NP TUS Library Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

1 Library Overview

Purpose and Characteristics

The NP Title User Storage library (NP TUS library) is a library for using the storage provided per title and per user on the server of PSN℠.

When using this title user storage service, 64 64-bit integers (TUS variables) and binary data (TUS data) of up to 1 MiB per user, and 256 64-bit integers and binary data of up to 2 MiB per virtual user, can be stored on the server of PSN℠. (There can be a maximum of 8 virtual users per title as described later in this document.) Moreover, settings can be made for each data, to either enable or disable read/write operation by other users onto that area.

In addition to simple read and write operations, atomic additions and conditional writes can be performed on a TUS variable. Moreover, it is possible to operate multiple TUS variables of a single user at the same time, and also to operate the same TUS variable of multiple users at the same time.

Because the design of the title user storage is such that updates and references are made on the same database, there is no delay in gathering and analyzing information. However, there are no ranking features provided by this library, and its use must be coordinated with the ranking service as necessary. There are also limitations in enhancing server performance given this library's design. To distribute a fixed binary data, for example, use the patch system or the title small storage service.

Embedding into a Program

Include np.h in the source program. Various header files will be automatically included as well.

Load also the PRX module in the program as follows.

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

Upon building the program, link libSceNpTus_stub.a.

Sample Program

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

sample_code/network/api_np/console/

This sample exemplifies the basic usage of the NP ScoreRanking, TUS, and TSS libraries.

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

2 Using the Title User Storage

This chapter explains the elements making up the title user storage service, as seen from the application side, and examples for their usage.

Storage (TUS Variables/TUS Data)

The title user storage service provides two types of storage – the TUS variable and the TUS data. The TUS variable stores a signed 64-bit integer. The TUS data stores binary data.

For a single user, up to 64 TUS variables and 16 TUS data can be prepared. However, the size of a single TUS data must be 512 KiB or less and the total size of all the TUS data for a single user must be 1 MiB or less. It is necessary to have the number of TUS variables and the size and number of TUS data set in advance on the server.

The TUS variable and TUS data may be referred to as slots. Each slot can be identified by a slot ID.

The TUS variable is accompanied by information regarding the status of the storage area. Status information includes information indicating whether a variable has been set to the storage area, who the owner of this storage is, the person who last updated this variable, and the last update time.

In addition to the above status information, the TUS data is also accompanied by accessory information that can be used as index information.

Virtual User

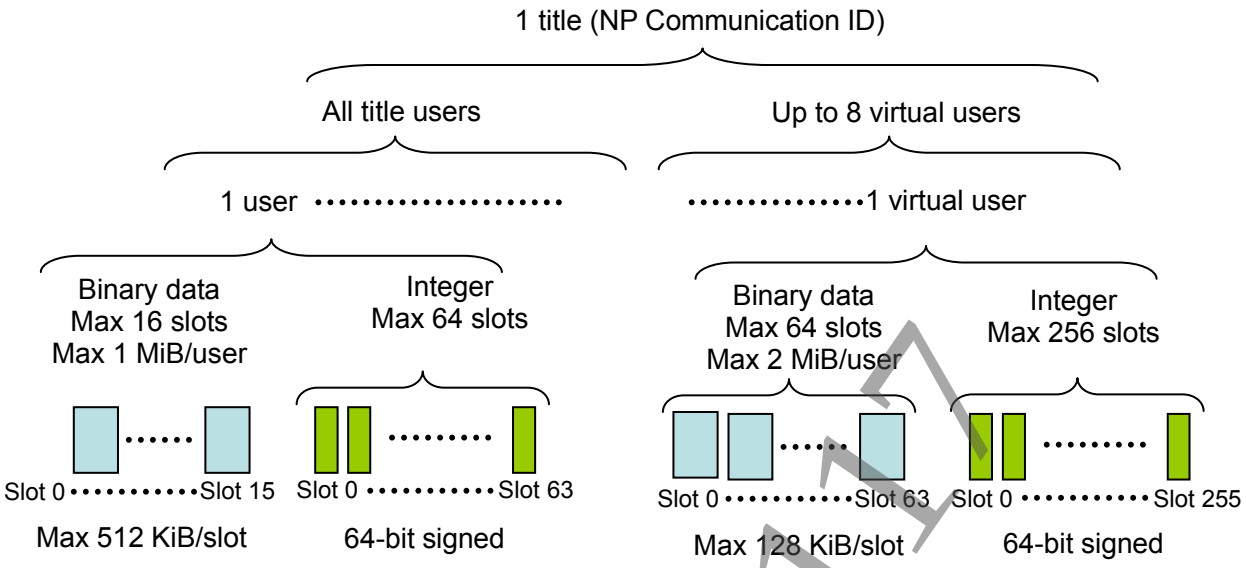
The virtual user is used to realize a common storage for all users. By using a TUS variable of the virtual user, for example, the total number of times that a certain game stage has been cleared by users all around the world can be counted.

A virtual user must be set in advance on the server. For 1 title (more specifically, for 1 NP Communication ID), up to 8 virtual users can be set. For each virtual user, up to 256 TUS variables and up to 64 TUS data can be prepared. Each TUS data must be 128KiB or less and the total size of all the TUS data for a single virtual user must be 2MiB or less.

Note
It is not possible to prevent load burdens from being concentrated on the server regarding virtual user accesses. This is why the size of each TUS data is limited. Make sure that the virtual user is not frequently accessed from the application side unless there is a good reason to do so. If you want to distribute data to a user, first consider using the patch system or the title small storage service.

Note
If you require a larger storage given the content of your title, please contact SCE.

Figure 1 Storage of an Actual User and of the Virtual User



Storage Allocation Timing

Storage for an actual user will be allocated for the first time when the user communicates with the title user storage server with the applicable NP Communication ID. A user who has never communicated with the title user storage server will not have a storage space allocated on the server, and will be unable to make access. Thus, it is not possible for the storage to be used in a case, for example, where a friend of the user writes to the user's storage in advance so that the user can use this information when the user someday plays this game.

The storage for a virtual user is allocated when a virtual user is set on the server.

APIs

The operations that can be performed on TUS variables and TUS data using the APIs of the NP TUS library are as follows.

Write to a TUS variable (multiple slots)

(Input) target user, slot IDs, 64-bit integers

Read from a TUS variable (multiple slots or multiple users)

(Input) target user(s), slot ID(s)

(Output) last update time, last updated by, 64-bit integer

Read from a TUS variable (top 100 users specified with the sort conditions from among friends)

(Input) sort conditions (ascending/descending TUS variable numerical values, ascending/descending update date/time), slot ID(s)

(Output) last update time, last updated by, 64-bit integer

Add to a TUS variable (possible to subtract by specifying a negative value)

(Input) target user, slot ID, 64-bit integer

(Output) last update time, last updated by, 64-bit integer

Compare TUS variable to current value and perform write if the specified condition is met

(Input) target user, slot ID, 64-bit integer, operator

(Output) target user, slot ID, 64-bit integer (before), 64-bit integer (after)

Delete a TUS variable (multiple slots)

(Input) target user, slot IDs

Upload TUS data (can be in installments)

(Input) target user, slot ID, binary data, maximum 384 bytes of accessory information (optional)

Download TUS data (can be in installments)

(Input) target user, slot ID

(Output) binary data, last update time, last updated by, accessory information (optional)

Delete TUS data (multiple slots)

(Input) target user, slot IDs

Get accessory information of a TUS data (multiple users or multiple slots)

(Input) target user(s), slot ID(s)

(Output) last update time, last updated by, accessory information (optional), size of the binary data

Get accessory information of a TUS data (top 100 users sorted by update date/time from among friends)

(Input) sort conditions (ascending/descending update date/time), slot ID(s)

(Output) last update time, last updated by, accessory information (optional), size of the binary data

> **Note**
> The downloading of TUS data must be performed one at a time. However, accessory information can be obtained for multiple users or multiple slots all at once. Because of this, it is useful to store a summary of the data content or index information as accessory information.

Synchronous APIs and Asynchronous APIs

All APIs that operate TUS variables and TUS data are provided in pairs – one is synchronous and the other is non-synchronous.

For example, `sceNpTusTryAndSetVariable()` is an API that writes to a TUS variable by a synchronous process. It blocks other processing until communication with the server ends, and then returns. On the other hand, `sceNpTusTryAndSetVariableAsync()` is an API that writes to a TUS variable by an asynchronous process. It returns immediately without waiting for server communication to end; the result of the processing must be obtained after confirming the completion of the processing with `sceNpTusWaitAsync()` or `sceNpTusPollAsync()`.

Select and use appropriate versions of the APIs according to the implementation of your application.

APIs for an Actual User and APIs for a Virtual User

All APIs that operate on TUS variables and TUS data are provided in pairs – one targeting an actual user and another targeting a virtual user.

For example, `sceNpTusSetData()` is an API that uploads the TUS data of an actual user. The NP ID of the user must be specified as its argument. On the other hand, `sceNpTusSetDataVUser()` is an API that uploads the TUS data of a virtual user. A virtual user ID must be specified as its argument.

Select and use appropriate versions of the APIs according to the target user and specify arguments as appropriate. Note, especially, that the virtual user ID and the Online ID are in the same format. When the Online ID of an actual user is specified to an API that targets a virtual user, an error will return.

Examples

Examples of the title user storage usage are described as follows.

Reference Count and Voting

By preparing a TUS variable that others can write to, and using `sceNpTusAddAndGetVariable()`, you can see how many times you have been referenced.

The TUS variable of a virtual user can be counted up using `sceNpTusAddAndGetVariable()` to realize a vote-casting mechanism.

If you use this mechanism, however, adopt a scheme on the application side where each user's number of references or votes can be limited. For example, make a recording when the save data or title user storage is used to make a reference or to cast a vote, and control the count or frequency of this operation. A specification that lacks this control will lead to excessive burden on the server and/or result in an unrealistic score. Make sure the necessary precautions are taken to avoid such situations.

Display New User Content

By using this library in conjunction with the score ranking service, contents uploaded by a user can be displayed in the order of that they arrive. In other words, after a user content is stored in a TUS variable, that date and time can be registered as a score to display user contents in the order that they arrive.

Depending on the user content, however, there needs to be a process where inappropriate content can be rejected.

Total Number of Clears for All Users

The total number of times a game stage has been cleared in the world, for example, can be counted by counting up the TUS variable of a specific virtual user every time the stage is cleared, using `sceNpTusAddAndGetVariable()`.

However, the addition of a large value even by a single player due to a bug can easily corrupt the count; program your application with care. In addition, please design application implementation so that its progress will not be disturbed even if an unexpected value is set to the TUS variable.

Controlling the Access Privilege

The access privilege of a TUS variable or TUS data is set and fixed in advance. However, a specific TUS variable can be used as an application's unique access privilege flag and its operation can be determined by the application to dynamically control accesses to it. For example, the reference of a TUS data can be permitted to an opponent fought immediately before.

Check for Game Abandonment

When their position becomes disadvantageous during online play, some users unplug the network cable or switch the power off to abandon the game without losing. To check this type of game abandonment, use `sceNpTusAddAndGetVariable()` and add 1 to their TUS variable upon game start. Subtract 1 from the same TUS variable when the game completes normally, without a sign-out occurring for example. In this way, the number of times a user has abandoned a game can be counted. Switching the power off or deleting save data can be checked in this manner as well.

However, because there are cases when abnormal termination occurs contrary to the user's intention, given a network bug or a power out, make sure that game abandonment is not immediately deemed as illegal behavior.

Keeping Users' Best Scores

By using `SCE_NP_TUS_OPETYPE_GREATER_THAN` as the condition for a write in `sceNpTusTryAndSetVariable()`, a user's highest score can be recorded in a TUS variable. Unlike the score ranking service, there is no limit placed on the number of players whose scores can be recorded, as each user's score can be stored in their respective TUS variables, and the value will immediately be reflected, but the values are not collected for analysis. Use this feature in conjunction with the score ranking service to complement the drawbacks of each service.

Extended Storage (with a Cost)

A service can be realized by combining usage with service entitlements sold on PlayStation®Store, to enable users who pay to increase the size of data they can upload.

If you want to increase the size of TUS data or the number of slots for providing such a service, contact the SCE.

Online Save Data

TUS data can be used as online save data.

Notes on Coordination with PlayStation®3 and with PlayStation®4

Access privilege can be set separately for PlayStation®Vita, PlayStation®3 and PlayStation®4 per slot for the title user storage service. For example, a slot can be set as read-only from PlayStation®3 and PlayStation®4 and readable/writable from PlayStation®Vita.

If you want to set such a slot, a request is required on the PlayStation®Vita Developer Network (https://psvita.scedev.net/). By default, a title user storage registered/requested from the PlayStation®Vita Developer Network only allows access from PlayStation®Vita.

3 Using the Library

Initialization

To use the NP TUS library, first initialize the NP library and then initialize the NP TUS library.

(1) Load the PRX

Call `sceSysmoduleLoadModule()` with `SCE_SYSMODULE_NP_TUS` specified as the module ID to load the PRX of the NP TUS library.

(2) Initialize the NP Library

Call `sceNpInit()` to initialize the NP library. Set the NP Communication ID, NP communication passphrase and NP communication signature to the `SceNpCommunicationConfig` structure. These values are issued per application by applying on PlayStation®Vita Developer Network. Be sure to set the issued values correctly.

(3) Initialize the NP TUS Library

Call `sceNpTusInit()` to initialize the NP TUS library. With this process, a thread for inter-process communication is created in the application process and shell process respectively.

```
ret = sceNpTusInit(SCE_KERNEL_DEFAULT_PRIORITY_USER,
SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT, NULL);
if (ret < 0) {
    // Error handling
}
```

(4) Create a Title Context

Call `sceNpTusCreateTitleCtx()` to create a title context. At this time, if NULL is specified as an argument for NP Communication ID and NP communication passphrase, the value specified in `sceNpInit()` will be used. Specify target NP Communication IDs and NP communication passphrases other than NULL when using multiple NP Communication IDs.

```
int ret, titleCtxId;

ret = sceNpTusCreateTitleCtx(NULL, NULL, NULL);
if (ret < 0) {
    // Error handling
}
titleCtxId = ret;
```

Communication Processing

The NP TUS library provides synchronous and asynchronous APIs to obtain user information by communicating with the server. The procedure using synchronous APIs will be described first.

(1) Create a Request ID

Create a request ID to use for aborting or deleting a communication processing. Create this per communication process and delete it after communication process completion.

```
int ret, reqId, titleCtxId;

// Assuming that an appropriate value is stored in titleCtxId

ret = sceNpTusCreateRequest(titleCtxId);
if (ret < 0) {
    // Error handling
}
reqId = ret;
```

(2) Execute Communication Processing

Call the applicable communication function according to the information you want to obtain.

```
SceNpId targetNpId;
SceNpTusSlotId slotIdArray[1];
SceInt64 valueArray[1];

// Communication Processing
strncpy(onlineId.data, "user_foo", SCE_NET_NP_ONLINEID_MAX_LENGTH + 1);
ret = sceNpTusSetMultiSlotVariable (
    reqId,        // Request ID
    &targetNpId,  // Target user's NP ID
    slotIdArray,  // Pointer to the beginning of the array storing the target
slot
                  // IDs
    valueArray,   // Pointer to the beginning of the array storing the values
to
                  // set to the TUS variables
    1,            // Number of valid array elements. In this case, 1.
    NULL);        // Option reserved for future extension. Specify NULL.
if (ret < 0) {
    // Error handling
}
....
```

A synchronous API returns when communication processing completes. Check for an error once the function returns and obtain the result if there has been no error.

(3) Delete the Request ID

When the communication processing completes, delete the request ID.

```
// Delete request ID
sceNpTusDeleteRequest(reqId);
```

Asynchronous API Communication Processing

Asynchronous APIs are non-blocking, and they immediately return after starting a communication processing (without waiting for the result of the communication with the server). To know the completion of a communication processing, call either `sceNpTusWaitAsync()` or `sceNpTusPollAsync()`. `sceNpTusWaitAsync()` waits for the communication to complete if it hasn't already. On the other hand, `sceNpTusPollAsync()` returns immediately if the communication has not completed, with a return value of 1. The communication fully completes when these APIs return 0 and the application receives the result of the communication.

Communication Processing in Installments

For the following functions that upload and download TUS data, the data size to be sent/received per call can be specified to execute a communication processing over multiple function calls.

```
sceNpTusSetData()/sceNpTusSetDataAsync()
sceNpTusGetData()/sceNpTusGetDataAsync()
sceNpTusSetDataVUser()/sceNpTusSetDataVUserAsync()
sceNpTusGetDataVUser()/sceNpTusGetDataVUserAsync()
```

Note, however, that until the entire data gets sent/received, it will not be possible to start another communication.

```
int ret, reqId, titleCtxId;
SceNpId targetNpId;
char data[BLOCKSIZE];
SceSize remainSize, sendSize;

// Assuming that appropriate values are stored in titleCtxId, targetNpId, and
// data

ret = sceNpTusCreateRequest(titleCtxId);
if (ret < 0) {
    // Error handling
}
reqId = ret;

// Set total size of data to send
remainSize = TOTALSIZE;

while (remainSize != 0){
    // Determine send size
    if (remainSize < BLOCKSIZE){
        sendSize = remainSize;
    } else {
        sendSize = BLOCKSIZE;
    }
    ret = sceNpTusSetData (
        reqId,          // Request ID
        targetNpId,     // Target user's NP ID
        1,              // Slot number
        TOTALSIZE,      // Size of entire data to send
        sendSize,       // Size of data to send this time
        data,           // Data to send
        NULL,           // Specify NULL if there is no accessory information
        0,              // Set 0 when not setting accessory information
        NULL);          // Option for future extension. Specify NULL.
    if (ret < 0) {
        // Error handling
    }
    remainSize -= sendSize;
}
....

// Delete request ID
sceNpTusDeleteRequest(reqId);
```

Termination

(1) Destroy the Title Context

When you no longer need the title context, call `sceNpTusDeleteTitleCtx()` to destroy it.

```
int ret;
SceInt32 titleCtxId;

// Assuming that an appropriate value is stored in titleCtxId

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

(2) Terminate the NP TUS Library

Call `sceNpTusTerm()` to terminate the NP TUS library.

```
// The thread created for communication between processes will stop
sceNpTusTerm();
```

Note
When `sceNpTusTerm()` is called, the created title contexts and request IDs will be automatically deleted, however, it is recommended that this function be called after these are explicitly deleted from the application side.

(3) Terminate the NP Library

Call `sceNpTerm()` to terminate use of the NP library.

```
// Always succeeds
sceNpTerm();
```

Then, unload PRX by calling `sceSysmoduleUnloadModule()` with `SCE_SYSMODULE_NP_TUS` specified for the module ID.

4 Application Development

Certain steps are required when developing an application using the title user storage service. Registration to relevant services, and the use of the Server Management Tools for the title user storage service configuration are described below.

Registration

When developing an application that uses PSN℠ services including the title user storage, go to the PlayStation®Vita Developer Network to register and make various requests.

Registering the Title

Upon starting the development of your title, go to the PlayStation®Vita Developer Network and register your title. At this time, also make settings to use the title user storage service.

Request for the QA Environment

About a week before the master is planned to be submitted, conduct request for the QA environment from the PlayStation®Vita Developer Network.

Note
The title user storage service will become usable once the master disc is submitted and the QA completes. If there is a special reason to do so, the start date of the storage service can be set as well.

PSN℠ Server Management Tools

The server settings for the title user storage are configured using Server Management Tools (SMT). By using the SMT, the number of slots and data size to be prepared for each user and virtual user, as well as access privilege settings, can be set.

For details regarding the SMT, refer to the document "Server Management Tools NP Title User Storage Tools User's Guide" document.

5 Notes

Notes on Server Burden and the Frequency of Communication Executions

To prevent the unnecessary increase of the load burden on the server of PSN℠, note the following points when designing and implementing application specifications. Special care is necessary for large-scale applications. Contact SCE when in doubt.

Send Information Collectively Whenever Possible

When using a TUS variable to record the number of defeated enemies, for example, do not communicate with the server every time an enemy is defeated. Instead, select an appropriate timing, such as, at the end of each stage, to send information collectively to the server.

Do Not Access TUS Data of the Virtual User Unnecessarily

The title user storage server is designed in such a way that the concentration of processing burden regarding the virtual user cannot be avoided. Because of this, minimize access to the TUS data of virtual users.

Always consider beforehand whether the score ranking service or the title small storage service can be used instead of the virtual user for realizing the feature you seek, and use them wherever it is practical to do so.

Prevent Continual Reloads and Votes

When implementing the reload feature, prevent a user interface where the user can repeat reloads. For example, do not adopt a specification where the continued pressing of a button means continued reloading. Also, avoid specifications that encourage the user to perform reloads – such as, where the reference count is increased when the user downloads the same data repeatedly. Record the time lapse since the previous download in the save data, for example, and adopt a specification where unnecessary downloads cannot be repeated.

Do Not Obtain Information until Needed

Obtain information to display to the user when the user makes an operation, when that operation is made. Avoid a design where the information that does not get displayed until the option menu is opened, or until the user scrolls down, is obtained before that operation.

Directly Obtain Information of the Other Party by P2P Communication

Obtain the information of the party with which you are performing P2P communication directly from the other party wherever possible.

Communication Frequency

The frequency of TUS data communication with the server should be kept within approximately once per five minutes. Communication of the TUS variable 8 times is equivalent to communication of TUS data once. The frequency can be temporarily increased when a user performs a special performance, however, avoid a specification where the user is encouraged to do so.

Minimize the Number of Communication Processes to Execute at Once

When executing multiple communication processes at once, make sure the number of processes does not exceed 8. In a design where user operation is blocked until multiple processes end, an extremely long waiting time may occur depending on the network state, and smooth operation will not be possible. Setting **★Debug Settings > PSN℠ > NP Debug** to **On** from Settings application of the system software will generate a 2-second delay time after the end of a communication process of the NP TUS library. Make sure that user operation does not break down in this mode as well.

Note that "at one time" means one scene of a game. When calling multiple APIs at "one time", there are separate implementation limitations that apply. For details, refer to the "Communication Processing in Installments" item of the "Communication Processing" section in the "Using the Library" chapter.

In addition, for the purpose of coordinating with friends, APIs are provided for obtaining up to 100 users from among friends by specifying sort conditions such as ascending/descending TUS variable numerical values and ascending/descending update date/time.

Notes on Security

Data Security

The NP TUS library is not designed with privacy protection in mind. Information such as, the user's credit card number, telephone number, address, or a password for some service, should not be stored on the title user storage.

Although it is unlikely that a server will be falsely represented since server authentication is performed on the client side, client authentication on the server side is not performed. Thus, data set to the server may be obtained from a PC, for example. Make sure data that should not be seen by the user, such as, a secret key, are also not stored on this space.

Other Notes

Danger of a Deadlock

When a TUS variable is improperly used like in a semaphore, a deadlock may occur when the application hangs up and it may be impossible to recover from this deadlock, even when the application is restarted. Design your application so that it can be continued even if the TUS variable or TUS data becomes invalid in terms of application specifications.