WIND RIVER

Wind River® General Purpose Platform, VxWorks® Edition

GETTING STARTED

3.6

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

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

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

http://www.windriver.com

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

http://www.windriver.com/support

Wind River General Purpose Platform, VxWorks Edition Getting Started, 3.6

4 Dec 07
Part #: DOC-16139-ND-00

Contents

1
Overview

1.1 Introduction

Using Wind River General Purpose Platform, VxWorks Edition, you can develop a variety of projects or applications, such as VxWorks kernel-based applications, VxWorks real-time process (RTP) applications, BSPs, and drivers. This release provides VxWorks SMP for symmetric multiprocessing (as an optional product) in addition to uniprocessor VxWorks. For information about VxWorks SMP, and about migrating uniprocessor code to VxWorks SMP, see the *VxWorks Kernel Programmer's Guide: VxWorks SMP*. VxWorks SMP is a separately purchased product. Please contact your local Wind River representative for purchase details.

This document does not provide a tutorial of how to create each of the many types of projects or applications. However, it provides you with an overview of the development environment and some of the typical tasks you may want to perform. This document also provides cross-references to documents containing more detailed information about the tools and tasks.

The document is organized as follows:

- *1. Overview* (this chapter) – provides information on how the document is organized and product documentation. It also describes terminology and conventions used in the document.
- *2. Development Environment* – provides background information on tools and resources you should be familiar with before you begin the development process.
- *3. Getting Started with Development* – describes some typical development tasks and where you can find the information needed to complete these tasks.
- *4. Building VxWorks Source Code* – provides information on building VxWorks source code. In most cases, it is not necessary to build the VxWorks source.
- *5. Software Architecture* – provides information on the software architecture of the Wind River Network Stack and related applications, including the software building blocks, component structure, and interface to the operating system.

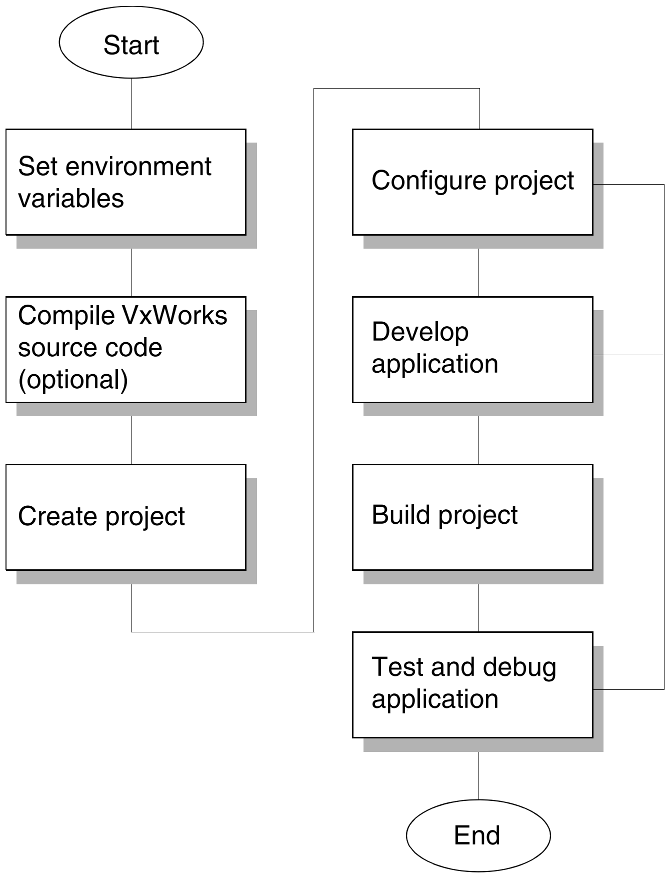
1.2 Project Development Workflow

Development of a typical VxWorks project usually consists of the following steps:

1. Set the environment variables. (See *2.2 Environment Variables*, p.10.)
2. Compile the VxWorks source code (optional). (See *4. Building VxWorks Source Code*.)
3. Create a VxWorks project.
4. Configure VxWorks to include the appropriate components. (See *3.2 Configuring a VxWorks System*, p.22.)
5. Develop either a kernel or RTP application. (See *3.3 Developing Kernel Applications*, p.23 or *3.4 Developing RTP Applications*, p.24.)
6. Build the VxWorks project.
7. Test and debug your application. (See *2.3 Wind River Workbench*, p.10.)

Figure 1-1 illustrates this workflow.

Figure 1-1 **Project Development Workflow**


Start
Set environment variables
Compile VxWorks source code (optional)
Create project
Configure project
Develop application
Build project
Test and debug application
End

1.3 Product Documentation

The documentation set for Wind River General Purpose Platform, VxWorks Edition consists of online documentation, **readme.txt** files, and some third-party documentation. This section provides an overview of the available documentation and where it can be found.

1.3.1 Introductory Documentation

This section describes the documentation that provides information on how to get started with Wind River General Purpose Platform, VxWorks Edition. It includes getting started guides, release notes, migration guides, and **readme.txt** files.

The introductory documentation is as follows:

- *Wind River General Purpose Platform, VxWorks Edition Release Notes* – contains the latest list of supported hosts and targets, information on compatibility with older releases, an outline of new features, and any caveats concerning the current release. This document is available from the Online Support Web site.
- *Wind River General Purpose Platform, VxWorks Edition Getting Started* (this guide).
- *Wind River General Purpose Platform, VxWorks Edition Migration Guide* – contains information for migrating to *Wind River General Purpose Platform, VxWorks Edition* 3.6. This document is available from the Online Support Web site.
- **readme.txt** – there are several **readme.txt** files that are included in your installation. For a complete listing of **readme.txt** files included with your installation, search *installDir* (the root installation directory).

1.3.2 Online Documentation

This section describes the online documentation available with Wind River General Purpose Platform, VxWorks Edition.

Online Manuals

This release includes manuals in HTML and PDF format. You can open the online manuals from the **Help > Help Contents** menu in Workbench. A full-text search facility is available within the Workbench help browser.

Print-ready PDFs are available from the title page of the corresponding online document. Navigate to the proper document, then click the PDF icon to the right of the book title. Links to the PDF files are also available by selecting **Wind River > Documentation** from your operating system Start menu.

In addition to the Wind River online documentation, important third-party documentation is also provided, including Dinkumware documents, Eclipse documents, and open-source documents for various products and applications.

Context Sensitive Help

Help buttons in Wind River Workbench and Wind River System Viewer provide information on the component you are currently using.

From a Workbench view, pressing **F1** on Windows, **CTRL+F1** on Linux, or the **Help** button on Solaris opens an infopop containing a brief description of the view, as well as links to related topics in the documentation. Pressing **F1** within an enabled area in System Viewer opens the appropriate help page.

System Viewer has additional ways to access context-sensitive help. Right-clicking a specific event within a tool takes you to the System Viewer User's Reference Event Dictionary information for that event or state stipple.

Man Pages

UNIX-style man pages for API reference entries are available for Solaris or Linux hosts.

To view the VxWorks API man pages, make the following modifications to your environment:

For sh or bash:

```
MANPATH=installDir/vxworks-6.x/man:$MANPATH
export MANPATH
```

For csh:

```
setenv MANPATH installDir/vxworks-6.x/man:$MANPATH
```

To display the man page, type:

```
man functionName
```

The most convenient way to access the VxWorks man pages is to create an alias for **man -M** such as **vxman**.

For sh or bash:

```
alias vxman='man -M installDir/vxworks-6.x/man'
```

For csh:

```
alias vxman 'man -M installDir/vxworks-6.x/man'
```

You can then display entries, such as the one for **printf( )**, from a shell prompt as follows:

```
% vxman printf
```

1.4 Terminology and Conventions

The following terms are used in this document:

host
A computer on which the Wind River development tools run.

target
A processor board that runs VxWorks (Wind River's real-time operating system) and applications developed with Wind River Workbench.

target server
A service that runs on the host and manages communications between host tools (such as the VxWorks development shell, debugger, and browser) and the target system itself. One target server is required for each target.

Wind River registry
A Wind River service that keeps track of, and provides access to, target servers. One registry may serve a network, or registries may run on each host.

The following conventions are used in this document:

- The root installation directory is identified as *installDir* in this document, but the environment variable **WIND_HOME** must be set to the root installation directory for Wind River Workbench to work properly.
- A series of items to be selected from the GUI is denoted by **A > B > C**. The elements **A**, **B**, and **C** may be menu items, buttons, or tabs.
- Pathnames that apply to both UNIX and Windows are shown with forward slashes (/).
- Plain italics of the default text font are applied to book titles, emphasis, special terms, and placeholders. A placeholder is a text string that is not to be interpreted literally, and represents some value that the user supplies or an element that will vary depending on context. The use of placeholders is confined mostly to command arguments, variable portions of pathnames, and function parameters.
- C subroutine names always include parentheses, as in **printf( )**.
- Combinations of keys that must be pressed simultaneously are shown with a + linking the keys. For example, **CTRL+F3** means to simultaneously press the key labeled **CTRL** and the key labeled **F3**.

This document uses the font conventions in the following table for special elements.

Table 1-1 **Typographical Conventions**

Term	**Example**
files, pathnames	*installDir*/**host**
libraries, drivers	**memLib.c**
command-line tools	**dir**
Tcl procedures	**wtxMemRead**
C subroutines	**semTake( )**
VxWorks boot commands	**p**
code display	main();
keyboard input display	-> **`wtxregd -v`**
output display	value = 0
user-supplied values	*name*
components	**INCLUDE_NFS**
keywords	**struct**
named key on keyboard	**RETURN**
key combination	**ALT+SHIFT+F5**
GUI titles and commands	**Help**
GUI menu paths	**Tools > Target Server > Configure**
references to other manuals	*Wind River Workbench User's Guide: Building Projects*

2
Development Environment

2.1 Introduction

Before you begin application development, you should be familiar with the Wind River General Purpose Platform, VxWorks Edition development environment. This chapter describes each of the common tools and resources, and provides cross references to where more information on each tool or resource can be found.

2.2 Environment Variables

A specific set of environment variables must be set on your host machine for Wind River development tools. On Windows, Wind River Workbench and the VxWorks development shell set these variables automatically whenever you start these tools. On Solaris or Linux, you must first set these environment variables manually by running the **wrenv** script. You must also run this script on a Windows computer if you use a shell other than the VxWorks development shell.

If the environment variables are not set automatically, you must run the script before using:

- the **vxprj** command-line VxWorks configuration and build tool
- the compilers
- the Wind River makefile system from the command line
- the VxWorks Simulator
- the Wind River host shell from the command-line
- other tools

For information about setting environment variables with the **wrenv** script, see the *VxWorks Command-Line Tools User's Guide* and *4.3 Setting Environment Variables*, p.29.

2.3 Wind River Workbench

Wind River Workbench is an Eclipse-based development suite that facilitates creating and building projects, establishing and managing host-target communication, and running, debugging, and monitoring VxWorks applications.

Creating Projects and Developing Source Code

Workbench includes a variety of preconfigured project types for which it provides full build support, as well as a user-defined project that uses your existing makefiles.

To assist you while writing code, the Workbench Editor provides code templates, parameter hints, and code completion suggestions. It also supports planting breakpoints directly on the line you are interested in.

The Workbench search tool allows you to search your code for normal text strings as well as regular expressions. You can then filter the matches according to location context and replace or restore text as necessary.

Parsing Source Files for Symbol Information

Workbench uses static analysis—the parsing of source code symbol information—as the basis for features you see in the Editor such as multilanguage syntax highlighting and definition/declaration navigation.

In addition, static analysis produces the data for features such as include browsing and call trees, as well as information used by the compiler to resolve include search paths. Once you configure your static analysis preferences and parse the source code of your project, you can share those settings and the generated data with your team.

Debugging Applications

Since debugging often requires you to repeatedly launch the same application on the same target, Workbench allows you to configure that information once and then relaunch your program (and even attach the debugger) with the click of a button.

When you launch processes under debugger control or attach the debugger to a running process, those processes appear in the Debug view where you can monitor and control them. Even when the debugger is not able to display the source code of your project in the Editor (such as when your code calls external libraries or the code was compiled without debug information) you can still examine it using the Disassembly view.

Integrating with Eclipse

Because Workbench is based on Eclipse, you can integrate third-party and open source plug-ins into the editing, parsing, and debugging tools provided by Workbench itself. In addition, you can integrate Workbench as a plug-in into an existing Eclipse installation.

Starting Workbench

To start Workbench on Windows, select **Start > All Programs > Wind River > Workbench 3.*x* > Wind River Workbench 3.*x*** (if you chose the default program group during installation).

To start Workbench on Solaris or Linux, navigate to your Workbench installation directory and type the following at the command line:

```
% ./startWorkbench.sh
```

Learning More about Workbench

When you start Workbench, by default, a welcome page appears. (You may first see a dialog box that prompts you to enter the path to your workspace.) The welcome page contains links to Workbench tutorials, Wind River Online Support, and so forth. To access this welcome page after the first time you start Workbench, select **Help > Welcome**.

For introductory information and a tutorial, see the *Wind River Workbench User's Guide*.

For instructions on creating and using projects, see *Wind River Workbench User's Guide: Projects Overview*. For guidelines on configuring launches and debugging projects, see *Wind River Workbench User's Guide, Part V: Debugging*.

Additional tutorials for this release are available on the Online Support Web site:

http://www.windriver.com/support

2.3.1 Wind River System Viewer

Wind River System Viewer is a logic analyzer for embedded software that lets you visualize and troubleshoot complex target multitasking activities.

Often the interactions between the operating system, the application, and the target hardware occur within specified time constraints, characterized by resolutions of microseconds or finer.

Commonly used debugging and benchmarking tools for embedded systems, such as source-level debuggers and profilers, provide only static information.

System Viewer logs activities on a running target; the type of data and aspects of a system that you want to view are highly configurable, and can be saved for later analysis.

Wind River System Viewer provides the ability to do the following:

- Detect race conditions, deadlocks, CPU starvation, and other problems relating to task interaction.
- Determine application responsiveness and performance.
- See cyclic patterns in application behavior.
- Save data for deferred analysis.

For more information on Wind River System Viewer, see the *Wind River System Viewer User's Guide*.

2.3.2 **Wind River Run-Time Analysis Tools**

Wind River Run-Time Analysis Tools is a set of real-time software debugging tools that enable data collection and analysis on a running program. The tool capabilities include memory usage analysis, call stack tracing and profiling, program variable tracking, and test coverage.

Wind River Run-Time Analysis Tools were previously called Wind River ScopeTools. The following table lists the old name and the new name of each of the debugging tools.

Old Name	**New Name**
Wind River MemScope	Wind River Memory Analyzer
Wind River ProfileScope	Wind River Performance Profiler
Wind River StethoScope	Wind River Data Monitor
Wind River CoverageScope	Wind River Code Coverage Analyzer
Wind River TraceScope	Wind River Function Tracer

Wind River General Purpose Platform, VxWorks Edition includes the following Wind River Run-Time Analysis Tools:

- Wind River Memory Analyzer, an online memory analyzer.
- Wind River Performance Profiler, a statistical profiler.
- Wind River Data Monitor, a real-time data monitor.

And optionally includes:

- Wind River Code Coverage Analyzer, a code coverage analysis tool.
- Wind River Function Tracer, an execution-flow trace tool.

For more information on Wind River Run-Time Analysis Tools, see the following guides:

- *Wind River Memory Analyzer User's Guide*
- *Wind River Performance Profiler User's Guide*
- *Wind River Data Monitor User's Guide*
- *Wind River Code Coverage Analyzer User's Guide*
- *Wind River Function Tracer User's Guide*

2.4 Command-Line Development Tools

Wind River provides command-line development facilities for configuring and building the VxWorks operating system, for developing kernel applications, real-time process (RTP) applications, static shared libraries, user-mode shared libraries, and so on. These facilities include a default makefile system, the **vxprj** VxWorks configuration and build tool, compilers, and other utilities. For information about these facilities and examples of their use, see the following:

- *VxWorks Command-Line Tools User's Guide*
- the Wind River compiler guides
- the GNU compiler and tools guides
- the VxWorks programmer and developer guides
- *Wind River Host Utilities API Reference*

NOTE: The prescribed methods for configuring and building VxWorks and VxWorks-based products and applications are Workbench and the command-line tool **vxprj**. The legacy *bspDir*/**config.h** method has been officially deprecated for application development and for most build and configuration scenarios. However, this method may be required for low-level BSP and device driver development and for build and configuration of certain technologies. For more information on BSP and device driver development, see the *VxWorks BSP Developer's Guide* or the appropriate volume of the *VxWorks Device Driver Developer's Guide*. For product build and configuration information, see the appropriate programmer's guide.

2.5 Shells

Wind River General Purpose Platform, VxWorks Edition provides a kernel shell and a host shell for managing, monitoring, and debugging VxWorks systems. These shells can also be used for downloading and running kernel application modules and for executing RTP (user-mode) applications.

Both shells provide more than one interpreter, with different sets of commands or APIs for each. The kernel shell supports a C interpreter and a command interpreter; in addition to these, the host shell supports a Tcl interpreter and a GDB interpreter.

In addition to the kernel and host shells discussed below, Wind River General Purpose Platform, VxWorks Edition also contains the VxWorks development shell (only on Windows platforms). This shell is an ordinary command prompt, but it automatically runs the **wrenv** environment variable script when you start it. To access this shell, select **Start > All Programs > Wind River > VxWorks 6.*x* and General Purpose Technologies > VxWorks Development Shell**.

Kernel Shell

The kernel shell is a target-resident shell that is accessed from a host system over a serial connection, independent of Workbench or other Wind River host facilities. It is therefore suitable for deployed systems as well as for development. For more information on the kernel shell, see the *VxWorks Kernel Programmer's Guide*. For more information on kernel shell commands, see the **usrLib** section of the *VxWorks Kernel API Reference* and the *VxWorks Kernel Shell Command Reference*.

VxWorks can be configured with various components that provide additional commands for the kernel shell. For example, components can be added for semaphore show routines, for commands used to configure the Wind River Network Stack, and so on.

For information on configuring a VxWorks Image Project with shell commands, see *5.4.3 IPCOM Shell Commands*, p.50. For information on accessing shell commands from the kernel shell, see *Running a Shell Command*, p.51.

For information about the shell commands available for a specific technology, and the VxWorks components that provide them, see the appropriate programmer's guide.

To access the kernel shell, right-click a target in the **Remote Systems Explorer** and select **Connect '***target_name***'**. A target shell window opens.

Host Shell

The Wind River Workbench host shell can be started from within Workbench or from the command-line. It communicates with a target system by way of a target server on the host, and the WDB target agent on the target. For more information on the host shell, see the *Wind River Workbench Host Shell User's Guide*. For more information on the host shell commands, see the *Wind River Host Shell API Reference*.

To access the host shell from Workbench, select **Project > Open Workbench Development Shell**.

2.6 Compilers

Wind River General Purpose Platform, VxWorks Edition includes two compilers for C and C++ development: the Wind River Compiler and the Wind River GNU Compiler.

The Wind River Compiler is a complete toolkit for embedded application development, including C and C++ compilers, assemblers, linkers, utilities, and standard libraries for a variety of target CPU architectures.

The Wind River GNU Compiler comprises C and C++ compilers, linker, assembler, and utilities.

For detailed information on these compilers and using them from the command line, see the compiler documentation.

2.7 Wind River VxWorks Simulator

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 before target hardware is available. The VxWorks simulator also allows you to set up a simulated target network for developing and testing complex networking applications. The VxWorks simulator can be used with Wind River Workbench or from the command line.

For information on using the VxWorks simulator, see the *Wind River VxWorks Simulator User's Guide*.

2.8 Wind River Wireless Ethernet Drivers

The Broadcom wireless driver, which is part of Wind River Wireless Ethernet Drivers, is subject to a separate software license agreement and is not available with your installation. To review the software license agreement and download the Broadcom wireless driver, please visit the Wind River Online Support Web site:

https://portal.windriver.com/windsurf/products/netprods/WRWirelessDriv ers

2.9 Setting up Target Hardware

In addition to the Wind River VxWorks Simulator, Wind River provides board support packages (BSPs) for a number of available target hardware reference boards. These BSPs allow you to get your system up and running quickly on reference hardware and can also serve as a base for developing your own custom hardware solution. For more information on developing BSPs, see the *VxWorks BSP Developer's Guide*.

For information on starting development with target hardware, see *Wind River Workbench User's Guide: Setting Up Your Hardware*.

Including Files in a VxWorks Image at Build Time

You can use the read-only memory file system (ROMFS) to include files in a VxWorks Image Project. These files will automatically be loaded into the target system when it boots.

To include a file in your VxWorks Image Project, follow this procedure:

1. Include the component **INCLUDE_ROMFS** in the VxWorks Image Project. This component creates a directory called **romfs** in your BSP directory .
2. Place the file you want to load to the target in the **romfs** directory.
3. Build the image.
4. Boot the target from the image.
5. Open the **/romfs** directory on the target system. The file you stored in *bsp*/**romfs** on the host system will be there.

Copying Files from Development Host to Target System at Run Time

There are three ways to transfer files from the development host to the target system:

- Using FTP
- Using the development file system
- Using the target server file system

Using FTP

Include the following components in the VxWorks image:

- FTP server component (**INCLUDE_IPFTPS**)
- FTP client component (**INCLUDE_IPFTPC**)
- FTP shell command (**IPFTP_CMD**)

Then build your project and boot the target from the resulting image. You can then FTP the file from the host to the target. To access FTP initially, use the default username (**ftp**) and the default password (**interpeak**).

Using the Development File System

Mount the development file system as a netDrv device, using the component **INCLUDE_NET_DRV**. You can then use the host shell or target shell to copy the file.

Using the Target Server File System

Use the target server file system and the **copy( )** command. For example:

```
copy "/tgtsvr/myfile" "location_on_target"
```

3 Getting Started with Development

3.1 Introduction

Before you begin the development process, you should be familiar with the development process, the development environment, and how to create VxWorks projects.

For an overview of the development process and the project workflow, see *1.2 Project Development Workflow*, p.2. For more information on Wind River General Purpose Platform, VxWorks Edition development environment and tools, see *2. Development Environment*.

This chapter describes some of the typical development tasks and provides cross-references to documents containing more detailed information.

3.2 Configuring a VxWorks System

Wind River General Purpose Platform, VxWorks Edition provides default VxWorks images and boot loaders that you can use to start development. However, if the default image does not match your initial development needs (for example, you need a different device driver), you may need to reconfigure and rebuild VxWorks or the boot loader.

Modular Configuration

VxWorks is a highly scalable operating system that can be configured in a multitude of ways, from a minimal kernel of less than 100 KB to a full-featured operating system that includes MMU-based memory protection, local file-systems, networking facilities, process-based RTP applications, POSIX support, and many other facilities.

VxWorks operating system facilities are provided as modular components. Some facilities are specifically designed for the development environment (such as target-side support for host tools and show routines for shell use), that you remove when you proceed to final product testing and deployment. Some facilities also provide alternate configurations for development and for deployment (such as the error detection and reporting facilities).

Configuration Tools

Both VxWorks and the boot loader can be configured and built using either Wind River Workbench or the **vxprj** command-line facility. These tools are the prescribed methods for configuring and building VxWorks.

NOTE: The prescribed methods for configuring and building VxWorks and VxWorks-based products and applications are Workbench and the command-line tool **vxprj**. The legacy *bspDir*/**config.h** method has been officially deprecated for application development and for most build and configuration scenarios. However, this method may be required for low-level BSP and device driver development and for build and configuration of certain technologies. For more information on BSP and device driver development, see the *VxWorks BSP Developer's Guide* or the appropriate volume of the *VxWorks Device Driver Developer's Guide*. For product build and configuration information, see the appropriate programmer's guide.

Wind River Workbench displays descriptions of components and parameters, as well as their names, in the **Components** tab of the **Kernel Configuration Editor**. To access the **Kernel Configuration Editor**, double-click the **Kernel Editor** node in your project tree in the **Project Explorer**. You can use the **Find** dialog to locate a component or parameter using its name or description. To access the **Find** dialog from the **Components** tab, type **CTRL+F**, or right-click and select **Find**.

Throughout the documentation, VxWorks components and their configuration parameters are identified by the names used in component description files, which take the form, for example, of **INCLUDE_***FOO* and **NUM_***FOO***_FILES** (for components and parameters, respectively). In Workbench, you can search for these names using the **Find** dialog. For command-line configuration facilities, use these names directly to configure VxWorks. For more information, on command-line facilities, see *2. Development Environment*.

References

For information about VxWorks and the associated technologies with which it can be built, see the following:

- *VxWorks Kernel Programmer's Guide*
- *VxWorks Application Programmer's Guide*
- the programmer guides for networking and middleware technologies

For information about the configuration tools, see the following:

- *Wind River Workbench User's Guide*
- *VxWorks Command-Line Tools User's Guide*

3.3 Developing Kernel Applications

VxWorks applications that execute in the kernel are created as relocatable object modules. Kernel-based application modules can be either downloaded and dynamically linked to the operating system by the object module loader, or statically linked to the operating system, making them part of the system image. Kernel applications run in kernel mode, and there is no memory protection between kernel applications or between kernel applications and the kernel itself. Kernel applications, however, have direct access to public kernel APIs and to hardware, and therefore require no system calls for their operation.

For information about the features and tools used to develop, execute, and debug kernel applications, see the following:

- *VxWorks Kernel Programmer's Guide*
- *Wind River Workbench User's Guide*
- *VxWorks Command-Line Tools User's Guide*

For APIs available for kernel applications, see the following:

- *VxWorks Kernel API Reference* for native VxWorks C libraries
- *Dinkum C++ Library Reference Manual* for C++ libraries
- *Dinkum EC++ Library* for C++ libraries

3.4 Developing RTP Applications

VxWorks Real-Time Process (RTP) applications execute in user mode, in memory spaces separate from the kernel and from other RTP applications. On systems with an MMU, RTP applications and the kernel are all protected from one another. VxWorks real-time processes are in many respects similar to processes in other operating systems—such as UNIX and Linux—including extensive POSIX compliance. However, RTPs are specifically designed for hard real-time systems. RTP applications can be built separately from the operating system and stored on a host or target file system; or bundled with the system image using the ROMFS file system. RTP applications make system calls for kernel services.

RTP applications can be designed and built for use with shared libraries and shared data regions. This allows them to share code and reduce the footprint of the applications, and to provide a shared region for applications that are otherwise separated by a memory barrier.

For information about the features and tools used to develop, execute, and debug RTP applications, see the following:

- *VxWorks Application Programmer's Guide*
- *Wind River Workbench User's Guide*
- *VxWorks Command-Line Tools User's Guide*

For APIs available for RTP applications, see the following:

- *VxWorks Application API Reference* for native VxWorks C libraries
- *Dinkum C++ Library Reference Manual* for C++ libraries
- *Dinkum EC++ Library* for C++ libraries

3.5 Developing BSPs

The *VxWorks BSP Developer's Guide* discusses VxWorks BSP development. In particular, it provides guidelines for writing a custom BSP based on an existing reference BSP. This includes information on configuring your development environment, accessing minimal hardware or a hardware simulator, creating a minimal kernel, adding VxWorks device drivers, and other clean-up tasks.

For more information on developing and using drivers with your BSP, see the appropriate volume of the *VxWorks Device Driver Developer's Guide*.

3.6 Developing Drivers

At a high level, VxWorks device drivers allow for communication between your VxWorks system and specific target hardware. If you are developing for custom hardware, you must provide a device driver for that hardware so that it can be included in your VxWorks system.

Before beginning any device driver development, you should have a good understanding of the overall VxWorks I/O system. (For more information, see the *VxWorks Kernel Programmer's Guide*.)

In recent VxWorks releases, device driver development is centered around the VxBus driver infrastructure. The VxBus infrastructure supports device drivers by defining interfaces that device drivers use to interact with the hardware and with the operating system. VxBus-enabled drivers must be used when working with symmetric multi-processing (SMP) systems and are optional (but recommended) for uniprocessor (UP) systems.

General driver development and the VxBus infrastructure are discussed in the following documents:

- *VxWorks Device Driver Developer's Guide, Volume 1: Fundamentals of Writing Device Drivers* discusses the VxBus infrastructure and the basic steps involved in writing a VxBus-enabled device driver.
- *VxWorks Device Driver Developer's Guide, Volume 2: Writing Class-Specific Device Drivers* discusses the specific requirements of the many driver classes supported by VxBus (network drivers, timer drivers, bus controller drivers, and so forth).

Wind River encourages developers to use the VxBus infrastructure for device driver development whenever possible. The ad-hoc (or legacy) driver model used in older VxWorks releases continues to be supported for uniprocessor systems. Legacy driver information (including migration to VxBus) is provided in *VxWorks Device Driver Developer's Guide, Volume 3: Legacy Drivers and Migration*.

In addition to the device driver guides, you may want to reference the *VxWorks BSP Developer's Guide*. This document discusses VxWorks BSP development. In particular, it provides guidelines for writing a custom BSP based on an existing reference BSP.

4 Building VxWorks Source Code

4.1 Introduction

In most cases, it is not necessary to compile the source code for the VxWorks operating system. However, you can recompile this code if your development situation requires it, for example, if you want to produce SMP-compatible archives, change the IP version of the network stack, or create archives without System Viewer instrumentation. The output of this build is a set of run-time libraries that you can use to create a VxWorks Image Project.

There are some limitations and restrictions you must be aware of when recompiling the VxWorks source:

- The source code must be built from the **vxworks-6.*x*** installation tree with the host tools and makefiles provided.
- There may be certain portions of the VxWorks object code for which source code has not been provided.
- For the unmodified source code that is included on the CD, the resulting binaries built using the Wind River Compiler should match the binaries distributed on the CD.
- Modifications to the source code (when permitted) may not be covered by Wind River Customer Support.

You can build the VxWorks source using Workbench or from the command line.

4.1.1 Precompiled Binary Files

Due to licensing restrictions, the VxWorks source code does not include source files licensed by Wind River from third parties. This code is instead distributed in the form of precompiled binary files with a **.o** extension.

When you build the source code using the procedure documented in this chapter, the relevant **.o** files are copied into the source build directory.

Precompiled **.o** files are located in *installDir***/vxworks-6.*x*/target/precomp**. For a complete list of the affected files and architectures, search this directory for files with a **.o** extension.

⚠ **CAUTION:** Do not delete the precompiled files in this directory. Doing so may prevent you from creating projects that rely on these files.

4.2 Back up VxWorks Archives

If you want to preserve the original VxWorks archives before recompiling the source code, back them up using the following procedure. Making a backup copy of these archives is advisable in case you later decide to use them again.

In a host shell, go to the library subdirectory of the target directory. Make a copy of the **vxworks-6.*x*** archive directory and files for the architecture you want to rebuild. For example, to back up your PowerPC files, type:

```
% cd installDir/vxworks-6.x/target/lib
% mkdir ppcBackup
% cp -r ppc ppcBackup/ppc
% cp libPPC*.a ppcBackup
% cp -r objPPC* ppcBackup
```

4.3 Setting Environment Variables

To use the command-line interface on the host computer to build source code or develop your project, you must configure some environment variables and other settings. The method you chose depends chiefly on your host computer's operating system.

Windows

If you run Workbench on a Windows computer, the environment variables are set automatically.

If you use the VxWorks development shell, they are also set automatically. To open this shell, select **Start > All Programs > Wind River > VxWorks 6.*x* and General Purpose Technologies > VxWorks Development Shell**.

Solaris and Linux

On Solaris and Linux, you must set the environment variables whether you use Workbench or the command line. Use the **wrenv** environment utility for this purpose.

Open a shell and set your environment to access the Wind River host tools and Wind River Compiler.

In your installation directory, run the following command:

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

NOTE: If your shell configuration file (**.profile**, **.cshrc**, **.tcshrc**, and so forth) overwrites the environment each time a new shell is created, the above command may not work. To test whether the environment variables have been set successfully, start Workbench. If you find that you cannot start the Workbench tools after running the above command, use the following command:

```
% eval `installDir/wrenv.sh -p platform -o print_env -f shell`
```

where *shell* is **sh** or **csh**, depending on the current shell program. For example:

```
% eval `./wrenv.sh -p vxworks-6.x -o print_env -f sh`
```

For more information about the **wrenv** utility, see the *VxWorks Command-Line Tools User's Guide*.

4.4 Building VxWorks Source Code—Workbench Procedure

This section describes how to build the VxWorks OS source using Workbench.

Step 1: Set your command environment (Solaris or Linux platform).

If you are using a Solaris or Linux platform, you must first set the appropriate environment variables as documented in *4.3 Setting Environment Variables*, p.29.

If you are using Windows, environment variables are automatically set when you start Workbench.

Step 2: Start Workbench.

To start Workbench on Windows, select **Start > Programs > Wind River > Workbench 3.*x* > Wind River Workbench 3.*x*** (if you chose the default program group during installation).

To start Workbench on Solaris or Linux, navigate to your Workbench installation directory and type the following at the command line:

```
% ./startWorkbench.sh
```

Step 3: Create a user-defined project.

1. Open Workbench and select **File > New > User-Defined Project**.
2. From the **Target Operating System** list, select **Wind River VxWorks 6.***x*, then click **Next**.
3. Provide a name for this project.
4. Select **Create project at external location**.
5. Click **Browse**, then navigate to (or enter):

   *installDir***/vxworks-6.***x***/target/src**

   NOTE: For user builds, navigate to *installDir***/vxworks-6.***x***/target/usr/src**.

   Click **OK** to close the **Select folder** dialog box.
6. Click **Next** twice. The **Build Support** page opens.
7. Ensure that **User-defined build** option is selected, then enter the following as your **Build Command:**

   ```
   make CPU=cpuType TOOL=toolChain
   ```

   For example:

   ```
   make CPU=PPC32 TOOL=diab
   ```

   NOTE: To identify your CPU and associated primary compiler, see *4.8 Supported CPU and TOOL Values*, p.37.
8. Click **Finish**.

Step 4: Rebuild the VxWorks source.

In the **Project Explorer**, right-click your project and select **Build Project**.

Step 5: Rebuild with a secondary compiler (optional).

If you intend to build your project with the GNU GCC compiler, you must rebuild the project with a secondary compiler listed in *4.8 Supported CPU and TOOL Values*, p.37.

NOTE: If you are building the user-side source—that is, the source code in *installDir***/vxworks-6.***x***/target/usr/src**—you do not need to perform a rebuild with the secondary compiler. Only the Wind River Compiler is supported for user-side builds.

You can change the build command by changing the values of the user build arguments.

1. In the **User Build Arguments** field in the the **Build Console**, add the **TOOL** argument for the secondary compiler. The information you type here overrides the **Tool chain** argument in the original project properties. For example, if you enter **TOOL=gnu** as a user build argument, the build command changes from:

   ```
   make -f Makefile CPU=PENTIUM2 TOOL=diab
   ```

   to

   ```
   make -f Makefile CPU=PENTIUM2 TOOL=gnu
   ```

2. In the **Project Explorer** toolbar, click the (**Build**) button to start building the project for your CPU and associated secondary compiler.

After a successful build with the secondary compiler, clear the **User Build Arguments** field to revert to the primary compiler value.

Additional options are available. See *4.6 Source Build Options*, p.34, for further information.

4.5 Building VxWorks Source Code—Command-Line Procedure

This section describes how to build the VxWorks OS source from the command line.

Step 6: Set your command environment.

Before you build the components, you must first set the appropriate environment variables as documented in *4.3 Setting Environment Variables*, p.29.

Step 7: Disable the GNU dependency (optional).

If you want to disable the GNU dependency in any BSP, modify the **config.h** file for the BSP to include the following line:

```
#undef INCLUDE_GNU_INTRINSICS
```

This file is located in *installDir*/**vxworks-6.***x*/**target/config/***bsp*.

The GNU intrinsics are required to be able to load C modules built with one compiler into an image built with another. Do not disable the GNU dependency if you plan to load C modules built with another compiler into your image.

Step 8: Build the source code.

Go to *installDir*/**vxworks-6.***x***/target/src** and start the build by running **make** to build the sources for your **CPU** and **TOOL**.

The syntax for the **make** command is:

```
% cd installDir/vxworks-6.x/target/src
% make CPU=cpuName TOOL=primaryCompilerName [other_build_options]
```

If you have **INCLUDE_GCC_INTRINSICS** in your project (or intend to build your project with the GNU GCC compiler), you need to run **make** twice for each CPU—once for the primary compiler and once for the secondary compiler. To identify your CPU and associated primary and secondary compilers, see *Supported CPU and TOOL Values*, p.37.

NOTE: If you do not need **INCLUDE_GCC_INTRINSICS** (which allows you to load **gnu**-built objects onto a **diab**-built image) or if you are performing a user-side build, do not run the secondary build.

To build for your secondary compiler, the syntax is the same as for the primary compiler. For example, run both of the following:

```
% make CPU=PPC32 TOOL=diab
% make CPU=PPC32 TOOL=gnu
```

NOTE: You can build a debug version of the source by providing a **-g** flag with **ADDED_CFLAGS** and **ADDED_C++FLAGS** in the following file:

installDir/**vxworks-6.***x***/target/src/Makefile**

For example:

```
% make CPU=cpuVal TOOL=toolChain ADDED_CFLAGS+=-g ADDED_C++FLAGS+=-g
```

Additional options are available. See *4.6 Source Build Options*, p.34 for further information.

4.6 Source Build Options

Whether you use Workbench or a command-line interface to build the VxWorks source, additional build options are available. These include:

- *Building SMP-Compatible Archives*, p.34
- *Building IPv6 Network Stack Archives*, p.34
- *Building Archives without System Viewer Instrumentation*, p.36

Building SMP-Compatible Archives

If you have purchased the SMP option, you can build SMP-compatible archives.

To build the source code for SMP, add **VXBUILD=SMP** to the **make** command. For example:

```
make CPU=cpuType TOOL=toolChain VXBUILD=SMP
```

Building IPv6 Network Stack Archives

In Wind River General Purpose Platform, VxWorks Edition, the source code libraries are prebuilt for an IPv4-only network stack. If you recompile the VxWorks source code, VxWorks builds only the IPv4 network stack archives by default. There are two ways to include an alternate IPv4/IPv6 network stack archive:

- by replacing the precompiled network stack libraries with dual IPv4/IPv6 libraries
- by generating a separate set of IPv6 libraries

You can also build a network stack that only supports IPv6.

The following sections provide additional details.

Replacing the Precompiled Network Stack Libraries

To replace the IPv4 library with an IPv4/IPv6 library, add an **ADDED_CFLAGS+=-DINET6** command-line flag. The **make** command then has the following form:

```
% make CPU=cpuType TOOL=toolChain ADDED_CFLAGS+=-DINET6
```

If you are using Workbench, add **ADDED_CFLAGS+=-DINET6** to the **User Build Arguments** field in the **Build Console** and build your source project.

This procedure overwrites the existing network stack libraries (**libnetcommon.a**, **libnetapps.a**, and **libnetwrap.a**). The libraries generated using this procedure support a dual IPv4/IPv6 network stack.

NOTE: When you build a dual IPv4/IPv6 library with the **ADDED_CFLAGS+=-DINET6** flag, the DHCP components for IPv6 are also built by default.

Generating a Separate Set of Libraries

To create an alternate set of IPv4/IPv6 network stack libraries while preserving the precompiled libraries, append **OPT=-inet6** to the **make** command. The **make** command then has the following form:

```
% make CPU=cpuType TOOL=toolChain OPT=-inet6
```

If you are using Workbench, add **OPT=-inet6** to the **User Build Arguments** field in the **Build Console** and build your source project.

Using this procedure preserves the precompiled IPv4 libraries while generating a separate set of IPv6 libraries (**libnetcommon-inet6.a**, **libnetapps-inet6.a**, and **libnetwrap-inet6.a**).

If you build separate IPv6 network stack libraries using this technique, you must also select the appropriate kernel libraries when you create a VxWorks Image Project. To do so, select the option **Use IPv6 enabled kernel libraries** in the **Options** page, which appears when you create a VxWorks Image Project.

NOTE: The build process may produce several build warnings, which you can ignore.

For more information about the IPv4 and IPv6 network stacks, see the network stack documentation set.

Building an IPv6-only Network Stack

To rebuild the source code libraries with support for an IPv6-only network stack, you must issue a **make** command in the following directory:

```
installDir/vxworks-6.x/target/src/ipnet
```

Use the **ADDED_CFLAGS+=-DINET6_ONLY** command-line flag:

```
make CPU=cpuType TOOL=toolChain ADDED_CFLAGS+=-DINET6_ONLY
```

Affected Modules—IPv6-Only Network Stack

Most code modules are unaffected by the way the network stack source code is built. If you build an IPv6-only network stack, however, modifications may be required in modules that make calls to IPv4 routines. Such modules include SNMP and BSPs.

To make these modules compatible with an IPv6-only network stack, perform the following steps:

- Enclose any IPv4-specific code with **#ifdef INET**.
- Enclose any IPv6-specific code fragment with **#ifdef INET6**.

Symbol Table Download and Network Drives

Symbol table download and network drive mounting are ordinarily performed over an IPv4 network. Special provisions are required when you build an IPv6-only network stack. For further information, see *Wind River Network Stack Programmer's Guide, Vol. 1: Configuring and Building the Network Stack*.

Building Archives without System Viewer Instrumentation

Specifying the **Use System Viewer free kernel libraries** option when creating a VxWorks Image Project creates a project that builds a VxWorks image without System Viewer instrumentation.

However, for this to work, you must have kernel source and must have previously compiled the kernel using the **OPT=-fr** option build (or **OPT=-inet6_fr**, if you want IPv6 support). This option places the System Viewer-free archives in an alternate location (for example, **libwind-fr.a** instead of **libwind.a**, **libnet-fr.a** instead of **libnet.a**, and so on).

The free build, in addition to automatically undefining **WV_INSTRUMENTATION** and **INCLUDE_WVNETD**, also defines a macro **_FREE_VERSION** that causes omission of some debug and sanity checks in performance-critical code (primarily in the networking code).

To build the kernel without System Viewer instrumentation using Workbench, add **OPT=-fr** or **OPT=-inet6_fr** to the **User Build Arguments** in the **Build Console** field and build your source project.

If you build the kernel without System Viewer instrumentation, you must also select the appropriate kernel libraries when you create a VxWorks Image Project.

To do so, select the option **Use System Viewer free kernel libraries** in the **Options** page, which appears when you create a VxWorks Image Project.

4.7 Restoring Original Archives and Object Directories

In the VxWorks development shell, go to the library subdirectory of the target directory. Move the recompiled version of the **vxworks-6.***x* archive directory by renaming it, and then restore the original archive directory from the copy you made in *4.2 Back up VxWorks Archives*, p.28.

For example:

```
% cd installDir/vxworks-6.x/target/lib
% mv ppc ppcRef
% mv ppcBackup/ppc ppc
% mv ppcBackup/libPPC*.a .
% cp -rf ppcBackup/objPPC* .
```

In this example, the **ppcRef** and the original files are now located in *installDir***/vxworks-6.***x***/target/lib**.

4.8 Supported CPU and TOOL Values

The source tree build system has been designed and tested to compile source code with a primary compiler only. The secondary compiler is provided for the application level only, but some run-time support is required. Therefore, the source tree build system builds only the directories necessary to support the secondary compiler.

NOTE: User (RTP) builds support the CPU options listed in Table 4-1. However, only the Wind River Compiler (**diab**) is supported.

Table 4-1 lists the primary and secondary compilers for each architecture. In most cases, the CPU values in this table can be used in **make** commands. In a few cases, however, using a variant architecture from the same family (such as PPC32 instead of PPC405) may provide better results. To verify the **CPU** argument in a **make** command and the make variable settings, consult one or both of the following sources:

- *VxWorks Architecture Supplement*
- *installDir*/**vxworks-6.***x*/**target/config/***bsp*/**Makefile**

NOTE: For **SIMNT**, **SIMSPARCSOLARIS**, and **SIMLINUX**, the **libprocfs.a**, **libtffs.a**, **libusb.a**, and **libusb2.a** libraries are not supported. Although they are built from source, they are not present on the product CD.

Table 4-1 **CPU and TOOL Values by Architecture**

		TOOL	
Architecture	**CPU**	**Primary Compiler**	**Secondary Compiler**
ARM Architecture Version 4 Processors	ARMARCH4	diab	gnu
ARM Architecture Version 4 Processors (big-endian)	ARMARCH4	diabbe	gnube
ARM Architecture Version 5 Processors	ARMARCH5	diab	gnu
ARM Architecture Version 5 Processors (big-endian)	ARMARCH5	diabbe	gnube
ARM Architecture Version 6 Processors	ARMARCH6	diab	gnu
ARM Architecture Version 6 Processors (big-endian)	ARMARCH6	diabbe	gnube

Table 4-1 **CPU and TOOL Values by Architecture** (cont'd)

		TOOL	
Architecture	**CPU**	**Primary Compiler**	**Secondary Compiler**
Cavium cn3xxx	MIPSI64R2	diab	gnu
Cavium cn3xxx (little-endian)	MIPSI64R2	diable	gnule
Cavium cn3xxx (software floating point)	MIPSI64R2	sfdiab	sfgnu
Cavium cn3xxx (software floating point, little-endian)	MIPSI64R2	sfdiable	sfgnule
ColdFire 5200	MCF 5200	diab	Not supported
ColdFire 5400	MCF 5400	diab	Not supported
ColdFire 5400 (software floating point)	MCF 5400	sfdiab	Not supported
Intel XScale	XSCALE	diab	gnu
Intel XScale (big-endian)	XSCALE	diabbe	gnube
MIPS32 (software floating point)	MIPS32	sfdiab	sfgnu
MIPS32 LE (software floating point)	MIPS32	sfdiable	sfgnule
MIPS64	MIPS64	diab	gnu
MIPS64 LE	MIPS64	diable	gnule
MTI 4kc	MIPSI32	diab	gnu
MTI 4kc (little-endian)	MIPSI32	diable	gnule

Table 4-1 **CPU and TOOL Values by Architecture** (cont'd)

		TOOL	
Architecture	**CPU**	**Primary Compiler**	**Secondary Compiler**
MTI 4kc (software floating point)	MIPSI32	sfdiab	sfgnu
MTI 4kc (software floating point, little-endian)	MIPSI32	sfdiable	sfgnule
MTI 4kec, 24kc, 24kf, 74kc, 74kf	MIPSI32R2	diab	gnu
MTI 4kec, 24kc, 24kf, 74kc, 74kf (little-endian)	MIPSI32R2	diable	gnule
MTI 4kec, 24kc, 24kf, 74kc, 74kf (software floating point)	MIPSI32R2	sfdiab	sfgnu
MTI 4kec, 24kc, 24kf, 74kc, 74kf (software floating point, little-endian)	MIPSI32R2	sfdiable	sfgnule
MTI 5kc, 5kf, Broadcom 1250, 1480, Raza XLR	MIPSI64	diab	gnu
MTI 5kc, 5kf, Broadcom 1250, 1480, Raza XLR (little-endian)	MIPSI64	diable	gnule
MTI 5kc, 5kf, Broadcom 1250, 1480, Raza XLR (software floating point)	MIPSI64	sfdiab	sfgnu

Table 4-1 **CPU and TOOL Values by Architecture** (cont'd)

		TOOL	
Architecture	**CPU**	**Primary Compiler**	**Secondary Compiler**
MTI 5kc, 5kf, Broadcom 1250, 1480, Raza XLR (software floating point, little-endian)	MIPSI64	sfdiable	sfgnule
Pentium	PENTIUM	diab	gnu
Pentium II	PENTIUM2	diab	gnu
Pentium III	PENTIUM3	diab	gnu
Pentium IV	PENTIUM4	diab	gnu
PowerPC 403	PPC403	diab	gnu
PowerPC 405	PPC405	diab	gnu
PowerPC 405 (software floating point)	PPC405	sfdiab	sfgnu
PowerPC 40x, 440, 8xx, 85xx	PPC32	sfdiab	sfgnu
PowerPC 440	PPC440	diab	gnu
PowerPC 440 (software floating point)	PPC440	sfdiab	sfgnu
PowerPC 603	PPC603	diab	gnu
PowerPC 604	PPC604	diab	gnu
PowerPC 60x, 7xx, 74xx, 82xx	PPC32	diab	gnu
PowerPC 85XX	PPC85XX	diab	gnu

Table 4-1 **CPU and TOOL Values by Architecture** (cont'd)

		TOOL	
Architecture	**CPU**	**Primary Compiler**	**Secondary Compiler**
PowerPC 85XX (software floating point)	PPC85XX	sfdiab	sfgnu
PowerPC 860 (software floating point)	PPC860	sfdiab	sfgnu
SH 7750	SH7750 (SH32 for RTP builds)	diab	gnu
SH 7750 LE	SH7750 (SH32 for RTP builds)	diable	gnule
SIMLINUX	SIMLINUX (SIMPENTIUM for RTP builds)	diab	gnu
SIMNT	SIMNT (SIMPENTIUM for RTP builds)	diab	gnu
SIMSOLARIS	SIMSPARCSOLARIS	diab	gnu
Toshiba tx4938, NEC vr5500, PMC-Sierra rm9000	MIPSI3	diab	gnu
Toshiba tx4938, NEC vr5500, PMC-Sierra rm9000 (little-endian)	MIPSI3	diable	gnule
tx4938 and vr5500 when operated in 32-bit mode (software floating point)	MIPSI2	sfdiab	sfgnu

Table 4-1 **CPU and TOOL Values by Architecture** (cont'd)

		TOOL	
Architecture	**CPU**	**Primary Compiler**	**Secondary Compiler**
tx4938 and vr5500 when operated in 32-bit mode (software floating point, little-endian)	MIPSI2	sfdiable	sfgnule

5
Software Architecture

5.1 **Introduction**

This chapter describes the software architecture of networking and middleware components. It provides information on the software components, or building blocks, of the network stack, the structure of these components, and their application programming interfaces (APIs). It also describes the multiple platform support facility **IPCOM**, which provides an interface between the networking layer and the operating system, and the shell commands used to configure and operate the networking layer and related products.

5.2 Building Blocks

The networking layer consists of separate source code modules, or building blocks, that can be compiled and linked to form a network stack and related applications for the target system. A supporting layer, called **IPCOM**, provides an interface between the networking layer and target operating system.

5.3 Component Structure

Source Code

The source code for the network stack and related applications resides in the following location:

*installDir***/components/ip_net2-6.***x***/***component*

Each component typically has the following structure:

- **config**
  - *feature_name***_config.h**, which specifies the included features and their default values
- **gmake**
  - **Makefile**
  - *feature_name***.mk**
- **include**
  - *feature_name***.h**, which contains the API for the feature
- **src**
  - *feature_name***.c**
  - *feature_name_xxx***.c**
  - *feature_name_xxx***.h**

NOTE: For the **IPCOM** component, public APIs are found only in **ipcom_auth.h**, **ipcom_ipd.h**, **ipcom_syslog.h**, and **ipcom_sysvar.h**. Other header files in **/ipcom/include** are internal and should not be used by applications.

Configuration Code

The directory *installDir***/components/ip_net2-6.***x***/osconfig/vxworks/src/ipnet** contains additional files used for configuration. These **.c** files contain the default values for all components.

Configuration information is also stored in CDF files, which are read by the Workbench **Kernel Configuration Editor** when you configure your project. These files store the names of individual configuration components, ranges of permissible values for each component, and similar information.

Starting a Network Stack Process

Each process in the network stack is started by calls to the following routines:

- *feature_name*_**create( )**. This routine is used to allocate, initialize, and clear memory.
- *feature_name*_**configure( )**. This routine is used to read default feature values and attributes.
- *feature_name*_**start( )**. This routine is used to start the process.

For example, starting a secure shell would require calls to the following routines:

- **ipssh_create( )**
- **ipssh_configure( )**
- **ipssh_start( )**

These routines are called automatically, and you should not include these routines in your application code. If you want to add specific configuration code to a particulare “feature,” or module, however, you can do so in the *feature_name*_**configure( )** routine.

Once a process has been started, component-specific APIs are used for further configuration. See the individual programmer’s guides and the Wind River Workbench online help for information on specific APIs.

5.4 Multiple Platform Support

The **IPCOM** layer provides an interface between the networking layer and the operating system. It ensures complete portability for the network stack and all related products.

IPCOM provides such facilities as services, data structures, and shell commands. It also provides the APIs used to enable and configure the network stack and related applications. See *5.4.1 Functional Specification*, p.48, for more information on the **IPCOM** facilities.

5.4.1 Functional Specification

IPCOM provides the following facilities:

- Network stack initialization
- Authentication (user and password API)
- System variables (**sysvars**) for system configuration (similiar to environment variables)
- Memory file system
- Target shell and Telnet server
- Syslog daemon
- Shell commands (**echo**, **ipd**, **sockperf**, **socktest**, **syslog**, **sysvar**, **tracert**, **ttcp**, **user**, **ipversion**)
- Timeout server
- Pseudorandom daemon implemented by the Entropy Gathering Daemon (EGD)
- Debug utilities: memory statistics (mem)

The following sections provide additional detail.

Initialization of the Network Stack and Related Components

The task is intilized and started from **IPCOM**.

The routine **ipcom_start(void)** (in **ipcom_init.c**) is the first line of code. Use this routine to set a breakpoint just before the network stack is initialized.

Authentication

A user and password API is available. This facility handles all system authentication (Telnet, PPP, etc.).

Sysvars

System variables, or **sysvars**, are used to configure the network stack and related applications. They are similar to environment variables.

System variables are not process-specfic. They are implemented as an ASCII stream and a value—e.g., *value_x* = *y*. You can change the values of **sysvars** at run time using the **sysvar** shell command. For further information, see *System Variables (sysvar)*, p.60.

Shell Commands

IPCOM provides shell commands that can be used to start, stop, or configure the network stack and related applications. These commands are available when you include the kernel shell and the individual command in your VxWorks Image Project. See *5.4.3 IPCOM Shell Commands*, p.50, for further information.

Syslog Daemon

The system logging daemon includes all run-time debugging information. It uses the same debug level as Linux. Debug levels are configured at build time and at run time. At build time, you can specify the debug levels that will be compiled into the module. At run time, you can specify the debug levels that are actually output to the system log. For further information, see *System Log (syslog)*, p.53.

5.4.2 **APIs**

A variety of public APIs are available for use in programming the network stack and related applications. These APIs are typically located in the include directory for each component in the file *feature_name***.h**. For example, the directory *installDir***/components/ip_net2-6.6/ipppp/include** contains the file **ipppp.h**. This file contains the APIs used to enable and configure the Wind River PPP.

For most APIs, documentation is available in individual programmer's guides and the Wind River Workbench online help. To access API documentation in the Wind River Workbench online help, select **Help > Help Contents**. The help contents appear in a Web browser. In the **Contents** pane, search within the **Wind River Documentation > References** tree to locate the APIs that are relevant to your product.

IPCOM Layer APIs

APIs to configure the **IPCOM** layer are defined in the following files in the *installDir*/**components/ip_net2-6.***x***/ipcom/include** directory.

- **ipcom_auth.h**
- **ipcom_ipd.h**
- **ipcom_syslog.h**
- **ipcom_sysvar.h**

The APIs defined in these files are documented in the reference entries for the Wind River Network Stack Kernel API Reference. The rest of the files in the *installDir*/**components/ip_net2-6.***x***/ipcom/include** directory are internal and should not be used by applications.

5.4.3 IPCOM Shell Commands

Including a Shell Command

IPCOM shell commands are available when you include the kernel shell and the individual command in your VxWorks Image Project. To include a kernel shell in your project:

1. Include the component **INCLUDE_USE_NATIVE_SHELL**.
2. Include an individual shell command. These shell commands are found in the **FOLDER_IPCOM_SHELL_CMD** folder, if you are using the Workbench **Kernel Configuration Editor**. Individual commands have names in the form **INCLUDE_***CommandName***_CMD**—for example, **INCLUDE_IPCOM_SYSLOG_CMD**.

   The **IPCOM** shell command interface, **INCLUDE_IPCOM_SHELL_CMD**, is included by default when you include the appropriate shell command.

Running a Shell Command

The shell commands are run from the shell in command-interpreter mode. To run the shell commands:

1. Open a VxWorks kernel shell.
2. At the command prompt, type **cmd** and press **ENTER** to switch to command-interpreter mode. The command prompt changes from **->** to **[vxWorks *] #**.
3. Run the appropriate shell command.

You can also call shell commands from an application. For more information on this technique, see *Calling Shell Commands from an Application*, p.63.

Interpeak Daemon (ipd)

Most networking component initialization routines and daemons are automatically controlled by a central module called **ipd**.

The **ipd** shell command controls other daemons, such as the multicasting proxy daemon. To include this command in your project, include **INCLUDE_IPD_CMD**.

Name

ipd – daemon process command

Synopsis

```
ipd command [ -options ]
```

Individual synopses are as follows:

```
ipd [-V vr] list
ipd [-V vr] start service
ipd [-V vr] kill service
ipd [-V vr] reconfigure service
ipd [-V vr] # service
```

Description

Command options are as follows:

-V *vr*
Use the virtual router identified by *vr*.

list
List daemon services.

start
Start the specified service.

kill
Stop the specified service.

reconfigure
Reconfigure the specified service.

Usage

Service names take the form **IP***service_name*—for example, IPike.

To list the services for IKE, use the following command:

```
[vxWorks *]# ipd list ipike
Services:
ipike                started
```

To stop the IKE service, use an **ipd kill** command:

```
[vxWorks *]# ipd kill ipike
ipd: kill ipike ok
```

To verify that the service has been stopped, use a second **ipd list** command:

```
[vxWorks *]# ipd list ipike
Services:
ipike                killed
```

To restart the service, use the following command:

```
[vxWorks *]# ipd start ipike
ipd: start ipike ok
```

To verify that the service has been restarted, use a third **ipd list** command:

```
[vxWorks *]# ipd list ipike
Services:
ipike                started
```

System Configuration (sysctl)

The **sysctl** command is used to get or set system parameters.

Name

sysctl - Get or set **sysctl** values

Synopsis

```
sysctl -w variable=value
sysctl -a
sysctl variable
```

Description

-a
List all sysctl parameters.

-w *variable*=*value*
Change the value of *variable* to the specified value.

value
Value of a system variable.

System Log (syslog)

The **syslog** command controls a system logging service with eight priority levels.

Setting Priority Levels at Build Time

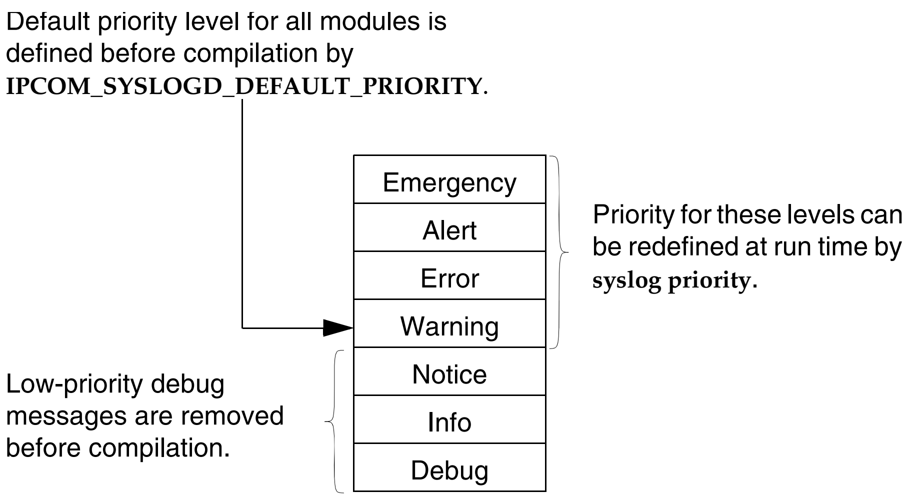
The configuration component **IPCOM_SYSLOGD_DEFAULT_PRIORITY** sets the default debug level for all modules. Options for this variable include:

- **IPCOM_LOG_EMERG**
- **IPCOM_LOG_CRIT**
- **IPCOM_LOG_ERR**
- **IPCOM_LOG_WARNING**
- **IPCOM_LOG_NOTICE**
- **IPCOM_LOG_INFO**
- **IPCOM_LOG_DEBUG**
- **IPCOM_LOG_DEBUG2**

Any level lower than that defined by **IPCOM_SYSLOGD_DEFAULT_PRIORITY** is excluded from the module during compilation. As a result, the system at run time cannot log lower priority levels than those set by this configuration component.

Figure 5-1 illustrates this functionality.

Figure 5-1 **Setting syslog Priority Levels**


Default priority level for all modules is defined before compilation by **IPCOM_SYSLOGD_DEFAULT_PRIORITY**.
Emergency
Alert
Error
Warning
Notice
Info
Debug
Priority for these levels can be redefined at run time by **syslog priority**.
Low-priority debug messages are removed before compilation.

You can also specify individual debug levels for each module. These settings are made in the configuration file *component*_**config.h**. There is one such file for each module, residing in the following location:

installDir/**components/ip_net2-6.***x*/*component*/**config**

For example, the configuration file for the SSH component is:

installDir/**components/ip_net2-6.***x*/**ipssh/config/ipssh_config.h**

Debug levels are specified in the following lines of code:

```
#ifdef IPSSH_DEBUG
#define IPSSH_SYSLOG_PRIORITY IPCOM_LOG_DEBUG
#else
#define IPSSH_SYSLOG_PRIORITY IPCOM_LOG_ERR
#endif
```

Separate levels can be set for debug and optimized builds.

Setting Priority Levels at Run Time

At run time, you can use the **syslog** command to set the priority level.

To include the **syslog** command in your project, include the following components:

- **INCLUDE_IPCOM_SYSLOGD_CMD**
- **INCLUDE_IPCOM_SYSLOGD_USE_LOG_FILE**

Set values for the following variables:

- **IPCOM_SYSLOGD_DEFAULT_PRIORITY**
- **IPCOM_SYSLOGD_QUEUE_MAX**
- **IPCOM_SYSLOGD_LOG_FILE**

Many facilities and components in the network stack generate useful debug information. Debugging is enabled when the stack is compiled with the **IPBUILD=debug** in the **make** file.

The **IPBUILD** flag is located in the **IPCOM make** file *installDir*/**components/ip_net2-6.***x***/ipcom/Makefile**.

If **IPBUILD=debug**, then **IP_DEBUG** must also be defined in the Workbench project. To define this build macro, perform the following steps:

1. Right-click the project in **Project Explorer** and select **Properties**.
2. Select **Build Properties**.
3. Click the **Build Macro** tab.
4. Click **New**.
5. Type **IP_DEBUG** in the **Name** field.
6. Leave the default value blank.

You can direct debug output to the console or to the **syslog** file.

If you need to contact Wind River Support, send the entire **syslog** file for better support.

Name

syslog – system log command

Synopsis

```
syslog echo prio message
syslog list
syslog priority facility prio
syslog log file [logfile]
```

Description

Command options are as follows:

prio
Priority level of the specified message. Options include:

- **Emerg**
- **Crit**
- **Error**
- **Warning**
- **Notice**
- **Info**
- **Debug**
- **Debug2**

message
The specified message.

facility prio
Priority facility. Some of the following options may not be available with your Platform. Options include:

- **kern**
- **user**
- **daemon**
- **auth**
- **syslog**
- **ipcom**
- **ipcom_shell**
- **ipcom_telnet**
- **ipcrypto**
- **ipipsec**
- **ipike**
- **ipl2tp**
- **ipldapc**

- **iplite**
- **ipnat**
- **ippppoe**
- **ipradius**
- **iprip**
- **ipssh**
- **ipssl**
- **ipsslproxy**
- **ipftp**
- **ipfirewall**
- **ipdhcpd**
- **ipdhcpc**
- **ipwebs**
- **ipnet**
- **iptftp**
- **ipsntp**
- **ipdhcps**
- **ipdhcps6**
- **ipsnmp**
- **ipdhcpr**
- **ipcom_drv_eth**
- **ipppp**
- **ipmip**
- **ipappl**
- **iptcp**
- **ipmlds**
- **ipemanate**
- **ipfreescale**
- **ipmcp**
- **ipprism**
- **ip8021x**
- **ipeap**
- **ipsafenet**
- **iphwcrypto**
- **ipnetsnmp**
- **ipquagga**
- **ipdhcpc6**
- **ipcci**
- **ipdiameter**
- ***ILLEGAL***

logfile
The specified log file.

Table 5-1 lists the **syslog** debug levels.

Table 5-1 **syslog Debug Levels**

Level Name	No.	Workbench Configuration Parameter	Priority
EMERGENCY	0	**IPCOM_LOG_EMERG**	Highest priority.
CRITICAL	1	**IPCOM_LOG_CRIT**	
ALERT		Not supported in VxWorks— only in Linux.	
ERROR	2	**IPCOM_LOG_ERR**	Default log output level at run time, as defined by **IPCOM_SYSLOGD_DEFAULT_PRIORITY**.
WARNNING	3	**IPCOM_LOG_WARNNING**	This level and higher compiled in for normal builds.
NOTICE	4	**IPCOM_LOG_NOTICE**	
INFO	5	**IPCOM_LOG_INFO**	
DEBUG	6	**IPCOM_LOG_DEBUG**	Lowest priority. This level and higher compiled in for debug builds.
DEBUG 2	7	**IPCOM_LOG_DEBUG2**	Default level when debug flag is used.

Usage

Use the **echo** option to print a message to the syslog with the specified priority. For example, the following sequence of commands output the system log, prints **this is my string** with the priority level **Emergency**, and outputs the system log again, showing the new string.

The first command displays the existing log:

```
[vxWorks *] cat /ram/syslog
SAT MAY 26 05:00:12 2007: dhcps6[6068d120]: Error: Interface  was not found
```

The second command prints the string, using the **syslog echo** command:

```
[vxWorks *]#  syslog echo Emerg  "this is my string"
```

The third command redisplays the log, showing the new string:

```
[vxWorks *]#   cat /ram/syslog
SAT MAY 26 05:00:12 2007: dhcps6[6068d120]: Error: Interface  was not found
SUN MAY 27 08:53:00 2007: ipcom[605eafb0]: Emerg: this is my string
[vxWorks *]#
```

Use the **list** option to list the priority levels of the current facility:

```
[vxWorks *]# syslog list
syslog facility    priority
ipike              Error
ipl2tp             Error
ipssh             Debug
ipssl              Error
```

In the preceding example, **IPIKE**, **IPL2TP**, and **IPSSL** are set to priority **Error**, while **IPSSH** is set to priority **Debug**.

Use the **priority** option to modify the priority for a specified facility.

First, a **syslog list** command displays the priorities for the current facilities:

```
[vxWorks *]# syslog list
syslog facility    priority
ipssh              Debug
ipssl               Error
```

Next, a **syslog priority** command modifies the priority for **IPSSH** to **Critical**.

```
[vxWorks *]# syslog priority ipssh Crit
syslog: facility ipssh priority set to Crit
```

Finally, a **syslog list** command verifies the new priority.

```
[vxWorks *]# syslog list
ipssh              Crit
ipssl              Error
```

Use the **syslog log file** command to change the name of the system log file.

```
[vxWorks *]# syslog log file /ram/mylog
```

To clear the log file, use the current file name. The **syslog log file** command closes the current file and opens a new file with the same name. For example:

```
[vxWorks *]# cat /ram/syslog
MON MAY 28 06:04:35 2007: ipppp[6068f120]: Error: [ppp0 sec=0 lcp=1 auth=0
ipcp=0 ipv6cp=0] [ppp0] Failed to open driver: No such file or directory

[vxWorks *]# syslog log file /ram/syslog

[vxWorks *]# cat /ram/syslog
[vxWorks *]
```

System Variables (sysvar)

The **sysvar** command lists and modifies the network stack's global variables, as follows.

System variables are similar to UNIX environment variables, except that they are available throughout the system to any process. For example, if you are running an IKE process, you can issue a **sysvar** command to alter a NAT variable.

The **sysvar** command uses a treelike data structure for all the network components and services. For example, the system variable **iptcp.ConnectionTimeout** defines the number of seconds the network stack tries to create connection before giving up.

The system variable **ipssh.service.port_fwd** controls whether port forwarding can be used.

System variables are similar to the components you configure at build time using the Workbench **Kernel Configuration Editor**. Unlike kernel components, however, the **sysvar** command modifies parameters at run time.

NOTE: Not all global variables can be changed at run time. Some parameters can only be reset at build time, using Workbench or **vxprj**.

To include this command in your project, include **INCLUDE_IPCOM_SYSVAR_CMD**.

Name

sysvar – lists, gets, and defines system variables

Synopsis

```
sysvar list [name[*]]
sysvar get name
sysvar unset name[*]
sysvar set [-c | -o | -r] name value
```

Description

Command options are as follows:

name
: Name of a system variable.

-c
: OK to create.

-o
: OK to overwrite.

-r
: Flag read-only.

value
: Value of a system variable.

Usage

Use the option **list** to output system variables. The following example outputs a list of system variables:

```
[vxWorks *]# sysvar list
System variables:
    ipcom.hostname=iptarget
    ipcom.syslogd.default_priority=2
    ipcom.syslogd.queue.max=256
    etc.
```

Add a service name to restrict output to a particular branch of the tree structure. The following example outputs only SSH global variables:

```
[vxWorks *]# sysvar list ipssh
System variables:
    ipssh.auth.max_fail=3
    ipssh.auth.pub_key.allowed=1
    ipssh.auth.pub_key.required=0
    ipssh.auth.pub_key_first=1
    ipssh.auth.pw.allowed=1
    ipssh.auth.pw.required=0
    ipssh.bind_addr=0.0.0.0
    ipssh.enc.3des=1
    ipssh.enc.aes=1
    ipssh.enc.arcfour=1
    ipssh.enc.blowfish=1
    etc.
```

Add a branch to the service name to further define the output. The following example outputs only SSH authentication parameters:

```
[vxWorks *]# sysvar list ipssh.auth
System variables:
    ipssh.auth.max_fail=3
    ipssh.auth.pub_key.allowed=1
    ipssh.auth.pub_key.required=0
    ipssh.auth.pub_key_first=1
    ipssh.auth.pw.allowed=1
    ipssh.auth.pw.required=0
```

Use a wildcard to filter selected fields. The following example outputs only SSH authentication parameters beginning with **m**:

```
[vxWorks *]#  sysvar list ipssh.auth.m*
System variables:
   ipssh.auth.max_fail=3
```

Use the option **get** to read the value of a parameter. The following example reads the value of **iptcp.ConnectionTimeout**:

```
[vxWorks *]# sysvar get iptcp.ConnectionTimeout
sysvar: iptcp.ConnectionTimeout=30
```

Use the option **unset** to remove a parameter. The following example removes the **iptcp.ConnectionTimeout** parameter:

```
[vxWorks *]# sysvar unset iptcp.ConnectionTimeout
sysvar: 'iptcp.ConnectionTimeout' unset ok
```

A subsequent **sysvar get** command verifies the removal:

```
[vxWorks *]# sysvar get iptcp.ConnectionTimeout
sysvar: 'iptcp.ConnectionTimeout' not found
```

Use the option **set** to assign a value to a parameter. Use this option in conjunction with the **-o** flag and the options **list** and **get** to read a parameter, assign it a new value, and verify the assignment of the new value.

The first command displays the value for the **ipcom.hostname** parameter:

```
[vxWorks *]# sysvar list ipcom.hostname
System variables:
    ipcom.hostname=iptarget
```

The second command assigns the value **MyTarget** to this parameter, but fails because the parameter already exists:

```
[vxWorks *]# sysvar set ipcom.hostname MyTarget
sysvar: set failed : duplicate entry
```

The third command overwrites the existing value of **ipcom.hostname** with **MyTarget**:

```
[vxWorks *]#  sysvar set -o ipcom.hostname MyTarget
sysvar: ipcom.hostname=MyTarget ok
```

The fourth command verifies the new value:

```
[vxWorks *]# sysvar get ipcom.hostname MyTarget
sysvar: ipcom.hostname=MyTarget
```

Use the **set** option with the **-c** flag to create a new parameter.

First, use a **sysvar get** command to search for the new parameter, in case it already exists:

```
[vxWorks *]# sysvar get MyParam
sysvar: 'MyParam' not found
```

The second command creates the new parameter and assigns it a value:

```
[vxWorks *]# sysvar set -c MyParam MyValue
sysvar: MyParam=MyValue ok
```

The third command verifies the creation of the parameter with the assigned value:

```
[vxWorks *]# sysvar get MyParam
sysvar: MyParam=MyValue
```

Use the **-r** flag to create a read-only parameter. After creation, this parameter cannot be modified or deleted.

The first command creates the new parameter and assigns it a value:

```
[vxWorks *]# sysvar set -cr NewParam NewValue
sysvar: NewParam=NewValue ok
```

The second command reads the value of the parameter:

```
[vxWorks *]# sysvar get NewParam
sysvar: NewParam=NewValue
```

The third command attempts to assign a new value to the parameter. This command fails because it is a read-only parameter.

```
[vxWorks *]# sysvar set NewParam VeryNewValue
sysvar: set failed : readonly entry
```

Calling Shell Commands from an Application

Shell commands provide a convenient method for optimizing the network stack and related applications. Using shell commands such as **sysvar**, you can test various parameters without having to rebuild the network stack with every change.

You can also call shell commands from an application, using one of the public APIs. This method would allow you to dynamically reconfigure the network stack, perhaps in response to changing conditions. You can start, stop, or reconfigure daemons for individual facilities, or change system variables.

For example, to list daemons, call the following routine:

```
Ip_err ipcom_ipd_send(const char *name, int msgtype)
```

Call this routine once for each module. For example:

```
ipcom_ipd_send("ipssh", IPCOM_IPD_MSGTYPE_PING)
ipcom_ipd_send("ipike", IPCOM_IPD_MSGTYPE_PING)
ipcom_ipd_send("ipftps", IPCOM_IPD_MSGTYPE_PING)
```

If the return code is **IPCOM_SUCCESS**, the module is running.

To get the value of a system variable, call the following routine:

```
IP_PUBLIC char * ipcom_sysvar_get(const char *name, char *value, Ip_size_t
*value_size);
```

For example, to get the value of the TCP **Connection.Timeout** variable, call the following routine:

```
ipcom_sysvar_get(" ipcom.tcp.timeout", IP_NULL, IP_NULL);
```

The return value is char *, which is the value in string format you get from the **syvar get** command.

For example, if the value of **ipcom.tcp.timeout**=20, the return code from

```
ipcom_sysvar_get(" ipcom.tcp.timeout", IP_NULL, IP_NULL);
```

is a string with the value of 20.

To set a sysvar programmatically, call the following routine:

```
ipcom_sysvar_set("sysvar_name", "sysvar_value", IPCOM_SYSVAR_FLAG_OVERWRITE)
```

For example, to change the value of **ipcom.tcp.timeout** to 40 and overwrite the original value, call the following routine:

```
ipcom_sysvar_set(" ipcom.tcp.timeout", 40 IPCOM_SYSVAR_FLAG_OVERWRITE);
```