

Wind River® General Purpose Platform, VxWorks® Edition

MIGRATION GUIDE

3.6

Copyright © 2007 Wind River Systems, Inc.

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

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

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

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

installDir\product_name\3rd_party_licensor_notice.pdf.

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

Corporate Headquarters

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

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

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

<http://www.windriver.com>

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

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

Contents

1	Overview	1
1.1	Introduction	1
1.1.1	About This Guide	1
1.1.2	Finding Additional Migration Information	2
1.2	Platform Migration Summary	2
1.2.1	Operating System Migration	3
1.2.2	Development Environment Migration	3
1.2.3	BSP Migration	3
1.2.4	Networking and Middleware Migration	4
1.3	Important Changes Requiring Migration	6
1.3.1	Changes Introduced in Wind River General Purpose Platform, VxWorks Edition 3.6	6
	Deprecated IPCOM Routines	6
	Changes to tNetTask	7
1.3.2	Changes Introduced in Wind River General Purpose Platform, VxWorks Edition 3.5	7
	Directory Structure	7
	Library Archive Changes	8
	Product or Component Initialization	9
	Configuration and Scalability	9

	Configuration Header Files	10
	Backward Compatibility	10
	Shell Commands	11
	Diagnostics and Debugging	12
1.3.3	Downloadable Kernel Modules	12
2	Wind River Network Stack Migration Overview	15
2.1	Introduction	15
2.2	Key Concepts	16
2.3	Evaluating the Migration Effort	16
2.4	Source Compilation	17
2.5	Network Stack Configuration and Migration	18
2.5.1	Component and Parameter Configuration	18
2.5.2	Network Stack Directory Structure	18
2.5.3	Ported Applications and Libraries	18
2.5.4	Backward Compatibility Wrappers	21
2.5.5	Removed Header Files	23
2.6	Migrating Applications	27
2.6.1	Migrating an Application that Uses Networking APIs	27
2.6.2	Migrating a Socket-Based Application	28
3	Wind River Network Stack: Transport and Network Protocols	31
3.1	Introduction	32
3.1.1	Feature Release Matrix	32
3.1.2	Changes in Wind River Network Stack 6.5	34
3.1.3	Changes in Wind River Network Stack 6.6	34
3.2	Migrating to SMP	35
	Creating an SMP-Capable VxWorks Image Project	35

3.3	Socket Options	36
	New Socket Commands	36
	Socket Option Definitions	36
	Interface-Related Socket Options	36
3.4	IPv4 and IPv6 Components	37
3.4.1	IPv4 and IPv6 Configuration	37
3.4.2	API Mapping	38
	TCP/IP Layer Core Networking Routines	38
	ICMP Routines	39
3.5	ARP Components	39
	API Mapping for ARP Routines	39
3.6	Proxy ARP	40
3.7	Multicasting Components	41
3.7.1	Socket Commands	41
	IPv4 Multicast Routing set/getsockopt Options	41
	IPv6 Multicast Routing set/getsockopt Options	42
3.8	Show Routine Components	42
3.8.1	Show Routine Configuration	42
3.8.2	API Mapping	43
3.9	Utility Components	44
3.10	RIP Components	44
3.10.1	RIP Configuration	44
3.10.2	API Mapping	45
3.11	RIPng Components	48
3.11.1	Changes to RIPng Files	48
3.11.2	RIPng Configuration	49
3.11.3	API Mapping	49

3.12	Routing APIs	50
3.13	Routing Sockets	51
3.13.1	Routing Socket Configuration	51
3.13.2	Ranking Routes in the Route Table	52
3.13.3	Routing Socket Messages	52
	Extended Messages	52
3.14	Virtual Stack	53
3.14.1	Overview	53
3.14.2	Virtual Stack Configuration	53
4	Wind River Network Stack: Application Protocols	55
4.1	Introduction	56
4.1.1	Feature Release Matrix	56
4.1.2	Shell Commands and API Changes	58
4.2	DHCP (IPv4 and IPv6) Components	58
4.2.1	DHCP Configuration	58
4.2.2	API Mapping	59
4.3	DNS Components	65
4.3.1	DNS Configuration	65
4.3.2	API Mapping	67
4.4	FTP Components	68
4.4.1	FTP Configuration	68
4.4.2	API Mapping	68
4.5	Ping Components	71
4.5.1	Ping Configuration	71
4.5.2	API Mapping	72

4.6	SNTP Components	72
4.6.1	SNTP Configuration	72
	Enabling the Client or Server	73
4.6.2	API Mapping	73
4.7	Telnet Components	74
4.7.1	Telnet Configuration	74
4.7.2	API Mapping	74
4.8	TFTP Components	75
4.8.1	TFTP Configuration	75
4.8.2	API Mapping	76
4.9	Internet and Local Domain Sockets	77
4.9.1	Sockets Configuration	77
4.9.2	API Mapping	77
4.9.3	Changes in Socket Options	77
	Wind River Network Stack 3.1 IPv4 Socket Options	78
	IPv6 Socket Options	79
4.9.4	New Socket Options	81
	New IPv4 Socket Options	81
	New IPv6 Socket Options	83
	Socket Options for Policy Routing	83
	New Socket Options for PPP	84
	Socket Options for Diffserv	85
4.10	RTP	86
4.11	NFS Client and Server	86
5	Wind River Network Stack: Interfaces and Drivers	87
5.1	Introduction	88
5.1.1	Feature Release Matrix	88

5.2	General Interface and Driver Configuration	89
5.2.1	Configuration Components	89
5.2.2	Shell Commands	90
5.2.3	Unchanged Libraries and APIs	91
5.2.4	Obsolete APIs	91
5.2.5	Checksum Offloading	91
5.3	Memory Management	92
5.3.1	Changes	92
	Drivers and Leading Spaces in Cluster Headers	92
	Use of netBufLib	93
5.4	MIB2 Statistics-Collection Support	94
5.4.1	M2IfLib	94
5.4.2	SNMP MIB-II Support	94
5.5	Routing, Router Advertisement, and Router Solicitation	95
5.5.1	API Mapping	95
	Routing	95
	Router Advertisement	96
	Router Solicitation	96
5.6	BPF	97
5.7	Interface Components	97
5.8	IP Attach Components	100
5.8.1	API Mapping	100
5.9	MUX-L2	101
5.10	Auto IP	101
5.11	zBuf and Fast UDP Sockets	101
5.12	Unnumbered Interfaces	102

6	Wind River PPP	103
6.1	Introduction	103
6.1.1	Feature Release Matrix	104
6.1.2	Changes in Wind River PPP 6.6	105
6.1.3	Additional Documentation	105
6.2	Migration Steps	105
6.3	Configuration	106
6.3.1	Configuring VxWorks Image Projects for Wind River PPP	106
	Disabling PPPoE at Compile Time	106
6.3.2	Initialization	107
6.3.3	Shell Commands	107
6.3.4	Configuration Routines	107
6.4	Library and Routine Changes	114

1

Overview

- 1.1 Introduction 1
- 1.2 Platform Migration Summary 2
- 1.3 Important Changes Requiring Migration 6

1.1 Introduction

This document provides information for migrating the Wind River General Purpose Platform, VxWorks Edition to release 3.6.

1.1.1 About This Guide

Changes and improvements that affect the way you use and program many of the products included in your Platform are documented in this guide. This guide provides release-to-release information for products that included major changes in the Wind River General Purpose Platform, VxWorks Edition 3.5 or include major changes in this release, the Wind River General Purpose Platform, VxWorks Edition 3.6. For general information about these changes, see [1.3 Important Changes Requiring Migration](#), p.6. For information specific to an individual product, see the product-specific chapter in this guide.

This guide contains product-specific chapters for the following products in your Platform that require migration:

- Wind River Network Stack (4 chapters)
- Wind River PPP

This guide does not contain detailed information about new features in your Platform unless they relate to features that have been replaced or require migration. For a comprehensive discussion of new features and changes in this release, see the release notes for your Platform.

1.1.2 Finding Additional Migration Information

This guide contains migration information for the products listed in [1.1.1 About This Guide](#), p.1. Additional migration information is provided in other sources, as follows:

- VxWorks migration information is included in the VxWorks guides. For details, see [1.2.1 Operating System Migration](#), p.3.
- Wind River Workbench migration information is included in the Workbench guides. For details, see [1.2.2 Development Environment Migration](#), p.3.
- BSP migration information is provided in *VxWorks Device Driver Developer's Guide*, 6.6 and the *VxWorks BSP Developer's Guide*, 6.6, as described in [1.2.3 BSP Migration](#), p.3.

1.2 Platform Migration Summary

This section summarizes the migration status of the products in your Platform. Migration issues that apply to many products or the entire Platform are described in [1.3 Important Changes Requiring Migration](#), p.6.

The following products have few or no migration issues in this release:

- Wind River Compiler
- Wind River GNU Compiler
- Wind River Run-Time Analysis Tools
- Wind River VxWorks Simulator

For information on changes to these products, see the release notes for your Platform.

1.2.1 Operating System Migration

For VxWorks migration information, see the following:

- If you are migrating from a VxWorks release earlier than 6.0, consult the *VxWorks 5.5 Migration Guide*, 6.6 before beginning development.
- For information on migrating legacy drivers to VxBus and other driver migration issues, see the *VxWorks Device Driver Developer's Guide, Volume 3: Legacy Drivers and Migration*.
- For information on migrating kernel applications to real-time process (RTP) applications, see the *VxWorks Kernel Programmer's Guide*, 6.6.
- For information on moving from a uniprocessor environment to symmetric multiprocessing (SMP) environment using the optional VxWorks SMP feature, see the *VxWorks Kernel Programmer's Guide*, 6.6.



NOTE: SMP support for VxWorks is available as an optional product. However, default SMP system images for the Wind River VxWorks Simulator are provided with the standard VxWorks installation as an introduction to the product.

1.2.2 Development Environment Migration

Wind River Workbench

Wind River Workbench 3.0 adopts the latest versions of the Eclipse C/C++ Development Toolkit, Device Debugging, and Target Management projects. As a result, some Workbench workflows and views have changed. For information on these changes, see the *Wind River Workbench User's Guide: What's New with CDT, CC, and TM*. For a comprehensive discussion of the changes in Wind River Workbench 3.0, see the release notes for your Platform.

1.2.3 BSP Migration

Information about migrating your BSPs can be found in the *VxWorks Device Driver Developer's Guide*, 6.6 and the *VxWorks BSP Developer's Guide*, 6.6.

For a list of BSPs supported in this release, see the release notes for your Platform.

BSP Makefile Change

A BSP makefile change was introduced in the Wind River General Purpose Platform, VxWorks Edition 3.5 and may require a change if you are migrating from an earlier release.

EXTRA_INCLUDE defines with the following format should not be used in BSP makefiles:

```
EXTRA_INCLUDE = $(path)
```

The **EXTRA_INCLUDE** define may already be defined coming into your makefile. To add directories to your include path, use the following:

```
EXTRA_INCLUDE += $(path)
```

1.2.4 Networking and Middleware Migration

The products discussed in this section have migration issues. For information on changes to networking and middleware products not discussed here, see the release notes for your Platform.

Wind River Network Stack

Major changes to Wind River Network Stack were introduced in the previous release, Wind River Network Stack 6.5. If you are migrating from Wind River Network Stack 3.x or earlier, see the following chapters in this guide for network stack migration information:

2. Wind River Network Stack Migration Overview

Provides introductory information on network stack migration. Describes key concepts and migration information common to most or all networking components.

3. Wind River Network Stack: Transport and Network Protocols

Provides migration details and component and API mapping for the components of the core network stack, including TCP/IP, multicast, and routing.

4. Wind River Network Stack: Application Protocols

Provides migration details and component and API mapping for the network application components, including DHCP and DNS, and information on programming with sockets.

5. Wind River Network Stack: Interfaces and Drivers

Provides migration details and information on changes to libraries and routines for lower-level network stack components, including the MUX and interface configuration.

Changes introduced in Wind River Network Stack 6.6 are minimal. For details, see the chapters listed above. In particular, the following changes should be noted:

- The IPCOM routines **ipcom_run_cmd()** and **ipmcp_cmd()** are deprecated in this release.
- The names of some DNS configuration parameters have changed. For details, see [4.3.1 DNS Configuration](#), p.65.

For a list of functional enhancements in Wind River Network Stack 6.6, see the release notes for your Platform.

Wind River PPP

Major changes to Wind River PPP were introduced in the previous release, Wind River PPP 6.5. Workbench components and API routines were replaced, and a shell command was added. If you are migrating from Wind River PPP 2.x, see [6. Wind River PPP](#) for details. If you are migrating from Wind River PPP 6.5 to Wind River PPP 6.6, there are no migration issues.

Wind River TIPC

There are no migration concerns for Wind River TIPC aside from a changed parameter requirement for the **shutdown()** routine. Specifically, the syntax of the **shutdown()** routine in the TIPC socket API is:

```
STATUS shutdown
(
    int sd,          /* identifies the socket to shut down */
    int how          /* function code */
)
```

In previous releases, only the complete shutdown of a socket is supported and the **how** parameter is ignored. In the current release, only a complete shutdown is supported, but you must now enter the following value for the **how** parameter:

SHUT_RDWR

Any other value for the parameter results in an error.

General interoperability issues between releases of Wind River TIPC are covered in the *Wind River TIPC for VxWorks 6 Programmer's Guide*.

Wind River USB

Wind River USB 2.4 (this release) can only be used with BSPs that support the VxBus device driver framework; USB support for BSPs that are not VxBus compatible is discontinued. For information on the VxBus framework, see *VxWorks Device Driver Developers Guide, Volume 1*. For information on adding VxBus support to your BSP, see the *VxWorks BSP Developers Guide*.

BSP VxBus support reduces the amount of BSP-specific software required to support the USB controllers. If your BSP supports VxBus, the only change typically required to port Wind River USB to Wind River USB 2.4 is to remove any unneeded PCI configuration routines and interrupt attach routines contained in the `usbPciStub.c` file in the BSP directory. The `usbPciStub.c` file should contain only those memory translation functions that are necessary to support memory space mapping that is not one-to-one. For further information on BSP configuration requirements, see the *USB Drivers* chapter of the *VxWorks Device Driver Developers Guide, Volume 2*.

USB Class drivers written for prior releases of the Wind River USB do not require migration.

1.3 Important Changes Requiring Migration

1.3.1 Changes Introduced in Wind River General Purpose Platform, VxWorks Edition 3.6

The following important changes are introduced in this release and apply to multiple products in your Platform.

Deprecated IPCOM Routines

The following IPCOM routines, which were introduced in Wind River Network Stack 6.5, are deprecated in Wind River Network Stack 6.6:

- `ipcom_run_cmd()`
- `ipmcp_cmd()`

Command interpreter commands should not be called programmatically using **ipcom_run_command()**. There are documented public APIs for all functionality that is available as shell commands; they should be used instead.

Changes to tNetTask

If you are programming in a uniprocessor system, the **tNetTask** task is now named **tNet0**. Functionality for this task is unchanged. If you are using VxWorks SMP, multiple instances of this task are available and named **tNet n** . For more information on **tNet0**, see the *Network Driver* chapter of the *VxWorks Device Driver Developer's Guide, Volume 2*.

1.3.2 Changes Introduced in Wind River General Purpose Platform, VxWorks Edition 3.5



NOTE: The content in this section applies to the products listed below and does not apply to all products in your Platform. However, products that make use of the networking facilities (but are not listed below) may also be impacted indirectly by these changes.

The following sections provide important information regarding some of the fundamental differences between the Wind River General Purpose Platform, VxWorks Edition 3.4 and later releases (that is, the Wind River General Purpose Platform, VxWorks Edition 3.5 and 3.6).

Changes in directory structure, configuration, and programming philosophy were introduced in the following products in the Wind River General Purpose Platform, VxWorks Edition 3.5:

- Wind River Network Stack
- Wind River PPP

Directory Structure

The directory structure has changed for the products listed above. The source code for these products is now located under the following directory:

installDir/components/ip_net2-6.6

The source tree generally contains one directory for each component or product. The directories follow a standard structure and typically include the sub-directories described in [Table 1-1](#).

Table 1-1 **Component Sub-Directory Structure**

Sub-Directory	Contents
config	Code that configures the product.
gmake	Makefiles for running GNU make utility.
include	Public header files that applications can use. (See the list below.)
src	Source code, including the product, APIs, and shell commands.

The following header files in *installDir/components/ip_net2-6.6/ipcom/include* contain public APIs:

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

These APIs are documented in the reference entries for the Wind River Network Stack Kernel API Reference. The rest of the files in the */ipcom/include* directory are for internal use only and *should not* be used by applications.

Wrapper routines and common network infrastructure source code are still located in the following directory:

installDir/vxworks-6.6/target/src/wrn

Library Archive Changes

The locations and names of the library archives have changed. **libnet.a** is obsolete and has been replaced by the files described in [Table 1-2](#).

These new files can be found under the following directory:

installDir/vxworks-6.6/target/lib/...

The exact name of the sub-directory depends on the CPU family and tool chain you are using.

Table 1-2 New Library Archive Files

Library Archive File	Description
libnetapps.a	Contains the object modules for components that have been ported.
libnetcommon.a	Contains the object modules for common networking components.
libnetwrap.a	Contains the object modules for wrapper components.

New library archives can be found in the following directory:

installDir/components/obj/vxworks-6.6/knl/lib/...

The exact name of the sub-directory depends on the CPU family and tool chain you are using.

Product or Component Initialization

Initialization and startup for changed components is automatically regulated by a central process, IPNET daemon (IPD). IPD can also be used to start, stop, and in some cases, reconfigure the components of the network stack and middleware products at run time. IPD is accessed through a shell command or hook routine. For more information, see [Shell Commands](#), p.11.



NOTE: The **ipd** command is described in *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

Configuration and Scalability

The configuration methods for the following products changed in the Wind River General Purpose Platform, VxWorks Edition 3.5:

- Wind River Network Stack
- Wind River PPP

While you can still configure components through Workbench, there is now more flexibility for scaling and configuration.

Configuration parameters can be statically configured through Workbench, and dynamically reconfigured at run time using shell commands. In addition, some components can be further scaled at library build-time by editing the configuration header file found in the product's **config** sub-directory.

The network stack must be recompiled and the VxWorks image must be rebuilt after scalability decisions are made. More details about including and excluding components can be found in the product-specific chapters in this guide.

Configuration Header Files

In release 3.4 and earlier of the Wind River General Purpose Platform, VxWorks Edition, the **configAll.h** file contained component definitions and configuration parameters. This file was used by the BSP command-line build, as well as Workbench or **vxprj**, when creating a VxWorks Image Project. In the Wind River General Purpose Platform, VxWorks Edition 3.5 and later, components, including excluded components, are in **configAllNetwork.h**, and configuration parameters are in **configNetParams.h**. These files are located in the following directory:

installDir/vxworks-6.6/target/config/all

Backward Compatibility

Some VxWorks-specific APIs and applications are carried forward to provide backward compatibility and ease migration. Some other common networking APIs and features from the previous release are provided as wrappers in the network stack. Inside the wrappers, the equivalent new functionality is mapped to maintain backward compatibility. Wrapper routines should be used only to maintain backward compatibility. They should not be used in new applications because they will be deprecated and eventually disappear from subsequent releases of the network stack.

For a list of the routine wrappers in Wind River Network Stack 6.6, see [2.5.4 Backward Compatibility Wrappers](#), p.21. For a list of applications and APIs carried forward in Wind River Network Stack 6.6, see [2.5.5 Removed Header Files](#), p.23.

Shell Commands

1

Many products included in your Platform now include additional shell commands that can be used to control components and products or view statistics relating to their operation.

To use the new commands, you must explicitly include the command components in your project. The `INCLUDE_USE_NATIVE_SHELL` component must be included, as well as the specific component for the command (`INCLUDE_component_CMD` or `INCLUDE_command_CMD`).

For descriptions and syntax for the shell commands common to the Platform, see *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

Accessing the Command Interpreter

In previous releases, most shell commands are C functions that are called using the VxWorks (host or target) shells' C interpreter, or programmatically. The new shell commands must be invoked from the command interpreter, and they in turn call the necessary C routine.

The C interpreter is the default option in the VxWorks shell. To access the command interpreter, enter **cmd** at the shell prompt; to return to the C interpreter, enter **C** at the prompt. For more information on the VxWorks shells, the C interpreter, or the command interpreter, see the *VxWorks Kernel Programmer's Guide: Target Tools*.

In some cases, C interpreter commands have been replaced by command-interpreter commands. This means that the facility can only be used with the shell, and not programmatically, unless a wrapper has been provided (as is the case, for example, with **ping**).

New Commands for Dynamic Control and Configuration

There are many new shell commands, but two of the most important commands are **sysvar** and **ipd**.

About sysvar

Components can now be configured at run time through the **sysvar** command. Almost all documented configuration parameters can be set with the **sysvar** command. The **sysvar** command can also be used to display the current run-time configuration setting for a component.

For details on using **sysvar**, see *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*. The **sysvar** variables are listed in the programmer's guides for each product.

The following is a simple example showing how to enable CHAP authentication for PPP:

```
[vxWorks *]# sysvar set -o ipppp.auth chap
```

About ipd

Components are implemented through a series of daemons that can be started and stopped through the **ipd** command (see also [Product or Component Initialization](#), p.9). Restarting a daemon after changing the configuration through the **sysvar** command allows a flexible approach to testing various network configurations without having to reboot your system.

The following is a simple example showing how to stop and start the IKE component:

```
[vxWorks *]# ipd stop ipike  
[vxWorks *]# ipd start ipike
```

Diagnostics and Debugging

The approach to diagnostics and debugging for many of the products and components in your Platform differs from releases prior to the Wind River General Purpose Platform, VxWorks Edition 3.5, largely due to the introduction of new shell commands described in [Shell Commands](#), p.11. Many of the diagnostic routines previously available have been replaced by commands and methods that do not necessarily provide a one-to-one mapping. However, the flexibility and efficiency of the shell commands allows for greater control over debugging and diagnostics. This guide provides mappings for obsolete APIs to new routines or shell commands. For further details, see the product-specific chapters in this guide.

1.3.3 Downloadable Kernel Modules

When creating or importing a VxWorks downloadable kernel module (DKM) project, include paths are set up as absolute paths. When you build your project for the first time and select **Generate Includes...**, the resultant environment variables are incorrect. To correct the paths, replace the absolute paths with **\$WIND_BASE**.

If you have a large project that makes use of many components, it may be more efficient to recreate your DKM project rather than using the import tool.

2

Wind River Network Stack Migration Overview

- 2.1 Introduction 15
- 2.2 Key Concepts 16
- 2.3 Evaluating the Migration Effort 16
- 2.4 Source Compilation 17
- 2.5 Network Stack Configuration and Migration 18
- 2.6 Migrating Applications 27

2.1 Introduction

The 3.1 and earlier releases of Wind River Network Stack are based on a port of the KAME/FreeBSD network stack release. With the 6.5 release, Wind River introduced a new proprietary network stack that is highly flexible, and continues to provide industry-standard socket interfaces.

Wind River Network Stack 6.6 is an update to the stack that was introduced in the 6.5 release. Application code that uses non-standard, Wind River Network Stack 3.1-specific APIs and interfaces must be revised before migrating to Wind River Network Stack 6.6.

This chapter and the subsequent three chapters will help you to plan your migration from Wind River Network Stack 3.1 to Wind River Network Stack 6.6.

They also discuss any changes between the 6.5 and 6.6 releases. This chapter provides general information about network stack migration, including key concepts and basic migration steps, and migration information common to most or all networking components. The subsequent chapters provide component or feature and API mapping when possible. The following chapters are organized generally by network stack layer and follow the organization of the three-volume *Wind River Network Stack Programmer's Guide, 6.6*:

- [3. Wind River Network Stack: Transport and Network Protocols](#)
- [4. Wind River Network Stack: Application Protocols](#)
- [5. Wind River Network Stack: Interfaces and Drivers](#)

2.2 Key Concepts

Wind River Network Stack 6.5 and 6.6 differs from the 3.1 release in the way it can be built and configured. Some commonly used features have been ported and wrapper routines have been provided for others. For important information regarding the fundamental differences between Wind River Network Stack 6.6 and 3.1-era releases, see the following sections in this guide:

- [1.3 Important Changes Requiring Migration](#), p.6
- [2.5 Network Stack Configuration and Migration](#), p.18

2.3 Evaluating the Migration Effort

Wind River Network Stack 3.1 is replaced by Wind River Network Stack 6.5 (which was then upgraded to 6.6). Some Workbench components and API routines have been replaced, and new shell commands have been added. These changes are due to an evolution of the product that incorporates a smaller, more scalable stack.

The following questions will help you to assess which aspects of your existing systems will require changes to migrate. If the answer is no to all questions, the migration effort should be small.

- **Does your application use Wind River Network Stack private APIs (libraries or APIs that are not documented as part of the standard stack)?**

Private APIs continue to be undocumented. For the Wind River Network Stack 6.6 release, many 3.1-era private APIs are obsolete. If you have used Wind River Network Stack private APIs, you should migrate to the documented public APIs. This will ease any migration efforts.

- **Does your application use any of the now obsolete header files?**

Header files that are obsolete in this release must be removed and replaced by the new header files, or commented out in your files. For a list of obsolete header files, see [2.5.5 Removed Header Files](#), p.23.

- **Does your application use hard-coded constants?**

Some of the constant values may change. If you have used hard-coded constants in your application, you must update your code.

- **Does your application make use of the virtual stack feature in the Wind River Network Stack?**

If your applications use the virtual stack feature, they must be updated to use the virtual router feature of Wind River Network Stack 6.6. The virtual stack feature and associated libraries are obsolete. For more information about the virtual router feature, see [3.14 Virtual Stack](#), p.53.

2.4 Source Compilation

Most protocol implementations, whether newly implemented in the current release, or carried over from the previous release, are precompiled and provided in both source and binary form.

If you wish to create a customized network stack, you must recompile the source code as described in the getting started guide for your Platform. If you subsequently add or remove components in the source code, you must recompile that code, then rebuild your VxWorks Image Project.

If you are migrating a network application that makes standard socket calls, you must recompile the application, even if no other part of the application calls into newly implemented protocols or other features of the current release.

2.5 Network Stack Configuration and Migration

This section provides important information regarding changes to the build and configuration processes and components in Wind River Network Stack 6.6. For details regarding changes to components, parameters, files, and configuring the individual network stack components, see the following chapters:

- [3. Wind River Network Stack: Transport and Network Protocols](#)
- [4. Wind River Network Stack: Application Protocols](#)
- [5. Wind River Network Stack: Interfaces and Drivers](#)

2.5.1 Component and Parameter Configuration

For information on VxWorks projects and component mapping, see [2.6 Migrating Applications](#), p.27. Just as there may not be one-to-one mappings for components, there is rarely a one-to-one mapping for configuration parameters. For most of the components, you must read the configuration parameter descriptions in the *Wind River Network Stack for VxWorks 6 Programmer's Guide, 6.6* (volumes 1 through 3).

2.5.2 Network Stack Directory Structure

The directory structure and file locations for new networking components have changed. For details, see [Directory Structure](#), p.7.

2.5.3 Ported Applications and Libraries

[Table 2-1](#) lists the applications that have been ported from Wind River Network Stack 3.1 to Wind River Network Stack 6.6.

For more information, see [Backward Compatibility](#), p.10.

Table 2-1 Applications Ported from Wind River Network Stack 3.1 to Wind River Network Stack 6.6

Application Name	Component Name(s)
NFS client/server ^a	INCLUDE_NFS_CLIENT_ALL, INCLUDE_NFS2_CLIENT, INCLUDE_NFS3_CLIENT, INCLUDE_CORE_NFS_CLIENT, INCLUDE_NFS_MOUNT_ALL, INCLUDE_NFS2_SERVER, INCLUDE_NFS3_SERVER, INCLUDE_NFS_SERVER_ALL, INCLUDE_CORE_NFS_SERVER, INCLUDE_NFS_SERVER_INSTALL
hostLib	INCLUDE_HOST_TBL
FTP (v4/v6) client backend APIs	INCLUDE_FTP, INCLUDE_FTP6
remlib	INCLUDE_REMLIB
rloglib	INCLUDE_RLOGIN
Telnet client	INCLUDE_TELNET_CLIENT
TFTP client	INCLUDE_TFTP_CLIENT
net sysctl	INCLUDE_NET_SYSCTL
l2config	INCLUDE_L2CONFIG

a. In Workbench, NFS has moved from under the **Network Applications** folder to **Operating System Components > IO system components > NFS Components**.

Table 2-2 lists the libraries that have been ported from Wind River Network Stack 3.1 to Wind River Network Stack 6.6.

Table 2-2 Libraries Ported from Wind River Network Stack 3.1 to Wind River Network Stack 6.6

Library Filename	Component Name
common/mux/muxLib.c	INCLUDE_MUX
common/mux/muxTkLib.c	none
common/mux/muxFunc.c	none
common/mux/muxL2Lib.c	INCLUDE_MUX_L2

Table 2-2 Libraries Ported from Wind River Network Stack 3.1 to Wind River Network Stack 6.6 (cont'd)

Library Filename	Component Name
common/daemon/jobQueueLib.c	INCLUDE_JOB_QUEUE
common/daemon/jobQueueUtilLib.c	INCLUDE_JOB_QUEUE_UTIL
common/daemon/daemon.c	INCLUDE_NET_DAEMON
common/libc/dlinklib/etherMultiLib.c	none
common/mem/netBufLib.c	INCLUDE_NETBUFLIB
common/mem/netBufPool.c	INCLUDE_NET_POOL
common/mem/netBufAdvLib.c	INCLUDE_NETBUFADVLIB
common/mem/linkBufPool.c	INCLUDE_LINKBUFPOOL
common/mem/netPoolShow.c	INCLUDE_NETPOOLSHOW
common/timer/gtf_core.ccommon/timer/ gtf_util.ccommon/timer/gtf_wrapper.c	INCLUDE_GTF, INCLUDE_GTF_TIMER_START
common/mem/uipc_mbuf.ccommon/ mem/uipc_mbuf2.c	none
common/utillslib/wvNetDLib.c	none
sysdep/os/vxWorks/socket/sockLib.c sysdep/os/vxWorks/socket/sockScLib.c	INCLUDE_SOCKETLIB, INCLUDE_SC_SOCKETLIB
dlink/qosIngressHooks.cdlink/ qosIngressLib.c	INCLUDE_QOS_INGRESS_HOOKS
apps/common/applUtilLib.c	INCLUDE_APPL_LOG_UTIL
common/libc/hostlib/hostLib.c common/libc/hostlib/hostUtils.c common/libc/hostlib/rtpGetnameinfo.c common/libc/hostlib/rtpGetaddrinfo.c common/libc/hostlib/rtpHostLib.c common/libc/hostlib/hostSetup.c common/libc/hostlib/hostSetupSysctl.c	INCLUDE_HOST_TBL, INCLUDE_HOST_TBL_SYSCT

2.5.4 Backward Compatibility Wrappers

[Table 2-3](#) lists the wrapper routines provided in Wind River Network Stack 6.6. With the exception of the standard POSIX routines, most wrapper routines will not be maintained indefinitely and will be deprecated in a future release.

Most wrapper components are available through Workbench in their original folders with their original names. Wrapper components are also available through Workbench in one place, under: **Network Components > Network Core Components > Backwards compatibility wrapper routines**.

For more information, see [Backward Compatibility](#), p.10.

Table 2-3 Wrapper Routines in Wind River Network Stack 6.6

Routine Name	Component Name	Comments
arpAdd(), arpDelete(), arpShow()	INCLUDE_ARP_API or INCLUDE_IPWRAP_ARP	See the API reference entry for functional changes.
getaddrinfo()	INCLUDE_GETADDRINFO or INCLUDE_IPWRAP_GETADDRINFO	See the API reference entry for functional changes.
gethostbyaddr()	none	See the API reference entry for functional changes.
gethostbyname()	none	See the API reference entry for functional changes.
getifaddrs(), freeifaddrs()	INCLUDE_IPWRAP_GETIFADDRS	
getnameinfo()	INCLUDE_GETNAMEINFO or INCLUDE_IPWRAP_GETNAMEINFO	See the API reference entry for functional changes.
getservbyname()	INCLUDE_GETSERVBYNAME or INCLUDE_IPWRAP_GETSERVBYNAME	See the API reference entry for functional changes.
getservbyport()	INCLUDE_GETSERVBYPORT or INCLUDE_IPWRAP_GETSERVBYPORT	See the API reference entry for functional changes.

Table 2-3 Wrapper Routines in Wind River Network Stack 6.6 (cont'd)

Routine Name	Component Name	Comments
if_nametoindex() , if_indextoname() , if_nameindex() , if_freenameindex()	none	ifLib.c::ifNameToIfIndex() and ifLib.c::ifIndexToIfName() are obsolete. Use ifname.c::if_nametoindex() and ifname.c::if_indextoname() .
ifconfig()	INCLUDE_IFCONFIG or INCLUDE_IPWRAP_IFCONFIG	See the API reference entry for functional changes.
inet_addr() , inet_ntoa() , inet_aton() , inet_ntop() , inet_pton()	INCLUDE_INETLIB or INCLUDE_IPWRAP_INETLIB	The Wind River-specific implementations, inet_lnaof , inet_makeaddr_b , inet_makeaddr , inet_netof , inet_network , and inet_ntoa_b are also ported.
ip6Attach()	INCLUDE_IP6ATTACH or INCLUDE_IPWRAP_IPPROTO	No distinction between IPv4/v6 attach. See the API reference entry for functional changes.
ipAttach()	INCLUDE_IPATTACH or INCLUDE_IPWRAP_IPPROTO	No distinction between IPv4/v6 attach. See the API reference entry for functional changes.
netstat()	INCLUDE_NETSTAT or INCLUDE_IPWRAP_NETSTAT	See the API reference entry for functional changes.
ping()	INCLUDE_PING or INCLUDE_IPWRAP_PING	See the API reference entry for functional changes.
ping6()	INCLUDE_PING6 or INCLUDE_IPWRAP_PING6	See the API reference entry for functional changes.
routecc()	INCLUDE_ROUTECCMD or INCLUDE_IPWRAP_ROUTECCMD	See the API reference entry for functional changes.
sntpTimeGet()	INCLUDE_IPWRAP_SntpTIMEGET	

2.5.5 Removed Header Files

Table 2-4 lists Wind River Network Stack 3.1 header files that have been removed from Wind River Network Stack 6.6. All references to the header files in Table 2-4 must be removed or commented out to avoid compilation errors.

Some of the removed files listed in Table 2-4 may not have appeared in the Wind River General Purpose Platform, VxWorks Edition, 3.4, but were part of Wind River VxWorks Platforms, 3.4.

Table 2-4 Header Files Removed in Wind River Network Stack 6.6

adv_net.h	altq/altq.h	altq/altq_config_var.h
altq/altq_var.h	altq/if_altq.h	arch/arm/ansi.h
arch/arm/endian.h	arch/arm/machdep.h	arch/arm/param.h
arch/coldfire/ansi.h	arch/coldfire/endian.h	arch/coldfire/machdep.h
arch/coldfire/param.h	arch/mips/ansi.h	arch/mips/endian.h
arch/mips/machdep.h	arch/mips/param.h	arch/pentium/ansi.h
arch/pentium/endian.h	arch/pentium/machdep.h	arch/pentium/param.h
arch/ppc/ansi.h	arch/ppc/endian.h	arch/ppc/machdep.h
arch/ppc/param.h	arch/sh/ansi.h	arch/sh/endian.h
arch/sh/machdep.h	arch/sh/param.h	arch/simpentium/ansi.h
arch/simpentium/endian.h	arch/simpentium/machdep.h	arch/simpentium/param.h
arch/simso/ansi.h	arch/simso/endian.h	arch/simso/machdep.h
arch/simso/param.h	arch/types.h	bootpLib.h
bpfDrv.h	bsdSockLib.h	dhcp/common.h
dhcp/common_subr.h	dhcp/copyright_dhcp.h	dhcp/database.h
dhcp/dhcp.h	dhcp/dhcpc.h	dhcp/dhcpcBoot.h
dhcp/dhcpcCommonLib.h	dhcp/dhcpcInit.h	dhcp/dhcpcInternal.h
dhcp/dhcpcShow.h	dhcp/dhcpcStateLib.h	dhcp/dhcps.h

Table 2-4 Header Files Removed in Wind River Network Stack 6.6 (cont'd)

dhcp/hash.h	dhcp6cShow.h	dhcp6Lib.h
dhcpcBootLib.h	dhcpcLib.h	dhcprLib.h
dhcpsLib.h	dlink/gifNpt.h	dlink/stfNpt.h
dlink/tunnelLib.h	dlink/vlanTagLib.h	fastPath/fastPathFib.h
fastPath/fastPathIp.h	fastPath/fastPathIpLib.h	fastPath/fastPathLib.h
fastPath/fastPathMon.h	fastPath/fastPathPatTree.h	fastPath/fastPathUtil.h
icmpLib.h	if6Lib.h	ifIndexLib.h
ifLib.h	IGMPv2/igmp_constants.h	IGMPv2/igmp_display_string.h
IGMPv2/igmp_extrns.h	IGMPv2/igmp_globals.h	IGMPv2/igmp_prototypes.h
IGMPv2/igmp_state_machine.h	IGMPv2/igmp_state_machine_structures.h	IGMPv2/igmp_structures.h
IGMPv2/igmpCacheLib.h	IGMPv2/igmpPortLib.h	IGMPv2/igmpR.h
IGMPv2/igmpRouterLib.h	iocom.h	ip6Lib.h
ipLib.h	machdep.h	mbufLib.h
mbufSockLib.h	md5.h	mib/cidrLeaf.h
mib/cidrMapi.h	mib/cidrSkel.h	mib/mibApi.h
mip6/mip6.h	mip6/mip6_babymdd.h	mip6/mip6_command.h
mip6/mip6_constants.h	mip6/mip6_had.h	mip6/mip6_mnd.h
mip6/mip6_shisad.h	mip6/mip6_stat.h	net/af.h
net/bpf.h	net/bpfdesc.h	net/fastUdp6Lib.h
net/fastUdpLib.h	net/if_clone.h	net/if_media.h
net/if_mip.h	net/if_subr.h	net/ifaddrs.h
net/inet.h	net/mipsock.h	net/radix.h
net/raw_cb.h	net/utls/altqConfig.h	net/utls/ip6addrctl.h

Table 2-4 Header Files Removed in Wind River Network Stack 6.6 (cont'd)

net/utils/ndp.h	net/utils/prefixcmd.h	netconf.h
netCore.h	netinet/icmp_var.h	netinet/icmp6.h
netinet/igmp_var.h	netinet/in_msf.h	netinet/in_pcb.h
netinet/ip_var.h	netinet/ip4_ext_in.h	netinet/ip4_ext_out.h
netinet/ip6mh.h	netinet/ipfw.h	netinet/ipprotosw.h
netinet/pim.h	netinet/pim_var.h	netinet/sctp.h
netinet/sctp_constants.h	netinet/sctp_header.h	netinet/sctp_uio.h
netinet/sl_compress.h	netinet/tcp_debug.h	netinet/tcp_fsm.h
netinet/tcp_seq.h	netinet/tcp_timer.h	netinet/tcp_var.h
netinet/tcpip.h	netinet/udp_var.h	netinet/vsArp.h
netinet/vsData.h	netinet/vsDhcps.h	netinet/vsFastUdp.h
netinet/vsHost.h	netinet/vsIcmp.h	netinet/vsIgmp.h
netinet/vsIgmpR.h	netinet/vsIp.h	netinet/vsLib.h
netinet/vsM2.h	netinet/vsMcast.h	netinet/vsNetCore.h
netinet/vsProxyArp.h	netinet/vsRadix.h	netinet/vsRdisc.h
netinet/vsRip.h	netinet/vsShow.h	netinet/vsTcp.h
netinet/vsUdp.h	netinet6/icmp6.h	netinet6/in6_msf.h
netinet6/in6_pcb.h	netinet6/ip6_ext_in.h	netinet6/ip6_ext_out.h
netinet6/ip6_var.h	netinet6/mip6.h	netinet6/mip6_var.h
netinet6/nd6.h	netinet6/pim6.h	netinet6/pim6_var.h
netinet6/raw_ip6.h	netinet6/scope6_var.h	netinet6/tcp6_var.h
netinet6/udp6_var.h	osdep.h	poll.h
private/bpfLibP.h	private/clarinet.h	private/fastPathIpP.h
private/fastPathLibP.h	private/fastPathPatTreeP.h	private/m2LibP.h

Table 2-4 Header Files Removed in Wind River Network Stack 6.6 (cont'd)

private/nfsCacheLibP.h	private/routeShowP.h	protos/icmpv6Lib.h
protos/igmpLib.h	protos/ip6protosw.h	protos/mld6_var.h
protos/mldLib.h	protos/mrouteLib.h	protos/nd6Lib.h
protos/rarpLib.h	protos/sctpLib.h	protos/tcpLib.h
protos/udpLib.h	proxyArpLib.h	random.h
rdiscLib.h	resolv/nameser.h	resolv/resolv.h
resolvLib.h	rip/defs.h	rip/interface.h
rip/m2RipLeaf.h	rip/m2RipLib.h	rip/md5.h
rip/rip2.h	rip/ripLib.h	rip/table.h
ripngLib.h	route/avlRouteNodeLib.h	route/avltree.h
route/ipRouteLib.h	route/ipRouteNodeData.h	route/ipRouteNodeLib.h
route/llRouteNodeLib.h	route/ptRouteNodeLib.h	routeLib.h
rtadv/rtadvd.h	rtadv/timer.h	rtadvLib.h
rtsolLib.h	sntp.h	sntpsLib.h
sys/callout.h	sys/ds_conf.h	sys/mem_stru.h
sys/sockio.h	vs/vsBsdSock.h	vs/vsDhcpCommon.h
vs/vsDhcpr.h	vs/vsDns.h	vs/vsFastPath.h
vs/vsFastUdp6.h	vs/vsIcmpUtil.h	vs/vsIcmpv6.h
vs/vsIf.h	vs/vsIp6.h	vs/vsIpRoute.h
vs/vsLog.h	vs/vsMcast6.h	vs/vsMip6.h
vs/vsMld.h	vs/vsMroute.h	vs/vsNatptLib.h
vs/vsNd6.h	vs/vsNdp.h	vs/vsRarp.h

Table 2-4 Header Files Removed in Wind River Network Stack 6.6 (cont'd)

vs/vsRaw.h	vs/vsRipng.h	vs/vsRtadv.h
vs/vsSoSup.h	vs/vsSysctl.h	vs/vsTunnelLib.h
zbufLib.h	zbufSockLib.h	

2.6 Migrating Applications

This section discusses migrating certain types of network applications to Wind River Network Stack 6.6.

2.6.1 Migrating an Application that Uses Networking APIs

You cannot import an existing VxWorks Image Project (VIP) directly into Workbench. You must first create a new VIP and add the appropriate components, which are documented fully in *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*. Once you have created a new VIP, you can then add your previous application source code to that project. In some cases, you may need to rewrite parts of the code to reflect API changes to the protocols or technology.

Most of the transport and network layer protocols have new implementations. Consequently, there is no one-to-one correspondence or mapping of the component names between versions 3.1 and 6.6 of the network stack. In some cases, where you previously had to include several interdependent components, the new stack incorporates all of that support into a single component. For example, the IPv4 and IPv6 components include the appropriate UDP and ICMP support. There are no longer separate components for UDP and ICMP.

Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1 documents how to add the appropriate configuration components for a minimal stack that runs **ping**, and for a stack that includes any of the supported transport and networking protocols, such as TCP, IPv4, IPv6, and ARP.

2.6.2 Migrating a Socket-Based Application

Most socket-based applications that interface with the network stack can be migrated to use the updated network stack with minor clean-up and changes. Standard socket calls continue to be supported.

The following general process will help you to work through the changes required to migrate a socket-based application. For assistance using Workbench, see the *Wind River Workbench User's Guide*.

Any new socket components required for your project are automatically included when you build the network stack. For more socket-related migration information, see [4.9 Internet and Local Domain Sockets](#), p.77.

1. Create a new VxWorks Image Project (VIP).
 2. Import your source files into the new VIP.
 3. Compile the project and check the error output.
- Missing header files cause compile errors similar to the following example:

```
blasteeTCPvx.c", line 83: error (dcc:1621): can't find include file
zbufSockLib.h

"blasteeTCPvx.c", line 84: error (dcc:1621): can't find include file
zbufLib.h
```

Remove references to all missing header files. For a list of header files that are obsolete, see [2.5.5 Removed Header Files](#), p.23. For information on replacing or functionally replacing obsolete features, see the appropriate section in this guide.

- References to obsolete features may cause compile error sequences similar to the following example:

```
"blasteeTCPvx.c", line 132: error (dcc:1525): identifier zid not declared
"blasteeTCPvx.c", line 132: error (dcc:1633): parse error near 'zid'
"blasteeTCPvx.c", line 132: error (dcc:1206): syntax error
"blasteeTCPvx.c", line 132: fatal error (dcc:1340): can't recover from
earlier errors
```

Check the error references to determine if the project is using an obsolete feature. In this example the project is attempting to use **zBuf**, which is not supported in Wind River Network Stack 6.6, and must be replaced by standard sockets.

4. Remove any references to the virtual stack from the project. The virtual stack is obsolete and has been replaced by a virtual router feature. For more information, see [3.14 Virtual Stack](#), p.53.
5. Recompile the project and ensure no errors remain.

3

Wind River Network Stack: Transport and Network Protocols

- 3.1 Introduction 32
- 3.2 Migrating to SMP 35
- 3.3 Socket Options 36
- 3.4 IPv4 and IPv6 Components 37
- 3.5 ARP Components 39
- 3.6 Proxy ARP 40
- 3.7 Multicasting Components 41
- 3.8 Show Routine Components 42
- 3.9 Utility Components 44
- 3.10 RIP Components 44
- 3.11 RIPng Components 48
- 3.12 Routing APIs 50
- 3.13 Routing Sockets 51
- 3.14 Virtual Stack 53

3.1 Introduction

Major changes to Wind River Network Stack were introduced in the Wind River Network Stack 6.5 release. Additional changes were also introduced in the Wind River Network Stack 6.6 release.

This chapter will help you plan your migration from Wind River Network Stack 3.x to the current release, Wind River Network Stack 6.6. Changes between the last three versions of Wind River Network Stack—3.x, 6.5, and 6.6—are highlighted in [3.1.1 Feature Release Matrix](#), p.32. These changes are further described in [3.1.2 Changes in Wind River Network Stack 6.5](#), p.34, and [3.1.3 Changes in Wind River Network Stack 6.6](#), p.34.

There are four chapters in this guide that contain migration information for the network stack:

[2. Wind River Network Stack Migration Overview](#)

Provides introductory information on network stack migration. Describes key concepts and migration information common to most or all networking components.

[3. Wind River Network Stack: Transport and Network Protocols](#) (this chapter)

Provides migration details and component and API mapping for the components of the core network stack, including TCP/IP, multicast, and routing.

[4. Wind River Network Stack: Application Protocols](#)

Provides migration details and component and API mapping for the network application components, including DHCP and DNS, and information on programming with sockets.

[5. Wind River Network Stack: Interfaces and Drivers](#)

Provides migration details and information on changes to libraries and routines for lower-level network stack components, including the MUX and interface configuration.

3.1.1 Feature Release Matrix

[Table 3-1](#) provides a list of Wind River Network Stack transport and network layer features, indicating which releases support them. Features that are currently not supported in the Wind River Network Stack 6.6 may be available in a future release.

Table 3-1 Network Stack Transport and Network Protocols - Feature Release Matrix

Feature	3.1	6.5	6.6
Differentiated Services	Not supported	●	●
IPv4-only network stack	●	● (New implementation)	●
IPv4/IPv6 dual stack	●	● (New implementation)	●
Fastpath	●	● (New implementation)	●
Multicast Proxy Router (host-side support)	●	● (New implementation)	●
IPv4 and IPv6	●	● (New implementation)	●
MPLS Dataplane Support	Not supported	●	●
NDP	●	● (New implementation)	●
Policy-Based Routing	Not supported	●	●
Proxy ARP Server	●	● (New implementation)	●
RIP v1, v2	●	● (New implementation)	●
RIPng	●	● (New implementation)	●

Table 3-1 Network Stack Transport and Network Protocols - Feature Release Matrix (cont'd)

Feature	3.1	6.5	6.6
Routing Sockets	●	● (New implementation)	●
Routing Table	●	● (New implementation)	●
SCTP	●	Not supported	Not currently supported
TCPv4 and TCPv6	●	● (New implementation)	●
UDPv4 and UDPv6	●	● (New implementation)	●
Virtual Router	Not supported	●	●
Virtual Stack (replaced by Virtual Router)	●	Obsolete	Obsolete
VRRP	Not supported	●	●

3.1.2 Changes in Wind River Network Stack 6.5

If you are upgrading from Wind River Network Stack 3.x, there are many changes in Wind River Network Stack 6.5 that may require migration of existing projects and code. See sections [3.4 IPv4 and IPv6 Components](#), p.37, through [3.14 Virtual Stack](#), p.53, for further information.

3.1.3 Changes in Wind River Network Stack 6.6

If you are upgrading from Wind River Network Stack 6.5, there are relatively few changes in Wind River Network Stack 6.6 that require migration of existing projects and code. Sections [3.4 IPv4 and IPv6 Components](#), p.37, through [3.14 Virtual](#)

[Stack](#), p.53, describe these changes in greater detail. See sections [3.5 ARP Components](#), p.39 and [3.14 Virtual Stack](#), p.53 for further information.

In addition, the following IPCOM routines that were introduced in Wind River Network Stack 6.5 are deprecated in Wind River Network Stack 6.6:

- `ipcom_run_cmd()`
- `ipmcp_cmd()`

Command interpreter commands should not be called programmatically using `ipcom_run_command()`. There are documented public APIs for all functionality that is available as shell commands; these should be used instead.

3.2 Migrating to SMP



NOTE: SMP support for VxWorks is available as an optional product. However, default SMP system images for the VxWorks simulator are provided with the standard VxWorks installation as an introduction to the product.

Wind River Network Stack 6.6 can optionally be implemented with symmetric multiprocessing (SMP) capabilities. To migrate an existing uniprocessor-enabled stack, you must import it into an SMP-capable VxWorks Image Project.

Creating an SMP-Capable VxWorks Image Project

Once you have rebuilt the Platform source code, you must create a new project with the SMP option enabled. There are two ways to enable this option:

- If you are using Workbench, select **Build with SMP options** in the Options page of the New VxWorks Image Project wizard.
- If you are using `vxprj`, add the flag `-smp` to the `create` command.

3.3 Socket Options

A number of socket commands are available in Wind River Network Stack 6.6 to provide backwards compatibility with Wind River Network Stack 3.x. The location where socket option definitions are stored has also changed. In addition, there an interface-related socket option is available.

New Socket Commands

New socket commands have been introduced for multicast routing. For further information, see [IPv4 Multicast Routing set/getsockopt Options](#), p.41, or [IPv6 Multicast Routing set/getsockopt Options](#), p.42.

Socket Option Definitions

In Wind River Network Stack 3.x and earlier releases, **coreip** IPv6 socket options are defined in the following location:

installDir/vxworks-6.x/target/h/wrn/coreip/netinet6/in6_var.h

In Wind River Network Stack 6.x, socket options and the required structure definitions are located in the *installDir/vxworks-6.x/target/h/wrn/coreip/ipnet* directory. These definitions can be found in the following new files:

- **ipioctl.h**
- **ipioctl_var.h**
- **mpls_var.h**
- **pfkeyv2.h**
- **policy_routing.h**
- **ppp_var.h**
- **qos.h**

Interface-Related Socket Options

The socket option **SIOCGIFCAP**, which takes the struct **ifreq** as an argument, gets interface capabilities.

3.4 IPv4 and IPv6 Components

This section describes the IPv4 and IPv6 components and API mapping.

3.4.1 IPv4 and IPv6 Configuration

Table 3-2 shows the configuration component mapping for the IPv4 components.

Table 3-2 IPv4 Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_HOSTCACHE	Obsolete
INCLUDE_ICMPV4	Support included in INCLUDE_IPCOM_USE_INET
INCLUDE_IPV4	INCLUDE_IPCOM_USE_INET
INCLUDE_RAWV4	Support automatically included
INCLUDE_TCP_DEBUG	Functionality included in INCLUDE_IPTCP
INCLUDE_TCPV4	INCLUDE_IPTCP ^a
INCLUDE_UDPV4	Support included in INCLUDE_IPCOM_USE_INET

a. Used for both IPv4 and IPv6, configurable at library build time.

Table 3-3 shows the configuration component mapping for the IPv6 components.

Table 3-3 IPv6 Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_ICMPV6	Support included in INCLUDE_IPCOM_USE_INET6
INCLUDE_IPV6	INCLUDE_IPCOM_USE_INET6
INCLUDE_MIPV6	INCLUDE_IPMIP6, INCLUDE_IPMIP6MN
INCLUDE_ND	Support included in INCLUDE_IPCOM_USE_INET6
INCLUDE_RAWV6	Support automatically included

Table 3-3 IPv6 Component Migration (cont'd)

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_TCPV6	INCLUDE_IPTCP ^a
INCLUDE_UDPV6	Support included in INCLUDE_IPCOM_USE_INET6

a. Used for both IPv4 and IPv6, configurable at library build time.

3.4.2 API Mapping

This section provides tables that list the Wind River Network Stack 3.x core stack routines and the replacement API, if available, in Wind River Network Stack 6.x.

Shell commands in Wind River Network Stack 6.x are used to replace VxWorks configuration commands and network show routines for many components, and are listed in the tables in this section.

TCP/IP Layer Core Networking Routines

Table 3-4 provides the API mapping for the core networking routines.

Table 3-4 Core Networking APIs

3.x API	6.x API or Method
sysctl	sysctl (with changes)
ifRouteDelete()	Obsolete
netVersionShow()	version shell command
rarpGet()	Obsolete
splimp()	Obsolete
splnet()	Obsolete
splnet2()	Obsolete
splx()	Obsolete
wakeup()	Obsolete

Table 3-4 Core Networking APIs (cont'd)

3.x API	6.x API or Method
ndp()	ndp shell command
prefixcmd()	Obsolete

ICMP Routines

ICMP functionality is now included in the IP components. Table 3-5 lists the old APIs and the new implementation solutions.

Table 3-5 ICMP APIs

3.x API	6.x API or Method
icmpShowInit()	Obsolete
icmpMaskGet()	A string representation of the mask is returned by the ifconfig command. A binary representation is returned through the SIOCGIFNETMASK ioctl.

3.5 ARP Components

This section provides a table that lists the Wind River Network Stack 3.x ARP routines and the replacement API, if available, in Wind River Network Stack 6.x.

API Mapping for ARP Routines

Table 3-6 provides the API mapping for the ARP protocols.

Table 3-6 ARP APIs

3.x API	6.x API or Method
arpFlush()	Two methods exist: <ul style="list-style-type: none">▪ Use the ioctl() function SIOCDARP with the struct arpreq as an argument.▪ Use the shell command arp -A to delete all ARP entries.
arpResolve()	Two methods exist: <ul style="list-style-type: none">▪ Use the ioctl() function SIOCPARP with the struct arpreq as an argument.▪ Use the shell command arp -r hostaddr, which sends an ARP request to the specified host address.

3.6 Proxy ARP

The proxy ARP implementation in Wind River Network Stack 6.x does not function exactly as it did in previous releases. You can create proxied networks that are on the same logical subnet as the main proxy interface. You must either add specific host routes to the individual nodes on the proxied network, or arrange the IP addresses of the nodes on the proxied network to be a subset of the larger main subnet, and add a route to the more specific (proxied) subnet on the proxy ARP server.

For details on the current proxy ARP implementation, see the *Wind River Network Stack Programmer's Guide, 6.6, Volume 1*.

3.7 Multicasting Components

The General Purpose Platform now includes host-side support for all versions of the IGMP and MLD multicasting protocols:

- IGMPv1
- IGMPv2
- IGMPv3
- MLDv1
- MLDv2

3.7.1 Socket Commands

This section provides tables that list the socket options available in Wind River Network Stack 6.x.

IPv4 Multicast Routing set/getsockopt Options

For backwards compatibility purposes, the IPv4 multicast routing socket options are defined in *installDir/vxworks-6.x/target/h/wrn/coreip/netinet/ip_mroute.h*. [Table 3-7](#) lists these new options.

Table 3-7 IPv4 Multicast Routing Socket Options

Socket Option	Comments
MRT_INIT	int ; initialize mrouterd
MRT_DONE	int ; shut down mrouterd
MRT_ADD_VIF	struct vifctl ; add virtual interface
MRT_DEL_VIF	struct vifctl ; delete virtual interface
MRT_ADD_MFC	struct mfcctl ; add a multicast forwarding cache entry
MRT_DEL_MFC	struct mfcctl ; delete a multicast forwarding cache entry
MRT_VERSION	int ; returns mrouterd version number
MRT_ASSERT	int ; enable assert processing
MRT_PIM	int ; enable PIM processing

IPv6 Multicast Routing set/getsockopt Options

For backwards compatibility purposes, the IPv6 multicast routing socket options are defined in *installDir/vxworks-6.x/target/h/wrn/coreip/netinet6/ip6_mroute.h*. [Table 3-8](#) lists these new options.

Table 3-8 IPv6 Multicast Routing Socket Options

Socket Option	Comments
MRT6_INIT	int; initialize mroute
MRT6_DONE	int; shut down mroute
MRT6_ADD_VIF	struct mif6ctl; add virtual interface
MRT6_DEL_VIF	struct mif6ctl; delete virtual interface
MRT6_ADD_MFC	struct mf6cctl; add a multicast forwarding cache entry
MRT6_DEL_MFC	struct mf6cctl; delete a multicast forwarding cache entry
MRT6_PIM	int; enable PIM processing

3.8 Show Routine Components

This section describes the show routine component and API mapping.

3.8.1 Show Routine Configuration

The following show routine components are no longer used:

INCLUDE_ICMP_SHOW

INCLUDE_NET_SHOW

INCLUDE_NET_ROUTE_SHOW

INCLUDE_UDP_SHOW

INCLUDE_IGMP_SHOW

INCLUDE_NET_IF_SHOW

INCLUDE_TCP_SHOW

However, the functionality that they provided is now available from shell commands. For more information, see [3.8.2 API Mapping](#), p.43.

3.8.2 API Mapping

Table 3-9 lists the previous show routine APIs and the replacement API, if available, in Wind River Network Stack 6.x.

Table 3-9 Show Routine APIs

3.x API	6.x API or Method
hostShow()	hostShow()
ipstatShow()	netstat() (wrapper)
ip6statShow()	netstat() (wrapper)
inetstatShow()	netstat() (wrapper)
inet6statShow()	netstat() (wrapper)
icmpstatShow()	netstat() (wrapper)
icmp6statShow()	Obsolete
igmpShow()	netstat() (wrapper)
mRouteShow()	netstat() (wrapper) or routeCmd() (wrapper)
netPoolShow()	netPoolShow()
netShow()	netstat() (wrapper)
routeShow()	netstat() (wrapper)
routeStatShow()	routeCmd() (wrapper)
tcpstatShow()	netstat() (wrapper)
tcpDebugShow()	Obsolete
tcpShowInit()	Obsolete
udpstatShow()	netstat() (wrapper)
udpShowInit()	Obsolete
vsShow()	Obsolete

3.9 Utility Components

The following NETSTAT components are obsolete:

INCLUDE_NETSTAT_UN_COMP	INCLUDE_NETSTAT_ICMPV4
INCLUDE_NETSTAT_ICMPV6	INCLUDE_NETSTAT_IGMP
INCLUDE_NETSTAT_MROUTEV6	INCLUDE_NETSTAT_IPV4
INCLUDE_NETSTAT_IPV6	INCLUDE_NETSTAT_RAWV6
INCLUDE_NETSTAT_SCTP	INCLUDE_NETSTAT_TCP
INCLUDE_NETSTAT_UDP	INCLUDE_NETSTAT_IF
INCLUDE_NETSTAT_MROUTEV6	INCLUDE_NETSTAT_ROUTE
INCLUDE_NETSTAT6	

All NETSTAT components are replaced by the **netstat** command, which can be included in a build using **INCLUDE_NETSTAT_CMD**.

In addition, the **microtime** utility is obsolete.

3.10 RIP Components

This section describes the RIP component and API mapping.

3.10.1 RIP Configuration

[Table 3-10](#) shows the configuration component mapping for RIP. For descriptions of the new components, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

Table 3-10 RIP Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_RIP	INCLUDE_IPRIP
No equivalent	SELECT_IPRIP_IFCONFIG INCLUDE_IPRIP_IFCONFIG_1 INCLUDE_IPRIP_IFCONFIG_2 INCLUDE_IPRIP_IFCONFIG_3 INCLUDE_IPRIP_IFCONFIG_4
No equivalent	SELECT_IPRIP_STATIC_ROUTES INCLUDE_IPRIP_STATIC_ROUTE_1 INCLUDE_IPRIP_STATIC_ROUTE_2 INCLUDE_IPRIP_STATIC_ROUTE_3
No equivalent	INCLUDE_IPRIP_CTRL_CMD

3.10.2 API Mapping

This section provides tables that list the RIP APIs and the replacement API, if available, in Wind River Network Stack 6.x.

Shell commands in Wind River Network Stack 6.x are used to replace VxWorks configuration commands and network show routines for many components, and are listed in the tables in this section.

For details on using RIP and RIPng see *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1* and the API reference entries.

[Table 3-11](#) provides the RIP API mappings.

Table 3-11 RIP APIs

3.x API	6.x API or Method	Comments
<code>ripAddrsXtract()</code>	<code>iprip_ifctrl()</code> with <code>IPRIP_IFCTRL_GET_IP_ADDRESS</code>	
<code>ripAuthHook()</code>	Obsolete	The new RIP implementation supports traditional simple and gated md5-style authentication without a function hook. It is enabled per interface by setting auth-simple= or auth-md5= configuration options.
<code>ripAuthHookAdd()</code>	Obsolete	See comments above.
<code>ripAuthHookDelete()</code>	Obsolete	See comments above.
<code>ripAuthKeyAdd()</code>	<code>ripctrl</code> shell command or <code>iprip_ifctrl()</code> or <code>iprip_ifopen()</code>	RIP authentication is done on a per-interface basis.
<code>ripAuthKeyDelete()</code>	none	Obsolete
<code>ripAuthKeyFind()</code>	none	Obsolete
<code>ripAuthKeyFindFirst()</code>	<code>ripctrl</code> shell command or <code>iprip_ifctrl()</code> with <code>IPRIP_IFCTRL_GET_rip2IfConfAuthType</code>	
<code>ripAuthKeyInMD5()</code>	none	Obsolete. RIP authentication is done on a per-interface basis.
<code>ripAuthKeyOut1MD5()</code>	none	Obsolete
<code>ripAuthKeyOut2MD5()</code>	none	Obsolete
<code>ripAuthKeyShow()</code>	<code>ripctrl</code> shell command or <code>iprip_ifctrl()</code> with <code>IPRIP_IFCTRL_GET_rip2IfConfAuthType</code>	

Table 3-11 RIP APIs (cont'd)

3.x API	6.x API or Method	Comments
ripDebugLevelSet()	none	Obsolete
ripFilterDisable()	none	Obsolete
ripFilterEnable()	none	Obsolete
ripIfAddrExcludeListAdd()	none	Obsolete. RIP is enabled on a per-interface basis using iprip_ifopen() . Excluding an interface from RIP is not required.
ripIfAddrExcludeListDelete()	none	Obsolete. RIP is enabled on a per-interface basis using iprip_ifopen() . Excluding an interface from RIP is not required.
ripIfAddrReset()	iprip_ifclose()	
ripIfExcludeListAdd()	none	Obsolete. RIP is configured on a per-interface basis using iprip_ifopen() .
ripIfExcludeListDelete()	none	Obsolete. RIP is configured on a per-interface basis using iprip_ifopen() .
ripIfExcludeListShow()	none	Obsolete. RIP is configured on a per-interface basis. Retrieve RIP interface-specific information using iprip_ifctrl() .
ripIfReset()	iprip_ifclose()	
ripIfSearch()	none	Obsolete

Table 3-11 **RIP APIs** (cont'd)

3.x API	6.x API or Method	Comments
<code>ripIfShow()</code>	none	Obsolete. RIP is configured on a per-interface basis. Retrieve RIP interface-specific information using <code>iprip_ifctrl()</code> .
<code>ripLeakHookAdd()</code>	none	Obsolete
<code>ripLeakHookDelete()</code>	none	Obsolete
<code>ripLibInit()</code>	<code>iprip_init()</code>	
<code>ripRouteHookAdd()</code>	<code>iprip_rtadd()</code>	
<code>ripRouteHookDelete()</code>	<code>iprip_rtdelete()</code>	
<code>ripRouteShow()</code>	<code>iprip_ctrl()</code>	
<code>ripSendHookAdd()</code>	none	Obsolete
<code>ripSendHookDelete()</code>	none	Obsolete
<code>ripShutdown()</code>	<code>iprip_exit()</code>	
<code>ripUpdateDelaySet()</code>	none	Obsolete

3.11 RIPng Components

The RIPng implementation from Wind River Network Stack 3.1 has been ported to use the internal functionality of Wind River Network Stack 6.x.

3.11.1 Changes to RIPng Files

The C files for RIPng have been moved from `installDir/target/src/wrn/coreip/apps/ripng` to `installDir/components/ip_net2-6.x/ipripng/src`.

The RIPng .h files have been moved from *installDir/target/h/wrn/coreip/apps/ripng* to *installDir/components/ip_net2-6.x/ipmapng/include*.

The file names have changed, and new files have been added, as shown in [Table 3-12](#).

Table 3-12 RIPng File Name Changes

3.x File Names	6.x File Names
ripngLib.c, route6d.c	ipmapng.c, ipmapng_cmd_show.c, ipmapng_daemon.c, ipmapng_util.c
route6d.h	ipmapng.h, ipmapng_constant.h

3.11.2 RIPng Configuration

[Table 3-13](#) shows the configuration component mapping for RIPng. For information on working with RIPng, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

Table 3-13 RIPng Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_RIPNG	INCLUDE_RIPNG
No equivalent	INCLUDE_RIPNG_CTRL_CMD

3.11.3 API Mapping

RIPng is fully ported and integrated with Wind River Network Stack 6.6. Although the APIs have not functionally changed, RIPng has new shell commands and can be started and stopped in different ways. The **ripngStart()** and **ripngStop()** routines are carried forward.

3.12 Routing APIs

Table 3-14 provides the API mapping for routing APIs. The **route** shell command is described in the routing chapter in the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

Table 3-14 Routing API Mapping

3.x API	6.x API	Comments
routeec()	routeec() (wrapper) or route command	routeec() is still provided and can be used to add and delete routes. See the reference entry for routeec() .
routeAdd()	RTM_ADD routing socket message or route command	Use route add command
routeNetAdd()	RTM_ADD routing socket message or route command	Use route add command. For example, to route to 10.0.0.0/8 through gateway 192.168.0.1: route add -net -prefixlen 8 10.0.0.0 192.168.0.1
mRouteAdd()	RTM_ADD routing socket message or route command	Use route add command. For example, to route to 10.1.2.3 through gateway 192.168.0.1: route add 10.1.2.3 192.168.0.1
mRouteEntryAdd()	RTM_ADD routing socket message or route command	Use route add command
routeDelete()	RTM_DELETE routing socket message or route command	Use route delete command. For example: route delete 10.1.2.3
mRouteDelete()	RTM_DELETE routing socket message or route command	Use route delete command
mRouteEntryDelete()	RTM_DELETE routing socket message or route command	Use route delete command
ipRouteLibLogLevel(S,G)et()	Use route monitor command	

Table 3-14 Routing API Mapping (cont'd)

3.x API	6.x API	Comments
<code>ipRouteLibLogLevelGet()</code>	none	Obsolete
<code>ipRouteLibLogLevelSet()</code>	none	Obsolete
<code>ipRouteTableCreate()</code>	RTM_NEWVR routing socket message	
<code>ipRouteTableDestroy()</code>	RTM_DELVR routing socket message	
<code>ipRouteTableSet()</code>	none	Obsolete

3.13 Routing Sockets

There are a few changes to routing sockets and how they are used in Wind River Network Stack 6.x. The extended messages provided in previous releases are now obsolete, but all standard messages are supported as in the previous releases.

The following section briefly describes the changes. For more information, see the routing sockets chapter in the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

3.13.1 Routing Socket Configuration

The routing sockets component is included when you build the network stack. Routing sockets are also enabled or disabled in a network stack configuration header file. Routing sockets are enabled by default in:

`installDir/components/ip_net2-6.x/ipnet2/ipnet_config.h`

To disable routing sockets you must edit the `ipnet_config.h` file, and rebuild your project. For more information see the routing sockets chapter in the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

[Table 3-15](#) shows the configuration component mapping for including routing sockets in a project.

Table 3-15 **Routing Socket Component Migration**

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_ROUTING_SOCKET	INCLUDE_IPNET_USE_ROUTE SOCK

3.13.2 **Ranking Routes in the Route Table**

Previously, routes were ranked by age in the host stack route table, and by a user-assigned weight in the router stack route table. Now, the routes are ranked by a hopcount field, and ECMP routing algorithms are used to determine the ranking for routes with the same hopcount.

3.13.3 **Routing Socket Messages**

The following routing socket messages previously used for multicast address additions and deletions are no longer supported:

- **RTM_NEWMADDR**
- **RTM_DELMADDR**

Multicast address additions to, or deletions from, the routing table are now reported using the following messages:

- **RTM_NEWADDR**
- **RTM_DELADDR**

Extended Messages

Wind River previously provided extended routing socket messages that were available in a router stack build of the network stack (using the now obsolete **-DROUTER_STACK** build option).

The following messages are now obsolete:

RTM_ADDEXTRA	RTM_DELEXTRA
RTM_NEWCHANGE	RTM_NEWGET
RTM_GETALL	RTM_NEWIPROUTE
RTM_OLDIPROUTE	

There are two new extended messages for routing sockets that are used for virtual router management:

- RTM_NEWVR
- RTM_DELVR

For details, see the virtual router and routing sockets chapters in the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

3.14 Virtual Stack

The virtual stack feature available in Wind River Network Stack 3.1 is now obsolete and has been replaced by a more flexible and scalable solution called *virtual routing*.

3.14.1 Overview

In Wind River Network Stack 3.1, you are limited to one virtual stack for a total of two stack instances on a single target. Because the new virtual router solution implements multiple routing tables and essential features instead of multiple stacks, you can now have over 65,000 virtual routers on a single target. You are limited only by the capacity of your hardware.

3.14.2 Virtual Stack Configuration

Along with the other compile flags from Wind River Network Stack 3.1, the **-DVIRTUAL_STACK** build option is no longer applicable. The virtual routing implementation has little impact on the size of the stack and support is therefore automatically included when you build Wind River Network Stack 6.6.

Creation, configuration, and management of the virtual routers is done through the **route** and **ifconfig** shell commands. Any code using the virtual stack feature must be redesigned to use virtual routing. For details on using the virtual routing feature, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 1*.

4

Wind River Network Stack: Application Protocols

- 4.1 Introduction 56
- 4.2 DHCP (IPv4 and IPv6) Components 58
- 4.3 DNS Components 65
- 4.4 FTP Components 68
- 4.5 Ping Components 71
- 4.6 SNTP Components 72
- 4.7 Telnet Components 74
- 4.8 TFTP Components 75
- 4.9 Internet and Local Domain Sockets 77
- 4.10 RTP 86
- 4.11 NFS Client and Server 86

4.1 Introduction

This chapter provides migration information on the following components of the network stack that implement standard application-layer protocols, and applications that use standard socket calls

There are four chapters in this guide that contain migration information for the network stack:

2. Wind River Network Stack Migration Overview

Provides introductory information on network stack migration. Describes key concepts and migration information common to most or all networking components.

3. Wind River Network Stack: Transport and Network Protocols

Provides migration details and component and API mapping for the components of the core network stack, including TCP/IP, multicast, and routing.

4. Wind River Network Stack: Application Protocols (this chapter)

Provides migration details and component and API mapping for the network application components, including DHCP and DNS, and information on programming with sockets.

5. Wind River Network Stack: Interfaces and Drivers

Provides migration details and information on changes to libraries and routines for lower-level network stack components, including the MUX and interface configuration.

4.1.1 Feature Release Matrix

Table 4-1 lists implementations of application-layer protocols and gives their status in the current release.

Table 4-1 Network Stack Application Protocols - Feature Release Matrix

Feature	3.1	6.5	6.6
BOOTP	●	Obsolete. Messages can be handled by DHCP.	Obsolete. Messages can be handled by DHCP.
DHCP (IPv4) Server, Relay Agent, and Client	●	● (New implementation)	●
DHCP (IPv6) Server, Relay Agent, and Client	●	● (New implementation)	●
DNS Client	●	● (New implementation)	●
FTP Client and Server	●	● (New implementation)	●
NFS Server and Client	●	● (New file locations)	●
Ping and Ping6	●	● (New implementation)	●
RLOGIN	●	●	●
RPC	●	●	●
RSH (remLib)	●	●	●
SNTP Client	●	● (New implementation)	●
Telnet Client	●	●	●
Telnet Server	●	● (New implementation)	●
TFTP Client and Server	●	● (New implementation)	●

4.1.2 Shell Commands and API Changes

The new implementations for certain protocols bring with them new components, shell commands, and APIs. The following sections compare the build components, shell commands, and APIs in Wind River Network Stack 3.1 and Wind River Network Stack 6.x for each protocol implementation.

For detailed information on the components and configuration parameters for any of the protocols, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 2*.

In addition, there are separate tables for changes in the socket options available with the `getsockopt()` and `setsockopt()` socket calls. For detailed information on the shell commands for an implementation, refer to the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 2*. For detailed information on APIs, see the API reference pages.

4.2 DHCP (IPv4 and IPv6) Components

This section describes DHCP (for IPv4 and IPv6) components and API mapping.

4.2.1 DHCP Configuration

[Table 4-2](#) provides the component mapping for DHCP (for IPv4).

Table 4-2 **DHCP (for IPv4) Component Migration**

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_DHCPC	INCLUDE_IPDHCPC
INCLUDE_DHCP_CORE	No equivalent
INCLUDE_DHCPC_BOOT	No equivalent
INCLUDE_DHCPC_LEASE_CLEAN	No equivalent
INCLUDE_DHCPC_LEASE_GET	No equivalent

Table 4-2 **DHCP (for IPv4) Component Migration** (cont'd)

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_DHCPC_LEASE_SAVE	No equivalent
INCLUDE_DHCPC_LEASE_TEST	No equivalent
INCLUDE_DHCPC_SHARE	No equivalent
INCLUDE_DHCPC_SHOW	No equivalent
INCLUDE_DHCPR	INCLUDE_IPDHCPR
INCLUDE_DHCPS_SHARE	No equivalent
INCLUDE_DHCPS	INCLUDE_IPDHCPS

[Table 4-3](#) provides the configuration component mapping for DHCP (for IPv6) components.

The DHCPv6 server and relay agent are implemented in a single module. For information on configuring these components and enabling either the server or the relay agent, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 2*.

Table 4-3 **DHCP (for IPv6) Component Migration**

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_DHCP6C	INCLUDE_IPDHCP6C
INCLUDE_DHCP6C_SHOW	No equivalent
INCLUDE_DHCP6R	No equivalent
INCLUDE_DHCP6S	INCLUDE_IPDHCP6S
No equivalent.	INCLUDE_IPDHCP6S_CMD

4.2.2 API Mapping

[Table 4-4](#) provides the API mapping for the DHCPv4 and DHCPv6 components. There are additional configuration parameters that can be configured at run time

using shell commands. For more information, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 2*.

Table 4-4 **DHCPv4 and DHCPv6 APIs and Shell Commands**

3.1 API	6.x Technique
bootpLibInit()	none (obsolete: BOOTP is no longer supported.)
bootpLibMultiInit()	none (obsolete)
bootpMsgGet()	none (obsolete)
bootpParamsGet()	none (obsolete)
dh6cInfoGet()	Configure by means of the sysvar shell command.
dhcp6c()	none (obsolete: The dhcpv6 client process is created and started by IPD start-up code upon boot.)
dhcp6cLibInit()	ipdhcpc_create() (routine) dhcp start (shell command)
dhcp6cStop()	none (obsolete)
dhcp6rLibInit()	Use the DHCPS6_MODE configuration parameter (or the ipdhcps6.mode sysvar) to switch from server to relay agent mode.
dhcp6rStop()	none (obsolete)
dhcp6relay()	none (obsolete)

Table 4-4 DHCPv4 and DHCPv6 APIs and Shell Commands (cont'd)

3.1 API	6.x Technique
dhcps6s()	dhcps6 shell command with the keyword start to start the ipdhcps6 daemon (" dhcps6 start "), and the following routines: <ul style="list-style-type: none"> ▪ ipdhcps6_subnet_add() ▪ ipdhcps6_subnet_delete() ▪ ipdhcps6_option_add() ▪ ipdhcps6_option_delete() ▪ ipdhcps6_prefix_add() ▪ ipdhcps6_prefix_delete() ▪ ipdhcps6_host_add() ▪ ipdhcps6_host_delete() ▪ ipdhcps6_user_add() ▪ ipdhcps6_user_delete()
dhcps6sLibInit()	none (obsolete)
dhcps6sStop()	dhcps6 shell command with the keyword stop to start the ipdhcps6 daemon: dhcps6 stop .
—	ipdhcps6_subnet_list()
—	ipdhcps6_subnet_to_string()
—	ipdhcps6_option_list()
—	ipdhcps6_option_to_string()
—	ipdhcps6_prefix_get()
—	ipdhcps6_prefix_list()
—	ipdhcps6_host_list()
—	ipdhcps6_host_to_string()
—	ipdhcps6_user_list()
—	ipdhcps6_version()
dhcpcBind()	ifconfig dhcp shell command
dhcpcBootBind()	none (obsolete)

Table 4-4 **DHCPv4 and DHCPv6 APIs and Shell Commands** (cont'd)

3.1 API	6.x Technique
dhcpcBootInformGet()	none (obsolete)
dhcpcBootInit()	none (obsolete)
dhcpcCacheHookAdd()	none (No event hooks exist. Negotiation is started from command and there is no way to do it synchronously/asynchronously.)
dhcpcCacheHookDelete()	none (obsolete)
dhcpcEventHookAdd()	none (obsolete)
dhcpcEventHookDelete()	none (obsolete)
dhcpcInformGet()	Use the ifconfig shell command to view the interface status for an interface configured to use DHCP.
dhcpcInit()	ifconfig with dhcp option (Negotiation is started from the command line.) dhcpcInit had an argument that you could set to false if you wanted to use the DHCP client but not have it assign the IP address from the server to the local interface (if instead you wanted to do that yourself). The current DHCP client implementation does not allow you to do this.
dhcpcLibInit()	none (obsolete: The IPDHCP daemon is automatically started after boot.)
dhcpcOptionAdd()	none (obsolete)
dhcpcOptionGet()	none (obsolete)
dhcpcOptionSet()	ipdhcpc.requested_options (sysvar) Static configuration only.
dhcpcParamsGet()	none (obsolete)
dhcpcParamsShow()	Obsolete

Table 4-4 DHCPv4 and DHCPv6 APIs and Shell Commands (cont'd)

3.1 API	6.x Technique
<code>dhcpcRelease()</code>	<code>ifconfig -dhcp</code>
<code>dhcpcServerGet()</code>	none (obsolete)
<code>dhcpcServerShow()</code>	none (obsolete)
<code>dhcpcShowInit()</code>	none (obsolete)
<code>dhcpcShutdown()</code>	<code>ipdhcpc_destroy()</code> or <code>dhcpc</code> shell command with the keyword <code>stop</code> .
<code>dhcpcTimerGet()</code>	none (obsolete)
<code>dhcpcTimersShow()</code>	none (obsolete)
<code>dhcpcVerify()</code>	none (obsolete)
<code>dhcpsAddressHookAdd()</code>	<code>ipdhcps_host_add()</code> The IPDHCPs daemon is automatically started after boot.
<code>dhcpsInit()</code>	<code>ipdhcps_start_hook()</code>
<code>dhcpsLeaseEntryAdd()</code>	<code>ipdhcps_pool_add()</code> <code>ipdhcps_host_add()</code> <code>ipdhcps_subnet_add()</code> <code>ipdhcps_class_add()</code> <code>ipdhcps_option_add()</code> <code>dhcps pool add</code> <code>dhcps host add</code> <code>dhcps subnet add</code> <code>dhcps class add</code> <code>dhcps option add</code> <code>dhcps lease add</code> <code>dhcps config add</code>

Table 4-4 **DHCPv4 and DHCPv6 APIs and Shell Commands** (cont'd)

3.1 API	6.x Technique
dhcpsLeaseEntryDelete()	ipdhcps_subnet_delete() ipdhcps_class_delete() ipdhcps_pool_delete() ipdhcps_host_delete() ipdhcps_option_delete() dhcps subnet delete dhcps class delete dhcps pool delete dhcps host delete dhcps option delete dhcps lease delete dhcps config delete
dhcpsLeaseEntryGet()	dhcps subnet list dhcps pool list dhcps class list dhcps host list dhcps lease list dhcps option list dhcps config list
dhcpsLeaseHookAdd()	ipdhcps_lease_db_dump() ipdhcps_lease_db_restore()
—	ipdhcps_interface_status_set()
—	ipdhcps_lease_db_restore()
—	ipdhcpr_server_add() (routine) dhcpr server add (shell command) Add server address.
—	ipdhcpr_server_delete()
—	ipdhcpr_interface_status_set()
—	ipdhcpr_start_hook() Start DHCP relay agent.

Table 4-4 DHCPv4 and DHCPv6 APIs and Shell Commands (cont'd)

3.1 API	6.x Technique
—	ipdhcpr_server_delete() (routine) dhcpr server delete (shell command) Delete server address
—	ipdhcpr_interface_status_set() (routine) dhcpr interface enable (shell command) dhcpr interface disable (shell command) Set relay agent interface status.
—	dhcpr server list Shell command to list DHCP servers.
—	dhcpr interface list Shell command to list interfaces known by the DHCP relay agent.

4.3 DNS Components

This section describes DNS component and API mapping.

4.3.1 DNS Configuration

[Table 4-5](#) provides the DNS component mapping.

Table 4-5 DNS Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_DNS_RESOLVER	INCLUDE_IPDNSC
INCLUDE_DNS_RESOLVER_DEBUG	No equivalent

The DNS component configuration parameters have changed, but they generally map by functionality. [Table 4-6](#) shows the mapping of the old and new configuration parameters. New configuration parameters may not be exact matches for the old ones, and may have different default values. For details on each new configuration parameter, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 2*.

Table 4-6 DNS Configuration Parameter Migration

Wind River Network Stack 3.1 DNS Parameters	Wind River Network Stack 6.5 DNS Parameters	Wind River Network Stack 6.6 DNS Parameters
RESOLVER_DOMAIN	DNSC_DOMAIN_NAME	DNSC_DOMAIN_NAME
RESOLVER_DOMAIN_SERVER	DNSC_PRIMARY _NAME_SERVER	DNSC_PRIMARY _NAME_SERVER
No equivalent	DNSC_SECONDARY _NAME_SERVER	DNSC_SECONDARY _NAME_SERVER
No equivalent	DNSC_TERTIARYNS _NAME_SERVER	DNSC_TERTIARY _NAME_SERVER
No equivalent	DNSC_QUATERNARYNS _NAME_SERVER	DNSC_QUATERNARY _NAME_SERVER
RES_TIMEOUT_CFG	DNSC_TIMEOUT	DNSC_TIMEOUT
RETRY_CFG	DNSC_RETRIES	DNSC_RETRIES
RES_OPTIONS_CFG	No equivalent	No equivalent
NSCOUNT_CFG	No equivalent. You can specify up to four domain name servers using the parameters above.	No equivalent. You can specify up to four domain name servers using the parameters above.
DNS_DEBUG	No equivalent	No equivalent
No equivalent	DNSC_SERVER_PORT	DNSC_SERVER_PORT
No equivalent	DNSC_IP4_ZONE	DNSC_IP4_ZONE
No equivalent	DNSC_IP6_ZONE	DNSC_IP6_ZONE

4.3.2 API Mapping

Table 4-7 provides the API mapping for DNS.

Table 4-7 DNS APIs and Shell Commands

3.1 API	6.x Technique
resolvDNComp()	none (obsolete)
resolvDNExpand()	none (obsolete)
resolvGetHostByAddr()	ipdnsc_getipnodebyaddr() Use ipdnsc_freehostent() to free space allocated by ipdnsc_getipnodebyaddr() .
resolvGetHostByAnyAddr()	none (obsolete)
resolvGetHostByName()	ipdnsc_getipnodebyname() Use ipdnsc_freehostent() to free space allocated by ipdnsc_getipnodebyname() .
resolvInit()	none (obsolete)
resolvMkQuery()	none (obsolete)
resolvParamsGet()	none (obsolete)
resolvParamsSet()	none (obsolete)
resolvQuery()	none (obsolete)
resolvSend()	none (obsolete)
No equivalent	nslookup ipdnsc_freehostent() ipdnsc_cache_flush() New shell command and routines for viewing statistics and managing DNS.

4.4 FTP Components

This section describes FTP component and API mapping.

4.4.1 FTP Configuration

Table 4-8 provides the FTP component mapping. The FTP client library is still available in Wind River Network Stack 6.5 and 6.6 for backward compatibility. The network stack also includes a new FTP client utility.

Table 4-8 **FTP Component Migration**

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_FTP6_SERVER	INCLUDE_IPFTPS
INCLUDE_FTPD6_GUEST_LOGIN	No equivalent
INCLUDE_FTPD6_SECURITY	No equivalent
INCLUDE_FTP6	INCLUDE_FTP6 (for backward compatibility only)
INCLUDE_FTP	INCLUDE_FTP (for backward compatibility only)
No equivalent	INCLUDE_IPFTPC
No equivalent	INCLUDE_IPFTP_CMD

The hard-coded FTP access account with user name="ftp" and password="interpeak" that, by default, had read and write access to the FTP server in the 6.5 version of the Wind River Network Stack has been removed in the 6.6 version. The read-only account with user name="anonymous" (without a password) still exists by default.

4.4.2 API Mapping

Table 4-9 provides the API mapping for FTP. The FTP APIs from Wind River Network Stack 3.1 listed in the 3.1 API column are retained in Wind River

Network Stack 6.5 and 6.6 for internal backward compatibility. These APIs are deprecated and will be removed in a future release.

Table 4-9 FTP APIs and Shell Commands

3.1 API	6.x Technique
<code>ftp6Command()</code>	<code>ftp6Command()</code> (deprecated)
<code>ftp6DataConnGet()</code>	<code>ftp6DataConnGet()</code> (deprecated)
<code>ftp6DataConnInit()</code>	<code>ftp6DataConnInit()</code> (deprecated)
<code>ftp6FileGet()</code>	<code>ftp6FileGet()</code> (deprecated)
<code>ftp6FileSend()</code>	<code>ftp6FileSend()</code> (deprecated)
<code>ftp6Hookup()</code>	<code>ftp6Hookup()</code> (deprecated)
<code>ftp6LibInit()</code>	<code>ftp6LibInit()</code> (deprecated)
<code>ftp6Login()</code>	<code>ftp6Login()</code> (deprecated)
<code>ftp6Ls()</code>	<code>ftp6Ls()</code> (deprecated)
<code>ftp6RemoteModTime()</code>	<code>ftp6RemoteModTime()</code> (deprecated)
<code>ftp6ReplyGet()</code>	<code>ftp6ReplyGet()</code> (deprecated)
<code>ftp6Xfer()</code>	<code>ftp6Xfer()</code> (deprecated)
<code>ftpLibInit()</code>	<code>ftpLibInit()</code> (deprecated)
<code>ftpCommand()</code>	<code>ftpCommand()</code> (deprecated)
<code>ftpCommandEnhanced()</code>	<code>ftpCommandEnhanced()</code> (deprecated)
<code>ftpXfer()</code>	<code>ftpXfer()</code> (deprecated)
<code>ftpReplyGet()</code>	<code>ftpReplyGet()</code> (deprecated)
<code>ftpReplyGetEnhanced()</code>	<code>ftpReplyGetEnhanced()</code> (deprecated)
<code>ftpHookup()</code>	<code>ftpHookup()</code> (deprecated)
<code>ftpLogin()</code>	<code>ftpLogin()</code> (deprecated)
<code>ftpDataConnInitPassiveMode()</code>	<code>ftpDataConnInitPassiveMode()</code> (deprecated)

Table 4-9 **FTP APIs and Shell Commands** (cont'd)

3.1 API	6.x Technique
ftpDataConnInit()	ftpDataConnInit() (deprecated)
ftpDataConnGet()	ftpDataConnGet() (deprecated)
ftpLs()	ftpLs() (deprecated)
ftpLibDebugOptionsSet()	ftpLibDebugOptionsSet() (deprecated)
ftpTransientConfigSet()	ftpTransientConfigSet() (deprecated)
ftpTransientConfigGet()	ftpTransientConfigGet() (deprecated)
ftpTransientFatalInstall()	ftpTransientFatalInstall() (deprecated)
—	ipftpc_open()
—	ipftpc_close()
—	ipftpc_getattr()
—	ipftpc_setattr()
—	ipftpc_login()
—	ipftpc_cmd()
—	ipftpc_list()
—	ipftpc_get()
—	ipftpc_put()
—	ipftpc_lastreply()
—	ipftpc_strerror()
—	ftp shell command
ftpd6Delete()	ipd kill ipftps shell command (Only available as a shell command, not as C API.)
ftpd6DirListGet()	ipftpc_list()

Table 4-9 FTP APIs and Shell Commands (cont'd)

3.1 API	6.x Technique
ftpd6DisableSecurity()	none (obsolete) Passwords are enabled/disabled for each session.
ftpd6EnableSecurity()	none (obsolete) Passwords are enabled/disabled for each session through IPCOM authentication.
ftpd6Fatal()	none (obsolete)
ftpd6Init()	ipftps_create() (called at boot) ipd start ipftps (shell command)

4.5 Ping Components

This section describes ping component and API mapping.

4.5.1 Ping Configuration

Table 4-10 provides the ping component mapping.

Table 4-10 Ping Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_PING	INCLUDE_PING (wrapper) INCLUDE_IPPING_CMD
INCLUDE_PING6	INCLUDE_PING6 (wrapper) INCLUDE_IPPING6_CMD

4.5.2 API Mapping

Table 4-11 provides the API mapping for ping and ping6.

Table 4-11 Ping and Ping6 APIs and Shell Commands

3.1 API	6.x Technique
ping6LibInit()	Use the ping shell command.
pingLibInit()	Use the ping shell command.

4.6 SNTP Components

SNTP includes a client and a server implementation. This section describes SNTP component and API mapping.



NOTE: The SNTP server is not available in Wind River General Purpose Platform, VxWorks Edition.

4.6.1 SNTP Configuration

Table 4-12 provides the SNTP component mapping.

Table 4-12 SNTP Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_SNTPC	INCLUDE_IPSNTPC
INCLUDE_SNTPS	INCLUDE_IPSNTPS
No equivalent	INCLUDE_IPSNTP_COMMON
No equivalent	INCLUDE_IPSNTP_CMD

Enabling the Client or Server

Either the SNTP client or the SNTP server can be included in a build. The SNTP client (`INCLUDE_IPSNTPC`) is automatically included if the network stack is built with SNTP. The SNTP server is disabled by default.



NOTE: Although only the SNTP client component or the SNTP server component can be included in a build, the `sntpcTimeGet` wrapper, the `ipsntp_query_time()` and `ipsntp_wait_time()` routines, and the SNTP shell command are always available as long as one of the SNTP components is included.

To enable the SNTP server:

1. Open `installDir/components/ip_net2-6.x/ipsntp/config/ipsntp_config.h`.
2. Comment out the client define:

```
#define IPSNTP_USE_CLIENT
```

3. Uncomment the server define:

```
#define IPSNTP_USE_SERVER
```

4. Rebuild the project.

4.6.2 API Mapping

[Table 4-13](#) provides the SNTP API mapping.

Table 4-13 SNTP APIs and Shell Commands

3.1 API	6.x Technique
<code>sntpsClockSet()</code>	none (obsolete)
<code>sntpsConfigSet()</code>	none (obsolete)
<code>sntpcTimeGet()</code>	<code>sntpcTimeGet()</code> (wrapper) or <code>ipsntp_query_time()</code> / <code>ipsntp_wait_time()</code> . <code>sntpcTimeGet()</code> , <code>ipsntp_query_time()</code> , and <code>ipsntp_wait_time()</code> are available as long as one of the SNTP components is included in the build.

Table 4-13 **SNTP APIs and Shell Commands** (cont'd)

3.1 API	6.x Technique
<code>sntpsNsecToFraction()</code>	none (obsolete)
<code>mSntpcTimeGet()</code>	none (obsolete)
No equivalent	<code>ipsntp_set_reference()</code> —the only routine for the SNTP server. It sets the reference timestamp and reference identifier for the SNTP server.

4.7 Telnet Components

This section describes telnet component and API mapping.

4.7.1 Telnet Configuration

[Table 4-14](#) provides the telnet component mapping.

Table 4-14 **Telnet Component Migration**

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
<code>INCLUDE_TELNET</code>	<code>INCLUDE_IPTELNETS</code>
<code>INCLUDE_TELNET_CLIENT</code>	<code>INCLUDE_TELNET_CLIENT</code> (ported)

4.7.2 API Mapping

[Table 4-15](#) provides the telnet API mappings.

Table 4-15 Telnet APIs and Shell Commands

3.1 API	6.x Technique
telnet()	telnet() (no change)
telnetdInit()	none (obsolete)
telnetdParserSet()	none (obsolete)
telnetdStart()	none (obsolete)
telnetdStaticTaskInitializationGet()	none (obsolete)

4.8 TFTP Components

This section describes TFTP component and API mapping.

4.8.1 TFTP Configuration

[Table 4-16](#) provides the TFTP component mapping.

Table 4-16 TFTP Component Migration

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
No equivalent	INCLUDE_IPTFTP_COMMON
INCLUDE_TFTP_CLIENT	INCLUDE_IPTFTPC (also INCLUDE_TFTP_CLIENT for backward compatibility only)
No equivalent	INCLUDE_IPTFTP_CLIENT_CMD
INCLUDE_TFTP_SERVER	INCLUDE_IPTFTPS

4.8.2 API Mapping

Table 4-17 provides the TFTP API mapping. The TFTP APIs from Wind River Network Stack 3.1 listed in the 3.1 API column are retained in Wind River Network Stack 6.5 and 6.6 for internal backward compatibility. These APIs are deprecated and will be removed in a future release.

Table 4-17 TFTP APIs and Shell Commands

3.1 API	6.x Technique
<code>tftpGet()</code>	<code>tftpGet()</code> (deprecated)
<code>tftpPut()</code>	<code>tftpPut()</code> (deprecated)
<code>tftpInfoShow()</code>	<code>tftpInfoShow()</code> (deprecated)
<code>tftpInit()</code>	<code>tftpInit()</code> (deprecated)
<code>tftpModeSet()</code>	<code>tftpModeSet()</code> (deprecated)
<code>tftpPeerSet()</code>	<code>tftpPeerSet()</code> (deprecated)
<code>tftpdDirectoryAdd()</code>	none (obsolete)
<code>tftpdDirectoryRemove()</code>	none (obsolete)
<code>tftpdInit()</code>	You can stop, start, and reconfigure the tftp server with the <code>ipd</code> shell command.
<code>tftpCopy()</code>	<code>tftpCopy()</code> (deprecated)
<code>tftpPut()</code>	<code>tftpPut()</code> (deprecated)
<code>tftpQuit()</code>	<code>tftpQuit()</code> (deprecated)
<code>tftpSend()</code>	<code>tftpSend()</code> (deprecated)
<code>tftpXfer()</code>	<code>tftpXfer()</code> (deprecated)
<code>tftpCopy()</code>	<code>tftpCopy()</code> (deprecated)
no equivalent	<code>iptftp_client_put()</code> (new)
no equivalent	<code>iptftp_client_get()</code> (new)
no equivalent	<code>tftp</code> shell command (new)

4.9 Internet and Local Domain Sockets

This section describes component and API mapping for internet and local domain sockets.

4.9.1 Sockets Configuration

All of the required sockets components are automatically included when you build the network stack. For information on the sockets components and how to use them, see the internet and local domain sockets chapter in the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 2*. For information on specific socket ioctl argument types and behavior, see the reference pages.

4.9.2 API Mapping

The following socket APIs are carried forward with no changes:

socket	bind	listen	accept
connect	sendto	send	sendmsg
recvfrom	recv	recvmsg	setsockopt
getsockopt	getsockname	getpeername	shutdown

connectWithTimeout is carried forward, but is not supported for **AF_INET** and **AF_INET6** address families. Use **select()** to specify the timeout value for the socket descriptor.

4.9.3 Changes in Socket Options

The tables in this section identify changes in the socket ioctl options (socket options) for the **getsockopt()** and **setsockopt()** socket calls between Wind River Network Stack 3.1 and Wind River Network Stack 6.x.

In addition, there are separate tables for changes in the socket options available with the **getsockopt()** and **setsockopt()** socket calls.

Wind River Network Stack 3.1 IPv4 Socket Options

Table 4-18 lists the IPv4 socket options that are available in the 3.1 release and identifies whether the options are available in the current release. For new IPv4 socket options, see Table 4-20.

Table 4-18 IPv4 Socket Option Mapping

3.1 Socket Option	6.x Socket Option
SIOSIFADDR	SIOSIFADDR
SIOCGIFADDR	SIOCGIFADDR
SIOSIFDSTADDR	SIOSIFDSTADDR
SIOCGIFDSTADDR	SIOCGIFDSTADDR
SIOSIFFLAGS	SIOSIFFLAGS
SIOCGIFFLAGS	SIOCGIFFLAGS
SIOCGIFBRDADDR	SIOCGIFBRDADDR
SIOSIFBRDADDR	none (obsolete)
SIOSARP	SIOSARP
SIOCGARP	SIOCGARP
SIODARP	SIODARP
SIOCGIFCONF	SIOCGIFCONF
SIOCGIFNETMASK	SIOCGIFNETMASK
SIOSIFNETMASK	SIOSIFNETMASK
SIOCGIFMETRIC	SIOCGIFMETRIC
SIOSIFMETRIC	SIOSIFMETRIC
SIODIFADDR	SIODIFADDR
SIOCAIFADDR	SIOCAIFADDR
SIOCADDMULTI SIODELMULTI SIODELMULTI	Use the MCAST_ family of socket options to add or delete multicast addresses.

Table 4-18 IPv4 Socket Option Mapping (cont'd)

3.1 Socket Option	6.x Socket Option
SIOCSIFMTU	SIOCSIFMTU
SIOCGIFMTU	SIOCGIFMTU
SIOCSIFASYNCMAP	none (obsolete)
SIOCGIFASYNCMAP	none (obsolete)
SIOCSIFASYNCFLAGS	none (obsolete)
SIOCMUXPASSTHRU	SIOCMUXPASSTHRU
SIOCMUXL2PASSTHRU	SIOCMUXL2PASSTHRU
SIOCGMTU	SIOCGIFMTU
SIOCIFCREATE	SIOCIFCREATE
SIOCIFDESTROY	SIOCIFDESTROY
SIOCSIFINFO_FLAGS	none (obsolete)
SIOCAADDRCTL_POLICY	none (obsolete)
SIOCADDRCTL_POLICY	none (obsolete)

IPv6 Socket Options

[Table 4-19](#) lists the IPv6 socket options that were available in the 3.1 release and identifies whether the options are available in the current release. For new IPv6 socket options, see [Table 4-21](#).

Table 4-19 IPv6 Socket Option Mapping

3.1 Socket Option	6.x Socket Option
SIOCSIFADDR_IN6	Use SIOCGIFADDR
SIOCGIFADDR_IN6	Use SIOCSIFADDR
SIOCSIFDSTADDR_IN6	Use SIOCSIFDSTADDR
SIOCSIFNETMASK_IN6	Use SIOCSIFNETMASK

Table 4-19 IPv6 Socket Option Mapping (cont'd)

3.1 Socket Option	6.x Socket Option
SIOCGIFDSTADDR_IN6	SIOCGIFDSTADDR_IN6
SIOCGIFNETMASK_IN6	none (obsolete)
SIOCDIFADDR_IN6	SIOCDIFADDR_IN6
SIOCAIFADDR_IN6	SIOCAIFADDR_IN6
SIOCSIFPHYADDR_IN6	none (obsolete)
SIOCGIFPSRCADDR_IN6	none (obsolete)
SIOCGIFPDSTADDR_IN6	SIOCGIFDSTADDR_IN6
SIOCSIFPHYNEXTTHOP_IN6	none (obsolete)
SIOCGIFPHYNEXTTHOP_IN6	none (obsolete)
SIOCGIFAFLAG_IN6	Use SIOCGIFFLAGS
SIOCGDRLST_IN6	none (obsolete)
SIOCGPRLST_IN6	none (obsolete)
SIOCGIFINFO_IN6	none (obsolete)
SIOCSIFINFO_IN6	none (obsolete)
SIOCSNDFLUSH_IN6	none (obsolete)
SIOCGNBRINFO_IN6	none (obsolete)
SIOCSPFXFLUSH_IN6	none (obsolete)
SIOCSRTRFLUSH_IN6	none (obsolete)
SIOCGIFALIFETIME_IN6	none (obsolete)
SIOCSIFALIFETIME_IN6	none (obsolete)
SIOCGIFSTAT_IN6	none (obsolete)
SIOCGIFSTAT_ICMP6	none (obsolete)
SIOCSDEFIFACE_IN6	none (obsolete)

Table 4-19 IPv6 Socket Option Mapping (cont'd)

3.1 Socket Option	6.x Socket Option
SIOCGDEFIFACE_IN6	none (obsolete)
SIOCSIFINFO_FLAGS	none (obsolete)
SIOCSSCOPE6	none (obsolete)
SIOCGSCOPE6	none (obsolete)
SIOCGSCOPE6DEF	none (obsolete)
SIOCSIFPREFIX_IN6	none (obsolete)
SIOCGIFPREFIX_IN6	SIOCGIFPREFIX_IN6
SIOCdifPREFIX_IN6	none (obsolete)
SIOCAIFPREFIX_IN6	none (obsolete)
SIOCCIFPREFIX_IN6	SIOCGIFPREFIX_IN6
SIOCSGIFPREFIX_IN6	SIOCGIFPREFIX_IN6
SIOCGETSGCNT_IN6	none (obsolete)
SIOCGETMIFCNT_IN6	none (obsolete)

4.9.4 New Socket Options

The socket options in the following sections are included for your reference. No migration is required.

New IPv4 Socket Options

[Table 4-20](#) lists socket options for IPv4 that are new in the 6.5 and 6.6 releases. For IPv4 socket options carried forward from Wind River Network Stack 3.1, see [Table 4-18](#).

Table 4-20 New IPv4 Socket Options

New Socket Option	Comments
SIOCADDRT	Add route.
SIOCDELRT	Remove route.
SIOCXGETRT	Get route.
SIOCAHOMEADDR	Add a home address.
SIOCGIFINDEX	Get interface index.
SIOCGIFLLADDR	Get link level address.
SIOCXIFLLADDR	Set link level address.
SIOCXDETACH	Detach interface.
SIOCXGDHCPRUNNING	Get DHCP status.
SIOCXSDHCPRUNNING	Enable(1)/Disable(0) DHCP.
SIOCGIFPRIVATE	Get private interface data.
SIOCXIFPRIVATE	Set private interface data.
SIOCXSIFFEVENTCB	Set interface callback.
SIOCXDIFFEVENTCB	Delete interface callback.
SIOCXPROMISC	Activate/deactivate promiscuous mode.
SIOCXRESETSTAT	Reset interface statistics counter.
SIOCGETVIFCNT	Get VIF statistics.
SIOCADDVR	Add a new virtual router and creates a route table with table ID == 0.
SIOCDELVR	Delete a virtual router and all tables owned by it.
SIOCADDROUTETAB	Add a route table to a (virtual) router.
SIOCDELROUTETAB	Delete a route table to a (virtual) router.
SIOCGETROUTETAB	Get/create a route table by name.

Table 4-20 New IPv4 Socket Options (cont'd)

New Socket Option	Comments
SIOCSROUTETABNAME	Set a name for a route table.
SIOCGROUTETABNAME	Maps a route table name to VR and table ID.
SIOCGIFVR	Get the route table index for an interface.
SIOCSIFVR	Set an interface to a specific route table.
SIOCGETTUNNEL	Get tunnel parameter.
SIOCCHGTUNNEL	Change tunnel parameters.
SIOCGETSGCNT	Get mcast route statistics.
SIOCXIPSEC_CTL	IPSEC ioctl.
SIOCXIPSEC_SA_CTL	IPSEC ioctl.
SIOCXIPSEC_CONF_CTL	IPSEC ioctl.

New IPv6 Socket Options

[Table 4-21](#) lists socket options for IPv6 that are new in the current release. For IPv6 socket options carried forward from Wind River Network Stack 3.1, see [Table 4-19](#).

Table 4-21 New IPv6 Socket Options

New Socket Option	Comments
SIOCMIP6	
SIOCXGETGW_IN6	Get IPv6 gateway.
SIOCXSETGW_IN6	Set IPv6 gateway.
SIOCGETSGCNT_IN6	Get mcast6 route statistics.

Socket Options for Policy Routing

Socket options for policy routing are a new feature of the 6.5 release.

Table 4-22 **Socket Options for Policy Routing**

New Socket Option	Comments
SIOCGPRRULE	Get policy routing rule.
SIOCSPRRULE	Set rule for policy routing.
SIOC DPRRULE	Delete a rule for policy routing.
SIOCEPRRULE	Enumerate rules for policy routing.

New Socket Options for PPP

The following table lists socket options that are new for PPP. For information on migrating PPP applications, see [6. Wind River PPP](#).

Table 4-23 **Socket Options for PPP**

New Socket Option	Comments
SIOXPPPGDRVCONF	Get PPP driver config (default).
SIOXPPPGDRVINFO	Get PPP driver info (current).
SIOXPPPSDRVBAUDRATE	Set PPP driver baud rate.
SIOXPPPSDRVUP	Make driver signal up (test ioctl).
SIOXPPPSDRVDOWN	Make driver signal down (test ioctl).
SIOXPPPSDRVWINCOMPAT	Enable wincompat mode.
SIOXPPPGFLAGS	Get PPP link layer main flags.
SIOXPPPSFLAGS	Set PPP link layer main flags.
SIOXPPPGCONF	Get PPP main config.
SIOXPPPSCONF	Set PPP main config.
SIOXPPPGUSER	Get local user.
SIOXPPPSUSER	Set local user.
SIOXPPPGPASSWD	Get local password.

Table 4-23 **Socket Options for PPP** (cont'd)

New Socket Option	Comments
SIOXPPPPASSWD	Set local password.
SIOXPPPGPEERUSER	Get peer user.
SIOXPPPSETIF	Set pppoe Ethernet interface.
SIOXPPPGDINFO	Get PPP debug info.

Socket Options for Diffserv

Diffserv is a Quality of Service (QoS) feature of the current release. The following table lists socket options for using Diffserv.

Table 4-24 **Socket Options for Diffserv**

New Socket Option	Comments
SIOXADSFILTER	Add a filter for differentiated services meter/marker.
SIOXDDSFILTER	Delete a filter for differentiated services meter/marker.
SIOXDSCREATE	Create a new differentiated services meter/marker entity.
SIOXDSDestroy	Destroys a differentiated services meter/marker entity.
SIOXADSMAP	Add a filter for differentiated services meter/marker.
SIOXDDSMAP	Delete a filter for differentiated services meter/marker.
SIOCGIFQUEUE	Get the queue type for an interface.
SIOCSIFQUEUE	Set the queue type for an interface.
SIOXAIFQFILTER	Add a filter to an interface queue.
SIOXDIFQFILTER	Delete a filter from an interface queue.

4.10 RTP

VxWorks Real-time Process (RTP) projects allow you to manage and build modules that exist outside of the kernel space as a separate executable. The RTP implementation has not changed significantly. The only exception is that network applications—with the exception of ping—that were previously provided as example applications running in an RTP are not provided in Wind River Network Stack 6.6. For information on running custom applications in an RTP, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide, Volume 2*.

4.11 NFS Client and Server

The only change to NFS between the 3.1 and 6.x versions of the Wind River Network Stack is the location of the source files and the location of the components in Workbench.

In the Workbench Kernel Configuration Editor, NFS components that were previously located under **Network Components > Network Applications > NFS Components**, are now under **Operating system components > IO system components > NFS Components**.

NFS files have been moved to the file system tree, and NFS is no longer covered in the Wind River Network Stack programmer's guides. For information on using NFS, see the *VxWorks Kernel Programmer's Guide*.

The C files for NFS have been moved from *installDir/target/src/wrn/coreip/apps/nfs* to *installDir/vxworks-6.x/target/src/fs/nfs*.

The NFS .h files have been moved from *installDir/vxworks-6.x/target/h/wrn/coreip* to *installDir/vxworks-6.x/target/h/nfs*.

5

Wind River Network Stack: Interfaces and Drivers

- 5.1 Introduction 88
- 5.2 General Interface and Driver Configuration 89
- 5.3 Memory Management 92
- 5.4 MIB2 Statistics-Collection Support 94
- 5.5 Routing, Router Advertisement, and Router Solicitation 95
- 5.6 BPF 97
- 5.7 Interface Components 97
- 5.8 IP Attach Components 100
- 5.9 MUX-L2 101
- 5.10 Auto IP 101
- 5.11 zBuf and Fast UDP Sockets 101
- 5.12 Unnumbered Interfaces 102

5.1 Introduction

This chapter covers interfaces and drivers and will help you to plan the following migration paths:

- from Wind River Network Stack 3.1 to Wind River Network Stack 6.5 or 6.6
- from Wind River Network Stack 6.5 to Wind River Network Stack 6.6.

There are four chapters in this guide that contain migration information for the network stack:

2. Wind River Network Stack Migration Overview

introductory information on network stack migration, key concepts and migration information common to most or all networking components

3. Wind River Network Stack: Transport and Network Protocols

migration details and component and API mapping for the components of the core network stack, including TCP/IP, multicast, and routing

4. Wind River Network Stack: Application Protocols

migration details and component and API mapping for the network application components, including DHCP and DNS, and information on programming with sockets

5. Wind River Network Stack: Interfaces and Drivers (this chapter)

migration details and information on changes to libraries and routines for lower-level network stack components, including the MUX and interface configuration

5.1.1 Feature Release Matrix

[Table 5-1](#) provides a list of Wind River Network Stack features and indicates in which releases they are available. Features that are not supported in Wind River Network Stack 6.6 may be available in a future release.

Table 5-1 Network Stack Interfaces and Drivers – Feature Release Matrix

Feature	3.1	6.5	6.6
Automatic IP configuration	●	● (New implementation)	●
netBufLib ^a	●	●	●
Router advertisement and solicitation	●	● (New implementation)	●
Zero-copy functionality (including zbuf)	●	Not supported	Not supported
M2IfLib (including MIB2-based statistics collection)	●	Not supported	Not supported
BPF devices	●	Not supported	Not supported

a. The network stack now uses **netBufLib** only in its communication with network devices. For details, see [Use of netBufLib](#), p.93.

5.2 General Interface and Driver Configuration

This section describes general interface and driver configuration changes, including component mappings and API changes.

5.2.1 Configuration Components

[Table 5-2](#) shows some of the new, changed, and obsolete components used in device and interface programming. This list is not exhaustive; see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 3* for more information.

Table 5-2 Migration of Wind River Network Stack Configuration Components

Wind River Network Stack 3.1 Components	Wind River Network Stack 6.x Components
INCLUDE_AIP	INCLUDE_IPAIP
No equivalent	SELECT_IPAIP_CONFIG
No equivalent	INCLUDE_IPAIP_GLOBAL_CONFIGS
No equivalent	INCLUDE_IPAIP_INTERFACE_CONFIGS
INCLUDE_PREFIX	No equivalent
INCLUDE_ADDIF	INCLUDE_IPNET_IFCONFIG_x
No equivalent	INCLUDE_NULLBUFPOOL
INCLUDE_LOOPBACK	INCLUDE_IPNET_USE_LOOPBACK
No equivalent	INCLUDE_USE_WLAN

5.2.2 Shell Commands

In this release, you use shell commands to configure and view statistics about the network stack and components. Some new shell commands replace APIs from the 3.1 release. See the API mapping tables in each section for API changes. For more information, see [Shell Commands](#), p.11.

Of particular interest affecting interfaces and drivers are the following:

- **ifconfig** replaces various routines and commands, such as the previous **ifconfig** and **ifShow**. **ifconfig** is provided both as a command-interpretter command and as a wrapper routine that you can call from the C interpreter. In both interpreters, the syntax for **ifconfig** has changed somewhat from that of the **ifconfig()** routine provided in 3.1-era releases.
- You now use the **sysvar** and **radvd** shell commands to set up routing advertisement and solicitation, in place of routines like **rtadv**.

For more information on using individual commands for interfaces and drivers, see the relevant sections of the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 3*.

5.2.3 Unchanged Libraries and APIs

The following libraries and APIs are carried forward without substantial change in Wind River Network Stack 6.x. For details about each library or API and how to use it, see the reference entry for the library or API.

applUtilLib	muxTkLib	daemon (network task support)
netBufAdvLib	etherMultiLib	netBufLib
jobQueueLib	netBufPool	jobQueueUtilLib
panicLib	linkBufPool	hostSetup()
muxL2Lib	netPoolShow()	muxLib

5

5.2.4 Obsolete APIs

The **tvToTicks** API is now obsolete.

5.2.5 Checksum Offloading

The driver interface for hardware checksum offloading has not changed for this release. That is, no changes need to be made to your driver code to support hardware checksum offloading if the driver was implemented for an earlier VxWorks 6.x release. However, checksum offloading support is no longer included in the network stack by default. In order to enable checksum offloading, you must include the **IPCOM_USE_HW_CHECKSUM** component when building your Wind River Network Stack source code. This support can be included by uncommenting the definition for **IPCOM_USE_HW_CHECKSUM** in the following file: *installDir/components/ip_net2-6.5/ipcom/port/vxworks/config/ipcom_pconfig.h*.

For more information on building network stack source code, see the getting started guide for your Platform. For more information on hardware checksum offloading, see the *Integrating a New Network Interface Driver* chapter of *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 3*.

5.3 Memory Management

This section describes the memory management changes in 6.x releases.

5.3.1 Changes

The 6.5 release introduced several changes to the way the network stack implements memory allocation.

One of the most important changes is that only the Ethernet and other link-layer drivers use **netBufLib** (as noted in [Use of netBufLib](#), p.93).

Another change is that the network stack now creates a single pool of data buffers. The network stack does not use **netBufLib** as it did in the 3.1 release. If your custom application or protocol relied upon the existence of this data pool, note that **netBufLib** is no longer used to create that pool.

Drivers and Leading Spaces in Cluster Headers

The 6.5 release modified the implementation of **netBufLib** to automatically provide extra memory space (also called *headroom* or *leading pad space*) at the beginnings of clusters in certain network pools. The pools affected are those that are created with a single size of clusters, when that size is at least 1500 bytes; this heuristic is intended to select network pools used as receive pools for network interface devices. When you create such a pool by calling **netPoolCreate()**, the sizes of all its clusters are increased by the leading pad space (which defaults to 64 bytes).

If you instead create the pool by calling **netPoolInit()** and providing a buffer of memory for the clusters; the **netPoolInit()** code decreases the total number of clusters while increasing the cluster size by the headroom, keeping the total memory usage within the buffer size specified in the **netPoolInit()** call. (For most network device receive pools, decreasing the total number of clusters by a small number does not have negative effects.)

Regardless of how you created the pool, when you allocate a tuple from the pool by calling **netTupleGet()**, the tuple's **M_BLK mBlkHdr.mData** pointer points not at the beginning of the cluster, but past the leading pad space (by default, 64 bytes into the cluster).

The reason for this change is that in some circumstances (such as when packets are tunneled or encrypted), additional network headers must be prefixed to a packet

received off the wire. If there is insufficient space at the start of the cluster for the additional network header, the stack (which now requires contiguous packet data) must copy the whole packet to prepend the header, and such copying decreases performance. Therefore, the extra leading pad space is provided to improve performance by allowing prepending of the additional headers without requiring a full packet copy. This change has been made implicit so that it benefits most network device drivers without requiring driver modifications.

Network drivers whose receive pools are subject to the implicit leading pad space addition will automatically benefit from the change if they allocate tuples for packets to be received by calling **netTupleGet()**, and cause the device to DMA the data into the tuple cluster starting at the location pointed to by **mBlkHdr.mData**.

Some older network drivers do not use **netTupleGet()**, but instead construct tuples themselves after calling **netMblkGet()**, **netClusterGet()**, and **netCIBlkGet()**, and use the **netCIBlkJoin()** and **netMblkCJoin()** routines to join the parts to form the tuple. These drivers will probably continue to function, but will not take advantage of the leading pad space unless you modify them to do so (by causing the device to DMA data after the leading pad space rather than at the start of the cluster). Such modifications are usually fairly easy.

Some unusual network device drivers that either are very sensitive to the number of clusters returned by a **netPoolInit()** call, or that assume that the **mBlkHdr.mData** pointer of the **M_BLK** of a tuple obtained from **netTupleGet()** must point at the start of the cluster, may be adversely affected by this change. In this case, you must either modify the affected device driver, or else set the default leading cluster pad space to zero by setting the **NETBUF_LEADING_CLSPACE_DRV** parameter of the **INCLUDE_NETBUFLIB** component to zero.

Future releases may remove the implicit addition of leading cluster pad space, and require such space to be explicitly requested on a per-pool basis.

Use of netBufLib

The network stack no longer uses **netBufLib** except in its communication with network devices. Because the drivers continue to use the pre-existing **netBufLib** APIs and cluster pool mechanism for their buffers, you can use most 3.1-era (VxWorks 6.4 and earlier) drivers without modification in Wind River Network Stack 6.6. However, you must recompile the drivers. If you have binary-only versions of a driver, you must obtain the source or a binary recompiled under Wind River Network Stack 6.6.

In addition, note the following caveats:

- The network stack requires that network device drivers describe those packets they deliver to the stack as a single **M_BLK/CL_BLK**/cluster tuple. 3.1-era versions of the stack required that the IP header be 4-byte aligned in memory, but the current stack only requires 2-byte alignment. It is still better to use 4-byte alignment, because this makes header processing faster.
- The network stack only delivers packets for transmission described by a single **M_BLK/CL_BLK**/cluster tuple.
- The network stack itself no longer uses the two **netBufLib**-style “system” and “data” pools for its own memory allocation.

These items are detailed in the *Configuring and Managing Memory* chapter of the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 3*.

5.4 MIB2 Statistics-Collection Support

This release does not support **M2IfLib**. Therefore, you cannot collect statistics by stack polling of drivers (using MIB-II). If you want to include statistics collection, the driver must pass packet statistics to the stack on a per-packet basis.

5.4.1 M2IfLib

This release does not support the **M2IfLib** library (for example, routines such as **M2IfInit()**).

5.4.2 SNMP MIB-II Support

This release does not support SNMP MIB-II APIs (for example, **M2Init** and **M2Delete**).

New SNMP stub routines are implemented for the following SNMP MIBs:

- RFC 2011 – IP-MIB
- RFC 4292 – IP-FORWARD
- RFC 2465 – IPV6-MIB

- RFC 2466 – IPV6-ICMP-MIB
- RFC 2863 – IF-MIB
- RFC 4022 – TCP-MIB
- RFC 4113 – UDP-MIB

The source files for the routines can be found in the following directory:

`installDir/components/ip_net2-6.5/wrsnmp/src`

5.5 Routing, Router Advertisement, and Router Solicitation

Router advertisement and solicitation is no longer handled by `rtadv()` and associated APIs like `rtsolStart()`. Advertisement and solicitation can be managed through the `radvd` and `sysvar` shell commands.

For more information, see the *Working with Drivers and Interfaces* chapter in *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 3*.

For information on routing sockets, see [3.13 Routing Sockets](#), p.51.

5.5.1 API Mapping

The tables in this section list routing-related APIs.

Routing

[Table 5-3](#) lists deprecated routing APIs.

Table 5-3 **Routing APIs**

3.1 API	6.x Technique
<code>Sysctl</code>	<code>Sysctl</code> (no change)
<code>kernSysctlInit</code>	none (obsolete)
<code>rarpDebugSet</code>	none (obsolete)
<code>rdCtl (rdiscLib)</code>	none (obsolete)

Table 5-3 **Routing APIs** (cont'd)

3.1 API	6.x Technique
rdiscIfReset	none (obsolete)
rdiscIfShow	none (obsolete)
rdiscStart	none (obsolete)
replyNopSlot	none (obsolete)

Router Advertisement

Router configuration using the **advCap** library (**rtadvConfSet**, **rtadvConfClr**, **rtadvConfRemove**, **rtadvConfShow**) is deprecated. Use the **radvd** shell command instead. The router advertisement daemon is initialized at start-up.

[Table 5-4](#) reflects changes to the router advertisement APIs.

Table 5-4 **Router Advertisement APIs**

Deprecated 3.1 API	6.x Technique
rtadv	radvd shell command (ipnet_cmd_radvd())
advCap library	none (obsolete)
rtadvConfClr	none (obsolete)
rtadvConfRemove	none (obsolete)
rtadvConfSet	none (obsolete)
rtadvConfSet	none (obsolete)
rtadvConfig	radvd shell command
prefixcmd	radvd add shell command

Router Solicitation

Wind River Network Stack 6.6 supports both host- and router-side router solicitation, for both IPv4 and IPv6. You can enable automatic router solicitation by

setting the `IPNET_RFC1256_ENABLE_SOLICITATION` build component. The following router solicitation APIs are deprecated:

- `rtsolStart()`
- `rtsolStop()`

5.6 BPF

Berkeley Packet Filter (BPF) functionality is not supported in this release. The new DHCP libraries do not require this component. The following BPF APIs are not included in Wind River Network Stack 6.6:

- `bpfDevCreate()`
- `pfDevDelete()`
- `bpfDrv()`

5.7 Interface Components

[Table 5-5](#) shows APIs related to interface-specific parameters management. Most of the GET/SET operations can also be done using the **ifconfig** (wrapper) command.

Table 5-5 Interface Configuration APIs

3.1 API	6.x Technique	Comments
<code>ifconfig</code>	<code>ifconfig</code> or <code>ipnet_cmd_ifconfig</code>	Functional changes (see the <i>Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 3</i>).
<code>ifShowInit()</code>	none (obsolete)	
<code>ifShow()</code>	<code>ifconfig</code>	

Table 5-6 Interface Library APIs

3.1 API	6.x Technique	Comments
freeaddrinfo()	freeaddrinfo() (wrapper)	
freehostent()	freehostent()	
gai_strerror()	none (obsolete)	
gifLoad()	none (obsolete)	
if6AddrAdd()	SIOCAIFADDR_IN6 ioctl command	
if6AddrDelete()	SIOCDIFADDR_IN6 ioctl command	
if6AddrGet()	SIOCAIFADDR_IN6 ioctl command	
if6DstAddrGet()	SIOCGIFDSTADDR_IN6 ioctl command	
if6DstAddrSet()	none (obsolete)	You can set the destination address as part of the IP_SIOCAIFADDR_IN6 ioctl request.
if6FlagChange()	SIOCGIFFLAGS and SIOCSIFFLAGS ioctl commands	Both flags are required.
if6FlagGet()	SIOCGIFFLAGS ioctl command	
if6FlagSet()	SIOCSIFFLAGS ioctl command	
if6LifetimeGet()	SIOCAIFADDR_IN6 command	
if6LifetimeSet()	SIOCXGIFADDR_IN6 command	You can set the preferred and valid lifetimes as part of the IP_SIOCAIFADDR_IN6 ioctl request.
if6PrefixlenGet()	SIOCGIFPREFIX_IN6 ioctl command	
if6PrefixlenSet()	none (obsolete)	You can set the prefix length as part of the IP_SIOCAIFADDR_IN6 ioctl request.
ifAddrAdd()	SIOCAIFADDR ioctl command	
ifAddrDelete()	SIOCDIFADDR ioctl command	

Table 5-6 Interface Library APIs (cont'd)

3.1 API	6.x Technique	Comments
<code>ifAddrGet()</code>	<code>SIOCGIFADDR ioctl</code> command	
<code>ifAddrSet()</code>	<code>SIOCSIFADDR ioctl</code> command	
<code>ifAllRoutesDelete()</code>	none (obsolete)	
<code>ifBroadcastGet()</code>	<code>SIOCGIFBRDADDR ioctl</code> command	Obsolete
<code>ifBroadcastSet()</code>	<code>SIOCGIFBRDADDR ioctl</code> command	The broadcast address is a function of the IP address and the netmask, and you cannot set it separately.
<code>ifDstAddrGet()</code>	<code>SIOCGIFDSTADDR ioctl</code> command	
<code>ifDstAddrSet()</code>	<code>SIOCSIFDSTADDR ioctl</code> command	
<code>ifFlagChange()</code>	<code>SIOCSIFFLAGS ioctl</code> command	
<code>ifFlagGet()</code>	<code>SIOCGIFFLAGS ioctl</code> command	
<code>ifFlagSet()</code>	<code>SIOCSIFFLAGS ioctl</code> command	
<code>ifIndexToIfName()</code>	<code>if_indextoname</code>	
<code>ifMaskGet()</code>	<code>SIOCGIFNETMASK ioctl</code> command	
<code>ifMaskSet()</code>	<code>SIOCSIFNETMASK ioctl</code> command	
<code>ifMetricGet()</code>	<code>SIOCGIFMETRIC ioctl</code> command	
<code>ifMetricSet()</code>	<code>SIOCSIFMETRIC ioctl</code> command	
<code>ifNameToIfIndex()</code>	<code>if_nametoindex</code>	
<code>ifProxyArpDisable()</code> <code>ifProxyArpEnable()</code>	To enable ARP for a single host, set the <code>ATF_PUBL</code> flag on the ARP entry using <code>SIOCSARP</code> . To enable ARP for the network stack, build the stack with the <code>INCLUDE_IPPROXYARP</code> configuration component.	
<code>ifUnnumberedSet()</code>	<code>ifconfig</code> command	obsolete (unnumbered interfaces are supported, only the API is obsolete, see 5.12 Unnumbered Interfaces , p.102)

5.8 IP Attach Components

The two IP attach components, `INCLUDE_IPATTACH` and `INCLUDE_IP6ATTACH` are now wrappers to a new implementation. If you include these two components, this automatically pulls in all necessary components for the wrapper support. For details, see the *Wind River Network Stack for VxWorks 6 Programmer's Guide 6.6, Volume 3*.

5.8.1 API Mapping

[Table 5-7](#) provides the API mapping for the IP attach protocols.

Table 5-7 **ipProto APIs**

3.1 API	6.x Technique
<code>ip6Attach()</code>	<code>ip6Attach()</code> (wrapper)
<code>ip6Detach()</code>	<code>ip6Detach()</code> (wrapper)
<code>ipAttach()</code>	<code>ipAttach()</code> (wrapper)
<code>ipDetach()</code>	<code>ipDetach()</code> (wrapper)

To attach an END or NPT interface to the network stack, you previously used `ipAttach()` and `ifAddrSet()` as in the following example:

```
-> ipAttach 0, "fei"
-> ifAddrSet "fei0", "128.224.195.196"
```

The `ifAddrSet()` API routine is no longer supported; use `ifconfig()` instead. Note that to use these commands, you must include the `INCLUDE_IFCONFIG` and `INCLUDE_IPWRAP_IPPROTO` components in your build. An example follows:

```
-> ipAttach 0, "fei"
-> ifconfig "fei0 inet 50.50.1.127/24"
-> ifconfig "fei0 up"
```

When you call `ipAttach()` this attaches IPv4 to the device. If you build the network stack for IPv6 support, when you call `ipAttach()` this also attaches IPv6 to the device. Thus, `ipAttach()` and `ip6Attach()` now behave identically.

5.9 MUX-L2

The MUX layer 2 library is unchanged in this release. For a list of unchanged libraries and APIs relating to interfaces and drivers, see [5.2.3 Unchanged Libraries and APIs](#), p.91.

5

5.10 Auto IP

[Table 5-8](#) lists APIs associated with Auto IP.

Table 5-8 **Auto IP APIs**

Deprecated 3.1 API	6.x Technique
autoIP	The address is configured automatically when the network interface is brought up.
aipStop	The address is not removed when the network interface is brought down, but the address is revalidated if the interface is brought back up.

5.11 zBuf and Fast UDP Sockets

The **zbufSockLib** zero-copy sockets library, and the associated **zbufLib** buffer management library, are not supported in this release. Fast UDP sockets are also not supported.

5.12 Unnumbered Interfaces

The Wind River Network Stack 3.1 releases used the **ifUnnumberedSet()** routine to create unnumbered interfaces for point-to-point type interfaces. In Wind River Network Stack 6.x, any interface can be turned into a unnumbered interface. While most stacks restrict unnumbered interfaces to point-to-point interface types, the Wind River Network Stack allows you to turn any type of interface into an unnumbered interface simply by assigning it the same IP address as another interface (for example, the main interface).

Wind River Network Stack 6.x no longer uses **ifUnnumberedSet()** to do this assignment; it uses **ifconfig()** as described below.

Suppose Ethernet interface **gei0** is assigned the address 1.2.3.4. You can turn interface **ppp0** into an unnumbered interface by assigning the same address to it as follows:

```
# ifconfig ppp0 inet 1.2.3.4
```

Note that this is exactly the same command you use to assign an address to an interface; the only difference is that, in this case, it is the same as the Ethernet interface, which thus turns **ppp0** into an unnumbered interface.

6

Wind River PPP

6.1 Introduction	103
6.2 Migration Steps	105
6.3 Configuration	106
6.4 Library and Routine Changes	114

6.1 Introduction

Major changes to Wind River PPP were introduced in the Wind River PPP 6.5 release. This chapter will help you to plan your migration from Wind River PPP 2.x to the current release, Wind River PPP 6.6. Releases of Wind River PPP numbered 6.5 and later are not backward compatible with earlier releases of Wind River PPP. [6.3 Configuration](#), p.106, describes equivalent functionality, where available.

Changes between Wind River PPP 6.5 and Wind River PPP 6.6 are highlighted in [6.1.1 Feature Release Matrix](#), p.104 and [6.1.2 Changes in Wind River PPP 6.6](#), p.105.

Wind River PPP 6.x does not support PPPoE over a virtual local area network (VLAN.)

For information on how to set Wind River PPP configuration parameters, see the *Wind River PPP for VxWorks 6 Programmer's Guide*, 6.6: *Configuring Wind River PPP*.

6.1.1 Feature Release Matrix

Table 6-1 provides a list of Wind River PPP features and indicates in which releases they are available.

Table 6-1 Wind River PPP Feature Release Matrix

Feature	2.x	6.5	6.6
Remote Access Framework	●	Obsolete	Obsolete
Multi-link PPP	●	Not supported	Not supported
SNMP MIBs	●	Not supported	Not supported
Van Jacobson header compression	●	Not supported	Not supported
Multiple network type support	●	● (New implementation)	●
Multiple driver type support	●	● (New implementation)	●
Multiple PPP framing support	●	● (New implementation)	●
Unlimited PPP connections	●	● (New implementation)	●
Dynamic stack configuration	●	● (New implementation)	●
Configuration profiles	●	● (New implementation)	●
Table management	●	● (New implementation)	●
Timers	●	● (New implementation)	●
Memory management	●	● (New implementation)	●
PPP Client	●	● (New implementation)	●

Table 6-1 Wind River PPP Feature Release Matrix (cont'd)

Feature	2.x	6.5	6.6
PPP Server	●	● (New implementation)	●
PPPoE Server	●	● (New implementation)	●
PPPoE Client	●	● (New implementation)	●
PPP shell command	Not supported	●	●

6.1.2 Changes in Wind River PPP 6.6

There are no functional changes in this release, which moves Wind River PPP to a new release of VxWorks.

6.1.3 Additional Documentation

For a description of the current implementation of Wind River PPP, see the *Wind River PPP for VxWorks 6 Programmer’s Guide*, 6.6.

6.2 Migration Steps

Wind River PPP 2.x has been replaced by Wind River PPP 6.x. All Workbench components and API routines have been replaced, and a shell command has been added. These changes are due to a new design and implementation of the product. For more information, see [6.3 Configuration](#), p.106, and [6.4 Library and Routine Changes](#), p.114.

For a description of the new implementation, see the *Wind River PPP for VxWorks 6 Programmer’s Guide*, 6.6.

6.3 Configuration

This section describes the changes to components, parameters, files, and configuration between Wind River PPP 2.x and Wind River PPP 6.x.

6.3.1 Configuring VxWorks Image Projects for Wind River PPP

In earlier releases of Wind River PPP, you created PPP as a Remote Access Framework, included the framework in your VxWorks image using the `pfwCreate()` routine, and configured all parameters as plug-in objects to the framework. Wind River PPP is no longer assembled as a framework; you include the PPP components in your image directly.

The Wind River PPP 6.x components are listed in [Table 6-2](#). To include Wind River PPP 6.5 in VxWorks using Workbench, include the `INCLUDE_IPPPP` and `INCLUDE_IPPPPoE` components. To enable the `pppconfig` shell command, include the `INCLUDE_IPPPP_CMD` component. To enable per-interface or per-user configuration, include the `INCLUDE_IPPPP_INTERFACE_CONFIG` and `INCLUDE_IPPPP_USERS_CONFIG` components. Workbench determines all PPP dependencies for functionality that is built into the PPP libraries. Rebuild the VxWorks image in Workbench as usual.

Table 6-2 Wind River PPP Component Migration

Wind River PPP 2.x Components	Replaced by Wind River PPP 6.x Components
All components are replaced.	<code>INCLUDE_IPPPP</code> <code>INCLUDE_IPPPP_CMD</code> <code>INCLUDE_IPPPPoE</code> <code>INCLUDE_IPPPP_INTERFACE_CONFIG</code> <code>INCLUDE_IPPPP_USERS_CONFIG</code>

Disabling PPPoE at Compile Time

The `INCLUDE_IPPPoE` component is enabled by default when the `IPPPP` library is built. To disable PPPoE at compile time, rebuild the binaries with `IPPPP_USE_PPPOE` undefined in `ipppp_config.h`.

6.3.2 Initialization

The Wind River PPP 2.x initialization routine **windNetPPPInit()** is no longer used or required by Wind River PPP 6.x. Initialization and startup is automatically regulated by a central process, the IPNET daemon (IPD). For more information, see [Product or Component Initialization](#), p.9.

6.3.3 Shell Commands

Wind River PPP 6.5 has a new shell command: **pppconfig**. For a description of this command see the *Wind River PPP for VxWorks 6 Programmer's Guide: Using the pppconfig Command*. For more information on shell commands in this release, see [Shell Commands](#), p.11.

6.3.4 Configuration Routines

All Wind River PPP 2.x routines have been deprecated. [Table 6-3](#) describes equivalent functionality in Wind River PPP 6.x.

Table 6-3 **Comparison of Routines**

Description	Wind River PPP 2.x	Wind River PPP 6.x
All Remote Access Framework routines	Any routine beginning with the prefix pfw	Obsolete. The Remote Access Framework has been obsoleted for Wind River PPP 6.x.
All Multilink PPP routines	Any routine beginning with the prefix mp apiMpLinkAssign() isNoEidNoAuthCase()	Obsolete. Wind River PPP 6.x does not support Multilink PPP.
All Port Manager routines	Any routine beginning with the prefix pm and portManagerSerial	Obsolete. The port manager interface is only used for Multilink PPP, which is not supported in Wind River PPP 6.x.
All PPP MIB II routines	Any routine beginning with the prefix m2ppp	Obsolete. Wind River PPP 6.x does not support SNMP MIBs.

Table 6-3 **Comparison of Routines** (cont'd)

Description	Wind River PPP 2.x	Wind River PPP 6.x
Van Jacobson header compression	<code>pppVjc()</code> <code>pppVjcComponentCreate()</code> <code>pppVjcComponentDelete()</code>	Obsolete. Wind River PPP 6.x does not support Van Jacobson header compression.
BACP configuration	Any routine beginning with the prefix bacp or bap <code>pppBacpComponent()</code> <code>pppbaptx()</code>	Obsolete. Wind River PPP 6.x does not support RFC 2515 or the Bandwidth Allocation Protocol.
LCP configuration	<code>lcpComponentCreate()</code> <code>lcpComponentDelete()</code>	<p>The LCP configuration options are defined using either configuration parameters or the sysvar variable.</p> <p>For example, to set the LCP maximum transmission unit (MTU) option in the Workbench GUI, use the PPP_LCP_MTU parameter.</p> <p>To set the MTU at the command line, use the following command:</p> <p>sysvar set -o ipppp.lcp.mtu <i>value</i></p> <p>For more information, see the <i>Wind River PPP for VxWorks 6 Programmer's Guide</i>, 6.6: <i>Configuration Options</i>.</p>

Table 6-3 Comparison of Routines (cont'd)

Description	Wind River PPP 2.x	Wind River PPP 6.x
PAP/CHAP authentication	<code>pppChapComponentCreate()</code> <code>pppChapComponentDelete()</code> <code>pppPapComponentCreate()</code> <code>pppPapComponentDelete()</code>	<p>Authorization is defined at build time by enabling the <code>IPCOM_USE_AUTH</code> component in the file <code>ipcom_config.h</code>, as well as the <code>IPPPP_USE_AUTH_CHAP</code> or <code>IPPPP_USE_AUTH_PAP</code> components in the file <code>ipppp_config.h</code>.</p> <p>Note that you should only define these components if you are going to use them.</p> <p>At run time, authentication is configured using configuration parameters or sysvar variable.</p> <p>For authentication, in the Workbench GUI, use the <code>PPP_AUTH_MODES</code> configuration parameter.</p> <p>At the command line, use the following command:</p> <p>sysvar set -o ipppp.auth mode</p> <p>See the <i>Wind River PPP for VxWorks 6 Programmer's Guide</i>, 6.6: <i>Configuration Options</i>.</p>

Table 6-3 Comparison of Routines (cont'd)

Description	Wind River PPP 2.x	Wind River PPP 6.x
IPCP configuration	<p>pppIpcpComponentCreate()</p> <p>pppIpcpComponentDelete()</p>	<p>To set the desired local IPv4 address in the Workbench GUI, use the PPP_IPCP_IPV4_ADDRESS parameter.</p> <p>At the command line, use the following command:</p> <p>sysvar set -o ipppp.ipcp.addr address</p> <p>See the <i>Wind River PPP for VxWorks 6 Programmer's Guide</i>, 6.6: <i>Configuration Options</i>.</p>
IPv6 Configuration	<p>pppIpv6cpComponentCreate()</p> <p>pppIpv6cpComponentDelete()</p>	<p>The IPv6CP is called when a peer sends an IPv6 Identifier Configure-Request. For details, see RFC 2742 (pp. 6-8.) Wind River PPP uses an action callback routine of action type IPPPP_ACTION_PEER_IDENTIFIER for this. For an example, see ipppp_example.c.</p> <p>See the <i>Wind River PPP for VxWorks 6 Programmer's Guide</i>, 6.6: <i>Configuration Options</i>.</p>
Closing a PPP interface	pppConnectionClose()	<p>Use the following shell command:</p> <p>pppconfig interface_name shutdown</p>
Show PPP or PPPoE configuration	<p>pppIpv4InfoGet</p> <p>pppIpv6InfoGet</p> <p>pppOEServiceListShow()</p> <p>pppOESessionListShow()</p>	<p>Use the following command:</p> <p>pppconfig show</p>

Table 6-3 Comparison of Routines (cont'd)

Description	Wind River PPP 2.x	Wind River PPP 6.x
Secret name	<code>pppLocalSecretAdd()</code> <code>pppPeerSecretAdd()</code> <code>pppPeerSecretDelete()</code>	<p>In the Workbench GUI, use the configuration parameter <code>PPPOE_SECRET_NAME</code>.</p> <p>At the command line, use the following command:</p> <pre>sysvar set -o ipppp.pppoe. secret_name <i>name</i></pre> <p>The default is “puttoanythinghere”.</p>
PPP driver	<code>pppModem()</code> <code>pppSioAdapter()</code> <code>sioAdapterCreate()</code> <code>sioAdapterDelete()</code>	<p>There is a PPP driver (<code>ipcom_drv_ppp.c</code>) that uses a serial port (rs232). To bring a driver down or up, use <code>ioctl()</code> system calls.</p> <p>All configuration with the driver is also done using <code>ioctl()</code> system calls.</p> <p>For the PPP driver info see the <i>Wind River PPP for VxWorks 6 Programmer’s Guide</i>, 6.6: <i>Configuration Options</i>.</p>

Table 6-3 Comparison of Routines (cont'd)

Description	Wind River PPP 2.x	Wind River PPP 6.x
Framing layer	<p><code>pppAsyncFraming()</code></p> <p><code>pppAsyncFramingComponentCreate()</code></p> <p><code>pppAsyncFramingComponentDelete()</code></p> <p><code>pppBitSyncFraming()</code></p> <p><code>pppBitSyncFramingComponentCreate()</code></p> <p><code>pppBitSyncFramingComponentDelete()</code></p> <p><code>pppFramingLayer()</code></p> <p><code>pppFramingLayerCreate()</code></p> <p><code>pppFramingLayerDelete()</code></p>	<p>Obsolete. Wind River PPP 6.x does not support the PPP Framing Layer.</p>
Control layer	<p><code>pppControlLayer()</code></p> <p><code>pppControlLayerCreate()</code></p> <p><code>pppControlLayerDelete()</code></p>	<p>Obsolete. Wind River PPP 6.x uses configuration parameters (also called sysvars) to configure PAP, CHAP, LCP, and IPCP.</p> <p>See the <i>Wind River PPP for VxWorks 6 Programmer's Guide</i>, 6.6: <i>Configuration Options</i>.</p>
Interface layer	<p><code>pppInterfaceLayer()</code></p> <p><code>pppInterfaceLayerCreate()</code></p> <p><code>pppInterfaceLayerDelete()</code></p>	<p>This component has been replaced with the PPP driver.</p> <p>Include the <code>IPPPP_IF_INIT</code> and <code>IPPPP_IF_ATTACH</code> components.</p> <p>Use the routines <code>ipcom_drv_ppp_create()</code> and <code>ipcom_drv_ppp_if_init()</code>.</p>

Table 6-3 Comparison of Routines (cont'd)

Description	Wind River PPP 2.x	Wind River PPP 6.x
Modem	modemConnect() modemDisconnect()	<p>To use a null-modem connection, in the Workbench GUI, set the global parameter PPP_RUNMODE to "wincompat".</p> <p>At the command line, use the following command:</p> <p>sysvar set -o ippmp.runmode wincompat</p> <p>See the <i>Wind River PPP for VxWorks 6 Programmer's Guide</i>, 6.6: <i>Configuration Options</i>.</p>
MUX adapter	muxAdapterCreate() muxAdapterDelete() pppMuxAdapter()	<p>The MUX adapter is obsolete. For equivalent functionality, use the routines ipcom_drv_ppp_create() and ipcom_drv_ppp_exit(), located in ipcom/port/vxworks/src.</p>
Initiate Wind River PPP	windNetPPPInit()	ippmp_create() ippmp_start()
PPPoE Access Concentrator name	pppOEServiceNameAdd()	<p>In the Workbench GUI, use the configuration parameter PPPOE_AC_NAME.</p> <p>At the command line, use the following command:</p> <p>sysvar set -o ippmp.pppoe.ac_name name</p> <p>For more information, see the <i>Wind River PPP for VxWorks 6 Programmer's Guide</i>, 6.6: <i>Configuration Options</i>.</p>

Table 6-3 **Comparison of Routines** (cont'd)

Description	Wind River PPP 2.x	Wind River PPP 6.x
Create PPPoE interface	pppOEthernetCreate()	Use the following shell command: ifconfig pppoe0 create (This is only required for PPPoE clients. PPPoE interfaces on the server are created automatically.)
Delete PPPoE interface	pppOEthernetDelete()	Use the following shell commands. To shut down a PPPoE interface: ifconfig interface_name down To detach from a PPPoE interface that is already down: ifconfig interface_name up down-detach

6.4 Library and Routine Changes

All Wind River PPP 2.x routines have been replaced. Wind River PPP 6.x uses only the following routines:

- **ippmpp_login()**
- **ippmpp_example_action_cb()**