NpWebApi Library Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Document serial number: 000004892117

Table of Contents

1 Library Overview

Purpose and Characteristics

The NpWebApi library is a library required for accessing PSN℠ Web APIs ("Web APIs" hereafter) from PlayStation®Vita. Normally, procedures such as obtaining the token and determining the base URL are required for executing Web APIs, but since many of these processes are internally performed with this library, applications can easily execute Web APIs by using this library. In addition, the NpWebApi library provides an interface for receiving Push events.

Main Features

The main features provided by the NpWebApi library are as follows.

- Feature for accessing arbitrary Web APIs
- Feature for receiving Push events
- Utility feature for converting data in Web API protocols to data used with the (non-Web API) NP library

Embedding into a Program

Include np.h in the source program.

Load also the PRX module in the program as follows.

```
if ( sceSysmoduleLoadModule(SCE_SYSMODULE_NP_WEBAPI) != SCE_OK ) {
    // Error handling
}
```

Upon building the program, link libSceNpWebApi_stub.a or libSceNpWebApi_stub_weak.a.

Sample Program

A sample program that uses the NpWebApi library is provided as follows.

sample_code/network/api_np/np_webapi/

This sample shows the basic usage of the NpWebApi library.

Reference Materials

Refer to the following document for an overview of the PSN℠ features.

- PSN℠ Overview

Refer to the following documents regarding the NP library, which is commonly required when using the PSN℠ features.

- NP Library Overview
- NP Library Reference

Refer to the following document regarding Network Check Dialog for switching service states of the NP library.

- Network Overview

For an overall explanation of the Web APIs and details on each Web API, refer to the following documents.

- PSN℠ Web APIs Overview

2 Using the Library

Preparation

nptitle.dat File Obtaining and Placement

An NP Title ID, NP Title Secret, and nptitle.dat file will be issued for each application by making a request on the PlayStation®Vita Developer Network (https://psvita.scedev.net/).

When **Release Check Mode** of ★**Debug Settings** is **Development Mode**, `sceNpWebApiSetNpTitleId()` can be called in the application to set the issued NP Title ID and NP Title Secret.

When **Release Check Mode** of ★**Debug Settings** is **Release Mode**, appropriately place the nptitle.dat file according to the application boot method. In **Release Mode**, the `sceNpWebApiSetNpTitleId()` settings will be ignored and the NP Title ID/NP Title Secret stored in the nptitle.dat file will be used.

- Booting from an application package or APP_HOME
  Place nptitle.dat in the same directory as the application system file (param.sfo). For application package creation, booting from APP_HOME, and system file placement directories, refer to the "Application Development Process Overview" document.
- Booting from the debugger
  Create an "sce_sys" directory below the Visual Studio working directory, and place nptitle.dat below it.

Note
The NP Title Secret and nptitle.dat file are secret information for verifying an NP Title ID owner. Take sufficient precautions so that they are not leaked to third parties. For example, ensure that these are removed when disclosing the source code/application package for support purposes on the PlayStation®Vita Developer Network.

Basic Procedure

This section explains overall processing in the following four sections.

- Initialization/Pre-processing
- Web API Execution
- Push Event Receiving
- Termination/Post-processing

Initialization/Pre-processing

(1) Initialize libhttp

Initialize libhttp. For details, refer to the "libhttp Overview" and "libhttp Reference" documents.

(2) Initialize the NpWebApi library

Call `sceNpWebApiInitialize()` to initialize the NpWebApi library. Specify the memory pool size to be used by the NpWebApi library.

Note
The memory size required varies depending on the application implementation and Web API execution frequency.

```
#define NP_WEBAPI_POOL_SIZE ( 64 * 1024 )
```

```
int32_t ret = 0;

ret = sceNpWebApiInitialize(NP_WEBAPI_POOL_SIZE);
if (ret < 0) {
    // Error handling
}
```

Web API Execution

(3) Prepare request data

When sending request data to the server upon Web API execution, store the request data in memory in advance. In addition, the content parameters represented with the `SceNpWebApiContentParameter` structure must be prepared.

The content parameters are comprised of Content-Length which represents the request data size and Content-Type which represents the request data type. These are set in the NpWebApi library as the Content-Length and Content-Type values in the HTTP request header.

For example, in the current Web APIs, request data is defined in JSON format with UTF-8 encoded character strings, therefore Content-Type is specified as:

```
"application/json; charset=utf-8"
```

In the NpWebApi library, this character string is defined as the macro `SCE_NP_WEBAPI_CONTENT_TYPE_APPLICATION_JSON_UTF8`, therefore applications can use this macro as the Content-Type value.

(4) Create a request

To execute Web APIs, a request must be created and sent every time a Web API is executed. First, specify the following parameters for creating a request then execute `sceNpWebApiCreateRequest()`. When the request has been properly created, a request ID will be issued.

- API group
  The Web APIs are divided into a number of API groups according to their features. Specify the character string that represents the API group where the Web API you want to execute belongs to. For details on API groups, refer to the "PSN℠ Web APIs Overview" document and the reference documents for each API group.
- Path
  Specify the path of the Web API you want to execute. The path is a character string that links the resource path and query string.
- HTTP method
  Specify the constant that represents the HTTP method when executing the Web API.
- Content parameters
  If sending request data to the server when executing the Web API, specify the content parameters prepared in the previous step.

```
// Character string that represents the API group of the User Profile Web APIs
#define API_GROUP "userProfile"

// Path of the Web API to execute (resource path and query string)
#define PATH "/v1/users/user000/friendList?friendStatus=friend"

int32_t ret = 0;
int64_t requestId = 0;

// Create request
ret = sceNpWebApiCreateRequest(
    API_GROUP,
```

```
    PATH,
    SCE_NP_WEBAPI_HTTP_METHOD_GET,
    NULL // Specify NULL if not sending request data
    &requestId
    );
if (ret < 0) {
    // Error handling
}
```

(5) Send the request and obtain the error code

Execute `sceNpWebApiSendRequest2()` to send the created request to the server. If sending request data to the server, specify the buffer and size of the data to send. If the total size of the request data to send is large, the request data can be partitioned then sent by calling this function multiple times.

Note
`sceNpWebApiSendRequest2()` is a blocking function. For the timing when this function returns, refer to the "NpWebApi Library Reference" document.

```
int64_t requestId; // Request ID of the request to send

int32_t ret = 0;

ret = sceNpWebApiSendRequest2(
    requestId,
    NULL,  // Specify NULL if not sending request data
    0,     // Specify 0 if not sending request data
    NULL   // Specify NULL if not obtaining response information upon server error
    );
if (ret < 0) {
    // Error handling
}
```

When sending a request and a server error has occurred as a result of executing the Web APIs, `sceNpWebApiSendRequest2()` will internally receive the response data, an error code that represents the server error will be generated from the error object included in the response data, and will return with the error code as the return value. The upper 8 bits of this error code are 0x82 and the lower 24 bits are the server error code represented by a decimal integer.

Note
When a pointer to an `SceNpWebApiResponseInformationOption` structure is specified for the 4th argument in `sceNpWebApiSendRequest2()`, it will be possible to obtain the response data upon server error. Use this for when investigating the details of a server error during development, etc.

(6) Receive response data

When the request sending is complete and `sceNpWebApiSendRequest2()` is successful, it will be possible to use `sceNpWebApiReadData()` to receive the response data.

Note
`sceNpWebApiReadData()` is a blocking function. For the timing when this function returns, refer to the "NpWebApi Library Reference" document.

If the request succeeds and the executed Web API is of the kind that returns response data, read the response data and process it.

```
int64_t requestId; // Request ID of the sent request

int32_t ret = 0;

char buf[BUFFER_SIZE];
```

```
ret = sceNpWebApiReadData(requestId, buf, sizeof(buf)-1);
if (ret < 0) {
    // Error handling
}
```

For details on HTTP status codes, response data, and server error codes regarding Web APIs, refer to the "PSN℠ Web APIs Overview" document and the reference documents for each API group.

(7) Delete request

When the request sending/receiving is complete, execute `sceNpWebApiDeleteRequest()` to delete the request.

```
int64_t requestId; // Request ID of request to delete

int32_t ret = 0;

if (requestId > 0) {
    ret = sceNpWebApiDeleteRequest(requestId);
    if (ret < 0) {
        // Error handling
    }
}
```

Push Event Receiving

There are Push events that require an NP service name to be specified for receiving, and those that do not require an NP service name to be specified. Push events that do not require an NP service name to be specified are notified regardless of the contexts held by games. Push events that do require an NP service name to be specified are notified only in contexts held by games.

In addition, some Push events have extended data attached in addition to standard data.

All Push events can be received by using APIs and callback functions for receiving extended Push events, regardless of the absence/presence of a specified NP service name or absence/presence of extended data. Here, the usage of APIs and callback functions for receiving extended Push events is explained.

Note
In versions of the NpWebApi library earlier than SDK 3.50, APIs for receiving Push events that required an NP service name to be specified and did not require an NP service name to be specified were provided separately (Push events with extended data attached were not supported). Therefore, Push events that required an NP service name to be specified were classified as "service Push events" and Push events that did not require an NP service name to be specified were classified as "Push events".
Currently, such classification is no longer required, but these names remain in the API categories.

(8) Create a Push event filter

Execute `sceNpWebApiCreateExtdPushEventFilter()` to create a Push event filter to indicate the extended data and data types of the Push events to receive. Note that this function is a blocking function in accordance with network access. When a Push event filter is created, a filter ID will be issued. Specify the filter ID when registering Push event callback functions.

A Push event filter must be created for each NP service name.

To receive Push events that do not require an NP service name to be specified, separately create a Push event filter without an NP service name. To receive Push events that require an NP service name to be specified, create a Push event filter with an NP service name and NP service label specified in order to resolve the contexts held by the game.

Specify an NP service label in order to separately use multiple contexts with the same NP service. When not explicitly specifying an NP service label during a request for NP service usage, specify the `SCE_NP_DEFAULT_SERVICE_LABEL` macro that indicates the default NP service label.

In order to obtain extended data when receiving Push events that include extended data, specify the extended data key that represents the extended data to obtain when creating the Push event filter.

For details on the strings that represent the NP service name/data type and extended data key, refer to each Web API reference document.

The following is an example of code for creating a Push event filter for Push events that do not require an NP service name to be specified.

```
int32_t ret = 0;
int32_t handleId = 0;
int32_t filterId = 0;

SceNpWebApiExtdPushEventFilterParameter filterParam[2];
SceNpWebApiPushEventDataType dataType[2];
SceNpWebApiExtdPushEventExtdDataKey extdDataKey;

// Data types to receive
memset(dataType, 0, sizeof(dataType));
snprintf(dataType[0].val, SCE_NP_WEBAPI_PUSH_EVENT_DATA_TYPE_LEN_MAX,
        "np:service:presence:onlineStatus");
snprintf(dataType[1].val, SCE_NP_WEBAPI_PUSH_EVENT_DATA_TYPE_LEN_MAX,
        "np:service:friendlist:friend");

// Extended data key to receive
memset(extdDataKey, 0, sizeof(extdDataKey));
snprintf(extdDataKey.val,
        SCE_NP_WEBAPI_EXTD_PUSH_EVENT_EXTD_DATA_KEY_LEN_MAX, "trigger");

// Push event filter parameters
memset(filterParam, 0, sizeof(filterParam));
memcpy(&filterParam[0].dataType, &dataType[0],
        sizeof(SceNpWebApiPushEventDataType));
filterParam[0].pExtdDataKey = NULL;
filterParam[0].extdDataKeyNum = 0;
memcpy(&filterParam[1].dataType, &dataType[1],
        sizeof(SceNpWebApiPushEventDataType));
filterParam[1].pExtdDataKey = &extdDataKey;
filterParam[1].extdDataKeyNum = 1;

// Create handle
ret = sceNpWebApiCreateHandle();
if(ret < 0){
        // Error handling
}
handleId = ret;

// Create Push event filter. Note that this is a blocking function.
ret = sceNpWebApiCreateExtdPushEventFilter(
        handleId,
        SCE_NP_WEBAPI_NP_SERVICE_NAME_NONE, // Indicates that there is no NP
service name
        SCE_NP_INVALID_SERVICE_LABEL,      // Indicates that there is no NP
service label
        filterParam, 2);
if(ret < 0){
        // Error handling
}
filterId = ret;
```

```
// Delete handle
ret = sceNpWebApiDeleteHandle(handleId);
if(ret < 0){
        // Error handling
}
```

If creating a Push event filter for Push events that require an NP service name to be specified, the part where `sceNpWebApiCreateExtdPushEventFilter()` is called will be as follows.

```
#define NP_SERVICE_NAME "inGamePresence"

// Create Push event filter. Note that this is a blocking function.
ret = sceNpWebApiCreateExtdPushEventFilter(
        handleId,
        NP_SERVICE_NAME,            // NP service name
        SCE_NP_DEFAULT_SERVICE_LABEL,  // Represents the default NP service
label
        filterParam, 2);
if(ret < 0){
        // Error handling
}
filterId = ret;
```

(9) Register Push event callback functions

Execute `sceNpWebApiRegisterExtdPushEventCallback()` to register a callback function to receive Push events. Specify the Push event filter created in advance as arguments. This will cause the registered Push event callback function to be called when a Push event that matches the NP service name and data type indicated by the Push event filter is received. In addition, extended data will be notified with a Push event callback function argument if extended data that matches the extended data key specified in the Push event filter is included in the Push event.

When a callback function is registered, a callback ID will be issued.

In order to check if a registered Push event callback function is in a state where it should be called or not, have the application call `sceNpWebApiCheckCallback()` at regular intervals.

```
void
extdPushEventCallbackFunc(
        int32_t callbackId,
        const char *pNpServiceName,
        SceNpServiceLabel npServiceLabel,
        const SceNpPeerAddress *pTo,
        const SceNpPeerAddress *pFrom,
        const SceNpWebApiPushEventDataType *pDataType,
        const char *pData,
        size_t dataLen,
        const SceNpWebApiExtdPushEventExtdData *pExtdData,
        size_t extdDataNum,
        void *pUserArg
        )
{
        // If Push events do not require NP service name to be specified
        // pNpServiceName is NULL
        // npServiceLabel is SCE_NP_INVALID_SERVICE_LABEL
}

int32_t ret = 0;
int32_t filterId;
int32_t callbackId = 0;

// Register Push event callback function
```

```
ret = sceNpWebApiRegisterExtdPushEventCallback(
        filterId, extdPushEventCallbackFunc, NULL);
if(ret < 0){
        // Error handling
}

callbackId = ret;
```

Note
Note that the function for checking the Push event callback is different from `sceNpCheckCallback()` for PlayStaion®4.

Termination/Post-processing

(10) Terminate the library

When the NpWebApi library is no longer needed, call `sceNpWebApiTerminate()` to perform termination processing.

```
int32_t ret = 0;

ret = sceNpWebApiTerminate();
if (ret < 0) {
   // Error handling
}
```

API Summary

The APIs used in basic processing for the NpWebApi library are shown below.

Table 1 APIs Used in Basic Processing

API	Description
`sceNpWebApiInitialize()`	Function that initializes the NpWebApi library
`SceNpWebApiContentParameter`	Content parameter structure for the request data
`sceNpWebApiCreateRequest()`	Function that creates a request
`sceNpWebApiSendRequest2()`	Function that sends a request and executes a Web API
`sceNpWebApiReadData()`	Function that receives a response body
`sceNpWebApiDeleteRequest()`	Function that deletes a request
`sceNpWebApiCreateExtdPushEventFilter()`	Function that creates a Push event filter
`sceNpWebApiRegisterExtdPushEventCallback()`	Function that registers a Push event callback function
`sceNpWebApiTerminate()`	Function that terminates the NpWebApi library

3 Notes

Library Compatibility

The NpWebApi library is based on the NpWebApi library provided in the SDK of PlayStation®4 and provides similar APIs. However, the following differences exist and must be noted upon porting an application.

	PlayStation®4 SDK	PlayStation®Vita SDK
Library context	Handled	Not handled
User context	Handled	Not handled (since PlayStation®Vita is essentially a single-user system)
Check callback function for Push events and service Push events	`sceNpCheckCallback()`	`sceNpWebApiCheckCallback()`

Note that library contexts and user contexts are not handled with APIs, but they exist as internal structures in the library. Therefore, when an invalid request ID is assigned to some APIs, errors such as "library context could not be found" or "user context could not be found" may return.

Connection Handling by libhttp and Multi-thread Processing

The NpWebApi library implicitly creates and manages Connection of libhttp within the library. A single Connection is managed per API group.

Because of this, when `sceNpWebApiSendRequest2()` and `sceNpWebApiReadData()` - both functions use a function generating libhttp communication - are called from multiple threads for one API group, an error may occur due to the simultaneous use of libhttp Connection.

Always carry out the calls of `sceNpWebApiSendRequest2()` and `sceNpWebApiReadData()` for a single API group using one thread.

Online State and Push Events

The NpWebApi library receives Push events when the service state (`SceNpServiceState`) of the NP library obtained by the application is the online state. Even when the service state of the NP library obtained by the application is not the online state, Push events can be received if another application or the system software is in the online state. However, in order to use the Push event, make sure to call Network Check Dialog from the application to receive the Push event and transition to the online state so that the Push event can be received without being dependent on another process.

Comparison Targets With the Times When Access Restrictions Are Released

It is possible to adjust the access rates from applications by referencing the X-RateLimit-Next-Available header value (the scheduled time for access restrictions to be released in response to the rate restriction for Web API server access being exceeded). In such cases, use the network time (that can be obtained using `sceRtcGetCurrentNetworkTick()`) as the time comparison target.