Application Development Process Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

1 Application Development Process Overview

This document describes the following contents, which are necessary for application development using SDK.

- About application package configuration
- About application configuration
- How to access files included in the main program files of an application
- How to access a patch
- How to access save data
- How to access additional contents
- How to use a memory card
- How to install additional contents within an application
- How to upgrade to the full version within an upgradable application
- How to access game data and save data from a development host computer
- How to load save data created with a product version application on a DevKit/TestKit
- How to create and install application package files
- About the feature for importing and exporting application game data and other data
- Troubleshooting for issues encountered during development

2 Application Package Configuration

Below is an explanation on how to place an application's programs and data, and arrange them into files.

What Are Packages?

A package is the file format (pkg file) used when submitting an application to SCE. A package created by an application developer must be submitted as a master to SCE in preparation for the application to be sold.

There are two types of selling for applications: the "PlayStation®Vita card version", with the PlayStation®Vita card as the media and intended mainly for store sales; and the "PlayStation®Store digital version", intended for download sales whereby data is distributed from PlayStation®Store and saved on memory cards. It is possible to create both the PlayStation®Vita card version and the PlayStation®Store digital version from a single package.

Packages are created by using Package Generator included in Publishing Tools. The packages created can be installed on the Development Kit (DevKit) or the Testing Kit (TestKit) to perform operation testing. For details on how to create packages, refer to the "Package Generator User's Guide" document included in the Publishing Tools.

What is Package Configuration?

Applications are composed by the following four types of elements:

- Game (App)
- Save data (SD)
- Additional contents (AC)
- Patches (Patch)

As the storage destination of each element, PlayStation®Vita provides two storage media - the PlayStation®Vita card and memory card. It is possible to select either of these media for storing a game; however, save data, additional contents, and patches can only be stored on a memory card.

Package configuration refers to the combination by which the above elements are configured on each media. How the application will be sold is determined by the storage media configuration; during development, consider package configuration pursuant to the way the application is to be sold.

Types of Package Configuration

The following two types of package configuration are available;

- VC-MC configuration
  "VC-MC (VC-MC/MC-MC or VC-MC/No MC)" on Publishing Tools
- No VC/MC-MC configuration
  "No VC/MC-MC" on Publishing Tools

VC stands for PlayStation®Vita card, while MC stands for memory card.

Assuming that configuration is indicated as "XX-YY", the XX part represents the media storing the game, while YY represents the media storing save data, patches and additional contents.

Furthermore, when VC-MC/MC-MC is indicated, VC-MC represents the PlayStation®Vita card version that is to be sold in retail stores and MC-MC represents the PlayStation®Store digital version for which data will be distributed and saved on a memory card.

Note
Although a package with VC-VC configuration exists with the game, save data, additional contents, and patches all stored on a PlayStation®Vita card, currently master submission of a new application with this type of VC-VC package configuration is prohibited (refer to TRC (Technical Requirements Checklist) [R3167]).

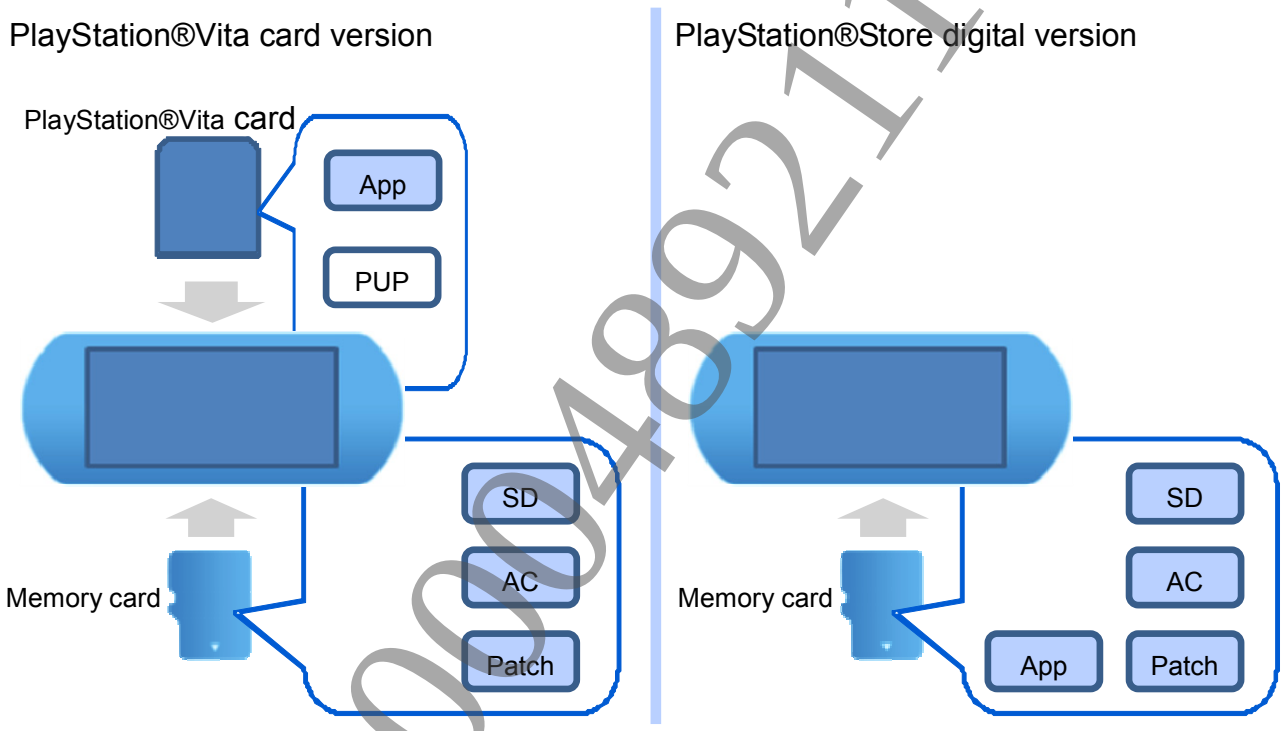
As evident from the above two types of package configurations, save data, additional contents, and patches are always stored on a memory card in either configuration. Thus, there is no package configuration that does not require a memory card.

How an Application Is Sold

Applications can be sold in two forms.

- PlayStation®Vita card version (VC-MC configuration)
- PlayStation®Store digital version (VC-MC configuration or No VC/MC-MC configuration)

Figure 1 Data Storing Media



PlayStation®Vita Card Version

In PlayStation®Vita card versions, the game is stored on the PlayStation®Vita card, and save data, additional contents, and patches are stored on the memory card. This version can only be created when VC-MC configuration is selected upon creating the package.

The size of the game must be decided so that the application can store it in a 2 GB or 4 GB PlayStation®Vita card. The maximum size of the game must be as follows.

2 GB PlayStation®Vita Card	4 GB PlayStation®Vita Card
1649 MiB	3441 MiB

Note
The maximum size of the game must be 1649 MiB (or 3441 MiB) as indicated above. **PS Vita card R/O Area** of the system software's application ★**Check** feature displays the sum of this value and the system update file (PUP) size (143 MiB), which is 1792 MiB (or 3584 MiB). The PUP storage area is reserved and cannot be used by the game.
For the application ★**Check** feature, refer to the "System Software Overview" document.

PlayStation®Store Digital Version

In PlayStation®Store digital versions, all configuration elements including the game are stored on the memory card. This version can be created when either VC-MC configuration or No VC/MC-MC configuration is selected upon creating the package.

Regarding a package created with the VC-MC configuration, the same restriction as a PlayStation®Vita card version applies for the capacity of the game.

Selecting the Package Configuration

Package configuration for applications can be chosen among the following three types:

Package Configuration	Maximum Size of Game	Notes
VC-MC configuration (2 GB PlayStation®Vita card)	1649 MiB	
VC-MC configuration (4 GB PlayStation®Vita card)	3441 MiB	
No VC/MC-MC configuration	6912 MiB	VC version cannot be created; only for digital version

Decide package configuration taking into account the following elements:

- Choose the No VC/MC-MC configuration only in cases where you wish to set a size that is larger than 3441 MiB for the game. If the size of the game is 3441 MiB or less, consider selecting the VC-MC configuration instead of the No VC/MC-MC configuration. In the case of VC-MC configuration, in the future it will be possible to release a PlayStation®Vita card version with the same package even if you do not plan to release a PlayStation®Vita card version at present. Moreover, even when selecting No VC/MC-MC configuration, keep the size of the game within 6912 MiB.
- In either configuration, a memory card is required to run an application.
- In either configuration, save data, additional content and patch data are stored on the memory card. Available space on the memory card varies according to the total size of the memory card and usage by the user. Note that the more space these elements require, there will be more users who will be unable to play the application.
- If you wish to create a package file whose save data exceeds 1 GB, inquire with Private Support through the PlayStation®Vita Developer Network website (https://psvita.scedev.net/) (refer to TRC [R3075]).
- In either configuration, users can back up the application and its save data, additional content and patch data on the PC or PlayStation®3.

Note
Regardless of the package configuration and how the application is sold, the maximum size for the application package, additional content package, and patch package is 6912 MiB each. A package larger than 6912 MiB cannot be master submitted.
Moreover, the maximum size of a file that can be included in a package is, for each file, 4 GiB minus 1 byte (4,294,967,295 bytes).

Conditions for Creating a PlayStation®Vita Card

Demo/Trial version applications and free full version applications with the DRM Type as Free cannot be created as PlayStation®Vita cards and distributed.

For an upgradable application that can be switched from a trial version to a full version, the created PlayStation®Vita card will run as a full version. For this as well, the PlayStation®Vita card cannot be created as a trial version.

To create a PlayStation®Vita card, make setting with Publishing Tools as follows.

- Setting with Param File Editor
  - Upgradable attribute of param.sfo: On
- Setting with Package Generator
  - DRM Type: Local
  - Storage Type: VC-MC (VC-MC/MC-MC or VC-MC/No MC)

For usage of each tool, refer to the "Param File Editor User's Guide" and "Package Generator User's Guide" documents.

Data Placement That Causes a Decrease in File Access Speed

Upon creating a package, make sure the total number of files and directories stored directly under one directory does not exceed 100. Given the characteristic of exFAT, file access speed will drastically deteriorate when the number of files and directories directly under one directory increases and the directory entry size increases.

A package with a data configuration that causes file access speed deterioration not only affects accesses to applicable data during gameplay, it also has a negative impact on the download or install speed of the applicable package on the system software. Make sure to place data in such a way to avoid file access speed deterioration.

In the example below, a drastic deterioration in file access speed occurs because a total of 200 files and directories are stored under one directory.

```
app0:dir000/
     dir001/
      :
     dir099/
     file000.dat
     file001.dat
      :
     file099.dat
```

When including a large number of files in a package, place data so that no more than 100 files and directories are directly under one directory and avoid file access speed deterioration.

```
app0:dir000/file000.dat
            file001.dat
             :
            file099.dat
     dir001/file100.dat
      :     file101.dat
      :      :
      :     file199.dat
      :
     dir009/file900.dat
            file901.dat
             :
            file999.dat
```

3 Mount Points

The mount points used in the DevKit are described below:

Mount Point	Description
ux0:	Root directory of the memory card
host0:	Directory on the HDD of the development host computer set as fsroot in Neighborhood for PlayStation®Vita (Neighborhood hereafter) and in the debugger
app0:	Game directory
addcont0:	Additional content root directory that is automatically mounted when the application starts up based on a param.sfo setting
addcont1:	Another title's additional content root directory that is mounted with the additional content mount API from inside the application
savedata0:	Save data directory that is automatically mounted when the application starts up based on a param.sfo setting
savedata1:	Another title's save data directory that is mounted with the save data mount API from inside the application
photo0:	Photo data directory that is mountable/unmountable by calling `sceAppUtilPhotoMount()/sceAppUtilPhotoUmount()`
music0:	Music data directory that is mountable/unmountable by calling `sceAppUtilMusicMount()/sceAppUtilMusicUmount()`
video0:	Video data directory that is mountable/unmountable by calling `sceAppUtilExtVideoMount()/sceAppUtilExtVideoUmount()`

4 Application Configuration

Contents that constitute an application are shown below.

Content	Access
Game	read only (a system enabling write is available during development)
Patches	read only
Save data	read/write
Additional contents	read only

Details on each content are described in the following chapters.

5 Game

Files that constitute an application itself are included in the game directory. The files, for example, are the program file, which is directly launched from the system and data files.

Files created by an application after running are not included in this directory. It simply shows the first state of the file configuration after installing the files on PlayStation®Vita (through a network, etc.).

All files of the game are read-only from the application. The application cannot create or update a file in any directory within this game directory.

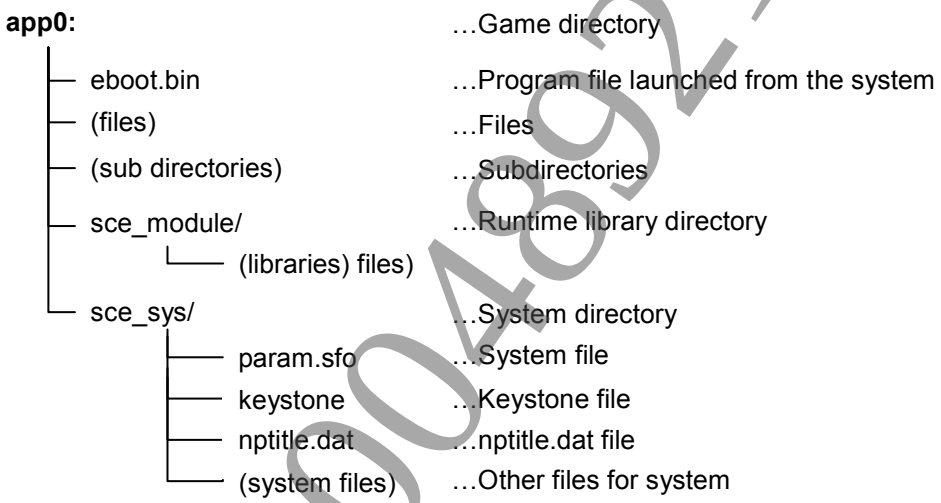
File Configuration

Ultimately, the game directory will be "read only" from the application, but a means to write will be provided for development.

This directory can be accessed from the application via the app0: mount point.

File configuration of the game is indicated in Figure 2.

Figure 2 File Configuration of Game



Each directory and file is explained below.

Game Directory (app0:)

This is the top of the game directory. This directory is used for saving the program file launched from the system software (eboot.bin), other program files and various types of data files. In this directory, as many subdirectories and files as necessary can be created, within a certain limit described later.

eboot.bin

This is the program file which is executed first when the system starting up an application. It is not necessary when starting up self files from Neighborhood or the debugger. However, when starting up an application from ★APP_HOME of the system software, this file will be necessary for a given directory. For details, refer to the "System Software Overview" document.

sce_module/

Here are placed the preload module and the runtime library loaded with the libsysmodule library.

> **Note**
> For details on the runtime library directory, refer to the "libsysmodule Overview" document.

sce_sys/

This is the directory where all system files are placed. The materials of LiveArea™ are also placed here. This directory is used by the system, and the application does not have the right to access it.

This directory can only be accessed by the application when the application is started from the debugger or from ★APP_HOME during development. However, note that a developer must not carry out implementation that entails the application to access this directory.

> **Note**
> For details on the necessary file configuration for LiveArea™, refer to the "LiveArea™ Specifications" document.

System File (param.sfo)

The parameters required for the game are set here. Refer to the "Param File Editor User's Guide" document contained in the Publishing Tools on how to create this. If the format of the file is invalid, a system file format error (error code: C0-11479-2/0x80800015) will occur when the application starts up.

Keystone File (keystone)

The passcode set to the game is saved here. This file is automatically placed upon creating the package with Publishing Tools.

nptitle.dat File (nptitle.dat)

This is a file used by the NpWebApi library. For details, refer to the "NpWebApi Library Overview" document.

Subdirectory and Filename Restrictions

Subdirectories that can be created within the game directory are limited to seven levels. The following is an example where a file is created in the deepest-possible level.

- app0:1/2/3/4/5/6/7/8.dat

Directories and files with the following names cannot be created immediately below the game directory.

- eboot*
- sce_*

The names of the directories or files that can be created within the game directory are limited to up to 63 bytes, while path length is limited to up to 199 bytes, excluding the mount point part. The supported character code is UTF-8, without distinction between upper and lower case letters.

How to Install the Game

Other than starting up the game on a debugger or starting up the game from ★APP_HOME of the system software, create a package and install it to start the application from the application icon on the home screen.

By installing the application package, you can run the application in an environment that is closer to the product version. Refer to the "Appendix 1: How to Install Packages" chapter on creating and installing packages.

How to Access the Game Directory

The mount point of the game directory app0: is decided as follows:

(1) If the file path setting file has been specified in Neighborhood or the debugger, the directory specified in the file path setting file will be mounted to app0:.

(2) If the working directory has been specified in Neighborhood or the debugger but there is no file path setting file or app0: has not been specified in the file path setting file, that directory will be mounted to app0:.

(3) If the working directory has not been specified, the directory where the started-up self file is placed is mounted to app0:. If the application has been installed as a package and started, the root directory of the package will be mounted as app0:.

However, the root directory of a drive on the development host computer cannot be mounted as app0:. Note that if the directory is determined in the above procedure and it is a root directory, application startup will fail.

Note
Regarding the file path setting file, refer to the "File Path Setting Guide" document."
Concerning the working directory, refer to the Neighborhood document "Neighborhood and Utilities User's Guide".
For the program start-up method by specifying the file path setting file and working directory from Neighborhood or the debugger, refer to the "Neighborhood and Utilities User's Guide" and "Debugger User's Guide" documents.

app0: is always mounted automatically when the application is started up. It cannot be unmounted. As a file access method, you can, for example, specify the path as described below to access eboot.bin from an application.

```
app0:eboot.bin
```

The file access APIs can be used to read files.

Differences in Behavior for Each Game Directory

Depending on the directory assigned to the game drive (app0:), behavior presents the differences described below. Ensure that operation gets close to normal in accordance with the stage of development of the application.

Directory on host0:

- Encryption: no
- Speed emulation: no
- Read/write enabled

Directory on ux0:data

- Encryption: no
- Speed emulation: no
- Read only

Directory Installed Using "★Package Installer"

- Encryption: no
- Speed emulation: yes
- Read only

Normal Location of PlayStation®Vita Card and Memory Card

- Encryption: yes
- Read only

Note
With regard to "★Package Installer", refer to the "Appendix 1: How to Install Packages" chapter.
The procedure for using ux0:data/ will be described later.
The normal location of PlayStation®Vita cards and memory cards cannot be chosen during development.

About the Game Directory of Sample Program

SDK sample programs perform file access using app0:.

Directory specified in app0: by an SDK sample program is set in the project file for Visual Studio on the development host computer .

In Visual Studio, it is set in Visual Studio project file's **Property -> Properties**' **Configuration -> Debugging -> Working Directory**.

Assignment of the Enter Button when Calling Common Dialog

The assignment of the enter button on Common Dialog called from within an application follows the specification in the game directory's system file

```
app0:sce_sys/param.sfo
```

This system file may sometimes not be in that place when starting up from the debugger or from ★ APP_HOME; in such cases, assignment will follow settings on the PlayStation®Vita.

Note
Refer to the "Programming Startup Guide" document for more information on how the enter button is assigned.
Refer to the "Param File Editor User's Guide" document in the Publishing Tools on how to create param.sfo.

Accessing Game Data from the Development Host Computer

You can access the game data of applications installed on a memory card from the development host computer. For details, refer to the "How to Access Game Data and Save Data from the Development Host Computer" chapter.

Upgradable Applications

Upgradable applications allow switching from the trial version to the full version of the application with a single package. Switching is performed based on right status, thus implying sale on PlayStation®Store. If an upgradable application is sold as a PlayStation®Vita card, it will always run as the full version due to the inclusion of the rights for the full-version of the application.

When implementing an application, create it so that it calls the `sceAppUtilAppParamGetInt()` function at application start-up, checks whether the application functions as the trial version or as the full version during runtime, and changes the application's operation accordingly.

When application development is complete, create a package by using the Package Generator. When creating an upgradable application package, you will need to perform the following operations in addition to the normal package creation process:

- Additional setting for the param file (param.sfo)
  Select the Upgradable checkbox in Param File Editor when creating the package.
- Addition of a directory for the full version of the application
  Create a directory named retail/livearea/ under root directory/sce_sys/, and add the LiveArea™ data used when the application runs as the full version.

After creating the package, perform PlayStation®Store settings. Upload the package on the Drop Point of the Network Platform Management Tool (NPMT). Then, create 2 SKUs with the NPMT, and match the packages (treated as DRM contents on NPMT) with their respective SKUs.

Moreover, you can specify an Entitlement Flag for each SKU. Specify Trial in the Entitlement Flag of the SKU you wish to set as the trial version, and Full in the Entitlement Flag of the SKU you wish to set as full version.

With this setting, if a user purchases the SKU whose Entitlement Flag is set to Trial, the rights file (generated automatically by the system) of the trial version and the package will be downloaded. Likewise, if a user purchases the SKU whose Entitlement Flag is set to Full, the rights file of the full version will be downloaded. If the package has already been downloaded when purchasing the trial version, it will not be downloaded again when purchasing the full version.

For details on the tools and libraries that are necessary to create an upgradable application, refer to the documents below:

- PlayStation®Store Content Guidelines for PlayStation®Vita
  The "(Basic) Free Trial with Restriction Unlock Feature + Paid-for Full Game Dual Package" chapter provides an overview and an explanation of the main points concerning the structure of upgradable applications.
- Package Generator User's Guide and Param File Editor User's Guide
  To create the package of an upgradable application, you need to use the Package Generator and Param File Editor. These documents provide a detailed explanation of the settings to be specified in each tool.
- System Software Overview
  During development only, the **Upgradable App Debug** function is available on the **PSN**℠ menu below ★**Debug Settings**, enabling testing by switching between trial version and full version rights without using the NPMT. This document explains how to use the **Upgradable App Debug** function.
- Application Utility Reference
  Describes the APIs required to determine switching between the trial version and the full version within the application.
- NP Product Management Guide
  Explains in detail the various settings on NPMT.

6 Patches (Game Updates)

The directory configuration of patches (game updates) is exactly the same as that of a game. However, in some cases it is acceptable not to include a file in a patch that is required for the game.

As in a game, patch files are all read only. Applications cannot create or update files in this directory.

For details, refer to the "Patch Overview" document.

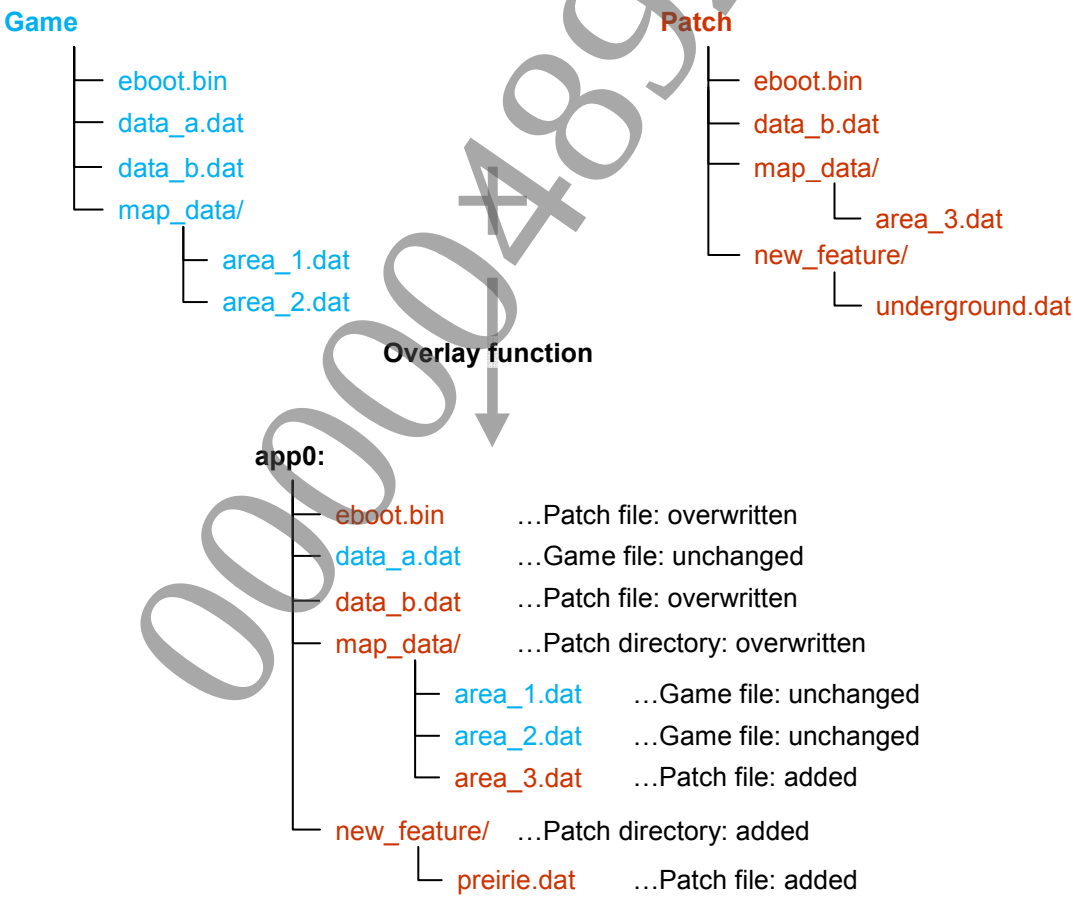
Overlaying Game and Patches

Patches are created in a separate directory from the game. Since they are created independently, it will not be possible for patch files to be actually overwritten on files in the game directory. However, when access to a game file is attempted from the application, if the file in question is intended to be overwritten by the patch, the file in the patch directory will be used. This is called the "patch overlay feature".

With the overlay feature, from the application all the files contained in the patch directory will appear to have been overwritten on the game. Like the game, applications can access the files with the mount point app0:, and there is no need for patch programs to distinguish between game and patch files (conversely, files that appear to have been overwritten cannot be distinguished).

Also, with the overlay feature it is not possible to make files appear to have been deleted.

Figure 3 Patch Overlay Feature



When a New Patch Is Installed Over a Patch

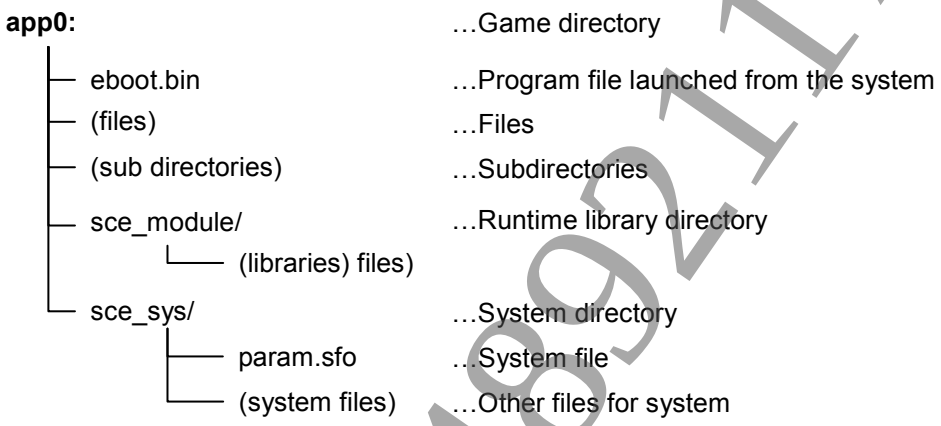
When a further patch is installed in a game that already has a patch, the game directory and the patch directory will always remain in a 1:1 ratio, without creating multiple patch directories. The space used up on the media of installation will remain unchanged when installing a cumulative patch package, or a hybrid patch package, also.

File Configuration

The patch directory can only be accessed from the application as read-only. Files and directories placed in this directory can be accessed from the application via the app0: mount point using the patch's overlay feature.

The file configuration of a patch is indicated in Figure 4.

Figure 4 File Configuration of Patch



Basically, each directory and file is the same as those of the game. However, the following points differ from files of the game.

eboot.bin

This file is required for a game. However, it is acceptable not to include it in the patch directory.

System File (param.sfo)

The parameters that need to be set are different from those for a game. Concerning setting, refer to the "Param File Editor User's Guide" document contained in the Publishing Tools.

How to Install a Patch

In order to install patches, the game must be installed first. Refer to the "Appendix 1: How to Install Packages" chapter on creating and installing packages.

How to Access the Patch Directory

Access using the app0: mount point, like for the game directory.

app0: is always mounted automatically when the application is started up. It cannot be unmounted. As a file access method, you can, for example, specify the path as described below to access eboot.bin from an application.

```
app0:eboot.bin
```

The file access APIs can be used to read files.

7 Save Data

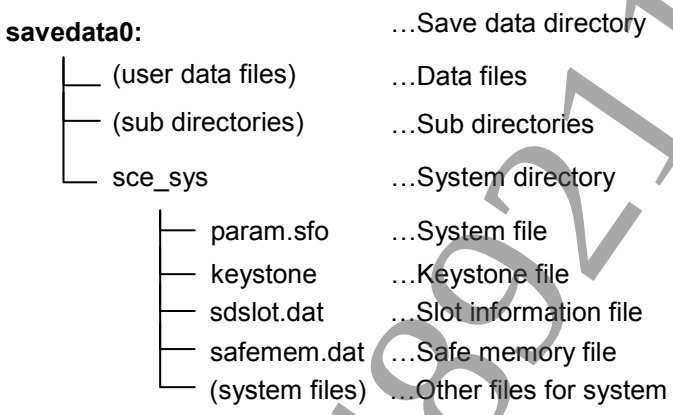
Save data includes data for indicating a game play progress, high score information, user's character edits and setting data such as volume setting for sound effect , button preferences, and message display speed setting. Save data is actually a directory where the files containing those data are stored. For PlayStation®Vita applications, the files can be saved only in the save data directory.

File Configuration

The save data directory can be accessed from the application via the savedata0: mount point.

The file configuration of save data is indicated in Figure 5.

Figure 5 File Configuration of Save Data



Each directory and file is explained below.

Save Data Directory

This directory contains all the files that make up save data. Arbitrary data files and subdirectories can be created in this directory, within the scope of restrictions described later.

Data Files and Subdirectories

These are data files and subdirectories created by an application. Data files are protected by means of an encrypted signature method provided by the save data feature.

Even if save data were to be copied to an external recording device, the content of this file cannot be read on a PC, etc. Moreover, falsification can be detected by the save data feature if the data is falsified.

Only the save data feature can update data files.

sce_sys/

All system files are places in this directory. This directory is used by the system and the application has no right to access it.

The application can only access this directory when save data is created on the development host computer during development. However, a developer must not carry out implementation that entails the application to access this directory.

System File (param.sfo)

This file is automatically created by the save data feature and contains parameters such as a title name and detailed information.

Keystone File (keystone)

The passcode set to the game of the application for which save data is created is saved here. The save data feature creates it automatically. If the passcode set to the application for which save data is to be mounted differs from the save data's passcode, a passcode error (error code: C0-11138-4/0x80800006) will occur and the application will fail to start up.

Slot Information File (sdslot.dat)

The information of the save data slots handled with the application utility library and with Save Data Dialog is saved here. Use the application utility library's functions for save data slot operation to create/manage slot information files.

Safe Memory File (safemem.dat)

The contents of the safe memory handled by the application utility library are saved here. The application utility library creates it automatically. During development, it is possible to initialize safe memory status at application start-up by deleting this file. During application runtime, it is renamed as _safemem.dat.

Subdirectory and File Name Restriction

Subdirectories that can be created within the save data directory are limited to three levels. The following is an example where a file is created in the deepest-possible level.

- savedata0:1/2/3/4.dat

Directories and files with the following name cannot be created immediately below the save data directory.

- sce_*

The names of the directories or files within the save data directory are limited to up to 63 bytes, while path length is limited to up to 199 bytes, excluding the mount point part. The supported character code is UTF-8, without distinction between upper and lower case letters.

How to Create Save Data

At the timing in which the application is first started up, one save data directory corresponding to that application will be created automatically. The application can freely create data files in this directory. It is also possible to create subdirectories, for example, to support to multiple save data.

The application can manage a single or multiple save data using the save data feature or Save Data Dialog. For details, refer to the "Save Data User's Guide" and "Save Data Dialog Overview" documents. It is also possible to implement a save data management feature using only the file access APIs of the save data feature, without using Save Data Dialog.

How to Access Save Data Directory

The save data directory mount point savedata0: is decided as follows.

(1) If the file path setting file has been specified in Neighborhood or the debugger, the directory specified in the file path setting file will be mounted to savedata0:.
However, savedata0: cannot be mounted below the directory mounted as app0:. At the same time, app0: cannot be mounted below the directory mounted as savedata0:.

Note
At this time, if a drive's root directory (C:\, D:\, etc.) on the development host computer is specified for savedata0:, the mounting of savedata0: will fail; therefore, make sure not to specify the root directory.

(2) If a directory named savedata in the root directory of host0: set as fsroot in Neighborhood or the debugger exists (that is, if host0:savedata/ is present), but there is no file path setting file or savedata0: has not been specified in the file path setting file, that directory will be mounted to savedata0:.

(3) If there is no host0:savedata/, ux0:data/savedata/ will be mounted to savedata0:.

(4) If there is no ux0:data/savedata/, savedata0: will not be mounted when starting up from the debugger or from ★APP_HOME.

(5) Only in the case of package start-up, the save data directory will be created automatically in the normal location shown below, and mounted as savedata0:.

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

This behavior varies based on whether **Release Check Mode** is set to **Development Mode** or to **Release Mode**.

	Development Mode / Release Mode		
	Debugger	**★APP_HOME**	**Package**
"savedata0=" in Mapping File	Valid/Invalid	Valid/Invalid	Invalid/Invalid
host0:savedata	Valid/Invalid	Valid/Invalid	Valid/Invalid
ux0:data/savedata	Valid/Invalid	Valid/Valid	Valid/Valid
ux0:user/<user id>/savedata/<title id>	Invalid/Invalid	Invalid/Invalid	Valid/Valid

Note
If ★**Debug Settings** -> **System** -> **TRC Check Notifications** is set to **On**, the save data's mount source will be displayed on the top left of the screen at application start-up, as follows:
When mounting is successful:
AppMgr:
savedata0: OK -
<Mount source path>

When mounting fails:
AppMgr:
savedata0: NG -
<Cause of the error>

If the mount source path is too long, its second half will be omitted.
For the <Cause of the error>, one of the followings is displayed:
- `SCE_ERROR_ERRNO_ENOENT` : the target directory does not exist.
- `SCE_ERROR_ERRNO_ENODEV` : the target drive does not exist.
- `0x8XXXXXX` : error code

The <user id> part is "00" in retail units; in the DevKit/TestKit, however, it will be the number of the account slot selected with the account selection function. If the memory card is inserted into another DevKit/TestKit, note that it might not be possible to read the save data due to the <user id> part being different from that of the original DevKit/TestKit. With regard to the account selection function, refer to the "PSN℠ - Account Selection" section in the "System Software Overview" document.

If INSTALL_DIR_SAVEDATA is set in the application's system file, the <title id> part will be its product code; if it is not set, the product code set to the application (TITLE_ID) will be applied.

Note
The method for accessing ux0:data will be described later.

savedata0: cannot be unmounted. As a file access method, you can, for example, specify the path as described below to access userdata.dat from an application program.

```
savedata0:userdata.dat
```

File access APIs can be used to read files. However, to write files, use the API for saving save data files that is provided by the save data feature.

Note
For details on the API for saving save data files, refer to the "Save Data User's Guide" and "Application Utility Reference" documents.

Differences in Behavior for Each Save Data Directory

Behavior differs depending on the directory assigned to the save data drive (savedata0:). Ensure that operation gets close to normal in accordance with the stage of development of the application.

host0:savedata

- Encryption: no
- Speed emulation: no
- Read/write enabled

ux0:data/savedata

- Encryption: no
- Speed emulation: yes
- Read enabled; use of exclusive APIs required for writing

ux0:user/<user id>/savedata/<title id> (Normal Location of Memory Card)

- Encryption: yes
- Read enabled; use of exclusive APIs required for writing

Feature for Reading Save Data From Another Title

By using the feature for reading the save data from another title, the save data directory of an arbitrary title can be mounted as "savedata1:". The savedata1: data structure is based on the savedata0: specifications, but writing cannot be performed for savedata1:. For details on the feature for reading the save data from another title, refer to the "Save Data User's Guide" document.

Save Data Sample Program

The save data sample program runs by using both the game directory mount point app0: and the save data directory savedata0:. In order to run the save data sample program correctly, ensure that the system can mount savedata0: according to the above-described "How to Access Save Data Directory" section.

Accessing Save Data from the Development Host Computer

You can access the save data of applications installed on a PlayStation®Vita card and memory card from the development host computer. For details, refer to the "How to Access Game Data and Save Data from the Development Host Computer" chapter.

8 Additional Contents

In additional contents, it is possible to include data files that can be accessed from the application. By downloading additional contents, it is possible to expand the data files that compose the application. Downloading and installation of additional contents are performed by the system software.

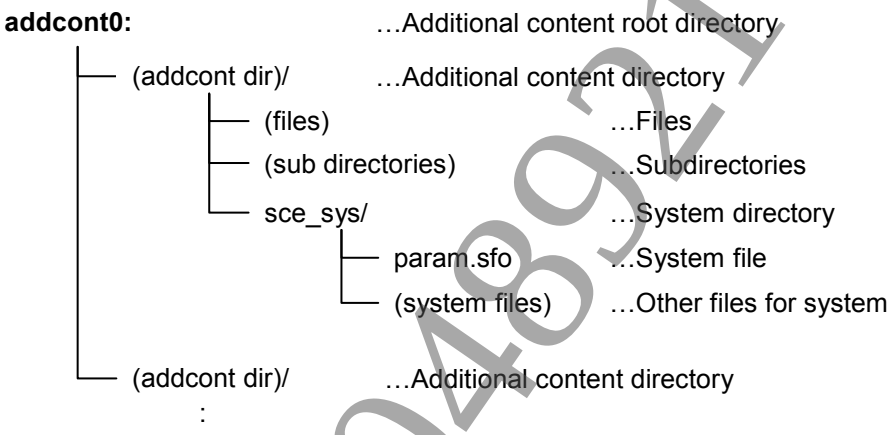
The additional content directory does not include files generated by the application after start-up. It simply shows file configuration immediately after installation on the PlayStation®Vita. All additional content files are read only, and the application cannot create or update a file in any directory within this directory..

How to Configure

The application can access additional contents via the addcont0: mount point. Note that directories for additional contents can only be created on memory cards (ux0:).

The file configuration of additional contents is indicated in Figure 6.

Figure 6 File Configuration of Additional Contents



Additional Content Root Directory (addcont0:)

This is the top directory for additional contents. In this directory, installed additional contents are created as directories.

Additional Content Directory

This is the directory where the data files actually contained in an additional content are stored. The label part (16 characters) of the content ID of the additional content is used as the name of this directory.

sce_sys/

This is the directory where all system files are placed. This directory is used by the system, and the application does not have the right to access it. A developer must not carry out implementation that entails the application to access this directory.

Subdirectory and File Name Restriction

Subdirectories that can be created within the additional content directory are limited to six levels. The following is an example where a file is created in the deepest-possible level.

- addcont0:0000000000000000/1/2/3/4/5/6/7.dat

Directories and files with the following name cannot be created immediately below the additional content directory.

- sce_*

The names of the directories or files within the additional content directory are limited to up to 63 bytes, while path length is limited to up to 199 bytes, excluding the mount point part. The supported character code is UTF-8, without distinction between upper and lower case letters.

How to Install Additional Contents

In order to use additional contents from the application, create and install an additional content package. Refer to the "Appendix 1: How to Install Packages" chapter on creating and installing packages.

Multiple additional contents can be installed. For each additional content, a directory will be created individually with the additional content directory name under addcont0:, and data files will be stored in this directory.

How to Mount Additional Content Directory

The addcont0: mount point of the additional content is mounted automatically at application startup if the following conditions are met. An application cannot unmount the mounted addcont0:.

- Additional contents are installed as a package in advance.
- The value of TITLE_ID or INSTALL_DIR_ADDCONT set to the application's param.sfo matches TITLE_ID set to the additional content's param.sfo.
- The passcode set to the application's package matches the passcode set to the additional content package.

param.sfo of the application and the package passcode must be specified in the development environment in order for addcont0: to be mounted when starting the application from the debugger or from ★ APP_HOME of the system software. Specifically, place to the game directory (app0:) the application's system file (param.sfo) and keystone file (keystone) created from the same passcode as that specified for the additional content package, as follows.

```
app0:sce_sys/param.sfo
app0:sce_sys/keystone
```

The location of app0: varies according to the operating environment of the application. Refer to the "Game" chapter on how the location of app0: is determined.

Refer to the "Param File Editor User's Guide" document included in Publishing Tools on how to create param.sfo.

Refer to the "Keystone Generator User's Guide" document included in Publishing Tools on how to create a keystone file.

> **Note**
> Because the keystone file is automatically placed in the package upon package creation, the only times the developer is required to manually place the keystone file is when the application is started from the debugger or from ★APP_HOME of the system software.

How to Access Additional Content Directory

For example, if you install an additional content package with the content ID shown below, the EXAM00000 part will be the title ID of the additional content, and the ADDCONTEXAMPLE00 part will be the name of the directory of the additional content:

```
XX0000-EXAM00000_00-ADDCONTEXAMPLE00
```

At this time, it will be possible to check the mounted additional content from the application as the following directory.

```
addcont0:ADDCONTEXAMPLE00/
```

In order to access the files and directories below the ADDCONTEXAMPLE00 directory, you need to use the application utility library's function for opening additional contents. After opening, the files can be read out in the same way as in normal file access. Refer to the "Application Utility Overview" document for the specific method to open additional contents.

Feature for Mounting Additional Content From Another Title

By using the feature for mounting the additional content from another title, the additional content root directory of an arbitrary title can be mounted as "addcont1:". The addcont1: data structure is based on the addcont0: specifications.

The procedure for using this feature is as follows.

(1) Set the product code for the additional content you want to use as INSTALL_DIR_ADDCONT_ADD_1 in the param.sfo of the application.
By performing this setting, additional content with a product code different from the application can be installed in a PlayStation®Vita.
If you further want to use the additional content of other product codes, continue to set INSTALL_DIR_ADDCONT_ADD_2 to 7.

(2) Mount the additional content of the other title from inside the application.
By calling the application utility library function that mounts additional content root directory (`sceAppUtilAddcontMount()`), the additional content root directory for the specified product code will be mounted as addcont1:.
Mounting also requires specifying the passcode set for the package of the additional content.

(3) Open the additional content and load it similar to standard additional content.
By calling the application utility library function that opens additional content (`sceAppUtilDrmOpen()`), by specifying the mount point called addcont1:, it can be accessed similarly to the addcont0: additional content.
The number of additional contents that can be open at the same time is 16 total between addcont0: and addcont1:.

Additional content from another title can only be mounted in addcont1:. To newly mount yet another additional content after mounting additional content from another title, the already mounted additional content root directory must be unmounted using the function for unmounting additional content root directories (`sceAppUtilAddcontUmount()`).

addcont0: is the application's own additional content root directory that is automatically mounted when the application starts up, so it cannot be unmounted.

In addition, when the procedure described in the "How to Install Additional Contents within an Application" chapter is performed, addcont1: will be automatically unmounted similar to addcont0: when the additional content is installed. addcont0: will be automatically mounted again after additional content installation, but note that addcont1: will not be automatically mounted.

Debug Feature of Additional Content Rights

A debug feature is provided to forcibly invalidate the right to an arbitrary additional content in order to check the behavior when the right to an installed additional content is lost. For details, refer to the "Game - Add-On Data (PS Vita)" section of the "System Software Overview" document.

9 How to Use Memory Cards

Only during application development, it is possible to use the ux0:data/ directory on the memory card as a read/write enabled area. If operating only the DevKit without connecting it to the development host computer, this directory can be used instead of host0:. Also, by creating save data in ux0:data/savedata, it is possible to simulate access with access speed inclusive of encryption/decryption, allowing to test operation more closely resembling that of the retail unit.

How to Access Memory Cards

Memory cards are mounted with the drive name ux0:. However, applications can only access the directory called ux0:data/. The ux0:data/ directory is created automatically when the DevKit is started up.

From the application, read/write access is possible by specifying a path beginning with ux0:data/. However, below the directories mounted as app0: and savedata0:, access will be subject to the access restrictions of that drive.

How to Browse Memory Cards

In order to browse ux0:data/ directly from outside the application, use psp2ctrl commands. For example, in order to edit ux0:data/savedata, execute the following commands from the command line of the development host computer:

```
> psp2ctrl mkdir ux0:data/savedata (create)

> psp2ctrl rm -R ux0:data/savedata (initialize)
```

While the application is mounting the psp2ctrl target directories, access will fail. When using psp2ctrl commands to browse the memory card, make sure that the application is not running.

Invalidation of File Descriptors

If an application is suspended while files or directories below ux0:data/ are open, when the application is resumed the file descriptors of those files or directories will be invalidated. ENODEV (error code: C1-2755-9/0x80010013) will result for all operations on the invalidated file descriptors, and only `sceIoClose()` will succeed.

If you wish to avoid this, execute the following command to create ux0:data/userdata.

```
> psp2ctrl mkdir ux0:data/userdata
```

Make sure to create the files and directories below this directory. Invalidation of file descriptors due to suspend/resume will not occur for files and directories created below ux0:data/userdata.

This behavior is applied to files and directories accessed through paths beginning with ux0:. Invalidation of file descriptors will not occur for files and directories below app0:, savedata0:, and addcont0:. However, if specifying a location on ux0: as the source directory when setting overlay with the file path setting file, the invalidation of file descriptors will occur even in case of access from the application through paths beginning with app0:.... Therefore, we recommend setting a location below ux0:data/userdata as the source directory.

Accessing ux0:data/savedata from the Development Host Computer

You can access save data saved in ux0:data/savedata from the development host computer. For details, refer to the "How to Access Game Data and Save Data from the Development Host Computer" chapter.

Precautions When Using Memory Cards

When using ux0:data/ during development, pay attention to the following points:

- In the DevKit (in **Release Mode**), TestKit and retail unit, the ux0: drive is not mounted
- When accessing ux0:data/savedata from the development host computer, pay attention to the following points:
  - If ux0:data/savedata is not mounted using the psp2ctrl mount-fs command, even if files are copied directly below ux0:data/savedata using the psp2ctrl cp command, it will not be possible to access them from the application. Before copying, mount with the psp2ctrl mount-fs command.
  - Likewise, if ux0:data/savedata is not mounted, copying the files created via savedata0: from the application to host0:, etc. using the psp2ctrl cp command will fail. Before copying, mount with the psp2ctrl mount-fs command.
  - When deleting everything below ux0:data/savedata using the psp2ctrl rm command, unmount with the psp2ctrl unmount-fs command. You cannot delete everything below savedata while ux0:data/savedata is mounted.

Note
For details on psp2ctrl commands, refer to the "Neighborhood and Utilities User's Guide" document.

10 How to Install Additional Contents within an Application

Normally, the additional contents used by a given application are installed only when the application is started up; below is an explanation of how to install additional contents from within an application, and use them without terminating the application.

Purchasing Additional Contents from within an Application

Use Store Checkout Dialog's or the application utility's Title Store application calling features (henceforth, both are referred to in this chapter as "store calling features" collectively) to enable users to purchase additional contents from within an application. An important point to note is that, when using these features, all access to additional contents must be terminated. If the store calling features are called while `sceIoOpen()`, `sceIoDopen()`, `sceIoGetstat()`, etc. have been called for addcont0: or addcont1: themselves or for files/directories under them, an error will occur. For details, refer to the "Store Checkout Dialog Overview" and "Application Utility Overview" documents.

Downloading Purchased Additional Contents

When a user purchases additional contents, the following will occur depending on that application's configuration and system status:

- Additional contents are downloaded in the foreground
  This will happen when the application is made with the VC-VC configuration, and no memory card is inserted in the PlayStation®Vita. Additional contents will be downloaded and installed immediately, and the store calling feature will terminate after installation is complete.
- Additional contents are downloaded in the background
  This will happen if a memory card is inserted in the PlayStation®Vita. The additional contents will be added to the system's background download list. The download will be performed asynchronously with the store calling feature.
- Additional contents are not downloaded
  The system's background downloads list can only contain up to 32 downloads at a time. If 32 background downloads are already being processed, additional contents will not be downloaded. Note that not only that application's additional contents, but also any other contents downloaded by the system may be added to the background download list. Moreover, downloading will also fail if there is not sufficient space available on the memory card.

Status of the Background Download List

By using the application utility's background download-related features, applications can obtain the following information concerning installable additional contents and an upgrade license on the background download list:

- Number of additional contents whose download is not complete
- Number of additional contents whose download is complete
- Whether a downloaded upgrade license exists or not

Likewise, the number of additional contents and upgrade licenses purchased/downloaded on the Regional Store accessible from "PlayStation®Store" on system software can also be obtained by using this feature; however, it is not possible to distinguish between contents purchased/downloaded with the store calling features and those purchased/downloaded from the Regional Store.

Installing Additional Contents

When Store Checkout Dialog is called in the install and upgrade mode, the additional contents whose download is complete at that time will be installed. As when using the store calling features, at this time, too, access to addcont0: or addcont1: or below must be terminated.

Behavior Before and After Calling Store Calling Features and Install and Upgrade Mode

If any one of additional contents is already installed, addcont0: exists on the application's process; however, while calling the store calling features and Store Checkout Dialog in the install and upgrade mode, addcont0:'s drive will be unmounted, and access to additional contents will temporarily be disabled; addcont0: will be mounted again after they are called (addcont1: will also be unmounted with the same timing, but it will not be automatically mounted).

If there isn't a single additional content installed, addcont0: will not be present from the beginning on the application's process; however, when additional contents are installed by calling a store calling feature or Store Checkout Dialog in the install and upgrade mode, after the call of those features is complete addcont0: will be mounted, enabling access to additional contents.

Note that if addcont0: is unmounted, any additional content opened with the application utility's additional content-related features will be closed forcibly.

The status of the download process of additional contents can be obtained with application utility's background download related feature; however, the status may change immediately after the install and upgrade mode is called. Thus, after calling the install and upgrade mode, make sure to check the installation status of the additional contents.

Installable Additional Contents

Additional contents are installable if the product code part of their content ID matches the product code set in TITLE_ID, INSTALL_DIR_ADDCONT or INSTALL_DIR_ADDCONT_ADD_1 to 7 in the game's system file (param.sfo). In other words, the accessible additional content mounted as addcont0: when the application starts up and the additional content mountable with the additional content root directory mount function apply.

INSTALL_DIR_ADDCONT can be set up arbitrarily, and is given priority over TITLE_ID.

11 How to Upgrade to Full Version within Upgradable Application

Below is an explanation of how to install an upgrade license from within an upgradable application, and switch to the operation of full version without terminating the application.

Purchasing Upgrade License from within an Application

The procedure is the same as that described in the "Purchasing Additional Contents from within an Application" section in the "How to Install Additional Contents within an Application" chapter.

Downloading Purchased Upgrade License

When a user purchases an upgrade license, the following will occur depending on the system status:

Upgrade license is downloaded in the background

This will happen if a memory card is inserted in the PlayStation®Vita. The upgrade license will be added to the system's background download list. The download will be performed asynchronously with the store calling features. Unlike the case of additional contents, the download process of the upgrade license will be completed immediately; thus, the status where the download is not done while the upgrade license is on the background download list does not occur.

Upgrade license is not downloaded

If a memory card is not inserted in PlayStation®Vita, or 32 background downloads are already being processed on the system's background download list, the upgrade license will not be downloaded. Note that not only that application's additional contents, but also any other contents downloaded by the system may be added to the background download list. Moreover, downloading will also fail if there is not sufficient space available on the memory card.

Status of the Background Download List

The procedure is the same as that described in the "Status of the Background Download List" section in the "How to Install Additional Contents within an Application" chapter.

Upgrade to Full Version

Even after the download of upgrade license is complete, the upgradable application is not automatically upgraded to the full version during the operation. To perform the upgrade to the full version without terminating the running application, it is necessary to explicitly call Store Checkout Dialog in the install and upgrade mode.

Behavior Before and After Calling Install and Upgrade Mode

By calling Store Checkout Dialog in the install and upgrade mode after the download of upgrade license is done, the upgradable application is upgraded after the completion of the call. Specifically, the SKU flag value that can be obtained with application utility's application parameter obtaining features changes.

The status of the upgrade license download process can be obtained with application utility's background download related feature; however, the download may be completed immediately after the install and upgrade mode is called. Thus, the upgradable application must be designed to check the SKU flag appropriately after the install and upgrade mode is called.

How to Debug the Upgradable Application's Operation (Trial/Full Version)

For the DevKit/TestKit, by setting **Upgradable App Debug** under ★**Debug Settings -> PSN**℠ of Settings application to other than **Off**, it is possible to give a false value to the SKU flag that can be obtained with application utility's application parameter obtaining features. Note that when the false value is set to 1(Trial), the SKU flag value is forcibly set to 1 even after the upgrade to the full version is done. The setting of **Upgradable App Debug** can be switched by interrupting the application at an arbitrary point. If this setting is changed, upgradable application's LiveArea™ will restart according to the switched state of trial version/full version.

12 How to Access Game Data and Save Data from the Development Host Computer

This chapter describes the feature for freely browsing/editing game data installed as a package and save data from the development host computer.

Examples

Below are some examples of the main uses of this feature:

- You can replace game data with corrected versions or make additions at a later time, without having to repackage/reinstall applications during development.
- You can copy files stored in save data to the development host computer, and check their contents. It is also possible to copy save data that an end user has saved to a full version PlayStation®Vita card or memory card to a development host computer.
- You can copy save data copied to the development host computer and save data created with save data creation tools, etc. on the development host computer onto the save data of the DevKit. This allows development and testing using save data of your choice.

Precautions

In some cases, rewriting the game data of installed applications may affect file access to game data.

Specifically, the followings may be expected:

- Longer time required to open files
- Decreased speed when reading out files
- Changes in the order of directory entries obtained with `sceIoDread()`

We recommend creating and installing the package again when conducting the application's final operation check.

Save data that an end user has saved to a full version PlayStation®Vita card or memory card is read-only. Adding a new file or overwriting an existing file is not possible.

When accessing save data that has been saved to a memory card, confirm the account settings of the DevKit to be used. Caution is particularly required when the DevKit (or retail unit) that saved the save data and the DevKit to be used for accessing the save data are different units. The account settings for the DevKit to be used for accessing the save data must be switched to the same account slot number as when the save data was saved. When accessing the save data saved by a retail unit (such as when a memory card has been collected from an end user), the account of the DevKit to be used must be switched to the account slot number 00 account. If the account settings of the DevKit to be used do not match the account slot number when the save data was saved, the save data saved on the memory card cannot be accessed. For details on the function for switching accounts, refer to the "PSN℠ - Account Selection" section in the "System Software Overview" document.

While this feature is being used to access (mount) arbitrary game data or save data on DevKit from a development host computer, application startup will return an error (error code: C0-12154-3/0x80800017). Unmount the data before starting an application.

Preparations

If you wish to access game data and save data using Windows Explorer, install File System Driver from the SDK Manager. For details on File System Driver, refer to the "Neighborhood and Utilities User's Guide" document.

Procedure for Accessing Game Data and Save Data

Below is an explanation of the procedure for accessing game data and save data:

(1) Convert the target application to a package using Publishing Tools.
For details, refer to the "Publishing Tools Overview" document.

(2) Install the package you have created.

(3) Use psp2ctrl to assign a drive letter to the DevKit's file system.
```
> psp2ctrl map <Drive Letter>
```

> **Note**
> As `<Drive Letter>`, specify the single alphabet character of an unused drive letter.
> You can also assign a drive letter with **File System Mapping** under Neighborhood's **Navigate**.
> Drives only need to be assigned once at the beginning.

(4) Use psp2ctrl to mount game data and save data.
```
> psp2ctrl mount-title <Title ID> <Passcode>
```

> **Note**
> As the title ID and passcode, input those set when creating the package. Mount operations and passcode input can also be performed with Windows Explorer.
> Open `<Assigned drive letter>:<devkit name>\Memory Card\game titles\<Title ID>` and click on **PS Vita -> Mount** or **PS Vita -> Manage Passcodes...** in the context menu.

(5) After mounting, access becomes possible from Windows Explorer and from the command prompt with the following path:
Game data
```
<Assigned drive letter>:<devkit name>\Memory Card\game titles\<Title ID>\app0
```
Save data
```
<Assigned drive letter>:<devkit name>\Memory Card\game titles\<Title ID>\savedata0
```

Note that, if you copy game data or save data to the development host computer, in its initial state it will be stored with a read-only attribute attached.

> **Note**
> In step (4), a full version PlayStation®Vita set in the DevKit can be mounted. In such cases, only the save data will be mounted.
> PlayStation®Vita card save data is read-only, writing is not possible.

While game data and save data are mounted, starting up an application will result in an error (error code: C0-12154-3/0x80800017). Unmount them before starting up the application.

The procedure for unmounting is as follows:

(1) If game data or save data are open in applications such as editors, close them.

(2) Use psp2ctrl to unmount game data and save data.
```
> psp2ctrl unmount-title <Title ID>
```

> **Note**
> Unmounting is also possible in Windows Explorer.
> Open `<Assigned drive letter>:<devkit name>\Memory Card\game titles\<Title ID>` and click on **PS Vita -> Unmount** in the context menu.

Procedure for Accessing ux0:data/savedata

In order to access ux0:data/savedata from outside the application, mount it with the following procedure:

(1) Use psp2ctrl to mount ux0:data/savedata.
```
> psp2ctrl mount-fs ux0
```

(2) After mounting, access will be possible from Windows Explorer and from the command prompt with the following path:
```
<Assigned drive letter>:<devkit name>\ux0\data\savedata
```

While ux0:data/savedata is mounted, it will not be possible to delete everything below savedata. Unmount it before deleting.

The procedure for unmounting is as follows:

(1) If files below ux0:data/savedata are open in applications such as editors, close them.

(2) Unmount ux0:data/savedata by using psp2ctrl.
```
> psp2ctrl unmount-fs ux0
```

Note
Mounting and unmounting ux0:data/savedata is also possible in Windows Explorer.
Open ux0:data/savedata and click on **PS Vita -> Mount** or **Unmount** in the context menu.

Limitations

- Access to patches and additional contents is not supported.
- If you add game data many times over, the management information of game data will gradually increase, reducing free space on the memory card. If free space becomes insufficient, delete and reinstall the application.
- Save data access does not support renaming of the following files in the sce_sys folder.
  - `<Assigned drive letter>:<devkit name>\Memory Card\game titles\<Title ID>\savedata0\sce_sys`
  - `<Assigned drive letter>:<devkit name>\ux0\data\savedata\sce_sys`

13 Procedure for Loading Save Data Created with a product version application on a DevKit/TestKit

This chapter provides a practical explanation of how to read save data created with a product version application by using the features described in the "How to Access Game Data and Save Data from the Development Host Computer" chapter.

Purpose

Allows reading the save data of marketed applications that have malfunctioned in a development environment to make the malfunction reoccur and analyze it.

Procedure

(1) Obtain the user's save data
Depending on the application's configuration, there are two ways to do this:

- In the case of VC-VC configuration
  Given that the save data is stored on the PlayStation®Vita card, it will be necessary to receive the PlayStation®Vita card itself from the user. Proceed to step (3).
- In the case of VC-MC/MC-MC configuration
  Have the user use Content Manager Assistant for PlayStation® in the user's environment, and give you a USB memory or e-mail attachment with the savedata folder copied to the following location on a PC.

```
\PS Vita\APP\<16 alphanumeric characters>\<title id>\savedata
```

Note
If the user's version of the PlayStation®Vita system software is older than 1.800, the following file that is necessary for loading the user's save data on the DevKit/TestKit will not be created:

\PS Vita\APP\<16 alphanumeric characters>\<title id>\savedata\savedata.psvinf

In this case, have the user report the <16 alphanumeric characters> and create the savedata.psvinf anew. For example, if the <16 alphanumeric characters> are "0123456789abcdef", savedata.psvinf will be an 8-byte data file in binary format (from the top, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef).

(2) Import the save data to the DevKit/TestKit
Install the application's package (the .pkg file) on the DevKit/TestKit. Copy the application's contents from the Content Manager application on the DevKit/TestKit's home screen to the development host computer, checking off the **Saved Data only** check box.
The following directory will be created on the development host computer; overwrite the contents of the savedata folder received in step (1) here:

```
\PS Vita\APP\0000000000000000\<title id>\savedata
```

Again, use the Content Manager application to copy the contents of the application copied on the development host computer to the DevKit/TestKit, checking off the **Saved Data only** check box.

By following the procedure above, the contents of the user's savedata folder will be imported to the following location on the memory card:

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

(3) Extract save data as plain text
Use the psp2ctrl command to mount save data stored on the PlayStation®Vita card or memory card, and extract it to the development host computer as plain text data.

> **Note**
> The save data files and directories extracted to the development host computer will be stored with the read-only attribute attached; make sure to cancel the read-only attribute manually.

(4) Have the application load the save data in plain text
Place the save data in plain text extracted in step (3) in a save data directory such as host0:savedata/. In such cases, save data directories are configured as follows:

host0:savedata/sce_sys/	: System directory
host0:savedata/sce_sys/param.sfo	: System file
host0:savedata/sce_sys/sdslot.dat	: Slot information file
host0:savedata/sce_sys/safemem.dat	: Safe memory file
host0:savedata/sce_sys/safemem.dat	: Files/directories saved by the application

Files below the system directory will be created automatically when the application is started up with an empty host0:savedata/ folder ready; overwrite the save data in plain text extracted in step (3) after first starting up the application and creating the system directory.

14 Appendix 1: How to Install Packages

Below is explanation on how to create application, additional content and patch packages (pkg files), to install them on the DevKit, and to start them up from the home screen.

How to Create Application Packages

Use Publishing Tools to create packages (pkg files) for installing applications, additional contents and patches on the DevKit. For details on how to create packages, refer to the "Publishing Tools Overview" document.

Application Packages Storage Location

In order to install packages (pkg files), it is necessary to save pkg files in a predetermined location. The storage location differs depending on how the DevKit and development host computer are connected. Files will be saved in either of the following locations on the development host computer:

- When using host0:
  Create a "package" directory on the file server directory (which can be set using Neighborhood or the psp2ctrl fsroot command). Save the created package (pkg file) inside this "package" directory.
- When using Content Manager Assistant for PlayStation®Vita DevKit
  Start up the Content Manager Assistant for PlayStation®Vita DevKit on the development host computer, and create a "package" directory on the directory set in the "Application/Backup File" folder. Save the created package (pkg file) inside this "package" directory.

How to Install Packages on the DevKit Using ★Package Installer Application

Install package files with the following procedure:

(1) Connect DevKit to the development host computer.
If the package (pkg file) is saved on host0:, connect the DevKit and the development host computer with the USB port for development (mini-B), and either execute psp2ctrl connect from the development host computer, or make the development host computer detect the connection with the DevKit by selecting **Connect** from Neighborhood. Connect with a multi-use port if using the Content Manager Assistant for PlayStation®Vita DevKit. In this case, the connection will be detected automatically.

(2) Start up the DevKit and tap the icon of the "★Package Installer" application on the home screen. Tap the gate from LiveArea™ and start up the application.

(3) The two buttons "Content Manager Assistant" and "host0:" will be displayed. Depending on the method for saving the package (pkg file), tap either "host0:" or "Content Manager Assistant".

(4) A list of the packages (pkg files) saved on the development host computer will be displayed. Begin installation by tapping on the package file to be installed. You can also install all listed packages. In this case, tap the option button on the bottom right of the screen, and tap **Yes** on the dialog box that is displayed.

(5) Installation is complete, provided that it terminates normally.
However, when installing patch packages, installing a different game from the one specified when creating the patch will result in an error (error code: C0-12574-9).

How to Install Packages on DevKit Using Neighborhood

Install package files with the following procedure:

(1) Connect DevKit to the development host computer.

(2) Start Neighborhood and select the DevKit to install package to.

(3) Right-click to display menu and select **Install Package…**, and select the package file to install. Select **Uninstall Package…** in the same manner to uninstall a package.

How to Install Packages on DevKit Using psp2ctrl Commands

Install package files with the following procedure:

(1) Connect DevKit to the development host computer.

(2) Start command prompt and execute "psp2ctrl pkg-install XXXX.pkg" (specify package to install to XXXX.pkg). The install process will be started. Execute uninstall in the same manner with "psp2ctrl pkg-uninstall XXXX.pkg" to uninstall a package.

Condition to Install a Patch Package

To install a patch package, the application package on which the patch is to be applied (application package specified as the target on which the patch is to be applied when creating the patch package with Publishing Tools) must be installed in advance.

Condition to Install an Additional Content Package

To install additional content package, the application package with settings to use that additional content package must be installed in advance. Regarding settings of the application to use additional contents, refer to the "Additional Contents" chapter in this document.

Moreover, when installing multiple additional contents with the same product code, the same passcode must be set to each of the additional content packages.

How to Use Installed Contents

When application package installation is completed normally, an application icon is added to the home screen. Tap it to open LiveArea™, and tap the gate to start up the application.

When patch package installation is completed normally, the patch will automatically applied upon application startup.

With regard to additional content package, data files can be accessed from an application by calling an additional content mount API from the application. For the additional content mount API, refer to the "Application Utility Reference" document.

15 Appendix 2: Feature for Importing/Exporting Applications

The system software has an import/export feature enabling backup by application. This is provided in all system software, including the PlayStation®Vita and the DevKit/TestKit. However, part of the specifications is different in the DevKit/TestKit due to considerations of convenience during development.

Content Manager Application

First, we will describe the Content Manager application the PlayStation®Vita is equipped with. A Content Manager icon is displayed on the home screen. The application import/export feature can be used from there.

Preparations Before Start-Up

Import/export of applications is performed from/to the development host computer. First, you will need to start up the Content Manager Assistant for PlayStation®Vita DevKit from the development host computer, and connect the DevKit/TestKit and the development host computer.

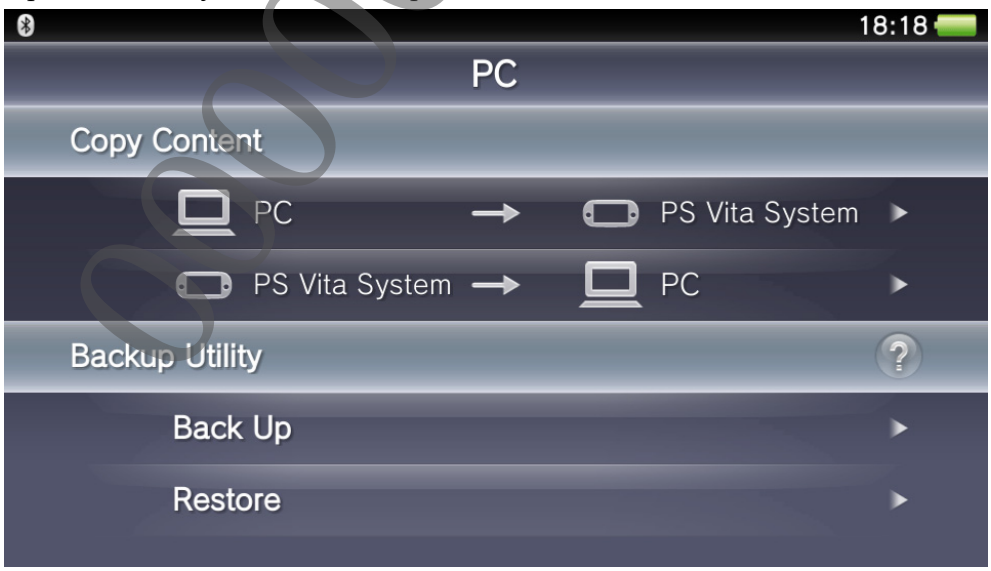
For connection with the development host computer, use multi-use port for DevKit/TestKit (PTEL-1000 series) and use the USB port for TestKit (PTEL-2000 series).

Also, sign up to Sony Entertainment Network from the PlayStation®Vita. This feature cannot be used with PlayStation®Vita units on which you have not signed up and do not have a Sony Entertainment Network account. (For the DevKit/TestKit, signing up is not necessary. For details, refer to the "Differences in the DevKit/TestKit's Specifications" section described below.)

Import/export

After tapping on the Content Manager application and starting it up by tapping the gate, perform the following operation to actually start importing/exporting applications:

(1) Tap on either the import or export option under the **Copy Content** item.
Tap on **PC -> PS Vita System** to import.
Tap on **PS Vita System -> PC** to export.



(2) Tap on **Application**

(3) Tap on **Application (PS Vita)**

(4) A list of the applications installed will be displayed; select the application/s (multiple choice is possible) and tap **Copy**

(5) Tap **OK** in the confirmation dialog.

> **Note**
> In the DevKit/TestKit, if you only wish to backup save data, tap **Saved Data only** on the confirmation dialog to display the check mark.

Game-related Contents for Back-Up

The import/export feature collectively handles application game data, patches, save data and additional data. Even if contents are imported after having been exported, they cannot be lost during that time, but it is not possible to only back-up save data. (In the DevKit/TestKit, it is possible to import/export save data only. For details, refer to the "Differences in the DevKit/TestKit's Specifications" section described below.)

Encryption of Back-up Files

The application back-up files saved on the development host computer are encrypted. Also, they cannot be imported if the PlayStation®Vita unit is not signed up with the same Sony Entertainment Network account used at the time of exporting. In other words, the same account must be used when exporting and importing. (In the DevKit/TestKit, where signing up is not necessary, the account does not need to be the same. For details, refer to the "Differences in the DevKit/TestKit's Specifications" section described below.)

Differences in the DevKit/TestKit's Specifications

In the DevKit/TestKit's import/export feature, restrictions are alleviated in the following ways:

- Signing up to Sony Entertainment Network is not required.
- It is possible to import/export save data only.

Below is an explanation of the respective usage methods during development.

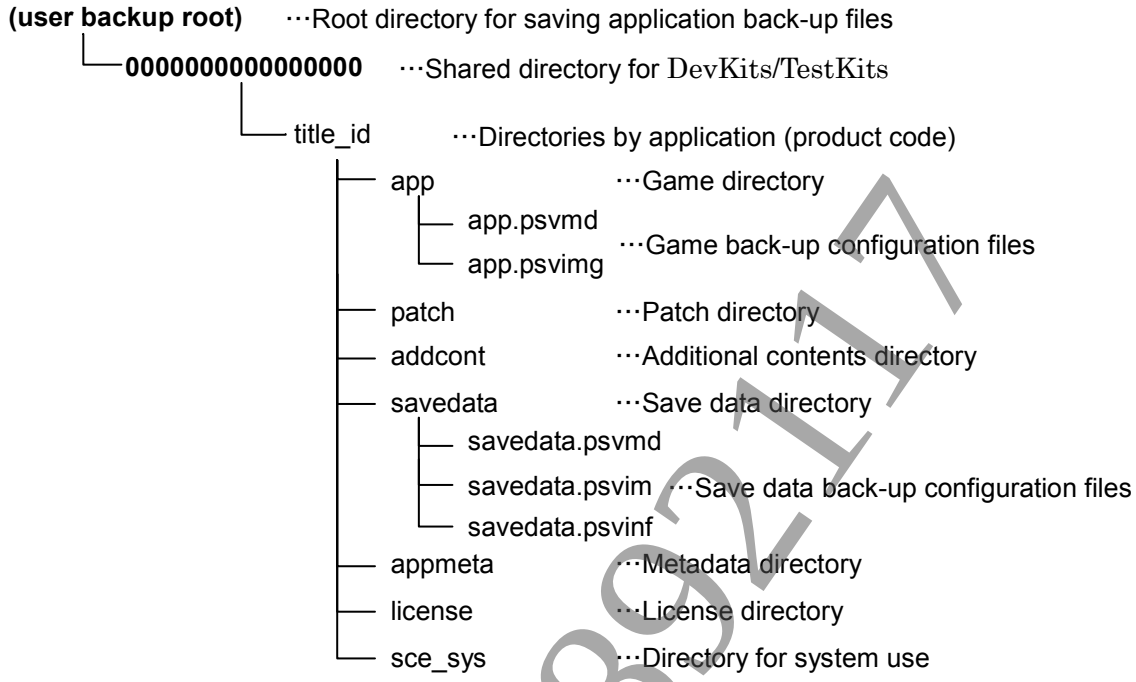
Sign Up not Required with the DevKit/TestKit

In the DevKit/TestKit, it is not necessary to sign up to Sony Entertainment Network when using the feature for importing/exporting applications. Between DevKits/TestKits, it is possible to share backup files on the development host computer. In addition, save data backed up from a PlayStation®Vita requiring sign-up can also be imported to a DevKit/TestKit, but save data exported from a DevKit/TestKit cannot be imported by a PlayStation®Vita requiring sign-up.

Possible to Import/Export Save Data Only

In the DevKit/TestKit, the file configuration of the applications backed-up on the development host computer is as follows:

Figure 7 Configuration of DevKit/TestKit Back-Up Files



When only importing/exporting save data, only the savedata/ directory out of the files above will be read/written. In this way, the time for importing/exporting will be significantly shortened. This makes it easier to release the save data only to a team that performs debugging using the DevKit/TestKit.

However, bear in mind the following points.

Conditions for Exportable Save Data

Exportable save data is limited to the save data created in ux0:user/<user_id>/savedata/<title_id>/, and the following conditions need to be met:

- An application installed as a package is running;
- There is no save data directory for development available (host0:savedata, ux0:data/savedata).

Also, save data cannot be exported from the following save data directories for development:

- host0:savedata
- ux0:data/savedata
- In addition, any directories on host0: specified in the file path setting file's "savedata="

user_id and title_id of the Save Data Directory

Below is an explanation of the <user_id> and <title_id> of the exportable save data directory ux0:user/<user_id>/savedata/<title_id>/

- <user_id>
  This is a number allotted with the DevKit/TestKit's account selection function. Account selection is performed with **Account Selection** under **PSN**℠ in the settings application's ★**Debug Settings**. Use **Create New Account** here to automatically allot a number and create an account. This number will be the user_id.

  > **Note**
  > The last two figures of the Account Name's initial setting value when selecting an account (for example: account00) will be the user_id. The Account Name can be changed, but it can be helpful to avoid changing the last two figures in order to be able to subsequently confirm the user_id.

- <title id>
  This is the application's product code. If necessary, <title id> can be set to a product code different from that of the application. To set the product code to a different one, use param.sfo's INSTALL_DIR_SAVEDATA parameter with Param File Editor for PlayStation®Vita.

Notes on the user_id Directory

For example, if you have used the account number 05 and created save data with an application whose product code is ABCD00001, the save data directory will be ux0:user/05/savedata/ABCD00001/.

When inserting a memory card on which this save data is saved into another DevKit/TestKit, the save data directory will not change; therefore, it will not be possible to use that save data unless the application is started up with an account having 05 as its user_id number on the other DevKit/TestKit as well.

On the other hand, if using the feature for importing/exporting applications (or, likewise, the feature for importing/exporting save data only) the user_id of the save data directory will automatically change to the account number used when importing. If the game is started up with the imported account, it will be possible to use the save data.

Note that, in the two cases above, the user_id will be handled differently by the system.

Other Restrictions

Importing save data only to a DevKit/TestKit on which the game has not been installed is not possible and will result in an error.

16 Appendix 3: Features to Install/Delete Save Data

This chapter describes the feature to install (copy) save data on the development host computer onto a DevKit memory card and the feature to delete save data on a DevKit memory card.

Save Data That Can Be Installed

There are two types of save data that can be installed.

- Save data of an application that has been exported onto the development host computer (For details, refer to the "Appendix 2: Feature for Importing/Exporting Applications" chapter.
- Save data that has been created on the development host computer (host0:savedata/ or in a directory specified in the file path setting file)

How to Install

To install save data, execute the psp2ctrl command in one of the following formats.

- `> psp2ctrl sd-install <path>`
  Installs the save data directory on the development host computer specified in `<path>` onto the appropriate location on the memory card as save data of the current user on DevKit.
  Only application save data exported onto the development host computer can be specified in `<path>`.
- `> psp2ctrl sd-install /userid:<user id> <path>`
  Installs the save data directory on the development host computer specified in `<path>` onto the appropriate location on the memory card as save data of the DevKit user specified in `<user id>`.
  Only application save data exported onto the development host computer can be specified in `<path>`.
- `> psp2ctrl sd-install /raw <path>`
  Installs the save data directory on the development host computer specified in `<path>` onto ux0:data/savedata/ of DevKit.
  Only save data created on the development host computer can be specified in `<path>`.

How to Delete

To delete save data, execute the psp2ctrl command in one of the following formats.

- `> psp2ctrl sd-format <title id>`
  Of the application save data specified in `<title id>` on DevKit, only deletes save data of the current user.
- `> psp2ctrl sd-format /userid:<user id> <title id>`
  Of the application save data specified in `<title id>` on DevKit, only deletes save data of the user specified in `<user id>`.
- `> psp2ctrl sd-format /allusers <title id>`
  Deletes all application save data specified in `<title id>` on DevKit.
- `> psp2ctrl sd-format /raw`
  Deletes the entire directory of ux0:data/savedata/ on DevKit.

Operation Using Neighborhood

Neighborhood GUI can be used to perform the same operations as each command of psp2ctrl explained in this chapter. For details, refer to the "Accessing the File System of Development Kits" section of the "Neighborhood and Utilities User's Guide" document.

Runtime Error

When a parameter specified to a command is invalid or when an operation fails, one of the following messages will be displayed.

Value	Description
`Savedata was not found.`	Directory does not exist in the specified path
`Invalid savedata format.`	Specified directory is invalid
`User **** does not exist.`	Specified user does not exist on DevKit (****:<user id>)
`Application **** is not installed.`	Specified application does not exist on DevKit (****:<title id>)
`No application using savedata **** is installed.`	Application that will use the save data to be installed does not exist on DevKit (****:<title id>)
`There is not enough free space on the memory card.`	Free space on the memory card is insufficient
`Failed!`	Other error

17 Troubleshooting

This chapter explains the issues that are frequently encountered during development, and how to resolve them.

Application Start-Up

Below is a description of the issues that may occur when performing start-up from the debugger, ★ APP_HOME or a package.

- Error (C0-11471-4/0x8080000d) occurs at start-up
  Occurs when starting up from ★APP_HOME. It means that the application's system file (param.sfo) is not placed in the correct location. Refer to the "System Software Overview" document on the location of system files.
- Error (C0-11479-2/0x80800015) occurs at start-up
  Occurs when starting up from the debugger or from ★APP_HOME. It means that the format of the system file (param.sfo) of the application that has been loaded is invalid. Refer to the "Param File Editor User's Guide" document included in Publishing Tools on how to create system files.
- Error (C1-2738-0/0x80010002) occurs at start-up
  Occurs when a directory on host0: is mounted as savedata0:. It means that the necessary files are not present in the save data system directory (savedata0:sce_sys). If the DevKit/TestKit is rebooted while the application is running, the creation of the system directory may be interrupted, causing this situation. Delete the system directory.
- Error (C0-12594-1/0x80800019) occurs at start-up
  This is caused by data above the size set in Save Data Quota already being saved in the save data directory.
  It may occur when ux0:data/savedata or ux0:user/<user id>/savedata/<title id> is shared among different applications. Delete the directory where save data is mounted.
  ux0:data/savedata can be deleted with "psp2ctrl rm -R ux0:data/savedata".
  ux0:user/<user id>/savedata/<title id> can be deleted with ★ **Delete Savedata**.
- Error (C0-11138-4/0x80800006) occurs at start-up.
  Occurs when starting up from a package. It means that the passcode set to that application does not match the passcode set to the save data that the application is attempting to mount. This may occur when a package with the same product code but a different passcode is overwrite-installed.
  You can enable application start-up either by creating a new package passcode to match that of the previous package and overwrite-installing the package again, or by deleting the save data with ★ **Delete Savedata**.
  Moreover, this error will also occur when attempting to mount save data created with a Fake version package to a product version package, and vice versa. The version of the package must be matching between the package that created the save data and the package to which save data is to be mounted.

  Note
  For ★ **Delete Savedata**, refer to the "System Software Overview" document.
- Error (C0-12154-3/0x80800017) occurs at start-up
  Occurs when starting up from the debugger, ★APP_HOME or a package. It occurs if attempting to start up an application when the game data or save data directory on the DevKit has been mounted with psp2ctrl mount-title or mount-fs. Before starting up the application, unmount these with psp2ctrl unmount-title or unmount-fs.
- Error (C0-11146-3/0x80800022) occurs at start-up
  Occurs when starting up from the debugger or ★APP_HOME. This means that the working directory (app0:) is set to the development host computer's root drive (C:\, D:\, etc.). Specify a different location for the working directory. Refer to the "File Path Setting Guide" document on setting the working directory.

Application Runtime

Below is a description of the issues that may occur during application runtime.

- File operation fails with (C1-2755-9/0x80010013)
  The file descriptor has been invalidated after suspension/resuming. Refer to the "Invalidation of File Descriptors" section in the "How to Use Memory Cards" chapter of this document.
- Additional content operation fails with (C1-2755-9/0x80010013)
  addcont0: is not mounted correctly at application start-up. Check the following:
  - Has the additional content package been installed correctly?
  - Has the product code of the additional contents been specified correctly in the application's system file?
  - Does the passcode of the additional contents match that of the application?

  For details, refer to the "Additional Contents" chapter of this document.
- Trouble mounting savedata0:
  savedata0:'s mounting destination varies depending on the start-up method (debugger, ★APP_HOME or package) and on the **Release Check Mode** (**Development Mode** or **Release Mode**). For details, refer to the "How to Access Save Data Directory" section in the "Save Data" chapter of this document.
- Trouble mounting addcont0:

  This might be due to the following causes:
  - The application's system file (param.sfo) or keystone file (keystone) is not placed correctly (when starting up from the debugger or ★APP_HOME).
  - The product code (TITLE_ID) written in the application's system file, or the shared additional content setting (INSTALL_DIR_ADDCONT) does not match the product code specified in the additional content package.
  - The application's passcode does not match that of the additional contents.
  - The package formats (Fake/Finalized) of the application package and the additional content package do not match. Although the packages you create are Fake packages, they may become Finalized packages when placed on the server.

  For details, refer to the "How to Mount Additional Content Directory" section in the "Additional Contents" chapter of this document.
- Error dialog (C2-12828-1/0x80103909) is displayed
  An exception occurs on the application process. Check the call stack or core dump to identify the causes.

Package Installation

Below is a description of the issues that may occur when installing package files (.pkg) from the Fake Package Installer:

- Error (C0-12574-9/0x80871c04) occurs during patch package installation
  This error occurs when a game package that differs from the game package specified upon patch creation is installed. Please install the correct game package.
- Error (C0-11138-4/0x80800006) occurs during additional content package installation
  This error occurs when an additional content package of the same product code but different passcode is already installed. Either delete all of the already-installed additional content packages with the same product code or set the same passcode to all additional content packages that have the same product code.