Save Data User's Guide

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Document serial number: 000004892117

Table of Contents

1 Save Data Overview

About Save Data

Save data includes data for indicating progress status of a game play, high score information, character data edited by a user and setting data such as volume setting for sound effect , button preferences, and message display speed setting.

Media for Saving the Data

Only the save data which is saved in the memory card or PlayStation®Vita card can be accessed by an application directly. Save data is created automatically when the application is started up. The media where it is created are decided when that application is packaged.

> **Note**
> Although save data may be copied in an external recording device by a feature of the system software, application cannot access such data.

When running an application from debugger

When running an application under development from a debugger, save data can be saved in the HDD of a development host computer or the memory card.

> **Note**
> For the differences in operation depending on the storage destination, refer to the "Application Development Process Overview" document.

Save Data Features

Save/load operating procedure of save data must be easy to understand, and a system to appropriately deal with a lack of free space or input/output errors should be established because save data is an important data for users. Also, it is desired that the user interface of each application is designed to have the same basic commonality. Save data feature provides appropriate user interfaces and supports rule-based file creation.

2 Characteristics of Save Data Features

The following are the characteristics of the save data feature.

- Save data feature consists of three API groups of Common Dialog, library and file access API, and samples.
- The features provided by each API offer basic operations.
- From a simple access method to a complicated method are completely covered by combining the basic operations.
- Samples for the simple access method are provided so the method can be incorporated into games without difficulty.
- Programmers who are familiar with the save data feature can develop a complicated access and an optimized high-speed access by using the API directly instead of the samples.
- The unencrypted data files supported in PSP™ (PlayStation®Portable) and PlayStation®3 are not supported for security purposes. All of the data files are automatically encrypted. For convenience, unencrypted data is allowed to use only during development.

Typical Processing for Accessing Save Data

When an application accesses save data, various types of screen processing and access processing are required depending on the configuration of the save data (single/multiple, etc.). The save data feature supports typical processing including the screen processing. The typical processing are shown below, and an explanation on whether the processing can be performed by the save data feature is added.

List selection save/load/delete

Display a list of save data specified by an application, and then save/load/delete the save data selected by the user, having required the user's confirmation if necessary.

-> Can be performed by the save data feature

Fixed save/load/delete

Save/load/delete the save data specified by an application upon asking the user to confirm and respond, if needed.

-> Can be performed by the save data feature

Automatic save/load/delete

Save/load/delete the save data specified by an application without notification to or response from the user.

-> Can be performed by the save data feature

3 Save Data Free Space

When save data free space is insufficient, the writing of save data will fail. Strictly speaking, there are two types of save data free space: "file system free space" and "save data quota". If either of these free spaces is insufficient, the writing of save data will fail. Their meaning and handling methods also present differences, which will be explained in this chapter.

Note
As a general rule, save data quota in applications created using SDK 3.50 or later is fixed at 1 GiB. Although this chapter describes the method for calculating save data size, 1 GiB is considered sufficient as save data quota for most applications, and it is not necessary for most applications to be aware of insufficient save data quota.
However, the TRC (Technical Requirements Checklist) prohibits the occurrence of insufficient save data quota. If you are developing an application with specifications that may store save data of 900 MiB or more, please contact SCE in advance.

Location of the Directory where Save Data is Mounted

Depending on the application's settings, save data will be mounted either to the PlayStation®Vita card or to the memory card. It is possible to specify either of these for mounting when creating the pkg file.

Save Data Quota

The save data quota can be set by writing to the application's system files. The application can write save data within the quota. If the writing of save data is attempted when there is not sufficient remaining space for the save data quota, an insufficient save data quota error will occur.

Applications must be made so as to avoid the occurrence of insufficient save data quota errors.

The formula for calculating the free space consumed when files and directories are written will be described later.

Note that the save data quota is only declared as the quota that will be used by an application; thus it is not meant that the quota will be physically allocated at the time of application start-up. Even if there remains sufficient save data quota, the file system free space may become insufficient (as described below).

File System Free Space

This is the free space of the file system itself on PlayStation®Vita cards and memory cards. If file system free space is insufficient, a file system free space error will return when writing save data, even if there is enough remaining space for the save data quota.

If this error occurs, the application must inform the user that file system free space has become insufficient by using the system message of Message Dialog or Save Data Dialog for displaying the file system insufficient free space errors.

Handling of File System Free Space Insufficiencies

In the case of applications that store save data on the PlayStation®Vita card, consider in advance the file system free space of the PlayStation®Vita card and the size of patches and additional contents that may be stored on it; set the save data quota so that an insufficient free space does not occur for the file system..

Applications that store save data on memory cards cannot estimate the memory card's file system free space because other applications and system software also write to the memory card. For this reason, the application must handle insufficiencies of file system free space in an appropriate manner.

Note
Currently, master submission of applications where save data is saved to the PlayStation®Vita card (VC-VC configuration) is prohibited (refer to TRC [R3167]), but an already sold VC-VC configuration application must continue to consider the save data quota when creating patches, etc.

How to Calculate Save Data Size

The size that is consumed when writing a file and directory as save data is calculated in cluster units. A cluster size of save data is 32 KiB. Thus, the consumed size will be CLUSTER SIZE * 32 * 1024 (bytes).

- Cluster quantity consumed for files
  (fileSize + (CLUSTER_SIZE-1)) / CLUSTER_SIZE
- Cluster quantity consumed for directories
  1 cluster for 1 directory

Note, however, that when writing a file and/or directory to save data, an extra one cluster will be consumed in addition to the clusters consumed for the actual file and/or directory.

Free Space Required for a Write and the Actually Consumed Size

The size derived in the above calculation is the size of actual data used for calculating the free space. The sizes of directory entry information and the save data management file (system file) must additionally be included for a write; thus, an additional free space of the size indicated below will also be required.

- When creating a new file: 5 clusters
- When creating a new directory: 3 clusters
- When overwriting without changing the file size: 0 clusters
- When overwriting with a file size increase: file size after the overwrite converted to the number of clusters + 1 cluster

Note
The above are maximum values of requested sizes for calculating free space. Note that it is not always true that the above sizes will be consumed after a write.

Due to the above behavior, it is necessary to be careful when attempting to write a size equivalent to the remaining space of the save data quota.

- Example 1: Calculation when creating a new savedata0:/dir/data.dat
  (where dir is the new directory and the size of data.dat is 10 KiB)

Consumed clusters of savedata0:dir (a)	:	1 + 1 = 2 [clusters]
Requested clusters for savedata0:dir (b)	:	a + 3 = 5 [clusters]
Consumed clusters of data.dat (c)	:	(10240 + (32*1024-1)) / (32*1024) + 1 = 2 [clusters]
Requested clusters for data.dat (d)	:	c + 5 = 7 [clusters]
Total requested size	:	(b + d) * 32 = (5 + 7) * 32 = 384 [KiB]
Total consumed size after write	:	(a + c) * 32 = (2 + 2) * 32 = 128 [KiB] (Depending on the file size and number of files, the total consumed size may vary up to 384 KiB of the total requested size)

- Example 2: Calculation when overwriting without increasing the size of savedata0:/dir/data.dat (where data.dat is 10 KiB)
  Requested clusters for savedata0:dir/data.dat : 0 [clusters]
  Consumed size increased after write : 0 [clusters]
- Example 3: Calculation when overwriting with the size of savedata0:/dir/data.dat increased from 10 KiB to 11 KiB
  Requested clusters for savedata0:dir/data.dat : (1 * 1024 + (32*1024-1)) / (32*1024) + 1 = 2 [clusters]
  Consumed size increased after write : 0 [clusters] (Because the file size after conversion to number of clusters does not increase)
- Example 4: Calculation when overwriting with the size of savedata0:/dir/data.dat increased from 10KiB to 65 KiB
  Requested clusters for savedata0:dir/data.dat : (55 * 1024 + (32*1024-1)) / (32*1024) + 1 = 3 [clusters]
  Consumed size increased after write : 2 [clusters] (Depending on the file size, the number of requested clusters may vary up to 3 clusters)

Save Data Overwrites That Do Not Cause Insufficient Free Space

As mentioned above, for an overwrite without an increase in file size or number of directories, the save data quota and file system free space will never be insufficient. Using this behavior, for a game's system and progress data with frequent updates, create data with maximum sizes that can be estimated at the start. From thereon, by only performing overwrites without a file size increase, it will not be necessary to be aware of the occurrence of insufficient free space every time save data is written since after its initial creation timing.

Reserved System Size for Save Data Quota

Applications cannot use the entire save data quota specified when creating pkg. Given that, at the time of the first startup, the system automatically creates system files (not visible from the application) for a total of 24 clusters (768 KiB) under save data, the space allocated as save data quota that can actually be used by the application will decrease proportionately.

How to Handle the Situation Where File System Free Space Becomes Insufficient

If file system free space becomes insufficient, an error indicating that the free space on the file system is insufficient will be returned by the `sceAppUtilSaveDataDataSave()` save data writing function. In response, the application should display an error using the Message Dialog's or Save Data Dialog's defined message for indicating insufficient free space of the file system. At this time, the lacking space will automatically be displayed as complementary information by the system. Because this lacking space includes the earlier described system margin, the displayed lacking space may differ from the size of data that the application attempted to write.

Note
In this SDK, the system margin for the file system free space is 2048 KiB. This value is subject to change in the future. Applications need not be aware of this size.

4 Save Data Writing Frequency

The frequency with which save data is written to PlayStation®Vita cards is subject to restrictions from the point of view of operation (refer to TRC [R3074]). Specifically, the total of the writing sizes derived with the calculation methods below must not exceed the writing size allowed within a unit of time.

- When writing files
  The size obtained by rounding up the writing size (byte) to cluster size (32 KiB)
  *In cases where writing is performed from an offset position greater than the current file size, the difference will also be added up as write volume.
- When file size is changed
  The size obtained by rounding up file size variation (byte) to cluster size (32 KiB)
  *Specifying only the offset position with 0 bytes as writing size when writing save data will enable processing for changing file size.
- When creating directories
  1 cluster size per directory (32 KiB)
- When deleting files or directories
  1 cluster size per file/directory (32 KiB)

Example: writing size when creating a new savedata0:/dir/data.dat (setting data.dat to 10 KiB)

Writing size of savedata0:dir	:	32 KiB
Writing size of data.dat	:	(10240 + (32*1024-1)) / (32*1024) = 32 KiB
Total writing size	:	32 + 32=64 KiB

Note
Note that the method for calculating writing size described here is simplified, and differs slightly from that for calculating the actual save data size against the save data quota.

5 API Group

API of the save data feature has three groups. Each group's purpose is different, and an application accesses save data by combining necessary API.

Common Dialog

The GUI for the save data feature is included in the system software and is called from an application for use. The following GUI parts are prepared and can be used in any combination of the parts.

- Save data list
- Confirmation dialog (confirmation of start/overwriting/completion)
- Error dialog (lack of free space/data breakage/text specification)

A seamless screen transition can be performed by inserting animated images when transitioning. In addition, the GUI status management adopts the polling method done by `Init()/GetStatus()/GetResult()/Term()`.

For details on how to use Save Data Dialog, refer to the "Save Data Dialog Overview" document.

Note
The system of Common Dialog is used by other features as well as the save data feature.

Library

The features which require a special access authority, such as mount/unmount feature for the save data directory are provided as a library. The following processing can be performed as a library.

- Mount/unmount processing of save data directory
- Creating save data slot
- Deleting save data slot
- Obtaining a list of save data slot
- Writing save data
- Deleting save data
- Obtaining save data free space

Files for the system (param.sfo, etc.) which are required for displaying on the system software, etc. are automatically created by the library by using the given argument, etc.

File Access API

Data files created by an application which is included in save data can be accessed through file access APIs, but use of a dedicated library APIs are required for processing related to writing files. Below are the listed file access APIs which will result in an error when called for save data.

- `sceIoOpen()` *when specifying `SCE_O_RDWR`, `SCE_O_CREAT`, `SCE_O_TRUNC`
- `sceIoWrite()`
- `sceIoPwrite()`
- `sceIoChstat()`
- `sceIoChstatByFd()`
- `sceIoRemove()`
- `sceIoMkdir()`
- `sceIoRmdir()`
- `sceIoRename()`
- `sceIoSync()`
- `sceIoSyncByFd()`

The data files are encrypted for writing and are decrypted for reading automatically. An application does not need to particularly consider the processing.

Library APIs for Writing Save Data

As stated above, write processing for save data cannot be performed using `sceIo*` APIs. Instead, use the following application utility library APIs.

- `sceAppUtilSaveDataDataSave()`
  - Up to 5 files you wish to save can be listed and saved with on-memory specification.
  - While calling the APIs, no-suspension state will be entered, and the listed files will be written atomically.
  - If there are no directories in the middle of the file path, they will be created automatically.
  - By specifying a save data slot, it is possible to synchronize file storage with the slot.
  - The total specifiable file writing size is 1 MiB.
  - If free space is calculated before actual writing and there is not sufficient free space, an error for insufficient free space will return together with the lacking size.

Note
When large-sized data is written in no-suspension state, suspension of the application will be delayed proportionately, thus diminishing the ease of user operations. For this reason, writing volume is limited to 1 MiB.

- `sceAppUtilSaveDataDataRemove()`
  - Up to 5 files/directories you wish to delete can be listed and deleted.
  - While calling the APIs, no-suspension state will be entered, and the listed files/directories will be deleted atomically.
  - If the specified path is a directory, files under that directory will be deleted recursively.
  - By specifying a save data slot, it is possible to synchronize file deletion with the slot.
- `sceAppUtilSaveDataGetQuota()`
  - It is possible to obtain currently used space and the maximum space set to save data.
  - Maximum save data space can be specified with the application's system files.

6 Save Data Usage Procedure

Specific procedure to access the save data is described below.

Example of Fixed Save/Load

In order to use the save data feature, it is necessary to make the application utility library ready to use.

(1) Preparation for use of application utility library

Load the module

The application utility library is preloaded automatically when an application process is launched.

Initialize the library

In order to use the application utility library, the library must be initialized by calling the `sceAppUtilInit()` function. Call this function after setting appropriate parameters in the `SceAppUtilInitParam` structure. At this time, the start parameters of the application can be received. All of the following processing of `sceAppUtilXxx()` function will return errors if the library is not initialized.

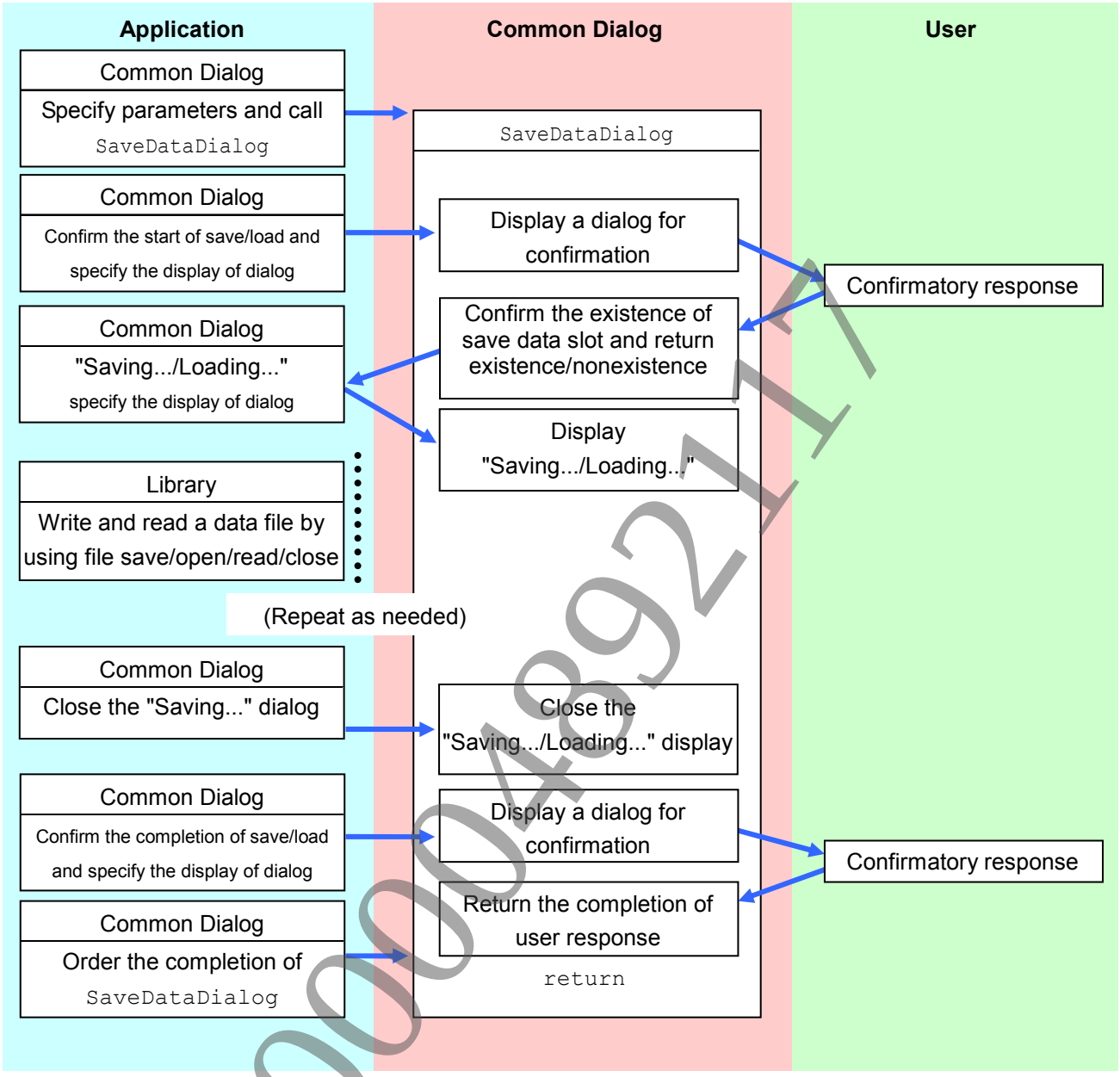
```
/* Initialize the library */
ret = sceAppUtilInit(
              &param,
              &bootParam );
```

(2) Use the library

The application utility library is available after loading the libapputil module and initializing the library. Execute arbitrary processing including the save data feature by using `sceAppUtilXxx()` function.

(3) Using Common Dialog and an access through file access APIs

Figure 1 Basic Processing Procedure of Fixed Save/Load



(Note) The prefix of library functions `sceAppUtilSaveData` has been omitted.

For details on how to use Common Dialog and the library, refer to the "Save Data Dialog Overview" and "Application Utility Overview" documents respectively.

(4) Terminate the use of the application utility library

Terminate the library

After completing the arbitrary processing which uses the application utility, call `sceAppUtilShutdown()` to terminate the library.

```
/* Terminate the library*/
ret = sceAppUtilShutdown();
```

Unload the module

The module is unloaded automatically at the end of the application process.

Unencryption Feature of Save Data

For convenience during development, this feature is provided to save the data file without encryption. In this case, however, the readable data file is also limited to an unencrypted file.

> **Note**
> For details on save data during development, refer to the "Application Development Process Overview" document.

7 Breakage of Save Data Slots

Managing Slot Status

For example, if multiple data files are coordinated and stored in a single save data slot, there are cases in which it is necessary to maintain consistency between those files. If the memory card or PlayStation®Vita card is removed, or the power supply is forcibly interrupted during this type of save, it will not be possible to guarantee the validity of the data files corresponding to that save data slot.

In order to detect this kind of situation, save data slots have a feature to manage slot status. There are two types of slot status: "Available status" and "Broken status". These are managed with the following procedures.

When saving data

(1) The slot status of the save target save data slot is checked.

(2) If a save data slot does not exist, the save will be handled as a new save and the required data file set will be created.

(3) If a save data slot exists, the save will be handled as an overwrite save.

- If the slot status is "Available", the required data files will be updated.
- In the case of a "Broken status", in the same manner as for a new save, the required data file set will be saved. At this time, it is recommended that the user be warned if any data will be lost.

Note
`sceAppUtilSaveDataDataSave()` automatically carries out processing to store data after setting the specified slot status to the "Broken status" and setting the slot status to "Available status" after normal termination.
This can also be performed by calling `sceAppUtilSaveDataDataRemove()` with `SCE_APPUTIL_SAVEDATA_DATA_REMOVE_MODE_KEEP_SLOT` specified.

When loading data

(1) The slot status of the load target save data slot is checked.

(2) Load is not carried out if a save data slot does not exist.

(3) When a save data slot exists:

- If the slot status is "Available", the data file is loaded.
- In the case of a "Broken status", notify the user of this fact ("The file is corrupt.", displayed with the system defined message display feature of Save Data Dialog). The data will not be loaded.

Slot broken Status Debugging Feature

The debugging feature ★**Debug Settings -> Game -> Fake Save Data Slot Broken**, which is used to change the status of any desired save data slot to always be "broken", is available for performing an operation check of behavior when a save data slot's status is "broken". For details, refer to the "System Software Overview" document.

Precautions

Requirement in Using Slot Status

Save data slots are mainly intended to display GUIs by using Save Data Dialog; however, due to the reasons listed below, important data such as application progress data must be processed by including save data slot status even in the case of autosave/load processes that do not employ Save Data Dialog:

- To enable check processing when data is broken with the aforementioned Slot broken Status Debugging Feature.
- To enable the detecting of broken data before it is actually read, because any failure of the previous save processing to terminate normally is found immediately.
  This is particularly useful when wishing to maintain consistency among data saved over multiple times, such as when upgrading multiple parts of a file simultaneously or upgrading multiple files at the same time.

8 Notes Regarding the Speed of Save Data Writes

To prevent speed of save data writes from slowing down, implement with the following points in mind.

Buffer Alignment

Align the write buffer to 64 bytes. When buffer alignment is not set to 64 bytes, write speed will notably deteriorate.

Write Size

When performing a data update of size 256 KiB or more, it is recommended that the update be performed at once and not in increments. The smaller the size is, the slower the write speed will be.

Write Offset

The offset of the write start position should be a multiple of 32 KiB or 512 bytes. When the offset is not a multiple of 512 bytes, write speed will notably deteriorate. When the offset is a multiple of 32 KiB, write speed will be faster.

Number of Files/Directories

Make sure the total number of files and directories saved directly under one directory does not exceed 100. When the directory entry size increases, file access speed (including read speed) will drastically deteriorate due to the characteristic of exFAT.

Note
Also refer to the `sceAppUtilSaveDataDataSave()` section of the "Application Utility Reference" document and the "Data Placement That Causes a Decrease in File Access Speed" section of the "Application Development Process Overview" document.
The conditions for write speed decrease described in this chapter do not apply to the subsequently-described safe memory feature.

9 Safe Memory

Safe memory is a nonvolatile memory area applications can use. In PlayStation®Vita, applications may be suspended/terminated at any time. The memory contents saved to the safe memory can be restored the next time the application is started. For example, the state of the application can be saved into the safe memory step by step while the application is running. In this way, the user can restore the state at the termination of the application from the safe memory when starting up the application the next time.

Safe Memory Size

Safe memory size is 64 KiB, and it is filled with all zeros when the application is started up for the first time and is initialized.

How to Access Safe Memory

The memory area of the safe memory cannot be accessed directly. To perform such access, use the dedicated safe memory access APIs for performing memory copy from/into the safe memory.

Since safe memory access is a memory copying process, it is fast and does not require paying attention to writing frequency like with common overwritable flash memories.

Note
Concerning the safe memory access APIs, refer to the "Application Utility Reference" document.

Timing of Safe Memory Storage

Safe memory is stored as savedata0:sce_sys/safemem.dat when the application is suspended. Since it is not saved as a file if savedata0: is not mounted (only during development), contents will be initialized every time the application is started up.

Application suspension occurs in the following cases:

- When the PS button is pressed during application running, the screen returns to the LiveArea™ screen
- When the power button is pressed during application running and PlayStation®Vita enters sleep state.
- When batteries are low and PlayStation®Vita automatically enters sleep state.
- When the target is rebooted during development

Timing of Safe Memory Initialization

Safe memory will be lost in the following cases, and will be in initialized state when the application is next started up.

- When the PlayStation®Vita card is removed and the application is terminated during running the application (excluding suspension) that has been set to save save data on the PlayStation®Vita card.
- When the memory card is removed and PlayStation®Vita is restarted during running the application (excluding suspension) that has been set to save save data on the memory card.
- When the system unexpectedly crashes while an application is running (except during suspension).

Precautions when Using Safe Memory

As stated above, safe memory will sometimes be initialized. Store as save data information whose loss would be serious for the user, such as the application progress status, etc.

Note
In the case information depending on save data is placed in the safe memory, as a precaution for the possible loss of the safe memory we recommend keeping the same information on the save data side also, and synchronizing information of safe memory and save data at an appropriate timing when the application is started up or is running.

Safe Memory Debugging Feature

The debugging feature ★**Debug Settings -> Game -> Init Safe Memory**, which always initializes safe memory at application cold boot, is available for performing an operation check of behavior when safe memory is initialized. For details, refer to the "System Software Overview" document.

10 Saving Large Amounts of Save Data

This chapter describes how to save save data whose size exceeds 1 MiB.

Background

The application utility save data saving function can only save up to 1 MiB of data each time the function is called. Save data is saved atomically, so when the save data saving function is called, suspending the application via the PS button will be delayed. Save processing of a large file, at this time, will cause decreased suspend response. In order to avoid this, the amount of data which can be written at any one time is strictly limited.

This limit is also in place out of consideration for the durability of flash memory.

How to Save

If, after giving close consideration to the background described above, you still wish to save more than 1 MiB of save data, you can do so by following the procedure described below.

```
/* How to save over 1 MiB of save data */
SceAppUtilSaveDataSlotId = 0;
SceAppUtilSaveDataSlotParam slotParam;
memset(&slotParam, 0x0, sizeof(slotParam));

/* Obtain the parameters for the save data slot to be saved to */
ret = sceAppUtilSaveDataSlotGetParam( slotId, &slotParam, NULL );

/* Manually set the slot's status to "broken" */
slotParam.status = SCE_APPUTIL_SAVEDATA_SLOT_STATUS_BROKEN;

/* Reflect the broken status */
ret = sceAppUtilSaveDataSlotSetParam( slotId, &slotParam, NULL );

/* Save up to the actual amount of data to be saved -1 MiB */
while( ... ) {
        /* Save the file without specifying the slot */
        ret = sceAppUtilSaveDataDataSave( NULL, ... );
}
/* Specify the slot only when writing the last section of actual data */
SceAppUtilSaveDataSlot slot;
memset(&slot, 0x0, sizeof(slot));
slot.id = slotId;
slot.param = &slotParam;

ret = sceAppUtilSaveDataDataSave( &slot, ... );
```

If the application is suspended and terminated during the while statement, the next time the application is started, the save data slot's status will be detected as "broken". `sceAppUtilSaveDataDataSave()` automatically sets the status of the specified save data slot to "broken" when it begins processing, and, if processing is successful, automatically sets it to "available", so if the above save process is completed, the save data slot will be set to "available".

11 How to Read Save Data from Another Title

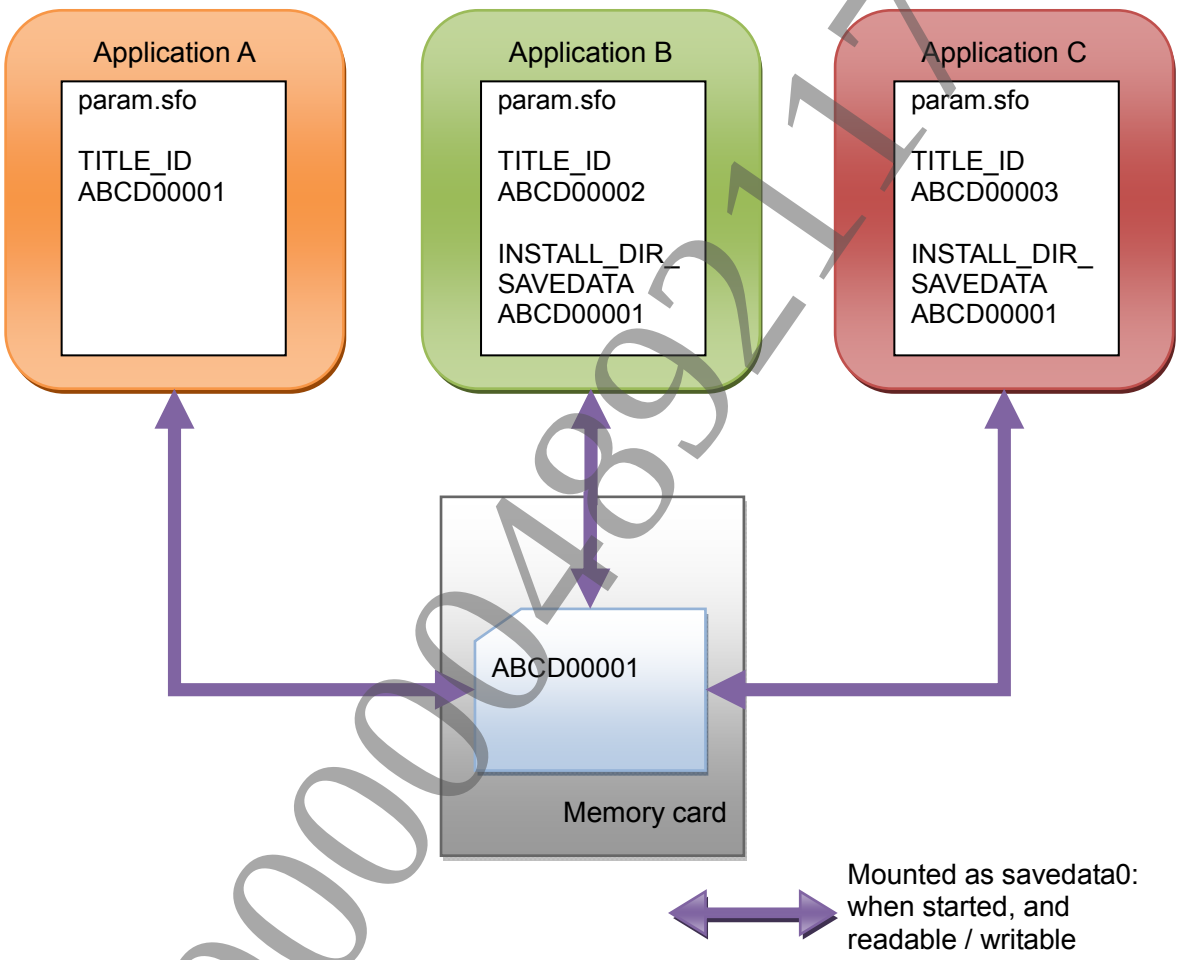
This chapter describes how to have an application read save data of another title.

There are two ways to read save data of another title: "shared save data" and "save data transfer".

Shared Save Data

This feature makes it possible for multiple applications to write to and read from a shared save data directory. All applications sharing the save data directory can write and read save data.

Figure 2 Save Data Directory Sharing



Usage Procedure

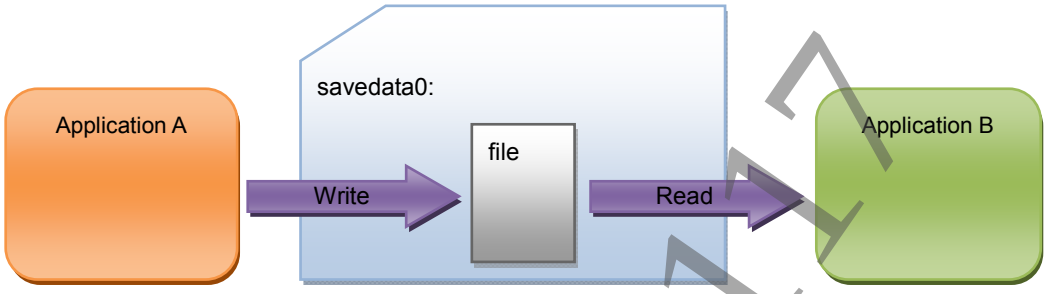
The save data sharing feature can be used by multiple applications to share save data by setting the same product code for the param.sfo INSTALL_DIR_SAVEDATA parameter.

Usage Conditions

- The save data must be stored on a memory card
  Applications with a VC-VC configuration can only access save data on the PlayStation®Vita card, so they cannot use the save data sharing feature.

- The same passcode must be set for all applications sharing save data
  The passcode set for the application which created the save data is stored in the save data directory. In order for the save data directory to be mounted as savedata0: when the application is started, the passcode set for the application which is started must be identical to the passcode stored in the save data directory.
- Special attention must be paid to save data compatibility
  Save data sharing allows save data to be freely read from and written to by different applications. Care must be taken to avoid saved data from one application being not read by another application which cannot interpret it, causing it to hang.

Figure 3 Save Data File Sharing

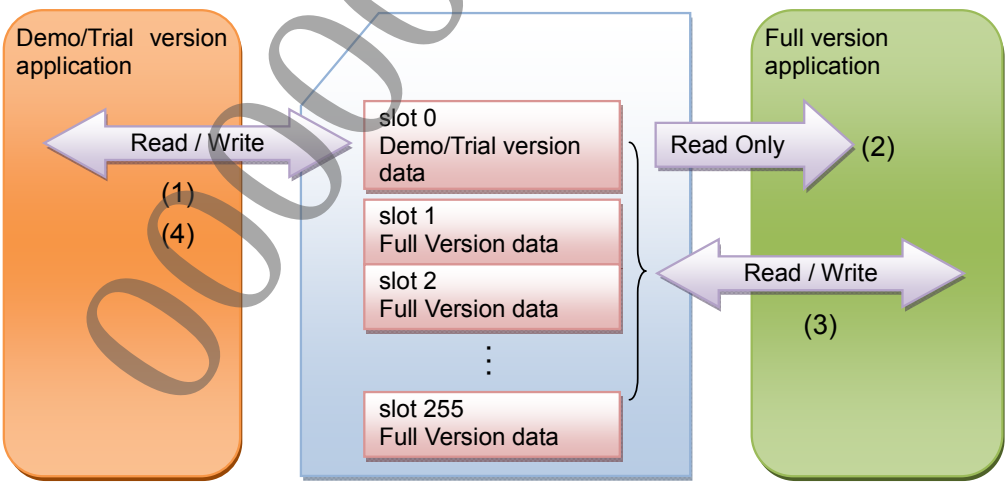


Because of the above, sufficient consideration must be given in advance on the application side in order to use the save data sharing feature. Use of the feature is strongly recommended only in the following situations.

- When the same application binary is released in different regions with a different product code for each region, you wish to share save data between those applications
- When you release a non-upgradable application demo/trial version, and wish to carry save data over to the full version

Below is an example of save data design when using the save data sharing feature to share save data between the demo/trial version application and the full version application.

Figure 4 Carryover of Demo/Trial Version Application Save Data



(1) The demo/trial version application only writes data to and reads data from slot 0.

(2) The full version application reads demo/trial version data from slot 0, and saves it as full version data to slots other than slot 0.

(3) The full version application does not use slot 0 thereafter, except to carry over data.

(4) Even after the user has played the full version application, the demo/trial version application does not read full version data.

Save Data Transfer

The save data transfer feature provides read-only access to save data from arbitrary applications.

Usage Procedure

(1) Set the following parameters in the param.sfo of the application which will read the save data.

INSTALL_DIR_SAVEDATA_ADD_* (*: 1 to 7)	:	Can contain up to 7 product codes for save data to be read. (Setting required)
IMPORT_SAVEDATA_INFO	:	Can contain up to 1024 bytes (including NULL characters) of explanation for users regarding save data content which will be migrated. (Setting may be required. Described later)
IMPORT_SAVEDATA_INFO_* (*: 2-digit number)	:	IMPORT_SAVEDATA_INFO multilingual support field. (Optional setting when IMPORT_SAVEDATA_INFO is set)

(2) Use the save data mount function on the application which is to read the save data to read the data.

`sceAppUtilSaveDataMount()`	:	Mounts the save data directory of the product code(s) specified in INSTALL_SAVEDATA_ADD_* as savedata1:.
`sceAppUtilSaveDataUmount()`	:	Unmounts savedata1:.
Other application utility save data APIs	:	Only read-related features can be used on savedata1:.
Save Data Dialog features	:	Displays save data slot for savedata1: in the same manner as savedata0:.
`sceIoOpen()/sceIoRead()/sceIoClose()`	:	Reads files in savedata1:.

Usage Conditions

- Requires a memory card
  The save data mount function can only be used with save data on a memory card. In order to read save data from an application with a VC-VC configuration, with save data stored on a PlayStation®Vita card, the user must first import the save data to a memory card via the system software (described later).
- Save data is read-only
  savedata1: is a read-only drive, so attempting to use write processes will result in an error.

Save data transfer can be configured and implemented entirely by the application reading the save data, making it easier to implement than save data sharing. It can be used in the following situations.

- To have a sequel application carry over some of the save data from the preceding application
- To carry over demo/trial version save data to a non-upgradable full version application
- To confirm that the user possesses save data for a specific application (this makes it possible to perform special promotions in which applications check for the presence of save data for a specific application, and give the user a present if the save data exists)

Save Data Transfer Procedure

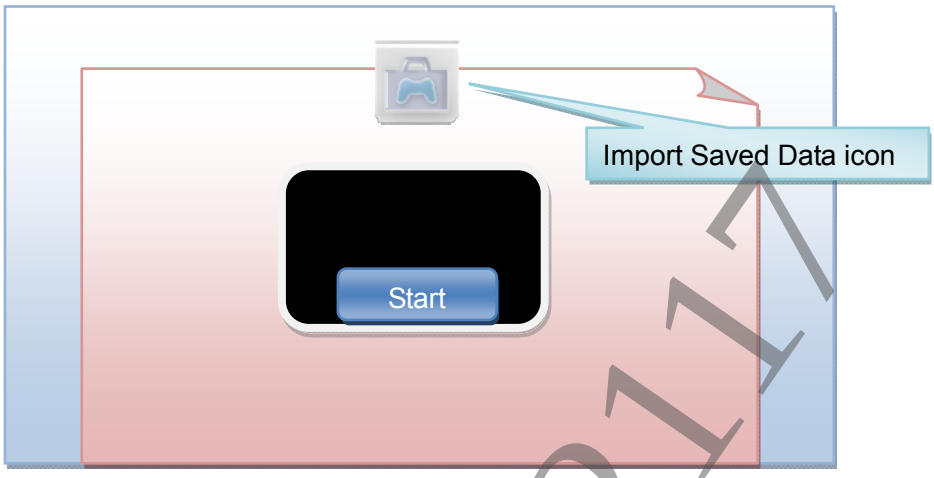
Below is a detailed description of the save data transfer procedure.

Save Data Import

When the save data to be read in is located on a PlayStation®Vita card, the user must first import that save data to a memory card. The LiveArea™ of applications with IMPORT_SAVEDATA_INFO set in their param.sfo will display an import saved data icon. Tapping this import saved data icon will begin the process of importing the save data from the PlayStation®Vita card.

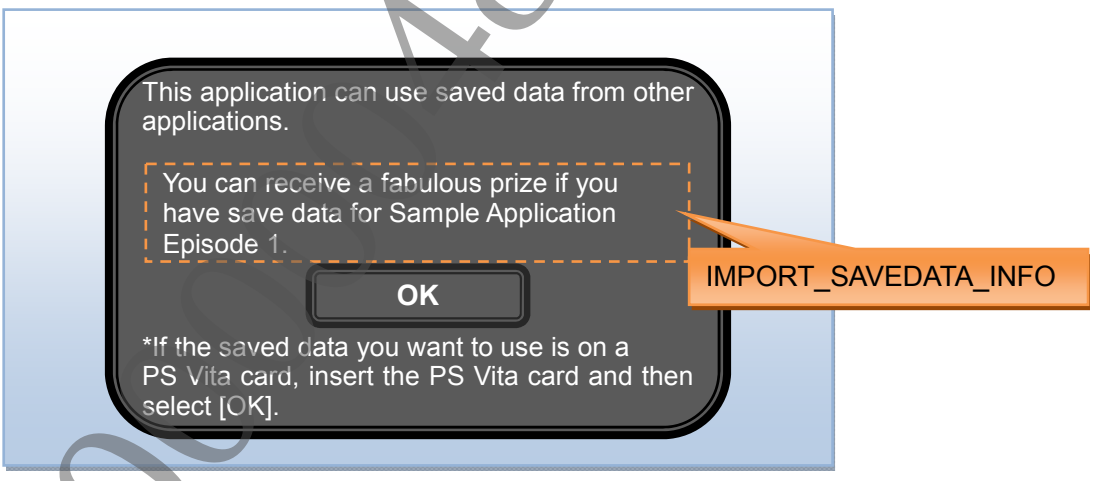
Note
Importing save data is only necessary when wishing to transfer save data from an application with a VC-VC configuration that stores save data on a PlayStation®Vita card. If the save data you wish to read is on a memory card (belonging to a VC-MC/MC-MC-configured application), follow the flow in "Mounting Save Data". In such a case, do not set IMPORT_SAVEDATA_INFO to param.sfo.

Figure 5 LiveArea™ of Application Performing Save Data Transfer



When the import saved data icon is selected, a message will appear indicating that save data will be transferred (Figure 6). An explanation of the feature will be displayed by the system, as well as the text set in the param.sfo IMPORT_SAVEDATA_INFO. If IMPORT_SAVEDATA_INFO_* is set, the string of the language corresponding to the current system language setting will be displayed.

Figure 6 Save Data Transfer Content Notification Screen



If **OK** is selected, the system will look in the PlayStation®Vita card for the save data specified in INSTALL_DIR_SAVEDATA_ADD_* in param.sfo. At this time, save data on the memory card will not be a search target. If the save data is found, the message shown in Figure 7 will be displayed. If it is not found, the message shown in Figure 8 will be displayed. Selecting **OK** on the screen shown in Figure 8 will cause the screen to return to the LiveArea™.

Figure 7 Save Data Import Confirmation Screen

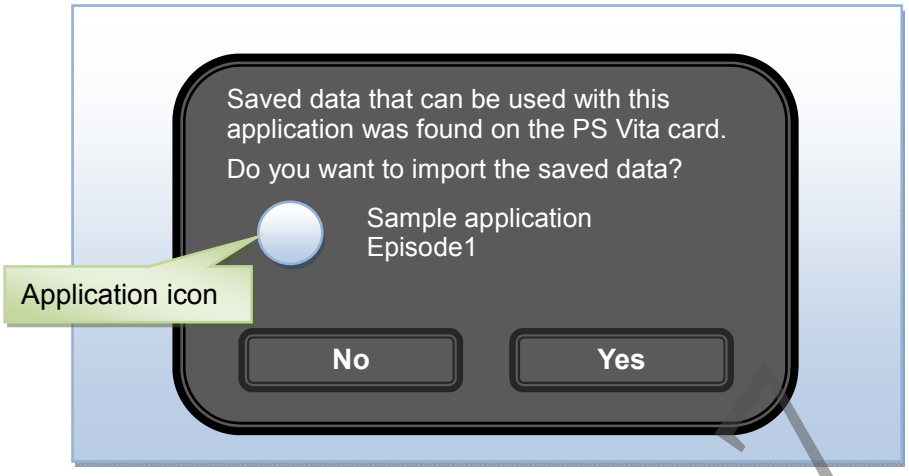
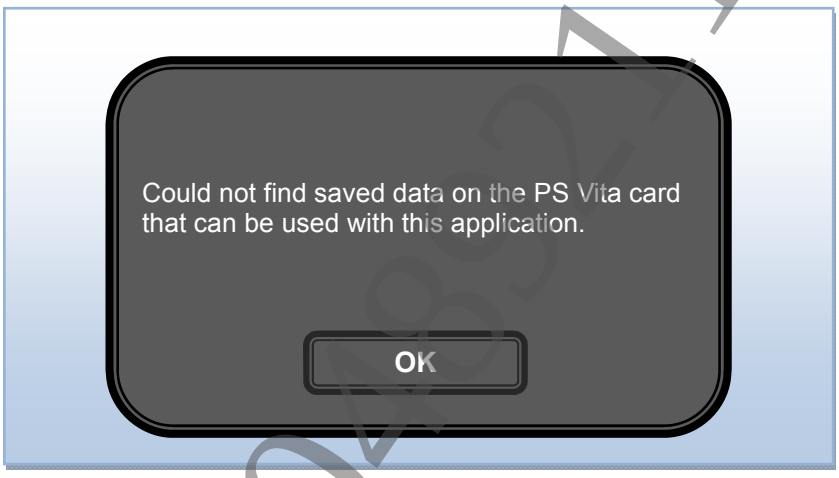
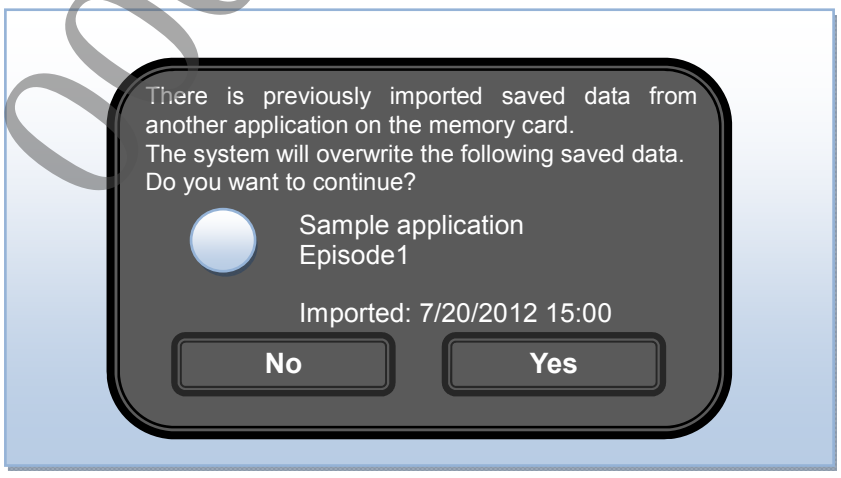


Figure 8 Screen Shown when Save Data Is not Found



When **Yes** is selected on the screen shown in Figure 7, and save data has previously been imported, a save data overwrite confirmation screen (Figure 9) will be displayed. There is only one save area when save data is imported to a memory card from the PlayStation®Vita card. It is completely overwritten each time save data is imported.

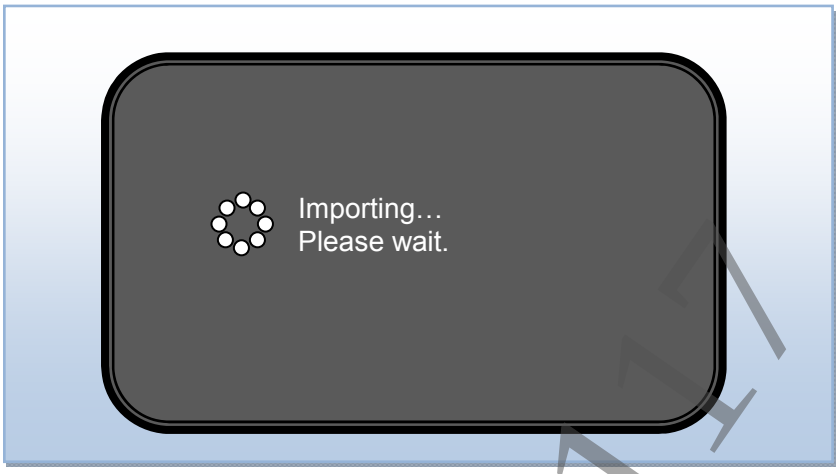
Figure 9 Transfer Save Data Overwrite Confirmation Screen



After the user has confirmed that they wish to import, the screen in Figure 10 is displayed, and save data is imported from the PlayStation®Vita card to the memory card.

```
grw0:savedata/  ->  ux0:user/<user id>/savedata_vc/
```

Figure 10 Importing Screen



The save data importing process will then be completed, and the screen will return to the LiveArea™.

Save data located on the memory card as follows can be read from the application without having to carry out a save data import.

```
ux0:user/<user id>/savedata/
```

Mounting Save Data

The next step in transferring save data is for the running application to mount save data of a different title, that is located on the memory card. Call the application utility library save data mount function, specifying the product code and passcode of the save data you wish to mount.

Note
The save data's product code specified here must be set with INSTALL_DIR_SAVEDATA_ADD_* of the application's param.sfo.

```
/* Mount savedata1: */
SceAppUtilTitleId titleId;
SceAppUtilPassCode passCode;

memset(&titleId, 0x0, sizeof(titleId));
memset(&passCode, 0x0, sizeof(passCode));

strncpy( titleId.data, "ABCD00001", sizeof(titleId)-1 );
strncpy( passCode.data, "passcodepasscodepasscodepasscode",
sizeof(passCode));

ret = sceAppUtilSaveDataMount( &titleId, &passCode );
if( ret == SCE_ERROR_ERRNO_ENOENT ) {
        /* The save data to be carried over does not exist on the memory card
*/
}
else if( ret < 0 ) {
        /* Mounting fails */
}
```

The mount function searches for save data on the memory card in the following order.

(1) Save data imported from the PlayStation®Vita card
`ux0:user/<user id>/savedata_vc/<title id>/`

(2) Save data from VC-MC or MC-MC configuration applications
`ux0:user/<user id>/savedata/<title id>/`

If the save data directory with the specified product code exists, the passcodes will be compared. If the passcodes are identical, the save data directory will be mounted. The mounted save data directory can be accessed by specifying the savedata1: drive.

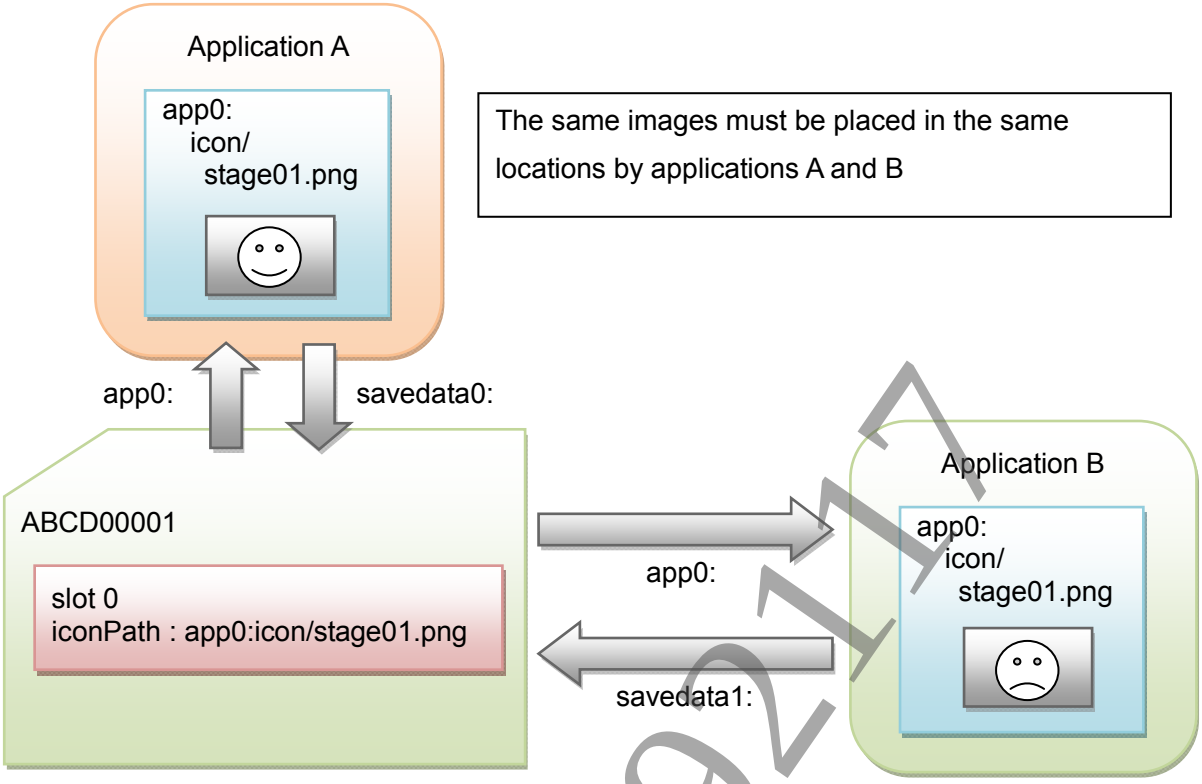
Save Data Dialog Transfer Data Display

To display the save data slot which exists in the mounted savedata1: in Save Data Dialog, specify savedata1: with the `SceSaveDataDialogParam` structure *`slotConfParam`*.

```
/* Settings for displaying the save data slot on savedata1: */
SceSaveDataDialogParam param;
SceSaveDataDialogSlotConfigParam slotConfParam;
SceAppUtilMountPoint mountPoint;

memset(&param, 0x0, sizeof(param));
memset(&slotConfParam, 0x0, sizeof(slotConfParam));
memset(&mountPoint, 0x0, sizeof(mountPoint));

strncpy( mountPoint.data, "savedata1:", sizeof(mountPoint)-1 );

slotConfParam.mountPoint = &mountPoint;
param.slotConfParam = &slotConfParam;
```

The save data slot thumbnail path is stored as the `SceAppUtilSaveDataSlotParam` structure *`iconPath`* member, but special care needs to be given when a path on app0: is written here. The original application's app0: and the app0: of the application which is now mounting the save data are at different locations, so unless images are placed in exactly the same location as the original application, the save data slot thumbnail will not be displayed correctly.

Figure 11 Reference to Thumbnail Path Specified on app0:

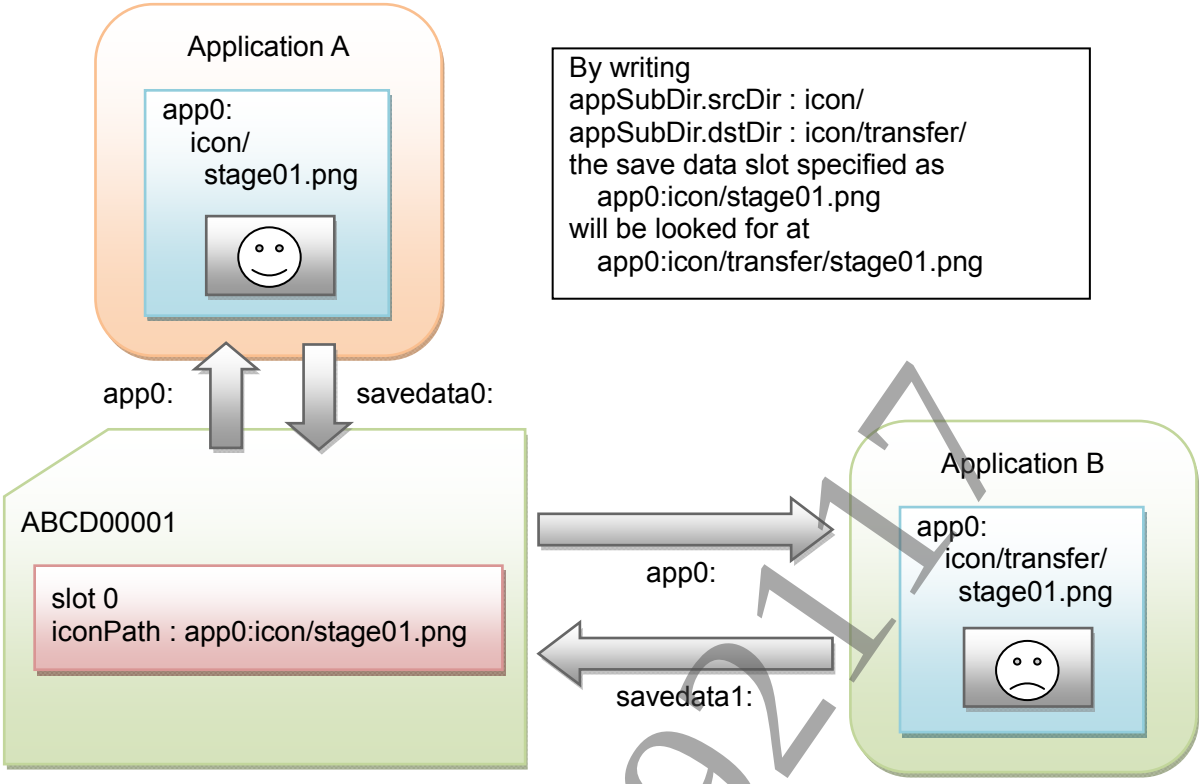


This severely limits the app0: data configuration for the application carrying over the save data. In order to avoid this, a thumbnail path subdirectory can be set.

```
/* Specify thumbnail subdirectory */
SceSaveDataDialogAppSubDirParam appSubDir;

memset(&appSubDir, 0x0, sizeof(appSubDir));
strncpy( appSubDir.srcDir, "icon/", sizeof(appSubDir.srcDir)-1 );
strncpy( appSubDir.dstDir, "transfer/", sizeof(appSubDir.dstDir)-1 );

slotConfParam.appSubDir = &appSubDir;
```

By doing this, the application carrying over the save data can place the thumbnail data of the original application wherever it would like in its own app0:.

Figure 12 Reference to Thumbnail Path when Specifying Subdirectory



Reading Transferred Data

Access to the save data slots on savedata1: are read-only, with no writing. Specify the savedata1: mount point as follows.

```
/* Access save data slots on savedata1: */
SceAppUtilMountPoint mountPoint;
memset(&mountPoint, 0x0, sizeof(mountPoint));
strncpy( mountPoint.data, "savedata1:", sizeof(mountPoint)-1 );

SceAppUtilSaveDataSlotId slotId = 0;
SceAppUtilSaveDataSlotParam slotParam;
memset(&slotParam, 0x0, sizeof(slotParam));

ret = sceAppUtilSaveDataSlotGetParam( slotId, &slotParam, &mountPoint );
```

Because savedata1: cannot be written to, operations such as `sceAppUtilSaveDataDataSave()` and `sceAppUtilSaveDataDataRemove()` cannot be used.

Save data files can be read by specifying a savedata1: path with the IO/File manager functions, but note that files cannot be opened in read/write mode.

Unmounting Save Data

When the reading of the save data has been completed, unmounts the mounted save data.

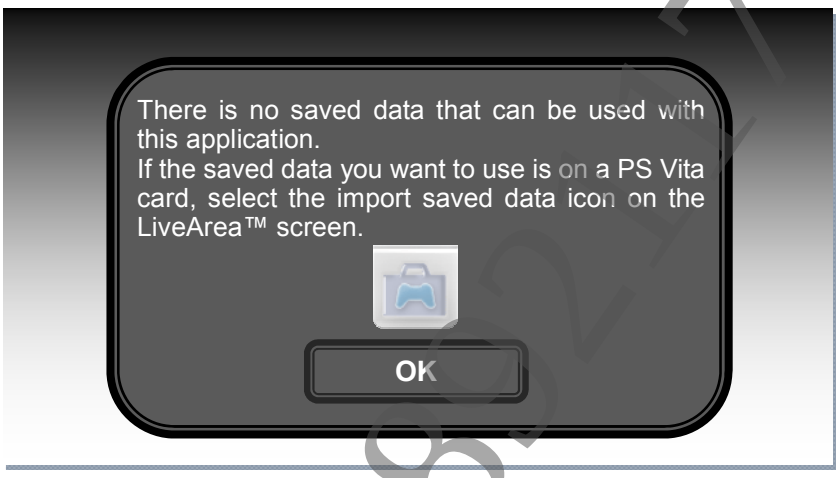
```
/* Unmount savedata1: */
ret = sceAppUtilSaveDataUmount();
```

Informing Users about Save Data Importing

When save data mounting fails with `SCE_ERROR_ERRNO_ENOENT` (no mountable save data can be found) because the save data to be loaded is on a PlayStation®Vita card, specify `SCE_SAVEDATA_DIALOG_SYSMSG_TYPE_NODATA_IMPORT` to the system defined message display feature of Save Data Dialog as necessary and prompt the user to import save data to the memory card.

Note
Only perform notifications with this system defined message if there is a possibility that the save data will be transferred from a VC-VC-configured application.
Flow when save data is not found is free if only transferring save data from VC-MC/MC-MC-configured applications.

Figure 13 Display of System Defined Message Informing User about Save Data Importing



Notes Upon Loading Save Data of Another User

Save data to be loaded by save data sharing and save data transferring may be save data of another user in the following cases.

- Save data of a VC-VC-configured application (only for save data transferring)
- Save data of a VC-MC/MC-MC-configured application that was played without signing up

Uploading to a Server

When uploading data to a title user storage (TUS) or unique server, take measures to ensure that another user's data is not uploaded; for example, store the user's account ID in the save data and have the application check that the save data owner is the same user at the transfer source as the transfer destination.

Unlocking Trophies

It is forbidden to unlock a trophy based on the content of save data immediately after it is loaded. The unlocking of a trophies must be based on the result of a user's gameplay and carried out at each appropriate unlock timing.

Example:

Even when save data of a game cleared up to stage 3 is transferred, do not enable the unlocking of trophies for clearing stage 1, stage 2, and stage 3. When the user next clears the applicable stage, re-evaluate the unlocking of the trophy for that stage. However, it is permissible to enable the unlocking of a trophy for clearing all stages at the timing when stage 5 is cleared after save data of a game with stage 1 to 4 cleared has been transferred.

As long as this rule is adhered to for PlayStation®Vita applications, it is not a problem even if save data that could be of another user is loaded by save data sharing or save data transferring and trophies are unlocked by gameplay based on that loaded data.

12 Handling of Save Data for PSP™

This chapter explains how applications handle save data for PSP™.

> **Note**
> Save data for PSP™ is in a state where it can be modified by third parties. Be aware of this when an application for PlayStation®Vita loads save data for PSP™; note that applications may hang depending on the content of the loaded data, and be careful to not perform operations that may cause security holes.

Allowed Processing

Save data for PSP™ can only be displayed and loaded. New save data cannot be created and old save data cannot be updated/overwritten. The Save Data Dialog library is used for display, and the application utility library is used for loading. The relations with these libraries are the same as when handling save data for PlayStation®Vita.

Display

One major difference with the display of save data for PlayStation®Vita is that save data for PSP™ is handled at the save data directory level, while save data for PlayStation®Vita is handled at the save data slot level. The Save Data Dialog library provides a set of structures and APIs for displaying save data for PSP™ by specifying directory names, but the feature is almost exactly the same as the structures and APIs for handling save data for PlayStation®Vita.

For details on the display methods, refer to the "Save Data Dialog Overview" and "Save Data Dialog Reference" documents.

Loading Methods

Save data for PlayStation®Vita can be directly loaded using file system APIs, but save data for PSP™ is loaded using exclusive APIs. The application utility library provides the following two features.

- `sceAppUtilPspSaveDataGetDirNameList()`
  Obtain the list of directory names of the save data for PSP™ in a memory card
- `sceAppUtilPspSaveDataLoad()`
  Feature for loading a file saved in a directory for save data for PSP™ in a memory card

Only a loading method that loads all of the files at once is provided. This is the same as the specifications for the original save data utility library for PSP™.

For PlayStation®Vita, save data for PSP™ is saved to a memory card with the following methods.

- Using the Content Manager application to copy save data for PSP™ saved in a PC or PlayStation®3
- Playing PSP™ Game on a PlayStation®Vita

For details on the loading methods, refer to the "Application Utility Overview" and "Application Utility Reference" documents.

Conditions and Restrictions

The following conditions and restrictions apply to loading save data for PSP™ with an application for PlayStation®Vita.

- The save data format version must be explicitly specified
- The save data file must be explicitly specified as a protected file or an unprotected data file
- When loading a protected data file, the protected data file ID must be known (a protected data file ID is a code phrase specified by the save data utility when a PSP™ Game saves the protected data file)
- Memory for loading all of the save data files at once must be prepared
- A list of files saved in the save data directory and their sizes cannot be obtained; only loading by filename specification is possible regardless of file absence/presence
- The maximum size of save data files that can be loaded is restricted to 2 MiB

13 Reference Information

Save Data Format

Regarding save data format created by using the save data feature, refer to the "Save Data" chapter in the "Application Development Process Overview" document.

Minimum Save Data Size Required to Run the Application

"Reserved system size for save data quota" described in the "Save Data Free Space" chapter must be included in "Minimum save data size required to run the application (also to be written on a package)" that is declared at the time of application submission. However, it is not necessary to include "system margin for file system free space".

Specifically, display ★ **Save Data Quota xxxx KB / yyyy KB** from the application icon on the Home screen after executing a series of applications, and then declare the value xxxx.

For details on how to display ★ **Save Data Quota**, refer to the "System Software Overview" document.