5.2 Hardware Breakpoint .............................................................. 25
  3.5.3 Floating-Point Support ............................................................. 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.4</td>
<td>ISR Stack Protection</td>
<td>26</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Interrupts</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>Solaris and Linux Systems</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>Windows Systems</td>
<td>28</td>
</tr>
<tr>
<td>3.5.6</td>
<td>Memory Layout</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>Using the VxWorks Simulator</td>
<td>33</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>VxWorks Simulator Configuration</td>
<td>33</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Boot Parameters</td>
<td>34</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Memory Configuration</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Physical Memory Address Space</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Virtual Memory Address Space</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Memory Protection Level</td>
<td>35</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Miscellaneous Configuration</td>
<td>36</td>
</tr>
<tr>
<td>4.3</td>
<td>Hardware Simulation</td>
<td>36</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Pass-Through File System (passFS)</td>
<td>36</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Virtual Disk Support</td>
<td>37</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Non-Volatile RAM Support</td>
<td>38</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Standard I/O</td>
<td>39</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Timers</td>
<td>39</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Timestamp Driver</td>
<td>39</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Serial Line Support</td>
<td>40</td>
</tr>
<tr>
<td>4.3.8</td>
<td>Shared Memory Network</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Migrating Applications to a Hardware-Based System</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>Networking with the VxWorks Simulator</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>45</td>
</tr>
</tbody>
</table>
5.2 Building Network Simulations ................................................................. 46

5.3 Setting Up the Network Daemon ............................................................. 48
  5.3.1 Starting the Network Daemon ........................................................... 48
    Starting the Network Daemon as a Service ............................................ 48
    Starting the Network Daemon From the Command Line ..................... 52
  5.3.2 Network Daemon Debug Shell ......................................................... 53
  5.3.3 Network Daemon Configuration File ................................................. 56
    Configuring Multiple External Subnets ................................................. 61

5.4 Installing the Host Connection Driver ................................................. 62
  5.4.1 Managing the WRTAP Driver on Windows Hosts .............................. 64

5.5 Configuring a Simulated Subnet .......................................................... 66
  Starting a Simulator Instance With Multiple Network Interfaces .......... 66
  Starting a Simulator Instance Without an IPv4 Address ...................... 66

6 Networking Tutorials ........................................................................... 69

6.1 Introduction .......................................................................................... 69

6.2 Simple Simulated Network ................................................................. 69
  6.2.1 Set Up the Network Daemon ........................................................... 70
  6.2.2 Start a VxWorks Simulator Instance ............................................... 71
  6.2.3 Test the Simulated Network ........................................................... 73

6.3 Basic Simulated Network with Multiple Simulators ......................... 73
  6.3.1 Static Configuration ....................................................................... 74
    Configure and Launch vxsimnetd ......................................................... 74
    Prepare a VxWorks Image for Use with the Simulated Network ......... 75
    Launch the VxWorks Simulator Instances ......................................... 78
    Run the Ping Application ................................................................. 81
  6.3.2 Dynamic Configuration Using the vxsimnetd Shell ....................... 82
    Launch the vxsimnetd Shell Server ..................................................... 82
    Configure vxsimnetd Dynamically Using the Shell ............................ 82
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 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 the Wind River Workbench or 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.

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

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 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.
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
2.5 Where to Go Next 15

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 Wind River Workbench Command-Line 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.

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
- C++ 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*.

### 2.3.2 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.7.

**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

```bash
> vxsim -b backplaneAddress
```

or:

```bash
> 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 the Wind River Workbench New VxWorks Image Project wizard. To open the wizard, select File > New > VxWorks Image Project. You should base your project on the appropriate VxWorks simulator BSP for your host type. Table 2-1 lists the available simulator BSPs.

For more information on building VxWorks image projects, see the Wind River Workbench User’s Guide: 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 Wind River Workbench Command Line User’s Guide: Building Applications and Libraries.

### 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. For more information, see 4.2.1 Boot Parameters, p.34.

<table>
<thead>
<tr>
<th>VxWorks Simulator Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-backplane inetOnBackplane</code></td>
<td>Backplane address of the target system.</td>
</tr>
<tr>
<td><code>-d bootDevice</code></td>
<td>Type of device to boot from. Default: <code>passDev</code></td>
</tr>
<tr>
<td><code>-file fileName</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>-flags flags</code></td>
<td>Configuration flags. Default: 0.</td>
</tr>
<tr>
<td><code>-gateway gatewayInet</code></td>
<td>Internet address of the gateway.</td>
</tr>
<tr>
<td><code>-help</code></td>
<td>Print a help message listing the command-line options.</td>
</tr>
<tr>
<td>VxWorks Simulator Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-hostinet hostNet</td>
<td>Host internet address.</td>
</tr>
<tr>
<td>-h hostNet</td>
<td></td>
</tr>
<tr>
<td>-hostname hostName</td>
<td>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.</td>
</tr>
<tr>
<td>-hn hostName</td>
<td></td>
</tr>
<tr>
<td>-kill processNumber</td>
<td>Kills the VxWorks simulator referenced by processNumber.</td>
</tr>
<tr>
<td>-k processNumber</td>
<td></td>
</tr>
<tr>
<td>-logfile log file</td>
<td>Enables output logging (a Windows-only option). Default: VxSIMn.log.</td>
</tr>
<tr>
<td>-l log file</td>
<td></td>
</tr>
<tr>
<td>-n -nvram</td>
<td>VxWorks simulator non-volatile RAM file.</td>
</tr>
<tr>
<td>-netif additionalInterface</td>
<td>VxWorks simulator additional network interfaces.</td>
</tr>
<tr>
<td>-ni additionalInterface</td>
<td></td>
</tr>
<tr>
<td>-other other</td>
<td>Unused, available for applications.</td>
</tr>
<tr>
<td>-o other</td>
<td></td>
</tr>
<tr>
<td>-password ftpPassword</td>
<td>User password (for FTP only).</td>
</tr>
<tr>
<td>-pw ftpPassword</td>
<td></td>
</tr>
<tr>
<td>-processor number</td>
<td>Sets the processor number, which is effectively an identifier for this simulator instance. Default value: 0.</td>
</tr>
<tr>
<td>-p number</td>
<td></td>
</tr>
<tr>
<td>-prot-level protectionLevel</td>
<td>Sets the VxWorks simulator memory protection level. Values included min (0), max (1), or integer. Default value is max (1).</td>
</tr>
<tr>
<td>-pl protectionLevel</td>
<td></td>
</tr>
<tr>
<td>-size memorySize</td>
<td>VxWorks simulator memory size in bytes. Default: 32 MB.</td>
</tr>
<tr>
<td>-memsize memorySize</td>
<td></td>
</tr>
<tr>
<td>-startup script</td>
<td>Startup script for the target shell.</td>
</tr>
<tr>
<td>-s script</td>
<td></td>
</tr>
<tr>
<td>-targetname targetName</td>
<td>Name of the target. Default: vxTarget.</td>
</tr>
<tr>
<td>-tn targetName</td>
<td></td>
</tr>
</tbody>
</table>

Table 2-2 Command-Line Options for the VxWorks Simulator (cont’d)
When launching your VxWorks simulator from Workbench (Target > New Connection > Wind River VxWorks 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.

Starting a Standalone VxWorks Simulator Instance

To start a VxWorks simulator instance from the command line, use the `vxsim` command. 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).
To start a VxWorks simulator using the default configuration values, use `vxsim` as follows:

```
> vxsim -f pathToVxWorksImage
```

If you run `vxsim` from the directory containing the VxWorks image 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 Wind River Workbench Command-Line User’s Guide: Creating a Development Shell with `wrenv`.

### Starting Instances to Run on a Simulated Subnet

If you intend to use network services, do not start a VxWorks simulator instance until after the VxWorks simulator network daemon is started and has completed its initialization (for more information, see 5.3 Setting Up the Network Daemon, p.48). 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:

```plaintext
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 *Starting a Simulator Instance With Multiple Network Interfaces*, p.66.

For more information on setting up a simulated network, see *5. Networking with the VxWorks Simulator*.

### 2.4.2 Launching the VxWorks Simulator From Workbench

The VxWorks simulator can be launched from the 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 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*.

### 2.4.3 Rebooting and Exiting the VxWorks Simulator

As with other targets, the VxWorks simulator can be rebooted by typing `Ctrl+X` in the VxWorks simulator console window. On Windows systems, the VxWorks simulator can be exited 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 target 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.
2.4.4 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 `simnet` 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 should now be able to connect to the VxWorks simulator instance from the remote host.
2.5 Where to Go Next

The remainder of this document provides:

- detailed information on building applications for the VxWorks simulator as well as a detailed information regarding the VxWorks simulator target architecture (see 3. Introduction to the VxWorks Simulator Environment).

- additional configuration information, information on hardware simulation, and basic guidelines for migrating a VxWorks simulator application to a target hardware environment (see 4. Using the VxWorks Simulator).

- information on setting up a simulated subnet for developing networking applications, as well as tutorials for setting up a simulated subnet (see 5. Networking with the VxWorks Simulator and 6. Networking Tutorials).

For information on getting started with VxWorks and Workbench, see your Platform getting started. For general information and tutorials for building and running applications, see the Wind River Workbench User’s Guide.
3 Introduction to the VxWorks Simulator Environment

3.1 Introduction 17
3.2 VxWorks Simulator BSP 18
3.3 Building Applications 19
3.4 Interface Variations 21
3.5 Architecture Considerations 25

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.48, and variations in the VxWorks simulator BSP and architecture behavior, which are discussed in the following sections.
3.2 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 functions: sysModel(), sysHwinit(), and sysClkConnect() through sysNvRamSet(). However, because there is no bus, sysBusToLocalAdrs() and related functions 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 process’s standard input and output. Nevertheless, it supports all 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 Makefile is the standard version for VxWorks BSPs. 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

The recommended way to build VxWorks simulator modules is to use the Wind River Workbench. However, if you need 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

<table>
<thead>
<tr>
<th>CPU Definition</th>
<th>Host (Run Type)</th>
<th>TOOL</th>
<th>Compiler Command-Line Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMNT</td>
<td>Windows (kernel)</td>
<td><strong>diab</strong></td>
<td><code>-tx86lc:vxworks61 -DCPU=SIMNT</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>gnu</strong></td>
<td><code>-mcpu=1486 -march=i486 -DCPU=SIMNT</code></td>
</tr>
</tbody>
</table>
For example, to specify CPU for an application that will be 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 compile for the SPARC architecture.

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:vxworks61 -DCPU=SIMNT -I installDir/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 -I installDir/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, subroutines, 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.

MMU Simulation

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

MMU Translation Model

All VxWorks simulator implementations share a common programming model for mapping memory pages. The physical memory address space is described by the data structure `sysPhysMemDesc[]`, defined in `sysLib.c`. 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.
3 Introduction to the VxWorks Simulator Environment

3.4 Interface Variations

MMU Page Size

The page size used by the VxWorks simulator is limited by the host operating system routines used to map and unmap pages. On Solaris and Linux-based simulators, this page size is 8 KB. On Windows simulators, this page size is 64 KB.

MMU limitations

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

Running the VxWorks Simulator Without MMU Support

The VxWorks simulator can be configured 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 the 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: Defining a New Target Server Connection.
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 these mathematical routines. All ANSI floating-point routines have been optimized using libraries from U. S. Software.

<table>
<thead>
<tr>
<th>acos()</th>
<th>asin()</th>
<th>atan()</th>
<th>atan2()</th>
</tr>
</thead>
<tbody>
<tr>
<td>cos()</td>
<td>cosh()</td>
<td>exp()</td>
<td>fabs()</td>
</tr>
<tr>
<td>floor()</td>
<td>fmod()</td>
<td>log()</td>
<td>log10()</td>
</tr>
<tr>
<td>pow()</td>
<td>sin()</td>
<td>sinh()</td>
<td>sqrt()</td>
</tr>
<tr>
<td>tan()</td>
<td>tanh()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The following floating-point routines are *not* available on the VxWorks simulator:

- `cbrt()`
- `ciel()`
- `infinity()`
- `irint()`
- `iround()`
- `log2()`
- `round()`
- `sincos()`
- `trunc()`
- `cbrtf()`
- `infinityf()`
- `irintf()`
- `iroundf()`
- `log2f()`
- `roundf()`
- `sincosf()`
- `truncf()`

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

- `acosf()`
- `asinf()`
- `atanf()`
- `atan2f()`
- `cielf()`
- `cosf()`
- `expf()`
- `fabsf()`
- `floorf()`
- `fmodf()`
- `logf()`
- `log10f()`
- `powf()`
- `sinf()`
- `sinhf()`
- `sqrtf()`
- `tanf()`
- `tanhf()`

### 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()` functions.

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

<table>
<thead>
<tr>
<th>Interrupt Vectors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Host signal 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>User signal 1</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>User signal 2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Host signal 32</td>
</tr>
<tr>
<td>33</td>
<td>Interrupt vector for host file descriptor 1 (SIGPOLL)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>288</td>
<td>Interrupt vector for host file descriptor 256 (SIGPOLL)</td>
</tr>
</tbody>
</table>

Pseudo-drivers can be created to use these interrupts. Interrupt code must be connected 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:

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

On Linux:

```c
> 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.

⚠️ **CAUTION:** 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)

<table>
<thead>
<tr>
<th>Signal</th>
<th>Solaris Value</th>
<th>Linux Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGUSR1</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>17</td>
<td>12</td>
</tr>
</tbody>
</table>

**Windows Systems**

On Windows simulators, hardware interrupt simulation is performed using Windows messages. 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()` functions.

**Table 3-4** shows how the Windows simulator interrupt vector table is set up.
Pseudo-drivers can be created to use these interrupts. Interrupt code must be connected with 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 2 ticks per second using `sysAuxClkRateSet()` so 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.

---

Table 3-4  Interrupt Assignments (Windows Simulators)

<table>
<thead>
<tr>
<th>Interrupt Vectors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>system clock interrupt</td>
</tr>
<tr>
<td>0x0001</td>
<td>auxiliary clock interrupt</td>
</tr>
<tr>
<td>0x0002</td>
<td>timestamp rollover interrupt</td>
</tr>
<tr>
<td>0x0003</td>
<td>back end pipe interrupt</td>
</tr>
<tr>
<td>0x0004</td>
<td>SIO driver interrupt</td>
</tr>
<tr>
<td>0x0005</td>
<td>bus interrupt</td>
</tr>
<tr>
<td>0x0006-0x00ef</td>
<td>reserved for internal use</td>
</tr>
<tr>
<td>0x00f0-0x00ff</td>
<td>Wind River Media Library interrupt range</td>
</tr>
<tr>
<td>0x0100-0x017f</td>
<td>ULIP interrupt range</td>
</tr>
<tr>
<td>0x0180-0x01ff</td>
<td>simulated network interrupt range</td>
</tr>
<tr>
<td>0x0200-0x02ff</td>
<td>user interrupt range</td>
</tr>
</tbody>
</table>
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 \texttt{WDB\_POOL\_SIZE} macro.

System Memory Pool
Size depends on the size of the system image. The \texttt{sysMemTop( )} routine returns the address of the end of the free memory pool.

Interrupt Stack
Size is defined by \texttt{ISR\_STACK\_SIZE} under \texttt{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; this 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)

LOCAL_MEM_LOCAL_ADRS

sysBootLine = BOOT_LINE_ADRS = LOCAL_MEM_LOCAL_ADRS+BOOT_LINE_OFFSET

sysExcMsg = EXC_MSG_ADRS = LOCAL_MEM_LOCAL_ADRS+EXC_MSG_OFFSET

RAM_LOW_ADRS = LOCAL_MEM_LOCAL_ADRS+0x10000

sysMemTop()=LOCAL_MEM_LOCAL_ADRS+LOCAL_MEM_SIZE-(USER_RESERVED_MEM+PM_RESERVED_MEM)

sysMemTop()=LOCAL_MEM_LOCAL_ADRS+LOCAL_MEM_SIZE- PM_RESERVED_MEM

SM_ANCHOR_ADRS=
LOCAL_MEM_LOCAL_ADRS+LOCAL_MEM_SIZE

SIMNET_MEM_ADRS= SM_MEM_ADRS+SM_TOTAL_SIZE

SIMNET_MEM_ADRS + SIMNET_MEM_SIZE

intVecBaseGet( )
4

Using the VxWorks Simulator

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 VxWorks Simulator Configuration

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

All parameters available in the standard VxWorks boot line can be specified on a VxWorks simulator target using the command-line interface. However, these parameters are lost when the simulator instance is exited even if non-volatile RAM support is included. This occurs because boot parameters are derived from the VxWorks simulator command line.

Once the VxWorks simulator is started, you can use the VxWorks `bootChange()` routine to modify boot line parameters. The new parameters will be 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 memory settings are displayed 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.
Virtual Memory Address Space

The VxWorks simulator virtual memory size is limited to 1 GB and has the following characteristics:

On Windows:

VxWorks simulator virtual base address: 0x10000000
VxWorks simulator virtual top address: 0x4FFFFFFF

On Solaris and Linux:

VxWorks simulator virtual base address: 0x60000000
VxWorks simulator virtual top address: 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>
<thead>
<tr>
<th>Protection Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (min)</td>
<td>No specific protection</td>
</tr>
<tr>
<td>1 (max)</td>
<td>Enable stack overflow protection</td>
</tr>
</tbody>
</table>

### 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 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:

```bash
> vxsim -hn hostname
```

or

```bash
> vxsim -hostname hostname
```
On Linux or 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>
<thead>
<tr>
<th>Host Type</th>
<th>passFS Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux or Solaris</td>
<td>passFSDevice:/dir/file</td>
<td>ls myhost:/myDir/myFile</td>
</tr>
<tr>
<td></td>
<td>(where host name is myHost)</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>passFSDevice:/disk/dir/file</td>
<td>ls host:/c/myDir/myFile</td>
</tr>
<tr>
<td>Windows (deprecated syntax preserved for backward compatibility)</td>
<td>passFSDevice:disk:/dir/file</td>
<td>ls host:c:/myDir/myFile</td>
</tr>
</tbody>
</table>

**NOTE:** passFS uses UNIX-style path separators (/) even on the Windows 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.

Support for virtual disk is included by default. 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( ):

```c
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, such as dosFsDevInit( ) or dosFsMkfs( ). Consider the following code fragment:

```c
BLK_DEV * pBlkDev;
pBlkDev = virtualDiskCreate ("c:/tmp/filesys1", 512, 400, 400);
dosFsMkfs ("C:", pBlkDev);
```

This code creates C, a 200 KB DOS disk with 512-byte blocks, and only one track. 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 /ttyCo/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

An SIO driver is provided with VxWorks simulator BSPs to handle standard input and output. On Windows simulators, this is the `winSio.c` file. On Linux and Solaris simulators, it is the `unixSio.c` file.

**Linux and Solaris Simulators**

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.
4.3.7 Serial Line Support

A sample host serial I/O driver (hostSio) is provided with the VxWorks simulator. 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.
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 must also enable IP forwarding between the simulator interfaces. To enable IP forwarding, set the IPFORWARDING_CFG parameter to TRUE.

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

Starting the Master Simulator

The master simulator is used 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
```
Starting the Slave Simulators

Once the master simulator instance is started, slave simulator instances can be started with a gateway set to the master simulator using the `-gateway` (or `-g`) option as follows:

```bash
> 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:

```bash
> 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 target shell as follows:

```bash
-> routec "add -net 0.0.0.0 161.27.0.1"
```

**NOTE:** If you choose to use the alternative option described above, you must include the `INCLUDE_ROUTECMD` 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:

```bash
> route add 161.27.0.0 MASK 255.255.255.0 192.168.200.1
```

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

```bash
% 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 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 appropriate 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 45
5.2 Building Network Simulations 46
5.3 Setting Up the Network Daemon 48
5.4 Installing the Host Connection Driver 62
5.5 Configuring a Simulated Subnet 66

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. For information on using the VxWorks simulator with IPv6, see 6.4 IPv6 Tutorial, p.84.
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 `etherreal`, 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](image)

Virtual subnet 192.168.3.0 is an entirely internal subnet. It is isolated from the host, although you can use a target 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 192.168.2.0 is an “external” subnet. It is networked to the host workstation through
the host adapter interface, 192.168.2.254. If you set up the host system route table correctly, the host system can route packets for the 192.168.2.0 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.

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. 62.

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), this 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 vxminnetd 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.

**Starting vxsimnetd as a Windows Service**

To install vxsimnetd as a Windows service:

1. Log in to the Windows host with administrator privileges.
2. From the Windows Start menu, select Run....
3. Browse to `installDir/vxworks-6.1/host/x86-win32/bin/vxsimnetds_inst.exe` (where `installDir` is the name of your VxWorks install 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** and start the service.

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 on the **Wind River Network Daemon for VxWorks**, 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.

**Removing the vxsimnetd Service**

To remove the network daemon service, open a VxWorks development shell (Start > Programs > Wind River > VxWorks 6.1 > VxWorks Development Shell) and enter the following:

```
> vxsimnetds_inst.exe /u
```

This uninstalls the vxsimnetd service.

**Starting vxsimnetd as a Root Service**

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:
   ```bash
   #!/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:
   ```bash
   #!/bin/sh
   
   # chkconfig: 3 91 02
   # description: Starts and stops the vxsimnetd daemon \ 
   # used to provide VxWorks Simulator network services.
   
   # 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
5 Networking with the VxWorks Simulator
   5.3 Setting Up the Network Daemon

   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:

```bash
> ln -s /etc/init.d/vxsimnetd S91vxsimnetd
```

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

```bash
> /sbin/chkconfig --add vxsimnetd
```

This creates two links on /etc/init.d/vxsimnetd, /etc/rc3.d/S91vxsimnetd and /etc/rc6.d/K02vxsimnetd.

4. Start vxsimnetd using:

```bash
> /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 5.3.3 Network Daemon Configuration File, p.56.

- `-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 5.3.2 Network Daemon Debug Shell, p.53.
 Networking with the VxWorks Simulator

5.3 Setting Up the Network Daemon

-sv or -shellserver
This option starts the network daemon in server/background mode. When in background mode, 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 with the host node set in promiscuous mode.

5.3.2 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):

<table>
<thead>
<tr>
<th>#</th>
<th>COMM</th>
<th>STATUS</th>
<th>IP</th>
<th>PROMISC</th>
<th>RCVQ</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>special(*)</td>
<td>UP</td>
<td>192.168.200.254</td>
<td>Yes</td>
<td>0</td>
<td>24076</td>
</tr>
</tbody>
</table>

If a node number (nodeNb) or node IP address (nodeIp) is specified, specifics about the particular node, such as the number of packets sent and received, are displayed. 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 1 is currently in use, and 7 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 vxsimnetd was started 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.48.

source configFile
This command reads subnet configuration information from a file and adds the corresponding subnets. This option must be able to add all configured subnets or no subnets will be added.

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 5 percent, 5 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.
5.3.3 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-CIDR\(^1\) default mask applies. For 192.168.200.0, that would be the mask for a class C address, which is `0xffffff00`.

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.
### VxWorks Simulator Network Daemon Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Parameters:</strong></td>
<td></td>
</tr>
</tbody>
</table>
| DEFAULT_GARBAGE    | Alias: \texttt{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: \texttt{dmacprefix}  
Default Value: 7a:7a  
Specifies the first bytes of simulator Ethernet addresses. |
| DEFAULT_UID        | Alias: \texttt{duid}  
Default Value: user ID of the user that started the network daemon  
Defines the user ID (UNIX only). |
| DEFAULT_GID        | Alias: \texttt{dgid}  
Default Value: group ID of the user that started the network daemon  
Defines the group ID (UNIX only). |
| DEFAULT_ACCESSMODE | Alias: \texttt{daccessmode}  
Default Value: “0666”  
Defines access mode (UNIX only). The three parameters (\texttt{duid}, \texttt{dgid}, and \texttt{daccessmode}) can be used 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: \texttt{dexternal}  
Default Value: no  
Defines the default subnet type. |
### Table 5-1 VxWorks Simulator Network Daemon Configuration Parameters (cont’d)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| DEFAULT_EXTPROMISC   | Alias: dextpromisc  
                       | Default Value: yes  
                       | Defines whether the external subnet host node is set in promiscuous mode. |
| DEFAULT_E RATE       | 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:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Alias</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBNET_MACPREFIX</td>
<td>macprefix</td>
<td>DEFAULT_MACPREFIX</td>
<td>Specifies the first two bytes of the Ethernet address on this subnet. Overrides DEFAULT_MACPREFIX.</td>
</tr>
<tr>
<td>SUBNET_UID</td>
<td>uid</td>
<td>DEFAULT_UID</td>
<td>Specifies the user IP for this subnet. Overrides DEFAULT_UID.</td>
</tr>
<tr>
<td>SUBNET_GID</td>
<td>gid</td>
<td>DEFAULT_GID</td>
<td>Specifies the group ID for this subnet. Overrides DEFAULT_GID.</td>
</tr>
<tr>
<td>SUBNET_ACCESSMODE</td>
<td>accessmode</td>
<td>DEFAULT_ACCESSMODE</td>
<td>Specifies the access mode for this subnet. Overrides DEFAULT_ACCESSMODE.</td>
</tr>
</tbody>
</table>
### Table 5-1  VxWorks Simulator Network Daemon Configuration Parameters (cont'd)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Topology Parameters:</strong></td>
<td></td>
</tr>
</tbody>
</table>
| 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. Only one external subnet is supported for each host. |  |
| 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. |  |
### Resource-Related Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Alias</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBNET_MAXBUFFERS</td>
<td>maxbuffers</td>
<td>100</td>
<td>Specifies the maximum number of packet buffers available.</td>
</tr>
<tr>
<td>SUBNET_MAXNODES</td>
<td>maxnodes</td>
<td>32</td>
<td>Specifies the maximum number of simulators that can attach to this subnet.</td>
</tr>
<tr>
<td>SUBNET_RECVLEN</td>
<td>recvqlen</td>
<td>64</td>
<td>Specifies how many packets can be queued to a simulator.</td>
</tr>
<tr>
<td>SUBNET_SHMKEY</td>
<td>shmkey</td>
<td>IP address</td>
<td>Specifies the shared memory key.</td>
</tr>
</tbody>
</table>

### Option Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Alias</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBNET_BROADCAST</td>
<td>broadcast</td>
<td>yes</td>
<td>Specifies whether to allow MAC broadcast packets.</td>
</tr>
<tr>
<td>SUBNET_MULTICAST</td>
<td>multicast</td>
<td>yes</td>
<td>Specifies whether to allow multicast packets.</td>
</tr>
<tr>
<td>SUBNET_ERATE</td>
<td>errorate</td>
<td>DEFAULT ERATE</td>
<td>Defines the subnet error rate (the percentage of packet loss on this subnet)</td>
</tr>
</tbody>
</table>
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:

A VxWorks simulator host connection driver (WRTAP) driver must be installed and configured for each external subnet. (For information on installing and configuring the WRTAP driver, see 5.4 Installing the Host Connection Driver, p.62.) 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 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 attach to this host interface to monitor traffic on the internally simulated subnet.

Windows Hosts

To install the driver on an 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-6.1\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 2000 host:

1. Open the Control Panel.
2. Double-click **Add/Remove Hardware** to open the **Add Hardware Wizard**.
3. Click **Next**.
4. Choose the hardware task, **Add/Troubleshoot a device**.
5. Select **Add a new device**.
6. When prompted to search for the hardware, select **No, I want to select the hardware from a list**.
7. Select **Network Adapters**.
8. Click the **Have Disk...** button.
9. Click **Browse**. Browse to `installDir\vxworks-6.1\host\x86-win32\bin` (where `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 **Finish**.

**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.64.

**Solaris Host**

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

1. Copy `installDir/vxworks-6.1/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 Host

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.62.

**Migrating from the ULIP Driver**

The WRTAP driver replaces the ULIP driver used in earlier VxWorks simulator releases. The WRTAP driver can be used 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.
5 Networking with the VxWorks Simulator

5.4 Installing the Host Connection Driver

To access Windows network connection settings, select **Start > Control Panel > Network Connections** (on Windows XP hosts) or **Start > Settings > Control Panel > Network and Dial-up Connections** (on Windows 2000 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).

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 Dial-up Connections**) window, select **Advanced > Advanced Settings**...
- 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 > Settings > Control Panel > Network and Dial-up Connections** (on Windows 2000 hosts). In the **Network Connections** (or **Network and Dial-up Connections**) window, right-click the WRTAP network interface you want to disable and select **Disable**.

NOTE: You will need administrator privileges on the Windows host to make the following changes.
5.5 Configuring a Simulated Subnet

This section describes how to configure a subnet of VxWorks simulator instances.

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` supports a `-ni` option. The syntax for this option is as follows:

```
devicenameDevicenumber:subnet=IP_address:IP_netmask
```

This describes one interface. You can chain these together using a ‘;’ delimiter. For example:

```bash
> vxsim -ni "simnet2=192.168.2.1:0xfffffff0;simnet3=192.168.3.1:0xfffffff0;simnet4=192.168.4.1:0xfffffff0"
```

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 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.

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:

```bash
> 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"
```

This command is used 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 69
6.2 Simple Simulated Network 69
6.3 Basic Simulated Network with Multiple Simulators 73
6.4 IPv6 Tutorial 84

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.

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. This
simple network can be set up 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 the Wind River Workbench IDE.

### 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, the network daemon is started on the host before any VxWorks simulator instances are configured.

**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.62.

**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.

**WARNING:** This 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 5.3.3 *Network Daemon Configuration File*, p.56.
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. 48 for complete instructions. If you have already started `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:
   ```
   > installDir\vxworks-6.1\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-6.1/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-6.1/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 so you do not need to reconfigure or rebuild the default VxWorks image for the VxWorks simulator.
Start the Simulator 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 6.1 > VxWorks Development Shell.
   
   On Solaris and Linux hosts, run the `wrenv` utility program to open a development shell as follows:
   
   ```sh
   wrenv.sh -p vxworks-6.1
   ```
   
2. In the VxWorks development shell, type the following:
   
   ```sh
   vxsim -d simnet -e 192.168.200.1 -f installDir\vxworks-6.1\target\config\bsp\vXWorks
   ```
   
   where `installDir` is the name of your VxWorks install directory and `bsp` is the name of the BSP directory for the VxWorks simulator on your host (For example, `simpc` on Windows hosts).
   
   The above command starts a VxWorks simulator instance and attaches it to the default subnet. The VxSim0 console windows appears.

Start the Simulator 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 Simulator Connection from the selection list. Click Next.
   
3. In the Select Boot File Name field of 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.
   
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. 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 (all 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 **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 of 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.48.

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, for more information on building and configuring VxWorks, see the *VxWorks Application Programmer’s Guide* or the Wind River Workbench documentation.
6.3.1 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.

Configure and Launch vxsimnetd

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:

- Stopped any previously started VxWorks simulator network daemons (including those started as a service).
- 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.62.)

Now, create a vxsimnetd configuration file. Create the following file and save it as vxsimTest.conf.

```plaintext
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;
};
```

Next, launch vxsimnetd using the configuration file you just created.
On Windows hosts:

Log in with administrator privileges and start \texttt{vxsimnetd} from a command window as follows:

\begin{verbatim}
> \texttt{installDir/vxworks-6.1/host/x86-win32/bin/vxsimnetd} -f \texttt{vxsimTest.conf}
\end{verbatim}

Be sure to provide the full path to your \texttt{vxsimTest.conf} file if it is not in your current directory.

On Solaris or Linux hosts:

Log in as root and start \texttt{vxsimnetd} from a host shell as follows:

\begin{verbatim}
% \texttt{installDir/vxworks-6.1/host/myHost/bin/vxsimnetd} -f \texttt{vxsimTest.conf}
\end{verbatim}

In this command line, \texttt{myHost} is your host type: \texttt{x86-linux2} for Linux or \texttt{sun4-solaris2} for Solaris.

Be sure to provide the full path to your \texttt{vxsimTest.conf} file if it is not in your current directory.

\begin{itemize}
\item \textbf{NOTE:} You can also start \texttt{vxsimnetd} as a service. For instructions, see Starting the Network Daemon as a Service, p. 48. Note that if you choose to start \texttt{vxsimnetd} as a service, you will need to configure the daemon as directed in this section before proceeding with the tutorial.
\end{itemize}

\begin{itemize}
\item \textbf{NOTE:} If you do not start \texttt{vxsimnetd} with administrator (or root) privileges, \texttt{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.
\end{itemize}

\section*{Prepare a VxWorks Image for Use with the Simulated Network}

In this tutorial, the \texttt{VxSim0} simulator instance acts as a router. This requires that IP forwarding be enabled in the VxWorks image. This tutorial also uses ping to communicate between simulator instances so ping functionality must also 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 set the \texttt{IPFORWARDING_CFG} parameter to \texttt{TRUE}. In addition, the \texttt{INCLUDE_ROUTECMD} and \texttt{INCLUDE_PING} components must be included in the 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:
   
   `> 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. Now, add the `INCLUDE_ROUTECMD` and `INCLUDE_PING` components to the image and set the `IPFORWARDING_CFG` parameter to `TRUE`:

   `> cd c:\myInstallDir\network_demo`
   
   `> vxprj component add INCLUDE_ROUTECMD INCLUDE_PING`
   
   `> vxprj parameter set IPFORWARDING_CFG TRUE`

3. Now, rebuild VxWorks. In the project directory, 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, `simpcc` on Windows hosts) and your desired tool chain (for example, `diab`). Click **Next**.

d. In the **Networking Options** dialog, click **Next** to accept the default options (no options are selected by default).

e. In the **Configuration Profile** dialog, select **(Default)** from the **Profile** drop-down list. Click **Finish**.

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

2. Configure your kernel to include the appropriate networking components.

   a. Expand the new `network_demo` project and right-click on **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 items** 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. You can also use the **Find** tool to locate the `IPFORWARDING_CFG` parameter. Once you have located the parameter and returned to the component tree, change the value for the `IPFORWARDING_CFG` property (located in the frame below the component tree) to **TRUE**. The component tree should now show **Enable IP forwarding between interfaces = TRUE**.

   e. Right-click in the component tree and select **Save**.

3. Build your project.

   f. 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).

### 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 (`installDir\network_demo`).

---

---
Configure 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"
```

The VxSim0 console window appears. The VxSim0 instance acts as the simulated router.

Configure the Simulated Network Devices

To configure the VxWorks simulator instances for the simulated network devices, 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
```

You should now have three simulated network devices; VxSim1, VxSim2, and VxSim3.

Start the VxWorks Simulator Instances from Workbench

Configure the Simulated Router

First, start the VxWorks simulator instance that will act as the router in the simulated subnet. 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 Simulator Connection from the selection list. Click Next.
3. In the Select Boot File Name field of VxWorks Boot Parameters dialog, click the Custom Simulator radio button. In the VxWorks Kernel Image field, browse to your project location and click Open to select your VxWorks image (projectLocation\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"

   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.


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

Configure the Simulated Network Devices

Now, configure each of the remaining simulated devices (repeat the following process for each instance). To configure each of the devices, complete the following steps:

1. Select Target > New Connection....

2. In the Connection Type dialog box, select Wind River VxWorks Simulator Connection from the selection list. Click Next.

3. In the Select Boot File Name field of VxWorks Boot Parameters dialog, click the Custom Simulator radio button. In the VxWorks Kernel Image field, browse to your project location and click Open to select your VxWorks image (projectLocation\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 192.168.200.3 when starting the second instance and 192.168.200.3 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 (all 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 **Connection Summary** dialog appears. Click **Finish**.

Once you have repeated this process for all three VxWorks simulator instances, you will have three simulated network devices; **VxSim1**, **VxSim2**, and **VxSim3**.

### Run the Ping Application

Before pinging between simulators, you must add the appropriate routes.

In the **VxSim1** shell, type:

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

In the **VxSim2** shell, type:

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

In the **VxSim3** shell, type:

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

**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:

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

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

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

```bash
-> ping "192.168.3.2", 5
-> ping "192.168.4.2", 5
```
6.3.2 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 Static Configuration, p. 74. You must use the VxWorks image you created in the earlier tutorial to launch the VxWorks simulator instances in this tutorial. Therefore, you should complete the earlier tutorial before beginning this section.

### 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.

Now, 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-6.1\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.

### Configure vxsimnetd Dynamically Using the Shell

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

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

Save this file as `sub_3_4.conf`.

Now, 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 as follows:

```
Escape character is '
'.
```

```

vxsimnetd Debug Shell
```

Source the new configuration file as follows:

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

Subnet <sub3> added.
Subnet <sub4> added.

**Launch the VxWorks Simulator Instances**

The VxWorks simulator instances are launched in the same configuration and manner described in *Launch the VxWorks Simulator Instances*, p.78.

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

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:

```plaintext
> route add 192.168.3.0 MASK 255.255.255.0 192.168.200.1
> 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:

```plaintext
% 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:

```plaintext
-> routec ("add -net 192.168.200.0 192.168.3.1");
-> routec ("add -net 192.168.200.0 192.168.4.1");
```

Now, verify the network connection by pinging the VxSim2 instance from the host shell as follows:

```plaintext
> ping 192.168.3.2
```

### 6.4 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*.

This tutorial describes how to:

- enable IPv6 support on your host system.
6 Networking Tutorials
6.4 IPv6 Tutorial

- 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.

6.4.1 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.

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:

`> 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, assume 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.

Configure and Start the Network Daemon

This tutorial uses a network configuration that includes 2 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 Starting a Simulator Instance Without an IPv4 Address, p.66).
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 *Configure and Launch vxsimnetd*, p.74 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 *Launch the vxsimnetd Shell Server*, p.82. 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
     ```
6.4.2 Configure VxWorks

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.

This tutorial uses two VxWorks simulator configurations to demonstrate router advertisement. One configuration is for the solicitor and the other is for the advertiser.

Solicitor Configuration

The solicitor configuration requires that your VxWorks image be configured with the following components:

- INCLUDE_IFCONFIG
- INCLUDE_RTSOL
- INCLUDE_PING6

You must also set the IPV6CTL_ACCEPT_RTADV_CFG parameter to TRUE and you must specify the interface setting for RTSOL_COMMAND as "simnet0".

You can set these components using the vxprj command-line utility using the following steps:

1. Create a project called demo_solicitor. For example:

   ```bash
   > vxprj create -inet6 bsp TOOL demo_solicitor
   ```
   
   where bsp is the BSP for your host system type (simpC, linux, or solaris) and TOOL is your desired toolchain (diab or gnu).

2. Change to the demo_solicitor directory:

   ```bash
   > cd installDir/demo_solicitor
   ```

3. Add the INCLUDE_RTSOL, INCLUDE_IFCONFIG, and INCLUDE_PING6 components:

   ```bash
   > vxprj component add INCLUDE_RTSOL INCLUDE_IFCONFIG INCLUDE_PING6
   ```

4. Set the IPV6CTL_ACCEPT_RTADV_CFG parameter to TRUE:

   ```bash
   > vxprj parameter set IPV6CTL_ACCEPT_RTADV_CFG TRUE
   ```
5. Specify the interface setting for RTSOL_COMMAND:

```shell
> vxprj parameter setstring RTSOL_COMMAND "simnet0"
```

**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.76 or the *Wind River Workbench User’s Guide*.

---

**Advertiser Configuration**

The advertiser configuration requires that your VxWorks image be configured with the following components:

- INCLUDE_IFCONFIG
- INCLUDE_RTADV
- INCLUDE_PING6
- INCLUDE_NDP

You must also set the IPV6CTL_ACCEPT_RTADV_CFG value to TRUE and you must specify the interface setting for RTADV_COMMAND as "simnet0".

You can set these components using the vxprj command-line utility using the following steps:

1. Create a project called `demo_advertiser`. For example:

```shell
> vxprj create -inet6 bsp TOOL demo_advertiser
```

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

2. Change to the `demo_advertiser` directory:

```shell
> cd installDir/demo_advertiser
```

3. Add the INCLUDE_RTADV, INCLUDE_NDP, INCLUDE_IFCONFIG, and INCLUDE_PING6 components:

```shell
> vxprj component add INCLUDE_RTADV INCLUDE_NDP INCLUDE_IFCONFIG INCLUDE_PING6
```

4. Set the IPV6CTL_ACCEPT_RTADV_CFG parameter to TRUE:

```shell
> vxprj parameter set IPV6CTL_ACCEPT_RTADV_CFG TRUE
```

5. Specify the interface setting for RTADV_COMMAND:

```shell
> vxprj parameter setstring RTADV_COMMAND "simnet0"
```
Build Your Projects

Now, build the `demo_solicitor` and `demo_advertiser` projects as follows:

```
> cd installDir/demo_solicitor
> vxprj build
> cd installDir/demo_advertiser
> vxprj build
```

**NOTE:** Your projects can also be built using Workbench.

### 6.4.3 Test the IPv6 Connection

**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.

**Start the Simulator Instances on the Default Subnet**

First, create a startup script (`default_startup`) that specifies the prefix to advertise:

```
rtadvConfig("simnet0 add 3ffe:1::");
```

Next, 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 demo_advertiser/default/vxworks -s demo_advertiser/default_startup
```

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

Now, start a solicitor VxWorks simulator instance on the default subnet as follows:

```
> vxsim -p 1 -d simnet -e 192.168.200.2 -f demo_solicitor/default/vxworks
```
This starts a VxWorks simulator instance (*VxSim1*) with the solicitor configuration on the default subnet.

**Start the Simulator Instances on Subnet 3**

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

```
rtdvConfig ("simnet0 add 3ffe:2::");
```

Next, 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
demo_advertiser/default/vxworks -s demo_advertiser/sub3_startup
```

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

Now, start the solicitor VxWorks simulator instance as follows:

```
> vxsim -p 3 -d simnet -e 192.168.3.2 -f demo_solicitor/default/vxworks
```

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

**Check Your Connections**

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

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

In the *VxSim0* console, type:

```bash
-> ifconfig "simnet0"
simnet0: flags=e8043 mtu 1500
<UP,BROADCAST,RUNNING,MULTICAST,NOTRAILERS,INET_UP,INET6_UP>
inet6 fe80::787a:c0ff:fe8a:c801%simnet0 prefixlen 64 scopeid 0x2
inet 192.168.200.1 netmask 0xffffff00 broadcast 192.168.200.255
inet6 3ffe:1::787a:c0ff:fe8a:c801 prefixlen 64 autoconf
  ether 7a:7a:c0:a8:c8:01
value = 0 = 0x0
```
In the VxSim1 console, type:

```
-> ifconfig "simnet0"
```

```
simnet0: flags=8043 mtu 1500
<UP,BROADCAST,RUNNING,MULTICAST,NOTRAILERS,INET_UP,INET6_UP>
    inet6 fe80::787a:c0ff:fe8a:c802%simnet0 prefixlen 64 scopeid 0x2
    inet 192.168.200.2 netmask 0xffffff00 broadcast 192.168.200.255
    inet6 3ffe:1::787a:c0ff:fe8a:c802 prefixlen 64 autoconf
    ether 7a:7a:a8:c8:02
    value = 0 = 0x0
```

In the VxSim2 console, type:

```
-> ifconfig
```

```
lo0: flags=8049 mtu 1536
<UP,LOOPBACK,RUNNING,MULTICAST,INET_UP,INET6_UP>
    inet 127.0.0.1 netmask 0xff000000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
```

In the VxSim3 console, type:

```
-> ifconfig "simnet0"
```

```
simnet0: flags=8043 mtu 1500
<UP,BROADCAST,RUNNING,MULTICAST,NOTRAILERS,INET_UP,INET6_UP>
    inet6 fe80::787a:c0ff:fe8a:c802%simnet0 prefixlen 64 scopeid 0x2
    inet 192.168.3.2 netmask 0xffffff00 broadcast 192.168.3.255
    inet6 3ffe:2::787a:c0ff:fe8a:c803 prefixlen 64 autoconf
    ether 7a:7a:a8:c8:02
    value = 0 = 0x0
```

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.

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

```
-> ping6 "fe80::787a:c0ff:fe8a:c801%simnet0"
```

```
PING6(56=40+8+8 bytes) fe80::787a:c0ff:fe8a:c801 -->
fe80::787a:c0ff:fe8a:c801 16 bytes from fe80::787a:c0ff:fe8a:c801 (fe80::787a:c0ff:fe8a:c801):
    icmp_seq=0 hlim=64 time=0 ms
--- fe80::787a:c0ff:fe8a:c801%simnet0 ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
```
round-trip min/avg/max = 0/0/0 ms
value = 0 = 0x0

You can also ping **VxSim0** with the automatically configured address:

```bash
-> ping6 "3ffe:1::787a:c0ff:fea8:c801"
PING6(56=40+8+8 bytes) 3ffe:1::787a:c0ff:fea8:c802 -->
3ffe:1::787a:c0ff:fea8:c801
16 bytes from 3ffe:1::787a:c0ff:fea8:c801 (3ffe:1::787a:c0ff:fea8:c801):
icmp_seq=0 hlim=64 time=450.022 ms

--- 3ffe:1::787a:c0ff:fea8:c801 ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 450.022/450.022/450.022 ms
value = 0 = 0x0
```

You can also ping the host:

```bash
-> ping6 "3ffe:1::787a:c0ff:fea8:c8fe"
PING6(56=40+8+8 bytes) 3ffe:1::787a:c0ff:fea8:c802 -->
3ffe:1::787a:c0ff:fea8:c8fe
16 bytes from 3ffe:1::787a:c0ff:fea8:c8fe (3ffe:1::787a:c0ff:fea8:c8fe):
icmp_seq=0 hlim=64 time=16.706 ms

--- 3ffe:1::787a:c0ff:fea8:c8fe ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 16.706/16.706/16.706 ms
value = 0 = 0x0
```

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.73.
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:

```c
/* 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 a host routine is called from VxWorks code, you must always lock interrupts before calling the routine. Failure to do so can result in unexpected VxWorks simulator behavior.

- When a host routine is called and the routine is system blocking, it 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.99.

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.95.

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 \textit{A.2 Accessing Host OS Routines}, p.94.

\section*{A.4 Host Application Interface (\textit{vxsimapi})}

The \textit{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 \textit{vxsimapi}.

\subsection*{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 is exited or rebooted.

\subsection*{A.4.2 Configuring a Host Device to Generate interrupts (UNIX Only)}

The host file descriptors can be put in asynchronous mode, such that a \texttt{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 \texttt{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.
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.

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

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.

```c
#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) {
    /* Funcion 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 */
    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);

---

Running The Code

The sample code can be executed directly from the VxWorks target shell or using the host shell. A sample 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 94
accessing the VxWorks Simulator from a remote host 14
application compatibility 2, 43
assigning a processor number 12
AUX_CLK_RATE_MAX 39
AUX_CLK_RATE_MIN 39
auxiliary clock 39
auxiliary clock interrupts 27, 28

B
-b 7, 9
-backplane 7, 9
boot parameters 9, 34
BOOT_NO_AUTOBOOT 13
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 7, 75
   with vxprj 8
   with Workbench 8
building applications 19
byte order 25

C
C++ modules 19
commSio 99
compiler options 19
   -g 21
   -O 21
   -O0 21
   -Xno-optimized-debug 21
   -XO 21
config.h 18, 34
configuring a simulated subnet 66
configuring IPv6 support on your host system 85
configuring multiple external subnets 61
configuring multiple network interfaces 13
connection timeout 24
CPU 19
D
-d 9
deprecated 21
DEFAULT_ACCESSMODE 57
DEFAULT_ERATE 58
DEFAULT_EXTERNAL 57
DEFAULT_EXTPROMISC 58
DEFAULT_GARBAGE 57
DEFAULT_GID 57
DEFAULT_MACPREFIX 57
DEFAULT_TIMEOUT 58
DEFAULT_UID 57
development limitations 3, 43
-device 9
devs() 38
diab 19
DLL 93
dosFsDevInit() 38
dosFsMkfs() 38
dynamic-link library
  see DLL 93

E
-e 9, 12
ELF object module format 20
emacs 53
-ethernet 9
exit hook facility 95
exiting the VxWorks Simulator 13
-exitOnError 9, 13

F
-f 9, 12, 52
-file 9, 52
file system support 3, 23
  pass-through file system 24, 36
  virtual disk 24, 37
-flags 9
floating-point routines 25
floating-point support 25
-force 53

G
-g 9
-gateway 9
gnu 19

H
-h 10
hardware breakpoint support 25
hardware simulation 36
-help 9
-hn 10, 36
host application interface 95
host connection driver 62
  Linux 64
  Solaris 63
  Windows 62
host system requirements 6
HOST_SIO_PORT_NUMBER 40
-hostinet 10
-hostname 10, 36
hostSio 40

I
ifconfig() 67
INCLUDE_BOOT_LINE_INIT 7
INCLUDE_HOST_SIO 40
INCLUDE_IFCONFIG 87, 88
INCLUDE_KERNEL 30
INCLUDE_NDP 88
INCLUDE_NET_BOOT_CONFIG 67
INCLUDE_PASSFS 36
INCLUDE_PING 75
INCLUDE_PING6 87, 88
INCLUDE_ROUTECMD 75
INCLUDE_RTADV 88
INDEX

INCLUDE_RTSOL 87
INCLUDE_SM_COMMON 7, 41
INCLUDE_SM_NET 7, 41
INCLUDE_SM_OBJ 7, 41
INCLUDE_SYS_TIMESTAMP 39
INCLUDE_TIMESTAMP 39
INCLUDE_VIRTUAL_DISK 37
intConnect() 27, 29, 96
intDisconnect() 96
interrupt assignments
  Windows simulator 28
interrupt simulation 26
  host signals 26
  on Linux and Solaris 26
  on Windows 28
  Windows messages 28
IPFORWARDING_CFG 41, 75
IPv6
  configuring IPv6 support on your host system 85
  support 45
tutorial 84
IPV6CTL_ACCEPT_RTADV_CFG 87, 88
ISR_STACK_SIZE 30

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
  LOCAL_MEM_SIZE 34
  MACH_EXTRA 18
  NV_RAM_SIZE 38
  SYS_CLK_RATE_MAX 39
  SYS_CLK_RATE_MIN 39
Makefile 7, 18
memory configuration 34
memory layout 30
memory management support
  running without MMU 23
memory management unit
  limitations 23
  MMU simulation 22
  page size 23
  translation Model 22
memory protection level
  -memsize 10, 34
migrating applications 43
MMU
  see memory management unit 3
  MMU simulation 22

N

-nvram 10
-netif 10
network daemon
  configuration file parameters 56
  configuration files 56
network simulation 46
networking address space
  -ni 10, 13
non-volatile RAM support 38
NV_RAM_SIZE 38
-nvram 38
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| `O` | -o 10  
-o other 10 |
| `P` | -p 10, 12  
packet loss 55  
packet sniffers 46, 55, 62  
passDevice 9  
passFS  
see pass-through file system 24  
pass-through file system 24, 36  
-password 10  
physical memory address space 34  
-pl 10  
-processor 10  
-prot-level 10, 35  
-pw 10 |
| `R` | remote host 14  
router configuration 13  
routines  
bootChange() 9, 12, 34  
devs() 38  
dosFsDevInit() 38  
dosFsMkfs() 38  
ifconfig() 67  
intConnect() 96  
intDisconnect() 96  
sysAuxClkRateSet() 39  
sysClkRateSet() 39  
sysMemTop() 30  
sysNvRamGet() 38  
sysNvRamSet() 38  
virtualDiskClose() 38  
virtualDiskCreate() 38  
virtualDiskInit() 38  
vxsimExitHookAdd() 95 |
| `S` | -s 10, 52, 53  
serial I/O driver 40  
serial line support 40  
shared memory address space 30  
shared memory END driver 7  
shared memory network 40  
shared memory pool size 7  
shared memory size 7  
-shell 52, 53  
-shellport 53  
-shellserver 53  
SIGALRM 26  
SIGPOLL 26, 95  
SIGUSR1 28  
SIGUSR2 28  
SIMLINUX 20  
simpc 8  
SIMPENTIUM 20  
SIMSPARCSOLARIS 20  
simulated hardware support 3  
simulating packet loss 55  
SIO driver 39  
-size 10, 34  
SM_MEM_SIZE 7  
SM_OBJ_MEM_SIZE 7  
smEnd 7  
solaris 8  
-sp 53  
specifying the passFS device name 36  

vxsimFdIntDisable() 96  
vxsimFdIntEnable() 95  
vxsimHostDllLoad() 94  
vxsimHostProcAddrGet() 94, 95  
vxsimIntAckRtnAdd() 96  
vxsimIntRaise() 96  
vxsimIntToMsg() 97  
vxsimMsgToInt() 97  
RTADV_COMMAND 88  
RTP support 3, 23  
RTSOL_COMMAND 87
standard I/O 18, 39
  on Linux and Solaris simulators 39
starting a standalone VxWorks Simulator instance 11
starting simulator instances for use with network services 12
starting the VxWorks Simulator from Workbench 13
-startup 10
SUBNET_ACCESSMODE 58
SUBNET_ADDRESS 59
SUBNET.Broadcast 60
SUBNET.ERATE 60
SUBNET_EXTCONNNAME 61
SUBNET_EXTDEVNUM 59, 61
SUBNET.EXTERNAL 59
SUBNET.EXTPROMISC 59
SUBNET_MACPREFIX 58
SUBNET_MASK 59
SUBNET_MAXBUFFERS 60
SUBNET_MAXNODES 60
SUBNET_MTU 61
SUBNET_MULTICAST 60
SUBNET_REQLEN 60
SUBNET_SHMKEY 60
SUBNET_TIMEOUT 61
SUBNET_UID 58
supported VxWorks features 2, 6
-sv 53
SYS_CLK_RATE_MAX 39
SYS_CLK_RATE_MIN 39
sysAuxClnRateSet() 39
sysBootLine 18
sysBusToLocalAdrs() 18
sysClkRateSet() 39
sysLib.c 18
sysMemTop() 30
sysNvRamGet() 38
sysNvRamSet() 38
system clock 39

T
-targetname 10
timers 39
timestamp driver 39
-tmpdir 11	n
-TN 10
TOOL 19
ttySio 99
tun module 64
tutorial
  IPv6 84
  simple simulated network 69
  simulated network with multiple simulators 73
tyLib 39
tyLib.c 18

U
-unit 11
UNIX disk driver library 37
unixDrv 37
unixSio.c 18, 39
-user 11
USER_INT_RANGE_BASE 97
USER_INT_RANGE_END 97
-username 11

V
-v 11
-vaddr 11, 35
-version 11
vi 53
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 76
vxsim 9, 11
  configuration options 9
vxsimapi 93, 95
vxsimExitHookAdd() 95
vxsimFdIntDisable() 96
vxsimFdIntEnable() 95
vxsimHostArchLib 93
vxsimHostDllLoad() 94
vxsimHostProcAddrGet() 94, 95
vxsimIntAckRtnAdd() 96
vxsimIntRaise() 96
vxsimIntToMsg() 97
vxsimMsgToInt() 97
vxsimnetd 48
  command line options 52
debug shell 53
removing the Windows service 49
shell commands 53
  ? 55
delete 55
erate 55
extpromisc 55
help 55
mode emacs 55
mode vi 55
node 54
packet 54
quit 55
source 55
subnet 53
timeout 55
starting as a root service 49
starting as a Windows service 49
vxsimnetds_inst.exe 49
vxTarget 10
vxWorks 8

VxWorks Components
  INCLUDE_SM_COMMON 41
VxWorks components
  INCLUDE_BOOT_LINE_INIT 7
  INCLUDE_HOST_SIO 40
  INCLUDE_IFCONFIG 87, 88
  INCLUDE_NDP 88
  INCLUDE_NET_BOOT_CONFIG 67
  INCLUDE_PASSFS 36
  INCLUDE_PING 75
  INCLUDE_PING6 87, 88
  INCLUDE_ROUTECMD 75
  INCLUDE_RTADV 88
  INCLUDE_RTSOL 87
  INCLUDE_SM_COMMON 7
  INCLUDE_SM_NET 41
  INCLUDE_SM_OBJ 7, 41
  INCLUDE_SYS_TIMESTAMP 39
  INCLUDE_TIMESTAMP 39
  INCLUDE_Virtual_DISK 37
VxWorks image
  vxWorks 8
  vxWorks.st 8
VxWorks image projects 8
VxWorks simulator network daemon
  see vxsimnetd 48
  vxWorks.st 8

W
watchdog timer facilities 27, 28
WDB
  back end 24
  WDB_POOL_SIZE 30
Wind River System Viewer 39
winSio.c 18, 39
WRTAP 61, 62

X
xterm 8