Network Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 About This Document

Overview

This document provides an overview of the network features.

The supported devices, the system software that forms the base of the network, and the library configuration are described in the "Network Feature Configuration" chapter. Subsequent chapters cover the communication procedure details and debug-related matters.

2 Network Feature Configuration

Device Configuration

PlayStation®Vita and PlayStation®TV - including their development machines - have Wi-Fi installed as a network device. Moreover, Development Kits (DevKit hereafter) and Testing Kits (TestKit hereafter) also support USB Ethernet devices.

In addition, some retail units are also installed with the 3G device and wired Ethernet device.

Hard Ware	Wi-Fi Device	USB Ethernet Device	Wired Ethernet Device	3G Device
PlayStation®Vita - PCH-1000 series - PCH-2000 series	Yes	No	No	No
PlayStation®Vita - PCH-1100 series	Yes	No	No	Yes
PlayStation®TV	Yes	No	Yes	No
DevKit/TestKit	Yes	Yes	No	No

Wi-Fi Device

IEEE 802.11n compliant devices

USB Ethernet Device

USB Ethernet (100 Mbps) devices are devices that perform on a substitute basis the communication features normally assumed by Wi-Fi devices, realizing stable communication on a DevKit/TestKit, and are offered in order to solve restrictions on the in-house use of Wi-Fi.

Wired Ethernet Device

Internet communication is carried out via a wired Ethernet circuit.

Regarding Internet connection for PlayStation®TV, refer to the "PlayStation®TV Internet Connection" section.

3G Device

3G devices perform Internet communication through a 3G line.

Switching of Wi-Fi Device and USB Ethernet Device

On the system software level, upon detection of a USB Ethernet device, the system automatically switches to USB Ethernet use.

Therefore, when a USB Ethernet device is used, begin use with the USB Ethernet device connected to the DevKit/TestKit.

Switching of Wi-Fi Device and 3G Device

In the retail units, the system software will automatically determine whether to use the Wi-Fi or the 3G device when using the Internet communication mode described below.

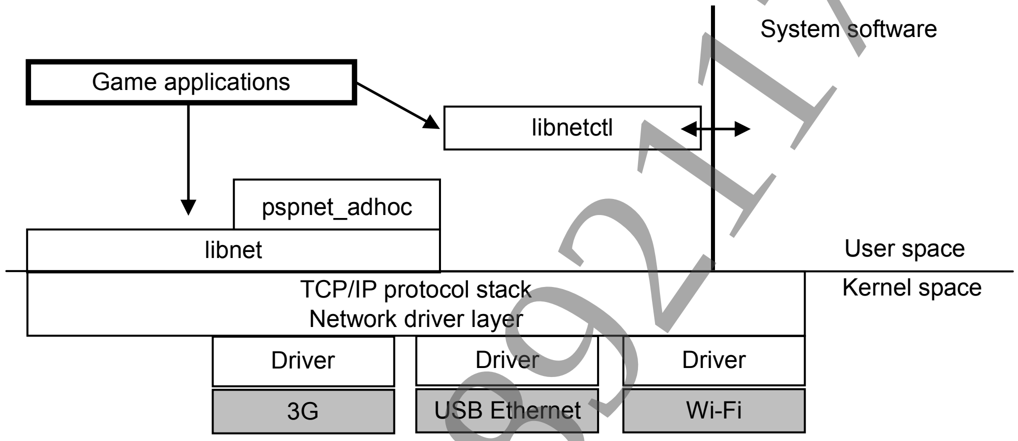
Library Configuration

The following libraries are provided regarding the network features. For details, refer to the overview and the reference of each library.

Library Name	Description
libnet	Features used as the base for socket communication, etc.
libnetctl	Features for acquiring network connection information, and that support connection for Internet communication and ad hoc communication.
pspnet_adhoc	Communication feature with PSP™ (PlayStation®Portable) compatibility

Feature Configuration Diagram

The system software configuration regarding the network features is as follows.



3 Communication Modes

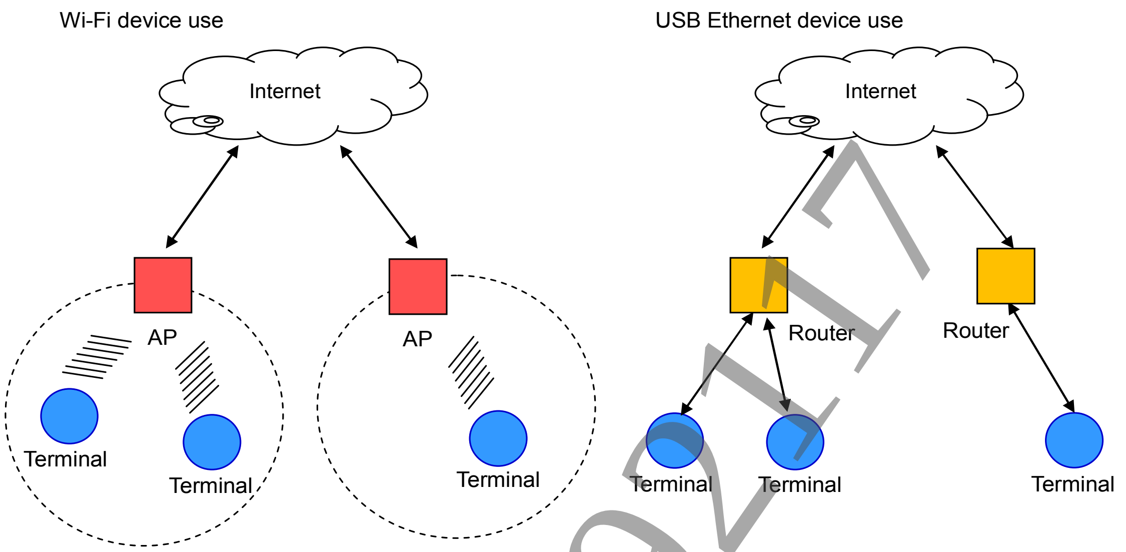
The PlayStation®Vita offers three Wi-Fi communication modes, the Internet communication mode, the ad hoc communication mode and the PSPNET ad hoc communication mode which has compatibility with the PSP™ ad hoc mode. Use the appropriate mode according to the application purposes. Communication for the Internet communication mode and the ad hoc communication mode is based on IP addresses. Therefore, in either case, applications can employ the TCP/IP protocol using socket function communication. In the PSPNET ad hoc communication mode, communication is based on MAC addresses. The communication processing uses PTP/PDP protocols exclusive to PSPNET ad hoc communication.

Communication Mode	Protocol	Communication with PlayStation®Vita Units	Communication with Other than PlayStation®Vita and PSP™ Units	Communication with PSP™ Units
Internet communication	TCP/IP	Yes	Yes	No
Ad hoc communication	TCP/IP	Yes	No	No
PSPNET ad hoc communication	PTP/PDP	Yes	No	Yes

Note that these communication modes operate exclusively and cannot be used at the same time.

3 Internet Communication Mode

The Internet communication mode is used to connect to a router or access point and perform Internet communication via this router or access point. An outline diagram is shown below. ("Terminal" when using a USB Ethernet device refers to a DevKit/TestKit.)



Characteristics

When using the Internet communication mode, there is no need to explicitly perform connection processing on the application level. The connection processing is taken care of on the system software level. For the connection processing, the system software checks whether communication is possible and outputs dialog boxes as needed.

Network Setting

To use the Internet communication mode, it is necessary to make the network setting beforehand. The network setting can be made from PlayStation®Vita Settings.

Characteristics

PlayStation®Vita supports multiple network settings for Internet communication (up to 30 are supported in the current SDK). At the time of the connection, based on the neighboring access points or the signal strength, the appropriate setting is automatically selected and the connection is made on the system software level.

Network Setting Cautions

- USB Ethernet use
  During connection, if an USB Ethernet device is inserted and a setting to use a wired connection exists, the wired connection setting is always used for the connection. To use the USB Ethernet, insert the device in the DevKit/TestKit beforehand, and connect an Ethernet cable to a connectable router or access point.

- Multiple wired connection settings
  Multiple wired connection settings can be made, but since it is not possible to select the correct connection setting from among multiple connection settings, this is meaningless. Actually even in such a case, which connection setting is selected is undefined during system software connection. Thus do not make multiple wired connection settings.

Network Setting Contents

Various setting items are included in one network setting. The network setting items are listed in the table below.

Table 1 Network Setting Items

Item	Description
Connection Name	Name assigned to each network setting
MTU	Interface MTU
Networking Device	Device used for Internet communication. Can be either wireless (Wi-Fi) or wired (USB Ethernet).
SSID	SSID of connection destination access point
Wi-Fi Security	Security type during wireless connection Either None, WEP, or WPA-PSK/WPA2-PSK
WEP Key	WEP Key
WPA Key	WPA Key
IP Address Setting	IP address acquisition method Either automatic acquisition (DHCP), manual, or PPPoE.
DHCP Host Name	Host name used for DHCP (Normally this item does not need to be set.)
PPPoE User ID	User ID used for PPPoE
PPPoE Password	Password used for PPPoE
IP Address	IP Address
Subnet Mask	Subnet Mask
Default Router	Default Router
DNS Setting	DNS setting method Either automatic acquisition or manual
Primary DNS	Primary DNS
Secondary DNS	Secondary DNS
Proxy Server	HTTP proxy server setting (Either use or don't use)
Proxy Server Address	Hostname of HTTP proxy server
Proxy Server Port Number	Port number of HTTP proxy server

Internet Communication Procedure

Preliminary Preparations

To operate applications that use Internet communication with the USB Ethernet, insert the USB Ethernet in the correct location of the DevKit/TestKit beforehand, and connect the Ethernet cable to a router or hub that supports Internet communication. Also, make the network settings for using a wired connection.

Processing Procedure

(1) Module loading
Call `sceSysmoduleLoadModule()` to load `SCE_SYSMODULE_NET`.

(2) Initialization
Call `sceNetInit()` to initialize libnet. `sceNetInit()` returns 0 when initialization has been successful. Next, call `sceNetCtlInit()` to initialize libnetctl. `sceNetCtlInit()` also returns 0 when initialization has been successful.

(3) Internet communication processing
Once initialization is completed, the Internet communication processing can be called at this point. Actually, in the disconnected state, the connection processing is included in the first Internet communication processing.

(4) Use termination
When Internet communication is no longer needed, call `sceNetCtlTerm()` and `sceNetTerm()` to end libnetctl and libnet.

(5) Module unloading
Call `sceSysmoduleUnloadModule()` and unload `SCE_SYSMODULE_NET`.

Acquisition of Connection State (libnetctl Feature)

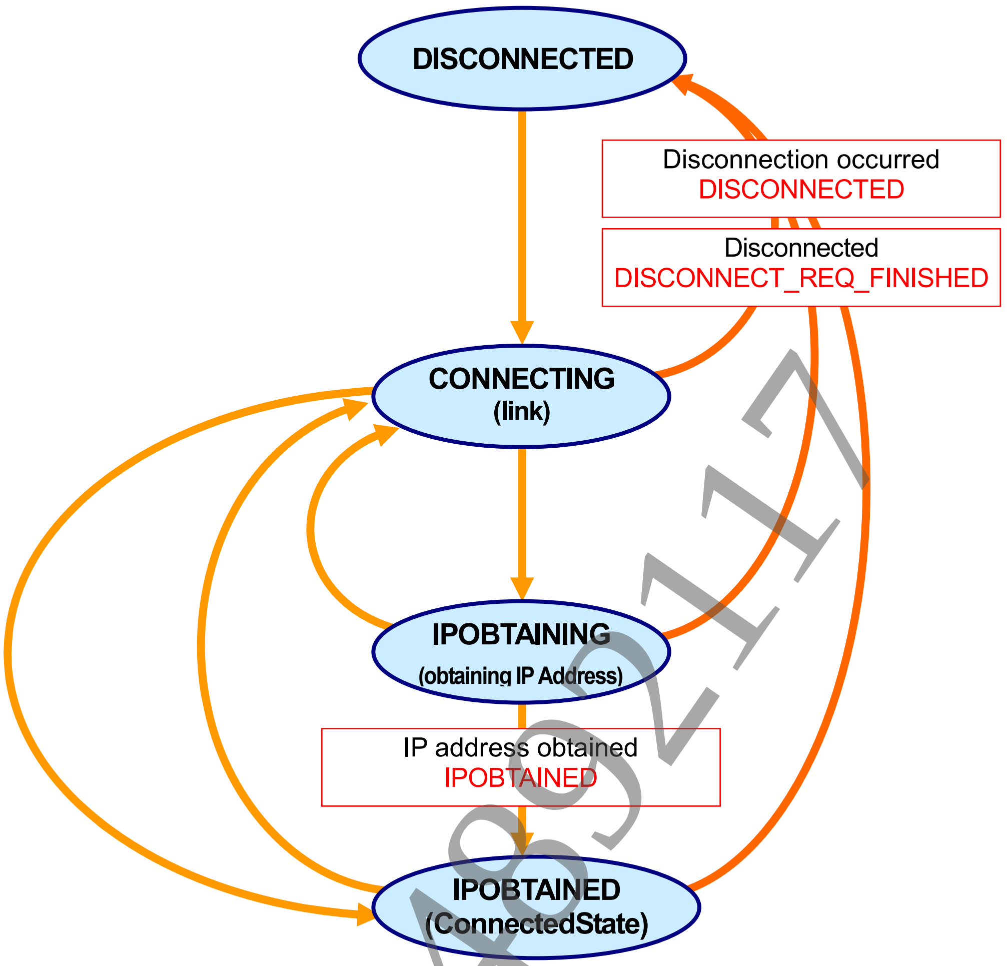
Normally, libnetctl is not required for applications that use Internet communication.

By using this library, it is possible to know whether the network is currently connected to, and if connected, the various connection information items. Further, by registering a callback function, notification can be received when the main connection state has changed.

Transitions in Network Connection States

The following figure shows transition of network connection states. Network connection states are indicated as blue ellipses, and events that cause state transitions are indicated in red.

Figure 1 Transition of Network Connection States



- DISCONNECTED
  In this state, there is no network connection. This is an initial state, as well as the state returned to when an error occurs or when connection is disconnected in another state.
  When attempting connection to a network, a transition is made from this state to the CONNECTING State.
- CONNECTING
  In this state, connection to a network is being attempted.
  Once connection is established, a transition is made to the IPOBTAINING state.
  If an error occurs or the connection is terminated, a transition will be made to the DISCONNECTED state. When an Ethernet cable is pulled out in a cable connection, or when disconnection occurs from an access point in a wireless connection, a transition will be made not to the DISCONNECTED state but to the CONNECTING state.
- IPOBTAINING (Obtaining IP Address)
  In this state, the IP address is being obtained. Depending on network settings, the IP address will be obtained by automatic acquisition (DHCP), manual, or PPPoE.
  Once the IP address is obtained, a transition will be made to the IPOBTAINED state.
  If an error occurs or the connection is terminated, a transition will be made to the DISCONNECTED state. When an Ethernet cable is pulled out in a cable connection, or when disconnection occurs from an access point in a wireless connection, a transition will be made not to the DISCONNECTED state but to the CONNECTING state. If a cable is thereafter inserted or communication with an access point is secured again, a transition will be made back to the IPOBTAINING state.

- IPOBTAINED (IP Address Obtained)
  In this state, the IP address has been obtained and connection to a network is established. The application can call the communication API regardless of the state of libnetctl, but communication is actually possible internally only after this state has been entered.
  If an error occurs or the connection is terminated, a transition will be made to the DISCONNECTED state. When an Ethernet cable is pulled out in a cable connection, or when disconnection occurs from an access point in a wireless connection, a transition will be made not to the DISCONNECTED state but to the CONNECTING state. If, at this point, the IP address is kept and once cable connection or communication with an access point is reestablished, a transition will be made back to IPOBTAINED state.

Obtainable Connection Information

Network connection information that can be obtained by `sceNetCtlInetGetInfo()` of libnetctl are listed in the following table. The code column indicates what should be specified for the 1st argument, *`code`* (the macro prefix, `SCE_NET_CTL_INFO` has been omitted here). The info column indicates to which member of the union the obtained information will be returned.

For details, refer to "libnetctl Reference".

Connection Information	code (*1)	info	Description
Connection Name	`_CNF_NAME`	*`cnf_name`*	Name assigned to network setting used for connection
Device	`_DEVICE`	*`device`*	Device used for Internet communication (cable or wireless)
Ethernet address	`_ETHER_ADDR`	*`ether_addr`*	Ethernet address of interface used for Internet communication
MTU	`_MTU`	*`mtu`*	Interface MTU
Connection state	`_LINK`	*`link`*	Link connection state (disconnected or connected)
BSSID	`_BSSID`	*`bssid`*	BSSID of connection-target-AP
SSID	`_SSID`	*`ssid`*	SSID of connection-target-AP
Wi-Fi Security	`_WIFI_SECURITY`	*`wifi_security`*	Security type of wireless connection
Receive signal strength indicator (dBm)	`_RSSI_DBM`	*`rssi_dbm`*	Receive signal strength indicator (dBm)
Receive signal strength indicator (%)	`_RSSI_PERCENTAGE`	*`rssi_percentage`*	Receive signal strength indicator (%)
Channel	`_CHANNEL`	*`channel`*	Channel of access point
IP setting	`_IP_CONFIG`	*`ip_config`*	Method to obtain IP address (Automatic acquisition, manual, PPPoE)
DHCP hostname	`_DHCP_HOSTNAME`	*`dhcp_hostname`*	Hostname to be used in DHCP
PPPoE user ID	`_PPPOE_AUTH_NAME`	*`pppoe_auth_name`*	User ID to be used in PPPoE
IP address	`_IP_ADDRESS`	*`ip_address`*	IP address
Net mask	`_NETMASK`	*`netmask`*	Net mask
Default route	`_DEFAULT_ROUTE`	*`default_route`*	Default router
Primary DNS	`_PRIMARY_DNS`	*`primary_dns`*	IP address of primary DNS
Secondary DNS	`_SECONDARY_DNS`	*`secondary_dns`*	IP address of secondary DNS
Proxy Server	`_HTTP_PROXY_CONFIG`	*`http_proxy_config`*	HTTP proxy server setting (use or don't use)
Proxy Server Address	`_HTTP_PROXY_SERVER`	*`http_proxy_server`*	Hostname of HTTP proxy server

Connection Information	code (*1)	info	Description
Proxy Server Port Number	`_HTTP_PROXY_PORT`	*`http_proxy_port`*	Port number of HTTP proxy server

*1: The macro constant prefix `SCE_NET_CTL_INFO` is omitted.

libnetctl Usage Method

- Acquisition of event by using callback function
  An occurrence notification of a pre-defined event can be received by registering the callback function of libnetctl. Registration is done with `sceNetCtlInetRegisterCallback()`.
  Note that in order to activate the callback function, it is necessary to call `sceNetCtlCheckCallback()`. The callback function is called within `sceNetCtlCheckCallback()`.
  If the callback function has been registered but is no longer needed, delete the function using `sceNetCtlInetUnregisterCallback()`.
- Acquisition of connection state
  The current connection state can be acquired with `sceNetCtlInetGetState()`.
- Acquisition of connection information
  As described above, the various connection information items can be acquired with `sceNetCtlInetGetInfo()`.

4 Intermittent Connection and Intermittent Disconnection (Internet Communication Mode)

In its Internet communication mode, the PlayStation®Vita implements a distinctive connection management policy for the purpose of energy conservation that was absent in the PSP™ and in the PlayStation®3.

This chapter describes the two concepts that characterize the policy: intermittent connection and intermittent disconnection.

Intermittent Connection

In the Internet communication mode, when an application starts communication from the disconnected state, the system software detects this and starts the connection processing. During this time, the communication API called by the application stays in the send/receive wait state, and upon completion of the connection processing, the actual send/receive processing starts.

The feature whereby the communication state is detected on the system software level and connection is performed as needed in this manner is hereafter called **intermittent connection**.

Intermittent Disconnection

After the system changes to the connected state, the connection is automatically disconnected on the system software level when the non-communication state continues for a given length of time (currently, 2 minutes).

The feature whereby the communication state is detected on the system software level and disconnection is performed as needed in this manner is hereafter called **intermittent disconnection**.

By introducing intermittent connection and intermittent disconnection, it is possible to make connection management independent from the application, and to guarantee connection in situations where the application requires it. In addition, by performing connection only for the minimum necessary periods on the system software level, it is possible to save energy.

Cautions for Intermittent Connection and Intermittent Disconnection

Necessary Processing when Performing Standby Processing on the PlayStation®Vita Side

A premise of the concepts of intermittent connection and intermittent disconnection is that the PlayStation®Vita performs client operation (transmission is always performed from the local terminal). For this reason, problems will arise if server operation (standby processing) is performed on the PlayStation®Vita.

Specifically, since connection management is made independent from the application, and the connection is started by the communication API from the PlayStation®Vita on the system software level, the PlayStation®Vita will not enter connected state just by performing standby processing. As a result, connection to the PlayStation®Vita cannot be processed via the network.

Based on PlayStation®Vita product specifications, it is appropriate to set client operation as a prerequisite. However, given that standby processing is performed on a day-to-day basis for development purposes, the following methods are provided for establishing a connection during standby processing for development purposes.

Periodical transmission to `SCE_NET_INADDR_BROADCAST` with UDP

Given that the system software will maintain the connection if communication from the PlayStation®Vita occurs periodically, it is possible to achieve standby processing by performing periodical transmission on the application level.

(e.g. broadcasting on the LAN every 30 seconds)

However, this method is inadequate if there are periods during which the application is suspended and the system software is running (such as when the application is interrupted by the debugger).

Disabling intermittent connection and intermittent disconnection from ★Debug Settings

A feature for disabling intermittent connection and intermittent disconnection of the system software is available exclusively for development purposes. By using this feature, it is possible to maintain the connection on the system software level without requiring awareness of connection management on part of the application.

In order to use this feature, launch **Settings** from the home screen, then turn Off the **Intermittent Connection** option under **Network** in ★**Debug Settings**. The default setting is **On**.

Inclusion of Connection Processing Time in Communication API

Through the introduction of intermittent connection, when the communication API is called from the disconnected state, the wait time for the connection processing is included in the API processing in addition to the regular send/receive processing. The wait time for the connection processing is the time taken to allocate the IP address. The wait time may be long because it is sometimes required to wait for the user operation for displaying the dialog of network connection or it may take a certain time to acquire the IP address distributed through DHCP.

As an example, consider the situation where intermittent connection occurs when the API `sceNetConnect()` is called in the state that the timeout time of `sceNetConnect()` is set to 30 seconds. In this case, the API does not return for timeout even if 30 seconds actually elapsed. The API processing can be aborted through `sceNetSocketAbort()` at an arbitrary timing, and after the execution of this function the control returns from the API.

Non-communication Time until Intermittent Disconnection

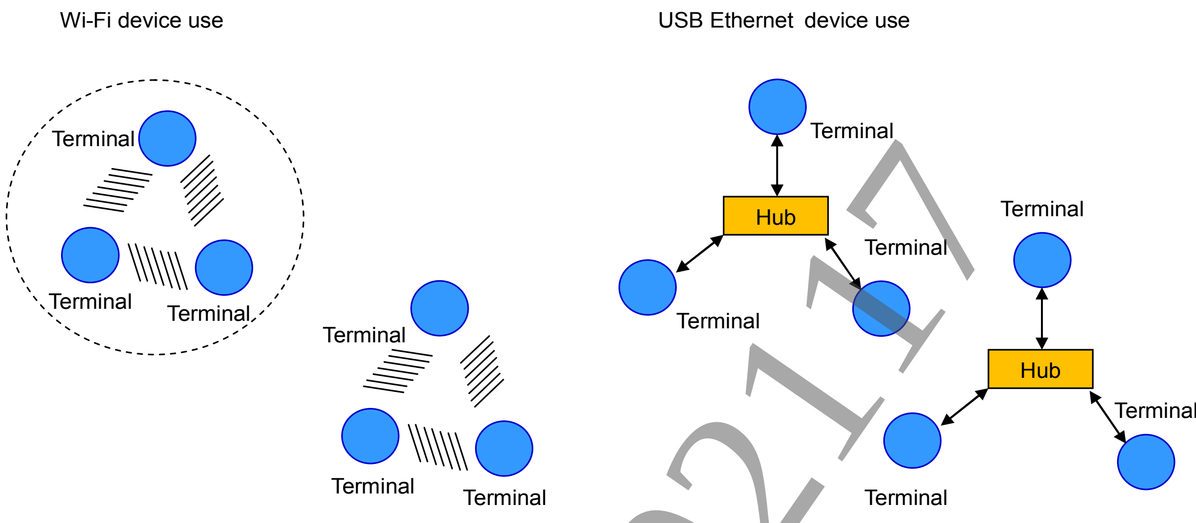
If there is no communication for a given length of time in the connected state, disconnection is executed on the system software level by the intermittent disconnection. Since this length of time may change in the future, avoid dependence on this time on the application level.

Behavior when Disconnecting from Connected State

After transitioning to connected state using intermittent connection, with communication from the application as a trigger, disconnection may occur due to network causes or system causes. In this case, the system software will simply return to disconnected state without performing reconnection. However, given that intermittent connection will occur when the application next attempts communication, the application will not be affected by the return to disconnected state.

5 Ad Hoc Communication Mode

This mode is used to perform direct network communication or play network games between neighboring terminals. No router or access point is required. During development using USB Ethernet devices, one hub per communication group is provided and communication is done via this hub.



Characteristics

When using the ad hoc communication mode, explicit connection processing and disconnection processing must be done on the application level. Network Check Dialog is in charge of this connection processing. For connection processing, it is necessary to specify NP Communication ID as an ID for preventing interferences. This value is issued for each application by applying on PlayStation®Vita Developer Network (https://psvita.scedev.net/).

Network Check Dialog is described later on in this document.

Differences with PSP™

The differences with the ad hoc communication mode in PSP™ are listed below.

Communication Method

In the case of the PlayStation®Vita, even in the ad hoc communication mode, communication is done based on IP addresses. For the transmission/reception function, use the same function as that for the Internet communication mode. The IP address allocated to the local terminal is not fixed and changes each time.

Moreover, because of the encryption method and in order to prevent interference between multiple titles, use of UDPP2P is indispensable. Non-UDPP2P socket creation and communication cause the `SCE_NET_ERROR_EADHOC` error to be returned in the ad hoc communication mode.

Specification of the NP Communication ID

On the PlayStation®Vita, the NP Communication ID specified at the time of connection is used as an ID to prevent crosstalk between applications. Normally, use the value issued by the PlayStation®Vita Developer Network.

Contact SCE if you wish to use other IDs than the issued NP Communication ID for communication between different title applications or other purposes.

Encryption Method

In the ad hoc communication mode, be sure to use the UDPP2P system. Data encryption and signature, on the other hand, are not required. If using UDPP2P without enabling encryption, note that communication will be performed without encryption.

Ad Hoc Communication Procedure

Preliminary Preparations

To operate applications that use ad hoc communication with the USB Ethernet, insert the USB Ethernet in the correct location of the DevKit/TestKit beforehand, and connect the Ethernet cable to a hub.

Processing Procedure

(1) Module loading
Call `sceSysmoduleLoadModule()` to load `SCE_SYSMODULE_NET`.

(2) Initialization
Call `sceNetInit()` to initialize libnet. `sceNetInit()` returns 0 when initialization has been successful. Next, call `sceNetCtlInit()` to initialize libnetctl. `sceNetCtlInit()` also returns 0 when initialization has been successful.

(3) Connection processing
Perform the connection processing using Network Check Dialog. For details on the procedure, refer to the "Network Check Dialog" chapter.
If connection processing using Network Check Dialog results in an error, further communication processing is not possible. To try again, perform the procedure again from the connection processing.

(4) Ad hoc communication processing
Following successful ad hoc connection, ad hoc communication processing can be called. Be sure to use the UDPP2P system for communication. UDPP2P can be used by specifying either `SCE_NET_SOCK_DGRAM_P2P` or `SCE_NET_SOCK_STREAM_P2P` for the socket type.

(5) Ad hoc disconnection processing
Call `sceNetCtlAdhocDisconnect()` and disconnect the ad hoc connection. This function returns immediately after a disconnection request is sent to the system. A separate notification must be received from the callback function in order to wait for completion of the disconnection.

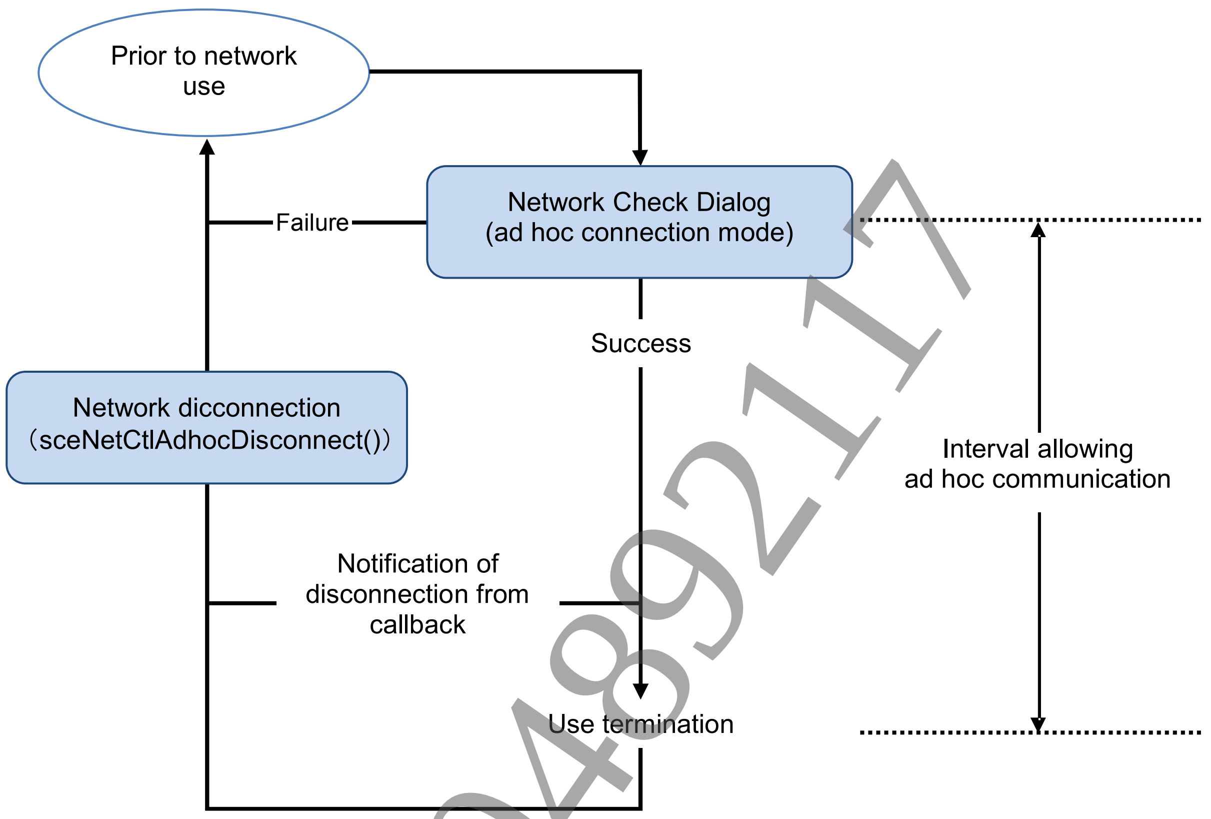
(6) Use termination
When ad hoc communication is no longer needed, call `sceNetCtlTerm()` and `sceNetTerm()` to end libnetctl and libnet.

(7) Module unloading
Call `sceSysmoduleUnloadModule()` and unload `SCE_SYSMODULE_NET`.

Operation Flow of Applications

The diagram below shows the general operation flow of applications that use the ad hoc communication mode.

Note that applications can only perform ad hoc communication from the moment Network Check Dialog is successful to the moment in which disconnection occurs.


Prior to network use
Network Check Dialog (ad hoc connection mode)
Failure
Success
Network dicconnection (sceNetCtlAdhocDisconnect())
Interval allowing ad hoc communication
Notification of disconnection from callback
Use termination

Acquisition of Connection State (libnetctl Feature)

Like during Internet connection, a function for acquiring the ad hoc connection state and a callback function are provided. The ad hoc connection state can be acquired with `sceNetCtlAdhocGetState()`. Callback registration and cancellation is done by calling `sceNetCtlAdhocRegisterCallback()` and `sceNetCtlAdhocUnregisterCallback()`, respectively. The transitions in network connection states are exactly the same as for Internet connection. For the transition diagram, refer to "Figure 1 Transition of Network Connection States".

Acquisition of Information of Other Terminals

In the connected state ,the information of other terminals that belong to the same IBSS can be acquired. This is done by using `sceNetCtlAdhocGetPeerList()`. In the current SDK, the IP address is the only information that can be acquired.

Maximum Number of Terminals

The maximum number of terminals that can be connected in a single communication group is 16.

Notes

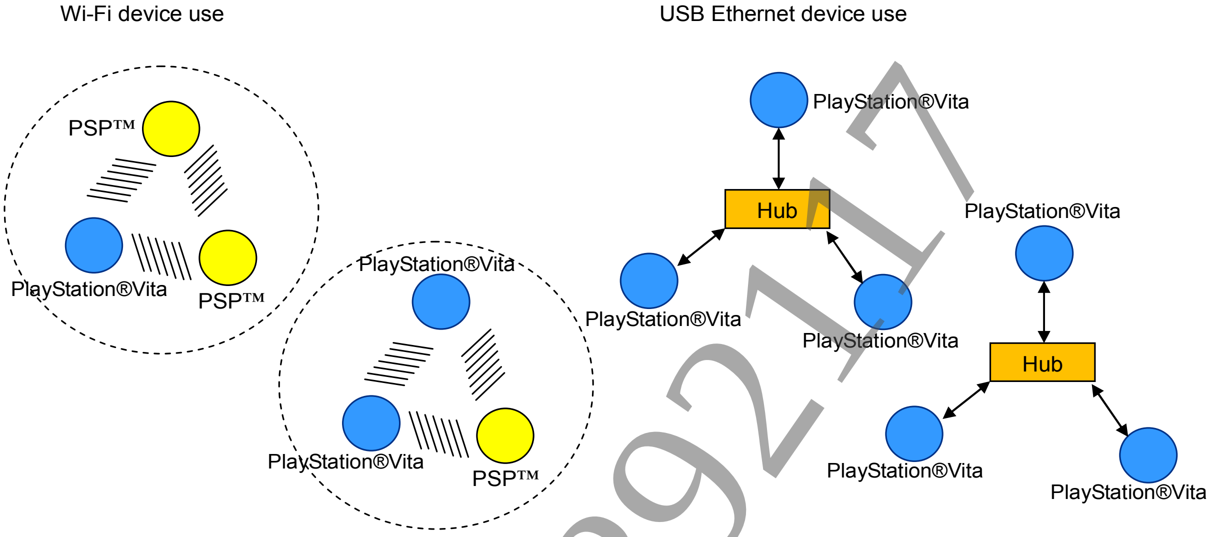
Exceeding the Maximum Number of Terminals

When terminals exceeding the maximum number of terminals connect to a single communication group, there is a possibility the communication environment may become undefined. In many cases, communication with other terminals becomes impossible on the terminal connected last (exceeding the limit). When connecting multiple terminals at the same time, in an event for example, take distribution measures to distribute channels or prepare differing NP Communication ID groups.

6 PSPNET Ad Hoc Communication Mode

This mode has compatibility with the PSP™ ad hoc communication mode. A PlayStation®Vita application can perform communication with a PSP™. Direct network communication or playing network games between neighboring terminals is possible. No router or access point is required.

During development using USB Ethernet devices, one hub per player group is provided and communication is done via this hub. When a USB Ethernet device is used, connection with a PSP™ is not possible.



Characteristics

When performing ad hoc communication with a PSP™ application, the exclusive protocol stack explained later is used.

When using the PSPNET ad hoc communication mode, explicit connection processing and disconnection processing must be done on the application level. Network Check Dialog is in charge of this connection processing.

For Network Check Dialog, refer to the "Network Check Dialog" chapter.

PSPNET Ad Hoc Communication Mode Exclusive Protocol Stack

The PSPNET ad hoc communication mode performs communication based on MAC addresses.

TCP/IP protocols cannot be used. Instead, use the PSPNET ad hoc communication mode exclusive PDP/PTP protocols.

PDP (PSPNET Datagram Protocol)

A replacement for the TCP/IP protocol UDP.

It is datagram-oriented and does not have reliability for data delivery. Unicast and broadcast transmission is possible.

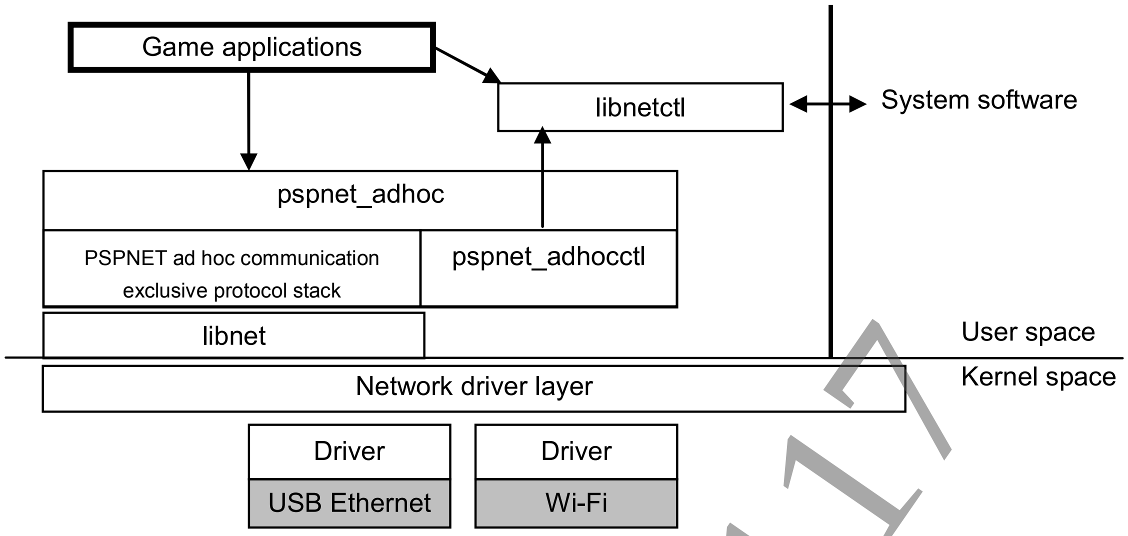
PTP (PSPNET Transport Protocol)

A replacement for the TCP/IP protocol TCP.

It is stream-oriented with connections. It has reliability for data delivery and can perform retransmission using acknowledgement and timers. Only unicast transmission is possible.

Feature Configuration Diagram

The system software configuration regarding the PSPNET ad hoc communication mode is as follows.



Communication Method

PSPNET ad hoc communication mode connections use Network Check Dialog. The connection method can be selected from among the following three.

PSPNET Ad Hoc Connection Mode (SCE_NETCHECK_DIALOG_MODE_PSP_ADHOC_CONN)

In this mode, the application can connect to an IBSS without the need to be aware of IBSS creation and participation.

This method has the same behavior as `sceNetAdhocctlConnect()` and network configuration utility ad hoc mode connection (`SCE_UTILITY_NETCONF_TYPE_CONNECT_ADHOC`) for the PSP™.

PSPNET Ad Hoc Creation Mode (SCE_NETCHECK_DIALOG_MODE_PSP_ADHOC_CREATE)

This mode is used to create IBSSs. An IBSS can be created even if it has the same SSID as an existing one. This method has the same behavior as `sceNetAdhocctlCreate()` and network configuration utility ad hoc mode (creation) (`SCE_UTILITY_NETCONF_TYPE_CREATE_ADHOC`) for the PSP™.

PSPNET Ad Hoc Join Mode (SCE_NETCHECK_DIALOG_MODE_PSP_ADHOC_JOIN)

This mode is used to join an IBSS. If there are multiple IBSSs with the same SSID, the IBSS to join is determined through BSSID comparison.

This method has the same behavior as connection using `sceNetAdhocctlScan()` and `sceNetAdhocctlJoin()`, and network configuration utility ad hoc mode (join) (`SCE_UTILITY_NETCONF_TYPE_JOIN_ADHOC`) for the PSP™.

Group Name and Ad Hoc ID Specification Method

Specify the group name in the PSP™ ad hoc communication mode as a Network Check Dialog parameter. For the PSPNET ad hoc communication mode, an ad hoc ID is required for connecting, so call `sceNetAdhocctlInit()` and specify an ad hoc ID before the Network Check Dialog call processing.

Note
In principle, specify the product code (four letters and five numbers) for `SceNetAdhocctlAdhocId`. For non-product code specification such as for communication between different applications, contact SCE.

An ad hoc ID is an ID for preventing interference between applications. Always specify an ad hoc ID. A group name is a parameter for creating another group in an application. If it is not required, you can specify NULL.

PSPNET Ad Hoc Communication Procedure

Preliminary Preparations

To operate applications that use PSPNET ad hoc communication with the USB Ethernet, insert the USB Ethernet in the correct location of the DevKit/TestKit beforehand, and connect the Ethernet cable to a hub.

Processing Procedure

(1) Module loading
Call `sceSysmoduleLoadModule()` to load `SCE_SYSMODULE_NET` and `SCE_SYSMODULE_PSPNET_ADHOC`.

(2) Initialization
Call `sceNetInit()` to initialize libnet. `sceNetInit()` returns 0 when initialization has been successful. Next, call `sceNetCtlInit()` to initialize libnetctl. `sceNetCtlInit()` also returns 0 when initialization has been successful.

(3) PSPNET adhoc library Initialization
Call `sceNetAdhocInit()` to initialize pspnet_adhoc. `sceNetAdhocInit()` returns 0 when initialization has been successful. Next, call `sceNetAdhocctlInit()` to initialize pspnet_adhocctl. `sceNetAdhocctlInit()` also returns 0 when initialization has been successful.

(4) Connection processing
Perform the connection processing using Network Check Dialog. For details on the procedure, refer to the "Network Check Dialog" chapter.
If connection processing using Network Check Dialog results in an error, further communication processing is not possible. To try again, perform the procedure again from the connection processing.

(5) PSPNET ad hoc communication processing
Following successful PSPNET ad hoc connection, PSPNET ad hoc communication processing can be called. For communication, always use the PSPNET ad hoc communication mode exclusive protocol stack.

(6) PSPNET ad hoc disconnection processing
Call `sceNetCtlAdhocDisconnect()` and disconnect the PSPNET ad hoc connection. This function returns immediately after a disconnection request is sent to the system. A separate notification must be received from the callback function in order to wait for completion of the disconnection.

(7) PSPNET adhoc library use termination
When PSPNET ad hoc communication is no longer needed, call `sceNetAdhocctlTerm()` and `sceNetAdhocTerm()` to endpspnet_adhocctl and pspnet_adhoc.

(8) Use termination
Call `sceNetCtlTerm()` and `sceNetTerm()` to end libnetctl and libnet.

(9) Module unloading
Call `sceSysmoduleUnloadModule()` and unload `SCE_SYSMODULE_NET` and `SCE_SYSMODULE_PSPNET_ADHOC`.

Operation Flow of Applications

The general operation flow of applications that use the PSPNET ad hoc communication mode is the same as standard ad hoc communication. Note that PSPNET ad hoc communication is possible in applications from after Network Check dialog success until a disconnect occurs. For an operation flowchart, refer to the appropriate section in the "Ad Hoc Communication Mode" chapter.

Acquisition of Connection State (libnetctl Feature)

Acquisition of the connection state should also be performed similarly to standard ad hoc communication. For details, refer to the appropriate section in the "Ad Hoc Communication Mode" chapter.

Acquisition of Information of Other Terminals

When in a connected state, it is possible to acquire the information of other terminals that belong to the same IBSS. `sceNetAdhocctlGetPeerList()` is used for acquisition.

Maximum Number of Terminals

The maximum number of terminals that can be connected in a single communication group is 17.

Notes

Exceeding the Maximum Number of Terminals

When terminals exceeding the maximum number of terminals connect to a single communication group, there is a possibility the communication environment may become undefined. In many cases, communication with other terminals becomes impossible on the terminal connected last (exceeding the limit). When connecting multiple terminals at the same time, in an event for example, take distribution measures to distribute channels or prepare differing ad hoc ID groups.

7 P2P Exclusive Protocol

Overview

Below is an overview of the P2P exclusive protocol provided with libnet

UDPP2P

Characteristics

libnet provides the UDPP2P protocol as a protocol for P2P communication by UDP, and for encryption as well as to prevent interference between multiple titles in the ad hoc communication mode. The UDPP2P protocol introduces virtual port numbers to enable a UDP port number to be shared between multiple applications.

To improve communicability and homogeneity, the NP Signaling library's NAT Traversal feature limits the local port numbers to be used to one predetermined port number. Therefore, using the UDPP2P protocol for P2P communication enables multiples applications to share one port number.

Socket Type

To use the UDPP2P protocol, specify `SCE_NET_SOCK_DGRAM_P2P` as the socket type when creating the socket.

Socket Address

There are multiple virtual ports for the UDPP2P protocol, and they are linked with one UDP port. Therefore, the socket address of an endpoint is expressed as a triplet consisting of the IP address, UDP port number, and virtual port number.

Socket Address Structure

The socket address structure that is used with the UDPP2P protocol is `SceNetSockaddrIn`. It is defined in net/in.h included by net.h. To use this socket address structure, include net.h.

```
typedef struct SceNetSockaddrIn {
    SceUChar8 sin_len;
    SceNetSaFamily_t sin_family;
    SceNetInPort_t sin_port;
    SceNetInAddr sin_addr;
    SceNetInPort_t sin_vport;
    SceChar8 sin_zero[6];
} SceNetSockaddrIn;
```

For details, refer to `SceNetSockaddrIn` in the "libnet Reference" document.

Port Number

In the Internet communication mode, use the port numbers below. `SCE_NP_PORT(=3658)` is defined in np/common.h included by np.h. To use `SCE_NP_PORT(=3658)`, include np.h. As with conventional UDPs, the following game ports include specifying 0 for a port number specified by the game, or to have the system specify an arbitrary port number.

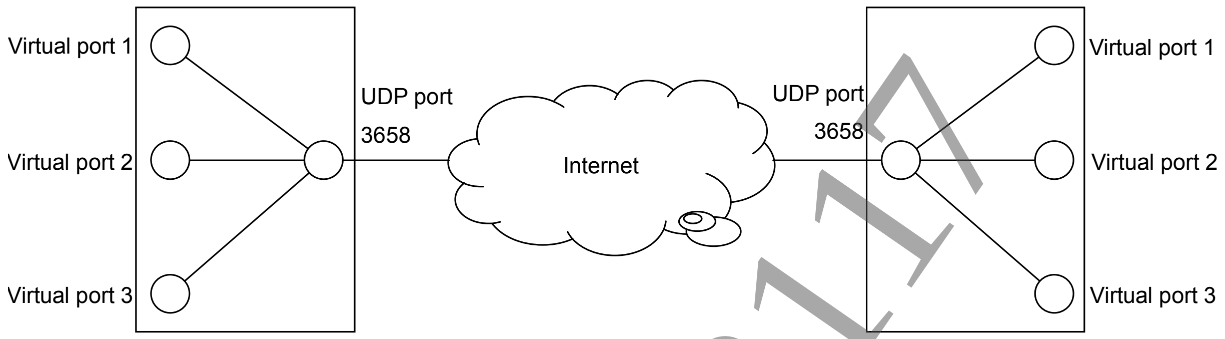
	Host Binding	Client Binding	Connection, Transmission
sin_port	SCE_NP_PORT	SCE_NP_PORT	(*1)
sin_vport	Game port	Game port	Game port

(*1) This is the port number actually opened by the other party seen from the Internet. For example, it can be obtained by using the NP Matching 2 library (refer to the "NP Matching 2 System Overview" document).

Use the following port numbers in ad hoc communication mode. `SCE_NET_ADHOC_PORT(=3658)` is defined in net/socket.h included by net.h. To use `SCE_NET_ADHOC_PORT(=3658)`, include net.h.

	Host Binding	Client Binding	Connection, Transmission
sin_port	SCE_NET_ADHOC_PORT	SCE_NET_ADHOC_PORT	SCE_NET_ADHOC_PORT
sin_vport	Game port	Game port	Game port

Virtual Ports



A virtual port number, which is represented as 16 bits, has a value from 0 to 65535. Virtual port numbers are assigned as follows.

- 0, 65535
  Not used
- 1 to 32767
  For use by users
- 32768 to 65534
  For use by the system

Packet Format

A 2- or 4 -byte UDPP2P protocol tag is attached to the beginning of the UDP protocol payload. Otherwise, the packet format obeys the UDP format.

Socket APIs

BSD socket APIs can be used for UDPP2P protocol socket APIs. The operations that can be performed are similar to those that can be performed when using a UDP socket.

However, because the local port number used has been decided, be sure to call `sceNetBind()` to bind the address to the socket before using it.

Data Size

The maximum data size that can be sent in the UDPP2P protocol is 9216 bytes.

Data Encryption and Signatures

The UDPP2P protocol supports data encryption and the creation/verification of signatures. The key is exchanged when the signaling connection is established and can be used when the connection is in ACTIVE state.

To use these features, specify the applicable socket option with `sceNetSetsockopt()`, with `SCE_NET_SOL_SOCKET` as the level. Each option can be specified independently.

SCE_NET_SO_USECRYPTO

This option enables the encryption and decryption of data sent/received by the socket. Encryption adds an 8-byte initial vector per packet. If the key corresponding to the packet is not set, the packet will be implicitly discarded if it is not encrypted when received.

SCE_NET_SO_USESIGNATURE

This option enables the creation and verification of signatures added to the data sent/received by the socket. An 8-byte signature is added per packet. If the key corresponding to the packet is not set, if the verification fails when the packet is received, or if there is no signature, the packet will be implicitly discarded.

In addition, it is possible to specify encryption and signature creation per packet. In this case, specify the flags using `sceNetSend()`, `sceNetSendto()` and `sceNetSendmsg()`. Each flag can be specified independently.

SCE_NET_MSG_USECRYPTO

This option performs encryption on the data to be sent. Operation and format are the same as when `SCE_NET_SO_USECRYPTO` is specified.

SCE_NET_MSG_USESIGNATURE

This option attaches a signature to the data to be sent. Operation and format are the same as when `SCE_NET_SO_USESIGNATURE` is specified.

When specifying the above flags to a socket to which `SCE_NET_SO_USECRYPTO` and `SCE_NET_SO_USESIGNATURE` have already been set, the encryption and signature applied to the packet will be determined by taking the OR of each option.

TCP over UDPP2P

Characteristics

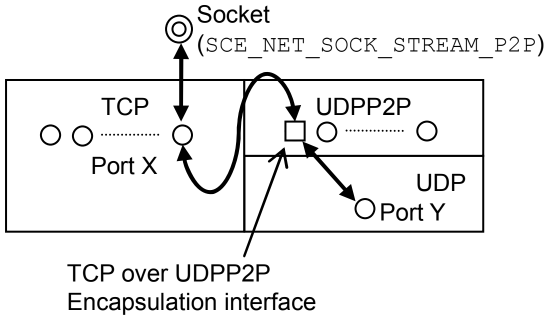
TCP over UDPP2P implements TCP communication in the ad hoc communication mode by encapsulating TCP segments using the UDPP2P protocol. This enables communication via a more reliable connection than what was possible with just the UDPP2P protocol.

Socket Type

To use the UDPP2P protocol, specify `SCE_NET_SOCK_STREAM_P2P` as the socket type when creating the socket.

Socket Address

The TCP port numbers of TCP over UDPP2P use the same space as regular TCP. However, a UDP port number is also needed to encapsulate TCP segments using the UDPP2P protocol. Therefore, the socket address of an endpoint is represented by a triplet consisting of an IP address, TCP port number (Port X), and UDP port number (Port Y).



Socket Address Structure

The socket address structure used by the UDPP2P protocol is `SceNetSockaddrIn`. It is defined in net/in.h included by net.h. To use `SceNetSockaddrIn`, include net.h.

```
typedef struct SceNetSockaddrIn {
    SceUChar8 sin_len;
    SceNetSaFamily_t sin_family;
    SceNetInPort_t sin_port;
    SceNetInAddr sin_addr;
    SceNetInPort_t sin_vport;
    SceChar8 sin_zero[6];
} SceNetSockaddrIn;
```

For details, refer to `SceNetSockaddrIn` in the "libnet Reference" document.

Port Number

In the Internet communication mode, use the port numbers below. `SCE_NP_PORT(=3658)` is defined in np/common.h included by np.h. To use `SCE_NP_PORT(=3658)`, include np.h. As with conventional TCPs, the following game ports include specifying 0 for a port number specified by the game, or to have the system specify an arbitrary port number.

	Host Binding	Client Binding	Connection, Transmission
sin_port	Game port	Game port	Game port
sin_vport	`SCE_NP_PORT` or 0	`SCE_NP_PORT` or 0	(*1)

(*1) This is the port number actually opened by the other party seen from the Internet. For example, it can be obtained by using the NP Matching 2 library (refer to the "NP Matching 2 System Overview" document).

Use the following port numbers in ad hoc communication mode. `SCE_NET_ADHOC_PORT(=3658)` is defined in net/socket.h included by net.h. To use `SCE_NET_ADHOC_PORT(=3658)`, include net.h.

	Host Binding	Client Binding	Connection, Transmission
sin_port	Game port	Game port	Game port
sin_vport	`SCE_NET_ADHOC_PORT` or 0	`SCE_NET_ADHOC_PORT` or 0	`SCE_NET_ADHOC_PORT` or 0

Packet Format

The TCP over UDPP2P tag is attached to the first 4 or 6 bytes of the UDP protocol payload, and the subsequent section of the payload encapsulates the TCP segment.

Socket APIs

BSD socket APIs can be used for TCP over UDPP2P protocol socket APIs. The operations that can be performed are similar to those that can be performed when using a TCP socket.

Data Encryption and Signature

TCP over UDPP2P supports data encryption, and signature creation and authentication. Specify the `SCE_NET_SO_USECRYPTO` and/or `SCE_NET_SO_USESIGNATURE` socket option(s) in `sceNetSetsockopt()` (level is `SCE_NET_SOL_SOCKET`). Each option can be independently specified.

The socket option of the socket sending the data and the socket receiving the data must be the same. Communication is not possible between sockets with different socket options.

The data payload overhead occurring for each of the options is the same as for a UDPP2P protocol.

Note that data encryption and signature creation cannot be specified per packet.

Connection Reset

When the connection for signaling becomes INACTIVE, the TCP over UDPP2P connection established or being established between the applicable peers will be reset. `SCE_NET_ERROR_ECONNRESET` will return to the socket API operating this connection.

8 Network Check Dialog

Overview

Network Check Dialog is provided as Common Dialog that is in charge of network connections and GUI part of the PSN℠ processing. For the common specifications of Common Dialog, refer to the "Common Dialog Overview" document.

Features

Features provided by Network Check Dialog are as follows.

Feature	Mode Name
Connection using the ad hoc communication mode	Ad hoc connection mode
Connection using the PSPNET ad hoc communication mode	PSPNET ad hoc connection mode
Creation in PSPNET ad hoc communication mode	PSPNET ad hoc creation mode
Joining in PSPNET ad hoc communication mode	PSPNET ad hoc join mode
Transition to the signed-in state for the service state of the NP library	PSN℠ mode
Transition to the online state for the service state of the NP library	PSN℠ online mode
Connection with PlayStation®3	PlayStation®3 connection mode

Three PSPNET ad hoc connection methods are provided: connection, creation, and join. Applications that perform ad hoc communication with PSP™ must preliminarily use one of these three methods.

Ad Hoc Connection Mode

This feature is in charge of connection processing in ad hoc communication mode. Applications that perform the ad hoc communication mode must use this feature in advance.

PSPNET Ad Hoc Connection Mode

This feature is in charge of connection processing in PSPNET ad hoc communication mode. It automatically creates, searches for, and joins ad hoc networks that are compatible with PSP™.

PSPNET Ad Hoc Creation Mode

This feature is in charge of network creation processing in PSPNET ad hoc communication mode.

PSPNET Ad Hoc Join Mode

This feature is in charge of join processing in PSPNET ad hoc communication mode.

PSN℠ Mode

This feature is in charge of transition processing to the signed-in state for the NP library service state. Applications that use the each feature of PSN℠ related libraries for which the signed-in state is required must use this feature in advance.

For details on the signed-in state or the PSN℠ related libraries (or features) which require the signed-in state, refer to the "NP Library Overview" document.

After transition to the signed-in state completes, a check for parental control will be carried out.

PSN℠ Online Mode

This feature is in charge of transition processing to transition to the online state for the NP library service state. Applications that use the each feature of PSN℠ related libraries for which the online state is required must use this feature in advance.

For details on the online state or the PSN℠ related libraries (or features) which require the online state, refer to the "NP Library Overview" document.

After transition to the signed-in state completes and before transitioning to the online state, a check for parental control will be carried out.

PlayStation®3 Connection Mode

This handles processing for performing communication with applications that have started up the "access point utility" of the PlayStation®3. The details of this mode will be described in the next chapter.

Network Check Dialog Running Procedure

Network check dialog starting up procedure is outlined below using ad hoc connection mode as an example.

Preliminary Preparations

The API of Network Check Dialog is included in the Common Dialog library. Applications must be linked with the libSceCommonDialog_stub.a.

Processing Procedure

(1) Initialization
Initialize the startup parameter of Network Check Dialog by calling `sceNetCheckDialogParamInit()`. Next, specify the parameters for each member of the `SceNetCheckDialogParam` structure and start Network Check Dialog by calling `sceNetCheckDialogInit()`. In the case of ad hoc communication mode connection, *`mode`* is `SCE_NETCHECK_DIALOG_MODE_ADHOC_CONN` and to *`npCommunicationId`*, specify the value issued through application on PlayStation®Vita Developer Network (https://psvita.scedev.net/).
Specify the common parameters of Common Dialogs in *`commonParam`*. Given that it is not possible to perform specifications concerning the info bar in Network Check Dialog, always specify NULL in *`commonParam.infobarParam`*. If a value other than NULL is specified, `sceNetCheckDialogInit()` will return `SCE_COMMON_DIALOG_ERROR_INVALID_INFOBAR_PARAM`. For details, refer to the "Common Dialog Reference" document.
In the PSN℠ mode or PSN℠ online mode, appropriate parental control information must be passed as a parameter. For details, refer to the "libnetctl Reference" document.

(2) Wait for completion
Following the completion of Network Check Dialog startup, Network Check Dialog starts operating in the specified operating mode. During this time, the application must call `sceCommonDialogUpdate()` at every frame in order to perform Common Dialog drawing and processing progression. Similarly, call `sceNetCheckDialogGetStatus()` at every frame and wait until the return value becomes `SCE_COMMON_DIALOG_STATUS_FINISHED`.
When `SCE_COMMON_DIALOG_STATUS_FINISHED` is returned, this means the termination of the Network Check Dialog.

(3) Result obtainment
Once `SCE_COMMON_DIALOG_STATUS_FINISHED` is obtained, call `sceNetCheckDialogGetResult()` to get the result. If the *`result`* is 0, the connection was successful. In the case of an error, a negative value is returned to *`result`*.

(4) Termination processing
Regardless of whether the result was successful connection or an error, terminate the Network Check Dialog by calling `sceNetCheckDialogTerm()`.

Precautions

The preliminary preparations for Network Check Dialog differ according to the mode.

Ad Hoc Connection Mode

The following modules must be loaded using the libsysmodule library.

- `SCE_SYSMODULE_NET`

After this is done, the `sceNetInit()` and `sceNetCtlInit()` initialization functions must be called.

PSPNET Ad Hoc Connection Mode

The following modules must be loaded using the libsysmodule library.

- `SCE_SYSMODULE_NET`
- `SCE_SYSMODULE_PSPNET_ADHOC`

After this is done, the `sceNetInit()`, `sceNetCtlInit()`, `sceNetAdhocInit()` and `sceNetAdhocctlInit()` initialization functions must be called.

PSN℠ Mode and PSN℠ Online Mode

The following modules must be loaded using the libsysmodule library.

PSN℠ Mode

- `SCE_SYSMODULE_NET`
- `SCE_SYSMODULE_HTTPS`
- `SCE_SYSMODULE_NP`

PSN℠ Online Mode

- `SCE_SYSMODULE_NET`
- `SCE_SYSMODULE_HTTPS`
- `SCE_SYSMODULE_NP`

After this is done, the `sceNetInit()`, `sceNetCtlInit()`, `sceSslInit()`, `sceHttpInit()`, and `sceNpInit()` initialization functions must be called.

When Network Check Dialog is started, the respective remaining capacity of the memory pools of libssl and libhttp must equal or exceed the specified capacity. The sizes that are actually required are indicated by the following macros of sdk/target/include/netcheck_dialog.h

`SCE_NETCHECK_DIALOG_LEAST_HTTP_POOL_SIZE`

`SCE_NETCHECK_DIALOG_LEAST_SSL_POOL_SIZE`

PlayStation®3 Connection Mode

The following module must be loaded using the libsysmodule library.

- `SCE_SYSMODULE_NET`

After this is done, the `sceNetInit()` and `sceNetCtlInit()` initialization functions must be called.

9 PlayStation®3 Connection Mode

Overview

The PlayStation®3 connection mode is a mode for connecting PlayStation®Vita and PlayStation®3 via wireless LAN. By using this mode, it is possible to communicate with applications using the access point utility of the PlayStation®3.

Operation

There are two actions available in the PlayStation®3 connection mode: ENTER and LEAVE.

If ENTER is specified as the action when calling Network Check Dialog, connection will be made either to the internal access point of the PlayStation®3, or to an access point or router in accordance with the network settings of the PlayStation®Vita. Then, the PlayStation®3 that is present on the same subnet as the PlayStation®Vita will be searched for, and the IP address of the PlayStation®3 matching the title ID specified in the argument will be obtained.

The application can communicate arbitrarily with the PlayStation®3 by using the IP address that has been obtained.

Behavior during connection with the PlayStation®3 is the same as during Internet communication mode. However, it is subject to the following limitations

- Communication to the Internet is not possible.
- Intermittent connection does not occur.

Given that intermittent connection does not occur, the application should monitor the network's connection status. If this is notified as DISCONNECTED, when necessary call Network Check Dialog to connect with the PlayStation®3.

After communication with the PlayStation®3 is finished, call Network Check Dialog, specifying LEAVE as the action. When LEAVE processing is complete, the above limitations will be released and Internet communication will again become possible.

Processing Procedure

The general overview of the procedure is the same as described in the previous chapter. Below is a description of the differences:

- At initialization, set `SCE_NETCHECK_DIALOG_MODE_PS3_CONNECT` to *`mode`* of the `SceNetCheckDialogParam` structure; in *`ps3ConnectParam`*, specify the pointer to the structure for which the required values for connection to the PlayStation®3 (action, TITLE ID, etc.) have been set.
- When obtaining results, call `sceNetCheckDialogGetPS3ConnectInfo()` to obtain the information required for communication with the PlayStation®3 after checking that connection has been successful with `sceNetCheckDialogGetResult()`. After connection succeeds, monitor the network's connection status, and perform reconnection when DISCONNECTED is notified.

Precautions

Do not use the UDP's port no. 19293 since it is reserved by the PlayStation®3 connection mode.

Out of the SSID and WPA key setting methods of the access point utility provided by the PlayStation®3 SDK, the PlayStation®3 connection mode does not support the method using settings for remote play (`CELL_SYSUTIL_AP_TYPE_USE_REMOTEPLAY_SETTING`). We recommend using the method specified on the application side (`CELL_SYSUTIL_AP_TYPE_USE_GAME_SETTING`).

Concerning the PlayStation®3 access point utility, refer to the following documents, provided as documents of the SDK for PlayStation®3.

- Access Point Utility Overview
- Access Point Utility Reference

10 Debug-Related Matters

Overview

Features for acquiring network information such as socket information and interface information, and features for emulating packet losses and delays on the network are provided.

Using the Features in a Program

Network Information

Applications can output network information below to TTY or a file by using the libnet functions in the specified location in the program.

- Socket information
- Interface information, statistics information
- Routing information
- tcpdump acquisition feature

Network Emulation

Network emulation is offered as a system for verifying beforehand the stability of communication with Wi-Fi devices as well as differences in performance when switching from a USB Ethernet device which is used during development to a Wi-Fi device. Applications can manipulate the network emulation parameters from a program using the functions provided by libnet. For details on the features, refer to the "libnet Overview" document.

Using the psp2ctrl Utility

It is possible to manipulate part of the network information and the network emulation described above from the psp2ctrl utility. For details, refer to the "Neighborhood and Utilities User's Guide" document (this document is included in the Neighborhood for PlayStation®Vita Help package and Target Manager Server Help package.).

Using ★Debug Settings

Network Emulation

Preset values can be set. For details, refer to the "★Debug Settings Functions" chapter in the "System Software Overview" document. In addition, for the preset values, refer to the "libnet Overview" document.

Disabling the Intermittent Connection

It is possible to change intermittent connection operation for debugging purposes. For example, we believe this may be useful during development, when debugging while using the development support host tool for preparing applications via the network. Refer to the "★Debug Settings Functions" chapter of the "System Software Overview" document for setting methods.

11 Support Features for 3G Lines During Development

Overview

Currently, DevKits/TestKits do not support 3G lines. However, if you wish to take into account the use of 3G lines for the purpose of application development, a feature to show simulated 3G line usage state is available.

3G Line Fake Feature Setting Method

To enable the 3G line fake feature, launch **Settings** from the home screen, then turn **On** the **Fake 3G Interface** option under **Network** in ★**Debug Settings**. The default setting is **Off**.

Feature Details

When the 3G line fake feature is enabled, the "network connection device" information that the application can obtain from libnetctl will change.

Specifically, Network Connection Device information can be obtained by calling `sceNetCtlInetGetInfo()` specifying `SCE_NET_CTL_INFO_DEVICE` in the *`code`* argument.

If `SCE_NET_CTL_INFO_DEVICE` is specified when calling, the device in use can be obtained, provided that it is in connected state. However, if the 3G line fake feature is enabled, `SCE_NET_CTL_DEVICE_PHONE` (meaning the 3G line) will always be returned.

Note that only the above information is faked, and the behavior of the devices actually used for communication, etc. remains as normal.

12 Other Matters

Influence of Suspending/Resuming Applications

If an application is suspended, the system software will maintain the connection without any special handling, regardless of the communication mode which the application was using.

On the other hand, from the point of view of applications, it is possible that disconnection may occur due to various causes during suspension. Therefore, there is no guarantee that the connection will be maintained until the application is resumed, even if suspension occurred during connected state.

If disconnection occurs while the application is suspended, disconnection will be notified after the application is resumed, provided that a callback function is registered to libnetctl.

In the case of the Internet communication mode, intermittent connection will be performed normally when a communication API is next issued from the application, even if the connection is in disconnected state when the application is resumed.

In terms of connection management, the system software will not react to the application being suspended. However, it may happen that the socket in use by the application cannot be maintained due to having been disconnected while the application was suspended, or other reasons. In this case, the socket will enter error state after the application is resumed. For details, refer to the "libnet Overview" document.

Causes of Disconnection

After the application transitions to connected state, disconnection may occur due to a variety of reasons.

Below are some typical causes of disconnection:

- Disconnection from the access point (when using Wi-Fi in Internet communication mode)
- Ethernet link-down (when using USB Ethernet in Internet communication mode)
- Starting up other applications and beginning use of a different communication mode
- Starting up the settings application and disconnecting from the UI
- Occurrence of system suspension

PlayStation®TV Internet Connection

Internet connection on PlayStation®TV differs from that on PlayStation®Vita as follows.

- Rather than an intermittent connection, PlayStation®TV will attempt to always maintain a connected state. Even when it is disconnected from a network, an attempt to reconnect will immediately be carried out.
- Intermittent disconnection due to non-communication after a certain time is not carried out and the connected state is maintained even when there is no communication exchange.
- Although the operation to select appropriate settings upon connection is the same as with PlayStation®Vita, the wired Ethernet device is prioritized on PlayStation®TV if an Ethernet cable is inserted.

PlayStation®TV Ad Hoc Connection

A Wi-Fi device and wired Ethernet device is installed onto PlayStation®TV; however, for ad hoc connection and PSPNET ad hoc connection, the wired Ethernet device is not used and the Wi-Fi device will always be used instead. This is a measure to enable connection with PlayStation®Vita.

The same is true on DevKit when **PS TV Emulation** is set to **On**.