libhttp Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Scope of This Document

This document describes libhttp, whose purpose is to support applications which use HTTP. libhttp works on top of libnet, transmitting an HTTP request to a URI specified by an application, and subsequently receiving the response to that request.

Overview

libhttp is a library that provides support for applications that use HTTP. The library can send an HTTP request to a URI specified by an application, and subsequently receive the response to that request. libhttp runs on top of libnet.

libhttp supports the following functions.

- HTTP/1.0, 1.1
- GET, HEAD, POST, PUT, DELETE, TRACE
- HTTP Proxy
- Redirect
- Keep-Alive
- GZIP encoding
- Basic/Digest authentication
- Cookie
- SSL

Files

libnet must be initialized in order to use libhttp. For the libnet usage procedure, refer to the "libnet Overview" document.

The files which are specifically required by libhttp are shown below.

Filename	Description
libhttp.h	Header file
libSceHttp_stub.a	Stub library file
libSceHttp_stub_weak.a	Weak import stub library file

libhttp can be linked only using the PRX format. To use libhttp, statically link libSceHttp_stub.a or libSceHttp_stub_weak.a. The PRX module is stored in the storage managed by the system software, and it is loaded/unloaded by the libsysmodule API.

For details regarding the PRX format, refer to the "libsysmodule Overview" document.

Sample Programs

The sample programs provided for libhttp are as follows.

- http_get/simple: HTTP_GET Simple sample
- http_get/https: HTTPS GET sample
- http_get/non_blocking: HTTP GET sample using non-blocking mode

For information on samples, refer to the sample readmes.

Reference Materials

libhttp complies with HTTP/1.1.

2 Overview of Usage Procedure

Loading Modules

The libhttp module and dependent modules are executed by the libsysmodule API. When using SSL, specify `SCE_SYSMODULE_HTTPS`, and when not using SSL, specify `SCE_SYSMODULE_HTTP`.

Initializing libnet

Prior to using libhttp, it is necessary to initialize libnet. For details regarding these procedures, refer to the "libnet Overview" document.

Initializing libssl

When using SSL, libssl must be initialized. For details regarding the procedure, refer to the "libssl Overview" document.

Initializing the Library

The related modules of libhttp are loaded by libsysmodule, and upon completion of libnet and libssl initialization, call `sceHttpInit()`. Specify the size of the memory pool to be used within libhttp as the argument.

HTTP Processing Procedures

(1) Creating Templates and Connections

First, perform the settings which are required for sending HTTP requests. With libhttp, settings for sending HTTP requests are managed in two stages: templates and connections.

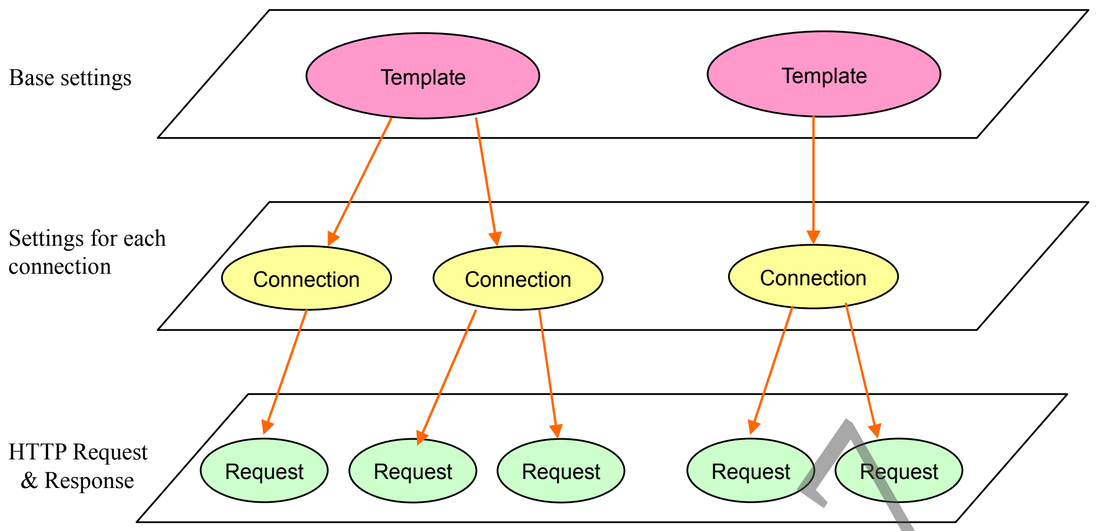
Setting	Description
Template	Template settings for creating connections. Comprises the settings which do not change from connection to connection. - HTTP version - user-agent value - Whether or not to use system settings
Connection	Per-connection settings. - Host name of the server - Connection scheme (HTTP) - Port number - Whether or not to use keep-alive

Templates and connections are created by calling `sceHttpCreateTemplate()` and `sceHttpCreateConnection()`. Unique IDs are assigned by the library to created templates and connections. Hereafter, these IDs will be referred to as the template ID and the connection ID.

(2) Creating a Request

Next, create requests which correspond to actual HTTP requests. Requests are created by calling `sceHttpCreateRequest()`. Specify a connection ID, method, path, etc. as arguments. Unique IDs are assigned by the library to created requests. Hereafter, this ID will be referred to as the request ID.

The relationship between templates, connections, and requests can be illustrated as follows.



(3) Sending an HTTP Request and Receiving the Response Header

Actual communication with a server is carried out by creating a request and then calling `sceHttpSendRequest()`. If a TCP connection with the server is already open, then the request will be sent on this same connection using keep-alive.

Since libhttp does not create internal threads, `sceHttpSendRequest()` blocks until processing is completed. Specifically, the function returns after the HTTP request is sent and the response header is received from the server.

After calling `sceHttpSendRequest()`, the request ID can be used to obtain the response data from the server. The status code (such as 200 (OK)) can be obtained using `sceHttpGetStatusCode()`, the content length of the body of the response from the server can be obtained using `sceHttpGetResponseContentLength()`, and other headers can be obtained using `sceHttpGetAllResponseHeaders()`.

(4) Receiving the Message Body

Once the response header has been received, call `sceHttpReadData()` to receive the message body from the server. The size of the message body can be obtained beforehand by calling `sceHttpGetResponseContentLength()`. After reading *`content_length`* bytes of the message body, HTTP request processing is complete.

(5) Deleting the Request

To delete the request, call `sceHttpDeleteRequest()` and specify the request ID.

To send the HTTP request again, repeat the process starting with request creation. To change the connection target, create a connection corresponding to the new target, and then create the request.

```
/*Example of the HTTP processing procedure*/
#define LIBHTTP_POOLSIZE     (20 * 1000)
#define USER_AGENT           "-sample-agent/0.1"
#define SCHEME               "http"
#define SERVER               "www.scei.co.jp"
#define PATH                 "/"
#define PORT                 (80)
#define URL                  "http://www.scei.co.jp"
int           ret, templateId, connectionId, requestId, statusCode, counter;
unsigned char buf[128];
SceULong64    contentLength;
(Network initialization processing)
```

```
...
/*Initialize libhttp*/
sceHttpInit(LIBHTTP_POOLSIZE);
/*Create template*/
templateId = sceHttpCreateTemplate(USER_AGENT, SCE_HTTP_VERSION_1_1,
        SCE_TRUE);
if (templateId < 0){
        ERROR;
}
/*Create connection*/
connectionId = sceHttpCreateConnection(templateId, SERVER, SCHEME,
        PORT, SCE_TRUE);
/* A connection can also be created using a URL string.
connection_id = sceHttpCreateConnectionWithURL(templateId, URL, SCE_TRUE);
*/
if (connectionId) < 0){
        ERROR;
}
/*Create request*/
requestId = sceHttpCreateRequest(connectionId, SCE_HTTP_METHOD_GET, PATH, 0);
/* A request can also be created using a URL string.
requestId = sceHttpCreateRequestWithURL(connectionId, SCE_HTTP_METHOD_GET,
        URL, 0);
*/
if (requestId < 0){
        ERROR;
}
/*Send request and receive response header */
ret = sceHttpSendRequest(requestId);
if (ret < 0){
        ERROR;
}
ret = sceHttpGetStatusCode(requestId, &statusCode);
if ( (ret < 0) || (statusCode != 200) ){
        ERROR;
}
ret = sceHttpGetResponseContentLength(requestId, &contentLength);
if (ret < 0){
        ERROR;
}
/*Receive message body*/
while (contentLength > 0){
        ret = sceHttpReadData(requestId, buf, sizeof(buf));
        if (ret < 0){
             ERROR;
        } else if (ret == 0){
             break; /*The connection was closed*/
        } else {
             for (counter = 0; counter < ret; counter++){
                    putchar(buf[counter]);
             }
        }
}
printf("\n");
/*Delete request*/
sceHttpDeleteRequest(request_id);
/*Library termination processing*/
sceHttpDeleteConnection(connection_id);
sceHttpDeleteTemplate(template_id);
sceHttpTerm();
(Network termination processing)
```

Library Termination Processing

Prior to terminating libhttp, applications must delete templates, connections, and requests using `sceHttpDeleteTemplate()`, `sceHttpDeleteConnection()`, and `sceHttpDeleteRequest()`. First delete requests, then delete connections, and finally delete templates, in that order.

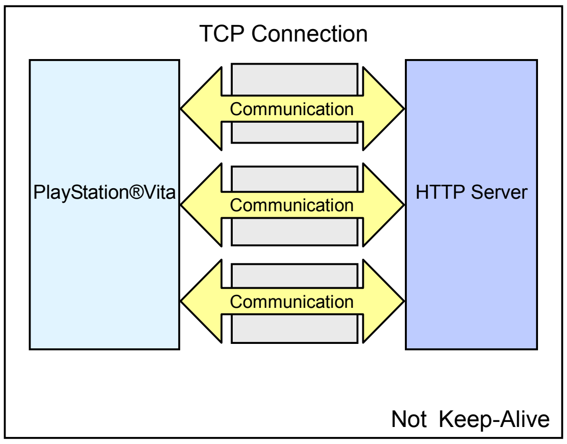
After these deletions have been completed, call `sceHttpTerm()`.

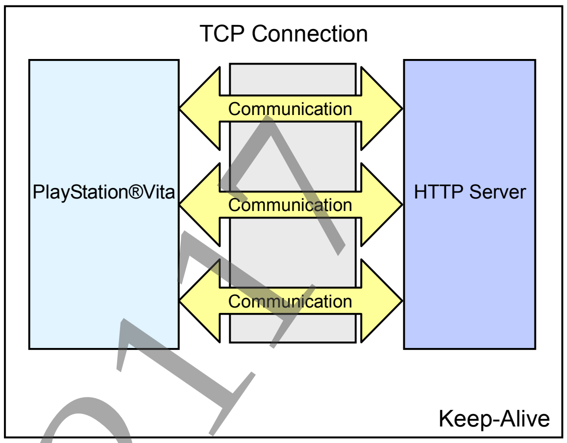
```
/* Delete all previously created templates, connections, and requests */

/* Termination processing for libhttp */
sceHttpTerm();
```

3 Description of Operation

Keep-Alive Handling Mechanism

In the interest of improving communication efficiency, starting with HTTP version 1.1, a feature called Keep-Alive was adopted which performs multiple HTTP exchanges without closing the TCP connection.





libhttp has the capability to automatically perform Keep-Alive. libhttp will carry out communication using Keep-Alive when all of the following conditions are met.

- The Keep-Alive feature is enabled in the connection settings.
- The connected server supports Keep-Alive.
- A request object is being used that uses the same connection settings.

Keep-Alive can be enabled or disabled for each individual set of connection settings. In addition, the information as to whether or not the connection to the current server is being kept alive is managed within the connection settings, so when the `sceHttpSendRequest()` function is called, the connectivity with the server is checked, and a connection is created only when a connection does not already exist.

When Keep-Alive is used, multiple requests and responses can be exchanged over a single TCP connection. Furthermore, a server will send responses in the same order as it receives requests. Please be aware that, for this reason, deadlock can occur if software is structured such that an application sends multiple requests using the same connection settings, and a receiving thread for a prior request ends up waiting to receive a response to a subsequent request.

Redirection Handling Mechanism

In HTTP, in order to notify a client that a URL has been changed, a type of response called a redirection can be returned from the server. libhttp has the capability to automatically send a new request to the specified URL when a redirection response is received from a server, Specifically, when a response status of 300, 301, 302, 303 or 307 is returned from a server, a request is again sent to the specified URL. This functionality can be enabled or disabled using the `sceHttpSetAutoRedirect()`, `sceHttpEnableRedirect()` and `sceHttpDisableRedirect()` functions. It is enabled by default. When this functionality has been disabled, the application must analyze the response from the server, and then explicitly create and send a request to the redirection target.

How storage of the redirection target is handled differs depending on the response status of the server.

- For 300, 302, 303 and 307, the redirection target is not stored.
- For 301, the redirection target is stored, and when there are subsequent accesses to the same URL, they are automatically converted within the library into requests to the redirection target.

Basic / Digest Authentication

libhttp supports basic authentication and digest authentication. By specifying the callback function beforehand to obtain a user and password in `sceHttpSetAuthInfoCallback()` function, the callback function is called to set a user and the password when these authentications are required. In addition to the callback function, a user name and password can be included in the URL specified in `sceHttpCreateConnectionWithURL()` and `sceHttpCreateRequestWithURL()`. Use the `sceHttpSetAuthEnabled()`, `sceHttpEnableAuth()` and `sceHttpDisableAuth()` functions to enable and disable authentication.

The username and password passed by the callback function are stored within the library and reused when necessary.

Cookie Processing Mechanism

libhttp has a function to automatically save and send cookies. Use `sceHttpSetCookieEnabled()`, `sceHttpEnableCookie()` and `sceHttpDisableCookie()` functions to enable and disable this function. It is enabled by default. Since cookie data is placed in memory, cookies are enabled until `sceHttpTerm()` is called. Use `sceHttpCookieExport()` and `sceHttpCookieImport()` to save cookies beyond their lifetime in libhttp. To set the sizes and number of cookies, use `sceHttpSetCookieTotalMaxSize()`, `sceHttpSetCookieMaxSize()`, `sceHttpSetCookieMaxNum()`, and `sceHttpSetCookieMaxNumPerDomain()`.

Server Authentication

A function to verify server certificates during SSL communication is provided. The verify policy can be set with the `sceHttpsEnableOption()` and `sceHttpsDisableOption()` functions. In addition, the callback function can be made to be called when the server certificate is verified by specifying the callback function beforehand with the `sceHttpsSetSslCallback()` function. Using this callback function, the certificate chain can be verified and whether SSL communication is enabled or disabled can be specified to libhttp.

If verification against a certificate other than the RootCA certificate already installed on PlayStation®Vita is required, it can be added with the `sceHttpsLoadCert()` function.

HTTP Proxy

libhttp supports communication via the HTTP proxy server. The proxy settings are automatically loaded from the network connection settings. It is possible to connect directly by disabling the loading operation with an argument that is set at the time of creation of template settings. However, it is recommended to load the proxy settings from the network connection settings because, the network environment of PlayStation®Vita to be connected may change frequently, and there will be a high possibility of changes on whether or not to use the proxy server or on which proxy server will be used.

GZIP Encoding

In HTTP version 1.1, a feature to compress and transfer content is provided, and libhttp supports the GZIP-compressed responses. When receiving a response header indicating that the response is GZIP-compressed, libhttp automatically decompresses the body of the response and then returns the response through the `sceHttpReadData()` function; however, the request header (Accept-Encoding: gzip) that indicates the acceptance of GZIP compression is not sent out automatically. If you wish to actively receive the GZIP-compressed response, add the request header with the `sceHttpAddRequestHeader()` function.

4 Precautions

Restrictions in Multithreaded Environments

Note that a connection ID can be assigned to only one TCP socket, excluding special cases such as redirection. For that reason, calling functions that invoke communication, such as `sceHttpReadData()` and `sceHttpSendRequest()`, simultaneously from multiple threads will return an error. To avoid creating a situation such as this, you must create only one connection for each thread. As long as you don't call these functions at the same time from multiple threads, or if the requests created by different threads use different connection IDs, you won't have a problem. You can also call `sceHttpAbortRequest()` at the same time as a function that invokes communication.

Redirection by the Network Environment

PlayStation®Vita is frequently connected to a network environment with a special configuration, including when a user connects to a network using wireless LAN service provided in public establishments. In this type of network environment, the service provider often redirects the user to their own URL to promote their services or to perform user authentication. Thus, when automatic redirection is enabled, a request may complete after obtaining content of a URL that was not the originally intended URL. Moreover, Note that when a redirect to https occurs before libssl is loaded/initialized, it can cause the application to crash.

To prevent obtaining unintended content while the automatic redirection feature is enabled, use the callback function registered with `SceHttpRedirectCallback()` to check the URL specified as the redirect destination and prevent redirection to an unknown URL by returning a negative value.

Library Compatibility

libhttp is based on libhttp provided with the PSP™ (PlayStation®Portable) SDK, and basically provides the same APIs. An API whose name has changed can call the original API with <libhttp/libhttp_compat.h>. Note, however, that for the constant macro `SCE_HTTP_INVALID_ID` only, the use has changed from error codes to values of context IDs that do not exist, so care is required regarding this point when porting an application.

PSP™ (PlayStation®Portable)

Macro	Value	Description
`SCE_HTTP_INVALID_ID`	0x80431100	The specified ID does not exist

This library

Macro	Value	Description
`SCE_HTTP_INVALID_ID`	0	Default constant macro for specifying non-existent ID
`SCE_HTTP_ERROR_INVALID_ID`	0x80431100	Error code indicating that the specified ID does not exist