

# **PlayStation®2 IOP Library Overview**

## **Release 2.4.3**

### **Network Libraries**

© 2002 Sony Computer Entertainment Inc.

Publication date: January 2002

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.

The *PlayStation®2 IOP Library Overview - Network Libraries* manual is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® license agreements.

The *PlayStation®2 IOP Library Overview - Network Libraries* manual is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® license agreements.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® license agreements.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *PlayStation®2 IOP Library Overview - Network Libraries* manual is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

 and PlayStation are registered trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

# Summary Table of Contents

<b>About This Manual</b>	<b>v</b>
Changes Since Last Release	v
Related Documentation	v
Typographic Conventions	vi
Developer Support	vi
<b>Chapter 1: Network (INET) Configuration Library</b>	<b>1-1</b>
<b>Chapter 2: Network Library</b>	<b>2-1</b>
<b>Chapter 3: Modem Driver Development Library</b>	<b>3-1</b>
<b>Chapter 4: Common Network Configuration Library</b>	<b>4-1</b>
<b>Chapter 5: Network Device Library</b>	<b>5-1</b>



---

## About This Manual

This is the Runtime Library Release 2.4.3 version of the *PlayStation®2 IOP Library Overview - Network Libraries* manual.

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

## Changes Since Last Release

### Chapter 2: Network Library

- Additional information has been added to the "Supported Equipment" section of "Library Overview".
- Errors in the paths to the following modules have been fixed in "Confirming Operation".

route.irx

nsconfig.irx

ifconfig.irx

### Chapter 4: Common Network Configuration Library

- In the "Method for starting up netcnf.irx" section of "NET Configuration File Overview" a description of arguments has been added.

### Chapter 5: Network Device Library

- In the "Ethernet Interface-Dependent Settings" section of "Control Functions and Control Codes" descriptions on the following codes have been added.

scelnetNDCC\_SET\_NEGO\_MODE

scelnetNDCC\_GET\_LINK\_STATUS

scelnetNDCC\_SET\_MULTICAST\_LIST

## 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
<code>courier</code>	Indicates literal program code.
<i>italic</i>	Indicates names of arguments and structure members (in structure/function definitions only).
<b>medium bold</b>	Indicates data types and structure/function names (in structure/function definitions only).
<a href="#">blue</a>	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
<i>In North America:</i> Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd. Foster City, CA 94404, U.S.A. Tel: (650) 655-8000	<i>In North America:</i> E-mail: PS2_Support@playstation.sony.com Web: <a href="http://www.devnet.scea.com/">http://www.devnet.scea.com/</a> 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
<i>In Europe:</i> Attn: Production Coordinator Sony Computer Entertainment Europe 30 Golden Square London W1F 9LD, U.K. Tel: +44 (0) 20 7859-5000	<i>In Europe:</i> E-mail: ps2_support@scee.net Web: <a href="https://www.ps2-pro.com/">https://www.ps2-pro.com/</a> 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	1-3
Starting Up inetctl.irx	1-3
Configuration Procedure	1-4
Determining Timeouts	1-5





---

## 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= <i>thpri</i>	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 `sceInetCtlGetInterfaceList()` 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  
scelNETCNF\_IEV\_Conf is reported followed by scelNETCNF\_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

---

<b>Library Overview</b>	<b>2-3</b>
Software Configuration	2-3
Supported Equipment	2-5
Managing Network Configuration Information	2-5
Restrictions and Precautions	2-6
<b>Confirming Operation</b>	<b>2-6</b>
<b>inet.irx Startup Options</b>	<b>2-9</b>
<b>ppp.irx Startup Options</b>	<b>2-9</b>
<b>INET API Overview</b>	<b>2-10</b>
Connection	2-10
Timeouts	2-10
Multithread Support	2-11
<b>Precautions</b>	<b>2-11</b>
Power OFF processing	2-11



## 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)

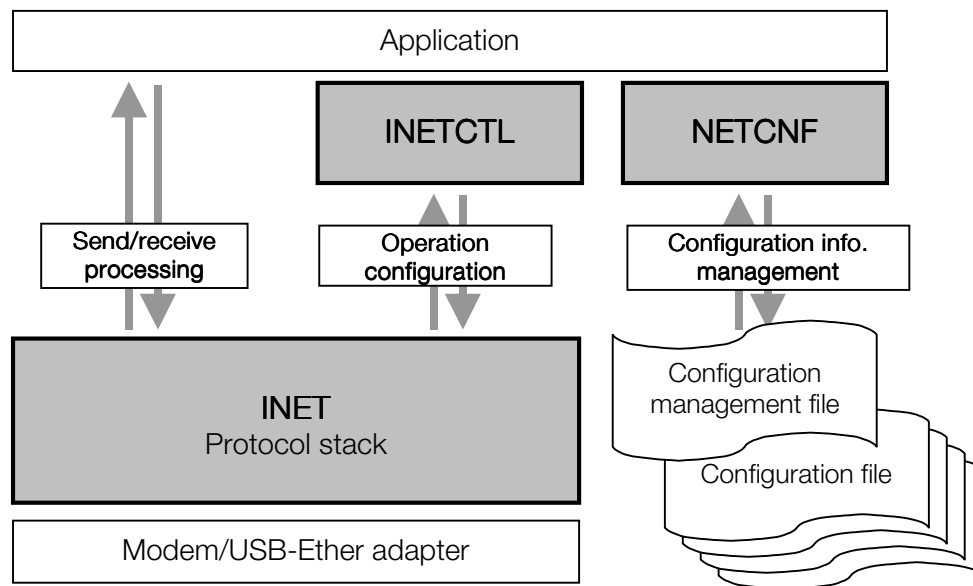
Note: Although the drivers provided by SCEI are only for development and cannot be used in a title application, there are no restrictions on the use of independently developed drivers.

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

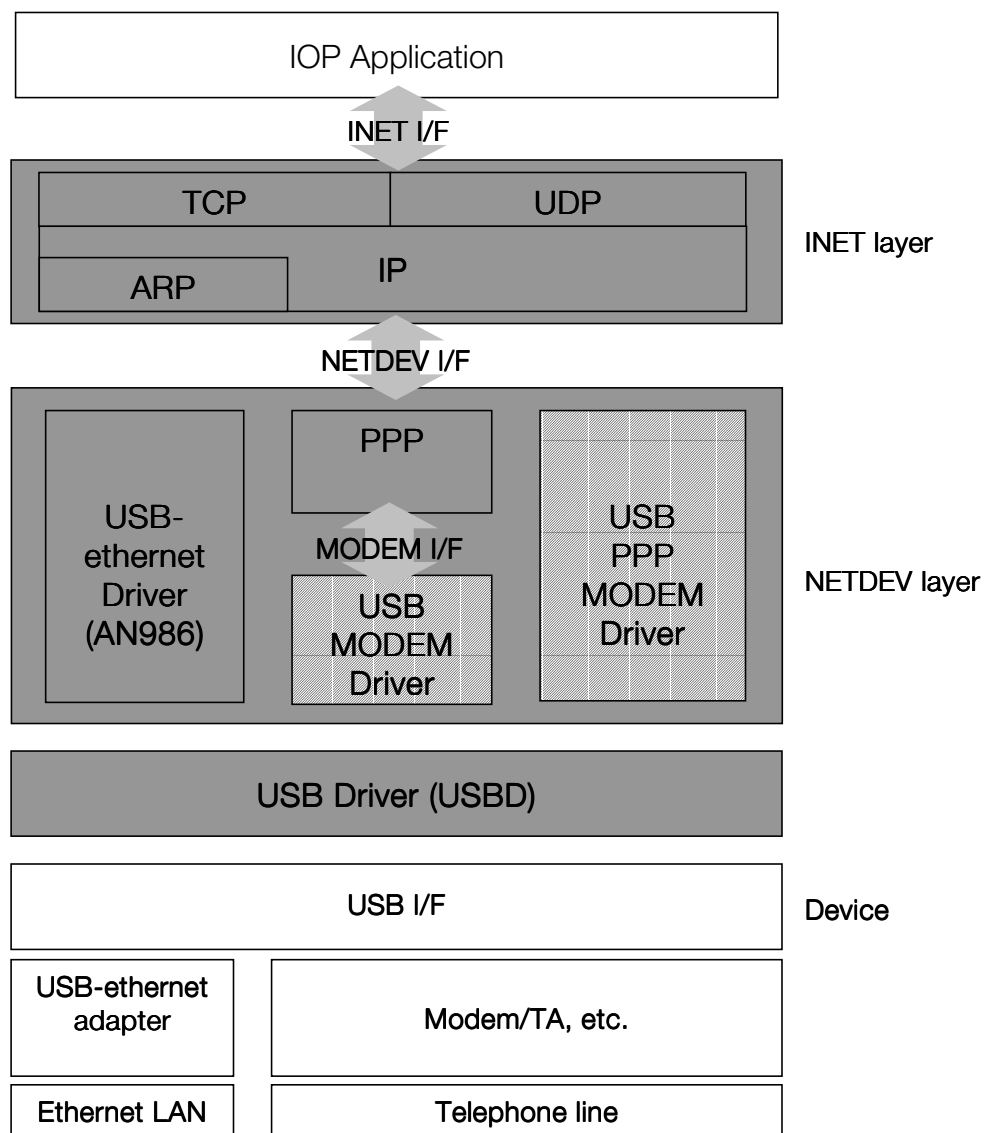
The network support library is divided into three main parts as shown below.

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	MELCO	LUA2-TX
Japan	I-O DATA	USB-ET/TX
Japan	PLANEX	UE-100TX
Japan	PLANEX	UE-200TX
Japan	Corega	FEther USB-TX
Japan	Elecom	LD-USBL/TX
U.S.	Linksys	USB100TX
U.S.	D-Link	DSB-650TX
Europe	ADMtek	Pegasus
Europe	D-Link	DU-E100
Europe	Linksys	USB100TX B
Europe	D-Link	DU-E100 B1

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 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 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 /usr/local/sce/iop/util/inet/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 errors:0 dropped:0
          TX packets:4 bytes:764 errors:0 dropped:0
```

## 8. Confirm routing information with route

Execute route to confirm routing information.

```
dsidb R> mstart /usr/local/sce/iop/util/inet/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 /usr/local/sce/iop/util/inet/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

---

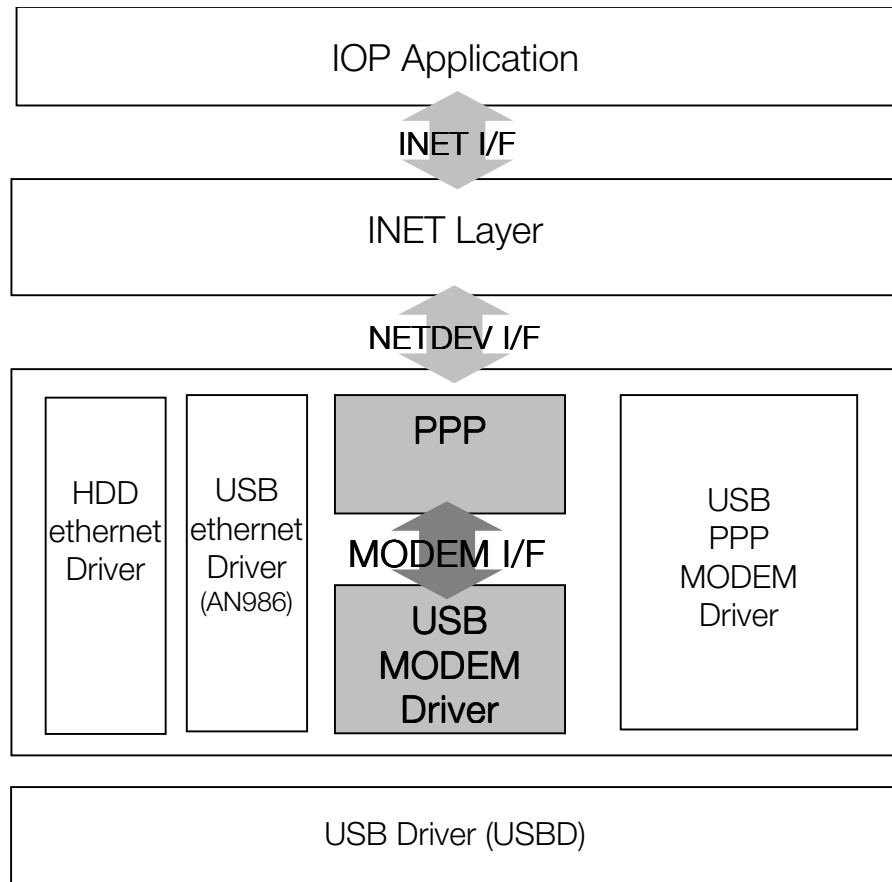
<b>Overview</b>	<b>3-3</b>
<b>PPP Layer and Modem Driver Layer</b>	<b>3-4</b>
Exchanges Between the PPP Layer and Modem Driver Layer	3-4
Modem and Processing Function Registration	3-5
Modem Information Structure	3-5
Registration Deletion	3-6
Modem State Change Notification	3-7
<b>Modem Driver Layer Functions</b>	<b>3-7</b>
Modem Driver Layer Common Control Codes	3-8
Modem Driver Layer Command Control codes (for Serial Devices)	3-9
<b>Reference Driver</b>	<b>3-9</b>
Operation Verification Procedure	3-9
Action Taken if a Connection Cannot be Established	3-11
<b>Cautions</b>	<b>3-11</b>
Timing of Notification of Increase in Sendable or Receivable Byte Count	3-11
Sendable and Receivable Byte Count Updates	3-12
USB Autoload Support	3-12



## 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



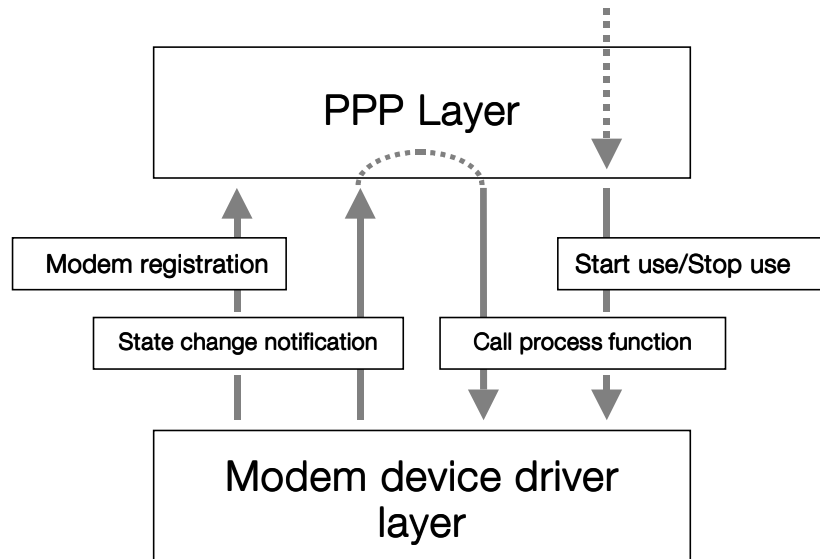
Operations that the PPP layer performs for the modem via the modem driver layer are described in the PPP layer specifications and are not covered in this document. For example, modem calling command strings are not covered here. Even if the processing of those commands is actually implemented by the 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
<code>sceModemEFP_StartDone</code>	<code>0x00000001</code>	Start use processing completed
<code>sceModemEFP_PlugOut</code>	<code>0x00000002</code>	Modem disconnected
<code>sceModemEFP_Connect</code>	<code>0x00000010</code>	Line connected
<code>sceModemEFP_Disconnect</code>	<code>0x00000020</code>	Line disconnected
<code>sceModemEFP_Ring</code>	<code>0x00000040</code>	Incoming call
<code>sceModemEFP_Recv</code>	<code>0x00000100</code>	Receivable byte count increased
<code>sceModemEFP_Send</code>	<code>0x00000200</code>	Sendable byte count increased
<code>sceModemEFP_UpperUse</code>	<code>0xffff0000</code>	(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 receive 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 rcv(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
<code>sceModemCC_GET_OE_COUNT</code>	0xc0010002	Get overrun error count
<code>sceModemCC_GET_PE_COUNT</code>	0xc0010003	Get parity error count
<code>sceModemCC_GET_FE_COUNT</code>	0xc0010004	Get framing error count
<code>sceModemCC_GET_BO_COUNT</code>	0xc0010005	Get buffer overflow count
<code>sceModemCC_GET_PARAM</code>	0xc0020000	Get serial communication parameter
<code>sceModemCC_SET_PARAM</code>	0xc1020000	Set serial communication parameter
<code>sceModemCC_GET_LINE</code>	0xc0030000	Get state of each signal line
<code>sceModemCC_SET_LINE</code>	0xc1030000	Control each signal line
<code>sceModemCC_SET_BREAK</code>	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:usbmlload.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 `usb_d`, that is, by a USB 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:

## Common Network Configuration Library

---

<b>NETCNF Application Overview</b>	<b>4-3</b>
<b>NET Configuration File Overview</b>	<b>4-3</b>
NET Configuration File Types	4-3
Undefined Keywords	4-3
Configuration Processing	4-3
Method for starting up netcnf.irx	4-4
<b>Syntax of NET Configuration Files</b>	<b>4-4</b>
General Format	4-4
Comments	4-4
CR Code	4-4
Strings and Escape Processing Within a String	4-4
Kanji Code	4-5
Header Comment	4-5
<b>NET_CNF Keywords</b>	<b>4-5</b>
interface	4-5
chat_additional	4-6
redial_count	4-6
redial_interval	4-7
outside_number	4-7
outside_delay	4-7
dialing_type	4-7
<b>ATTACH_CNF Keywords</b>	<b>4-8</b>
type	4-10
vendor	4-10
product	4-10
location	4-11
dhcp	4-11
dhcp_host_name	4-11
dhcp_host_name_null_terminated	4-12
dhcp_release_on_stop	4-12
address	4-12
netmask	4-13
nameserver	4-13
route	4-13
phy_config	4-13
mtu	4-14
pppoe	4-14
pppoe_service_name	4-15
pppoe_ac_name	4-15
pppoe_host_uniq_auto	4-15
chat_additional	4-15
redial_count	4-16

redial_interval	4-16
outside_delay	4-16
dialing_type	4-17
phone_number	4-17
answer_mode	4-18
answer_timeout	4-18
chat_login	4-18
auth_name, auth_key, peer_name, and peer_key	4-18
want.auth	4-19
allow.auth	4-20
lcp_timeout	4-20
ipcp_timeout	4-20
idle_timeout	4-21
connect_timeout	4-21
mru_nego and mru	4-21
accm_nego and accm	4-22
magic_nego	4-22
prc_nego	4-22
acc_nego	4-23
address_nego, ip_address, and ip_mask	4-23
vjcomp_nego	4-24
dns1_nego, dns1, dns2_nego, and dns2	4-24
force_chap_type	4-25
omit_empty_frame	4-25
log_flags	4-26
<b>DIAL_CNF Keywords</b>	<b>4-26</b>
dialing_type_string	4-26
redial_string	4-27
chat_init	4-27
chat_dial	4-27
chat_answer	4-27
<b>Chat Script Specifications</b>	<b>4-28</b>
Script Specifications	4-28
Chat Script Setup Examples	4-29
Precautions	4-30

## NETCNF Application Overview

netcnf.irx is a resident library that provides functions for managing the configuration management file and configuration files.

In addition to defining structures that correspond to configuration files, netcnf.irx provides functions for obtaining a list of configuration files that are saved on a memory card, reading the contents of a configuration file into structures in memory, etc.

The common network configuration library (netcnf.irx) is used to create network configuration applications for handling network configuration files, which are shared resources. Standards for creating these must be observed.

For creating network configuration applications, see the information under the EE/IOP common document directory (general/).

- Network Configuration Application Creation Standards (setapp\_r)

## NET Configuration File Overview

Individual information required for a network connection such as the IP address within the network or the telephone number of the access point for a dialup connection is maintained for each individual Playstation 2 as a configuration file conforming to the specifications in this document. All programs that create, edit, or interpret configuration files must conform to these specifications.

### NET Configuration File Types

There are three types of NET configuration files as shown below. The keywords that can be specified within a configuration file are defined for each type of configuration file.

Table 4-1

File Type	Contents
NET_CNF	Information related to lines and filenames of other configuration files
ATTACH_CNF	Configuration information related to an individual network interface, that is, specifications of processing to be performed when a network device is connected
DIAL_CNF	Information about the dialing procedure of an individual modem

### Undefined Keywords

To prepare for the addition of new keywords in the future, a program that edits NET configuration files must maintain any undefined or unsupported keywords that appear in the original file as is, without making changes.

### Configuration Processing

Processing for setting up the configuration by interpreting the contents of NET configuration files is performed by starting up netcnf.irx. For details, refer to the netcnf.irx document.

## Method for starting up netcnf.irx

Load netcnf.irx using the following method:

```
netcnf.irx [<option>] icon=<icon-path> iconsys=<iconsys-path>
```

Note: make sure to attach the following arguments:

```
icon      icon file
iconsys   icon management file (icon.sys)
```

The following is provided for option:

- *-no\_check\_capacity*  
No capacity checks are performed with the sceNetCnfAddEntry() and sceNetCnfEditEntry() functions.

---

## Syntax of NET Configuration Files

This section explains the basic rules for describing NET configuration files.

### General Format

NET configuration files are interpreted in units of “lines”, that terminate with either a “\n” (LF:0x0a), or the end of the file. However, if “\\” (0x5c) occurs immediately before “\n”, the two bytes for that “\\” and “\n” are ignored, and the next line is treated as if it were a logical continuation.

Generally, a line within a NET configuration file has the following format.

```
keyword  arg1  arg2  ...
```

A keyword and its arguments can be described in a single set on one logical line. The keyword and arguments are each separated by one or more consecutive spaces (SP:0x20) or tabs (TAB:0x08). Besides spaces and tabs, any character that is considered to be white space by the isspace() macro, which is defined in the file ctype.h in the IOP development environment, will be treated as a delimiter.

### Comments

A “#” at the beginning of a line indicates a comment line. In addition, if a “#” is placed anywhere other than within a string, the portion from “#” until the end of that line will be treated as a comment.

### CR Code

All CR codes (0x0d) within a NET configuration file are ignored except when they are specified with an escape.

### Strings and Escape Processing Within a String

When an argument is a string, that entire string should be specified enclosed in double quotes (“”). The setting for usr\_name in the configuration management file need not be enclosed in double quotes (“”). Although there are exceptional cases in which strings do not need to be enclosed in double quotes (“”), this is generally not the case. Therefore, descriptions related to this condition will be omitted.

The following escape specifications can be used to specify control codes within a string that is enclosed in double quotes (“”).



Table 4-2

Escape Specification	Corresponding Character
\a	BEL (0x07)
\b	BS (0x08)
\t	TAB (0x09)
\n	LF (0x0a)
\v	VT (0x0b)
\f	FF (0x0c)
\r	CR (0x0d)
\ooo	Octal code specification. ooo is a one- to three-digit octal number
\xxx	Hexadecimal code specification. xx is a one- or two-digit hexadecimal number
\X	Character X specification. X is a character other than the above ([^abtnvfr0-9xa-fA-F]), which is mainly used for specifying '\ or ''

When an actual escape character is specified in a configuration file, it should be specified as “\\a” in order to escape the “\”.

## Kanji Code

UTF8 should be used for kanji code. Kanji and kana characters must not be used for configuration items (configuration contents) other than configuration names. Only hiragana and katakana can be used in a configuration name. Also, a comma “,” cannot be used because it is treated as a delimiter in the configuration management file.

## Header Comment

The following comment must be entered at the beginning of a NET configuration file.

```
# <Sony Computer Entertainment Inc.> comment
```

The contents after the “>” up to the end of the line can be arbitrary.

The header comment is used by netcnf.irx for error detection during individual encoding/decoding. (Although only ATTACH\_CNF is subject to individual encoding/decoding, the header comment is required in all NET configuration files.)

## NET\_CNF Keywords

A NET\_CNF-format file, which is the root of a configuration file group, contains individual information for each user and the filenames of other configuration files. The keywords that can be specified within NET\_CNF are as follows.

### interface

This is a keyword for defining a network interface. More specifically, it is a keyword for specifying ATTACH\_CNF. This keyword can be specified multiple times within NET\_CNF.

#### Syntax

```
interface “display_name” “ifc_cnf” [“dev_cnf”]
```

**Description**

*display\_name* is the display name that enables the user to distinguish each interface definition in a configuration file creation or editing program. As long as there is no specific limitation, an arbitrary character string can be specified.

*ifc\_cnf* and *dev\_cnf* specify the filenames of ATTACH\_CNF files in which configuration information related to that network interface is specified. The filenames can be specified as a relative pathname based on the directory in which NET\_CNF exists.

For a PPP connection, ISP-dependent configuration information is entered in *ifc\_cnf* and modem-dependent configuration information is entered in *dev\_cnf*. Separate usage conventions have been established for each individual file.

However, the type in the *dev\_cnf* file alone is never merged during connection processing. Only the type in *ifc\_cnf* is used.

If a certain keyword is duplicated in the three files NET\_CNF, *ifc\_cnf*, and *dev\_cnf*, the last specification becomes effective. That is, the priority order for interpreting duplicate keywords is NET\_CNF < *ifc\_cnf* < *dev\_cnf*.

**chat\_additional**

This is a keyword for defining an additional script for a modem.

**Syntax**

chat\_additional "*additional\_script*"

**Description**

This is a keyword for defining an additional script to be sent to the modem following the initialization script that is defined by the chat\_init keyword for the PPP interface. It is for user-specific support such as adjusting the modem transmission level. The definition in NET\_CNF becomes a default additional script that does not depend on the interface. If this keyword is specified in the relevant ATTACH\_CNF beforehand, when a definition that depends on an individual interface is required, the setting in NET\_CNF will be overwritten.

For information about script specifications, see "Chat Script Specifications."

**redial\_count**

This is a keyword for specifying the upper limit of the redial count.

**Syntax**

redial\_count *redial\_count*

**Description**

This is a keyword for specifying the maximum number of times redialing is to be attempted when the modem driver returns "busy" for the PPP interface. When this keyword is not specified, the default is 0 times and no redialing is attempted.

The specification in NET\_CNF becomes the default redial count that does not depend on the interface. If this keyword is specified in the relevant ATTACH\_CNF beforehand when the redial count is to be changed for each interface, the setting in NET\_CNF will be overwritten.

## redial\_interval

This is a keyword for specifying the redial interval.

### Syntax

`redial_interval redial_interval`

### Description

This is a keyword for specifying the delay interval for redialing in seconds. The default when no `redial_interval` is specified is 10 seconds.

The specification in `NET_CNF` becomes the default redial interval that does not depend on the interface. If this keyword is specified in the relevant `ATTACH_CNF` beforehand when the redial interval is to be changed for each interface, the setting in `NET_CNF` will be overwritten.

## outside\_number

This is a keyword for specifying the outside line calling string.

### Syntax

`outside_number "outside_number"`

### Description

This is a keyword for specifying the outside line calling string that is to be dialed at the beginning of a telephone number when outside line calling is used. The default when no `"outside_number"` is specified is `""` (empty string).

## outside\_delay

This is a keyword for specifying the outside line calling delay string.

### Syntax

`outside_delay "outside_delay"`

### Description

This is a keyword for specifying the outside line calling delay string that is to be inserted between the outside line calling string and the telephone number when outside line calling is used. Although this depends on the model and configuration of the modem, for many AT command modems, a comma `,` means a delay of approximately 2 seconds. The default when no `"outside_delay"` is specified is `""` (empty string).

The specification in `NET_CNF` becomes the default outside line calling delay string that does not depend on the interface. If this keyword is specified in the relevant `ATTACH_CNF` beforehand when the outside line calling delay string is to be changed for each interface, the setting in `NET_CNF` will be overwritten.

This outside line calling delay string is not used when the outside line calling string (`outside_number`) is an empty string.

## dialing\_type

This is a keyword for specifying the line type.

### Syntax

`dialing_type [tone | pulse | any]`

**Description**

The line type is specified with the following arguments.

**Table 4-3**

Argument	Meaning
tone	Tone line (analog) [default]
pulse	Pulse line (analog)
any	Line other than the above (such as digital)

The specification in NET\_CNF becomes the default line type that does not depend on the interface. If this keyword is specified in the relevant ATTACH\_CNF beforehand when the line type is to be changed for each interface, the setting in NET\_CNF will be overwritten.

---

## ATTACH\_CNF Keywords

ATTACH\_CNF is the configuration file format corresponding to the NETDEV layer. The keywords that are specified in ATTACH\_CNF are divided into two groups. One group consists of keywords for specifying the NETDEV layer modules and network devices to which the configuration file is to be applied. The keywords that belong to this group are shown below.

**Table 4-4**

Keyword	Specification Contents
type	NETDEV layer type specification (USB-Ether or PPP)
vendor	Network device's vendor specification
product	Network device's product name specification
location	Network device's destination port specification

The other group consists of keywords for specifying settings related to the network interface and network device. This group has many keywords that are specific to PPP connections (indicated by [\*] in the following table).

**Table 4-5**

Keyword	Specification Contents
dhcp	Whether or not to use DHCP
dhcp_host_name	Specifies the host name to be reported to the DHCP server
dhcp_host_name_null_terminated	Specifies whether the terminating NUL character is to be included in the host name that is reported to the DHCP server
dhcp_release_on_stop	Specifies whether to report the release of the IP address to the DHCP server when an interface goes down
address	Specifies the IP address
netmask	Specifies the subnet mask
nameserver	Specifies the name server
route	Specifies routing information

Keyword	Specification Contents
phy_config	Specifies the method of configuring the physical layer chip
mtu	Specifies the default values for want.mru, allow.mru and the value of MTU
pppoe	Specifies whether PPPoE (PPP over Ethernet) is enabled
pppoe_service_name	Specifies the service name for PPPoE (PPP over Ethernet)
pppoe_ac_name	Specifies the access concentrator name for PPPoE(PPP over Ethernet)
pppoe_host_uniq_auto	Specifies whether host unique data should be automatically created to distinguish clients for PPPoE(PPP over Ethernet)
chat_additional	Definition of additional script for the modem
redial_count	Specifies the upper limit of the redial count
redial_interval	Specifies the redial interval
outside_delay	Specifies the outside line calling delay string
dialing_type	Specifies the line type
phone_number	Specifies the calling destination telephone number (*)
answer_mode	Specifies whether or not to establish a network connection because of an incoming call
answer_timeout	Specifies the timeout interval when the connection is established by answering
chat_login	Specifies the login procedure (*)
auth_name / auth_key / peer_name / peer_key	Authentication name and authentication key (*)
want.auth / allow.auth	Authentication method (*)
lcp_timeout	LCP timeout interval (*)
ipcp_timeout	IPCP timeout interval (*)
idle_timeout	Idle timeout interval (*)
connect_timeout	Line connection timeout interval (*)
mru_nego / mru	Maximum reception data length (*)
acm_nego / acm	Asynchronous control character map (ACCM:Async-Control-Character-Map) negotiation (*)
magic_nego	Magic number negotiation (*)
prc_nego	Protocol field compression (PRC:Protocol-Field-Compression) negotiation (*)
acc_nego	Address and control field compression (Address-and-Control-field-Compression) negotiation (*)
address_nego / ip_address / ip_mask	IP address and subnet mask negotiation

Keyword	Specification Contents
<code>vjcomp_nego</code>	VJ compression negotiation (*)
<code>dnsn_nego</code> / <code>dnsn</code>	Name server negotiation (*)
<code>force_chap_type</code>	Limit the CHAP authentication algorithms that are allowed
<code>omit_empty_frame</code>	Empty frame omission (*)
<code>log_flags</code>	Log display contents (*)

The PPP connection keywords are further divided into those for making settings related to the ISP (those that should be entered in *ifc.cnf*) and those for making settings related to the modem (those that should be entered in *dev.cnf*). Separate usage conventions have been established for each individual file.

## type

This is a keyword for specifying the type of layer that is supported by ATTACH\_CNF among the layers that are beneath the INET layer.

### Syntax

`[-]type [eth | ppp | nic]`

Any of the following can be specified.

<code>-type</code>	No lower level layer type specification [default]
<code>type eth</code>	Supports USB-Ethernet
<code>type ppp</code>	Supports PPP connections
<code>type nic</code>	Supports Ethernet (Network Adaptor)

## vendor

This is a keyword for specifying the vendor name of an interface device supported by ATTACH\_CNF.

### Syntax

`[-]vendor [vendor]`

Any of the following can be specified.

- `-vendor` No vendor specification [default]
- `vendor vendor` Supports only devices having vendor name “*vendor*”

### Description

The vendor name is not the one that is defined as a name with the USB String Descriptor. It is the string specified by the driver of the device.

## product

This is a keyword for specifying the product name of a device supported by ATTACH\_CNF.

### Syntax

`[-]product [product]`

- `-product` No product name specification [default]

- `product product` Supports only devices having product name “*product*”

### Description

The product name is not the one that is defined as a name with the USB String Descriptor. It is the string specified by the driver of the device.

## location

This is a keyword for specifying location information, namely the port to which the device is connected, of a device supported by ATTACH\_CNF.

### Syntax

`[-]location [location]`

Any of the following can be specified.

- `-location` No location specification [default]
- `location location` Supports only the device at the location specified by “*location*”

Currently, *location* is represented by the following kinds of strings.

Table 4-6

<i>location</i>	Meaning
“USB-1”	USB port 1 of the PlayStation 2 console
“USB-2”	USB port 2 of the PlayStation 2 console
“USB-1,3”	Port 3 of the Hub connected to USB port 1 of the PlayStation 2 console

### Description

The *location* specification is provided for supporting situations when more than one instance of the same product made by the same manufacturer is to be used simultaneously. For practical purposes, it is probably unnecessary.

## dhcp

This is a keyword for specifying whether or not a setting such as the IP address is to be made using DHCP.

### Syntax

`[-]dhcp`

Any of the following can be specified.

- `-dhcp` DHCP is not used. [default]
- `dhcp` DHCP is used.

## dhcp\_host\_name

This is a keyword for specifying the host name to be reported to the DHCP server.

### Syntax

Any of the following can be specified.

- `-dhcp_host_name` No host name is reported [default]
- `dhcp_host_name host_name` *host\_name* is reported as the host name

**Description**

Depending on the DHCP server, a host name specified from the server side may have to be included in DHCP messages. This is a keyword for specifying the host name for this kind of DHCP server. The host name that is set is only used during DHCP processing.

**dhcp\_host\_name\_null\_terminated**

This is a keyword for specifying whether or not the terminating NUL character is to be included in the host name that is reported to the DHCP server.

**Syntax**

Any of the following can be specified.

- `-dhcp_host_namnull_terminated` NUL character is not included in host name
- `dhcp_host_namnull_terminated` NUL character is included in host name [default]

**Description**

Although a terminating NUL character essentially need not be included in the host name reported to the DHCP server, since many DHCP clients that are generally used today include a NUL character, a configuration that matches this situation is the default. As long as no particular problem occurs, leave the default unchanged.

**dhcp\_release\_on\_stop**

This is a keyword for specifying whether or not to report the release of the IP address to the DHCP server when an interface goes down.

**Syntax**

Any of the following can be specified.

- `-dhcp_release_on_stop` Release of IP address is not reported [default]
- `dhcp_release_on_stop` Release of IP address is reported

**address**

This is a keyword for specifying the IP address.

**Syntax**

`[-]address [<address>]`

- `-address` The IP address is not directly specified [default]
- `address address` The IP address is set to “*address*”

For information about the syntax of “*address*” see the description of `scelnetName2Address()`.

**Description**

If DHCP is not used and the connection is not a PPP connection, the IP address must be specified directly.

When DHCP is used, even if “*address address*” is specified, the directly specified address is overwritten by the IP address that was assigned by DHCP.



## netmask

This is a keyword for specifying the subnet mask.

### Syntax

[**-**]netmask [**<**netmask**>**]

- **-netmask** The subnet mask is not specified directly [default]
- **netmask *netmask*** The subnet mask is set to "*netmask*"

For information about the syntax of "*netmask*" see the description of `scelnetName2Address()`.

### Description

If DHCP is not used and the connection is not a PPP connection, the subnet mask must be specified directly. However, If the subnet mask is Class A, B, or C, the subnet mask is automatically set by INET.

When DHCP is used, even if "**netmask *netmask***" is specified, the directly specified subnet mask is overwritten by the subnet mask that was assigned by DHCP.

## nameserver

This is a keyword for adding or deleting a name server.

### Syntax

nameserver [**add** | **del**] *address*

## route

This is a keyword for adding or deleting routing information.

### Syntax

Any of the following can be specified.

- **route add *address* [gw *address*] [netmask *netmask*]**  
Adds the specified routing information.
- **route add default gw *address***  
Adds the default gateway.
- **route del *address***  
Deletes the specified routing information.
- **route add default**  
Adds the default routing information.

## phy\_config

This is a keyword for specifying the method of configuring a physical layer chip. It is valid only for an Ethernet-connection ATTACH\_CNF(ATTACH\_CNF for which type eth is specified).

**Syntax**

Any of the following can be specified.

- `-phy_config`                      The configuration method for the physical layer chip is not specified [default]
- `phy_config auto`                  Auto Negotiation Mode
- `phy_config 10`                     10BaseT, Half-Duplex
- `phy_config 10_fd`                  10BaseT, Full-Duplex, No-Flow-Control
- `phy_config 10_fd_pause`        10BaseT, Full-Duplex, Flow-Control
- `phy_config tx`                     100BaseTX, Half-Duplex
- `phy_config tx_fd`                  100BaseTX, Full-Duplex, No-Flow-Control
- `phy_config tx_fd_pause`        100BaseTX, Full-Duplex, Flow-Control

**mtu**

Specifies the default values for `want.mru`, `allow.mru` for the PPP and PPPoE(PPP over Ethernet) interfaces, and the value of the MTU for the Ethernet interface.

**Syntax**

`mtu mtu`

**Description**

For PPP or PPPoE(PPP over Ethernet), the default values for `want.mru` as well as `allow.mru` are specified in `mtu`. If `want.mru`, `allow.mru` have been set, the setting of `want.mru`, `allow.mru` takes precedence. The MTU value of the Ethernet interface is specified in `mtu`. If `mtu` is not specified, it takes a value of 1500 for the PPP and Ethernet interfaces, and a value of 1492 for PPPoE(PPP over Ethernet).

**pppoe**

Specifies whether or not PPPoE(PPP over Ethernet) is enabled.

**Syntax**

`[-]pppoe`

**Description**

If `pppoe` is specified, PPPoE(PPP over Ethernet) will be used. If `-pppoe` is specified, PPPoE(PPP over Ethernet) will not be used. The following keywords beginning with `pppoe` are valid only when PPPoE(PPP over Ethernet) is used. In all other cases, they are ignored even if they have been specified.

**pppoe\_service\_name**

Specifies the service name for PPPoE(PPP over Ethernet).

**Syntax**

```
pppoe_service_name service_name
```

**Description**

If *service\_name* is not specified, it will be empty.

**pppoe\_ac\_name**

Specifies the access concentrator name for PPPoE(PPP over Ethernet).

**Syntax**

```
pppoe_ac_name ac_name
```

**Description**

If *ac\_name* is not specified, it will be empty.

**pppoe\_host\_uniq\_auto**

Specifies whether host unique data should be automatically created to distinguish clients for PPPoE(PPP over Ethernet).

**Syntax**

```
[-]pppoe_host_uniq_auto
```

**Description**

If *pppoe\_host\_uniq\_auto* is specified, host unique data is created automatically and reported to the server. *pppoe\_host\_uniq\_auto* is specified by default. When *-pppoe\_host\_uniq\_auto* is specified, host unique data will not be reported to the server.

**chat\_additional**

This is a keyword for defining an additional script for the modem. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

```
chat_additional “chat”
```

**Description**

This is a keyword for defining an additional script for the modem for a PPP interface. It is used for user-specific support such as adjusting the modem transmission level.

Although the default additional script is defined in NET\_CNF, ATTACH\_CNF defines an additional script that depends on an individual interface. If an additional script is defined in ATTACH\_CNF, any additional script that was defined in NET\_CNF is ignored.

**redial\_count**

This is a keyword for specifying the upper limit of the redial count. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

`redial_count redial_count`

**Description**

This is a keyword for specifying the maximum number of times redialing is to be attempted when “busy” is returned by the modem driver in a PPP interface. The default when no `redial_count` is specified is 0, and no redialing is performed.

Although the default redial count is set in NET\_CNF, ATTACH\_CNF sets a redial count that depends on an individual interface. If the redial count is set in ATTACH\_CNF, any redial count that was set in NET\_CNF is ignored.

**redial\_interval**

This is a keyword for specifying the redial interval. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

`redial_interval redial_interval`

**Description**

This is a keyword for specifying the interval in seconds to wait before redialing when “busy” is returned by the modem driver. The default when no “redial\_interval” is specified is 10 seconds.

Although the default redial interval is set in NET\_CNF, ATTACH\_CNF sets a redial interval that depends on an individual interface. If the redial interval is set in ATTACH\_CNF, any redial interval that was set in NET\_CNF is ignored.

**outside\_delay**

This is a keyword for specifying the outside line calling delay string. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

`outside_delay “outside_delay”`

**Description**

This is a keyword for specifying the outside line calling delay string that is to be inserted between the outside line calling string and the telephone number when outside line calling is used. Although this depends on the model and configuration of the modem, for many AT command modems, a comma “,” means a delay of approximately 2 seconds. The default when no “outside\_delay” is specified is “” (empty string).

Although the default outside line calling delay string is set in NET\_CNF, ATTACH\_CNF sets an outside line calling delay string that depends on an individual interface. If the outside line calling delay string is set in ATTACH\_CNF, any outside line calling delay string that was set in NET\_CNF is ignored.

The outside line calling delay string is not used when the outside line calling string is an empty string.

# dialing\_type

This is a keyword for specifying the line type. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

## Syntax

dialing\_type [tone | pulse | any]

## Description

The line type is specified with the following arguments.

Table 4-7

Argument	Meaning
tone	Tone line (analog) [default]
pulse	Pulse line (analog)
any	Line other than the above (such as digital)

Although the default line type is set in NET\_CNF, ATTACH\_CNF sets a line type that depends on an individual interface. If the line type is set in ATTACH\_CNF, any line type that was set in NET\_CNF is ignored.

# phone\_number

This is a keyword for specifying the calling destination telephone number. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

## Syntax

phone\_number        “*phone\_number*”  
 phone\_number0      “*phone\_number*”  
 phone\_number1      “*phone\_number*”  
 (omitted)  
 phone\_number9      “*phone\_number*”

## Description

This is a keyword for specifying the calling destination telephone number.

phone\_number0 to phone\_number9 indicate the calling destination priority order, that is, the redialing order. Redial processing is performed in the following order. First, an attempt is made to call the telephone number specified by phone\_number0. If phone\_number0 has not been specified or is busy, an attempt is made to call the telephone number specified by phone\_number1, and so on. (If only one telephone number is specified, that same number is redialed repeatedly.)

phone\_number is equivalent to phone\_number0.

**answer\_mode**

This is a keyword for specifying whether or not to establish a network connection due to an incoming call. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

Any of the following can be specified.

- `-answer_mode` Connection is established due to calling [default]
- `answer_mode` Connection is established due to answering

**answer\_timeout**

This is a keyword for specifying the timeout interval when the connection is established due to answering. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

`answer_timeout` *answer\_timeout*

**Description**

This is a keyword for specifying the timeout interval in seconds when the connection is established due to answering. If no “answer\_timeout” is specified, the timeout interval will be the same as the calling timeout interval specified by the `connect_timeout` keyword.

**chat\_login**

This is a keyword for specifying the script string that specifies the login procedure. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

`chat_login` “*chatscript*”

**Description**

The default when no “chatscript” is specified is “”, which means that no login procedure is executed.

For information related to the specifications for this chat script, see “Chat Script Specifications”.

**auth\_name, auth\_key, peer\_name, and peer\_key**

These are keywords for specifying the authentication name and authentication key. They are valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

`auth_name` “*auth\_name*”

`auth_key` “*auth\_key*”

`peer_name` “*peer\_name*”

`peer_key` “*peer\_key*”

### Description

The user name and password that are assigned by a general ISP correspond to `auth_name` and `auth_key`. `peer_name` and `peer_key` are the authentication name and authentication key of the connection destination. No default setting exists for any of these keywords. Each parameter is used as follows according to the authentication method.

- When the remote side issues a PAP authentication request to the local side (remote side is the server)
  - `auth_name` Sent as authentication name
  - `auth_key` Sent as authentication key
- When the local side issues a PAP authentication request to the remote side (remote side is the server)
  - `peer_name` Compared with received authentication name
  - `peer_key` Compared with received authentication key
- When the remote side issues a CHAP authentication request to the local side (local side is the server)
  - `auth_name` Authentication name to be sent in Response
  - `auth_key` Authentication key to be used in Response digest
  - `peer_name` Compared with remote side name that is sent in Challenge (if `peer_name` is "\*", no comparison is performed)
- When the local side issues a CHAP authentication request to the remote side (local side is the server)
  - `peer_name` Authentication name to be used for Response verification
  - `peer_key` Authentication key to be used for Response verification
  - `auth_name` Local authentication name to be sent in Challenge

### want.auth

This is a keyword for specifying the authentication method for issuing a request to the remote side. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which "type ppp" is specified).

### Syntax

`want.auth [ any | pap | chap | pap/chap | chap/pap ]`

### Description

This keyword specifies the authentication method for issuing a request to the remote side.

Table 4-8

Argument	Authentication Method
any	Authentication by PAP or CHAP is not requested [default]
pap	Only authentication by PAP is requested
chap	Only authentication by CHAP is requested
pap/chap	First authentication by PAP is requested, and if the resulting connection is rejected, then authentication by CHAP is requested next
chap/pap	First authentication by CHAP is requested, and if the resulting connection is rejected, then authentication by PAP is requested next

The algorithm for requesting CHAP authentication is MD5.

**allow.auth**

This is a keyword for specifying the authentication method to be allowed by the local side. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

```
allow.auth [ any | pap | chap | pap/chap | chap/pap ]
```

**Description**

This is a keyword for specifying the authentication method to be allowed when authentication is requested by the local side.

**Table 4-9**

Argument	Authentication Method
any	Authentication by PAP or CHAP is not allowed [default]
pap	Only authentication by PAP is allowed
chap	Only authentication by CHAP is allowed
pap/chap	Authentication by PAP or CHAP is allowed
chap/pap	Authentication by PAP or CHAP is allowed

pap/chap and chap/pap have the same meaning. The three algorithms that are allowed for CHAP authentication are MD5, MS (Version 1), and MS (Version 2).

**lcp\_timeout**

This is a keyword for specifying the timeout interval for LCP. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

```
lcp_timeout sec
```

**Description**

This is a keyword for specifying the timeout interval for LCP in seconds. The default when no timeout interval is specified is one second.

This timeout interval will be the resend interval when there is no response to a request.

**ipcp\_timeout**

This is a keyword for specifying the timeout interval for IPCP. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

```
ipcp_timeout sec
```

**Description**

This is a keyword for specifying the timeout interval for IPCP in seconds. The default when no timeout interval is specified is one second.

This timeout interval will be the resend interval when there is no response to a request.



## idle\_timeout

This is a keyword for specifying the PPP idle timeout interval. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

### Syntax

idle\_timeout sec

### Description

The timeout interval in seconds is specified for the argument. PPP automatically disconnects the communication with the remote destination if there are no sends or receives during the time that the number of seconds specified here elapses. The default when no timeout interval is specified is 90 seconds.

## connect\_timeout

This is a keyword for specifying the line connection timeout interval. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

### Syntax

connect\_timeout sec

### Description

A timeout occurs if the modem does not report a connection completion within the number of seconds specified here from the time when the chat\_dial and chat\_login processing is completed. The default when no timeout interval is specified is 10 seconds.

## mru\_nego and mru

These are keywords for configuring the MRU (Maximum-Receive-Unit: Maximum receive data length) negotiation. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

### Syntax

[*-*]want.mru\_nego

[*-*]allow.mru\_nego

want.mru *mru*

allow.mru *mru*

### Description

Whether or not MRU negotiation is requested is specified with want.mru\_nego, and whether or not MRU negotiation is allowed is specified with allow.mru\_nego. Appending “-” at the beginning of each keyword means that negotiation is not requested or allowed. Not appending “-” means that negotiation is requested or allowed.

The maximum receive data length is specified for want.mru and allow.mru. The default value when either of these data lengths is not specified is 1500 bytes.

**accm\_nego and accm**

These are keywords for configuring ACCM (Async-Control-Character-Map: Asynchronous control character map) negotiation. They are valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

```
[~]want.accm_nego
[~]allow.accm_nego
want.accm_value accm
allow.accm_value accm
```

**Description**

Whether or not ACCM negotiation is requested is specified with want.accm\_nego, and whether or not ACCM negotiation is allowed is specified with allow.accm\_nego. Appending “-” at the beginning of each keyword means that negotiation is not requested or allowed. Not appending “-” means that negotiation is requested or allowed.

The accm value is specified for want.accm and allow.accm. The default value when these accm values are not specified are 0 for want.accm and 0xffffffff for allow.accm.

**magic\_nego**

This is a keyword for configuring magic number negotiation. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

```
[~]want.magic_nego
[~]allow.magic_nego
```

**Description**

Whether or not magic number negotiation is requested is specified with want.magic\_nego, and whether or not magic number negotiation is allowed is specified with allow.magic\_nego. Appending “-” at the beginning of each keyword means that negotiation is not requested or allowed. Not appending “-” means that negotiation is requested or allowed.

**prc\_nego**

This is a keyword for configuring PRC (Protocol-Field-Compression) negotiation. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

**Syntax**

```
[~]want.prc_nego
[~]allow.prc_nego
```

**Description**

Whether or not PRC negotiation is requested is specified with want.prc\_nego, and whether or not PRC negotiation is allowed is specified with allow.prc\_nego. Appending “-” at the beginning of each keyword means that negotiation is not requested or allowed. Not appending “-” means that negotiation is requested or allowed.

# acc\_nego

This is a keyword for configuring ACC (Address-and-Control-Field-Compression) negotiation. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

## Syntax

[*-*]want.acc\_nego  
 [*-*]allow.acc\_nego

## Description

Whether or not ACC negotiation is requested is specified with want.acc\_nego, and whether or not ACC negotiation is allowed is specified with allow.acc\_nego. Appending “*-*” at the beginning of each keyword means that negotiation is not requested or allowed. Not appending “*-*” means that negotiation is requested or allowed.

# address\_nego, ip\_address, and ip\_mask

These are keywords for configuring IP address and subnet mask negotiation. They are valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

## Syntax

[*-*]want.address\_nego  
 [*-*]allow.address\_nego  
 want.ip\_address *address*  
 want.ip\_mask *netmask*  
 allow.ip\_address *address*  
 allow.ip\_mask *netmask*

## Description

Whether or not IP address negotiation is requested when a PPP connection is established is specified with want.address\_nego, and whether or not IP address negotiation is allowed is specified with allow.address\_nego. Appending “*-*” at the beginning of each keyword means that negotiation is not requested or allowed. Not appending “*-*” means that negotiation is requested or allowed.

The IP address is determined as follows according to the address\_nego, ip\_address, and ip\_mask specifications and negotiation results.

Table 4-10

Negotiation Request	Negotiation Result	Local IP Address
Request negotiation want.address_nego	Ack	want.ip_address is used
Request negotiation want.address_nego	Nak	The logical AND of the assigned IP address and want.ip_mask is taken, and if that matches the logical AND of want.ip_address and want.ip_mask, that IP address is used
Do not request negotiation -want.address_nego	--	want.ip_address is used

Table 4-11

Negotiation Request	Negotiation Result	Remote IP Address
Allow negotiation allow.address_nego	Ack	allow.ip_address is used
Allow negotiation allow.address_nego	Nak	The logical AND of the IP address that was sent and allow.ip_mask is taken, and if that matches the logical AND of allow.ip_address and allow.ip_mask, that IP address is used
Do not allow negotiation - allow.address_nego	--	allow.ip_address is used

The default value of want.ip\_address or allow.ip\_address is “0.0.0.0”, and the default value of want.ip\_mask and allow.ip\_mask is “0.0.0.0”.

## **vjcomp\_nego**

This is a keyword for configuring negotiation related to VJ compression. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

### **Syntax**

[**-**]want.vjcomp\_nego

[**-**]allow.vjcomp\_nego

### **Description**

Whether or not VJ compression negotiation is requested is specified with want.vjcomp\_nego, and whether or not VJ compression negotiation is allowed is specified with allow.vjcomp\_nego. Appending “-” at the beginning of each keyword means that negotiation is not requested or allowed. Not appending “-” means that negotiation is requested or allowed.

## **dns1\_nego, dns1, dns2\_nego, and dns2**

These are keywords for configuring negotiation related to the name server. They are valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

### **Syntax**

[**-**]want.dns1\_nego                      [**-**]want.dns2\_nego

[**-**]allow.dns1\_nego                      [**-**]allow.dns2\_nego

want.dns1 *address*                      want.dns2 *address*

allow.dns1 *address*                      allow.dns2 *address*

### **Description**

Whether or not name server negotiation is requested is specified with want.dns1\_nego or want.dns2\_nego, and whether or not name server negotiation is allowed is specified with allow.dns1\_nego or allow.dns2\_nego. Appending “-” at the beginning of each keyword means that

negotiation is not requested or allowed. Not appending “-“ means that negotiation is requested or allowed.

The IP address of the name server is specified for want.dns1, want.dns2, allow.dns1, and allow.dns2.

Although an unlimited number of DNS addresses can be set in INET, at most two DNS addresses per network interface can be reported from the PPP layer to the INET layer.

### force\_chap\_type

This is a keyword for limiting the CHAP authentication algorithms that are to be allowed. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

#### Syntax

force\_chap\_type [no | md5 | ms | ms-v1 | ms-v2]

#### Description

This is a keyword for limiting the algorithms when CHAP authentication is allowed. The meanings of the arguments are as follows.

Table 4-12

Argument	Meaning
no	Authentication algorithms are not limited [default]
md5	Limited to only MD5
ms	Limited to only MS (Version 1)
ms-v1	Limited to only MS (Version 1)
ms-v2	Limited to only MS (Version 2)

When this keyword is not specified, the authentication algorithms are not limited.

This keyword is provided for forcing a specific algorithm to be used when the connection cannot be made by using the authentication algorithm that was requested by the remote side.

### omit\_empty\_frame

This is a keyword for controlling the transmission of empty frames. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

#### Syntax

[-]omit\_empty\_frame

#### Description

If omit\_empty\_frame is specified, an empty frame (Flag-only frame) is not added immediately after a PPP frame. If -omit\_empty\_frame is specified by appending “-“ at the beginning of the keyword, an empty frame is added. When this keyword is not specified, an empty frame is added.

During reception, data can be received regardless of whether or not an empty frame exists.

## log\_flags

This is a keyword for specifying the PPP log display contents. It is valid only for a PPP-connection ATTACH\_CNF (ATTACH\_CNF for which “type ppp” is specified).

### Syntax

log\_flags *flag*...

### Description

This is a debugging keyword that specifies the PPP log display contents using inetlog.c. Any of the following can be specified for “*flag*”. The default value when this keyword is not set is “phase cp auth chat”.

Table 4-13

<i>flag</i>	Log Display Contents
phase	PPP state
cp	LCP and IPCP states
auth	PAP and CHAP states
chat	Chat processing and its responses
private	Private information within the chat script
dll	The exchange of data at the DLL level
dump	DLL data packet dump
timer	Timer-related information
event	Events sent to PPP

Note: Each of these flags can also be specified with a numeric value.

## DIAL\_CNF Keywords

DIAL\_CNF is a configuration file for specifying modem-dependent settings using general parameters of a PPP connection.

### dialing\_type\_string

This is a keyword for defining the dialing type string.

### Syntax

dialing\_type\_string “*tone*” “*pulse*” “*any*”

### Description

This keyword defines the contents of the special string \D that is used in a chat script. Any of the values “*tone*”, “*pulse*”, and “*any*” that are specified here will be assigned for the special string \D according to the line type that is used by the user.

For example, by specifying dialing\_type\_string “ATDT” “ATDP” “ATD”, \D will be replaced by “ATDT” when dialing\_type is tone within NET\_CNF.

**redial\_string**

This is a keyword for specifying the reserved string that is returned from the modem when the circuit is busy.

**Syntax**

```
redial_string "redial_string"
```

**Description**

This is a keyword for specifying the reserved string that is returned from the modem when the circuit is busy. With many AT command modems, this string is "BUSY".

**chat\_init**

This is a keyword for defining the initialization part of the script for the calling/answering procedure.

**Syntax**

```
chat_init "init_script"
```

**Description**

This keyword defines the script string that specifies the required initialization procedure for both calling and answering. It must contain a description that aborts processing if the string specified by redial\_string is received.

For information related to script string specifications, see "Chat Script Specifications."

**chat\_dial**

This is a keyword for defining the calling procedure.

**Syntax**

```
chat_dial "dial_script"
```

**Description**

This keyword defines the script string that specifies the calling procedure. For information related to script string specifications, see "Chat Script Specifications."

If -answer\_mode is specified in NET\_CNF or ATTACH\_CNF, the calling script is generated as follows.

```
init_script + additional_script + dial_script
```

**chat\_answer**

This is a keyword for defining the answering procedure.

**Syntax**

```
chat_answer "answer_script"
```

**Description**

This keyword defines the script string that specifies the answering procedure. For information related to script string specifications, see "Chat Script Specifications."

If answer\_mode is specified in NET\_CNF or ATTACH\_CNF, the answering script is generated as follows.

```
init_script + additional_script + "TIMEOUT answer_timeout" + dial_script
```

# Chat Script Specifications

A chat script is used to specify the following five procedures for a PPP connection.

- Modem initialization (chat\_init)  
Script for setting the timeout interval for a modem connection, setting the abort string, and specifying the modem initialization script.
- Modem additional script (chat\_additional)  
Script for performing user-specific configuration for the modem.
- Modem calling script (chat\_dial)  
Script for specifying calling processing. This script is supplied by the modem manufacturer together with the modem driver.
- Modem answering script (chat\_answer)  
Script for specifying answer processing. This script is supplied by the modem manufacturer together with the modem driver.
- Login procedure (chat\_login)  
Script used when a login procedure for exchanging strings without PPP authentication is required.

## Script Specifications

Each script is specified as a single string. Note that when you specify a script in a file, you generally must specify the slash “\” as an escaped double slash “\\” as is done when coding in C.

### General Syntax

Generally, strings for receiving and transmitting are specified alternately in a script, separated by spaces.

recv1 send1 recv2 send2

In this example, processing waits until “recv1” is received, then “send1” is sent. Next, processing waits until “recv2” is received, then finally “send2” is sent.

### Special Strings

Within a script, the following special characters are interpreted as if they were replaced with the corresponding entries shown below.

Table 4-14

Special Character	Meaning
\s	Space (SP:0x20)
\t	Tab (HT:0x09)
\r	Return (CR:0x0d)
\n	Line feed (LF:0x0a)
\U	Authentication name (auth_name)
\P	Authentication key (auth_key)
\T	Telephone number (phone_number)
\D	Line type-specific dialing string
^X	CTRL-X (0x00-0x1f)
\c	\r is not appended at the end of the string (transmission string only)



Special Character	Meaning
\d	Wait 2 seconds (delay) (transmission string only)
\p	Wait 0.25 seconds (pause) (transmission string only)
\X	Character X itself (special character escape: transmission string only)

### Empty String

To send strings without waiting at all, specify an empty string as the receive string. An empty string is specified by two consecutive double quotes.

### Timeout Interval Specification

The upper limit of the interval for waiting for a receive string is specified as follows.

TIMEOUT *n* (*n* is a decimal number, units: seconds)

The default value when no timeout interval is specified is 30 seconds. If the receive string cannot be received within the timeout interval, the entire PPP connection process is terminated with an error.

### Retry Specification

Retry processing to be performed if the receive string cannot be received within the timeout interval is specified as follows.

recv1-send1-recv2 send2

If "recv1" is received within the timeout interval, "send2" is sent directly.

If "recv1" was not received within the timeout interval, "send1" is sent and processing waits for "recv2" to be received. If "recv2" is received within the timeout interval, "send2" is sent. If it is not, processing terminates with an error.

### Abort String Specification

The following specification can cause processing to be terminated with an error when a given specified string is received.

ABORT *string*

## Chat Script Setup Examples

In the following setup examples, one pair consisting of receive and send strings has been specified per line. This simplifies verification and helps reduce specification errors.

Although a NET configuration file can be written directly as text, these scripts are shown in the specification format used in C-language programs to facilitate explanations.

```

chat_init =                                // < Initialization procedure >
TIMEOUT 60                                // Sets timeout interval to 60 seconds
" ABORT NO\\sCARRIER"                     // Terminates with an error if "NO CARRIER" is
received"
ABORT ERROR"                               // Terminates with an error if "ERROR" is received
" ABORT BUSY"                              // Terminates with an error if "BUSY" is received
" ABORT DELAYED"                           // Terminates with an error if "DELAYED" is received
" \\\" \\dA\\p"                             // Sends "AT\r" with a delay
" OK \\\";                                  // Waits for "OK"

chat_additional =                          // < Additional procedure >

```

```

        "";                                // Sets an empty string when nothing is to be done

chat_dial =                                // < Calling procedure >
    "\"\" \\D\\T"                          // Initiates a call
    " CONNECT \\d\\c";                    // Waits for "CONNECT" and then waits an additional
    2 seconds

chat_answer =                              // < Answering procedure >
    "RING ATA"                            // Waits for "RING" and then sends "ATA"
    " CONNECT \\d\\c";                    // Waits for "CONNECT" and then waits an additional
    2 seconds

chat_login =                              // < Login procedure >
    "TIMEOUT 10"                          // Sets timeout interval to 10 seconds
    " ogin:--ogin: \\U"                  // Waits for "ogin:" to be resent and sends
    authentication name
    " ssword: \\P"                        // Waits for "ssword:" and sends authentication key
    " PPP\\sstart";                      // Finally waits for "PPP start"

```

## Precautions

- As long as “\c” is not specified in the send string, “\r” will be automatically appended to the end when the string is sent. However, this kind of automatic appending processing is not performed with receive strings.
- The maximum length of a send or receive string is 255 bytes. There is no specific limit (in the NET configuration file specifications) on the length of the entire script.
- The receive string buffer is cleared when the send string is sent.

---

# Chapter 5:

## Network Device Library

---

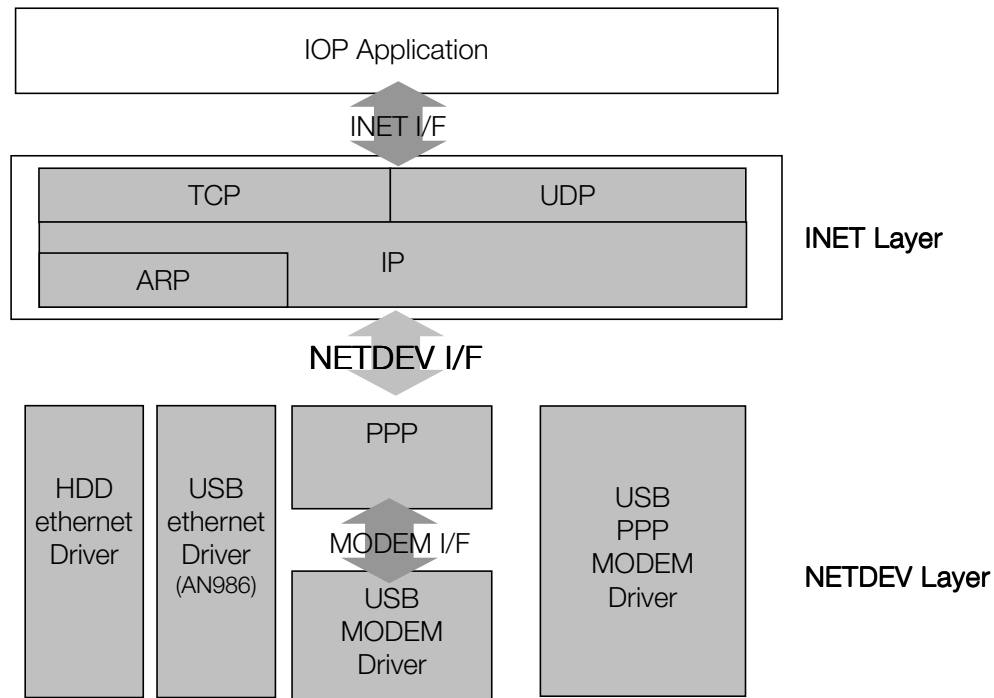
<b>Overview</b>	<b>5-3</b>
<b>Exchanges With the INET Layer</b>	<b>5-4</b>
scelnetDevOps_t	5-4
Start Use and Stop Use Functions	5-4
Notification of State Change by an Event Flag	5-4
<b>Packets and Packet Queues</b>	<b>5-6</b>
Packet and Packet Queue Structure	5-6
Packet and Packet Queue Operations	5-6
<b>Send/Receive Processing Overview</b>	<b>5-7</b>
Processing During Send	5-7
Processing During Receive	5-7
<b>Control Functions and Control Codes</b>	<b>5-8</b>
NETDEV Layer Common Control Codes	5-8
NETDEV Layer Module-Dependent Control Codes	5-9
PPP Control Codes	5-9
Ethernet Interface-Dependent Statistical Information	5-13
Ethernet Interface-Dependent Settings	5-13



## Overview

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

Figure 5-1



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

### scelnetDevOps\_t

The NETDEV layer module is recognized as a logical network interface by registering each `scelnetDevOps_t` structure in the INET layer via `scelnetRegisterNetDevice()`. The following information is contained in the `scelnetDevOps_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 `scelnetDevOps_t` structure was registered. For the second argument, specify any of the following values according to the circumstances to be reported.

Table 5-1

Constant	Value	Meaning
scelnetDevEFP_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.
scelnetDevEFP_PlugOut	0x00000002	Notification that data cannot be exchanged between the NETDEV layer and the device because the device was unplugged from the connector.
scelnetDevEFP_Recv	0x00000004	Notification that the received packet was added to the receive packet queue.
scelnetDevEFP_Error	0x00000010	Notification that an error (excluding a timeout) occurred for which send/receive could not continue for a reason other than a PlugOut.
scelnetDevEFP_TimeOut	0x00000020	Notification that a timeout error occurred for which send/receive could not continue for a reason other than a PlugOut.

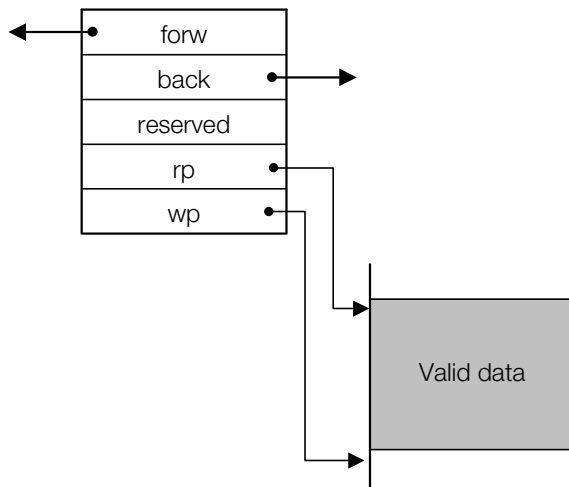
## Packets and Packet Queues

### 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.

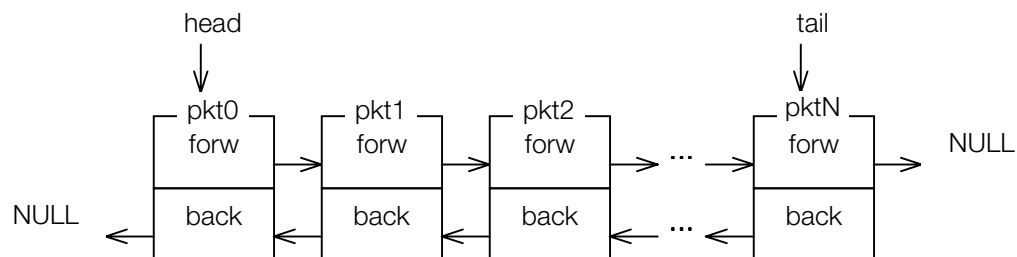
A packet is represented by a `scelnetPkt_t` structure, which has bidirectional links. A data area is established separately, and pointers pointing to the beginning and end of the valid data are updated during the progress of send/receive processing.

Figure 5-2



`scelnetPktQ_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 5-3



### 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 `scelnetPktDeQ()`
- Discard transmitted packet and free memory `scelnetFreePkt()`
- Allocate memory for a packet for collecting received data `scelnetAllocPkt()`
- Add a received packet to the end of the receive packet queue `scelnetPktEnQ()`

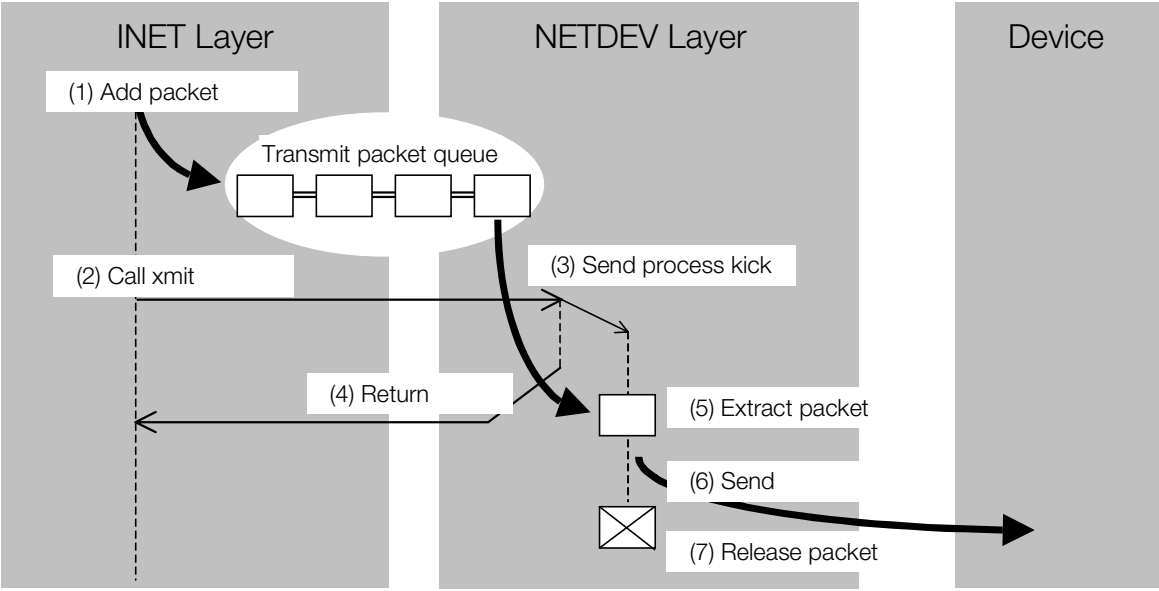


# Send/Receive Processing Overview

## 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.

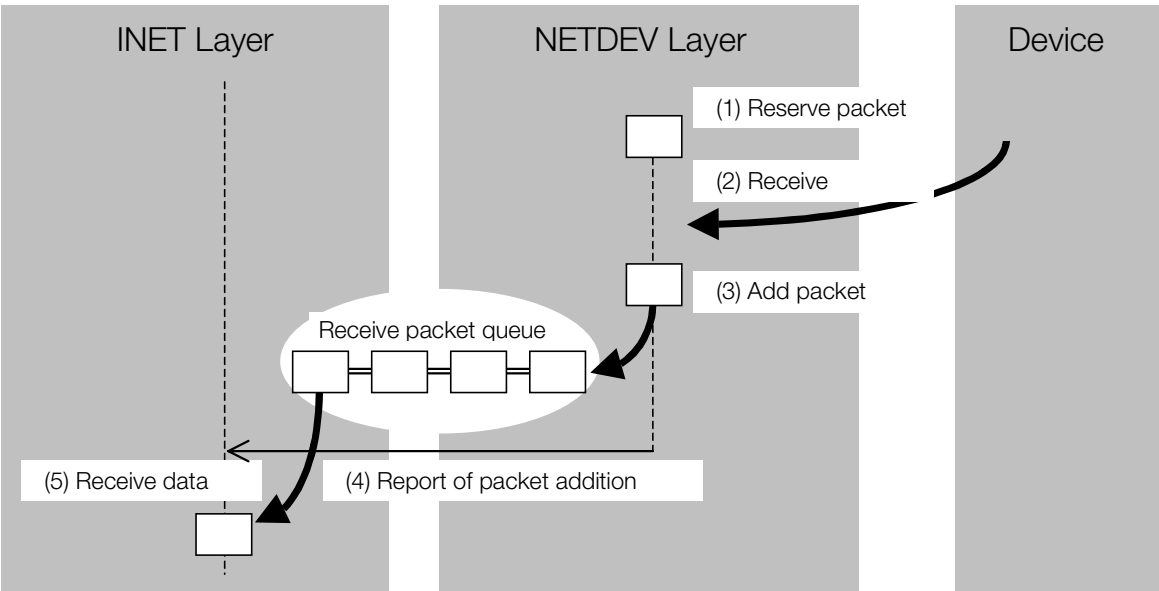
Figure 5-4



## 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 5-5



## Control Functions and Control Codes

The `scelnetInterfaceControl()` 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 5-2**

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, `scelnetInterfaceControl()` 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 5-3**

Control Code	Function
<code>scelnetNDCC_GET_THPRI</code>	Get thread priority
<code>scelnetNDCC_SET_THPRI</code>	Set thread priority
<code>scelnetNDCC_GET_IF_TYPE</code>	Get network interface type
<code>scelnetNDCC_GET_RX_PACKETS</code>	Get number of receive packets
<code>scelnetNDCC_GET_TX_PACKETS</code>	Get number of transmit packets
<code>scelnetNDCC_GET_RX_BYTES</code>	Get number of receive bytes
<code>scelnetNDCC_GET_TX_BYTES</code>	Get number of transmit bytes
<code>scelnetNDCC_GET_RX_ERRORS</code>	Get number of receive errors
<code>scelnetNDCC_GET_TX_ERRORS</code>	Get number of transmit errors
<code>scelnetNDCC_GET_RX_DROPPED</code>	Get number of dropped receive packets
<code>scelnetNDCC_GET_TX_DROPPED</code>	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 5-4**

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.irc 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 5-5**

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
scePPPCCC_GetAllowAccmNego	0x90008018	bool	allow.accm_nego
scePPPCCC_GetAllowMagicNego	0x90008019	bool	allow.magic_nego
scePPPCCC_GetAllowPrcNego	0x9000801a	bool	allow.prc_nego
scePPPCCC_GetAllowAccNego	0x9000801b	bool	allow.acc_nego
scePPPCCC_GetAllowAddressNego	0x9000801c	bool	allow.address_nego
scePPPCCC_GetAllowVjcompNego	0x9000801d	bool	allow.vjcomp_nego
scePPPCCC_GetAllowMruValue	0x9000801e	u_short	allow.mru_value
scePPPCCC_GetAllowAccmValue	0x9000801f	u_long	allow.accm_value
scePPPCCC_GetAllowAuth	0x90008020	u_char	allow.auth
scePPPCCC_GetAllowIpAddress	0x90008021	address	allow.ip_address
scePPPCCC_GetAllowIpMask	0x90008022	address	allow.ip_mask
scePPPCCC_GetWantDNS1Nego	0x90008023	bool	want.dns1_nego
scePPPCCC_GetWantDNS2Nego	0x90008024	bool	want.dns2_nego
scePPPCCC_GetWantDNS1	0x90008025	address	want.dns1
scePPPCCC_GetWantDNS2	0x90008026	address	want.dns2
scePPPCCC_GetAllowDNS1Nego	0x90008027	bool	allow.dns1_nego
scePPPCCC_GetAllowDNS2Nego	0x90008028	bool	allow.dns2_nego
scePPPCCC_GetAllowDNS1	0x90008029	address	allow.dns1
scePPPCCC_GetAllowDNS2	0x9000802a	address	allow.dns2
scePPPCCC_GetConnectTimeout	0x9000802b	time	connect_timeout
scePPPCCC_GetDialingType	0x9000802c	u_char	dialing_type
scePPPCCC_GetToneDial	0x9000802e	string	dialing_type_string (tone)
scePPPCCC_GetPulseDial	0x9000802f	string	dialing_type_string (pulse)
scePPPCCC_GetAnyDial	0x90008030	string	dialing_type_string (any)
scePPPCCC_GetLogFlags	0x90008033	int	log_flags
scePPPCCC_GetHisAddress	0x90008034	address	---
scePPPCCC_GetHisMask	0x90008035	address	---
scePPPCCC_GetOmitEmptyFrame	0x90008036	bool	omit_empty_frame
scePPPCCC_GetChapType	0x90008037	int	chap_type
scePPPCCC_GetCurrentStatus	0x90008038	string	---
scePPPCCC_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 5-6

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 scelnetAddress_t	sizeof (scelnetAddress_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.

#### scePPPC\_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.

#### scePPPC\_GetToneDial, scePPPC\_GetPulseDial, and scePPPC\_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.

#### scePPPC\_GetDialingType

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

Table 5-7

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 5-8**

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 5-9**

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 5-10**

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 5-11**

Constant	Value
scelnetNDCC_GET_NEGO_MODE	0x80020000
scelnetNDCC_SET_NEGO_MODE	0x81020000

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****scelnetNDCC\_SET\_NEGO\_MODE**

scelnetNDCC\_GET\_NEGO\_MODE sets the current PHY chip configuration information held by the NETDEV layer, in the area specified by (ptr, len=4).

scelnetNDCC\_SET\_NEGO\_MODE sets the configuration information specified by (ptr, len=4) as the current PHY chip configuration information that the NETDEV layer holds.

scelnetNDCC\_SET\_NEGO\_MODE has been made publicly available only for the purpose of creating the Ethernet driver itself. Setting this control code directly from a title is not allowed. However, if you want to set this control code from a title, set it via inetctl.irx using the phy\_config keyword in the configuration file.

The configuration information is the bitwise OR of the following values.

**Table 5-12**

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 Ethernet operating mode auto-sense function determines the operating mode. In this case, the remaining bits are used for negotiation.

When ScelnetNDNEGO\_AUTO is OFF, the operating mode is determined with the remaining bit specifications without using the Ethernet operating mode auto-sense function. The valid specifications are as follows.

**Table 5-13**

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.



**scelnetNDCC\_GET\_LINK\_STATUS**

This control code sets the current PHY chip Hub + Link information held by the NETDEV layer, in the area specified by (ptr, len=4). Link information will have one of the following values.

**Table 5-14**

Value	Link Information
-1	Link state is pending
0	No Link
1	Link exists

The pending state indicated by -1 only occurs right after the NETDEV layer is started. The state will transition to 0 or 1 within at most 10 seconds. Except for the case immediately after startup, the maximum delay time for reflecting the transition of the Link state is 10 seconds.

When "No link" is reported, it means that either the Ethernet cable is not connected or the Hub has not been powered on.

**scelnetNDCC\_SET\_MULTICAST\_LIST**

This control code specifies the multicast address list that must be received by the NETDEV layer.

**Table 5-15**

Condition	Meaning
(ptr != NULL) && (0 < len) && ((len % 6) == 0)	All multicast addresses included in the specified area are set to receivable. Multiple addresses can be specified simultaneously.
((len % 6) != 0)    (ptr != NULL && len < 0)	An error (scelNETE_INVALID_ARGUMENT) is returned.
len == 0	All multicast addresses except broadcast are set to nonreceivable.
ptr == NULL && len < 0	All multicast addresses are set to receivable.

A transmission to a multicast address can always be sent regardless of this scelnetNDCC\_SET\_MULTICAST\_LIST control code.

Whether or not a multicast address must be received is generally determined according to a mechanism that uses CRC and a hash. Since the size of the hash table is limited, as long as even one multicast address has been set, a multicast address may be received even if it is not the one that has been specified. For example, if the number of hash table entries is 64 and only one multicast address has been set, the probability that an arbitrary multicast address will be received is 1/64.

Since the NETDEV layer does not save the multicast address internally, the value within the area specified by (ptr, len) also need not be maintained after the call.

When the number of multicast addresses that are specified is large relative to the number of hash entries (when it exceeds 1/2), the NETDEV layer may behave in a manner as if all multicast addresses received were specified (this is implementation dependent).

