PlayStation®2 IOP Library Overview
Release 2.4

Inet Libraries

Publication date: October 2001

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052, Japan

Sony Computer Entertainment America
919 E. Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.

Summary Table of Contents

About This Manual

This is the Runtime Library Release 2.4 version of the *PlayStation®2 IOP Library Overview - Inet Libraries* manual.

The purpose of this manual is to provide overview-level information about the PlayStation®2 IOP inet libraries. For related descriptions of the PlayStation®2 IOP inet library structures and functions, refer to the *PlayStation®2 IOP Library Reference - Inet Libraries.*

Changes Since Last Release

None

Related Documentation

Library specifications for the EE can be found in the *PlayStation®2 EE Library Reference* manuals and the *PlayStation®2 EE Library Overview* manuals.

Note: the Developer Support Web site posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

Typographic Conventions

Certain Typographic Conventions are used throughout this manual to clarify the meaning of the text:

Convention	Meaning
`courier`	Indicates literal program code.
italic	Indicates names of arguments and structure members (in structure/function definitions only).
medium bold	Indicates data types and structure/function names (in structure/function definitions only).
blue	Indicates a hyperlink.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In North America:	*In North America:*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd. Foster City, CA 94404, U.S.A. Tel: (650) 655-8000	E-mail: PS2_Support@playstation.sony.com Web: http://www.devnet.scea.com/ Developer Support Hotline: (650) 655-5566 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In Europe:	*In Europe:*
Attn: Production Coordinator Sony Computer Entertainment Europe 30 Golden Square London W1F 9LD, U.K. Tel: +44 (0) 20 7859-5000	E-mail: ps2_support@scee.net Web: https://www.ps2-pro.com/ Developer Support Hotline: +44 (0) 20 7859-5777 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT)

Chapter 1:
Network (INET) Configuration Library

Overview

The INETCTL API is an API for writing configuration settings to INETCTL and managing the state of the INET protocol stack. It is implemented as the inetctl.irx resident library and has functions for storing the contents of configuration files read into memory with NETCNF API, reporting to the appropriate INET protocol stack (inet.irx), bringing the network interface up and down, and setting event handlers.

Starting Up inetctl.irx

The inetctl.irx startup options and arguments are shown below.

Syntax

mstart inetctl.irx [*option*] [*NET_CNF*]

options	Option specification (details are given below)
NET_CNF	Configuration file name. This can be omitted when the -no_auto option is specified.

Options

-no_check_magic	The magic line at the beginning of the file is not checked.
-no_decode	The ATTACH_CNF file is not individually encoded.
-no_auto	Non-automatic mode (not automatically brought up when the device is connected).
-verbose	Detailed messages are displayed.
-thpri=*thpri*	Specifies the thread priority of inetctl.irx itself. thpri is specified as a decimal number. If not specified, the default value is 48.

During development, inetctl.irx should be started up as follows.

```
dsidb R> mstart inetctl.irx -no_decode host1:/usr/local/sce/conf/net/net003.cnf
```

When a device is detected, connection processing is performed automatically. Also, since no individual encoding is performed, the file contents can be verified easily.

In a title application, on the other hand, it is undesirable to perform connection processing without asking the user "Which configuration should be used for connecting?" Therefore, inetctl.irx is started up with automatic connection prohibited.

```
dsidb R> mstart netcnf.irx
dsidb R> mstart inetctl.irx -no_auto
```

Configuration Procedure

The procedure for using the INETCTL API to configure INET using the network common configuration is shown below.

1. **Start up the modules**

   First, start up netcnf.irx, then start up inetctl.irx.

   ```
   dsidb R> mstart netcnf.irx
   dsidb R> mstart inetctl.irx -no_auto
   ```

2. **Register the handler**

   Use sceInetCtlRegisterEventHandler() to register the handler for processing interface state changes.

3. **Get the NET_CNF list**

   Use sceNetCnfGetCount() and sceNetCnfGetList() to get a list of entries (NET_CNF) with type=0 from the configuration management file. Since the configuration management file may exist on two memory cards and the hard disk drive, repeatedly execute these functions for devices supported by the title and collect a list for each device.

4. **Present the NET_CNF list**

   Present the NET_CNF list that was acquired for the user and have the user select the configuration to be used for the connection.

   At this time, use sceNetCnfLoadEntry() to load the configuration data into memory for each NET_CNF file in the list, and use sceInetGetInterfaceList() and the control code for getting the vendor name or device name to confirm that the files referenced from the NET_CNF file exist, and if the appropriate device is connected. For entries for which no file exists or no device is connected, show a grayed-out display to indicate that the entries cannot be used.

5. **Configure operation**

   Perform INET configuration by calling sceInetCtlSetConfiguration() with the user-selected configuration entry as the argument. Since -no_auto was specified during startup, network connection processing will not be started at this time, even if the associated device is connected.

6. **Wait for interface Attach report**

   Wait for the network interface connection (Attach) event to be reported to the handler that was registered in step 2.

7. **Request that the interface be brought up**

   Call sceInetCtlUpInterface(id=0) to issue an Up request to inetctl.irx for all interfaces.

   In response to this, the following notification is reported to the handler that was registered in step 2.

   - For an interface for which a device exists but no configuration data corresponding to it exists sceINETCNF_IEV_NoConf is reported followed by sceINETCNF_IEV_Attach. If multiple devices corresponding to the same configuration file had been connected, only the device that was detected first is considered to match the configuration data, and its state is maintained until that device is detached or reconfigured with sceInetCnfSetConfiguration().

- For an interface for which a device exists and the corresponding configuration data also exists sceINETCNF_IEV_Conf is reported followed by sceINETCNF_IEV_Attach.
- For an interface for which no device exists
  No event notification is reported.

Since the interface ID is passed as an argument when the event is reported, this interface ID is used to perform subsequent control.

7. **Reporting by events**

   If the connection to the network is performed normally, a Start event is reported to the handler that was registered in step 2.

Determining Timeouts

After an Up request, a certain amount of time is required until connection processing is performed and a Start event is reported. However, if too much time is taken, it should be assumed that the network connection has failed. Use the criteria shown below to decide the standard timeout interval for determining connection failure.

Table 1-1

Connection Method	Normal Time Required	Timeout Interval
Ethernet connection (fixed address)	Auto-Nego interval with Hub (approximately 3 seconds)	Approximately 6 seconds
Ethernet connection (using DHCP)	Auto-Nego interval with Hub + time required by DHCP protocol (approximately 3.5 seconds)	Approximately 7 seconds
PPP connection	Time required by PPP protocol including modem origination processing, modem Nego time, and remote-side response (although this depends on the modem, line, remote side, and configuration, it is approximately 30 to 60 seconds)	Approximately 60-120 seconds

Chapter 2:
Network Library Overview

Library Overview

The network support library is an IOP library that enables the PlayStation 2 to connect to the network (i.e. the Internet).

TCP and UDP are supported as protocols, and lower-level protocols IP, ARP, and PPP are also supported.

The following three types of network connection equipment are supported by the hardware.

- Dial-up connection using a USB modem
- LAN connection using the hard disk drive
- LAN connection using a USB-Ethernet adapter (for development only)

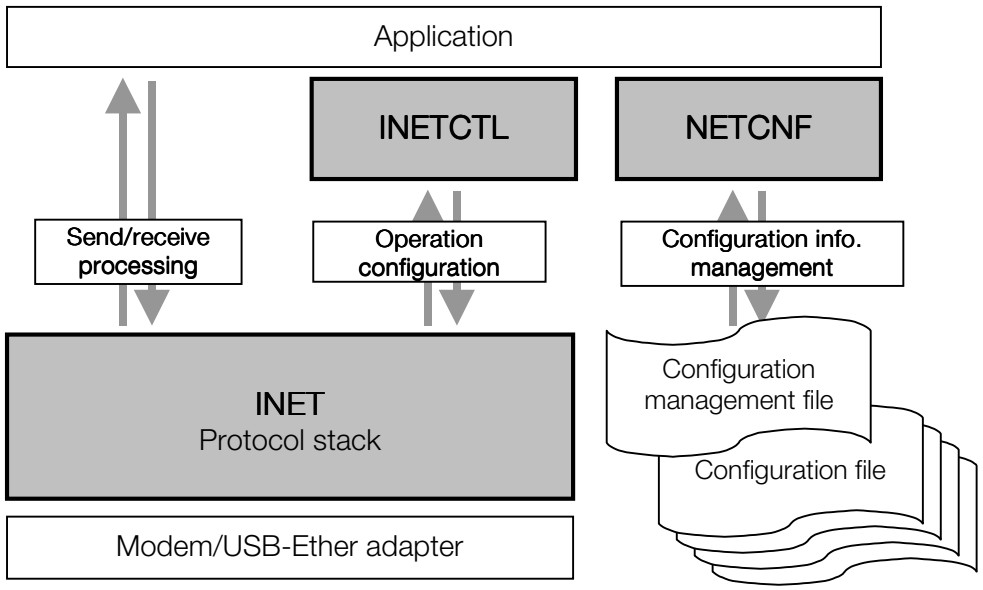
  Note: Although the drivers provided by SCEI are only for development and cannot be used in a title application, there are no restrictions on the use of independently developed drivers.

To connect to the network, the user must individually set information such as the IP address and subnet mask for a LAN connection, or the destination telephone number and user account for a dial-up connection. A configuration file format, a library for configuring software based on the configuration files, and a library for managing and editing the configuration files are provided to enable applications that access the network to share this information.

Software Configuration

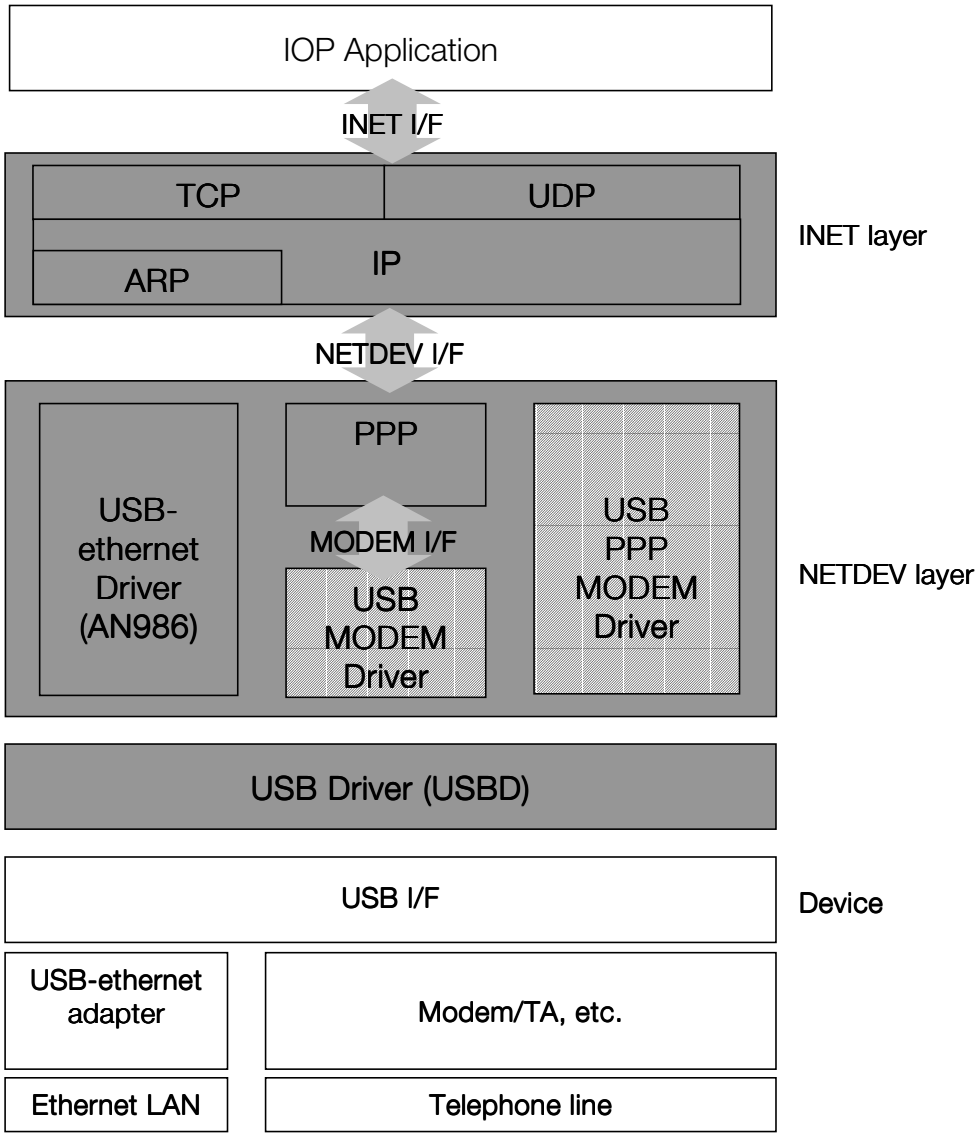
The network support library is divided into three main parts as shown below.

Figure 2-1: Network Support Library Configuration



The following figure shows a hierarchy diagram of the INET protocol stack. The drivers in the portion indicated by the diagonal lines are provided by tool and middleware licensees.

Figure 2-2: Hierarchical Structure of INET Protocol Stack



Supported Equipment

The network support library currently supports the following USB-Ethernet adapters for development purposes only. (Other USB-Ethernet adapters that support 10 and 100 Mbps and use the ADMtek AN986 (Pegasus) chip may be able to be used, but their operation is not guaranteed.)

Table 2-1

Region	Manufacturer	Product Name
Japan	MELCO	LUA-TX
Japan	I-O DATA	USB-ET/TX
Japan	PLANEX	UE-100TX
Japan	Corega	FEther USB-TX
U.S.	Linksys	USB100TX
U.S.	D-Link	DSB-650TX
Europe	ADMtek	Pegasus
Europe	D-Link	DU-E100

For USB modems, the following drivers are provided by the respective manufacturers for each equipment model (check the developer support website for the latest information).

Table 2-2

Region	Manufacturer	Product Type	Product Name
Japan	Sun-denshi	Modem	OnlineStation (56Kbps)
Japan	Sun-denshi	ISDN terminal adapter	TS128JX3
			TS128NS
Japan	Sun-denshi	Mobile phone/ PHS connection cable	PS64P1 (for DDI Pocket)
			PS64P2 (for NTT DoCoMo, ASTEL PHS)
			DS96L (for PDC)
			DS96LS (PDCp)
Japan	Omron	Modem	ME5614USB
Japan	Omron	ISDN terminal adapter	MT128S-D/U2
			MT128S-D/U2(S)

Although more than one device can be handled simultaneously, the number of devices may be limited by the amount of free IOP memory.

Managing Network Configuration Information

Network configuration information that differs for each user, such as the modem model, provider telephone number, and user account used for a dial-up connection, or the IP address and subnet mask used for a LAN connection, is saved on a memory card in either a configuration management file or a configuration file that has a predefined format. A Playstation 2 internal serial number is used for encryption (individually encoded) so that the configuration files cannot be read on another Playstation 2 (encryption can also be disabled during development).

The NETCNF API is used to read and write configuration management files or configuration files. Also, the INETCTL API is used to perform processing that configures the INET protocol stack operation based on the contents of the configuration file that was read into memory.

In a title application, a process is needed that allows the user to create and edit configuration management files or configuration files, and to confirm the start of a connection using these APIs.

Restrictions and Precautions

Currently, the following restrictions and precautions apply.

- When more than one interface is connected simultaneously, the interface names (ethX or pppX) will be eth0, eth1, ... in the order that the interfaces were registered. In the USB specification, when more than one device is connected, since the device that is recognized first is undefined (although the same result usually occurs), the correspondence between interface names and devices will also be undefined. A specific interface name cannot be assigned to a specific device.

Confirming Operation

This section describes the procedure for using a USB-Ethernet adapter to start up the network support library and confirm operation. In the descriptions below, $MODULES represents /usr/local/sce/iop/modules and $CONF represents /usr/local/sce/conf/net. Specifications should be made by replacing each of these with the appropriate directory name.

1. **Connect device**

   Obtain a USB-Ethernet adapter that is supported by the network support library and connect it to the USB port of the DTL-T10000, then connect it to the LAN.

2. **Prepare DHCP server**

   Although a DHCP server is not required, if possible, a DHCP server should be provided on the network since it will simplify the creation of the configuration files.

3. **Create configuration files**

   See the configuration file samples that are located in $CONF and create the configuration files.

   If using DHCP, the net003.cnf, ifc003.cnf, and dev003.cnf can be used directly.

   If not using DHCP, set suitable values for address and netmask in ifc000.cnf.

4. **Start up INET module**

   Move to the tools directory and start up dsidb. Then, after a reset, start up inet.irx.

   ```
   $ cd /usr/local/sce/iop/util/inet
   $ dsidb
   dsidb R> reset
   dsidb R> mstart $MODULES/inet.irx debug=18
   ```

   At this time, no interface exists. More detailed log messages can be output according to the debug argument. The type of log to be output can be changed by changing the two-digit hexadecimal number. For details, see the startup option section described later.

   Next, start up the log module for displaying or recording log information that was left behind by the INET module and its subordinate layers.

   ```
   dsidb R> mstart $MODULES/inetlog.irx host1:inet.log
   ```

   If a filename is specified in the argument as shown here, the log information is recorded in that file. If no filename is specified, the log information is output to the console with printf.

5. **Configure network-related operation**

   Make the configuration file management module resident and read the contents of the configuration management file (netXXX.cnf, XXX is 000~999) into memory. In this example, assume net003.cnf is used.

   ```
   dsidb R> mstart $MODULES/netcnf.irx
   dsidb R> mstart $MODULES/inetctl.irx -no_decode host1:$CONF/
   net003.cnf
   ```

   The contents of the configuration file that are indirectly specified in net003.cnf will also be read.

6. **Start up driver modules**

   a. Making a PPP connection with a modem

      First, start up the PPP module then start up the modem driver module.

      ```
      dsidb R> mstart $MODULES/ppp.irx
      dsidb R> mstart <modem_driver>
      ```

      At this point, the modem driver module performs a registration for the PPP module, and from this, the PPP module registers the PPP network interface for the INET module.

   b. Making a USB Ethernet connection

      First, start up the USB module then start up the AN986 module.

      ```
      dsidb R> mstart $MODULES/usbd.irx
      dsidb R> mstart $MODULES/an986.irx
      ```

      At this point, the AN986 module registers the network interface for the INET module. (If the USB-Ethernet adapter is not connected at this time, this operation will be performed when it is connected).

   c. Making a PPPoE(PPP over Ethernet) connection over USB Ethernet

      First, start up he USBD module then start up the AN986 module.

      ```
      dsidb R> mstart $MODULES/usbd.irx
      dsidb R> mstart $MODULES/an986.irx
      ```

      At this point, the AN986 module registers the Ethernet network interface for the INET module. (If the USB-Ethernet adapter is not connected at this time, this operation will be performed when it is connected). Then, to register the PPPoE interface, first start up the PPP module then start up the PPPoE module.

      ```
      dsidb R> mstart $MODULES/ppp.irx
      dsidb R> mstart $MODULES/pppoe.irx
      ```

      For the PPP module, the PPPoE module behaves like a normal modem. When the PPPoE module is loaded, registration is performed for the PPP module and from this, the PPP module registers the PPP network interface for the INET module.

   d. Making an HDD Ethernet connection

      First, start up the DEV9 module then start up the SMAP module.

      ```
      dsidb R> mstart $MODULES/dev9.irx
      dsidb R> mstart $MODULES/smap.irx
      ```

At this point, the SMAP module registers the network interface for the INET module.

e. Making a PPPoE(PPP over Ethernet) connection over HDD Ethernet

First, start up the DEV9 module then start up the SMAP module.

dsidb R> mstart $MODULES/dev9.irx

dsidb R> mstart $MODULES/smap.irx

At this point, the SMAP module registers the network interface for the INET module. Then, to register the PPPoE interface, first start up the PPP module then start up the PPPoE module.

dsidb R> mstart $MODULES/ppp.irx

dsidb R> mstart $MODULES/pppoe.irx

For the PPP module, the PPPoE module behaves like a normal modem. When the PPPoE module is loaded, registration is performed for the PPP module and from this, the PPP module registers the PPP network interface for the INET module.

In the above descriptions, if the network interface is registered in the INET module, then the INET module reports to the INETCTL module that "an interface has changed state (in this case, a device was attached)". INETCTL uses the INET Control API to configure INET operation using the configuration information that was read.

7. **Confirm interface state with ifconfig**

Execute ifconfig to confirm the interface state.

dsidb R> mstart $MODULES/ifconfig.irx

An example display when processing is normal is shown below.

```
eth0    Module:an986  Prot:0  Impl:0  Loc:USB-1
        Vendor:Melco  Device:LUA-TX  HWaddr:00:40:26:61:31:F9
        inet addr:192.168.0.16  Mask:255.255.255.0
        UP RUNNING ARP DHCP  MTU:1500
        RX packets:3 bytes:746 errros:0 dropped:0
        TX packets:4 bytes:764 errros:0 dropped:0
```

8. **Confirm routing information with route**

Execute route to confirm routing information.

dsidb R> mstart $MODULES/route.irx

An example display when processing is normal is shown below.

```
Destination   Gateway       Genmask       Flags  MSS    Window Iface
192.168.0.0   *      255.255.255.0 U      0      0      eth0
default       192.168.0.1   *      UG     0      0      eth0
```

When no default Gateway has been configured, verify the DHCP server-side configuration.

9. **Confirm name server configuration with nsconfig**

Execute nsconfig to confirm the name server configuration.

dsidb R> mstart $MODULES/nsconfig.irx

A sample display when processing is normal is shown below.

```
nameserver 192.168.0.2
```

When "nsconfig: No name server" is displayed, confirm the DHCP server-side configuration.

inet.irx Startup Options

The INET module inet.irx has the following startup options.

mem=*size*

Specifies the size of the memory pool to be allocated by inet.irx.

Specify a decimal number for *size*. "KB" or "MB" can also be specified at the end of the size specification. For example, "mem=256KB" can be specified. If this option is not specified, the default value is 128KB.

thpri=*thpri*

Specifies the priority of the thread that inet.irx is to create for each interface. Specify a decimal number for *thpri*. If this option is not specified, the default value is 48.

debug=*flags*

Specifies the type of log information that will be recorded by inet.irx.

For *flags*, specify a hexadecimal number that is the logical OR of the following bit flags.

Table 2-3

Bit	Meaning
bit0 (0x01)	Record error displays
bit1 (0x02)	Also record address part when displaying packets
bit2 (0x04)	Record DNS-related log
bit3 (0x08)	Record send packets
bit4 (0x10)	Record receive packets
bit5 (0x20)	Record BOOTP- and DHCP-related log
bit6 (0x40)	Enable memory-related debug functions
bit7 (0x80)	Perform byte dump of send/receive packets

If no value is specified, the default value is debug=00 and nothing will be recorded.

Recorded log information can be displayed on the console or transferred to a host1: file as specified when inetlog starts up.

ppp.irx Startup Options

The PPP module ppp.irx has the following startup options.

thpri=*thpri*

Specifies the priority order for each of the threads that ppp.irx creates for each interface. *thpri* is specified as a decimal number. If this option is not specified, the default value is 68.

debug=*flags*

Specifies the type of log information that will be recorded by ppp.irx.

For *flags*, specify a hexadecimal number that is the logical OR of the following bit flags.

Table 2-4

Bit	Meaning
bit0(0x01)	Record the PPP state
bit1(0x02)	Record the LCP and IPCP states
bit2(0x04)	Record the PAP and CHAP states
bit3(0x08)	Record chat processing and replies
bit4(0x10)	Record private information, etc. during authentication
bit5(0x20)	Record data exchange at the DLL level
bit6(0x40)	Perform packet dump when DLL data is displayed
bit7(0x10000)	Record timer information internal to PPP
bit7(0x20000)	Record PPP events

If no value is specified, the default value is debug=0f. These settings may be changed after startup by the high-level configuration control module.

Recorded log information can be displayed on the console or transferred to a host1: file as specified when inetlog starts up.

INET API Overview

The INET API supports TCP and UDP as protocols and also enables lower-layer IP packets to be directly handled by data send and receive services that the network support library provides for title applications.

Communication reliability is guaranteed for TCP, which is a protocol that provides error detection, resend control, and flow control. However, the data contents and arrival are not guaranteed for UDP, which is a simpler protocol with low overhead.

Connection

The meaning of a "connection" with the communication destination differs for TCP and UDP, however, these are combined in the INET API and referred to as simply a Connection. The process of communication using the INET API starts with specifying information such as the communication destination and protocol necessary to create a Connection. Then, the Connection is opened, data is transmitted and received, and finally, the Connection is closed.

Connections are distinguished from each other according to the Connection ID that is assigned when each connection is created.

Although there is essentially no concept of a "connection" in the IP layer, the expression "Connection" is used for convenience when the INET API directly handles IP packets (Raw IP type).

Timeouts

All INET API functions are synchronous I/O types. Control does not return to the calling source until processing ends or until processing is explicitly aborted. Since sleeping occurs within a function if a wait state occurs during send/receive processing, send/receive processing must be performed by creating at least one thread for each Connection.

A timeout interval can be specified with an argument for a function for which a wait state may occur. If the specified timeout interval elapses, a timeout error occurs and sceINETE_TIMEOUT is returned as the return value. If a negative value is specified as the timeout interval, it is treated as an infinitely large value, and the wait state continues while sleeping, until processing is terminated. Also, if 0 is specified, it is treated as if no

wait interval were specified. If a send/receive is repeated with no wait interval, control will no longer be passed to other threads. Therefore, processing must not be performed so that a send/receive is repeated with no wait interval as shown in the example below.

```
while(1){
    flags = 0;
    if(0 > (r = sceInetRecv(cid, buf, sizeof(buf), &flags, 0)))
        Error processing;
    else
        Receive processing;
}
```

Multithread Support

INET API functions can be called simultaneously from multiple threads. When more than one thread performs the same operation (send or receive) simultaneously for a given Connection, processing is queued and performed in thread priority order.

INET API functions cannot be called from thread-independent sections.

Otherwise, a KE_ILLEGAL_CONTEXT error will occur.

Precautions

Power OFF processing

When an HDD Ethernet is used and the PlayStation 2 power is turned OFF, processing must be performed as shown below using functions of the CD(DVD)-ROM library. This is necessary to maintain compatibility with hard disk drives (Expansion Bay type drives).

1. Detect interrupt processing using sceCdPOffCallback().
2. Use the DDIOC_OFF devctl command of the dev9 library to turn the power OFF.
   Example for EE: sceDevctl ("dev9x:",DDIOC_OFF, NULL, 0, NULL, 0);
   Example for IOP: devctl ("dev9x:", DDIOC_OFF, NULL, 0, NULL, 0);
3. Turn OFF the PlayStation 2 power using sceCdPowerOff().

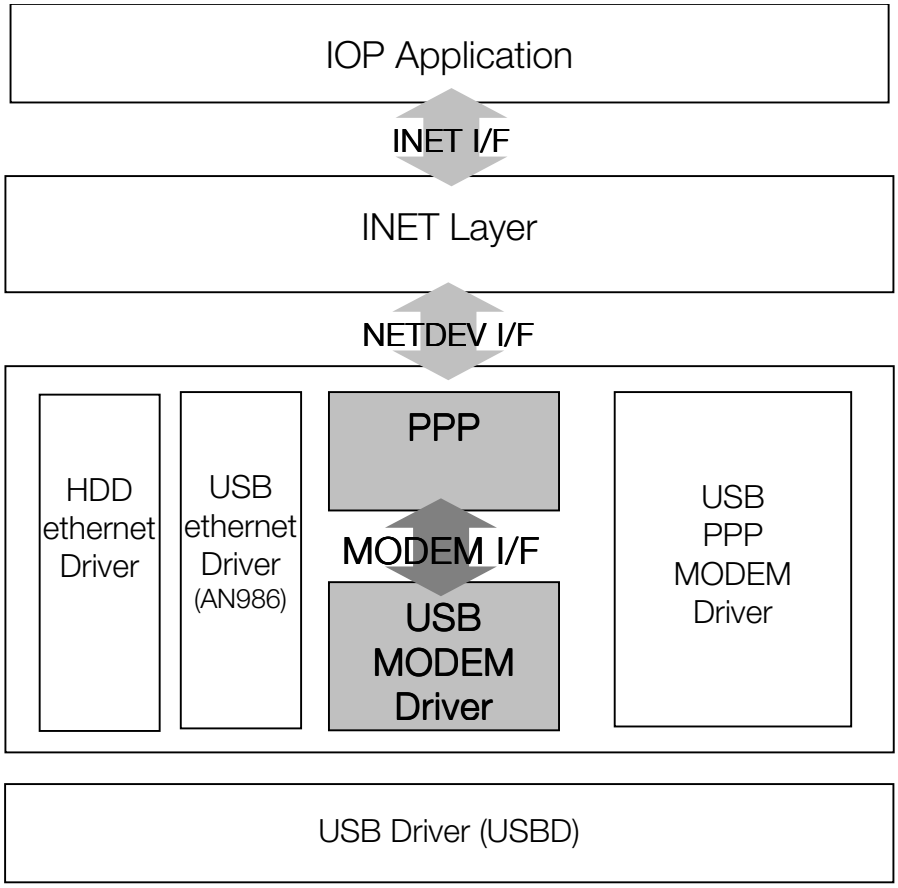
In addition, interrupts can also be detected using sceCdPOffCallback() for Expansion Bay type drives only. For more information, refer to the network sample (ee(iop)/sample/inet), hard disk library sample (ee/sample/hdd/basic), the CD(DVD)-ROM library reference (cdvd_rf) and the dev9 library reference.

Chapter 3:
Modem Driver Development Library

Overview

The PlayStation 2 USB MODEM interface specifications define the interface specifications with the PPP layer and modem driver layer in the network library (INET), and the functions that the modem driver layer must have.

Figure 3-1



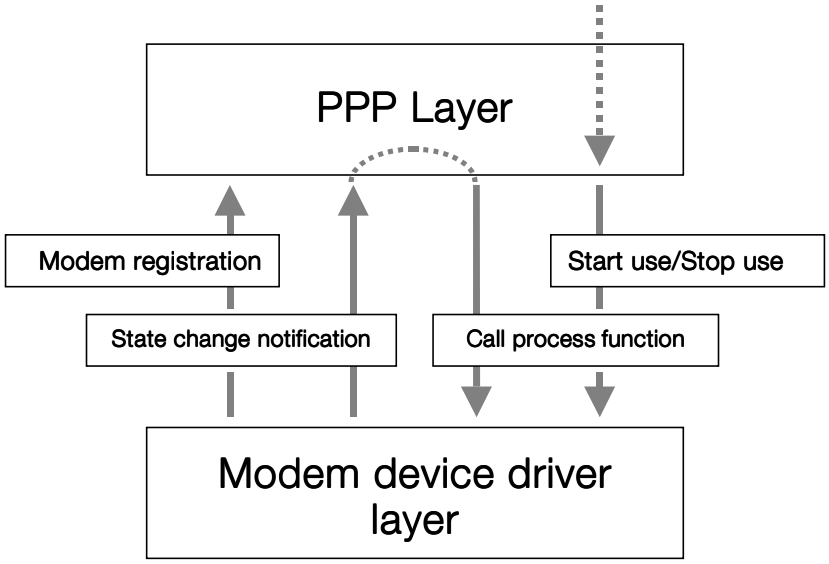
Operations that the PPP layer performs for the modem via the modem driver layer are described in the PPP layer specifications and are not covered in this document. For example, modem calling command strings are not covered here. Even if the processing of those commands is actually implemented by the modem driver layer, they are considered to be modem functions and are not covered here.

PPP Layer and Modem Driver Layer

Exchanges Between the PPP Layer and Modem Driver Layer

The following figure summarizes exchanges between the PPP layer and modem driver layer.

Figure 3-2



The modem driver layer performs the following operations for the PPP layer.

- Registers a connected modem and registers the five functions for starting modem use, ending modem use, sending, receiving, and control.
- Uses event flags to report modem and modem driver layer state changes.

The PPP layer performs the following operations for the modem driver layer.

- Calls a start use or stop use function according to instructions from a layer higher than the PPP layer.
- Calls a send, receive, or stop use function according to a notification by an event flag from the modem driver layer.

When the start use function is called from the PPP layer, the modem state will become "in-use." When the stop use function is called from the PPP layer, the modem state will become "not-in-use."

The data handled by the send and receive functions of the modem driver layer are 8-bit byte strings (one string for each send or receive for a single modem), and commands and responses for the modem are assumed to follow the same route as the data. That is, a similar data exchange model is assumed as the one using the connection format of an AT modem that is serially connected to a general computer. A modem that exchanges data using a different format cannot be handled according to these specifications.

Modem and Processing Function Registration

When the modem driver layer has confirmed a connection with a new modem, it registers modem information and processing functions for the PPP layer. Also, when the connection with the modem is broken, the registration is deleted.

When the modem driver layer confirms the modem connection, it allocates memory for the modem information structure, sets the required fields, and calls the following function with a pointer to that structure as an argument (this function is implemented in the PPP layer):

```
int sceModemRegisterDevice(struct sceModemOps *ops);
```

The memory area for storing this structure must be maintained until the registration is deleted.

Modem Information Structure

The modem information structure is defined as follows.

```
struct sceModemOps {
       struct sceModemOps *forw, *back;        // Higher layer links
       char *module_name                       // Module name
       char *vendor_name;                      // (Device's) vendor name
       char *device_name;                      // (Device's) device name
       u_char bus_type;                        // Bus type
       u_char bus_loc[31];                     // Location on bus
       u_short prot_ver;                       // Protocol version
       u_short impl_ver;                       // Implementation version
       void *priv;                             // Modem layer pointer
       int evfid;                              // Event flag ID
       int rcv_len;                            // Receivable byte count
       int snd_len;                            // Sendable byte count
       int (*start)(void *priv, int flags);          // Start use function
       int (*stop)(void *priv, int flags);           // Stop use function
       int (*recv)(void *priv, void *ptr, int len);  // Receive function
       int (*send)(void *priv, void *ptr, int len);  // Send function
       int (*control)(void *priv, int code, void *ptr, int len);  // Control
function
       void *reserved[4];                      // Reserved area
};
```

The higher layer links (forw, back) are fields that are used by the PPP layer side. Specify NULL for both of them during registration.

For the module name (module_name), specify the modem driver layer module name. Use the string obtained by eliminating ".irx" from the execution filename.

The vendor name (vendor_name) and device name (device_name) are the vendor name and device name of the device to be handled by that modem driver. Even when a virtual device is to be handled, some string should be set.

The following conditions apply for the module_name, vendor_name, and device_name.

- They must not be NULL.
- They must be strings that end with NUL ('\0').
- They must not be strings that contain commas (,) or equal signs (=).
- They must not be empty strings ("").

The string length is not specifically limited.

For the bus type (bus_type), set any of the following values during registration.

Table 3-1

Constant	Value	Meaning
sceModemBus_Unknown	0	Unknown (his setting is not recommended)
sceModemBus_USB	1	USB device
sceModemBus_1394	2	(Reserved)
sceModemBus_PCMCIA	3	(Reserved)
sceModemBus_PSEUDO	4	Pseudo device

The bus location information (bus_loc) is defined in the current specifications only for a USB device.

During registration, store the 7 bytes that are obtained with the sceUsbdGetDeviceLocation() function starting at the beginning of bus_loc.

The PPP layer manages the configuration information of each device based on the combination of the module_name, vendor_name, and device_name and the bus_type and bus_loc.

bus_type and bus_loc are used for identification when multiple devices having exactly the same vendor_name and device_name are connected at the same time. Set these values even when the modem layer does not support multiple devices.

For the protocol version (prot_ver), which is a field that was provided for a future extension of the MODEM interface specifications, the version of the MODEM interface specifications that is assumed by the modem driver layer must be set. In the current specifications, 0 should be set.

For the implementation version (impl_ver), set the modem layer implementation version for each protocol version. Setting a sequence number that starts at 0 and is incremented each time the modem layer implementation changes is recommended.

The modem layer pointer (priv) is a pointer to the data structure that the modem driver layer is to use for each modem device. The value of this field is passed in the argument priv for the start use, stop use, send, and receive functions. However, the contents and use of priv are freely determined by the modem driver layer side, and the PPP layer has nothing to do with this value.

The event flag ID (evfid), which is a field that is set by the PPP layer, stores the ID of the event flag that was generated by the PPP layer. The modem driver layer references this field and reports state changes to the PPP layer.

The receivable byte count (rcv_len) and sendable byte count (snd_len) should both be initialized to 0 beforehand.

The reserved area (reserved) is an area that is used by the PPP layer. This area need not be initialized, and its contents must not be referenced or changed.

Registration Deletion

When the connection with the modem is broken, the modem driver layer deletes the registration by calling the following function, which is implemented in the PPP layer:

```
int sceModemUnregisterDevice(struct sceModemOps *ops);
```

A pointer to the same modem information structure that was specified during registration is specified for the argument.

The registration can be deleted only when the modem state is not-in-use. If the connection with the modem is broken when the modem state is in-use, an event flag must be used to report the state change to the PPP layer, and the registration must be deleted by the stop use function that is called as a result.

If the connection with the modem was broken when the modem had been registered and its state was not-in-use, it is not necessary to notify the PPP layer, and the modem registration can be deleted immediately by calling sceModemUnregisterDevice().

Modem State Change Notification

The modem driver layer uses event flags to report the following events to the PPP layer.

Table 3-2

Constant	Value	Meaning
sceModemEFP_StartDone	0x00000001	Start use processing completed
sceModemEFP_PlugOut	0x00000002	Modem disconnected
sceModemEFP_Connect	0x00000010	Line connected
sceModemEFP_Disconnect	0x00000020	Line disconnected
sceModemEFP_Ring	0x00000040	Incoming call
sceModemEFP_Recv	0x00000100	Receivable byte count increased
sceModemEFP_Send	0x00000200	Sendable byte count increased
sceModemEFP_UpperUse	0xffff0000	(Used by PPP layer)

sceModemEFP_StartDone is a notification that processing of the start use function has completed. The PPP layer waits for this notification before starting send/receive processing.

sceModemEFP_PlugOut is a notification that a state in which the modem driver layer and modem cannot exchange data has occurred for some reason such as the modem was unplugged from the connector.

sceModemEFP_Connect is a notification that the line was connected.

sceModemEFP_Disconnect is a notification that the line was disconnected.

sceModemEFP_Ring is a notification that there is an incoming call from an external source.

sceModemEFP_Recv is a notification that the receivable data count increased (that is, that the number of data that were received from the modem device and stored in the recieve buffer in the modem driver layer increased). The modem driver layer must report this notification after rcv_len in the modem information structure is updated.

sceModemEFP_Send is a notification that the sendable data count increased (that is, that the free space in the send buffer in the modem driver layer increased). The modem driver layer must report this notification after snd_len in the modem information structure is updated.

The modem layer should report these events to the PPP layer during the interval from when the start use function is called until the stop use function is called.

sceModemEFP_UpperUse is a bit that is used by the PPP layer. The modem layer must not reference or set this bit.

Modem Driver Layer Functions

The following five functions must be implemented in the modem driver layer.

- Start use function int start(void *priv, int flags);
- Stop use function int stop(void *priv, int flags);
- Receive function int recv(void *priv, void *ptr, int len);
- Send function int send(void *priv, void *ptr, int len);

- Control function int control(void *priv, int code, void *ptr int len);

For details of each function, refer to the corresponding reference.

Each function must return 0 or a positive value when the function terminates normally and a negative value error code when an error occurs. Although the error codes are used for debugging displays, since the values themselves do not affect PPP layer operations, the values and meanings can be independently determined by the modem driver layer. However, the values that can be used as error codes for the control function (control) are limited to the range from -600 to -649.

Within each of the functions other than the stop use function (stop), processing that waits for another event cannot be performed. Make sure that control always returns in a short time to the calling source. This restriction makes it possible to interrupt start use processing or to perform full-duplex communication.

In response to the registration of one modem device, the PPP layer creates one thread for controlling that device, and each of the functions other than the control function is called from that thread. Therefore, only one function is called at a time with respect to a given connected modem device. However, if multiple modem devices are connected at the same time, multiple functions can be called at the same time, with each function targeting a separate modem. The creator of the modem driver layer is responsible for deciding whether or not multiple modems will be supported simultaneously. However, the decision to support multiple modems simultaneously must be made carefully.

The control function is called while the corresponding device is registered, regardless of the modem layer state.

When each function is called, no special processing is performed at the PPP side in relation to the $gp register. If the $gp register is to be used, make sure that holding, setting, and return processing are performed by the called function.

Modem Driver Layer Common Control Codes

The control function (control) performs various kinds of processing according to the control code specified in the argument. The following control codes are defined in these specifications as modem driver layer common control codes. The implementation of processing for these control codes in the modem driver layer is "strongly recommended."

Table 3-3

Control Code	Value	Function
sceModemCC_GET_THPRI	0xc0000000	Get thread priority
sceModemCC_SET_THPRI	0xc1000000	Set thread priority
sceModemCC_GET_IF_TYPE	0xc0000100	Get device type
sceModemCC_GET_DIALCONF	0xc0000200	Get dialing definition file path name
sceModemCC_FLUSH_RXBUF	0xc0000110	Clear receive buffer
sceModemCC_FLUSH_TXBUF	0xc0000111	Clear transmit buffer
sceModemCC_GET_RX_COUNT	0xc0010000	Get receive data size
sceModemCC_GET_TX_COUNT	0xc0010001	Get transmit data size

Among these control codes, the setting type codes (sceModemCC_SET_XXX and sceModemCC_FLUSH_XXX) can be used only when the modem layer is directly controlled from an application. In a normal application via PPP, use of the setting type codes is prohibited because they are incompatible with the inetctl.irx control library.

Modem Driver Layer Command Control codes (for Serial Devices)

The following control codes are defined as common control codes for modem drivers for which the device type is a serial device, that is, for modem drivers that return sceModemIFT_SERIAL in response to the sceModemCC_GET_IF_TYPE control code.

Table 3-4

Control Code	Value	Function
sceModemCC_GET_OE_COUNT	0xc0010002	Get overrun error count
sceModemCC_GET_PE_COUNT	0xc0010003	Get parity error count
sceModemCC_GET_FE_COUNT	0xc0010004	Get framing error count
sceModemCC_GET_BO_COUNT	0xc0010005	Get buffer overflow count
sceModemCC_GET_PARAM	0xc0020000	Get serial communication parameter
sceModemCC_SET_PARAM	0xc1020000	Set serial communication parameter
sceModemCC_GET_LINE	0xc0030000	Get state of each signal line
sceModemCC_SET_LINE	0xc1030000	Control each signal line
sceModemCC_SET_BREAK	0xc1040000	Send break

Among these control codes, the setting type codes (sceModemCC_SET_XXX) can be used only when the modem layer is directly controlled from an application. In a normal application via PPP, use of the setting type codes is prohibited because they are incompatible with the inetctl.irx control library.

Reference Driver

The reference driver rsaq.irx is being released for the development of modem drivers.

rsaq.irx is to be used as a reference when developing a modem driver for the USB RS-232C adapter, USB-RSAQ and USB-RSAQ2, which is manufactured by I-O Data Device, Inc. This section explains the procedure for verifying operation for a PPP connection using USB-RSAQ or USB-RSAQ2 and an analog modem.

rsaq.irx is provided only for use as a comparison reference when developing a modem driver. Any conversion for use in a title application is strictly forbidden.

Operation Verification Procedure

1. Configure modem

Connect the analog modem that is to be used for verifying operation to the PC and set the modem-side configuration registers as follows.

- Command echo back — Yes
- Connect display on connection — Yes
- Flow control — Enable only RTS/CTS control
- Baud rate setting — Automatic or 115.2Kbps
- Transition from DTR off to on — Initialize modem
- Transition from DTR on to off — Disconnect line and return to command mode

- DSR — On when initialization ends and commands can be accepted
- DCD — On when line is connected and off when it is disconnected

Operation can also be verified by a direct link with Linux or another operating system using RS-232C without connecting the modem. In this case, the peer-side settings should be 115.2Kbps, 1 stop bit, 8 bits/char, and rts/cts flow control.

2. Create configuration files

Look at conf/*.cnf and create configuration files as shown below. If a general-purpose AT modem is to be used, you should be able to connect to many ISPs by just changing the comment lines.

net001.cnf:

```
# <NETCNF>

interface "ifc001.cnf + dev001.cnf" "ifc001.cnf" "dev001.cnf"
```

ifc001.cnf:

```
# <NETCNF>

type ppp
-dhcp
auth_name "XXXXXXXX"         # User name specified from ISP
auth_key "XXXXXXXX"          # Password specified from ISP
peer_name "*"                # Peer-side authentication name for CHAP
allow.auth chap/pap          # Authentication method used by ISP
phone_number "XX-XXXX-XXXX"  # ISP telephone number

route add -net 0.0.0.0 netmask 0.0.0.0
```

dev001.cnf:

```
# <NETCNF>

type ppp
-vendor
-product
-location
dialing_type tone
idle_timeout 600
```

dial001.cnf:

(For a sample dial001.cnf, see the NET configuration file.)

3. Create autoload cnf file

Create the host1:usbmload.cnf file in advance as follows.

```
DeviceName "I/O-DATA USB-RSAQ"
      Use             1
      Category        Modem
      DriverPath      "host1:rsaq.irx"
      DriverArg       "dial=host1:dial001.cnf"
end
```

4. Start up module

Connect USB-RSAQ or USB-RSAQ2 and the modem to the DTL-T10000, connect the modem to the telephone line, and start up the modules required for the PPP connection as follows. (Change entries such as the path names as needed.)

```
dsidb R> reset
dsidb R> mstart modules/inet.irx debug=1f
dsidb R> mstart util/inetlog.irx host1:inet.log
dsidb R> mstart modules/netcnf.irx
dsidb R> mstart modules/inetctl.irx -verbose -no_decode host1:net001.cnf
dsidb R> mstart modules/ppp.irx
dsidb R> mstart modules/usbd.irx
dsidb R> mstart modules/usbmload.irx host1:usbmload.cnf
```

If the above operations are performed and connection processing is completed normally, the following will be displayed.

```
netcnf: (ID=X) [ 192.168.0.1 ]
```

If dsidb R> mstart util/ifconfig.irx is executed here and a display like the following sample display appears, the connection is normal.

```
ppp0      Module:ppp,rsaq  Prot:2  Impl:0  Loc:USB-2
          Vendor:I/O Data  Device:USB-RSAQ
          inet addr:192.168.0.1  Mask:255.255.255.0
          UP RUNNING PPP  MTU:1500  Status:"PHASE Idle"
          RX packets:6 bytes:370 errors:1 dropped:0
          TX packets:6 bytes:293 errors:0 dropped:0
```

5. Disconnect line

To disconnect the line without waiting for an automatic disconnection, perform the following operation.

```
dsidb R> mstart modules/ifconfig.irx ppp0 down
```

Depending on the register settings within the modem, the line may not be disconnected even if this operation is performed. In this case, turn off the modem power.

Action Taken if a Connection Cannot be Established

If a connection cannot be established normally, check the status by referring to the inet.log file and change the configuration files accordingly.

For example, if a timeout occurs even though the negotiation with the modem has not been completed yet, try increasing the timeout interval that is specified using the chat_int string within dial001.cnf. Also, changing the type of log to be saved by changing the value of the debug argument shown above may be effective. For details, see the network library overview.

Cautions

Timing of Notification of Increase in Sendable or Receivable Byte Count

The modem driver layer need not immediately notify the PPP layer when the receivable byte count or sendable byte count is increased. Since there is a risk of increasing the load on the entire system due to

the switching between processing for the PPP layer and modem driver layer if these notifications are reported too frequently, make sure that the notifications are reported only after waiting until the size has increased by an appropriate amount. However, if the notifications are reported too infrequently, the send/receive delay time may become too large.

When the modem driver layer can be concerned with the data contents, for example, when AT commands and response data are flowing, make sure that the notifications are actively reported at the breaks between that data.

If the modem driver layer cannot be concerned with the data contents, notification using the following criteria is recommended.

- When sending, if the modem driver layer has prepared a 1K-byte send buffer, for example, report the notification when the sendable byte count exceeds 3/4 of the entire send buffer.
- When receiving, if the modem driver layer has prepared a 1K-byte receive buffer, for example, report the notification when the receivable byte count exceeds 1/4 of the entire receive buffer or when no new data arrives after a 10ms interval.

To minimize delay when receiving, the PPP layer may call the receive function even if there has been no sceModemEFP_Recv notification.

Sendable and Receivable Byte Count Updates

The rcv_len and snd_len members within the modem information structure are updated by modem driver layer code according to the state of the send/receive buffer within the modem driver layer. However, exclusive control between threads is required in this case.

Normally, rcv_len or snd_len are increased within a callback function from usbd, that is, by a USBD thread. On the other hand, rcv_len or snd_len are decreased by a receive or send function that was called by the PPP layer, that is, by a PPP thread. Regardless of whether this value is increased or decreased, two steps are required for temporarily reading the contents of variables in memory into registers and performing calculations and writing the contents back into memory. Since the data may become inconsistent if these two steps are alternately executed by multiple threads, exclusive control is required.

There are several methods of performing exclusive control. However, in this case, since processing is performed for a very short interval, a method that uses CpuSuspendIntr() and CpuResumeIntr() to prohibit interrupts and thread switching as shown below is suitable.

```
int old;

CpuSuspendIntr(&old);
p->rcv_len += nrcv;
CpuResumeIntr(old);
```

USB Autoload Support

The USB modem driver module must support USB autoloading.

To support autoloading, the pathname of the dialing definition file must be passed when the modem driver module is started up. Therefore, pass this pathname with an argument having the following format.

```
dial=pathname
```

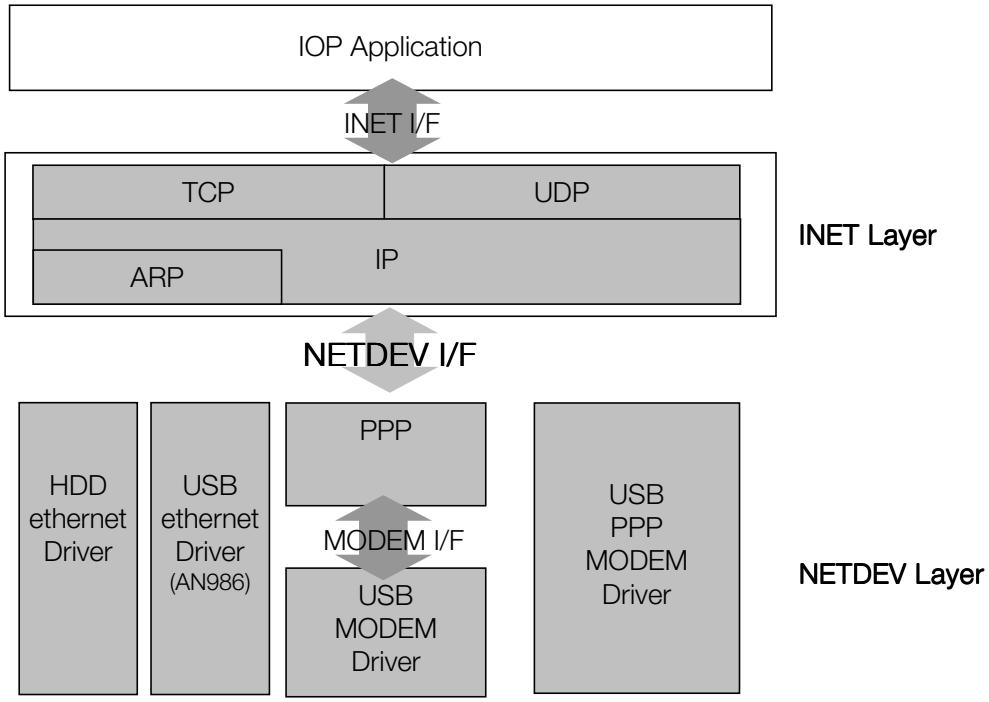
In addition, the "lmode=mode" argument must also be processed. For details, refer to the "USB Driver Autoloader Specifications," which is a separate document.

Chapter 4:
Network Device Library

Overview

The PlayStation 2 network device specifications define the interface between the IP layer and NETDEV layer in the network support library (INET).

Figure 4-1



The NETDEV layer is divided into the following three types according to the network connection medium.

1. For an Ethernet connection — Ethernet driver
2. For a PPP connection — PPP layer
3. Other — Layer that directly handles IP packets in a similar manner as the PPP layer

The data that is exchanged between the IP layer and NETDEV layer consists of Ethernet frames for case 1 above or IP packets for cases 2 and 3 above.

Exchanges With the INET Layer

sceInetDevOps_t

The NETDEV layer module is recognized as a logical network interface by registering each sceInetDevOps_t structure in the INET layer via sceInetRegisterNetDevice(). The following information is contained in the sceInetDevOps_t structure.

- Link information (information for management by the INET layer)
- Name
- Execution file (IOP module) name
- Vendor name and product name of target network device
- Version information
- Pointer to private data
- Flags and event flag ID (for information exchange with INET layer)
- Pointer to send/receive packet queue
- Pointers to start, stop, xmit, and control functions
- IP address and other network information (mainly information used by INET layer)

Start Use and Stop Use Functions

The start use function (start) is called from the INET layer before a registered network interface is used.

When processing such as device initialization is performed by the start use function and 0 is returned, the state of that network interface is considered to be "available" (once initialization processing has started, 0 can be returned even if that processing has not completed).

When the use of the network interface can be stopped, the stop use function (stop) is called. If the device is detached and a notification is reported to the INET layer from the NETDEV layer, the stop use function is called after which the state of that network interface will change to "unavailable."

Notification of State Change by an Event Flag

When preparations for sending are completed, data is received, or a communication error occurs, those events are reported from the NETDEV layer to the INET layer by setting event flags.

To set an event flag, use SetEventFlag() or iSetEventFlag() from the kernel API.

For the first argument of SetEventFlag, specify the value that was set in the evfid field by the INET layer side when the sceInetDevOps_t structure was registered. For the second argument, specify any of the following values according to the circumstances to be reported.

Table 4-1

Constant	Value	Meaning
sceInetDevEFP_StartDone	0x00000001	Notification that start processing has completed for the start (start use) function. The INET layer waits for this notification before starting send/receive processing.
sceInetDevEFP_PlugOut	0x00000002	Notification that data cannot be exchanged between the NETDEV layer and the device because the device was unplugged from the connector.
sceInetDevEFP_Recv	0x00000004	Notification that the received packet was added to the receive packet queue.
sceInetDevEFP_Error	0x00000010	Notification that an error (excluding a timeout) occurred for which send/receive could not continue for a reason other than a PlugOut.
sceInetDevEFP_TimeOut	0x00000020	Notification that a timeout error occurred for which send/receive could not continue for a reason other than a PlugOut.

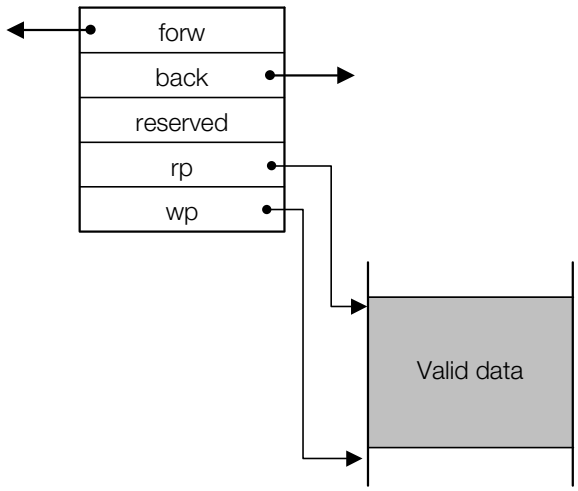
Packets and Packet Queues

Packet and Packet Queue Structure

Send/receive data are handled as packets between the INET layer and NETDEV layer, and multiple packets are consolidated and buffered in two packet queues (one for send and one for receive) within the NETDEV layer.

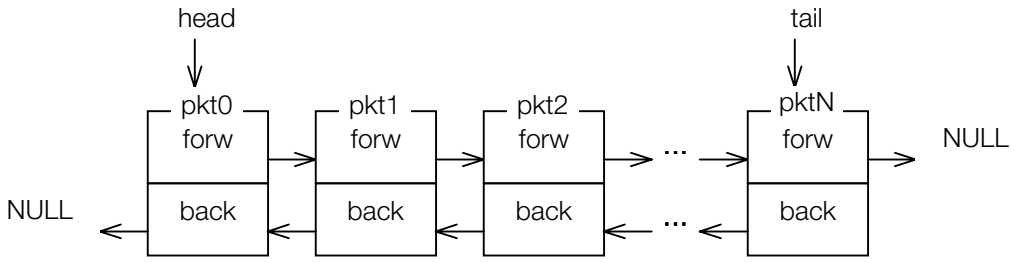
A packet is represented by a sceInetPkt_t structure, which has bidirectional links. A data area is established separately, and pointers pointing to the beginning and end of the valid data are updated during the progress of send/receive processing.

Figure 4-2



sceInetPktQ_t, which is a structure that indicates a linked bidirectional packet queue, has a pointer named head, which points to the beginning of the packet queue, and a pointer named tail, which points to the last packet.

Figure 4-3



Packet and Packet Queue Operations

The NETDEV layer can perform the following operations related to packets and packet queues.

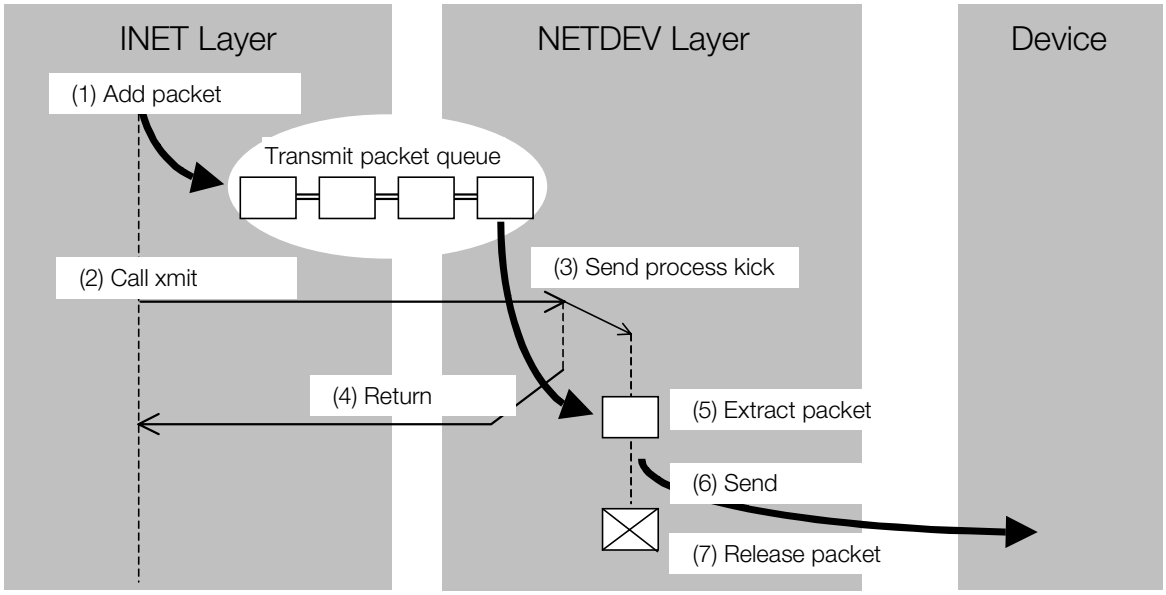
- Extract a packet from the beginning of the packet queue — sceInetPktDeQ()
- Discard transmitted packet and free memory — sceInetFreePkt()
- Allocate memory for a packet for collecting received data — sceInetAllocPkt()
- Add a received packet to the end of the receive packet queue — sceInetPktEnQ()

Send/Receive Processing Overview

Processing During Send

To send data, the INET layer adds a packet to the transmit packet queue and calls the xmit function. The NETDEV layer receives this, extracts a packet from the beginning of the transmit packet queue and sends it to the device. If send processing is kicked by the xmit function, control can return even if processing has not completed.

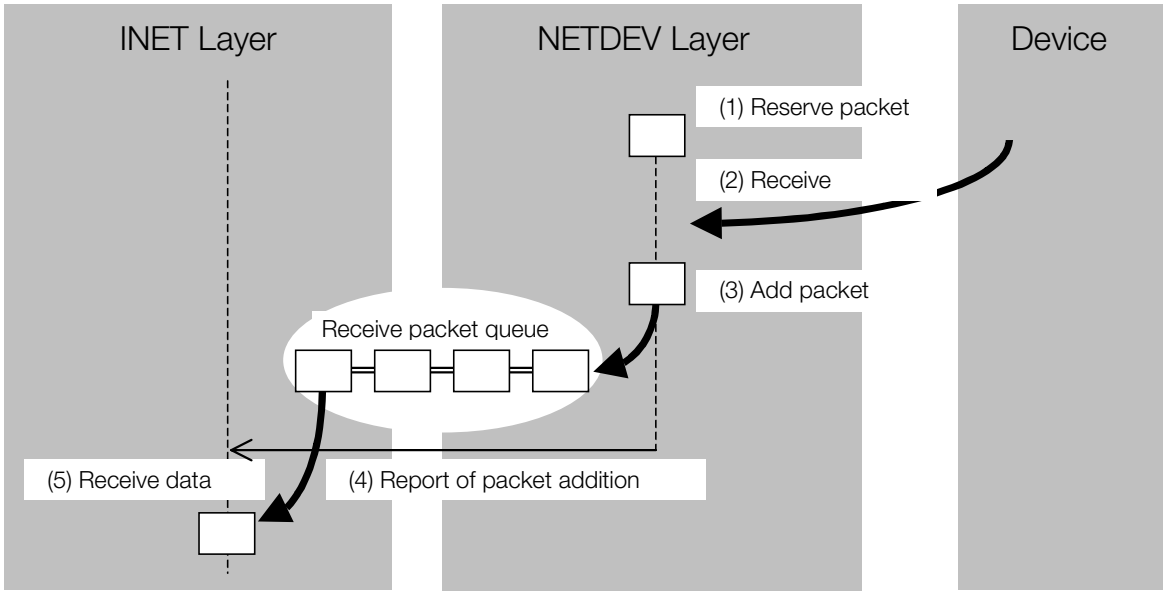
Figure 4-4



Processing During Receive

When the NETDEV layer adds a packet that was received from the device to the receive queue, it uses an event flag to report this to the INET layer. The INET layer receives this and extracts the packet.

Figure 4-5



Control Functions and Control Codes

The sceInetInterfaceControl() function, which is implemented in the INET layer, performs multiple functions depending on the value of the control code specified by the second argument. The control codes are divided into the following five types.

Table 4-2

Range	Type
0x00000000 - 0x7fffffff	INET layer control codes
0x80000000 - 0x8fffffff	NETDEV layer control codes
0x90000000 - 0xbfffffff	NETDEV module-dependent control codes
0xc0000000 - 0xcfffffff	MODEM layer common control codes
0xd0000000 - 0xffffffff	MODEM module-dependent control codes

If bit 31 of the specified control code is 1, sceInetInterfaceControl() relays code, ptr, and len directly to the control function of the appropriate NETDEV layer module. The NETDEV layer module should perform the appropriate processing according to the code value and return the appropriate return value. If bit 30 of code is 1, a NETDEV layer module that has the MODEM layer as a lower layer (for example, PPP) should relay the specified code, ptr, and len directly to the MODEM layer.

NETDEV Layer Common Control Codes

The NETDEV layer common control codes that are defined in these specifications are shown below. Although their implementation in each NETDEV layer module is not mandatory, it is recommended.

Table 4-3

Control Code	Function
sceInetNDCC_GET_THPRI	Get thread priority
sceInetNDCC_SET_THPRI	Set thread priority
sceInetNDCC_GET_IF_TYPE	Get network interface type
sceInetNDCC_GET_RX_PACKETS	Get number of receive packets
sceInetNDCC_GET_TX_PACKETS	Get number of transmit packets
sceInetNDCC_GET_RX_BYTES	Get number of receive bytes
sceInetNDCC_GET_TX_BYTES	Get number of transmit bytes
sceInetNDCC_GET_RX_ERRORS	Get number of receive errors
sceInetNDCC_GET_TX_ERRORS	Get number of transmit errors
sceInetNDCC_GET_RX_DROPPED	Get number of dropped receive packets
sceInetNDCC_GET_TX_DROPPED	Get number of dropped transmit packets

NETDEV Layer Module-Dependent Control Codes

NETDEV layer module-dependent control codes can be freely defined by the creator of each NETDEV layer module. For example, they can be used for special operations such as passing various types of PPP parameters or manipulating specific signal lines during installation.

Since control function return values must be set so that they do not overlap those of the INET interface, the values that indicate errors should be set in the following range.

Table 4-4

Layer	Error Code Range
NETDEV layer	-550 to -599

Error code values and their meanings are module-dependent and can be determined independently within the above range.

PPP Control Codes

The control codes shown below are defined for performing PPP-specific control. Since they all return values that are maintained by ppp.irx by storing them in the area specified by ptr and len, most of these control codes have a one-to-one correspondence with a keyword in the configuration file.

Table 4-5

Symbol Name	Value	Type	Corresponding Keyword
scePPPCC_GetChatDial	0x90008001	string	chat_dial
scePPPCC_GetChatLogin	0x90008002	string	chat_login
scePPPCC_GetPhoneNumber	0x90008003	string	phone_number*
scePPPCC_GetAuthName	0x90008004	string	auth_name
scePPPCC_GetAuthKey	0x90008005	string	auth_key
scePPPCC_GetPeerName	0x90008006	string	peer_name
scePPPCC_GetPeerKey	0x90008007	string	peer_key
scePPPCC_GetLcpTimeout	0x90008008	time	lcp_timeout
scePPPCC_GetIpcpTimeout	0x90008009	time	pcp_timeout
scePPPCC_GetIdleTimeout	0x9000800a	time	idle_timeout
scePPPCC_GetWantMruNego	0x9000800b	bool	want.mru_nego
scePPPCC_GetWantAccmNego	0x9000800c	bool	want.accm_nego
scePPPCC_GetWantMagicNego	0x9000800d	bool	want.magic_nego
scePPPCC_GetWantPrcNego	0x9000800e	bool	want.prc_nego
scePPPCC_GetWantAccNego	0x9000800f	bool	want.acc_nego
scePPPCC_GetWantAddressNego	0x90008010	bool	want.address_nego
scePPPCC_GetWantVjcompNego	0x90008011	bool	want.vjcomp_nego
scePPPCC_GetWantMruValue	0x90008012	u_short	want.mru_value
scePPPCC_GetWantAccmValue	0x90008013	u_long	want.accm_value
scePPPCC_GetWantAuth	0x90008014	u_char	want.auth
scePPPCC_GetWantIpAddress	0x90008015	address	want.ip_address
scePPPCC_GetWantIpMask	0x90008016	address	want.ip_mask
scePPPCC_GetAllowMruNego	0x90008017	bool	allow.mru_nego

Symbol Name	Value	Type	Corresponding Keyword
scePPPCC_GetAllowAccmNego	0x90008018	bool	allow.accm_nego
scePPPCC_GetAllowMagicNego	0x90008019	bool	allow.magic_nego
scePPPCC_GetAllowPrcNego	0x9000801a	bool	allow.prc_nego
scePPPCC_GetAllowAccNego	0x9000801b	bool	allow.acc_nego
scePPPCC_GetAllowAddressNego	0x9000801c	bool	allow.address_nego
scePPPCC_GetAllowVjcompNego	0x9000801d	bool	allow.vjcomp_nego
scePPPCC_GetAllowMruValue	0x9000801e	u_short	allow.mru_value
scePPPCC_GetAllowAccmValue	0x9000801f	u_long	allow.accm_value
scePPPCC_GetAllowAuth	0x90008020	u_char	allow.auth
scePPPCC_GetAllowIpAddress	0x90008021	address	allow.ip_address
scePPPCC_GetAllowIpMask	0x90008022	address	allow.ip_mask
scePPPCC_GetWantDNS1Nego	0x90008023	bool	want.dns1_nego
scePPPCC_GetWantDNS2Nego	0x90008024	bool	want.dns2_nego
scePPPCC_GetWantDNS1	0x90008025	address	want.dns1
scePPPCC_GetWantDNS2	0x90008026	address	want.dns2
scePPPCC_GetAllowDNS1Nego	0x90008027	bool	allow.dns1_nego
scePPPCC_GetAllowDNS2Nego	0x90008028	bool	allow.dns2_nego
scePPPCC_GetAllowDNS1	0x90008029	address	allow.dns1
scePPPCC_GetAllowDNS2	0x9000802a	address	allow.dns2
scePPPCC_GetConnectTimeout	0x9000802b	time	connect_timeout
scePPPCC_GetDialingType	0x9000802c	u_char	dialing_type
scePPPCC_GetToneDial	0x9000802e	string	dialing_type_string (tone)
scePPPCC_GetPulseDial	0x9000802f	string	dialing_type_string (pulse)
scePPPCC_GetAnyDial	0x90008030	string	dialing_type_string (any)
scePPPCC_GetLogFlags	0x90008033	int	log_flags
scePPPCC_GetHisAddress	0x90008034	address	---
scePPPCC_GetHisMask	0x90008035	address	---
scePPPCC_GetOmitEmptyFrame	0x90008036	bool	omit_empty_frame
scePPPCC_GetChapType	0x90008037	int	chap_type
scePPPCC_GetCurrentStatus	0x90008038	string	---
scePPPCC_GetLcpEchoTimeout	0x90008039	time	---

The Type column indicates the type of data that is returned. The values that must be specified for the ptr and len arguments, depending on the type, are as follows.

Table 4-6

Type	Returned Data	ptr	len
string	String	Starting address of string	Number of bytes in storage area
time	Interval in ms	Pointer to int	sizeof(int)
bool	True (not 0) or False (0)	Pointer to u_char	sizeof(u_char)
u_short	Unsigned 2-byte numeric value	Pointer to u_short	sizeof(u_short)
u_long	Unsigned 4-byte numeric value	Pointer to u_long	sizeof(u_long)
u_char	Unsigned 1-byte numeric value	Pointer to u_char	sizeof(u_char)
address	IP address	Pointer to sceInetAddress_t	sizeof (sceInetAddress_t)
int	Signed 4-byte numeric value	Pointer to int	sizeof(int)

The functions of various control codes are shown below only for the control codes for which the correspondence with a keyword is not obvious.

scePPPCC_GetChatDial

This control code returns the chat script string to be used when calling or answering, depending on the answer mode setting.

When the answer mode is False, the calling string, that is, the string formed by concatenating chat_init, chat_additional, and chat_dial, is returned.

When answer_mode is True, the answering string, that is, the string formed by concatenating chat_init, chat_additional, "TIMEOUT %d", and chat_answer, is returned. However, "TIMEOUT %d" is included only when answer_timeout is specified.

scePPPCC_GetToneDial, scePPPCC_GetPulseDial, and scePPPCC_GetAnyDial

These control codes return the dialing string that corresponds to the line type among the three dialing strings that are defined by the dialing_type_string keyword.

They are set only when answer_mode is False.

scePPPCC_GetDialingType

This control code returns the line type that is defined by the dialing_type keyword according to the following values.

Table 4-7

Line Type	Symbol Name (Value)
Tone line	sceNetCnf_DIALING_TYPE_TONE (0)
Pulse line	sceNetCnf_DIALING_TYPE_PULSE (1)
Other line	sceNetCnf_DIALING_TYPE_ANY (2)

This is set only when answer_mode is False.

scePPPCC_GetPhoneNumber

This control code returns the dialing string that is set. It is the string formed by concatenating the outside_number, outside_delay, and the telephone number (any of phone_number[0-9]) that is currently set for PPP.

This is set only when answer_mode is False.

scePPPCC_GetWantAuth and scePPPCC_GetAllowAuth

These control codes return the authentication method that is set according to the following values.

Table 4-8

Authentication Method	Value
any (no authentication)	0
pap (PAP authentication)	1
chap (CHAP authentication)	2
pap/chap (PAP authentication followed by CHAP authentication)	3
chap/pap (CHAP authentication followed by PAP authentication)	4

scePPPCC_GetHisAddress and scePPPCC_GetHisMask

These control codes return the remote-side IP address and subnet mask that were obtained as a result of IPCP negotiation.

scePPPCC_GetChapType

This control code returns the CHAP authentication algorithm restriction that has been set according to the following values.

Table 4-9

Algorithm	Value
no (no restriction)	0x00
md5 (MD5 only)	0x05
ms (MS Version 1 only)	0x80
ms-v1 (MS Version 1 only)	0x80
ms-v2 (MS Version 2 only)	0x81

scePPPCC_GetCurrentStatus

This control code returns the status string that is being maintained by the PPP layer.

scePPPCC_GetLcpEchoTimeout

This control code gets the setting state of the Keep-Alive function in the LCP level.

When zero or a negative value is returned, the Keep-Alive function is disabled. When a positive value is returned, the function is enabled so that the connection will be disconnected at the PPP level if there is no response five consecutive times within that interval.

By default the Keep-Alive function is set to disabled state.

Ethernet Interface-Dependent Statistical Information

The control codes shown below are used for getting statistical information about the Ethernet interface. They are module-dependent control codes that are also defined for NETDEV layer modules only if those modules support the Ethernet interface.

Table 4-10

Constant	Value
scelnetNDCC_GET_MULTICAST	0x80011000
scelnetNDCC_GET_COLLISIONS	0x80011001
scelnetNDCC_GET_RX_LENGTH_ER	0x80011002
scelnetNDCC_GET_RX_OVER_ER	0x80011003
scelnetNDCC_GET_RX_CRC_ER	0x80011004
scelnetNDCC_GET_RX_FRAME_ER	0x80011005
scelnetNDCC_GET_RX_FIFO_ER	0x80011006
scelnetNDCC_GET_RX_MISSED_ER	0x80011007
scelnetNDCC_GET_TX_ABORTED_ER	0x80011008
scelnetNDCC_GET_TX_CARRIER_ER	0x80011009
scelnetNDCC_GET_TX_FIFO_ER	0x8001100a
scelnetNDCC_GET_TX_HEARTBEAT_ER	0x8001100b
scelnetNDCC_GET_TX_WINDOW_ER	0x8001100c

All of the statistical information values shown above are defined as values that are returned as an unsigned int value in the area indicated by (ptr, len=4). However, the hardware information that is associated with each symbol depends on the implementation of the NETDEV interface module. Also, the operation that is performed when an overflow occurs depends on the implementation of the NETDEV interface module.

Ethernet Interface-Dependent Settings

The control codes shown below, which are Ethernet interface-dependent settings, are module-dependent control codes that are also defined for NETDEV layer modules, only if those modules support the Ethernet interface.

Table 4-11

Constant	Value
scelnetNDCC_GET_NEGO_MODE	0x80020000

All of the statistical information values shown above are defined as values that are returned as an unsigned int value in the area indicated by (ptr, len=4). However, the hardware information that is associated with each symbol depends on the implementation of the NETDEV interface module. Also, the operation that is performed when an overflow occurs depends on the implementation of the NETDEV interface module.

ScelnetNDCC_GET_NEGO_MODE

This control code pertains to Ethernet operating mode. It sets the logical OR of the following values in the area indicated by (ptr, len=4) as configuration information for the current hardware device held by the NETDEV layer module.

Table 4-12

Constant	Value
scelnetNDNEGO_10	0x0001
scelnetNDNEGO_10_FD	0x0002
scelnetNDNEGO_TX	0x0004
scelnetNDNEGO_TX_FD	0x0008
scelnetNDNEGO_PAUSE	0x0040
scelnetNDNEGO_AUTO	0x0080

When ScelnetNDNEGO_AUTO is ON, the operating mode is determined by the Ethernet operating mode auto recognition function. In this case, the remaining bits are used for negotiation.

When ScelnetNDNEGO_AUTO is OFF, the operating mode is determined by the remaining bit specifications without using the Ethernet operating mode auto recognition function. The valid specifications are as follows.

Table 4-13

Value	Operating Mode
scelnetNDNEGO_10	10Base-T Half-Duplex
scelnetNDNEGO_10_FD	10Base-T Full-Duplex (no Flow Control)
scelnetNDNEGO_10_FD | scelnetNDNEGO_PAUSE	10Base-T Full-Duplex (with Flow Control)
scelnetNDNEGO_TX	100Base-TX Half-Duplex
scelnetNDNEGO_TX_FD	100Base-TX Full-Duplex (No Flow Control)
scelnetNDNEGO_TX_FD | scelnetNDNEGO_PAUSE	100Base-TX Full-Duplex (with Flow Control)

Operations for combinations other than the ones shown above depend on the NETDEV layer implementation.