PlayStation®2 EE Library Overview
Release 2.4.3

Network Libraries

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.

Summary Table of Contents

About This Manual

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

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

Changes Since Last Release

Chapter 1: HTTP Library

- In the "Basic Code Flow" section of "Functions and Code Flow", a description of program code that ends use of the library has been added.
- In the "Non-Blocking Mode" section of "Functions and Code Flow", a description of arguments defined by the user has been added.
- In "Chunk Receive Processing" of "Functions and Code Flow", a description of arguments defined by the user has been added.

Chapter 4: Network Configuration GUI Library

- A description of "Set up the buffer for receiving the name of the selected settings and its device" has been added to the "Initialization Processing Needed Before sceNetGuiCnf_Do() is Called" section of "Library Startup and Shutdown".
- A description on "When an I/O error occurs" has been added to the “Determining Errors" section of "Library Functions

Related Documentation

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

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

Typographic Conventions

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

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

Developer Support

Sony Computer Entertainment America (SCEA)

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

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

Sony Computer Entertainment Europe (SCEE)

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

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

Chapter 1: HTTP Library

Library Overview

libhttp is a library that provides HTTP support for applications.

Its most basic operation is sending an HTTP request to a specified URI from an application and receiving a response. libhttp implements various functions for this basic processing in the library such as proxy specification and authentication processing.

To support various network libraries, libhttp uses netglue as a low-level network API. netglue provides a BSD socket-compliant wrapper API. Since the netglue library is accessed internally by libhttp, the application need only link to the netglue library.

The SDK includes netglue_insck.a as a standard library file. netglue_insck.a is a netglue library that uses libinsck, which provides a BSD socket library for the SCEI network library (inet). Therefore, when using inet, be sure to link to netglue_insck.a so that it will be available for use.

To use libhttp with any other protocol stack, you must provide a dedicated netglue library.

Related Files

The following files are required to use libhttp.

Table 1-1

Category	Filename
Header files	include/libhttp.h
	(the following three files are included internally)
	include/libhttp/http_method.h
	include/libhttp/http_options.h
	include/libhttp/http_status.h
Library files	libhttp.a
	When inet library is used
	netglue_insck.a
	libinsck.a
	libnet.a
Module files	When inet library is used
	netcnf.irx
	inet.irx
	inetctl.irx
	msifrpc.irx
	libnet.irx

Normally, it is only necessary to include include/libhttp.h as the header file.

Sample Programs

The following sample programs are provided that use libhttp via the inet library.

- ee/sample/inet/libhttp/normal — Performs GET, HEAD, or POST for a specified URI
- ee/sample/inet/libhttp/proxy — Gets a specified URI via a proxy
- ee/sample/inet/libhttp/mime — Gets a specified URI and performs MIME processing
- ee/sample/inet/libhttp/blocking — Blocking version of http_test
- ee/sample/inet/libhttp/chunk — Example of chunk transfer processing
- ee/sample/inet/libhttp/cookie — Example of cookie processing
- ee/sample/inet/libhttp/redirect — Example of redirection processing
- ee/sample/inet/libhttp/auth — Example of authentication processing

Sample programs are also provided that perform BASE64 and quoted-printable encoding / decoding as well as URL escape / unescape processing.

- ee/sample/inet/libhttp/base64 — BASE64 processing sample
- ee/sample/inet/libhttp/qp — quoted-printable processing sample
- ee/sample/inet/libhttp/urlesc — URL escape and unescape processing sample

Functions and Code Flow

Basic Code Flow

With libhttp the basic code flow is as follows.

The application should first create and set up a client structure. It will then use that structure to send an HTTP request. After an HTTP response to the request is received, the response structure can be obtained and the results analyzed.

Specifically, the application first initializes the HTTP library using sceHTTPInit(). Then, it creates a client structure with sceHTTPCreate() and sets up that structure using various functions provided by the library. Once the structure is set up, the application uses sceHTTPOpen() and sceHTTPRequest() to send the actual HTTP request. When the HTTP response is received, the application can use sceHTTPGetResponse() to get the response structure.

```
#define ERR_STOP while(1)
int main(int argc, char **argv){
       sceHTTPClient_t *client;
       sceHTTPResponse_t *response;
       sceHTTPParsedURI_t *p;
       int ret;
       char *target_uri;

       ...
       /* Initialize the network library in advance */
```

```
...

/* Initialize libhttp */
if((ret = sceHTTPInit()) < 0){
       printf("sceHTTPInit() failed.\n");
       ERR_STOP;
}

/* Create HTTP client structure. The client structure is
   internally generated by libhttp. */
if((client = sceHTTPCreate()) == NULL){
       printf("sceHTTPCreate() failed.\n");
       ERR_STOP;
}

/* Convert target URI to URI structure for libhttp.
  Assume the target URI is referenced using target_uri. */
if((p = sceHTTPParseURI(target_uri, (sceHTTPParseURI_FILENAME
       | sceHTTPParseURI_SEARCHPART))) == 0){
       printf("sceHTTPParseURI() failed.\n");
       ERR_STOP;
}

/* Set the target method in the client structure. */
if((ret = sceHTTPSetOption(client, sceHTTPO_Method,
 sceHTTPM_GET)
) < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}

/* Set the URI in the client structure. */
if((ret = sceHTTPSetOption(client, sceHTTPO_ParsedURI, p)) <
 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}

/* Before sending an HTTP request, set options for the
    connection. */
sceHTTPSetOption(client, ...);
...

/* Start HTTP connection. Establish actual connection. */
if(sceHTTPOpen(client) < 0){
       printf("sceHTTPOpen() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}

/* Send HTTP request.
For blocking mode, block until HTTP response is received. */
if((ret = sceHTTPRequest(client)) < 0){
       printf("sceHTTPRequest() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
```

```
        /* Get HTTP response. */
        response = sceHTTPGetResponse(client);

        ...
        /* Perform arbitrary processing for the HTTP response that was
           obtained. */
        ...

        /* Termination processing. Close the connection. */
        if((ret = sceHTTPClose(client)) < 0){
              printf("sceHTTPClose() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
        }

        /* Free the client structure that was generated internally by
            the library. All associated HTTP response structures and
            header lists are also freed. */
        if((ret = sceHTTPDestroy(client)) < 0){
              printf("sceHTTPDestroy() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
        }

        /* Terminate library use */
        if((ret = sceHTTPTerminate()) < 0){
              printf("sceHTTPTerminate() failed.(%d)\n",
                      sceHTTPGetClientError(client));
              ERR_STOP;
        }

        return 0;

}
```

Adding Optional Headers

With libhttp, the following optional headers are added by default to an HTTP request.

Table 1-2

Header	Value
HOST:	Destination host name
Accept:	*/*;q=0.01
Accept-Encoding:	gzip,compress
Accept-Charset:	iso-8859-1;q=0.01
User-Agent:	Set from sceHTTPSetOption()

(When redirection is set by sceHTTPSetRedirection())

Table 1-3

Header	Value
Pragma:	no-cache
Cache-Controll:	no-cache

(When method is POST and the body is URL-encoded)

Table 1-4

Header	Value
Content-Type:	application/x-www-form-urlencoded

(When method is POST)

Table 1-5

Header	Value
Content-Length:	Body byte count

(When basic authentication is to be performed due to sceHTTPSetBasicAuth() or digest authentication is to be performed due to sceHTTPSetDigestAuth())

Table 1-6

Header	Value
Authorization:	Authentication value

(When a cookie is specified using sceHTTPSetCookie())

Table 1-7

Header	Value
Cookie:	Cookie

If you want the application to set arbitrary values for these headers or to add additional headers, use sceHTTPAddHeaderList() to specify arbitrary headers and values, then use the sceHTTPO_RequestHeaders code of sceHTTPSetOption() to add the headers.

However, the headers that are provided separately by the setting API as described above, are set with that API.

```
sceHTTPClient_t *client;
sceHTTPHeaderList_t headers, *list;

/* Create client structure */
client = sceHTTPCreate();

/* Add X-XXX: foobar to the header list */
if((list = sceHTTPAddHeaderList(headers, "X-XXX", "foobar"))
 == NULL){
       printf("sceHTTPAddHeaderList() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
/* Register header list in the client structure */
if(sceHTTPSetOption(client, sceHTTPO_RequestHeaders, list, 0)
 < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}

/* Set User-Agent: header to "SCEI Brower 1.0" */
```

```
if(sceHTTPSetOption(client, sceHTTPO_ClientName, "SCEI Brower
 1.0") < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
```

Getting Optional Headers From The HTTP Response And The Body

To get the optional headers contained in the HTTP response, access the headers member of the sceHTTPResponse_t structure.

To get the HTTP response body, access the entity member of the HTTP response structure.

The body length can be found from the length member.

```
sceHTTPClient_t *client;
sceHTTPResponse_t *response;
sceHTTPHeaderList_t *list;
int i;

/* Get HTTP response */
response = sceHTTPGetResponse(client);

/* Access optional headers */
list = response->headers;
for(list = response->headers; list != NULL; list =
sceHTTPNextHeader(list)){
       printf("response header:\n");
       printf("%s: %s\n", list->name, list->value);
}

/* Get body */
printf("response body:\n");
for(i = 0; i < response->length; i++)
       printf("%c", response->entity[i]);
```

Setting The HTTP Request Body

For the POST method, the HTTP request body is set with the sceHTTPO_RequestEntity code of sceHTTPSetOption().

```
sceHTTPClient_t *client;
char entity[256];

/* Create client structure */
client = sceHTTPCreate();

/* Set contents of request body */
strncpy(entity, "foo bar", 256);

/* Set request body */
if(sceHTTPSetOption(client, sceHTTPO_RequestEntity, entity,
strlen(entity), sceHTTPInputF_LINK) < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
```

```
}
```

Setting Timeouts And Aborting HTTP Processing

You can set timeouts in case no HTTP response is returned from the server. By default, the timeout values are infinite.

The two available types of timeouts are the "response header acquisition timeout," which applies until the HTTP response header is obtained and the "response body acquisition timeout," which applies until the response body is obtained.

HTTP processing can also be aborted using sceHTTPAbortRequest().

```
sceHTTPClient_t *client;

/* Create client structure */
client = sceHTTPCreate();

/* Set timeouts (60 seconds for header timeout and 120 seconds
   for body timeout) */
if(sceHTTPSetOption(client, sceHTTPO_ResponseTimeout, 60) <
0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
if(sceHTTPSetOption(client, sceHTTPO_TransferTimeout, 120) <
0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
```

Setting A Proxy

Connecting via a proxy is also supported for the client. To use a proxy, set the proxy server URI in addition to the target URI in the client structure.

```
sceHTTPClient_t *client;
sceHTTPParsedURI_t *p;

/* Create client structure */
client = sceHTTPCreate();

/* Convert proxy URI (http://proxy.foobar.com) to URI
   structure for HTTP library */
if((p = sceHTTPParseURI("http://proxy.foobar.com", 0)) == 0){
       printf("sceHTTPParseURI() failed.\n");
       ERR_STOP;
}
/* Set proxy URI in the client structure */
if(sceHTTPSetOption(client, sceHTTPO_ProxyURI, p) < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
```

Non-Blocking Mode

With libhttp, blocking is performed by default from the time an HTTP request is sent until an HTTP response is received. However, processing can also wait in non-blocking mode. In this case, a callback function will be called after the HTTP response has been received. This callback function can be registered in advance, and at the same time, a user-defined argument for the callback function can be assigned. This argument will take the value of the third argument of the callback function when it is called.

Non-blocking mode is set by executing sceHTTPOption() with the sceHTTPO_BlockingMode code, and the termination callback function and user-defined argument are registered by executing sceHTTPOption() with the sceHTTPO_EndOfTransactionCB code.

```
static void end_cb(sceHTTPClient_t *p, int flags, void *uopt){
       ...
}

sceHTTPClient_t *client;
void *uopt;

/* Create client structure */
client = sceHTTPCreate();

/* Set non-blocking mode */
if(sceHTTPSetOption(client, sceHTTPO_BlockingMode, 0) < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
/* Register termination callback function and user-defined
   argument */
uopt = ...;
if(sceHTTPSetOption(client, sceHTTPO_EndOfTransactionCB,
 end_cb, uopt) < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
```

Redirection

When redirection is indicated from the server by the presence of a Location header, the library provides an API that converts it to a URI structure and reflects it in the client structure.

```
sceHTTPClient_t *client;
sceHTTPResponse_t *response;
sceHTTPParsedURI_t *p;
char **locations;
int ret;

/* Get HTTP response */
response = sceHTTPGetResponse(client);

/* Termination processing. Close the connection. */
sceHTTPClose(client);

/* Determine redirection processing */
switch(response->code){
case sceHTTPC_MultipleChoices:
case sceHTTPC_MovedPermanently:
```

```
case sceHTTPC_Found:
case sceHTTPC_SeeOther:
case sceHTTPC_TemporaryRedirect:
case sceHTTPC_UseProxy:
       /* Get URI of Location header in string array */
       if((locations = (char
       **)sceHTTPParseLocations(response)) == NULL){
              printf("can't redirect.\n");
              ERR_STOP;
       }

       /* The first URI of the Location header is assumed to be
          selected here */
       if((p = sceHTTPParseURI(locations[0],
       (sceHTTPParseURI_FILENAME
              | sceHTTPParseURI_SEARCHPART))) == NULL){
              printf("bad URI.\n");
              ERR_STOP;
       }

       /* Free redirection URI string array */
       if((ret = sceHTTPFreeLocations(locations)) < 0){
              printf("sceHTTPFreeLocations() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
       }

       /* Reflect target redirection URI in client structure */
       if((ret = sceHTTPSetRedirection(client, p,
                           (response->code ==
                            sceHTTPC_UseProxy))) < 0){
              printf("sceHTTPSetRedirection() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
       }

       /* Initialize client structure */
       if((ret = sceHTTPCleanUpResponse(client)) < 0){
              printf("sceHTTPCleanUpResponse() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
       }
       if(sceHTTPSetOption(client, sceHTTPO_RequestHeaders,
       NULL, 1) < 0){
              printf("sceHTTPSetOption() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
       }

       /* Reconnect to redirect target */
       if(sceHTTPOpen(client) < 0){
              printf("sceHTTPOpen() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
       }

       ...

default:
       break;
```

```
}
```

Authentication

libhttp supports digest authentication and basic authentication.

```
sceHTTPClient_t *client;
sceHTTPResponse_t *response;
sceHTTPAuthList_t *auth_list;
sceHTTPDigest_t dig;
int ret;

/* Get HTTP response */
response = sceHTTPGetResponse(client);

/* Termination processing. Close the connection. */
sceHTTPClose(client);

/* Determine authentication processing */
switch(response->code){
case sceHTTPC_Unauthorized:
       /* Get HTTP authentication structure. */
       if((auth_list = sceHTTPParseAuth(response)) == NULL){
              printf("can't get auth info.\n");
              ERR_STOP;
       }

       /* The first www-authenticate header is assumed to be
          selected here */
       if(auth_list->auth.type == sceHTTPAuth_BASIC){
              /* Basic authentication. */
              sceHTTPSetBasicAuth(client, "foo", "bar");
       }
       else{
              /* Digest authentication */

              /* Set digest request structure */
              dig.username = "foo";
              dig.password = "bar";
              ...

              /* Set digest authentication in client structure
                 */
              if((ret = sceHTTPSetDigestAuth(client, &dig)) <
              0){
                     printf("sceHTTPSetDigestAuth()
                     failed.(%d)\n",
                            sceHTTPGetClientError(client));
                     ERR_STOP;
              }
       }

       /* Free authentication structure */
       if((ret = sceHTTPFreeAuthList(auth_list)) < 0){
              printf("sceHTTPFreeAuthList() failed.(%d)\n",
                     sceHTTPGetClientError(client));
              ERR_STOP;
       }
```

```
        /* Initialize client structure */
        if((ret = sceHTTPCleanUpResponse(client)) < 0){
                printf("sceHTTPCleanUpResponse() failed.(%d)\n",
                        sceHTTPGetClientError(client));
                ERR_STOP;
        }
        if(sceHTTPSetOption(client, sceHTTPO_RequestHeaders,
        NULL, 1) < 0){
                printf("sceHTTPSetOption() failed.(%d)\n",
                        sceHTTPGetClientError(client));
                ERR_STOP;
        }

        /* Reconnect to the server */
        if(sceHTTPOpen(client) < 0){
                printf("sceHTTPOpen() failed.(%d)\n",
                        sceHTTPGetClientError(client));
                ERR_STOP;
        }

        ...

default:
        break;
}
```

Controlling Cookies

When a cookie is returned from the server, sceHTTPParseCookie() can be used to obtain the cookie as a structure, then sceHTTPSetCookie() can be used to reflect it in the client structure. The cookie will become effective the next time the client connects to the server.

```
sceHTTPClient_t *client;
sceHTTPResponse_t *response;
sceHTTPCookieList_t *cookie;
int ret;

/* Get HTTP response */
response = sceHTTPGetResponse(client);

/* Termination processing. Close the connection. */
if((ret = sceHTTPClose(client)) < 0){
        printf("sceHTTPClose() failed.(%d)\n",
                sceHTTPGetClientError(client));
        ERR_STOP;
}

/* Get cookie */
if((cookie = sceHTTPParseCookie(response)) == NULL){
        return 0;
}

/* Initialize client structure */
if((ret = sceHTTPCleanUpResponse(client)) < 0){
        printf("sceHTTPCleanUpResponse() failed.(%d)\n",
                sceHTTPGetClientError(client));
        ERR_STOP;
}
```

```
if(sceHTTPSetOption(client, sceHTTPO_RequestHeaders, NULL, 1)
 < 0){
       printf("sceHTTPSetOption() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}

/* Set cookie in client structure */
if((ret = sceHTTPSetCookie(client, cookie)) < 0){
       printf("sceHTTPSetCookie() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
/* Free cookie structure */
if((ret = sceHTTPFreeCookieList(cookie)) < 0){
       printf("sceHTTPFreeCookieList() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}

/* Reconnect to server */
if(sceHTTPOpen(client) < 0){
       printf("sceHTTPOpen() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
```

MIME Processing

libhttp provides MIME processing functions for getting data that has been MIME multipart-encoded.

MIME processing uses the concept of filters for processing a MIME multipart-encoded document. There is one filter for each multipart level.

The application first creates a root filter for the MIME document to be processed. sceHTTPMimeFilterParseHeaders() is used to perform header processing for that root filter, and sceHTTPMimeFilterApply() is used to process the parts and place them into a filter structure. As sceHTTPMimeFilterParseHeaders() and sceHTTPMimeFilterApply() are repeated, each part at that level is processed sequentially.

If sublevels exist, a filter for a sublevel will be automatically generated within sceHTTPMimeFilterParseHeaders(). When sceHTTPMimeFilterApply() finishes processing the part for the last sublevel, the subfilter will be automatically freed.

```
sceHTTPClient_t *client;
sceHTTPResponse_t *response;
sceHTTPMimeFilter_t *filter;
int ret;

/* Get HTTP response */
response = sceHTTPGetResponse(client);

/* Create MIME root filter for HTTP response */
if((filter = sceHTTPMimeFilterCreate(sceHTTPMimeFilter_STRING,
       (void *)response->entity, response->length,
       sceHTTPMimeFilter_STRING, (void *)0)) == NULL){
       printf("sceHTTPMimeFilterCreate() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
```

```
}

/* Parse MIME header and add subpart filter to root filter
   list */
if((ret = sceHTTPMimeFilterParseHeaders(filter)) < 0){
       printf("sceHTTPMimeFilterParseHeaders() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}

/* Perform processing for subpart 1. Check whether a sublevel
    really exists and perform required processing. However,
    this processing requires that the layout of parts be flat.
    */
if((ret = sceHTTPMimeFilterApply(filter, NULL)) < 0){
       printf("sceHTTPMimeFilterApply() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
/* Free header list of subpart 1 */
if((ret = sceHTTPFreeHeaderList(filter->headers)) < 0){
       printf("sceHTTPFreeHeaderList() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
filter->headers = NULL;

/* Perform processing for subpart 2 */
if((ret = sceHTTPMimeFilterApply(filter, NULL)) < 0){
       printf("sceHTTPMimeFilterApply() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
/* Free header list of subpart 2 */
if((ret = sceHTTPFreeHeaderList(filter->headers)) < 0){
       printf("sceHTTPFreeHeaderList() failed.(%d)\n",
              sceHTTPGetClientError(client));
       ERR_STOP;
}
filter->headers = NULL;

...
```

Chunk Receive Processing

libhttp supports the receipt of HTTP/1.1 chunk-format encoding. Specifically, it enables a chunk receive callback function to be set with the sceHTTPO_ReceiveChunkCB code of sceHTTPSetOption(). At the same time, a user-defined argument for the callback function can be assigned. This argument will take the value of the fourth argument of the callback function when it is called.

```
static void end_cb(sceHTTPClient_t *client, int flags, void
*uopt){
       ...
}

static void chunk_cb(sceHTTPClient_t *client, char *cdata, int
clen, void *uopt){
```

```
        ...
}

sceHTTPClient_t *client;
void *uopt;

/* Create client structure */
client = sceHTTPCreate();

uopt = ...;
/* Set chunk receive callback function and user-defined
   argument */
if(sceHTTPSetOption(client, sceHTTPO_ReceiveChunkCB, chunk_cb,
 uopt) < 0){
        printf("sceHTTPSetOption() failed.(%d)\n",
                sceHTTPGetClientError(client));
        ERR_STOP;
}
```

Encoding and Decoding

libhttp provides routines for encoding and decoding the HTTP body. The following encoding methods, and encoding and decoding functions, are supported.

- Base64 sceBASE64Encoder(), sceBASE64LineDecoder()
- quoted-printable sceQPrintableEncoder(), sceQPrintableLineDecoder()
- url-encode sceURLEscape(), sceURLUnescape()

Precautions

About Error Codes

When an error occurs internally in libhttp, each function will return a value that indicates an error, and the detailed error cause can be obtained with sceHTTPGetClientError().

Response code errors specified by HTTP, which are not library errors, can be obtained with sceHTTPErrorString().

About User Agents

libhttp provides "unknown (sceHTTPLib-X.X.X)" (X.X.X is the version number) as the default User-Agent header value. However, since this value is only for use in sample programs, the sceHTTPO_ClientName code of sceHTTPSetOption() must be used to overwrite the User-Agent value with an actual value used in a title.

Chapter 2:
Network Socket Library Overview

Library Overview

libinsck is a socket API-compatible wrapper library that runs on top of the network library (inet). It enables a general socket API to be used on top of inet that can handle the network. Although inet runs on the IOP, a library for transparently using the inet API from the EE is provided as sample code in ee/sample/inet/libnet. libinsck is an EE library that runs on top of libnet.

Each socket API function in libinsck has a function name which starts with the prefix sceInsock. General API names are provided as aliases by #define in the include file.

Example: For bind()

API: sceInsockBind()

Alias: #define bind sceInsockBind()

However, libinsck does not provide all socket API functions that are available under UNIX. For information about the functions that are provided, see the corresponding reference.

Related Files

The following files are required to use libinsck.

Table 2-1

Category	File Name
Header files	sample/inet/libnet/libnet.h
	include/libmrpc.h
	include/libinsck.h
	(the following five files are included internally)
	include/ libinsck /netdb.h
	include/ libinsck /arpa/inet.h
	include/ libinsck /netinet/in.h
	include/ libinsck /netinet/tcp.h
	include/ libinsck /sys/socket.h
Library files	libinsck.a
	libnet.a
Module files	netcnf.irx
	inet.irx
	inetctl.irx
	libmrpc.irx
	libnet.irx

For compatibility with the socket API, the header files also provide socket API functions in the include/libinsck/ directory. Normally, you need only include include/libinsck.h, which internally includes all of these socket API functions.

Since libinsck uses libnet internally, libnet.a and libnet.irx must be created in advance. libnet is provided in the sample code.

Sample Programs

The following sample programs that use libinsck were obtained by changing the samples that were provided with inet to use the socket API.

- ee/sample/inet/socket/http_get — Sample program for getting index.html from a specified web server
- ee/sample/inet/socket/echo_server — Sample TCP server that runs on the echo port

Using libinsck Together With the libnet API

Since libinsck is provided on top of libnet, the API provided by libnet (the inet API) can also be used at the same time as the socket API. However, the inet functions must be applied to a socket that has already been created; no API is provided otherwise.

Initialization Procedure

libinsck uses the libnet initialization procedure asis.

The initialization procedure is as follows. For details, refer to the individual samples.

Initialize SIFRPC

```
sceSifInitRpc(0);
```

Load Each Module

```
#define MAIN_MOD_ROOT            "host0:/usr/local/sce/iop/modules/"
#define IOP_MOD_INET             MAIN_MOD_ROOT "inet.irx"

while(sceSifLoadModule(IOP_MOD_INET, 0, NULL) < 0)
       printf("inet.irx cannot be loaded.\n");

...
```

Initialize MSIFRPC

```
sceSifMInitRpc(0);
```

Initialize the libnet Library

```
sceSifMClientData cd;

...

libnet_init(&cd);
```

Load Network Settings on Stack

Wait for network interface initialization

```
 #define ERR_STOP    while(1)
 #define NETBUFSIZE 512
 #define NET_DB           "host0:/usr/local/sce/conf/net/net.db"
```

```
#define USR_CONF_NAME     "Combination4"

static u_int net_buf[NETBUFSIZE]  __attribute__((aligned(64)));

...

int if_id;
struct sceInetAddress myaddr;

if((ret = reg_handler(&cd, net_buf)) < 0){
      printf("reg_handler() failed.\n");
      ERR_STOP;
}
if((ret = load_set_conf(&cd, net_buf, NET_DB, USR_CONF_NAME)) < 0)
{
      printf("load_inet_cnf() failed.\n");
      ERR_STOP;
}
if((ret = wait_get_addr(&cd, net_buf, &if_id, &myaddr)) < 0){
      printf("wait_get_addr() failed.\n");
      ERR_STOP;
}
```

errno and h_errno

A problem occurs when an errno that is provided by the current EE libc.a is directly accessed from multiple threads of the EE kernel. As a result, libinsck keeps internally an errno for each thread independently, and these are used for errno or h_errno by the socket API. Access to these independent errno or h_errno is performed using the sceInsockErrno and sceInsockHErrno macros.

When libinsck is used by a single thread, the EE errno may be used.

Relationship to I/O Functions Such as sceRead() and sceWrite()

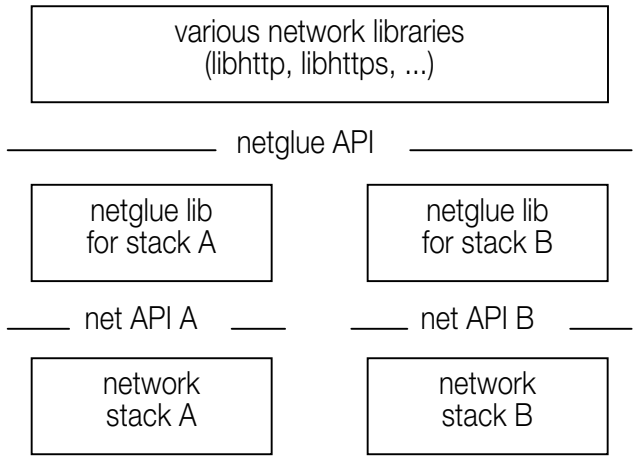
With a socket API such as that available under UNIX, a general I/O function such as read() or write can be used for input from or output to a socket. However, sceRead(), sceWrite(), sceClose() and other functions provided on the EE cannot be used with a socket that was created using libinsck.

Chapter 3: General-Purpose Network Wrapper API (netglue)

Library Overview

The general-purpose network wrapper API (netglue) is a low-level network library that has a BSD socket-based API. It was designed so that various kinds of network-related libraries can be used independently of the network stack library.

Figure 3-1



To use a library that created netglue as a low-level API, a netglue API wrapper corresponding to the network stack library to be used is required.

Since a netglue library for the SCEI stack (libinet) is included in the SDK, you need not create a netglue library and you need not be aware of the netglue API.

To use a library that uses netglue with a network stack library other than libinet, you must create a netglue library.

Related Files

The following files are required to create a netglue library. The only libraries that are required when an application uses a network-related library that supports netglue, are each network stack library and the netglue library that it uses.

Table 3-1

Category	Filename
Header file	include/netglue.h
	(the following three files are included internally)
	include/netglue/netdb.h
	include/netglue/arpa/inet.h
	include/netglue/netinet/in.h
	include/netglue/netinet/tcp.h
	include/netglue/sys/socket.h
	Header file for each network stack library being used
Library files	Each network stack library

About the API

netglue API

The netglue API basically conforms to the BSD socket API. The netglue API is listed below. To create a netglue library, all of the following APIs must be supported.

For details about each API, refer to the library reference.

Table 3-2

Function name	Function description
sceNetGlueAbort()	Aborts specified socket processing
sceNetGlueThreadInit()	Performs initialization for thread that uses netglue
sceNetGlueThreadTerminate()	Performs termination processing for thread that uses netglue
__sceNetGlueErrnoLoc()	Gets socket function error value
__sceNetGlueHErrnoLoc()	Gets host structure function error value
sceNetGlueGethostbyaddr()	Gets host structure from 32-bit IPv4 address
sceNetGlueGethostbyname()	Gets host structure from host name
sceNetGlueInetAddr()	Gets 32-bit address from dot-format IPv4 address
sceNetGlueInetAton()	Gets 32-bit address from dot-format IPv4 address
sceNetGlueInetLnaof()	Gets local network address from IPv4 address
sceNetGlueInetMakeaddr()	Constructs IPv4 address from network address
sceNetGlueInetNetof()	Gets network address from IPv4 address
sceNetGlueInetNetwork()	Gets 32-bit address from dot-format IPv4 address
sceNetGlueInetNtoa()	Gets dot-format address from 32-bit IPv4 address
sceNetGlueHtonl()	Converts 4-byte numeric value from local byte order to network byte order
sceNetGlueHtons()	Converts 2-byte numeric value from local byte order to network byte order
sceNetGlueNtohl()	Converts 4-byte numeric value from network byte order to local byte order
sceNetGlueNtohs()	Converts 2-byte numeric value from network byte order to local byte order
sceNetGlueAccept()	Gets socket with established TCP connection
sceNetGlueBind()	Binds address to socket
sceNetGlueConnect()	Connects to server
sceNetGlueGetpeername()	Gets connection destination information for socket
sceNetGlueGetsockname()	Gets local information for socket
sceNetGlueGetsockopt()	Gets socket option
sceNetGlueListen()	Accepts TCP connection
sceNetGlueRecv()	Receives data

Function name	Function description
sceNetGlueRecvfrom()	Receives data (also gets address structure of sending host)
sceNetGlueSend()	Sends data
sceNetGlueSendto()	Sends data (specifies address structure of destination host)
sceNetGlueSetsockopt()	Sets socket option
sceNetGlueShutdown()	Closes socket
sceNetGlueSocket()	Creates socket

BSD Socket API

Almost all of the functions of the netglue API conform to the BSD socket API. Blocking is a prerequisite for each BSD socket API of netglue. To cancel blocking, call sceNetGlueAbort() from another thread. sceNetGlueAbort() is a unique netglue API and is described below.

Unique netglue APIs

netglue has the following three unique APIs in addition to the BSD socket APIs.

sceNetGlueAbort()

Cancels blocking of BSD socket APIs by other threads for the specified socket.

sceNetGlueThreadInit()

Performs initialization processing so that internal information can be maintained for each thread when the netglue library is used. A thread that will use netglue should call this function first.

sceNetGlueThreadTerminate()

Performs termination processing corresponding to sceNetGlueThreadInit(). A thread that uses netglue should call this function when it terminates.

About errno

The error value for each netglue API is obtained using sceNetGlueErrno, and each API simply returns a negative value when an error occurs, in a similar manner as the BSD socket API. Refer to the library reference for each error value of errno.

Chapter 4:
Network Configuration GUI Library

Library Overview

The network configuration GUI library (ntgui) is a program for adding, editing, deleting, copying and selecting "Your Network Configuration" files, which can be shared by applications that use the network. ntgui is designed to conform to the TRC (the master validation document), including the user interface. By incorporating ntgui in a title application, the effort required to develop code for manipulating or selecting "Your Network Configuration" files can be reduced.

Related Files

The following files are required by ntguicnf.

Table 4-1

Category	Filename
Library files	libkernel.a
	libgraph.a
	libvu0.a
	libpad.a
	libmc.a
	ntguicnf.a
Header file	ntguicnf.h
Module files	sio2man.irx
	padman.irx
	mcman.irx
	mcserv.irx
	netcnf.irx
	inet.irx
	inetctl.irx
	ppp.irx
	pppoe.irx
	usbd.irx
	usbmload.irx (only if the USB hardware device driver is autoloaded)
	dev9.irx (only if using a hard disk drive)
	atad.irx (only if using a hard disk drive)
	hdd.irx (only if using a hard disk drive)
	pfs.irx (only if using a hard disk drive)
	ntguicnf.irx
	and hardware device driver being used

Note: The only memory card that is supported by ntguicnf is the PS2 Memory Card.

Functions That Must be Defined

A program that uses ntguicnf must define the following callback functions. For details about each callback function, refer to the ntguicnf library reference.

Memory Allocation Functions

sceNetGuiCnfCallback_Malloc

sceNetGuiCnfCallback_Memalign

sceNetGuiCnfCallback_Realloc

sceNetGuiCnfCallback_Free

Software Keyboard Functions

sceNetGuiCnfCallback_SKBInit

sceNetGuiCnfCallback_SKBDestroy

sceNetGuiCnfCallback_SKBGetVifPktTopAddr

sceNetGuiCnfCallback_SKBGetStatus

sceNetGuiCnfCallback_SKBSendMouseMessage

sceNetGuiCnfCallback_SKBEnableKey

sceNetGuiCnfCallback_SKBEveryFrame

Character Conversion Functions

sceNetGuiCnfCallback_SJIStoUTF8

sceNetGuiCnfCallback_UTF8toSJIS

USB Mouse Function (only if using a USB Mouse)

sceNetGuiCnfCallback_UsbMouseRead

USB Keyboard Function (only if using a USB Keyboard)

sceNetGuiCnfCallback_UsbKbRead

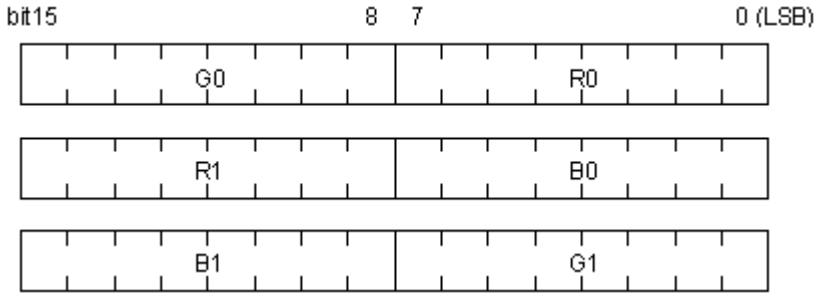
Controller Function

sceNetGuiCnfCallback_PadRead

Background Files

ntguicnf can swap backgrounds that correspond to application images by reading in backgrounds as files. A background file is limited to 640 x 448 pixels consisting of R (8 bits) + G (8 bits) + B (8 bits) binary data as shown below.

Figure 4-1



R0, R1: Red components (8 bits)

G0, G1: Green components (8 bits)

B0, B1: Blue components (8 bits)

Library Startup and Shutdown

The main processing for ntgui is performed by calling the sceNetGuiCnf_Do() function. Once the appropriate initialization is performed, sceNetGuiCnf_Do() should be called and a screen for selecting and manipulating "Your Network Configuration" files will be displayed. Subsequently, a network configuration can be selected and manipulated by the user.

The initialization processing that is needed before sceNetGuiCnf_Do() is called by the application, and the required termination processing after sceNetGuiCnf_Do() has finished are explained below.

Initialization Processing Needed Before sceNetGuiCnf_Do() is Called

1. Load required IOP modules

Required IOP modules are not loaded by sceNetGuiCnf_Do(). The title application must load IOP modules that are needed by sceNetGuiCnf_Do() before the function is called. For information about required modules, see "Related Files" in the Library Overview.

2. Define required callback functions

sceNetGuiCnf_Do() uses arguments to obtain functions that are used as callbacks, such as a memory allocation function or software keyboard function. Callback functions allow behavior to change depending on the application's purpose or image. Consequently, the following types of functions must be defined by the application.

- Memory allocation function
- Software keyboard function
- Character conversion function
- USB mouse function (only if using a USB mouse)
- USB keyboard function (only if using a USB keyboard)
- Controller function

For information about these types of functions and their definitions, see "Functions That Must be Defined" in the Library Overview. Also, refer to the Library Reference.

3. Create a semaphore that performs SignalSema when a v-blank occurs

After one frame's worth of processing has completed, sceNetGuCnf_Do() will issue WaitSema() and sleep. Since the semaphore that is reported to WaitSema() is not created internally by sceNetGuiCnf_Do(), the application must use CreateSema() to create a semaphore and pass the ID of that semaphore to sceNetGuiCnf_Do(). Also, SignalSema is not invoked within sceNetGuiCnf_Do() so the application must include a mechanism for invoking SignalSema when a v-blank occurs.

4. Terminate libpad

Internally, sceNetGuiCnf_Do() initializes libpad with the following procedure.

```
scePadInit( 0 );
scePadPortOpen( 0, 0, ... );
```

As a result, the application must perform the following sort of termination processing for libpad before invoking sceNetGuiCnf_Do().

```
scePadPortClose( 0, 0 )
scePadEnd( );
```

5. Configure the background file path

sceNetGuiCnf_Do() reads in the background from a file. As a result, the application must pass the path where the background file resides as an argument to sceNetGuiCnf_Do(). Moreover, since the file path is accessed as a string by a pointer reference in sceNetGuiCnf_Do(), the application must allocate this string in a static area.

6. Configure a buffer for receiving the configuration

The application must provide a buffer that sceNetGuiCnf_Do() will use to receive the user-selected configuration. A static area must be allocated for the buffer, and it will be referenced by a pointer in sceNetGuiCnf_Do().

The buffer is defined by the sceNetGuiCnfEnvData structure.

7. Set up the buffer for receiving the name of the selected settings and its device

The application must provide a buffer that sceNetGuiCnf_Do() will use to receive the name of the user-selected settings. A static area must be allocated for the buffer, and it will be referenced by a pointer in sceNetGuiCnf_Do().

The buffer for receiving the name of the selected settings and its device is defined by the sceNetGuiCnf_Selected structure.

8. Set default values to be used when adding (optional)

To set default values to be used when adding, the application must define a sceNetGuiCnfEnvData structure where the default values can be set. A static area must be allocated for this structure, and it will be referenced by a pointer in sceNetGuiCnf_Do().

When setting default values to be used when adding, all of the following members of the sceNetGuiCnfEnvData structure must be set. Members other than those listed below are ignored.

address	IP address
netmask	Netmask
gateway	Default router
dns1_address	Primary DNS
dns2_address	Secondary DNS
phone_numbers1	Tel. Number1
phone_numbers2	Tel. Number2
phone_numbers3	Tel. Number3
auth_name	User ID
auth_key	Password
chat_additional	Additional AT command
outside_set	Outside number
dhcp_host_name	DHCP host name
dialing_type	Dialing type
idle_timeout	Line timeout (minutes)
phy_config	Ethernet hardware operating mode
dhcp	DHCP used/unused setting
pppoe	PPPoE (PPP over Ethernet) used/unused setting

For details about the values that can be set for each member, refer to the "Guidelines for Creating a Network Configuration Application" document. To not configure a string-format member, set '\0' at the str[0] position.

9. Configure startup options

To change the startup method as needed by the application, you must configure startup options. Startup options are defined with the flag member of the sceNetGuiCnf_Arg structure. The following startup options are available.

Table 4-2

Constant	bit	Meaning
SCE_NETGUICNF_FLAG_USE_HDD	0	Use hard disk drive
SCE_NETGUICNF_FLAG_USE_USB_MOUSE	1	Use USB mouse
SCE_NETGUICNF_FLAG_USE_USB_KB	2	Use USB keyboard
SCE_NETGUICNF_FLAG_USE_SELECT_OPTION	3	Enable the bit 3 startup option
SCE_NETGUICNF_FLAG_SELECT_ONLY	4	0: Skip configuration selection 1: Select configuration only
SCE_NETGUICNF_FLAG_MC_SLOT1_ONLY	5	Only use memory card slot 1

When a hard disk drive is used and the SCE_NETGUICNF_FLAG_USE_HDD bit is set to 1, the hdd0:__sysconf partition will be mounted on pfs0 by sceNetGuiCnf_Do(). Therefore, the application must delete any mount on pfs0 before using sceNetGuiCnf_Do().

10. Initialize USB device driver autoloading

No setting is performed by sceNetGuiCnf_Do() to autoload the USB device driver. The application must perform USB device driver autoload initialization processing when usbmload.irx is loaded.

Termination Processing Needed After Calling sceNetGuiCnf_Do()

1. Module loaded in the IOP (optional)

The IOP is not reset by sceNetGuiCnf_Do().

To delete IOP modules that are no longer needed after using sceNetGuiCnf_Do(), you must reset the IOP or unload the module.

An application that subsequently reuses sceNetGUiCnf_Do() must reload the required IOP modules.

2. Reinitialize the graphic system

sceNetGuiCnf_Do() will internally initialize the graphic system. If the application is to display a graphic again after using sceNetGuiCnf_Do(), the graphic system must be reinitialized.

3. Reinitialize libpad

When processing ends, sceNetGuiCnf_Do() performs libpad termination processing with the following procedure.

```
scePadPortClose( 0, 0 )
scePadEnd( );
```

If the application is to reuse libpad after using sceNetGuiCnf_Do(), libpad must be reinitialized.

4. Initialize libmc (optional)

The sceNetGuiCnf_Do() internally initializes libmc with the following procedure.

```
sceMcInit( );
```

If the application is to reuse libmc after using sceNetGuiCnf_Do(), libmc must be reinitialized with sceMcInit(), as necessary.

Library Functions

Selecting a Network Configuration

1. **Selecting the configuration during startup**

   When sceNetGuiCnf_Do() starts, an automatic selection of the configuration is performed as follows.

   - If combinations are present, the combination that was added / selected last is selected in priority order of PS2 memory card, hard disk drive.
   - If no combination is present, the hardware settings and network service provider settings that were added / selected last are selected in priority order of PS2 memory card, hard disk drive.

2. **Selecting with a combination**

   sceNetGuiCnf_Do() can select a combination that combines previously registered hardware settings and network service provider settings. When sceNetGuiCnf_Do() is terminated, the configuration is read in from the PS2 memory card or hard disk drive and its contents are returned in the buffer pointed to by the *result_env_data* or *selected_configuration* member of the sceNetGuiCnf_Arg structure.

   Example:

Combination	Combination 1
Hardware	SCE/Ethernet (Network Adaptor)
Network service provider	Local Area Network settings

3. **Selecting hardware and network service provider settings individually**

   sceNetGuiCnf_Do() can select hardware and network service provider settings individually without being aware of the device where the settings were saved. When sceNetGuiCnf_Do() is terminated, the settings are individually read in from the PS2 memory card or hard disk drive and the contents are returned in the buffer pointed to by the *result_env_data* or *selected_configuration* member of the sceNetGuiCnf_Arg structure. Settings that have not been saved as files can also be selected in a similar manner.

   Example:

Combination	(Unselected)
Hardware	SCE/Ethernet (Network Adaptor)
Network service provider	Local Area Network settings

Saving to a Memory Card

Among memory cards, only the PS2 memory card is supported by sceNetGuiCnf_Do(). The slots that are supported are given below.

If the startup option SCE_NETGUICNF_FLAG_MC_SLOT1_ONLY bit is 1

- Only slot 1 is supported.

If the startup option SCE_NETGUICNF_FLAG_MC_SLOT1_ONLY bit is 0

- Both slots 1 and 2 are supported. However, slot 1 takes precedence when automatic slot recognition is performed.

When the latter startup option is selected, that condition must be described in the reference manual of the application. For other information about saving the network configuration to a memory card, refer to the "Guidelines for Creating a Network Configuration Application" document.

Saving to a Hard Disk Drive

When the startup option SCE_NETGUICNF_FLAG_USE_HDD bit is 1, sceNetGuiCnf_Do() allows the network configuration to be saved to the hard disk drive. The network configuration is saved in the /etc/network directory in the hdd0:__sysconf partition which was mounted on pfs0. For information about saving the network configuration to a hard disk drive, refer to the "Guidelines for Creating a Network Configuration Application" document.

Adding Hardware Settings

sceNetGuiCnf_Do() can be used to add hardware settings. The user can set the following information when hardware settings are added.

For a modem or TA

- Dialing type
- Additional AT command
- Outside number
- Line timeout (minutes)

For USB Ethernet hardware or a network adaptor

- Ethernet hardware operating mode

The application can also set default values independently when adding hardware settings. The hardware devices that can be added are limited to devices that are physically connected and do not have existing settings (the settings don't exist on either a PS2 memory card or the hard disk drive, when both are used). Hardware settings can be saved either on a PS2 memory card or the hard disk drive. The user can also specify that settings should not be saved.

Editing Hardware Settings

sceNetGuiCnf_Do() can edit hardware settings. The user can set the following information when hardware settings are edited.

For a modem or TA

- Dialing type
- Additional AT command
- Outside number
- Line timeout (minutes)

For USB Ethernet hardware or a network adaptor

- Ethernet hardware operating mode

Any hardware settings that exist can be edited. Edited hardware settings can be saved either on a PS2 memory card or the hard disk drive. However, the edited settings must be saved on the same device where the selected hardware settings reside. The user can also specify that edited settings should not be saved.

Deleting Hardware Settings

sceNetGuiCnf_Do() can delete hardware settings. Any hardware settings that exist can be deleted.

Copying Hardware Settings

sceNetGuiCnf_Do() can copy hardware settings provided that both the PS2 memory card and hard disk drive are available. Any hardware settings that exist can be copied. Copied hardware settings can be saved either on a PS2 memory card or the hard disk drive. However, the selected hardware settings cannot be copied if they already exist on the destination device.

Adding Network Service Provider Settings

sceNetGuiCnf_Do() can be used to add network service provider settings. The user can set the following information when network service provider settings are added.

For a modem or TA

- User ID
- Password
- Tel. Number1
- Tel. Number2
- Tel. Number3
- Primary DNS (if you set the DNS server manually and it is not automatically obtained)
- Secondary DNS (if you set the DNS server manually and it is not automatically obtained)
- Setting name

For USB Ethernet hardware or a network adaptor

- User ID (when PPPoE (PPP over Ethernet) is used)
- Password (when PPPoE (PPP over Ethernet) is used)
- IP address (when neither PPPoE (PPP over Ethernet) nor DHCP is used)
- Netmask (when neither PPPoE (PPP over Ethernet) nor DHCP is used)
- Default router (when neither PPPoE (PPP over Ethernet) nor DHCP is used)
- Primary DNS (when PPPoE (PPP over Ethernet) is used and if you set the DNS server manually and it is not automatically obtained, and when neither PPPoE (PPP over Ethernet) nor DHCP is used)
- Secondary DNS (when PPPoE (PPP over Ethernet) is used and if you set the DNS server manually and it is not automatically obtained, and when neither PPPoE (PPP over Ethernet) nor DHCP is used)
- Setting name

Except for the setting name, the application can set default values independently when adding network service provider settings. The network service provider settings that can be added are limited to those with setting names that do not already exist. The added network service provider settings can be saved either on a PS2 memory card or on the hard disk drive. However, the added settings must be saved on the same device where the selected network provider settings reside. The user can also specify that settings should not be saved.

Editing Network Service Provider Settings

sceNetGuiCnf_Do() can edit hardware settings. The user can set the following information when network service provider settings are edited.

Network service provider settings that are created when a modem or TA was selected as the hardware device, when settings are added

- User ID
- Password
- Tel. Number1
- Tel. Number2
- Tel. Number3
- Primary DNS (if you set the DNS server manually and it is not automatically obtained)
- Secondary DNS (if you set the DNS server manually and it is not automatically obtained)
- Setting name

Network service provider settings created when USB Ethernet hardware or a network adaptor was selected as the hardware device, when settings are added and PPPoE (PPP over Ethernet) is used

- User ID
- Password
- Primary DNS (if you set the DNS server manually and it is not automatically obtained)
- Secondary DNS (if you set the DNS server manually and it is not automatically obtained)
- Setting name

Network service provider settings that are created when USB Ethernet hardware or a network adaptor was selected as the hardware device, when settings are added and PPPoE (PPP over Ethernet) is not used

- IP address (when DHCP is not used)
- Netmask (when DHCP is not used)
- Default router (when DHCP is not used)
- Primary DNS (when DHCP is not used)
- Secondary DNS (when DHCP is not used)
- Setting name

Any network service provider settings that exist can be edited. Edited network service provider settings can be saved either on a PS2 memory card or the hard disk drive. However, the edited settings must be saved on the same device where the selected network service provider settings reside. The user can also specify that edited settings should not be saved.

Deleting Network Service Provider Settings

sceNetGuiCnf_Do() can delete network service provider settings. Any network service provider settings that exist can be deleted.

Copying Network Service Provider Settings

sceNetGuiCnf_Do() can copy network service provider settings provided that both the PS2 memory card and hard disk drive are available. Any network service provider settings that exist can be copied. Copied network service provider settings can be saved either on a PS2 memory card or the hard disk drive. However, the selected network service provider settings cannot be copied if they already exist on the destination device.

Creating New Settings

The "New Settings" item enables sceNetGuiCnf_Do() to add hardware settings and network service provider settings at the same time. New settings can be added in the following two situations.

- When an existing hardware setting is selected
- When a physically connected hardware device that does not have an existing setting is selected

When an existing hardware setting is selected, the hardware device that is defined by that setting must be physically connected.

As soon as an existing setting is selected, processing will shift to the addition of a network service provider setting and no hardware setting will be added.

For the items that the user can set when adding a network service provider setting, see "Adding Network Service Provider Settings."

The application can independently set default values when adding network service provider settings. The network service provider settings that can be added are limited to those with setting names that do not already exist. The added network service provider settings can be saved either on a PS2 memory card or on the hard disk drive. However, the added settings must be saved on the same device where the selected network provider settings reside. The user can also specify that settings should not be saved.

When a hardware device that is physically connected but does not have an existing setting is selected, processing will shift to the addition of a hardware setting. For the items that the user can set when adding a hardware setting, see "Adding Hardware Settings."

The application can independently set default values when adding hardware settings.

After completing the hardware settings, processing will shift to the addition of a network service provider setting. For the items that the user can set when adding a network service provider setting, see "Adding a Network Service Provider Setting."

The application can independently set default values when adding network service provider settings. The network service provider settings that can be added are limited to those with setting names that do not already exist. The added network service provider settings can be saved either on a PS2 memory card or on the hard disk drive. The user can also specify that settings should not be saved.

After the network service provider settings have been saved, the hardware settings will also be saved. If processing for saving the network service provider setting fails, the hardware settings will not be saved. However, the network service provider settings are saved even if saving of the hardware settings fails.

Adding and Editing Combinations

sceNetGuiCnf_Do() can add hardware and network service provider settings to a combination. When a combination that has been added is selected, it can also be edited to combine different hardware and network provider settings. The hardware settings and network service provider settings that can be added and edited are limited to settings that satisfy the following conditions.

- The settings must be saved as files (a setting that was created by selecting "Do not save" cannot be added to a combination)
- The settings must reside on the same device (for example, a hardware setting that resides on a PS2 memory card can only be combined with a network service provider setting that also resides on the memory card)

If the above conditions are satisfied, the "Next" button for performing the add or edit can be pressed.

Deleting Combinations

sceNetGuiCnf_Do() can delete a combination. The combinations that can be deleted are limited to the following.

- Combinations that were created by adding a hardware setting and a network provider setting as a combination
- Combinations that were previously created by adding a hardware setting and a network service provider setting as a combination, but from which only the hardware setting, only the network service provider setting, or both the hardware setting and the network service provider setting were later deleted

If the above conditions are satisfied, the "Delete" button for performing the delete can be pressed.

Testing the Connection

sceNetGuiCnf_Do() can perform a connection test when a configuration that satisfies the following conditions is selected.

- A combination is selected, the hardware and network provider settings that have been added to the combination exist, and the hardware device specified in the hardware setting is physically connected
- A hardware setting and network service provider setting are selected and the hardware device specified in the hardware setting is physically connected

When the above conditions are satisfied, the "Connection Test" button for performing the connection test can be pressed. If the connection test succeeds, the connection is disconnected and a message indicating that the connection test succeeded is displayed. If the connection test failed, a message indicating the failure is displayed.

Determining Errors

When reading a network configuration that was saved by another PlayStation 2

When sceNetGuiCnf_Do() reads a network configuration that was saved by another PlayStation 2, it executes the following process.

1. First, a message is displayed indicating that the network configuration was saved by another PlayStation 2, then another message is displayed asking whether or not the entire network configuration of the device where that configuration resides should be deleted.
2. If "Delete" is selected, the deletion is reconfirmed. If "Do not delete" is selected, a message is displayed indicating that the entire network configuration of the device where that configuration resides cannot be used (it is unavailable).
3. If "Delete" is selected on the deletion reconfirmation screen, the entire network configuration of the device where that configuration resides is deleted. If "Do not delete" is selected, a message is displayed indicating that the entire network configuration of the device where that configuration resides cannot be used (it is unavailable).
4. After confirming that deletion has completed, initialization processing for the displayed screen is performed again.

When a network configuration that has been saved is corrupted

When sceNetGuiCnf_Do() reads a corrupted network configuration, it displays a message indicating that the network configuration is corrupted, then performs the same processing as if it were reading a network configuration that was saved by another PlayStation 2.

When an I/O error occurs

When an I/O error occurs, the sceNetGuiCnf_Do() function will display an appropriate error message, and the device that caused the error will be made unavailable.

Errors during addition, editing, deletion, or copying

When an error occurs during addition, editing, deletion, or copying, sceNetGuiCnf_Do() displays a message indicating that an error has occurred. The following types of errors may occur.

- The settings cannot be saved because the maximum number of settings that can be saved have been exceeded.
- There is insufficient free space available.
- A setting having the same name already exists.
- The memory card is not a PS2 memory card.
- Some other problem caused addition, editing, deletion, or copying to fail.

Errors while testing a connection

When an error occurs while testing a connection, sceNetGuiCnf_Do() displays a message indicating that an error has occurred. The following types of errors may occur.

- The connection test failed.
- The connection could not be made using the specified combination of hardware settings and network service provider settings.

USB Hardware Device Driver Autoload

sceNetGuiCnf_Do() can autoload the USB hardware device driver. The driver can be autoloaded by the application when it loads and initializes usbmload.irx. The autoload function can also be disabled by not loading usbmload.irx. However, in that case, the application must explicitly load the USB hardware device driver.

USB Mouse

A USB mouse can be used with sceNetGuiCnf_Do() by setting the SCE_NETGUICNF_FLAG_USE_USB_MOUSE bit to 1 in the *flag* member of the sceNetGuiCnf_Arg structure during startup. and setting the *cb_mouse_read* member of the sceNetGuiCnf_Arg structure appropriately.

USB Keyboard

A USB keyboard can be used with sceNetGuiCnf_Do() by setting the SCE_NETGUICNF_FLAG_USE_USB_KB bit to 1 in the *flag* member of the sceNetGuiCnf_Arg structure during startup and setting the *cb_kb_read* member of the sceNetGuiCnf_Arg structure appropriately.

Supported Controllers

The controllers that are supported by sceNetGuiCnf_Do() are not specifically defined. The title application can specify the supported controllers by specifying a function for reading controller button information in the *cb_pad_read* member of the sceNetGuiCnf_Arg structure.

Supported Connection Hardware

sceNetGuiCnf_Do() supports the following types of connection hardware. Vendor support depends on the configuration of the application.

- Modem or TA
- USB Ethernet hardware
- Network adaptor

Changing the Background and Colors

sceNetGuiCnf_Do() can change the background image of the network configuration application and the colors of the following components.

- Color of the title bar that is always displayed at the top of the screen
- Background color of the window that is always displayed in the center of the screen
- Color of each of the Quit, Back, and Next buttons that are always displayed at the bottom of the screen
- Color of the title bar of a one-choice dialog box (* In the current version, the color specified here will also be used for the title bar of an error dialog box)
- Color of the title bar of a two-choice dialog box
- Color of the title bar of an error dialog box (* Unused in the current version)
- Color of the title bar of a non-selectable dialog box

Network Application Operating Procedure

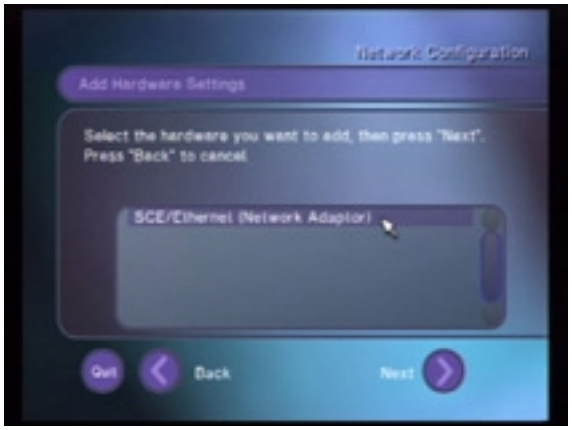
Adding Hardware Settings

The following procedure is used to add a hardware setting.

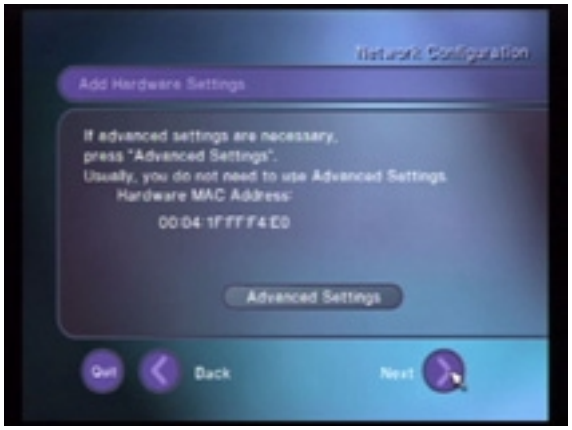
After starting up the "Network Configuration Selection and Connection Test" screen, click "Next." Depending on the setting of the flag member of the SceNetGuiCnf_Arg structure, this screen may be skipped and it may not be possible to click the "Next" button. For details, see "Configure startup options" in "Initialization Processing Needed Before sceNetGuiCnf_Do() is Called" of the "Library Startup and Shutdown" chapter in this document.

When the display changes to the "Connection Settings" screen, select "Hardware Settings."

When the display changes to the "Hardware Settings" screen, select "Add Hardware Settings."



The display will change to the "Add Hardware Settings" screen where hardware devices that are connected to the PlayStation 2 that do not have existing settings on the PS2 memory card or hard disk drive will be shown. Select the hardware device for which you wish to add a hardware setting, then click "Next."



The display will change to the setting screen corresponding to the selected hardware. (The above figure shows the screen when USB Ethernet hardware or a network adaptor is selected.) After setting the required item(s), click "Next."

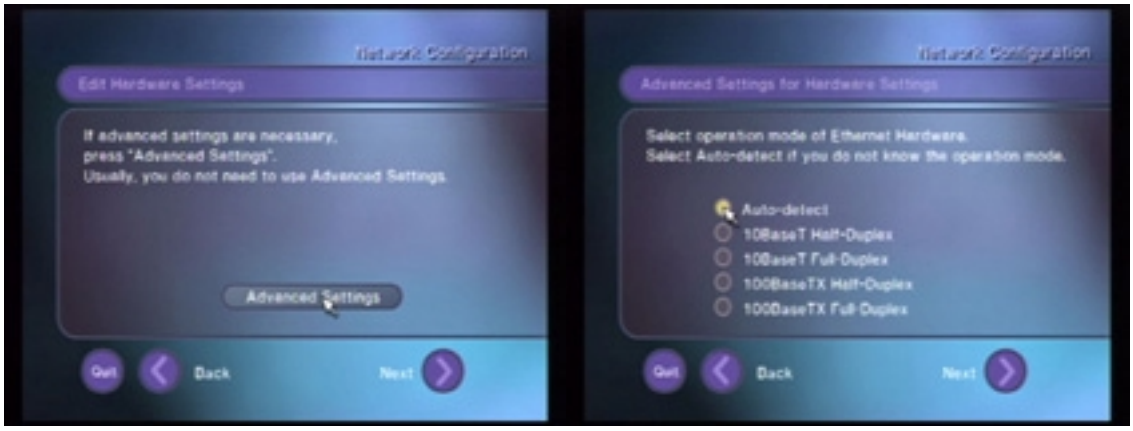
The display will change to the "Save Hardware Settings" screen. Select the destination for saving the hardware settings. After the settings are saved, the display will change back to the "Network Configuration Selection and Connection Test" screen. The hardware settings that were added will be shown selected on this screen.

Editing Hardware Settings

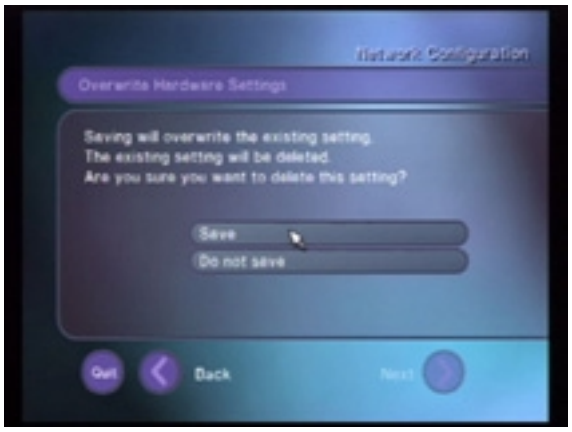
The following procedure is used to edit hardware settings. The operations for changing to the "Hardware Settings" screen are the same as those described for adding hardware settings.

After the display changes to the "Hardware Settings" screen, select "Edit Hardware Settings."

The display will change to the "Edit Hardware Settings" screen. The hardware settings that are present on the PS2 memory card and hard disk drive will be listed on this screen. Select the hardware setting that you wish to edit, then click "Next."



The display will change to the setting screen corresponding to the selected hardware. (The above figure shows the screen when USB Ethernet hardware or a network adaptor is selected.) After setting the required item(s) (the above figure shows the screen when Advanced Settings is selected and the operating mode of the Ethernet hardware is set), click "Next."



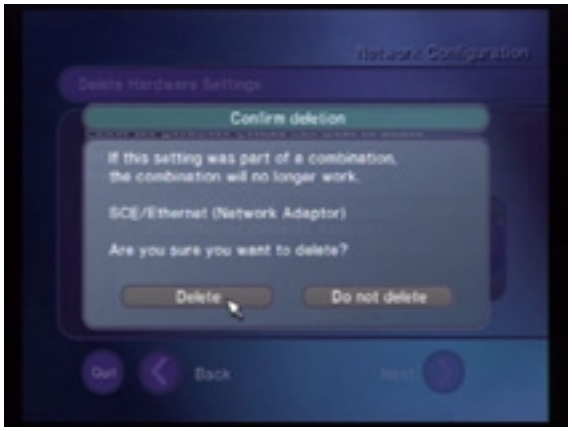
The display will change to the "Overwrite Hardware Settings" screen. The hardware settings will be saved to the device where the selected hardware settings reside. (If the connection state of the device changes during the procedure described above, the display is forcibly returned to the screen for selecting hardware settings.) After the settings are saved, the display will change back to the "Network Configuration Selection and Connection Test" screen. The hardware settings that were edited will be shown selected on this screen.

Deleting Hardware Settings

The following procedure is used to delete hardware settings. The steps for displaying the "Hardware Settings" screen are the same as those described for adding hardware settings.

After the display changes to the "Hardware Settings" screen, select "Delete Hardware Settings."

The display will change to the "Delete Hardware Settings" screen. The hardware settings present on the PS2 memory card and hard disk drive will be listed on this screen. Select the hardware settings that you wish to delete, then click "Delete."



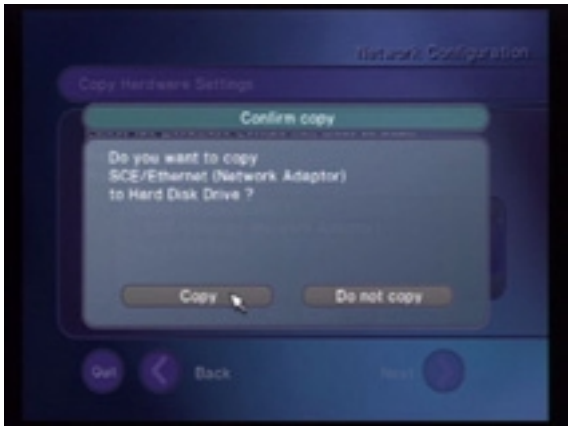
When a dialog box for confirming the deletion appears, click "Delete." After the delete has completed, the display will change back to the "Hardware Settings" screen.

Copying Hardware Settings

The following procedure is used to copy hardware settings. The steps for displaying the "Hardware Settings" screen are the same as those described for adding a hardware setting.

After the display changes to the "Hardware Settings" screen, select "Copy Hardware Settings." (This item cannot be selected unless the PS2 memory card and hard disk drive are connected. Copying can be executed only from the PS2 memory card to the hard disk drive or from the hard disk drive to the PS2 memory card.)

The display will change to the "Copy Hardware Settings" screen. The hardware settings that are present on the PS2 memory card and hard disk drive will be listed on this screen. Select the hardware settings that you wish to copy, then click "Copy."



When a dialog box for confirming the copy appears, click "Copy." After copying has completed, the display will change back to the "Hardware Settings" screen.

Adding Network Service Provider Settings

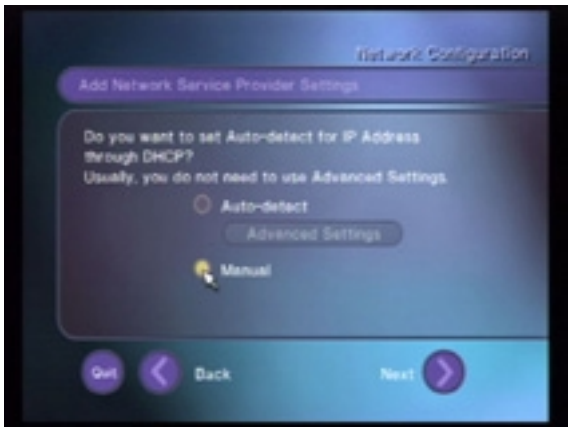
The following procedure is used to add a network service provider setting.

After starting up the "Network Configuration Selection and Connection Test" screen, click "Next." Depending on the setting of the flag member of the SceNetGuiCnf_Arg structure, this screen may be skipped and it may not be possible to click the "Next" button. For details, see "Configure startup options" in "Initialization Processing Needed Before sceNetGuiCnf_Do() is Called" of the "Library Startup and Shutdown" chapter in this document.

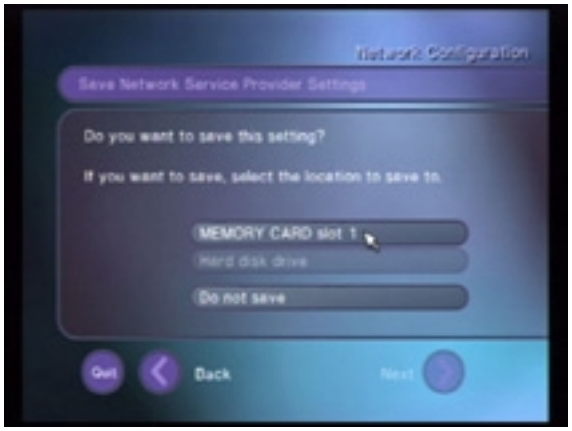
When the display changes to the "Connection Settings" screen, select "Network Service Provider Settings."

When the display changes to the "Network Service Provider Settings" screen, select "Add Network Service Provider Settings."

The display will change to the "Add Network Service Provider Settings" screen. The hardware settings that are present on the PS2 memory card and hard disk drive will be listed on this screen. (If no hardware settings have been added, processing cannot advance beyond this screen.) Select the hardware setting that will be used by the network service provider setting that you wish to add, then click "Next."



The display will change to the setting screen corresponding to the selected hardware. (The above figure shows the screen for selecting whether or not the IP address is to be automatically detected by DHCP when USB Ethernet hardware or a network adaptor is selected.) After setting the required item(s), click "Next." Items that are set can span multiple screens. The steps described above are repeated until the display changes to the "Save Network Service Provider Settings" screen.



On the "Save Network Service Provider Settings" screen, select the destination for saving the network service provider settings. After the settings are saved, the display will change back to the "Network Configuration Selection and Connection Test" screen. The network service provider setting that was added will be shown selected on the "Network Configuration Selection and Connection Test" screen.

Editing Network Service Provider Settings

The following procedure is used to edit network service provider settings. The steps for changing to the "Network Service Provider Settings" screen are the same as those described for adding a network service provider setting.

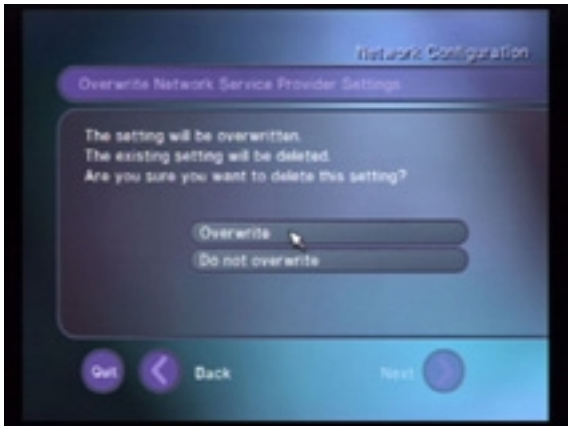
After the display changes to the "Network Service Provider Settings" screen, select "Edit Network Service Provider Settings."

The display will then change to the "Edit Network Service Provider Settings" screen. The network service provider settings that are present on the PS2 memory card and hard disk drive are listed on this screen. Select the network service provider setting that you wish to edit, then click "Next."

The display will change to the setting screen that corresponds to the selected network service provider. (The above figure shows the screen for selecting whether or not the IP address is to be automatically

detected by DHCP when USB Ethernet hardware or a network adaptor is selected, and when the selected network service provider setting is being edited.)

After setting the required item(s), click "Next." Items that are set can span multiple screens. The steps described above are repeated until the display changes to the "Overwrite Network Service Provider Settings" screen.



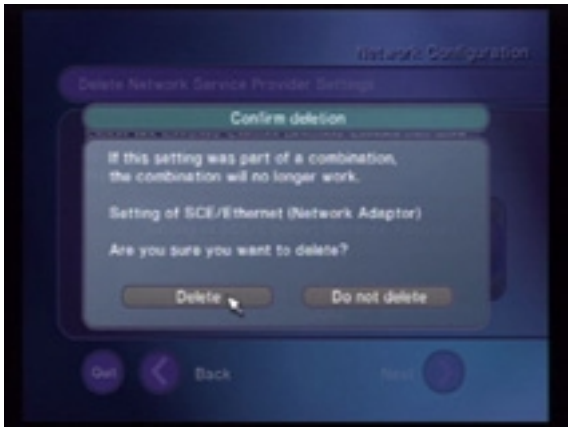
The hardware settings will be saved to the device where the selected network service provider settings reside. (If the connection state of the device changes during the procedure described above, the display is forcibly returned to the screen for selecting network service provider settings.) After the settings are saved, the display will change to the "Network Configuration Selection and Connection Test" screen. The hardware settings that were edited will be selected on this screen.

Deleting Network Service Provider Settings

The following procedure is used to delete a network service provider setting. The steps for changing to the "Network Service Provider Settings" screen are the same as those described for adding a network service provider setting.

After the display changes to the "Network Service Provider Settings" screen, select "Delete Network Service Provider Settings."

The display will change to the "Delete Network Service Provider Settings" screen. The network service provider settings that are present on the PS2 memory card and hard disk drive will be listed on this screen. Select the network service provider setting that you wish to delete, then click "Delete."



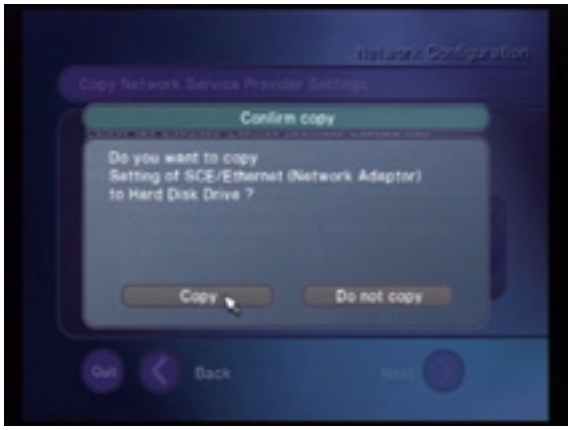
When a dialog box for confirming the deletion appears, click "Delete." After the deletion has completed, the display will change back to the "Network Service Provider Settings" screen.

Copying Network Service Provider Settings

The following procedure is used to copy a network service provider setting. The steps for displaying the "Network Service Provider Settings" screen are the same as those described for adding a network service provider setting.

After the display changes to the "Network Service Provider Settings" screen, select "Copy Network Service Provider Settings." (This item cannot be selected unless the PS2 memory card and hard disk drive are connected. Copying can be executed only from the PS2 memory card to the hard disk drive or from the hard disk drive to the PS2 memory card.)

The display will change to the "Copy Network Service Provider Settings" screen. The network service provider settings that are present on the PS2 memory card and hard disk drive will be listed on this screen. Select the network service provider setting that you want to copy, then click "Copy."



When a dialog box for confirming the copy appears, click "Copy." After copying has completed, the display will change back to the "Network Service Provider Settings" screen.

Creating New Settings

The following procedure is used to create new settings.

After starting up the "Network Configuration Selection and Connection Test" screen, click "Next." Depending on the setting of the *flag* member of the SceNetGuiCnf_Arg structure, this screen may be skipped and it may not be possible to click the "Next" button. For details, see "Configure startup options" in "Initialization Processing Needed Before sceNetGuiCnf_Do() is Called" of the "Library Startup and Shutdown" chapter in this document.

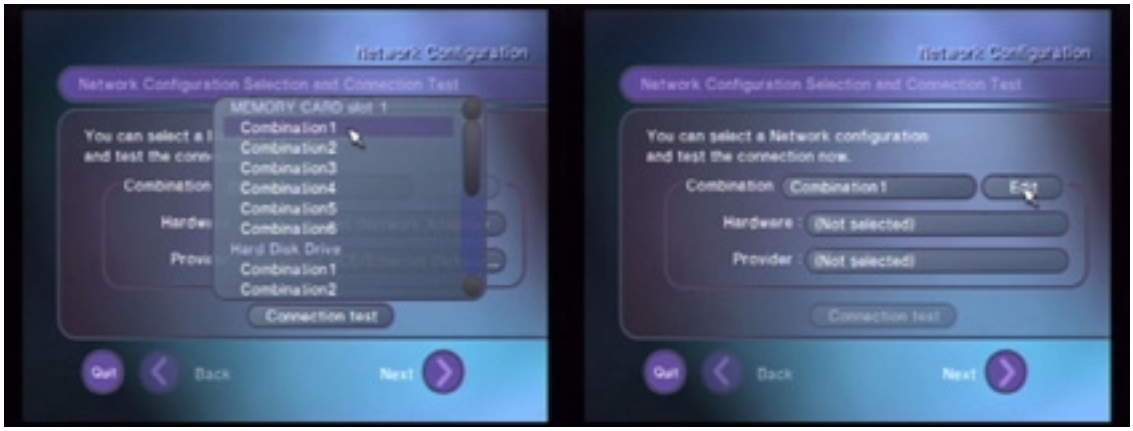
When the display changes to the "Connection Settings" screen, click "New." Selecting "New Settings" will add hardware and network service provider settings at the same time.

The hardware setting is made first, followed by the network service provider setting. Hardware settings that have been previously saved can also be used. In this case, processing shifts immediately to setting the network service provider, skipping the hardware setting. The settings are saved in the order of network service provider first, followed by the hardware setting. When a hardware setting that was previously saved is used, only the network service provider setting is saved.

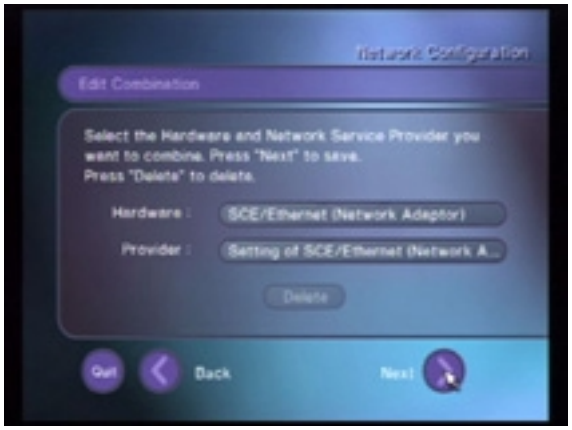
After the settings have been saved, the display will change back to the "Network Configuration Selection and Connection Test" screen. The hardware and network service provider settings that were saved will be shown selected on this screen.

Adding or Editing Combinations

The following procedure is used to add or edit a combination.



After starting up the "Network Configuration Selection and Connection Test" screen, select the combination that you wish to add or edit, then click "Edit."

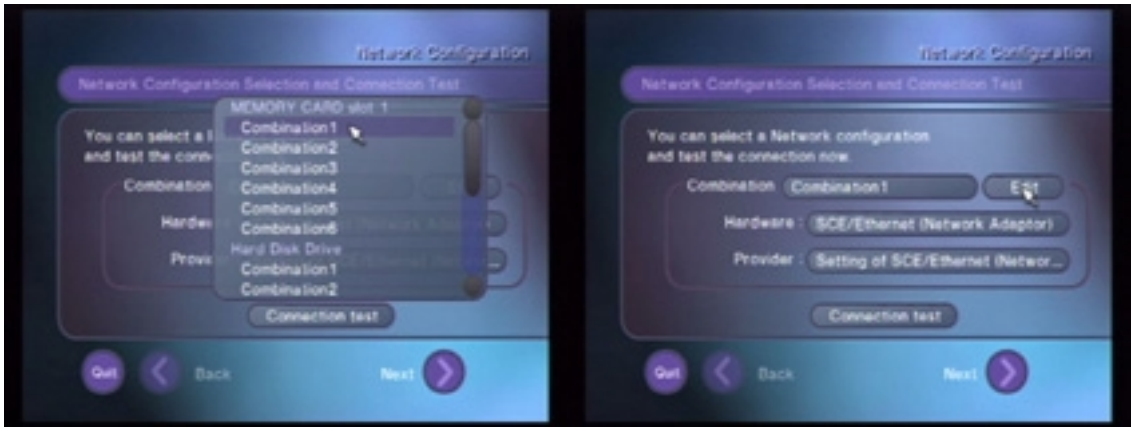


When the display changes to the "Edit Combination" screen, select the hardware and network service provider settings that you wish to combine, then click "Next." (Only hardware and network service provider settings that both reside on a PS2 memory card or that both reside on the hard disk drive can be combined. If any other combination is selected, "Next" cannot be selected.)

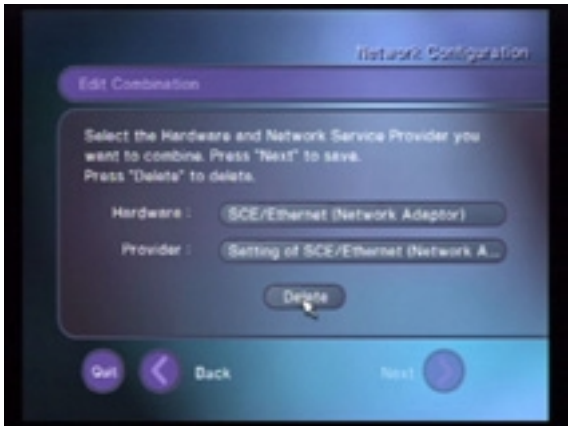
After the combination has been saved, the display will change back to the "Network Configuration Selection and Connection Test" screen. The combination that was added or edited will be shown selected on the "Network Configuration Selection and Connection Test" screen.

Deleting Combinations

The following procedure is used to delete a combination.



After starting up the "Network Configuration Selection and Connection Test" screen, select the combination that you wish to delete, then click "Edit."



When the display changes to the "Edit Combination" screen, select "Delete." "Delete" can be selected only in the following situations.

- If both the hardware and network service provider settings exist for the selected combination.
- If only the hardware settings have been deleted for the selected combination.
- If only the network service provider settings have been deleted for the selected combination.
- If both the hardware and network service provider settings have been deleted for the selected combination.

Even if the combination is deleted, the hardware and network service provider settings are not deleted.

When a dialog box for confirming the deletion appears, click "Delete." After deletion has completed, the display will change back to the "Network Configuration Selection and Connection Test" screen.

Selecting a Network Configuration

By combination

The following procedure is used to select hardware and network service provider settings with a combination.

After starting up the "Network Configuration Selection and Connection Test" screen, select "Combination."

After the combination is selected, the hardware and network service provider settings that were set in the combination will be shown selected.

By individual hardware and network service provider settings

The following procedure is used to individually select hardware and network service provider settings.

After starting up the "Network Configuration Selection and Connection Test" screen, choose "Hardware."

Next, choose "Provider." (You can also choose "Provider" first, then "Hardware".)

After these selections are made, the hardware and network service provider settings will be shown selected.

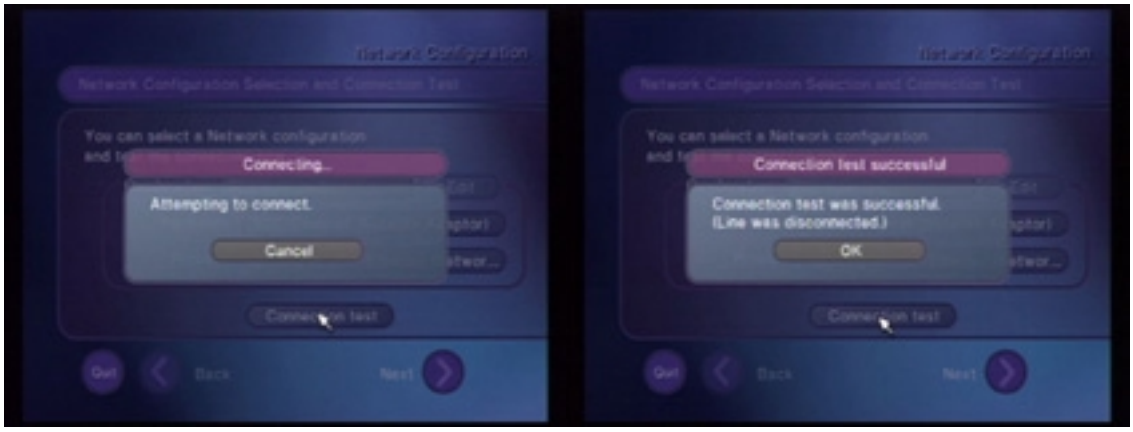
Connection Test

The following procedure is used to perform a connection test.

After starting up the "Network Configuration Selection and Connection Test" screen, if "Connection Test" is lit when hardware and network service provider settings are selected, click "Connection Test."

"Connection Test" will be lit when hardware and network service provider settings are selected, but only if the following condition is satisfied.

- The hardware that is set in the selected hardware setting is properly connected to the PlayStation 2 console.



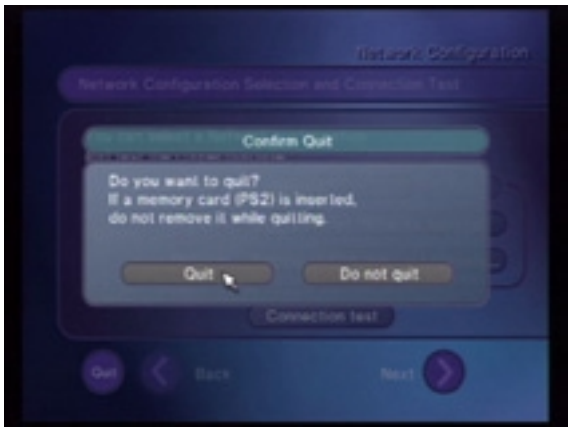
After clicking on "Connection Test," the connection test will be performed. If the test completed normally, a dialog box indicating that the test was successful will appear.

Quitting Network Configuration Processing

The following procedure is used to quit network configuration processing.

Click "Quit," which is always displayed at the lower left portion of the screen and can be selected anytime. Note that the contents of the selected hardware and network service provider settings are returned to the title application only when "Quit" is selected on the "Network Configuration Selection and Connection Test" screen.

When "Quit" is selected, sceNetGuiCnf_Do() returns the currently selected contents of the hardware and network service provider settings to a specified area. If no selections were made, sceNetGuiCnf_Do() returns the initial values (these are not the same as the default values when the settings were added, which can be specified by the title application).



When a dialog box for confirming the Quit appears, click "Quit."

Library Restrictions

TCP/IP Stack

The network configuration GUI library uses SCEI's Libinet TCP/IP stack. After sceNetGuiCnf_Do() terminates, connections are not limited to only those using the selected network configuration.

Specifying the Modem Driver

sceNetGuiCnf_Do() returns the vendor name and product name of the hardware settings as part of the selected network configuration. If you want the application to specify the modem driver, a correspondence table that associates the vendor name and product name with a modem driver is required. To obtain the vendor name and product name that corresponds to a modem driver, you must ask the company that provides the modem driver.

Graphic System

Once sceNetGuiCnf_Do() starts, it initializes the graphic system. The state of the graphic system cannot be guaranteed after sceNetGuiCnf_Do() returns. The title application must initialize or reinitialize the graphic system before and after using sceNetGuiCnf_Do().

Sound System

sceNetGuiCnf_Do() provides no support for the sound system.

IOP Modules

sceNetGuiCnf_Do() does not perform any pre-processing or post-processing for IOP modules. The title application must load the IOP modules that are required by sceNetGuiCnf_Do() before calling sceNetGuiCnf_Do() and must reset the IOP if any unnecessary IOP modules remain after sceNetGuiCnf_Do() terminates.

User Interface

Item selection method

sceNetGuiCnf_Do() uses the mouse pointer as a pointing method for item selection. It also supports only the direction keys of supported controllers for moving the mouse pointer, as well as USB mice.

Software keyboard

Software keyboard support for sceNetGuiCnf_Do() depends on the software keyboard defined by the application. This is defined with the software keyboard-related callback function of the sceNetGuiCnf_Arg structure. The mouse pointer is also used as the pointing method for key selection, in a similar fashion as with item selection.

Character Codes

sceNetGuiCnf_Do() supports the following character codes.

- Shift JIS is supported for the displayed character code.
- Shift JIS is supported for communication with the software keyboard.
- UTF-8 is supported for characters that are saved in a network configuration file.