WIND RIVER

Wind River® PPP for VxWorks® 6

PROGRAMMER'S GUIDE

6.6

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

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

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

http://www.windriver.com

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

http://www.windriver.com/support

Wind River PPP for VxWorks 6 Programmer's Guide, 6.6

13 Nov 07
Part #: DOC-16148-ND-00

Contents

1
Overview

1.1 Introduction

Wind River PPP implements a Point-to-Point Protocol (PPP). This chapter introduces you to Wind River PPP, lists the RFCs that it implements, and shows you where to look in this manual for more detailed information.

1.2 Technology Overview

1.2.1 Standards Compliance

Wind River PPP supports the following standards:

- RFC 1321, *The MD5 Message-Digest Algorithm* (used with CHAP)
- RFC 1332, *The PPP Internet Protocol Control Protocol* (Sections 3.2., *IP-Compression-Protocol*, and 4. *Van Jacobson TCP/IP header compression*, are not currently implemented.)
- RFC 1334, *PPP Authentication Protocols*
- RFC 1661, *The Point-to-Point Protocol* (Section 6.3, *Quality-Protocol*, is not currently implemented.)
- RFC 1662, *PPP in HDLC-like Framing*
- RFC 1877, *PPP IPCP Extensions for Name Server Addresses*
- RFC 1994, *Challenge/Handshake Authentication Protocol*
- RFC 2472, *IPv6 over PPP*
- RFC 2516, *A Method for Transmitting PPP over Ethernet (PPPoE)*

1.3 Product Overview

Wind River PPP allows the establishment, authentication, and control of PPP connections and PPP over Ethernet (PPPoE) implementations.

PPP provides a standard method for transporting multi-protocol datagrams over point-to-point links.

PPP is designed for simple links that transport packets between two peers. These links provide full-duplex, simultaneous, bidirectional operation, and are assumed to deliver packets in order.

Wind River PPP is capable of operating across most DTE/DCE interfaces, such as EIA RS-232-E, EIA RS-422, and CCITT V.35. The only absolute requirement imposed by PPP is the provision of a full-duplex circuit, either dedicated or circuit-switched, that can operate in asynchronous (start/stop), bit-synchronous, or octet-synchronous mode, transparent to PPP Data Link Layer frames.

PPPoE provides the ability to connect a network of hosts over a simple bridging device to a remote access concentrator. In this model, each host uses its own PPP stack, so each user is presented with a familiar user interface.

Access control, billing, and type of service can be managed on a per-user, rather than a per-site, basis.

To provide a point-to-point connection over Ethernet, each PPP session must learn the Ethernet address of the remote peer, as well as establish a unique session identifier. PPPoE defines a protocol that can be used to learn the Ethernet address of servers that provide PPPoE services.

1.3.1 Wind River PPP Features

Wind River PPP includes the following features:

- **PPP client**

  Wind River PPP can act as a PPP client and connect to PPP servers. The PPP client can optionally authenticate itself with the server using Password Authentication Protocol (PAP) or Challenge-Handshake Authentication Protocol (CHAP).

- **PPP server**

  Wind River PPP can act as a PPP server and accept an unlimited number of connections from PPP clients. The clients can optionally be authenticated using PAP or CHAP.

- **PPPoE server**

  Wind River PPP includes a PPPoE server implementation, which can accept any number of connections to PPPoE clients.

- **PPPoE client**

  Wind River PPP includes a PPPoE client implementation, capable of connecting to PPPoE servers.

- **PPP shell command**

  Wind River PPP includes a shell command in order to configure or list PPP and PPPoE interfaces.

- **Multiple network type support**

  Wind River PPP can run under different network layers and includes built-in support for IPv4 and IPv6.

- **Multiple driver type support**

  Wind River PPP can run over various network drivers, and it includes built-in support for Wind River serial drivers and for PPPoE with enhanced network drivers (ENDs).

- **Multiple PPP framing support**

  Wind River PPP includes bit-synchronous and asynchronous high-level data link control (HDLC) framing and allows you to implement and add your own framing techniques.

- **Unlimited PPP connections**

  Wind River PPP supports as many PPP connections as the system resources permit.

1.4 Additional Information

The following sections describe additional information about the technologies described in this book.

1.4.1 Delivery Structure

Wind River PPP files are located in the directory *installDir*/**components/ip_net2-6.*x*/ipppp**, as follows:

Table 1-1 **File Locations**

Folder	Filenames
config	**ipppp_config.h**
gmake	**ipppp.mk**
	Makefile

Table 1-1 **File Locations** (cont'd)

Folder	Filenames
include	**ipppp.h**
	ipppp_ipstack.h
src	**ipppp.c**
	ipppp_cmd_pppconfig.c
	ipppp_h.h
	ipppp_pppoe.c

The following Wind River PPP C files are located in the directory *installDir***/components/ip_net2-6.***x***/osconfig/vxworks/src/ipnet**:

- **ipppp_config.c**
- **ipppp_example.c**

1.4.2 Latest Release Information

The latest information on this release can be found in the release notes. Release notes are available from the Wind River Online Support site:

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

In addition, this site includes links to topics such as known problems, fixed problems, documentation, and patches.

NOTE: Wind River strongly recommends that you visit the Online Support Web site before installing or using this product. The Online Support Web site may include important software patches or other critical information regarding this release.

For information on accessing the Wind River Online Support site, see the *Customer Services* section of your Platform getting started guide.

2
Understanding Wind River PPP

2.1 Introduction

This chapter provides a brief overview of PPP and PPPoE. For a more detailed description of these protocols, see RFC 1661, *The Point-to-Point Protocol*, and RFC 2516, *A Method for Transmitting PPP over Ethernet (PPPoE).*

2.2 PPPoE

Wind River recommends that readers unfamiliar with PPPoE should read RFC 2516, since a basic understanding of PPPoE is required.

Modern access technologies are faced with several conflicting goals. It is desirable to connect multiple hosts at a remote site through the same access device on customer premises. It is also a goal to provide access control and billing functionality in a manner similar to dial-up services using PPP. In many access technologies, the most cost-effective method to attach multiple hosts to the customer on-site access device is using Ethernet. In addition, it is desirable to keep the cost of this device as low as possible, while requiring as little configuration as possible.

PPPoE allows several hosts to connect to a remote access concentrator. Each host uses its own PPP stack. Each PPP session must learn the Ethernet address of the remote peer to which it connects and establish a unique session identifier. PPPoE includes a discovery protocol that provides this.

2.2.1 The PPPoE Protocol

The PPPoE protocol uses six-byte headers sent in Ethernet frames with the MAC type set to either **0x8863** (discovery stage) or **0x8864** (PPP session stage). The PPPoE header consists of a one-byte version/type field, a one-byte code field, a two-byte session ID, and a two-byte length field.

The PPPoE protocol has two distinct stages: the discovery stage and the PPP session stage. When a host wishes to initiate a PPPoE session, it must first perform discovery to identify the Ethernet MAC address of the peer and establish a PPPoE session ID. While PPP defines a peer-to-peer relationship, discovery is inherently a client-server relationship.

In the discovery process, a host (the client) discovers an access concentrator (the server). Based on network topology, the host may be able to communicate with more than one access concentrator. The discovery stage allows the host to discover all access concentrators and then select one. When discovery completes successfully, both the host and the selected access concentrator have the information they need to build their point-to-point connection over Ethernet.

The discovery stage remains stateless until a PPP session is established. Once a PPP session is established, both the host and the access concentrator must allocate the resources for a PPP virtual interface.

2.2.2 PPPoE Discovery Stage

There are four steps to the discovery stage. When it completes, both peers have acquired the PPPoE session ID and the peer's Ethernet address, which together uniquely define the PPPoE session. These steps are as follows:

The host transmits a PADI

The host (the client) broadcasts a PPPoE Active Discovery Initiation (PADI) packet to the local Ethernet LAN. This packet signifies the need for a PPP server, that is, the PPPoE access concentrator.

The Access Concentrator replies with a PADO

If the access concentrator (the server) is willing to accept another PPP session, it replies with a PPPoE Active Discovery Offer (PADO) packet sent directly to the host, listing its available services.

The host sends a PADR to the chosen access concentrator

After the host broadcasts the PADI, it may receive more than one PADO. The host looks through the PADO packets it receives and chooses one. The choice may be based on the access concentrator name, or on the available services offered. The host then sends one PPPoE Active Discovery Request (PADR) packet to the access concentrator that it has chosen.

The access concentrator replies with a PADS

When the access concentrator receives a PADR packet, it prepares to begin a PPP session. It generates a unique session ID for the PPPoE session and replies to the host with a PPPoE Active Discovery Session-confirmation (PADS) packet. Immediately after sending the PADS, the access concentrator proceeds to the PPP session stage. When the host receives the PADS packet, it also proceeds to the PPP session stage.

NOTE: The discovery stage remains stateless until after the fourth step, when the host has chosen an access concentrator and been accepted. The discovery stage is then completed and the session stage begins.

2.2.3 PPPoE PPP Session Stage

After the discovery stage, the host and the access concentrator can transmit PPP frames in PPPoE packets over Ethernet, just as in any other PPP encapsulation. All PPPoE session packets are sent unicast and use a reserved Ethernet frame. The PPPoE payload contains one unfragmented PPP frame each, which begins with the PPP Protocol-ID.

NOTE: The maximum transmission unit (MTU) and maximum receive unit (MRU) of a PPPoE interface are both 1492 bytes (because the PPPoE header is six bytes long.)

3
Configuring and Building Wind River PPP

This chapter describes VxWorks features; it does not go into detail about the mechanisms by which VxWorks-based systems and applications are configured and built. The tools and procedures used for configuration and build are described in the *Wind River Workbench User's Guide* and the *VxWorks Command-Line Tools User's Guide*.

NOTE: In this guide, as well as in the VxWorks API references, VxWorks components and their configuration parameters are identified by the names used in component description files. The names take the form, for example, of **INCLUDE_FOO** and **NUM_FOO_FILES** (for components and parameters, respectively).

You can use these names directly to configure VxWorks using the command-line configuration facilities.

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

3.1 Introduction

This chapter describes how to build a VxWorks bootable image that includes Wind River PPP. Wind River PPP cannot run in a real-time process (RTP).

For run-time configuration, see *4. Run-time Configuration*.

3.2 Configuring and Building Wind River PPP

As part of the Wind River General Purpose Platform, VxWorks Edition, Wind River PPP is provided in binary form. For other Wind River Platform products PPP may need to be included in the top-level build before it can be used in applications. See the getting started guide included with your Platform product for details.

In addition to the build configuration described in the *Wind River Platforms Getting Started*, Wind River PPP configuration files contain macros that can be defined or undefined to change available options. Refer to the internal documentation in the following file for more details.

InstallDir/**components/ip_net2-6.***x***/ipcrypto/config/ipppp_config.h**

NOTE: The configuration changes will only take effect after performing a new top-level build for your Wind River Platform product.

The following macros can be set in **ipppp_config.h**:

IPPPP_SYSTEM_PRIORITY
Determines the level of logging.

IPPPP_USE_AUTH_PAP
Includes support for PAP authentication.

IPPPP_USE_AUTH_CHAP
Includes support for CHAP authentication.

IPPPP_USE_RFC1877_CLIENT

IPPPP_USE_RFC1877_SERVER

IPPPP_USE_PPPOE
Includes support for PPPoE.

IPPPP_PPPOE_USE_MD5
Includes support for the PPPoE cookie (Workbench parameter **PPPOE_SECRET_NAME**) used for MD5 checksums to thwart denial of service attacks.

3.3 Configuring VxWorks with Wind River PPP

3.3.1 Components and Parameters

NOTE: Most configuration parameter values correspond to system variables (sysvars) and can be changed at run-time. See *4.2 System Variables*, p.28 for details.

Wind River PPP includes the following components:

Required Components

INCLUDE_IPPPP
This enables PPP. See Table 3-1 for information about configuration parameters.

INCLUDE_IPPPP_CMD
This enables the PPP shell command **pppconfig**. See *4.4 pppconfig*, p.36 for details. This component also requires **INCLUDE_USE_NATIVE_SHELL**, which enables the use of command interpreter commands.

INCLUDE_IPPPPOE
This enables PPPoE. See Table 3-2 for information about configuration parameters.

Optional Components

INCLUDE_IPPPP_INTERFACE_CONFIG
This includes the various parameters to be set for each interface. See Table 3-3 for information about configuration parameters.

INCLUDE_IPPPP_USERS_CONFIG
This includes various parameters to be set for each user. See Table 3-4 for information about configuration parameters.

INCLUDE_IPCOM_SYSVAR_CMD
This enables **sysvar** commands on the target shell. It is necessary if you want to use system variables. See *4.2 System Variables*, p. 28 for more information.

Global, Per-interface, and Per-user Configuration

Many Wind River PPP configuration parameters can be set on a global, per-interface, or per-user basis. Global parameters apply to all users and interfaces. The per-interface and per-user parameters override the global parameters for a given interface or user. For example, the authentication mode can be set for all interfaces using the **PPP_AUTH_MODES** parameter of the **INCLUDE_IPPPP** component. To set an authentication mode for a single interface, use the **PPP_IF_AUTH_MODES_LIST** parameter of the **INCLUDE_IPPPP_INTERFACE_CONFIG** component.

Parameter names that include **LIST**, such as **PPP_IF_AUTH_MODES_LIST**, accept a list of interfaces or users and corresponding values. Each entry in the list is separated by semicolons. For example, to set the authentication mode for interfaces **ppp0** and **ppp1**, set the **PPP_IF_AUTH_MODES_LIST** parameter to:

```
"ppp0=chap;ppp1=pap"
```

The per-user parameters are in the **INCLUDE_IPPPP_USERS_CONFIG** component. Most per-interface parameters are in the **INCLUDE_IPPPP_INTERFACE_CONFIG** component. The **INCLUDE_IPPPPOE** component does have the following per-interface parameters:

- **PPPOE_IF_MAX_ETH_SESSIONS_LIST**
- **PPPOE_IF_SERVICE_NAME_LIST**

Parameters and Default Values

Table 3-1 **Configuration Parameters for INCLUDE_PPP**

Parameter Name	Default Value
PPP_AUTH_MODES Sets the authentication requirements on a PPP interface. The parameter value is an ASCII string with one or more authentication options. There are six options defined. **auth** This requires the peer to authenticate itself before allowing sending or receiving of IPCP or IPV6CP. **noauth** This does not require the peer to authenticate itself (that is, do not ask for authentication.) **pap** The peer must authenticate using the PAP protocol. **refuse-pap** Do not agree to authenticate using PAP. **chap** The peer must authenticate using the CHAP protocol. **refuse-chap** Do not agree to authenticate using CHAP.	**noauth**
PPP_LCP_ECHO_REQ_FAILURE Sets the maximum number of echo request failures before link termination.	**5**
PPP_DEFAULT_BAUDRATE The RS232 serial driver initial baud rate.	**9600**
PPP_INSTALL_CALLBACK_HOOK When **TRUE** compiles in support for user-defined PPP action callback hooks. See the **PPP_ACTION_CALLBACK_HOOK** parameter and *5.4 Callback Routine*, p.42 for details.	**FALSE**

Table 3-1 **Configuration Parameters for INCLUDE_PPP** (cont'd)

Parameter Name	Default Value
PPP_LCP_ECHO_REQ_INTERVAL Sets the link control protocol (LCP) echo request output interval in seconds. Set to **0** to disable.	**60**
PPP_LCP_MRU The LCP maximum receive unit.	**1500**
PPP_LCP_MTU The LCP maximum transmit unit (MTU).	**1500**
PPP_IPCP_IPV4_ADDRESS Local IPv4 address. Setting it to **0** means that the stack will ask the peer to suggest its IPv4 address.	**0.0.0.0**
PPP_ACTION_CALLBACK_HOOK Name for callback routine. Wind River PPP calls this routine to handle various PPP actions, such as initialization, startup, peer IPv4 address requests, peer authentication, login, and so on. See the **PPP_INSTALL_CALLBACK_HOOK** parameter and *5.4 Callback Routine*, p.42 for details. **PPP_ACTION_CALLBACK_HOOK** is applicable only if **PPP_INSTALL_CALLBACK_HOOK** is set to **TRUE**. To avoid compiler errors, **PPP_ACTION_CALLBACK_HOOK** is defined as **NULL** by default.	**NULL**

Table 3-1 **Configuration Parameters for INCLUDE_PPP** (cont'd)

Parameter Name	Default Value
PPP_FLAGS Sets behaviors and options on an interface. The parameter value is an ASCII string with one or more flag options. There are two flag options defined: **defaultroute** This adds a default route to the system routing tables using the peer as the gateway, when the IPCP/IPV6CP negotiation is successfully completed. **proxyarp** This adds a proxy ARP route flag (**RTF_PROTO2**) to the peer host route when IPCP negotiation is successfully completed. When this is set, the target will act as proxy for the host. One or more additional interfaces on the target must have proxy ARP enabled. ARP requests for the host address that come in on these interfaces will be processed by the target. See the *Wind River Network Stack for VxWorks 6 Programmer's Guide Volume 1* for information about enabling proxy ARP.	**proxyarp**
PPP_PASSWD The PPP client password used when the peer PPP server requires authentication using PAP or CHAP.	**kallekula**
PPP_USERNAME The PPP client username used when the peer PPP server requires authentication using PAP or CHAP.	**ppp**
PPP_IPCP_PEER_IPV4_ADDRESS Suggests the peer IPv4 address. The peer IPv4 address will be suggested to the peer only if the peer requests it by requesting IP address **0**.	**10.1.4.1**
PPP_IPCP_PEER_IPV4_ADDRESS_POOL Defines which IPv4 address pool should be used for a session to supply peers with IPv4 addresses when running as a PPP or PPPoE server.	**10.1.3.1-10.1.3.255**

Table 3-1 **Configuration Parameters for INCLUDE_PPP** (cont'd)

Parameter Name	Default Value
PPP_IPCP_PEER_IPV4_POOL_NAME Defines an IPv4 address pool used to supply peers with IPv4 addresses when running as a PPP or PPPoE server. The pool must be defined with a name and with starting and ending IPv4 addresses.	**default**
PPP_IPCP_PRIMARY_DNS_ADDRESS PPP client IPv4 primary DNS address. The configured IPv4 address is given to the peer (the client) if the peer requests a primary DNS address.	"
PPP_IPCP_PRIMARY_NBNS_ADDRESS PPP client IPv4 primary NBNS address. The configured IPv4 address is given to the peer if the peer requests a primary NBNS (WINS) address.	**Not set**
PPP_IPCP_SECONDARY_DNS_ADDRESS PPP client IPv4 secondary DNS address. The configured IPv4 address is given to the peer if the peer requests a secondary DNS address.	**Not set**

Table 3-1 **Configuration Parameters for INCLUDE_PPP** (cont'd)

Parameter Name	Default Value
PPP_IPCP_SECONDARY_NBNS_ADDRESS PPP client IPv4 secondary NBNS address. The configured IPv4 address is given to the peer if the peer requests a secondary NBNS (WINS) address.	**Not set**
PPP_RUNMODE Sets the runmode on a PPPoE interface. The parameter value is an ASCII string with one or more runmode options. There are four runmode options defined: **start** When set, the interface automatically starts when attached. Otherwise requires an **IP_SIOCSIFFLAGS ioctl( )** request with the **IP_IFF_UP** flag. PPPoE interfaces are always started automatically when they are created. **passive** Enables the passive option in the LCP. When set a PPP interface attempts only once to initiate a connection when started. If no reply is received from the peer after a number of tries, the interface waits passively for a valid LCP configuration request packet from the peer. **exit** Exit LCP after a connection; that is, do not accept new connections unless the PPP interface is reopened with an **IP_SIOCSIFFLAGS ioctl( )** request with the **IP_IFF_UP** flag. **wincompat** Supports connection to a Windows machine using PPP and a null-modem connection. This define is necessary because Windows requires CLIENT/CLIENTSERVER strings before starting standard PPP.	**start, passive, wincompat**

Table 3-2 **Configuration Parameters for INCLUDE_PPPoE**

Workbench Name and Macro Name	Default Value
PPPOE_AC_NAME The PPPoE access concentrator name (AC-Name).	**Windriver PPPoE Server**
PPPOE_SERVER Enables a PPPoE server on a device or an interface. If the option is set to "**1**", the PPPoE server is enabled. If the option is set to "**0**", the PPPoE server is disabled for the whole device or for a specific interface.	**0**
PPPOE_SECRET_NAME Secret name for a PPPoE cookie. Use this parameter in an MD5 checksum calculation to thwart denial-of-service (DoS) attacks. The definition of the string does not need to match any remote configuration. Wind River recommends that you change this value from its default setting.	**puttoanything here**
PPPOE_MAX_SESSIONS Maximum number of PPPoE sessions allowed before incoming PPPoE PADI packets are ignored. That is, the PPPoE server will not accept any more connections.	**16**
PPPOE_MAX_ETH_SESSIONS Global setting for the maximum number of PPPoE sessions per Ethernet interface.	**8**
PPPOE_IF_MAX_ETH_SESSIONS_LIST Maximum number of PPPoE sessions per Ethernet interface, if the maximum number varies by interface. Note that the interface name used is the Ethernet interface name and not the PPPoE interface name. This parameter cannot be configured from the command line.	**Not set**
PPPOE_SERVICE_NAME This is the service name the stack accepts. Do not set this option to accept the **"any"** service name.	**myservice**
PPPOE_IF_SERVICE_NAME_LIST Per-interface list of service names that the stack accepts. Do not set this option to accept the **"any"** service name.	**IP_NULL**

Table 3-3 **Configuration Parameters for INCLUDE_IPPPP_INTERFACE_CONFIG**

Workbench Name and Macro Name	Default Value
PPP_IF_AUTH_MODES_LIST Authentication mode.	**Not set**
PPP_IF_LCP_ECHO_REQ_FAILURE_LIST Echo failure.	**Not set**
PPP_IF_IPCP_PEER_IPV4_ADDRESS_POOL_LIST IPv4 pool address list.	**Not set**
PPP_IF_DEFAULT_BAUDRATE_LIST Initial baud rate.	**Not set**
PPP_IF_LCP_ECHO_REQ_INTERVAL_LIST Interface LCP echo request interval.	**Not set**
PPP_IF_RUNMODE_LIST Interface runmode.	**Not set**
PPP_IF_LCP_MRU_LIST LCP maximum receive unit.	**Not set**
PPP_IF_LCP_MTU_LIST LCP maximum transmit unit.	**Not set**
PPP_IF_FLAGS_LIST PPP flags.	**Not set**
PPP_IF_IPCP_PEER_IPV4_ADDRESS_LIST Peer IPv4 address list.	**Not set**
PPP_IF_IPCP_PRIMARY_DNS_ADDRESS_LIST Primary DNS address list.	**Not set**
PPP_IF_IPCP_SECONDARY_DNS_ADDRESS_LIST Secondary DNS address list.	**Not set**
PPP_IF_IPCP_IP4V_ADDRESS_LIST Local IPv4 address list.	**Not set**

Table 3-4 **Configuration Parameters for INCLUDE_IPPP_USERS_CONFIG**

Workbench Name and Macro Name	Default Value
PPP_USERS_IPCP_PEER_IPV4_ADDRESS_POOL_LIST Users IPv4 peer address pool list.	**Not set**
PPP_USERS_LCP_ECHO_REQ_INTERVAL_LIST Users LCP echo request interval.	**Not set**
PPP_USERS_IPCP_PEER_IPV4_ADDRESS_LIST Users peer IPv4 address list.	**Not set**
PPP_USERS_IPCP_IPV4_ADDRESS_LIST Users local IPv4 address list.	**Not set**
PPP_USERS_IPCP_PRIMARY_DNS_ADDRESS_LIST Users primary DNS address list.	**Not set**
PPP_USERS_IPCP_SECONDARY_DNS_ADDRESS_LIST Users secondary DNS address list.	**Not set**

3.3.2 Configuring an Additional Interface

If you are building a target, you will need two or more network interfaces. The following sections describe how to add and configure the necessary interfaces.

Which procedure you follow depends on whether your BSP supports VxBus. If it does, the system will automatically detect any additional drivers, and you only need to configure them. In such a case, perform only the procedure described in *Configuring an Additional Interface*, p.23.

Checking for VxBus Support

You can tell whether your BSP supports VxBus by examining the following file:

target/config/*bspName***/config.h**

If this file contains the line **#define INCLUDE_VXBUS**, it supports VxBus, and you do not need to perform a separate procedure to add a network interface.

If this file does not contain the line **#define INCLUDE_VXBUS**, you must edit the file to add the necessary interfaces. See *Adding a Network Interface—Legacy END Drivers*, **p.23**, for further information.

Adding a Network Interface—Legacy END Drivers

Perform this procedure only if your BSP does not support VxBus.

Before configuring PPP, check whether your BSP supports a second interface. If not, you can add that support. To learn whether your BSP already supports a second interface and how to enable it, read the BSP reference page in the Workbench online help.

To add a network interface, you must edit **target/config/***bspName***/configNet.h**.

Each BSP requires specific edits to add support for an interface. The following example shows how to add support for an additional **fei** interface for the **pcPentium** BSP.

Example 3-1 **Adding a Network Interface to a BSP (FEI Driver)**

1. Locate the following lines:

```
#ifdef INCLUDE_FEI_END
    { 0, FEI82557_LOAD_FUNC, FEI82557_LOAD_STRING, FEI82557_BUFF_LOAN,
    NULL, FALSE},
#endif /* INCLUDE_FEI_END */
```

2. Add the following line just before the **#endif** line:

```
    { 1, FEI82557_LOAD_FUNC, FEI82557_LOAD_STRING, FEI82557_BUFF_LOAN,
    NULL, FALSE},
```

3. If more than two interfaces are necessary, repeat step 2, incrementing the interface number for each additional interface.

4. Ensure that *installDir***/vxworks-6.***x***/target/config/***bspName***/config.h** includes the following **define**:

```
    #define INCLUDE_FEI_END
```

If you are using a different BSP or interface, read the BSP reference page in Workbench online help.

Configuring an Additional Interface

Once you have added a network interface, you must configure it with an IP address or network mask. You can configure the interface at build time or at run time.

Configuring an Additional Interface at Build Time

To configure an interface at build time, include an **INCLUDE_IPNET_IFCONFIG_*N*** component (one for each interface). Each of these components contains an **IFCONFIG_*N*** parameter.

For each **IFCONFIG_*N***, edit the following fields:

ifname
Specifies the name of the Ethernet interface, for example, **ifname fei0**. If the interface name is missing after **ifname** (the default setting), the END device name will be used.

devname
Specifies the driver to which this interface should attach itself, for example, **fei0**. The default setting **driver** instructs VxWorks to retrieve the device name from the device boot parameters.

inet
Specifies the interface IPv4 address and subnet, for example, **inet 10.1.2.100/24**. Instead of IPv4 address, the following syntaxes can also be used:

inet driver (default)
Specifies that the address and mask should be read from the BSP.

inet dhcp
Specifies that the address and mask should be received from a DHCP server. The gateway might also be received from that server (depending on the DHCP server configuration).

inet rarp
Specifies that the address and mask should be received from an RARP server.

gateway
Specifies the default gateway used for IPv4, for example, **gateway 10.1.2.1**. Only one default gateway can be specified. **gateway driver** can be used to take the gateway from the boot parameters.

inet6
Specifies the interface IPv6 address and subnet, for example, **inet6 3ffe:1:2:3::4/64**. The **tentative** keyword can be inserted before the address if the stack should perform duplicate address detection on the address before assigning it to the interface, for example, **tentative 3ffe:1:2:3::4/64**.

gateway6
The default gateway used for IPv6. Only one default gateway can be specified.

Configuring an Additional Interface by Editing config.h

You can also configure an additional interface by editing the **config.h** file for your BSP—that is, **target/config/***bspName***/config.h**. In this case, specify the values for **IFCONFIG_***N* directly in the file, using a **#define** statement. For example:

```
#define IFCONFIG_1 "ifname", "devname driver","inet driver", \
"gateway driver", "inet6 3ffe:1:2:3::10/64"
```

Configuring an Additional Interface at Run Time

If you are not ready to configure the interface at build time, you can configure it at run time. This procedure consists of two steps:

1. Attaching a protocol.
2. Configuring the address and subnet mask.

To perform these steps, run an **ipAttach** shell command on the target, followed by an **ifconfig**. For example:

```
[vxWorks *] # ipAttach 1,"fei"
[vxWorks *] # ifconfig "fei1 10.0.0.2 netmask 255.255.255.0 up"
```

The parameters for the **ifconfig** command are specified in *Configuring an Additional Interface at Build Time*, p.24.

3.4 Building VxWorks With Wind River PPP

For information about building VxWorks with Wind River PPP, including build options, image types, and so on, see the *Wind River Workbench User's Guide* and the *VxWorks Command-Line Tools User's Guide.*

3.5 Booting the Target and Testing PPP

NOTE: If you see an error message indicating undefined references to **ipfwSysctl** or **ipfwPushRegister**, you must must rebuild your Platform. For instructions, see the getting started guide for your Platform.

When you have finished building the image, verify that Wind River PPP was included in the build.

1. Boot the target with your VxWorks image.
2. Issue the following shell command:

   ```
   [vxWorks *]# ipversion
   ```

   This lists product versions for installed networking components. The list should include **IPPPP**.

NOTE: The **ipversion** command is only available if the **INCLUDE_USE_NATIVE_SHELL** compnent is included in the image.

4
Run-time Configuration

4.1 Introduction

To configure Wind River PPP at run-time, use the following methods:

- System variables (sysvars). Use sysvars to change configuration parameters set in Workbench. See *4.2 System Variables*, p.28 for details.
- **ioctl( )** requests. Use **ioctl( )** requests to change driver or link layer parameters. See *4.3 ioctl( ) Requests*, p.31 for details.
- **pppconfig** shell command. Use the **pppconfig** shell command to change Wind River PPP configuration from the command line. See *4.4 pppconfig*, p.36 for details.

4.2 System Variables

Wind River PPP system variables (sysvars) correspond directly to the Workbench configuration parameters described in *3.3 Configuring VxWorks with Wind River PPP*, p. 13. Global configuration parameters each correspond to a single sysvar. Per-interface and per-user configuration parameters correspond to lists of sysvars, one for each user or interface.

For example, the configuration parameter for the client password is **PPP_PASSWD**. The corresponding sysvar is **ipppp.password**. See Table 4-1 for global configuration sysvars.

The configuration parameter for per-interface authentication modes is **PPP_IF_AUTH_MODES_LIST**. If per-interface authentication is set for the interfaces **ppp0** and **ppp1**, the corresponding sysvars would be **ipcom.if.ppp0.ipppp.auth** and **ipcom.if.ppp1.ipppp.auth**. See Table 4-2 for per-interface configuration sysvars and Table 4-3 for per-user configuration sysvars.

You can manipulate sysvars programmatically using the generic API defined in *installDir***/components/ip_net2-6.***x***/ipcom/include/ipcom_sysvar.h** or using the **sysvar** command-line command. See the *Wind River Network Stack for VxWorks 6 Programmer's Guide Volume 1* for more information about sysvars.

Table 4-1 **Global Configuration Parameters and Sysvars**

Configuration Parameter	Sysvar
PPP_DEFAULT_BAUDRATE	**ipppp.baudrate**
PPP_RUNMODE	**ipppp.runmode**
PPP_FLAGS	**ipppp.flags**
PPP_AUTH_MODES	**ipppp.auth**
PPP_LCP_MRU	**ipppp.lcp.mru**
PPP_LCP_MTU	**ipppp.lcp.mtu**
PPP_LCP_ECHO_REQ_INTERVAL	**ipppp.lcp.echo_interval**
PPP_LCP_ECHO_REQ_FAILURE	**ipppp.lcp.echo_failure**
PPP_IPCP_IPV4_ADDRESS	**ipppp.ipcp.addr**
PPP_IPCP_PEER_IPV4_ADDRESS	**ipppp.ipcp.dstaddr**

Table 4-1 **Global Configuration Parameters and Sysvars** (cont'd)

Configuration Parameter	Sysvar
PPP_IPCP_PEER_IPV4_POOL_NAME	**ipppp.ipcp.pool**
PPP_IPCP_PEER_IPV4_ADDRESS_POOL	Formed by joining "**ipppp.ipcp.pool.**" and the **PPP_IPCP_PEER_IPV4_POOL_NAME** configuration parameter. For example, with a pool name of "default", the sysvar name would be **ipppp.ipcp.pool.default**.
PPP_USERNAME	**ipppp.username**
PPP_PASSWD	**ipppp.password**
PPP_IPCP_PRIMARY_DNS_ADDRESS	**ipppp.ipcp.primary_dns_address**
PPP_IPCP_SECONDARY_DNS_ADDRESS	**ipppp.ipcp.secondary_dns_address**
PPP_IPCP_PRIMARY_NBNS_ADDRESS	**ipppp.ipcp.primary_nbns_address**
PPP_IPCP_SECONDARY_NBNS_ADDRESS	**ipppp.ipcp.secondary_nbns_address**
PPPOE_SERVER	**ipppp.pppoe.server**
PPPOE_AC_NAME	**ipppp.pppoe.ac_name**
PPPOE_SECRET_NAME	**ipppp.pppoe.secret_name**
PPPOE_MAX_SESSIONS	**ipppp.pppoe.max_sessions**
PPPOE_MAX_ETH_SESSIONS	**ipppp.pppoe.max_eth_sessions**
PPPOE_SERVICE_NAME	**ipppp.pppoe.service_name**

Table 4-2 **Per-interface Configuration Parameters and Sysvars**

Configuration Parameter	Sysvar
PPP_IF_DEFAULT_BAUDRATE_LIST	**ipcom.if.***ifname***.ipppp.baudrate**
PPP_IF_RUNMODE_LIST	**ipcom.if.***ifname***.ipppp.runmode**
PPP_IF_FLAGS_LIST	**ipcom.if.***ifname***.ipppp.flags**
PPP_IF_AUTH_MODES_LIST	**ipcom.if.***ifname***.ipppp.auth**

Table 4-2 **Per-interface Configuration Parameters and Sysvars** (cont'd)

Configuration Parameter	Sysvar
PPP_IF_LCP_MRU_LIST	**ipcom.if.*ifname*.ipppp.lcp.mru**
PPP_IF_LCP_MTU_LIST	**ipcom.if.*ifname*.ipppp.lcp.mtu**
PPP_IF_LCP_ECHO_REQ_INTERVAL_LIST	**ipcom.if.*ifname*.ipppp.lcp.echo_interval**
PPP_IF_LCP_ECHO_REQ_FAILURE_LIST	**ipcom.if.*ifname*.ipppp.lcp.echo_failure**
PPP_IF_IPCP_IPV4_ADDRESS_LIST	**ipcom.if.*ifname*.ipppp.ipcp.addr**
PPP_IF_IPCP_PEER_IPV4_ADDRESS_LIST	**ipcom.if.*ifname*.ipppp.ipcp.dstaddr**
PPP_IF_IPCP_PEER_IPV4_ADDRESS_POOL_ LIST	**ipcom.if.*ifname*.ipppp.ipcp.pool**
PPP_IF_IPCP_PRIMARY_DNS_ ADDRESS_LIST	**ipcom.if.*ifname*.ipppp.ipcp.primary_dns_address**
PPP_IF_IPCP_SECONDARY_DNS_ ADDRESS_LIST	**ipcom.if.*ifname*.ipppp.ipcp.secondary_dns_address**
PPPOE_IF_MAX_ETH_SESSIONS_LIST	**ipcom.if.*ifname*.ipppp.pppoe.max_eth_sessions**
PPPOE_IF_SERVICE_NAME_LIST	**ipcom.if.*ifname*.ipppp.pppoe.service_name**

Table 4-3 **Per-user Configuration Parameters and Sysvars**

Configuration Parameter	Sysvar
PPP_USERS_LCP_ECHO_REQ_INTERVAL_LIST	**ipcom.users.*user*.ipppp.lcp.echo_ interval**
PPP_USERS_IPCP_IPV4_ADDRESS_LIST	**ipcom.users.*user*.ipppp.ipcp.addr**
PPP_USERS_IPCP_PEER_IPV4_ADDRESS_LIST	**ipcom.users.*user*.ipppp.ipcp.dstaddr**
PPP_USERS_IPCP_PEER_IPV4_ADDRESS_ POOL_LIST	**ipcom.users.*user*.ipppp.ipcp.pool**

Table 4-3 **Per-user Configuration Parameters and Sysvars** (cont'd)

Configuration Parameter	Sysvar
PPP_USERS_IPCP_PRIMARY_DNS_ADDRESS_LIST	**ipcom.users.*user*.ipppp.ipcp.primary_dns_address**
PPP_USERS_IPCP_SECONDARY_DNS_ADDRESS_LIST	**ipcom.users.*user*.ipppp.ipcp.secondary_dns_address**

4.3 ioctl( ) Requests

Use **ioctl( )** requests to get and set driver and link layer data.

4.3.1 ioctl( ) Data Structures

Important data types for Wind River PPP **ioctl( )** requests are defined in *InstallDir*/**components/ip_net2-6.*x*/ipppp/include/ipppp.h**.

All Wind River PPP **ioctl( )** requests use the **Ip_pppreq** struct for passing data. The **pppr_name** member must be set to the interface name. Different requests use different members of the **pppru** union. See Table 4-4 and Table 4-5 for details.

Ip_pppreq

```
struct Ip_pppreq
{
  char    pppr_name[IP_IFNAMSIZ];       /* if name, e.g. "ppp0" */
  int     link_index;   /* link_index, currently unused */

  union
  {
      Ip_u32               ppp_flags;    /* Get/Set PPP flags. */
      Ipcom_ppp_conf       pppconf;      /* Get/Set PPP protocol
                                            configuration. */
      char                 username[IPCOM_AUTH_USERNAME_MAX];
      char                 password[IPCOM_AUTH_PASSWORD_MAX];
      char                 if_name[IP_IFNAMSIZ];
      Ipcom_ppp_drvconf    drvconf;       /* Get PPP driver configuration. */
      Ip_u32               drv_baudrate;  /* Set PPP driver baudrate. */
      int                  drv_wincompat; /* Windows compat mode */
      Ipcom_ppp_debuginfo  debuginfo;    /* Get PPP debug info. */
  }
  pppru;
};
```

Ipcom_ppp_drvconf

The **Ipcom_ppp_drvconf** struct is used for driver configuration data.

```
typedef struct Ipcom_ppp_drvconf_struct
{
    Ip_u32     drv_flags;     /* PPP driver flags. */
    Ip_u32     baudrate;      /* Baudrate. */

    Ip_u32     rcv_accm;      /* Recv asyncmap. Default 0xffffffff */
    Ip_u32     snd_accm;      /* Send asyncmap. Default 0xffffffff */
    Ip_u16     mru;           /* Maximum Receive Unit. Default 1500. */
    Ip_u16     mtu;           /* Maximum Transmit Unit. Default 1500. */
}
Ipcom_ppp_drvconf;
```

The struct members are as follows:

drv_flags
The PPP driver flags. These flags cannot be modified, since they define the nature of the PPP driver.

The available flags are:

IP_DRVPPP_FLAG_NOFLAG
No initial/trailing PPP flag (0x7e).

IP_DRVPPP_FLAG_NOESCAPE
Do not byte-escape output/input packets.

IP_DRVPPP_FLAG_NOFCS
Do not input-check or add output FCS.

IP_DRVPPP_FLAG_NOACFC
Do not send or receive addr(0xff) ctrl(0x03).

baudrate
The current PPP RS232 serial driver baudrate.

rcv_accm
The current PPP receive asynchronous control character map (ACCM) bit-field.

snd_accm
The current PPP send asynchronous control character map (ACCM) bit-field.

mru
The current driver maximum receive unit (MRU).

mtu
The current driver maximum transmit unit (MTU).

Ipcom_ppp_conf

The **Ipcom_ppp_conf** struct is used for link layer configuration data. These values are set to default when the interface is first initialized. To change any of the default values use an **IP_SIOCXPPPSCONF ioctl( )** request in the action callback routine. See *5.4 Callback Routine*, p.42 for details.

```
typedef struct Ipcom_ppp_conf_struct
{
    Ip_u16      restart;
    Ip_u8       max_failure;
    Ip_u8       max_configure;
    Ip_u8       max_terminate;
    Ip_u8       max_authreq;
    Ip_u8       pad[2];
}
Ipcom_ppp_conf;
```

The struct members are as follows:

restart
Sets the PPP Configuration/Termination Request retransmission timeout in seconds. Default 3.

max_failure
Sets the maximum number of PPP configuration negative-acknowledges (NAKs) returned before sending Configuration Reject. Default 10.

max_configure
Sets the maximum number of PPP configuration request retransmissions. Default 10.

max_terminate
Sets the maximum number of PPP termination request retransmissions. Default 2.

max_authreq
Specifies the maximum number of PPP authentication retransmissions to be made before the link terminates. Default 3.

Ipcom_ppp_debuginfo

The **Ipcom_ppp_debuginfo** struct contains debugging data.

```
typedef struct Ipcom_ppp_debuginfo_struct
{
    Ip_u32  link_flags;  /* See ipnet_netif.h */
    Ip_u32  pppoe_eth_ifindex;
    Ip_u8   link_state;
    Ip_u8   lcp_state;
    Ip_u8   auth_state;
    Ip_u8   ipcp_state;
    Ip_u8   ipv6cp_state;
}
Ipcom_ppp_debuginfo;
```

4.3.2 Driver ioctl( ) Requests

Table 4-4 **Wind River PPP Driver ioctl( ) Requests**

ioctl( )	Ip_pppreq.pppru member	Description
IP_SIOCXPPPGDRVCONF	**drvconf**	Gets default driver configuration.
IP_SIOCXPPPGDRVINFO	**drvconf**	Gets current driver configuration.
IP_SIOCXPPPSDRVBAUDRATE	**drv_baudrate**	Sets the driver baud rate.
IP_SIOCXPPPSDRVUP		Brings the driver up. For testing purposes.
IP_SIOCXPPPSDRVDOWN		Shuts the driver down. For testing purposes.
IP_SIOCXPPPSDRVWINCOMPAT	**drv_wincompat**	Sets the driver to Windows compatibility mode. See the **PPP_RUNMODE** parameter in Table 3-1 for details.

4.3.3 Link Layer ioctl( ) Requests

Table 4-5 **Wind River PPP Link Layer ioctl( ) Requests**

ioctl( )	Ip_pppreq.pppru member	Description
IP_SIOCXPPPGFLAGS	**ppp_flags**	Bit-mask of the following flags: **IP_PPP_FLAG_NOACCOMP** Disable Address/Control compression in both directions (snd/rcv). **IP_PPP_FLAG_NOACCM** Do not send or ack ACCM confreq. **IP_PPP_FLAG_AUTH** Require the peer to authenticate itself before allowing snd/rcv NCP. **IP_PPP_FLAG_NOAUTH** Do not require the peer to authenticate itself. That is, do not ask for authentication. **IP_PPP_FLAG_REFUSE_PAP** Do not agree to authenticate using PAP. **IP_PPP_FLAG_REFUSE_CHAP** Do not agree to authenticate using MD5 CHAP.
IP_SIOCXPPPSFLAGS	**ppp_flags**	See **IP_SIOCXPPPGFLAGS**.
IP_SIOCXPPPGCONF	**pppconf**	Gets the PPP main configuration.
IP_SIOCXPPPSCONF	**pppconf**	Sets the PPP main configuration.
IP_SIOCXPPPGUSER	**username**	Gets the local user.
IP_SIOCXPPPSUSER	**username**	Sets the local user.
IP_SIOCXPPPGPASSWD	**password**	Gets the local password.

Table 4-5 **Wind River PPP Link Layer ioctl( ) Requests** (cont'd)

ioctl( )	Ip_pppreq.pppru member	Description
IP_SIOCXPPPSPASSWD	**password**	Sets the local password.
IP_SIOCXPPPGPEERUSER	**username**	Gets the peer user.
IP_SIOCXPPPSETIF	**if_name**	Sets the PPPoE Ethernet interface.
IP_SIOCXPPPGDINFO	**debuginfo**	Gets the PPP debug info.

4.4 pppconfig

The command **pppconfig** can be used to configure Wind River PPP from the command line. To enable **pppconfig**, include the components **INCLUDE_IPPPP_CMD** and **INCLUDE_USE_NATIVE_SHELL** in your image.

pppconfig

Name

pppconfig—Configure Wind River PPP

Synopsis

Many of the options can modify either global configuration or per-interface or per-user configuration. To specify per-interface configuration, type the interface name as the first parameter to the **pppconfig** command. To specify per-user configuration, use the option **-user** *username*.

pppconfig *ifname* [**-debug**]
Lists details about a single interface.

pppconfig -a [**-debug**]
Lists details about all interfaces.

pppconfig show
Lists all Wind River PPP sysvars.

pppconfig *ifname* **shutdown**
Shuts down an interface.

pppconfig [*ifname*] **runmode** *runmode_value*
Sets the runmode. Values are the same as the **PPP_RUNMODE** parameter defined in Table 3-1.

pppconfig [*ifname*] **flags** *flags_value*
Sets the Wind River PPP flags. Values are the same as the **PPP_FLAGS** parameter defined in Table 3-1.

pppconfig *ifname* **max-failure** *max_failure_count*
Sets the maximum number of Configure NAKs before sending Configure Reject.

pppconfig *ifname* **max-configure** *max_configure_count*
Sets the maximum number of Configure Request retransmissions.

pppconfig *ifname* **max-terminate** *max_terminate_count*
Sets the maximum number of Termination Request retransmissions.

pppconfig [*ifname*] **baudrate** *baudrate_value*
Sets the driver baud rate.

pppconfig [*ifname*] **mru** *mru_value*
Sets the Maximum Receive Unit.

pppconfig [*ifname*] **mtu** *mtu_value*
Sets the Maximum Transmit Unit.

pppconfig *ifname* **noaccomp** [**1** | **0**]
Enables or disables in/out address/control compression.

pppconfig *ifname* **noaccm** [**1** | **0**]
Enables or disables sending or acknowledging ACCM ConfReq.

pppconfig *ifname* **restart** *restart_value*
Sets the retransmission timeout in seconds.

pppconfig [*ifname* | **-user** *username*] **echo-interval** *echo_interval_value*
Sets the LCP echo request output interval in seconds.

pppconfig [*ifname*] **echo-failure** *echo_failure_value*
Sets the maximum number of LCP echo request failures before terminating.

pppconfig [*ifname*] **auth** [**auth** | **noauth** | **pap** | **chap** | **refuse-pap** | **refuse-chap**]
Sets authentication options. Values are the same as the **PPP_AUTH_MODES** parameter defined in Table 3-1.

pppconfig *ifname* **max-authreq** *max_authreq_value*
Sets the maximum number of authentication retransmissions.

pppconfig *ifname* **username** *username*
Sets the username when logging in on a peer. The interface must be shut down for this operation to succeed.

pppconfig *ifname* **password** *password*
Sets the password when logging in on a peer. The interface must be shut down for this operation to succeed.

pppconfig [*ifname* | **-user** *username*] **addr** *local_address*
Sets the local IPv4 address.

pppconfig [*ifname* | **-user** *username*] **dstaddr** *destination_address*
Sets the destination IPv4 address.

pppconfig [*ifname* | **-user** *username*] **pool** *poolname*
Chooses the pool.

pppconfig setpool *poolname start_address end_address*
Sets the pool.

pppconfig [*ifname* | **-user** *username*] **dns-primary** *address*
Sets the primary DNS IPv4 address.

pppconfig [*ifname* | **-user** *username*] **dns-secondary** *address*
Sets the secondary DNS IPv4 address.

pppconfig [*ifname* | **-user** *username*] **wins-primary** *address*
Sets the primary WINS (NBNS) IPv4 address.

pppconfig [*ifname* | **-user** *username*] **wins-secondary** *address*
Sets the secondary WINS (NBNS) IPv4 address.

pppconfig [*ifname*] **pppoe-server** [**1** | **0**]
Enables (**1**) or disables (**0**) the PPPoE server.

pppconfig max-sessions *max_sessions_count*
Sets the maximum number of PPPoE sessions.

pppconfig [*ifname*] **max-eth-sessions** *max_eth_sessions_count*
Sets the maximum number of PPPoE sessions per interface.

pppconfig *ifname* **pppoe-setif** *interface_name*
Sets the PPPoE client Ethernet interface.

pppconfig *ifname* **drv-up**
Brings up the driver.

pppconfig *ifname* **drv-down**
Shuts down the driver.

5 Wind River PPP Applications

5.1 Introduction

Applications that use Wind River PPP must manage the PPP interfaces and deal with PPP events.

5.2 PPP and Serial Interfaces

Wind River PPP integrates directly with the Wind River Network Stack. If PPP is enabled, the PPP protocol is automatically attached to serial interfaces when they are brought up. This creates a PPP interface named **ppp***n*, where *n* is an assigned interface number starting from 0. For example, the first PPP interface is named **ppp0**.

In addition, if the runmode for the interface is set to "start", PPP starts automatically on the attached interface. To start PPP on an interface that is not set to start automatically, issue an **IP_SIOCSIFFLAGS ioctl( )** request with the **IP_IFF_UP** bit set. See **ipppp_cmd_pppconfig.c**, in the directory *InstallDir***/components/ip_net2-6.***x***/ipppp/src** for examples of getting and setting interface flags using **IP_SIOCGIFFLAGS** and **IP_SIOCSIFFLAGS**.

5.3 PPPoE and Ethernet Interfaces

PPPoE interfaces are not created automatically. Each interface must be created individually and bound to an Ethernet interface. To create an interface, use **ifconfig**. Interface names beginning with "pppoe" are automatically created as PPPoE interfaces. See the *Wind River Network Stack for VxWorks 6 Programmer's Guide Volume 3: Interfaces and Drivers* for details about **ifconfig**. For example, the following will create PPPoE interface **pppoe0**.

```
[vxWorks *]# ifconfig pppoe0 create
```

To bind a PPPoE interface to an Ethernet interface, use the **pppconfig** command or the **IP_SIOCXPPPSETIF ioctl( )** request. See *4.4 pppconfig*, p.36 and *4.3 ioctl( ) Requests*, p.31 for details. For example, the following will bind PPPoE interface **pppoe0** to Ethernet interface **eth1**.

```
[vxWorks *]# pppconfig pppoe0 pppoe-setif eth1
```

5.4 Callback Routine

Wind River PPP requires a callback routine to handle all of the following actions:

IPPPP_ACTION_INIT
Called when a new interface is attached. Use this callback to set configuration parameters for the interface, like the restart timeout or max-failure settings in the **Ipcom_ppp_conf** struct.

IPPPP_ACTION_LCP_STARTING
Called when LCP starts.

IPPPP_ACTION_IPCP_STARTING
Called when IPCP starts.

IPPPP_ACTION_IPV6CP_STARTING
Called when IPV6CP starts.

IPPPP_ACTION_IP_UP
Called when IPV4 is up and ready for traffic.

IPPPP_ACTION_IP_DOWN
Called when IPv4 goes down.

IPPPP_ACTION_IPV6_UP
Called when IPv6 is up and ready for traffic.

IPPPP_ACTION_IPV6_DOWN
Called when IPv6 goes down.

IPPPP_ACTION_PEER_IDENTIFIER
Called when a peer sends an IPv6 Identifier-Configure request. Return 0 to accept the request, -1 to deny the request.

IPPPP_ACTION_LOGIN
Called when authentication (username and password) is required.

To define a callback routine and register it with Wind River PPP, set the Workbench parameters **PPP_INSTALL_CALLBACK_HOOK** and **PPP_ACTION_CALLBACK_HOOK**. When **PPP_INSTALL_CALLBACK_HOOK** is set to **TRUE**, the user-defined callback routine identified by **PPP_ACTION_CALLBACK_HOOK** is registered. If a user-defined callback routine is not added, the default callback routine in **ipppp_example.c**, in the directory *InstallDir***/components/ip_net2-6.***x***/osconfig/vxworks/src/ipnet/**, is used.

For more details, see **ipppp_example.c** and the reference entry for **ipppp_example_action_cb( )**.

5.5 Authentication

When Wind River PPP is running as a PPP or PPPoE server with PAP or CHAP authentication, it uses the username/password combinations stored using the **ipcom_auth** API to authenticate peers. See the routines in the **ipcom_auth** library in the *Wind River Network Stack Kernel API Reference* for details.

The Workbench components **INCLUDE_IPCOM_AUTH_***n* can be used to define username/password combinations at build time. To add and delete users at run-time, call **ipcom_auth_useradd( )** and **ipcom_auth_userdel( )**.

5.6 ifconfig

Many operations on Wind River PPP interfaces are handled by the network command **ifconfig**. See the *Wind River Network Stack for VxWorks 6 Programmer's Guide Volume 3: Interfaces and Drivers* for details about **ifconfig**. Some possible uses are as follows:

To bring an existing interface up:
```
ifconfig interface_name up
```

To shut an interface down:
```
ifconfig interface_name down
```

To create a PPPoE interface:
```
ifconfig interface_name create
```

To detach an interface:
```
ifconfig interface_name detach
```

A
Libraries

ipppp

NAME	**ipppp** – API of PPP module
ROUTINES	**ipppp_example_action_cb( )** – handle PPP actions **ipppp_login( )** – Function used to authenticate (i.e. login) to the peer/sevrer
DESCRIPTION	This library contains the API for IPPPP - PPP module
INCLUDE FILES	none

B

Routines

ipppp_example_action_cb( )

NAME

ipppp_example_action_cb( ) – handle PPP actions

SYNOPSIS

```
IP_PUBLIC int ipppp_example_action_cb
    (
    Ipcom_netif *netif,
    int         action,
    void        *data
    );
```

DESCRIPTION

In order for PPP to function the user must implement this callback function used by the PPP module to handle various PPP actions like initialization, startup, peer IPv4 address requests, peer authentication, login etc.

Parameters:

netif
Pointer to Ipcom_netif interface structure define in **ipcom_netif.h**

action
An action code identifying the basis for calling the callback action routine. The following action codes are available:

IPPPP_ACTION_INIT
The purpose of this callback is to give the PPP system administrator a chance to set various PPP, Driver, LCP and/or Authentication initialization configuration options for this interface. The callback is only called once per interface and immediately after the PPP interface is attached.

IPPPP_ACTION_LCP_STARTING
Called when LCP is starting. Configure session specific LCP parameters here.

IPPPP_ACTION_IPCP_STARTING
Called when IPCP is starting. This action callback can contain code to override the sysvar configurations of local and peer IPv4 address. More precise. the sysvar addresses can be overridden by calling **ioctl( )** with the commands SIOCSIFADDR and/or SIOCSIFDSTADDR. If the default local IP address is 0 then we want the peer to give us an IP address. If the peer address is 0, we refuse to suggest an IP address to the peer.

IPPPP_ACTION_IPV6CP_STARTING
Called when IPV6CP is starting.

IPPPP_ACTION_IP_UP
Called when IPv4 is up and ready for traffic on this PPP link.

IPPPP_ACTION_IP_DOWN
Called when IPv4 goes down on this PPP link.

IPPPP_ACTION_IPV6_UP
Called when IPv6 is up and ready for traffic on this PPP link.

IPPPP_ACTION_IPV6_DOWN
Called when IPv6 goes down on this PPP link.

IPPPP_ACTION_PEER_IDENTIFIER
Called when peer sends IPv6 Identifier Configure-Request. See RFC274:p6-8 for details. Return -1 if refuse to suggest peer IP identifier, else 0. **data** = pointer to 128-bit IPv6 identifier.

IPPPP_ACTION_LOGIN
Called when we need to authenticate ourselves (i.e. login) to the peer. User code must call **ipppp_login( )** with the interface and link index, **data** as the cookie, user name & password and the function call context when ready to login.

data
Action type specific additional data.

EXAMPLE

```
See components/ip_net2-6.5/osconfig/vxworks/src/ipnet/ipppp_example.c for
an example implementation of PPP action callback.
```

RETURNS

0 success, -1 failure.

ERRNO

SEE ALSO **ipppp**

ipppp_login( )

NAME **ipppp_login( )** – Function used to authenticate (i.e. login) to the peer/sevrer

SYNOPSIS

```
IP_PUBLIC int ipppp_login
    (
    int           ifindex,
    int           link_index,
    void          *login_cookie,
    IP_CONST char *name,
    IP_CONST char *pw,
    int           fcflags
    );
```

DESCRIPTION This function is used when PPP acts as a client and is required to authenticate itself (i.e. login) to a PPP server. The **ipppp_login( )** functions should be used after the **IPPPP_ACTION_LOGIN** event has been signaled via the ppp action callback. The function causes the PPP client to login on the remote side with the user name name and password pw. For example, If CHAP is used, this function causes the IPPPP module to send a CHAP response to the peer. The function can either be called directly from the action callback or at any time later (taking normal PPP connection timeouts in consideration).

The first three function arguments should be taken from the action callback as shown in the example below. The fourth and fifth arguments are the username and password. Finally, the sixth argument is of **IP_FLAG_FC_XXX** type (see **ipcom_cstyle.h**) and should be **IP_FLAG_FC_STACKCONTEXT** if called from the action callback context, or 0 if called from any other context.

Parameters:

ifindex>
PPP interface index.

link_index
PPP link index (currently always 0).

login_cookie
IPPPP specific, do not read/write. Pass the same cookie from the action callback to the login function.

name
PPP username, e.g. "bill"

pw
The cleartext password of user **name**.

fcflags
Function call flags from **ipcom_cstyle.h**. Either **IP_FLAG_FC_STACKCONTEXT** or 0.

EXAMPLE

```
See components/ip_net2-6.5/osconfig/vxworks/src/ipnet/ipppp_example.c for
an example implementation of how ipppp_login() is used.
```

RETURNS

0 if arguments are valid, or a negative errno code.

ERRNO

SEE ALSO

ipppp

C
Glossary

C.1 Terms

access concentrator

PPP is a peer-to-peer protocol, but also has client/server aspects. An access concentrator is a PPPoE server that responds to requests for particular services.

address field

1. In a public switched network directory number, the *address field* contains the international telephone directory number.
2. In HDLC-framed packets, the frame begins with an *address field* indicating the frame's destination.

address pool

A remote access server may assign IP addresses to its clients, and if so, it typically selects these from a predefined *address pool*.

"any"

When a PPPoE client transmits requests containing the *"any"* service name, it expects a server to respond with a list of available service names. The client can then send one of these service names in a true connection request.

application layer

In the OSI network layer model, the application layer is the topmost layer—furthest away from the physical connection and closest to the data's origination or destination.

asynchronous

When a process sends a message and then continues processing without waiting for a reply, or calls a function and continues without waiting for the function to complete its processing, these are said to be *asynchronous* messages or *asynchronous* function calls.

authentication

The process of determining that the machine or user on the other end of a connection is authorized to connect, or is who was expected.

authentication phase

If the peers of a PPP connection negotiate an *authentication phase* for the connection, the peers will perform authentication with an authentication protocol.

authenticator

In the process of authentication, the peer that presents the challenge or that inspects the other peer's claimed identity is called the authenticator.

bandwidth

The measure of how much data can be transmitted over a connection.

blocked state

If the underlying data link layer cannot accept any more packets or frames to send, the PPP stack enters a *blocked state* until this condition ends.

broadcast

A network packet or frame is *broadcast* if it is addressed not to a particular address but to all machines on a subnet or network segment.

configuration request

PPP peers negotiate the characteristics of their connection by exchanging *configuration request*s.

discovery packet

In PPP over Ethernet, clients find available servers by broadcasting a *discovery packet* that contains a session request and a service name. A server that supports that service name responds by transmitting a session offer to the client.

fast frame check (FCS)

A checksum used to detect data corruption during transmission.

favored peer option

The *favored peer option* determines which peer is favored in the event of a race condition in which both peers simultaneously transmit the same variety of BAP request.

magic number

LCP sends a randomly-selected *magic number* in its negotiation packets so that a stack can distinguish its own echoed packets from those coming from a peer; see *RFC 1661.*

network layer

The layer above the data link layer; IP is an example of an OSI *network layer* protocol.

network phase

In the state machine that governs a PPP connection, at the end of the link establishment phase, a connection enters the *network phase*.

non-blocking routine

A routine that returns control immediately to the caller, before it has finished doing the tasks for which it was called, is called a *non-blocking routine*.

octet

An *octet* is an eight-bit grouping; usually *octet* is synonymous with *byte* except in the rare contexts in which a byte is not an eight-bit quantity.

operational status

The *operational status* of the IP protocol indicates whether its state machine has or has not reached the "opened" state.

protocol field

The *protocol field* of a PPP packet contains the *protocol identifier* that indicates which protocol a given PPP packet is intended for. For instance, IP is indicated by a protocol identifier 0x0021. Under some circumstances the protocol field can be reduced to a single byte by enabling *protocol field compression*.

protocol identifier

The *protocol identifier* is a two-byte value that indicates which protocol a given PPP packet is intended for. For instance, IP is indicated by a protocol identifier 0x0021. This identifier is found in the *protocol field*, and under some circumstances can be reduced to a single byte, thereby enabling *protocol field compression*.

protocol layer

The transport and network layers in the OSI network stack model are referred to in combination as the *protocol layer*. For instance, TCP is a transport layer implementation and IP is a network layer implementation; TCP/IP is a *protocol layer* implementation.

subscribe

To *subscribe* to an event means to indicate that you want to be informed when an event is raised.

synchronous

When a process sends a message and then waits for a reply before continuing, or calls a function and waits for the function to complete its processing and return before continuing, these are said to be *synchronous* messages or *synchronous* function calls.

timer

A *timer* is set to invoke a callback routine when it expires after a certain amount of time. See also: *timestamp*.

timestamp

A *timestamp* is a passive timer that does nothing when it expires, but can be queried for its expiration status. See also: *timer*.

transport

In the OSI network stack model, the *transport* layer sits above the network layer; TCP is an example of a protocol that implements the transport layer.

WinPoET

WinPoET is a PPPoE client for Windows.

C.2 Abbreviations and Acronyms

Wind River PPP uses the following abbreviations and acronyms in documentation, development tools, code, filenames, and directory names.

Table C-1 **Abbreviations and Acronyms**

Abbreviation	**Description**
ACC	Asynchronous Control Character
ACCM	Asynchronous Control Character Map
ACFC	Address and Control Field Compression
ACK	Acknowledgement
API	Application Programming Interface
ARP	Address Resolution Protocol
bps	bits per second

Table C-1 **Abbreviations and Acronyms** (cont'd)

Abbreviation	Description
CHAP	Challenge-Handshake Authentication Protocol
CHAP-0x80	Microsoft PPP CHAP Extensions
DNS	Domain Name Server
DUN	Dial-up Networking
EID	Endpoint Identifier
END	Enhanced Network Driver
FCS	Fast Frame Check
HDLC	High-level Data Link Control
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force (**http://www.ietf.org**)
IPCP	Internet Protocol Control Protocol
ISDN	Integrated Services Digital Network
LCP	Link Control Protocol
ldpi	link, direction, protocol, identity
MAC	Media Access Control
MRRU	Maximum Receive Reconstructed Unit
MRU	Maximum Receive Unit
MSS	Maximum Segment Size
MTU	Maximum Transmit Unit
NAK	Negative Acknowledgement
NCP	Network Control Protocol
NPT	Network Protocol Toolkit
OSI	Open System Interconnection (standards body)

Table C-1 **Abbreviations and Acronyms** (cont'd)

Abbreviation	Description
PADI	PPPoE Active Discovery Initiation
PADO	PPPoE Active Discovery Offer
PADR	PPPoE Active Discovery Request
PADS	PPPoE Active Discovery Session-confirmation
PADT	PPPoE Active Discovery Termination
PAP	Password Authentication Protocol
PPP	Point-to-Point Protocol
PPPoE	PPP over Ethernet
RADIUS	Remote Authentication Dial-In User Service
RFC	Request For Comments (standards documentation)
RTOS	Real-Time Operating System
SIO	Serial I/O
SNMP	Simple Network Management Protocol
WDB	Wind River Debug

Index