PlayStation®2 EE Library Overview
Release 2.4.3

Device Libraries

Publication date: January 2002

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052, Japan

Sony Computer Entertainment America
919 E. Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.

Summary Table of Contents

About This Manual

This is the Runtime Library Release 2.4.3 version of the *PlayStation®2 EE Library Overview - Device Libraries* manual.

The purpose of this manual is to provide overview-level information about the PlayStation®2 EE Device libraries. For related descriptions of the PlayStation®2 EE Device library structures and functions, refer to the *PlayStation®2 EE Library Reference - Device Libraries*.

Changes Since Last Release

Chapter 4: Memory Card Library

- In the "File Attributes" section of "Filesystem Overview", the description of the copy prohibition attribute on the OSD browser screen has been partially deleted.

Chapter 7: Controller Library

- In the "Output Messages" section of "Notes", a description of the "padman: Over Consumpt Max" message has been added.
- In "Notes", a "Limitations of Analog Controllers" section has been added.

Related Documentation

Library specifications for the IOP can be found in the *PlayStation®2 IOP Library Reference* manuals and the *PlayStation®2 IOP Library Overview* manuals.

Note: the Developer Support Web site posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

Typographic Conventions

Certain Typographic Conventions are used throughout this manual to clarify the meaning of the text:

Convention	Meaning
`courier`	Indicates literal program code.
italic	Indicates names of arguments and structure members (in structure/function definitions only).
medium bold	Indicates data types and structure/function names (in structure/function definitions only).
blue	Indicates a hyperlink.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In North America:	*In North America:*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd. Foster City, CA 94404, U.S.A. Tel: (650) 655-8000	E-mail: PS2_Support@playstation.sony.com Web: http://www.devnet.scea.com/ Developer Support Hotline: (650) 655-5566 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In Europe:	*In Europe:*
Attn: Production Coordinator Sony Computer Entertainment Europe 30 Golden Square London W1F 9LD, U.K. Tel: +44 (0) 20 7859-5000	E-mail: ps2_support@scee.net Web: https://www.ps2-pro.com/ Developer Support Hotline: +44 (0) 20 7859-5777 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT)

Chapter 1: Hard Disk Library Overview

Library Overview

The hdd module controls the hard disk drive and manages partitions.

Partitions are arranged using a system known as APA (Aligned Partition Allocation). The individual partition size is a power of 2, however, up to 64 sub-partitions can be associated with a single main partition. Volumes can be managed flexibly with the support of the filesystem. This enables partitions to be created and deleted safely and ensures that no problems will occur even if there is an abrupt power failure during processing.

All operations for the hdd module are performed via the I/O manager. Device names are "hdd0:" or "hdd1:", and the number represents the number of the drive that is physically connected.

Related Files

The following files are required to use the hdd module.

EE

Table 1-1

Category	Filename
Header file	sifdev.h
Library file	libkernl.a

IOP

Table 1-2

Category	Filename
Header files	stdio.h
	dirent.h
	errno.h
	sys/file.h
	sys/ioctl.h
	sys/mount.h
	sys/stat.h
Library file	iop.ilb
Module files	dev9.irx
	atad.irx
	hdd.irx

Setting Environment

To use the hard disk drive, first set the time using the "Set PlayStation 2 RTC" on the Management Tool screen.

Note: To perform this setting, the "DTL-T10000 Management Tool PStoolSetup Ver.1.2" package released on the developer support website has to be installed.

Modules

Dependency Relationships Between Modules

To use the hdd module, the following 2 modules must be loaded sequentially prior to hdd.irx and made resident in memory.

dev9.irx	Device module for which card is connected
atad.irx	Hard disk drive driver module

If an error occurs during initialization of these modules, then they will not be resident in memory. When loading the hdd module, ensure that these modules are resident by using LoadStartModule() on the IOP and by using sceSifLoadStartModule() on the EE.

hdd Module Arguments

The following arguments can be specified when the hdd module is loaded.

-o Number of Partitions

Specifies the number of partitions that can be open simultaneously. In the current version each open consumes 564 bytes of memory.

In addition, the number of partitions that can be mounted simultaneously for pfs.irx can also be specified, however, it is recommended that it be set to the same value.

-n Number of Block Buffers

Specifies the number of block buffers the driver will use. By default, 3 buffers are used. In the current version, each buffer consumes 1048 bytes of memory.

The speed of module operation of the hdd module may be increased by increasing the number of block buffers.

Partition System

Partition Layout Method

Partitions are arranged using a method known as APA (Aligned Partition Allocation). In this method, the maximum size of a partition is restricted to a power of 2 (128M ~ 32GB). When a partition is layed out, the sector that is aligned with the size of the partition itself is made the header.

Partition Size

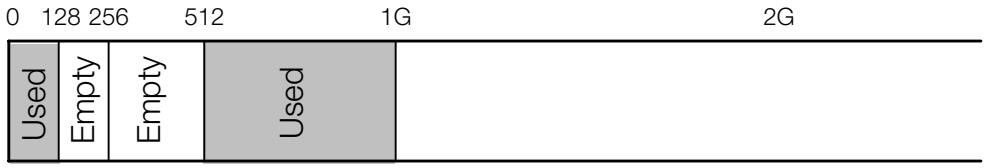
The partition size must be a power of 2 (2 ^ n) and in the range from 128MB ~ 32GB. Moreover, in the driver, the partition size is limited to 1/32 of the capacity of the entire hard disk drive. This means for a 10GB disk the maximum partition size is 256 MB, and for a 40GB disk the maximum partition size is 1GB.

Steps in Creating a Partition

1. Find an empty partition having the required size. If such an empty partition can be found, use it.
2. Otherwise, find an empty partition having twice the required size. If such an empty partition can be found, divide it in two and use one of its divisions.
3. Otherwise, find an empty partition having four times the required size. If such an empty partition can be found, divide it in four and use one of its divisions.
4. Otherwise, in a similar fashion, find empty partitions having sizes that are multiples of 8, 16, etc.. of the required size. If such empty partitions can be found, divide them and use one of their divisions.
5. If an empty partition of a suitable size cannot be found, then create a partition by making the sector that has been aligned with the required size after the end of the last partition, its header. If there exists any empty space between the last partition and the new partition, then make that section an empty partition.

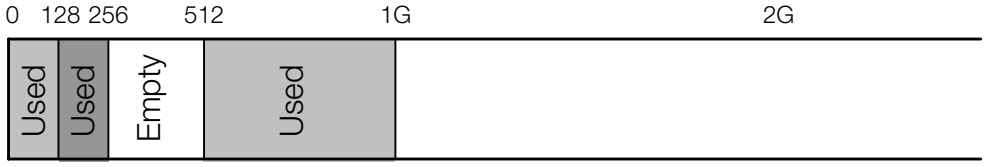
For example, partitions would be created as follows.

Figure 1-1



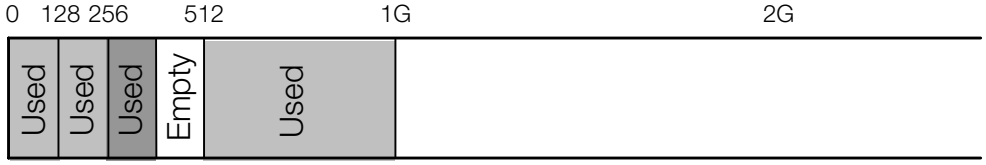
Here, if a 128M partition is allocated then the situation becomes as follows.

Figure 1-2



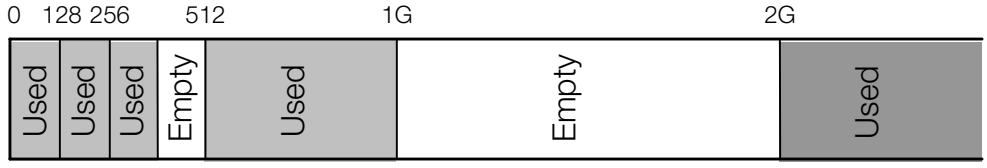
Also, if a 128M partition is allocated then the situation becomes as follows.

Figure 1-3



In this state, if a 2G partition is allocated then the situation becomes as follows.

Figure 1-4

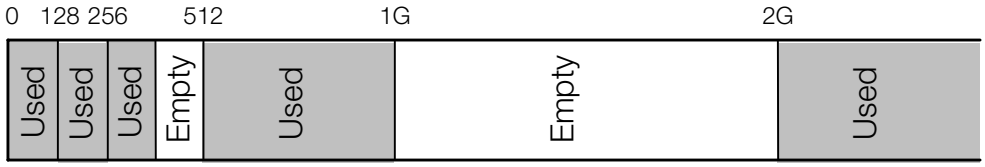


Steps in Deleting a Partition

1. If the specified partition is the last partition, it will be deleted. If the immediately preceding partition is an empty partition then it is also deleted.
2. If the specified partition is the last partition then it is made an empty partition.
3. If the partition preceding the empty partition is also an empty partition and if both of these when connected form an aligned partition for their combined size, then they are joined together to form a single empty partition. This is also the case when the partition following it is an empty partition.

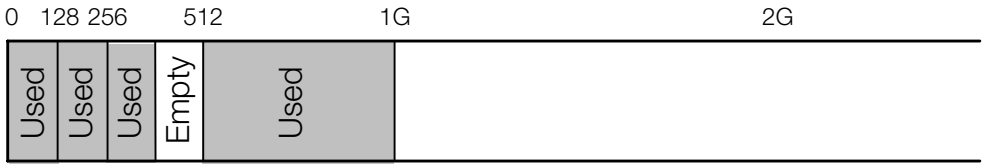
For example, consider the partitions shown below.

Figure 1-5



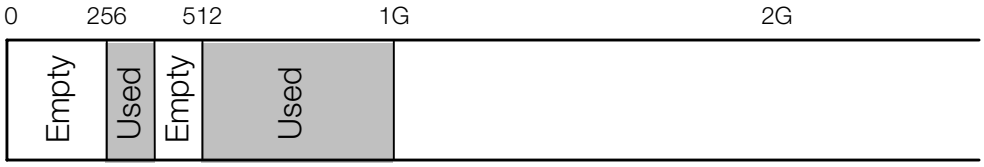
Here, if the 7th partition (2G) is deleted then the partitions become as follows.

Figure 1-6



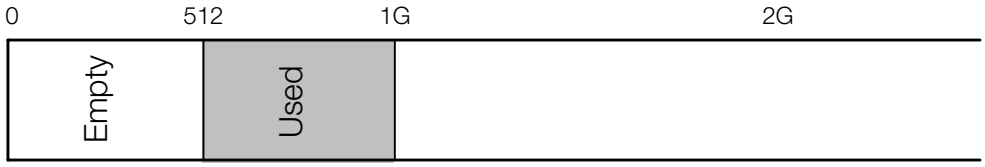
Moreover, if the 1st partition and the 2nd partition (0 and 128) are deleted then the partitions become as follows.

Figure 1-7



Moreover, if the 2nd partition (256) is deleted then the partitions become as follows.

Figure 1-8



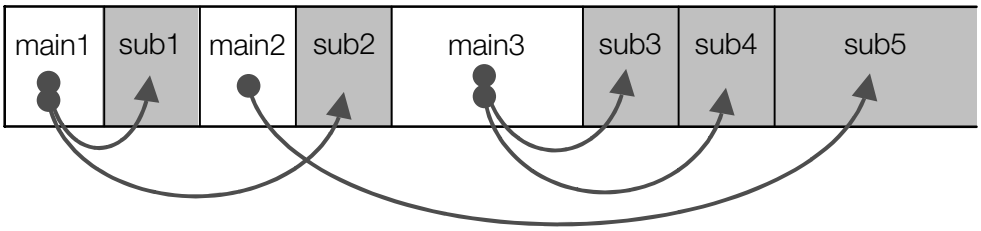
Main Partition and Sub-partitions

Partitions are distinguished as main partitions and sub-partitions.

Although the method of placing partitions is exactly the same in both cases, a main partition can hold linking information for up to 64 sub-partitions.

Since the filesystem driver handles multiple partitions together as a single block device, it can be seen that partitions having various sizes can be created through the addition of sub-partitions. The size of a partition can also be changed after it is created. When creating a partition, try to add multiple sub-partitions which are as small as possible in order to effectively use the disk capacity.

Figure 1-9



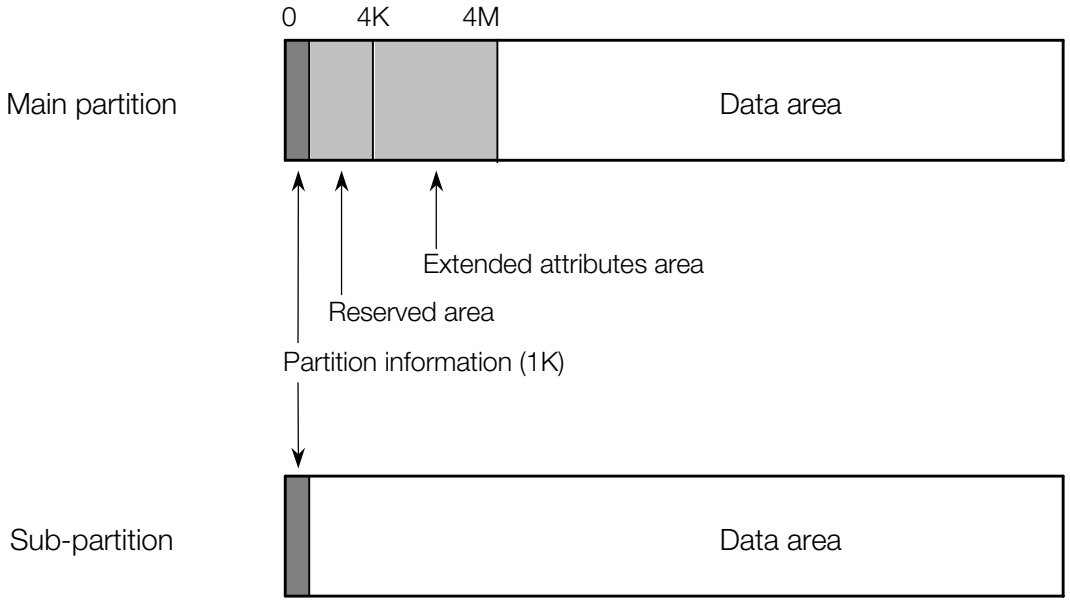
If there is an instruction to delete a main partition, the main partition is deleted after all linked sub-partitions are deleted. Basically, sub-partitions can only be deleted starting with the very last one that was added.

Partition Information

Information about a partition such as the partition ID, password, partition size, etc. is placed in the header of each partition. This information is common to both the main partition as well as the sub-partitions, and occupies 1024 bytes per partition.

In addition to this, a reserved area of 3K bytes and an extended attribute area of (4M - 4K) bytes exists within the main partition.

Figure 1-10



The size of each partition includes the size of the partition information / reserved area / extended attribute area.

Partition Identifier String

The partition identifier string is used to specify the partition for the various APIs of the hdd module. The partition identifier string consists of device name + unit number + ':' followed by a string made from a sequence of the following strings separated by commas.

- **Partition ID**
  This is a unique ID for the entire disk and is officially issued by SCE for each title application. However, any string can be used as long as it is unique during the creation stage and is no more than 32 characters long.
- **Password**
  Password required for read/write access. The password can be up to 8 characters long.
- **Password for exclusive read**
  Password for read-only access. The password can be up to 8 characters long.
- **Partition size**
  Character string which specifies the size of the partition. The valid character strings are:
  128M / 256M / 512M / 1G / 2G / 4G / 8G / 16G / 32G
- **Filesystem name**
  Specifies the name of the filesystem. At present, only "PFS" is valid.

The partition size and filesystem name specifications can be omitted except when the partition is created. The full password and the read-only password can be omitted if not needed. When items are omitted, specifications should be made in a continuous manner, with commas.

Access Restrictions

Restrictions can be imposed over access to each of the partitions based on the full password and the read-only password. When passwords are specified, in the least, the read-only password is required to open for read-only accesses, whereas the full password is required to open for both read and write.

Extended Attributes Area

An extended attributes area is set up in the main partition for storing icon files, etc.

Normal read/write/lseek are performed for the extended attributes area, however, the size of each read/write must be in units of 512 bytes. Specifying any other value than this results in an error. Also, when performing accesses from the EE, buffers must be 64-byte aligned.

This operation is identical to that of processing a file having a size of (4M - 4K) bytes.

Special Partitions

Special partitions all have a partition ID that starts with the character "_" . A partition with an ID starting with the character "_" can be created only during formatting and cannot be created or deleted at any other time.

Partitions that are created during formatting are shown below.

Table 1-3

Partition	Contents and Use
__mbr	Master boot record
__net	Reserved
__system	Partition for browser use only
__sysconf	Partition for configuration file use only
__common	Common area

There is also a _tmp partition besides those described above.

Similar to the /tmp directory, the _tmp partition is convenient for storage of temporary files. By placing temporary files in this partition, the fragmentation of other partitions can be reduced.

If the _tmp partition already exists, then delete once and use after constructing a new filesystem. A password cannot be set for the _tmp partition.

Controlling the Hard Disk Drive and Other Functions

The following functions are provided by the hdd module to control the hard disk drive.

- Obtaining the number of sectors on the entire disk
- Setting idle mode
- Flushing the disk cache
- Powering OFF the device
- Obtaining driver status
- Checking for drive failure

The following functions are also provided:

- Obtaining the size of the largest partition that can be created (in units of sectors)
- Obtaining the number of sub-partitions that have been added
- Obtaining the version of the partition system
- Swapping partition information with the _tmp partition
- Obtaining the available capacity

For details, refer to the function reference.

Chapter 2: CD(DVD)-ROM Library

Library Overview

libcdvd is the basic library for controlling the CD(DVD)-ROM drive. The library provides functions for controlling the CD(DVD)-ROM drive and reading data to EE and IOP memory. libcdvd also includes a function for getting the current date and time from the on-board real-time clock built into the CD(DVD)-ROM drive.

The various functions in libcdvd are implemented for both the EE and the IOP. By using the same function names, the same functions are essentially available both to EE programs as well as to IOP programs.

Related Files

The following files are required to use libcdvd.

Table 2-1

Category	Filename
Header file	libcdvd.h
EE library file	libcdvd.a
IOP library file	cdvdman.ilb
IOP module file	cdvdman.irx
IOP module file	cdvdfsv.irx

The include header file for libcdvd is “libcdvd.h” and is used by both the EE and the IOP.

There are two libcdvd library files. One is “libcdvd.a”, and is used on the EE-side, and the other is “cdvdman.ilb”, used on the IOP-side. The appropriate library file must be linked on the respective side.

Other files that are required at runtime are “cdvdman.irx”, which provides various services of libcdvd for the IOP, and “cdvdfsv.irx”, which calls the various modules of “cdvdman.irx” via requests from the EE.

cdvdfsv.irx can be unloaded to conserve IOP memory. However, after it has been unloaded, libcdvd services will no longer be available from the EE.

Sample Programs

The following libcdvd sample programs are provided for reference:

- **sce/ee/sample/cdvd/smp_ee**
  This program shows how to call basic functions of libcdvd from the EE.
- **sce/iop/sample/cdvd/smp_iop**
  This program shows how to call basic functions of libcdvd from the IOP.
- **sce/iop/sample/cdvd/stmread**
  This program shows how to use a stream function to read a file.
- **sce/iop/sample/cdvd/stmspcm**
  This program shows how to use a stream function for straight PCM playback.
- **sce/iop/sample/cdvd/stmadpcm**
  This program shows how to use a stream function for ADPCM playback.

CD(DVD)-ROM Specifications

Sector Size

The effective user area for data sector reads is 2048 bytes for both DVD-ROMs and standard CD-ROMs. libcdvd also supports the data sizes listed below.

Table 2-2

Disk	Mode	Data Size
DVD-ROM		USER_DATA (2048 bytes)*
CD-ROM	MODE1	USER_DATA (2048 bytes)
CD-ROM	MODE2/FORM1	USER_DATA (2048 bytes)
CD-ROM	MODE2/FORM2	USER_DATA (2324 bytes) + Reserved (4 bytes)
CD-ROM	MODE1	Header (4 bytes) + USER_DATA (2048 bytes) + Parity (288 bytes)
CD-ROM	MODE2	Header (4 bytes) + USER_DATA (2336 bytes)
CD-ROM	MODE2/FORM1	Header (4 bytes) + SubHeader (8 bytes) + USER_DATA (2048 bytes) + Parity (280 bytes)
CD-ROM	MODE2/FORM2	Header (4 bytes) + SubHeader (8 bytes) + USER_DATA (2324 bytes) + Reserved (4 bytes)

* For DVD-ROM, only USER_DATA(2048 byte) is supported.

Rotational Speed and Data Transfer Speed

The PlayStation 2 uses the CAV method in which the rotational speed of the drive is constant, regardless of head position. Therefore, data transfer speed will be different for the inner and outer tracks of the media.

The data transfer speeds of the specification are as follows.

Table 2-3

Head Position	Speed
CD inner track	10x speed = 1500KB/sec (12.3Mbps)
CD outer track	24x speed = 3600KB/sec (28.8Mbps)
DVD inner track	1.6x speed = 2000KB/sec
DVD outer track	4x speed = 5000KB/sec

In practice, approximately 20x (3000KB/sec = 24Mbps) is the upper limit on data transfer speed at the CD outer track because of error retry processing. To read data reliably, use approximately 8Mbps at the CD inner track, approximately 14Mbps at the CD outer track, and approximately 15Mbps for DVD as a criteria for streaming speeds.

The rotational speed can be controlled using the following two methods.

- Streaming mode (SCECdSpinStm): Rotational speed is controlled so that reading can be performed reliably at a constant data transfer rate.
- Normal data mode (SCECdSpinNom): Reading is first attempted at the maximum speed, and if an error occurs, rotational speed is decreased until reading can be performed reliably.

File System

The CD-ROM and DVD-ROM both support the ISO-9660 filesystem. There are some restrictions as to the number of directories, the number of files, and so on, as shown below, but by reducing the number of characters in the file name, etc., it is possible to handle more than the maximum number of files.

Table 2-4

Item	Restriction
Number of directories	40 directories/system
Number of directory levels	8 levels maximum
Number of files	30 files/directory

ISO-9660 also has the restrictions shown below for filenames and directory names.

Table 2-5

Item	Restriction
Valid characters for file and directory names	0 - 9, A - Z, _
Filename format	(8-char filename).(3-char extension);1 * No long filenames allowed
Directory name format	(8-character directory name) * No extensions

CD-XA and CD-DA

CD-XA is not used in PlayStation2 discs. The hardware does not support playback of CD-XA discs.

Likewise, the hardware does not support playback of CD-DA discs. Playback of audio, etc. is performed using SPU streaming, etc.

Access to CD(DVD)-ROM

Specifying Media Type

To prevent an illegal disk from being created, a means is provided for specifying the media type from a program. After initialization has been performed by sceCdInit(), the sceCdMmode() function should be called with the proper media specified as either CD-ROM or DVD-ROM. Then, if a media type is used that differs from this specification, all subsequent CD(DVD)-ROM read processing will fail.

The following example shows typical startup processing for a CD-ROM.

```
#define IOPRP "cdrom0:\\MODULES\\"IOP_IMAGE_FILE";1"

int main()
{
sceSifInitRpc(0);
sceCdInit(SCECdINIT);
sceCdMmode(SCECdCD); /* Media type = CD-ROM */
/* Replace default module */
while ( !sceSifRebootIop(IOPRP) );
while( !sceSifSyncIop() );
/* Reinitialize */
```

```
sceSifInitRpc(0);
sceCdInit(SCECdINIT);
sceCdMmode(SCECdCD); /* Media type = CD-ROM */
sceFsReset();
/* Subsequent code omitted */
```

File Access Procedure

A CD(DVD)-ROM is accessed by specifying a logical sector number, and performing non-blocking processing. The steps in the process are described below.

1. **Get the sector number from the filename**

   Use sceCdSearchFile() to get the logical sector number from the filename. The logical sector number will be stored in lsn of the sceCdlFILE structure. The structure is specified in the first argument of sceCdSearchFile()).

   (Example)

   ```
   ret = sceCdSearchFile(&fp, "\\SYSTEM.CNF;1");
   ```

2. **Begin reading**

   Use a function such as sceCdRead() to begin reading data.

   sceCdRead() is a non-blocking function. Once the command is issued, it will return immediately.

   (Example)

   ```
   ret = sceCdRead(fp.lsn, rsec, (u_int *)bf, &mode);
   ```

3. **Wait for the read to finish**

   Use the sceCdSync() function to wait for the read to complete. The two methods that can be used are blocking and polling.

   (Blocking example)

   ```
   ret = sceCdSync(0);
   ```

   (Polling example)

   ```
   while(sceCdSync(1)) {
         /* Other processing */
   }
   ```

4. **Get error information**

   Use the sceCdGetError() function to see whether or not an error occurred.

   (Example)

   ```
   if (sceCdGetError()==SCECdErNO) /* No error */
   ```

Using Callbacks

A callback can be used to wait for the termination of a non-blocking type function.

First, sceCdCallback() can be used to set a callback function. Then, when a non-blocking type function such as sceCdRead() completes, that callback function will be called together with a parameter indicating which non-blocking function completed.

Streaming Support

To support streaming, a ring buffer is created in IOP memory, and a group of functions is provided to read data from the disk in the background.

Table 2-6

Function	Description
sceCdStInit	Initialize streamer and set buffer
sceCdStStart	Start streaming
sceCdStRead	Read data from buffer
sceCdStPause	Pause streaming
sceCdStResume	Resume streaming
sceCdStSeek	Change stream reading position
sceCdStStat	Get read status to buffer
sceCdStStop	Stop streaming

After calling sceCdStInit() and performing initialization, calling sceCdStStart() will read data in the background, beginning with the specified sector, and store it in the buffer. Then, when sceCdStRead() is called, the data in the buffer can be read sequentially.

Note that the following libcdvd functions cannot be used to access the disk during streaming.

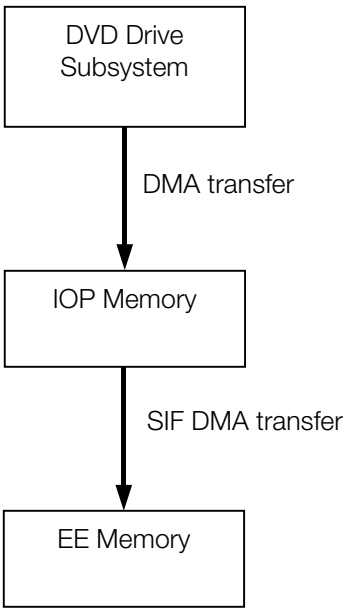
- sceCdChangeThreadPriority()
- sceCdGetToc()
- sceCdInit()
- sceCdMmode()
- sceCdPause()
- sceCdRead()
- sceCdReadChain()
- sceCdReadIOPm()
- sceCdSearchFile()
- sceCdSeek()
- sceCdStandby()
- sceCdStop()
- sceCdSync()
- sceCdDiskReady()
- sceCdGetError()
- sceCdGetReadPos()

Precautions

Differences between the EE and the IOP

libcdvd contains functions that provide essentially the same features, using the same function names, for both the EE and the IOP. However, since the CD(DVD)-ROM drive is connected to the IOP, as shown in the figure, the functions for the EE perform operations indirectly by transferring data read from the IOP via the SIF.

Figure 2-1: libcdvd Data Flow



Since the data flow differs in this manner, the EE library and the IOP library have the following differences:

- On the EE side, the completion of a data read indicates the completion of the transfer of data to EE memory rather than the completion of reading data from the drive.
- On the EE side, the success or failure of commands issued by the various functions to the subsystem are affected by SIF status in addition to the drive status. Attention must also be given to the reentry of an SIF RPC.
- When transferring data to the EE, correction for buffer address alignment may be performed. As much as possible, 64-byte alignment should be maintained.
- When a libcdvd function is called from the EE, the priority of the IOP-side operating module will be set to 81. However, when a libcdvd function is called from the IOP, the priority will be the same as that of the user application. Therefore, on the IOP, programs must be coded to carefully consider the priorities of other threads that are operating.

A libcdvd function called from the EE and a libcdvd function called from the IOP are mutually independent. The processing status of a function that was called from the EE cannot be obtained by calling sceCDSync() from the IOP. In addition, if commands issued for the drive subsystem overlap, libcdvd will consider the second command issue to have failed and will return the error value 0.

Effective Data Reading Speed

In addition to differences in data read speed of the CD(DVD)-ROM between the inner and outer tracks of the media, variations in speed caused by errors from media damage, or an eccentric center of gravity are unpredictable.

Therefore, be aware that there may be situations in which reading speed drops, and avoid programming that directly relies on reading speed.

Precautions Related to RPC Reentry

When called from the EE, almost all libcdvd functions internally use the SIF RPC. Therefore, when a libcdvd function is used with multiple threads, care must be taken that RPC reentry does not occur. RPC reentry is explained in the "SIF System" document. Refer to that document for further information.

RPC WAIT Functions

The following functions execute sceSifBindRpc() or sceSifCallRpc() with WAIT. In addition to being careful concerning RPC reentry, make sure that these functions are not called during an interrupt inhibited state or within the interrupt handler.

- sceCdBreak()
- sceCdDiskReady()
- sceCdGetDiskType()
- sceCdGetError()
- sceCdGetReadPos()
- sceCdGetToc()
- sceCdInit()
- sceCdReadClock()
- sceCdSearchFile()
- sceCdStatus()
- sceCdStInit()
- sceCdStPause()
- sceCdStRead()
- sceCdStResume()
- sceCdStSeek()
- sceCdStStart()
- sceCdStStat()
- sceCdStStop()
- sceCdTrayReq()

RPC NOWAIT Functions

The following functions are those in which the end of processing is detected with sceCdSync(). These functions execute sceSifBindRpc() or sceSifCallRpc() with NOWAIT. In addition to being careful concerning RPC reentry, make sure that these functions are not called within the interrupt handler.

- sceCdReadChain()
- sceCdRead()
- sceCdReadIOPm()
- sceCdSeek()
- sceCdStandby()
- sceCdStop()
- sceCdPause()

RPC Check Function

The following function calls the sceSifCheckStatRpc() function. When using this function, be careful concerning RPC reentry.

- sceCdSync()

Non-RPC Functions

The following functions do not use sceSifBindRpc() or sceSifCallRpc(). They can be used without taking into account RPC reentry. (This does not mean that these functions themselves are reentrant.)

- sceCdCallback()
- sceCdInitEeCB()
- sceCdIntToPos()
- sceCdPosToInt()
- sceCdSync()

Precautions on Processing After cdvdfsv.irx (cdvd_ee_driver) is Unloaded

After cdvdfsv.irx is unloaded, libcdvd functions will no longer be available from the EE. Therefore, standard I/O functions should be used to access a CD(DVD)-ROM from the EE after cdvdfsv.irx is unloaded.

Moreover, the standard I/O function scePowerOffHandler() and the devctl command CDIOC_POWEROFF are used for the sceCdPOffCallback() and sceCdPowerOff() functions that are used when performing power-off processing of the hard disk drive (EXPANSION BAY type).

The method of performing power-off processing after cdvdfsv.irx is unloaded is as follows.

1. Detect interrupt processing using scePowerOffHandler()
2. Close files
3. Turn off dev9 power
4. Turn off PS2 console power by issuing CDIOC_POWEROFF from the standard function sceDevctl()

For details, refer to the standard I/O function reference, CD(DVD)-ROM library reference, and PlayStation File System Overview.

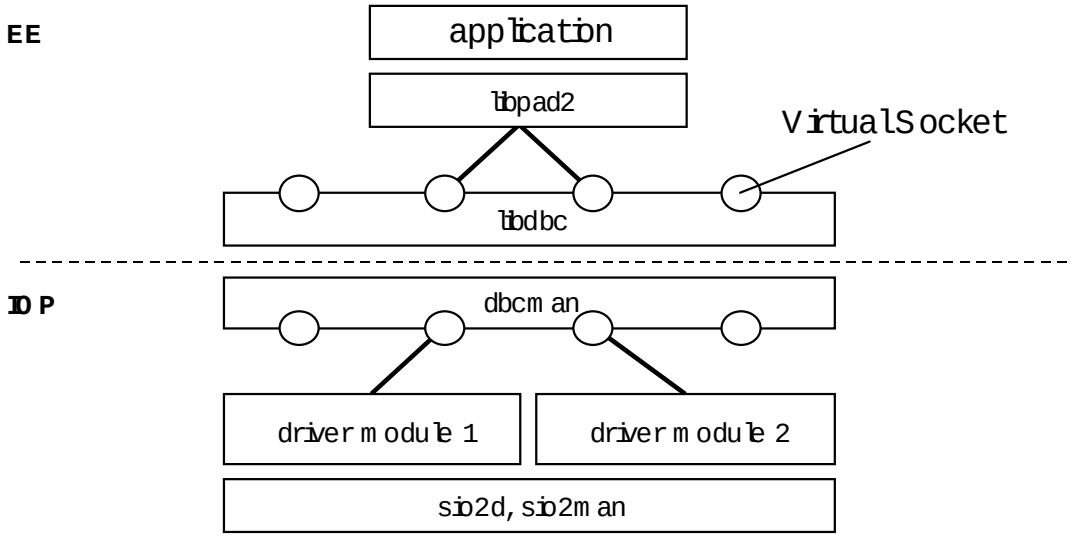
Chapter 3:
Device Control Library

Library Overview

The libdbc library works together with dbcman.irx and is used to bind the libpad2 controller library to controller drivers on the IOP. libdbc creates a virtual socket from a libpad2 request and associates the appropriate controller driver on the IOP with that socket.

libdbc is designed to support other peripheral devices, and is not limited to just controllers. However, it currently only supports libpad2.

Figure 3-1



Related Files

The following files are required to use libdbc.

Table 3-1

Category	Filename
Library file	libdbc.a
Header file	libdbc.h
IOP module file	dbcman.irx

The IOP module must be loaded with sceSifLoadModule() when the EE-side program is started.

Usage

Basic Procedure

Basically, since libdbc can be used only via libpad2, the user program need only perform initialization and termination processing.

1. **Load IOP module**

   Use sceSifLoadModule() to load dbcman.irx.

2. **Initialize library**

   Call sceDbcInit() to initialize libdbc. Then call scePad2Init() after sceDbcInit().

3. **Termination processing**

   If libpad2 is being used, call scePad2End() first. Then call sceDbcEnd() to terminate libdbc.

Chapter 4: Memory Card Library

Library Overview

libmc is provided as a library for controlling PS2 memory cards from the EE.

libmc hides differences in PS2 memory card hardware and provides support for accessing data through the PS2 memory card.

PS2 memory card access is performed via serial communications and generally requires between several and several tens of frames worth of time, depending on the operation. Thus, almost all libmc functions are performed as asynchronous operations. When a function is called, the function returns immediately after registering an operation. Confirmation of completion of an operation and status retrieval are performed by calling sceMcSync().

Related Files

libmc requires the following files.

Table 4-1

Category	Filename
Library file	libmc.a
Header file	libmc.h
Module file	sio2man.irx
	mtapman.irx (only when using a multitap)
	mcxserv.irx
	mcxman.irx

Sample Programs

The following sample programs are provided for libmc:

- **ee/sample/mc/basic**
  Demonstrates basic features of the memory card library
- **ee/sample/mc/icon**
  Linux tool for generating 3D icons

Filesystem Overview

Memory Card Compatibility

In addition to PS2 memory cards, the PlayStation 2 can also handle 128 KB memory cards and the PocketStation.

The table below shows levels of compatibility among the various memory cards.

Table 4-2

Memory Card Type	PS2 Game Titles	Memory Card Browser (PS2 console)	PS Game Titles	Memory Card Management screen (PS console)
PS2 Memory Card	O	O	X	X
PocketStation	Accessible only when PocketStation compatible applications are stored	Can only perform bi-directional copying for PS file format data	O	O
128 KB Memory Card	X	Can only perform bi-directional copying for PS file format data	O	O

* The format used to save PlayStation 2 files to the PS2 memory card will be referred to as the “PlayStation 2 file format". The format used to save the PlayStation data format will be referred to as the “PlayStation file format".

Access to the PocketStation is allowed only when PocketStation-compatible applications are stored. Please use libmcx to take advantage of PocketStation features in PlayStation 2 game titles.

Data from PlayStation 2 game titles cannot be read from or written to 128 KB memory cards.

With the memory card browser screen on the PlayStation 2 console, PlayStation file format data can be copied bi-directionally between a 128 KB memory card or PocketStation and a PS2 memory card. PlayStation file format data copied to a PS2 memory card in this manner can be accessed from a PlayStation 2 game title.

The PlayStation 2 file format is for the memory card (PS2) only. Data files in the PlayStation 2 file format cannot be copied to the PocketStation or a 128 KB memory card using the memory card browser screen.

Filesystem Characteristics

The main characteristics of the PS2 memory card file system used by the PS2 memory card are as follows:

- Support for hierarchical directories
- File management using 1 KB units
- Compatibility with PlayStation 2 file format data, PlayStation file format data (with PocketStation extended file headers), and PlayStation file format data (with memory card file headers)

Clusters

Files on the memory card (PS2) are managed using units known as "clusters".

The size of a cluster is 1024 bytes, and both 1 byte as well as 1024 bytes of data will use 1 cluster. The number of clusters can be determined from the number of data bytes using the following formula:

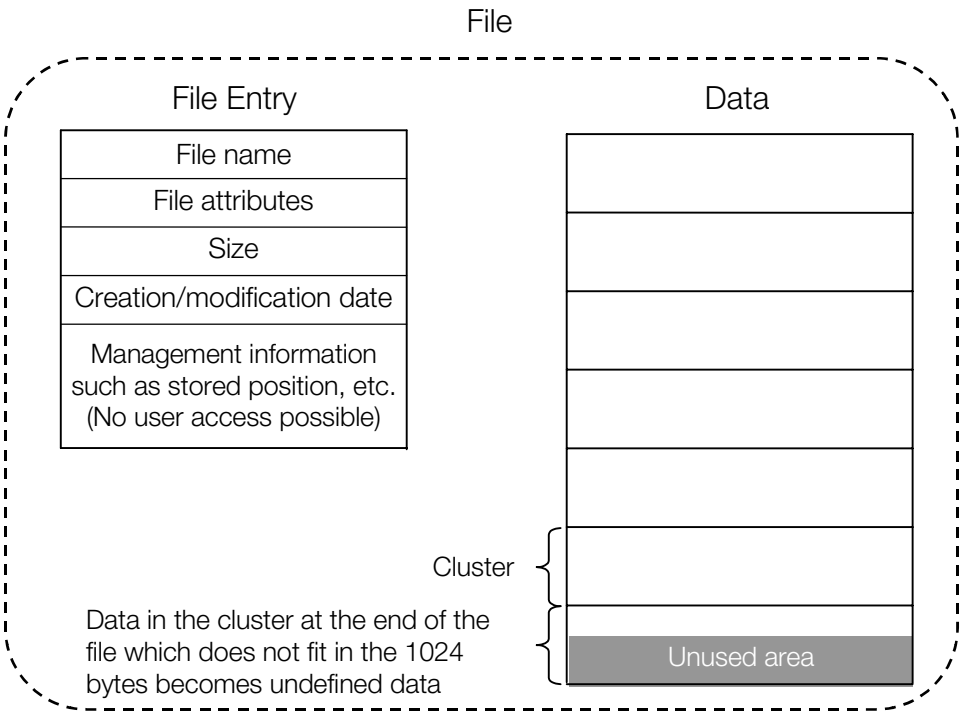
number of clusters = (number of bytes + 1023) / 1024 (remainder is truncated)

Right after formatting, a memory card (PS2) (an empty memory card) has an 8000-cluster capacity. The memory card browser screen on the PlayStation 2 will display "7998 kilobytes", but this is because 2 clusters are subtracted to account for requirements such as directory entries, described later.

File Entries

The data structures containing management information such as filenames, attributes, and creation dates are referred to as file entries. A file on a PS2 memory card consists of a file entry and the data itself. A schematic representation of the file structure is shown below.

Figure 4-1



User-accessible sections of file entries can be read using sceMcGetDir(). The size of a file entry itself is 512 bytes, with two file entries consuming one cluster.

File Attributes

File entries store 16-bit file attributes representing file format, read/write permissions, etc. The valid bits are as follows.

Table 4-3

Attribute	Bit	Macro
Read permission	bit 0	sceMcFileAttrReadable
Write permission	bit 1	sceFileAttrWriteable
Execute permission	bit 2	sceMcFileAttrExecutable
Copy protected	bit 3	sceMcFileAttrDupProhibit
Subdirectory	bit 5	sceMcFileAttrSubdir
Saved	bit 7	sceMcFileAttrClosed
PDA application	bit 11	sceMcFileAttrPDAExec
PlayStation format	bit 12	sceMcFileAttrPS1

File attributes are generated automatically when a file is created based on the access mode specified by sceMcOpen(). Attributes of an existing file can be changed using sceMcSetFileInfo(), but the following attribute changes are prohibited because they can lead to problems.

- Changing attributes on a file created by another title
- Turning off read permission or execute permission on a subdirectory
- Having both files with write permission on and files with write permission off in a single subdirectory

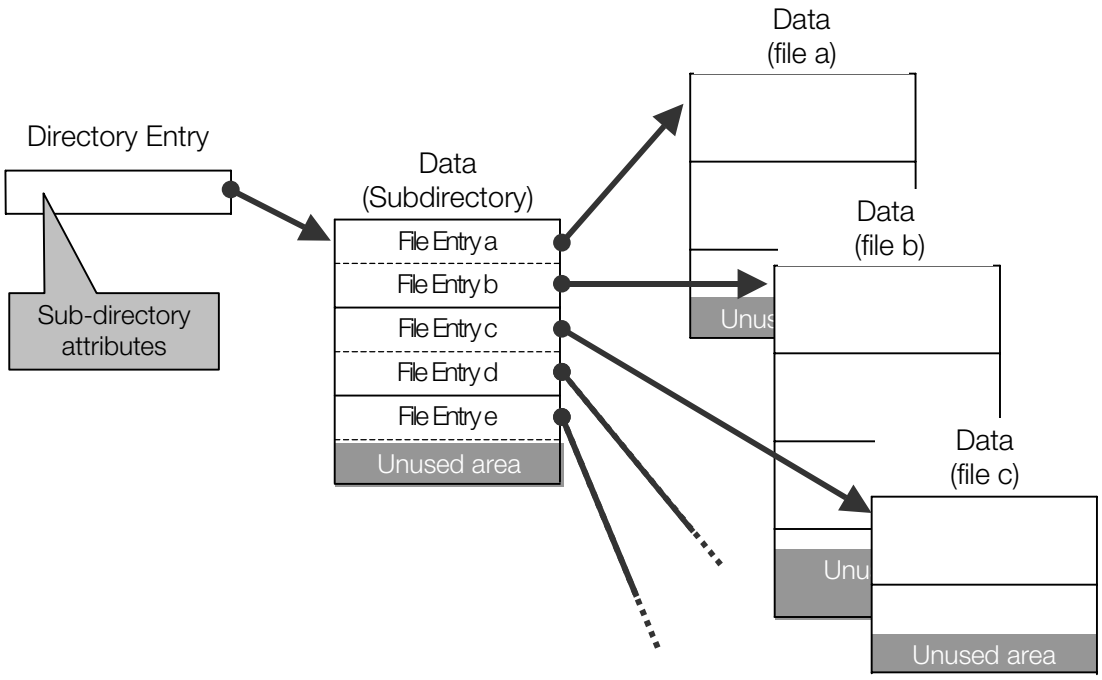
To prevent data from being easily copied using the memory card browser screen on the PlayStation 2, set copy protection on for the subdirectory. This will prevent the entire subdirectory from being copied through the browser screen.

The save completion attribute (sceMcFileAttrClosed bit) becomes 1 when the file write process has completed normally. However, in the browser screen of the domestic version of the PlayStation 2 and the DTL-H10000, this attribute is specified not to be copied when copying a file. Therefore, even if the bit of a specific file is sceMcFileAttrClosed is 0, it is a mistake to assume that saving has failed. To determine whether the file contents are correct or not, implement a mechanism such as a separate checksum in the application.

Structure of the Hierarchical Directory

The hierarchical directory structure is implemented through "files" for which the subdirectory attribute is set. A subdirectory is represented by setting the subdirectory attribute in a file entry (this will be referred to as a directory entry). The corresponding data section will contain the file entries of the files contained in the subdirectory.

Figure 4-2



The PS2 memory card specifications prohibit the creation of multi-level directories in which subdirectories contain more subdirectories.

Calculating Required Space

The space needed to create a file can be calculated as follows.

a. The number of clusters used by individual files

   The following formula is used to determine the cluster count by rounding the size of the file to be created to the nearest 1024 bytes.

   number of clusters = (number of bytes + 1023) / 1024 (remainder is truncated)

b. Number of clusters used in file entry

   The following formula is used to determine the number of clusters used in a file entry.

   number of clusters = (number of files to create + 1) / 2 (remainder is truncated)

c. Number of clusters used in directory entry

   Two clusters are used for the directory entry when a new directory is created. 0 if a file is being added to an existing directory.

The space needed to create files is determined by calculating a. for each file, taking the total, and then adding b. and c. The file size displayed in the PlayStation 2 memory card browser screen is determined by totaling the sizes of the files themselves and then rounding up to the highest cluster. Thus, the value will be smaller than the required number of clusters.

Depending on how the PS2 memory card is used, a previously deleted file entry area can be reused. In such cases, the required number of clusters calculated above will not all be used. The amount of reusable space can be determined using sceMcGetEntSpace(). The browser windows on the DTL-H30000 and later and the overseas PlayStation 2 display the actual cluster counts.

File Creation Rules

Follow the rules described below when creating files on the memory card (PS2). Refer to the libmcx documentation for information on creating files on the PocketStation.

Characters That Can Be Used in Entry Names

With the exception of '*'(0x2a), '/'(0x2f), and '?'(0x3f), characters in the ASCII code range 0x20 - 0x7e can be used as directory names and filenames in the PS2 memory card file system.

Both directory names and filenames are case sensitive.

Title Directories

First, a directory is created in the root directory using an entry name in the following format: "key code" + "product model number" + "arbitrary eight-character string". All data needed for a title should be placed here.

Data files cannot be placed in the root directory. An example of a directory name is BISLPS-00000xxxxxxxx.

The key codes and product model numbers are as follows.

Table 4-4

Key code	Product model number	Meaning
BI	SLPS-xxxxx	Title with SCEI license
BA	SLUS-xxxxx	Title with SCEA license
BE	SLES-xxxxx	Title with SCEE license

When creating a PlayStation format data file with extended memory card file headers for the PocketStation, the '-' (0x2d) in the fifth character of the product model number in the directory name must be replaced with 'P' (0x50).

Any eight-character string can be used at the end. This section should be used to distinguish directories if multiple directories are used in a single title.

The directory name should be null-terminated (0x00).

There is no restriction on the number of directories created at root. Subdirectories cannot be created within a title directory.

Icon Data

In the PlayStation 2 file format, a data file for a 3D icon displayed in the memory card browser screen as a symbol for a title must be placed in the title directory. Generally, the following four files serve as icon data.

Table 4-5

Data	Filename	Notes
Icon management file	icon.sys	Contents described later
List icon data	(arbitrary)	Required
Copying icon data	(arbitrary)	Can be bundled
Deleting icon data	(arbitrary)	Can be bundled

Except for icon.sys, filenames can be any string 31 characters or less. These filenames are indicated in icon.sys. These icon data files can be provided as separate files or can be combined in a single icon data file.

User Data (PlayStation 2 File Format)

In the PlayStation 2 file format, user data such as multiple game data files can be saved in a single directory. Of these, one of the files must have the same name as the directory name. The internal format and data format in the data files are not defined. All filenames must be no longer than 31 characters.

User Data (PlayStation File Format)

Only one file, having the same name as the directory name, can be saved in a title directory as game data in the PlayStation file format. The internal format and data format in the data files must be compatible with the PlayStation file format.

When the memory card browser is used to copy PlayStation file format data from the PocketStation or a 128 KB memory card to a PS2 memory card, directories and files will be created according to these standards.

Maximum Number of Files in a Subdirectory

A file used to save user data to a memory card (PS2) can be icon.sys, an icon file, a file having the same name as the directory name, or another data file, but the total number of these files cannot exceed 18.

In other words, the maximum number of files in a subdirectory is 18.

If there are three icon files, the subdirectory can be as follows:

icon.sys	: one
Icon files	: three
User data with same name as directory name	: one
Other user data	: up to 13

Example of the Standard File Configuration

Taking all of the above into account, a standard file configuration on a memory card (PS2) would be as follows.

```
/
├─ BISLPS-00000xx          .....PS2-formatted directory
│   ─ icon.sys             .....Icon management file
│   ├─ static.ico          .....Icon data file
│   ├─ copy.ico            .....Icon data file
│   ├─ delete.ico          .....Icon data file
│   ├─ BISLPS-00000xx      .....PS2-formatted data file (1st item: same name as directory)
│   ├─ Other user data     .....PS2-format data file (2nd item)
│   ├─ :                   .....PS2-format data file (3rd item)
├─ BISLPS-12345aaa0
│   └─ BISLPS-12345aaa0    .....PS-formatted data file
├─ BISLPSP12345zz
│   └─ BISLPSP12345zz      .....PS-formatted data file (Extended headers for PocketStation)
│
```

Deleting Files

Note the following points when deleting files.

- In the PlayStation 2 file format, icon.sys and icon data files are required. When deleting user data, either delete the entire directory or delete only the user data.
- In the PlayStation file format, there must always be one user data file with the same name as the directory name. When deleting user data, delete the entire directory.

Icon Definition Files

Icon definition files are files used to define icon displays on the PlayStation 2 memory card browser screen. Each title directory must have an icon definition file.

The filename must always be icon.sys.

The structure of icon.sys is shown below.

Table 4-6

Offset	Size(bytes)	Name	Contents
0	4	Header	'P','S','2','D'
4	2	(reserved)	0x0000
6	2	Title name line break position	Character position that becomes start of second line
8	4	(reserved)	0x00000000
12	4	Background transparency	0 (transparent) to 0x80 (opaque)
16	16	Background RGB/upper left	sceVu0IVECTOR, in {r,g,b,-} order; each value in the range from 0 to 0xff
	16	Background RGB/upper right	
	16	Background RGB/lower left	
	16	Background RGB/lower right	
80	16	Light source direction/light source 1	Light source direction vector, sceVu0FVECTOR
	16	Light source direction/light source 2	
	16	Light source direction/light source 3	
128	16	Light source RGB/light source 1	sceVu0FVECTOR, in {r,g,b,-} order
	16	Light source RGB/light source 2	
	16	Light source RGB/light source 3	
176	16	Ambient	sceVu0FVECTOR
192	68	Title name	Up to 32 full-width characters (SJIS only), null terminated
260	64	"List" display icon file name	Up to 63 characters, null terminated
324	64	"Copying" icon file name	Up to 63 characters, null terminated
388	64	“Deleting" icon file name	Up to 63 characters, null terminated
452	512	(reserved)	All 0

- **Title name line break position:**

  Specifies the relative byte position from the beginning of the title name at which a newline is to occur during display. Since the title name consists only of JIS full-width characters, if the newline is to occur at the eighth character, for example, specify 16. The values that can be specified are even numbers from 2 to 32. If any other value is specified, the title may not be displayed properly. Also, since the display area is 16 characters by 2 lines, if there are more than 16 characters in the second line due to the new line position, they will not be able to be displayed.

  If no newline is to occur (the title name fits on one line), specify 32. If 0 is specified, word wrapping will be performed. However, since this function is provided for supporting the 128 KB memory card display, the title name will not necessarily be displayed as expected. Therefore, 0 should not be specified on a PS2 memory card.

- **Background transparency:**

  Specifies the transparency by which the icon list in the background is visible through the background color when an icon is selected.

- **Light-source RGB, ambient:**

  Passed to sceVu0LightColorMatrix() and used in the form of a light color matrix.

- **Title name:**

  Specifies a title name. Up to 32 full-width characters, only non-kanji and Level 1 kanji. 0x84bf through 0x889e cannot be used. '\0' is required at the end, but '\0' is not included in the character count.

- **List icon filename:**

  Specifies a filename for icon data used when the list of files in the memory card is displayed and also when the icon is selected and is displayed larger. Must end with a '\0'.

- **Copying icon, deleting icon:**

  Specifies filenames for icon data used when copy or delete are selected. Must end with a '\0'.

Icon Files

Icon files are simple binary three-dimensional object definition files containing animation information and texture data. In the PS2 memory card browser screen, the icon files specified in icon.sys are read and drawn in an appropriate arrangement.

Geometry Definition

In icon files, all objects are defined as independent triangles. Primitives such as triangle strips and lines are not supported. The number of triangles that can be used in a single icon file depends on the animation settings as shown below.

Table 4-7

Number of MIME key shapes	Maximum number of triangles
1	600
4	500
6	450
8	400

Note: Number of key shapes 2 is not supported.

Models should be designed so that they fit in a cube having sides of 20.0, where x = -10.0 - 10.0, y=0.0 - -20.0, z=-10.0 - 10.0. If this is exceeded, the icon may interfere with adjacent icons on the screen.

MIME Animation

Icon files support vertex coordinate interpolation animation through MIME. The number of key shapes can be 1, 4, 6, or 8. At each frame, animation is performed by weighting shapes using calculations from the key frame and then calculating the model shape.

The key frame must be set to 1 for static icons with no animation. In this case, the settings should be 1 shape, 1st frame, and 1.0 weight.

Shading

Icon shading is calculated using the normal vectors, colors, and texture coordinates assigned to the triangle vertices as well as the flat light sources 1 - 3 and ambient settings defined in icon.sys. Lighting cannot be turned off.

Also, alpha blending and alpha test cannot be used.

Texture

One 16-bit (PSMCT16), 128 x 128 texel texture can be affixed to the icon model.

A bilinear filter is always applied to textures. Also, there will always bemodulation with the polygon color. Texture color cannot be output directly (DECAL).

If a texture is to be pasted, the same texture will be applied to all polygons. If there are polygons for which the texture is not to be applied, the ST of the vertices can be set to the same value and referenced to the same texel.

To reduce the size of an icon file, the texture can be compressed using run-length encoding of half-words (depending on the image, the size may actually increase).

Anti-Aliasing

When drawing icons, the GS must apply AA1 anti-aliasing. This is because anti-aliasing is very effective since icons are drawn in a relatively small area. However, the following restrictions apply. Please adjust models and parameters using these restrictions as a reference.

- Since polygons within a single icon object are not sorted, anti-aliasingmight not be applied between polygons.
- Since AA1 is applied, back-face clipping is performed. This prevents back surfaces, based on the vertex sequence, from being seen. This will happen even if AA1 is not applied, so if the back surface of the polygon is to be displayed, a separate polygon corresponding to the back surface should be defined.
- If a surface is tilted significantly relative to the viewing vector, the z value of a pixel inflated by AA1 may not be properly interpolated. This can result in an erroneous z value, leading to a polygon toward the back being drawn. To prevent this, an operation is provided to clip tilted polygons during drawing. In practice, a polygon is assumed to be tilted if the area of the polygon on the screen is small. The threshold value for the area used in this can be specified in the icon file (BFACE). The larger the value is, the more polygons will be clipped. Try a value of about 1.0 and make adjustments based on the shape of the model. 0.0 will provide standard back-face clipping. For certain model shapes, it may not be possible to completely eliminate this problem.
- Thin or very small polygons should be avoided since they can lead to drawing artifacts caused by AA1.

Data Structures

The data structures used in an icon file are shown below. Data is stored in an actual file by storing each of these sections in the order shown.

In an actual file, the sections are arranged and stored in this sequence. In the model section, triangles are represented by sets of three vertices beginning with the start of the vertex data. The format of the texture section depends on whether or not compression is performed.

Version Header

Table 4-8

Name	Size(bytes)	Contents
VERSION	4 [int]	VERSION No = 0x0001 0000

Model Section

Table 4-9

Name	Size(bytes)	Contents
NBSP	4 [int]	Number of key shapes Any of 1, 4, 6, or 8
ATTRIB	4	Model attributes (ref. attachment)
BFACE	4 [float]	Back-face clipping standard value (usually 1.0)
NBVTX	4 [int]	Number of vertices (multiple of 3)
VTX	8 x NBSP	Vertex coordinates of vertex 1
NORMAL	8	Normal vector
ST	4	Texture coordinates
COLOR	4	Vertex color
VTX	8 x NBSP	Vertex coordinates of vertex 2
NORMAL	8	Normal vector
ST	4	Texture coordinates
COLOR	4	Vertex color
---	---	Repeated NBVTX times

Animation Section

Table 4-10

Name	Size(bytes)	Contents
NBSEQ	4	Number of sequences (Currently, only 1 is supported.)
NBFRAME	4 [int]	Frame length of Sequence 1
SPEED	4 [float]	Play speed magnification
OFFSET	4 [int]	Play offset (frame number)
NBKSP	4	Number of shapes to be used (maximum value: 8)
KSPID	4	Shape number of Shape key 1
NBKF	4	Number of key frames (maximum value: 10)
KEYS	8 x NBKF	Key data (NBKF data values: See separate table)
KSPID	4	Shape number of Shape key 2
NBKF	4	Number of key frames (maximum value: 10)
KEYS	8 x NBKF	Key data (NBKF data values: See separate table)
---	---	Repeated NBKSP times

Texture section (compressed)

Table 4-11

Name	Size(bytes)	Contents
SIZE	4 [int]	Size of compressed texture
TEX	---	Compressed texture data

Texture section (uncompressed)

Table 4-12

Name	Size(bytes)	Contents
TEX	32768	Texture data (PSMCT16 format 128x128 image)

Model attribute: ATTRIB

The model attribute is defined as follows.

The ANTI bit must be set to 1. Also, note that the texture section format depends on the RLE bit setting.

Figure 4-3

Table 4-13

Name	Position	Contents
IIP	0	Shading method 0 : flat shading 1 : gouraud shading
ANTI	1	Anti-aliasing on/off
TEX	2	Texture pasting on/off
RLE	3	Texture pasting on/off 0 : Uncompressed (PSMCT16 raw image) 1 : Perform run-length encoded compression in units of short words

VTX: Vertex Coordinates

For vertex coordinates, (x, y, z) are each represented as 6:10 fixed-point values. 2 bytes of padding are added so that 8 bytes form a set. A vertex is represented by NBSP (shape count) sets of these.

A macro for converting float to 6:10 fixed-point is provided in /usr/local/sce/ee/sample/mc/icon/iconconv.

NORMAL: Normal Vector

For normal vectors, (x, y, z) are each represented as 4:12 fixed-point values (2 bytes). 2 bytes of padding are added, resulting in 8 bytes.

A macro for converting float to 4:12 fixed-point is provided in /usr/local/sce/ee/sample/mc/icon/iconconv.

ST: Texture Coordinates

For texture coordinates, (s, t) are each represented as 4:12 fixed-point values (2 bytes).

COLOR: Vertex Color

Vertex color is represented in 4 bytes, with 1 byte each for r, g, b, and a. The a value is ignored.

KEYS: Key Data

As shown below, key data is formed as an 8-byte data set consisting of a frame number to serve as a key and weights to apply to the shape (the shape specified by the corresponding KSPID) for the frame.

Table 4-14

Name	Size(bytes)	Contents
FRAME	4 [float]	Frame number of key frame
WEIGHT	4 [float]	Shape weight

SIZE: Compressed Texture Size/TEX: Texture Data

For texture data, either a PSMCT16 format 128x128 image can be used asis or data that was compressed according to short word-based Run-Length encoding can be used. The RLE bit of the ATTRIB field indicates whether the data is compressed or uncompressed.

When the data is compressed, set its size in the SIZE field. The SIZE field itself is unnecessary for uncompressed data.

Support Tools

A program to convert an ASCII format containing the necessary data to and from a binary file in the icon format is provided as a tool to support creation of icon files. The tool runs on Linux and the source code is available in /usr/local/sce/ee/sample/mc/icon/iconconv.

Also, a program is provided to compress/expand image data with run-length encoding for creating compressed texture data. This program also runs on Linux and the source code is available in /usr/local/sce/ee/sample/mc/icon/rle.

/usr/local/sce/tools/viewer is also provided as a tool for previewing created icon data. The preview display appears on the DTL-T10000. However, since the display differs somewhat from the actual device, you should save the icon data to a memory card and perform a final verification on the actual device.

Overview of PS2 Memory Card Usage Procedure

The following is a description of the procedure used to handle a PS2 memory card in a program. The initialization operation will depend on whether multitap support is provided.

Initialization (Multitap Not Supported)

1. **Load memory card driver (IOP module)**

   Use sceSifLoadModule() to load sio2man.irx, mcman.irx, and mcserv.irx, in that order. padman.irx can be loaded before or after mcman.irx or mcserv.irx as long as it is loaded after sio2man.irx.

2. **Initialize memory card driver**

   Call sceMcInit().

Initialization (Multitap Supported)

1. **Load memory card driver (IOP module)**

   Use sceSifLoadModule() to load sio2man.irx, mtapman.irx, mcman.irx, and mcserv.irx, in that order. padman.irx can be loaded before or after mcman.irx or mcserv.irx as long as it is loaded after mtapman.irx.

2. **Initialize multitap driver**

   Call sceMtapInit().

3. **Declare use of multitap slot**

   Call sceMtapPortOpen(2 or 3) to declare that multitap access is available through memory card slots 1, 2 on the PS2.

4. **Initialize memory card driver**

   Call sceMcInit().

5. **Check for multitap connection**

   Call sceMtapGetConnection(2 or 3) to see if a multitap is connected (this can be done at any time, not just during initialization).

6. **Check for number of usable slots**

   If a multitap is found to be connected, call sceMcGetSlotMax(0 or 1) to determine the maximum number of slots that is provided by the multitap. The slot numbers used for libmc functions are 0 - (maximum number - 1).

Check for Insertion of PS2 Memory Card

Once initialization has been performed, sceMcGetInfo() must be called to see if a PS2 memory card is inserted.

As with most of the PS2 memory card functions, this function returns immediately after registering the operation. Completion of the operation is determined by polling with sceMcSync(). While waiting for completion, other PS2 memory card operations cannot be registered. Programs should look like as one of the following:

- Synchronous waiting

```
sceMcGetInfo(  ,  ,  ,  ,  );
sceMcSync(0,  ,  );
```

- Asynchronous waiting

```
sceMcGetInfo(  ,  ,  ,  ,  );
while(!sceMcSync(1,  ,  )) {
   /* Other operations */
}
```

With sceMcGetInfo(port, slot, type, free, format), the time it takes for the operation to finish can be reduced by setting unneeded parameters among type, free, or format to 0.

Once the program has checked to see if a PS2 memory card is inserted, other PS2 memory card functions besides sceMcGetInfo() can be used. These functions are used in the same way as sceMcGetInfo(), where operations are registered and completion of the operation is confirmed by polling with sceMcSync().

If a PS2 memory card becomes undetectable during the program and produces an error, the program must wait for a new PS2 memory card to be inserted. If this happens, perform the insertion check once again, described above.

Checking Empty Space

With titles that create files on a PS2 memory card, empty space on a PS2 memory card that is inserted must be checked at start up.

Use the method described previously to calculate the space needed to create files and then compare the results to the free cluster count obtained with sceMcGetInfo().

Formatting Operations

New PS2 memory cards must be formatted when they are first used. If sceMcGetInfo() indicates that the card is unformatted, a message should be displayed on the screen to get user confirmation. The card should then be formatted using sceMcFormat().

Evaluating and Handling Damaged Files

The contents of a file may be saved improperly or the contents of an existing file may be destroyed due to unforseen circumstances such as the PS2 memory card being removed during a write operation. Programs should check to see that file writes are completed successfully and that the contents of a loaded file are correct. If a file is damaged, it should be handled appropriately.

The closed file attribute (sceMcFileAttrClosed) can be used to determineif a write operation has finished. This attribute indicates whether data was successfully written with sceMcWrite() and saved to the PS2 memory card with sceMcClose() or sceMcFlush().

However, with files created or copied via the PlayStation 2 memory card browser screen using libraries from releases up to 1.4.5, the closed file attribute is not set and cannot be used to determine if an existing file is damaged. It is suggested that other methods should be used to determine whether or not a loaded file is damaged, e.g., embedding a checksum.

Notes

Notes Regarding RPC Re-entry

libmc functions use SIF RPC internally. Thus, if multiple threads are used, RPC re-entry must be avoided. Please refer to the discussion on RPC re-entry in the "SIF System" document for more information.

RPC WAIT Functions

For the following function, sceSifBindRpc() / sceSifCallRpc() is executed in WAIT state. Besides being aware of RPC re-entry, make sure that interrupts are disabled and that the function is not called from within an interrupt handler.

- sceMcInit()

RPC NOWAIT Functions

Except for sceMcInit(), all the functions in libmc execute sceSifBindRpc() / sceSifCallRpc() in NOWAIT state. Besides being aware of RPC re-entry, make sure that the functions are not called from within an interrupt handler.

RPC Check Functions

The following function calls sceSifCheckStatRpc(). Avoid RPC re-entry.

- sceMcSync()

Changing Thread Priorities

The default thread priority of the mcserve.irx IOP module for libmc is 104.This priority can be changed when loading the module or during execution.

To indicate a priority when loading the module, use the third parameter of sceSifLoadModule() as shown here.

```
unsigned char *param = "thpri=100";
sceSifLoadModule( "host0:/usr/local/sce/iop/modules/mcserv.irx",
strlen(param)+1, param);
```

Call sceMcChangeThreadPriority() to change priority during execution. For more information, please refer to the function reference.

Notes Regarding File Creation Date and Time

File creation date and time are recorded using Japan Standard Time (JST). Since this is true even for files saved on a PS2 memory card in an export-model Playstation 2, the date and time should be converted to local time as necessary. The system configuration library (libscf) can be used for this conversion. For details, see the libscf document.

Chapter 5: PDA Library

Library Overview

libmcx is provided as a library for controlling PDAs (PocketStations) from the EE. The libmcx library provides support for functions such as retrieving and updating PDA information, retrieving and updating user interface information, and accessing memory.

Serial communication requiring an interval from several frames to several dozen frames depending on the processing is used for starting and exiting applications and for accessing the PDA. As a result, almost all functions of libmcx are processed asynchronously. When a function is called, only its processing is registered and control returns immediately. To verify that processing terminated or to get the processing status, sceMcxSync() is called separately.

Related Files

The following files are required to use libmcx.

Table 5-1

Category	File Name
Library file	libmcx.a
Header file	libmcx.h
Module files	sio2man.irx
	mtapman.irx (only when multitap is used)
	mcxserv.irx
	mcxman.irx

Sample Program

The following sample program uses libmcx:

- **ee/sample/mcx/basic**
  This sample program can be used to verify the basic operations of the PocketStation.

Related Documents

The following files are also available as PDA-related documents. Use them together with this document.

"PDA Specifications"

"PDA Kernel Specifications"

"ARM Architecture Reference Manual"

"File System (FAT)"

Differences with the PDA Library for the PlayStation

The PlayStation 2 PDA library differs from the PlayStation PDA library in that it treats the "PDA application flag" as a PS2 memory card file attribute. That is, with a conventional PlayStation, the McxExecFlag() function is used to access or set the PDA application flag. However, with the PlayStation 2, the sceMcGetDir() or sceMcSetFileInfo() function of libmc is used to access or set the sceMcFileAttrPDAExec bit of the AttrFile member.

PocketStation Overview

The PocketStation is a compact computer that has an ARM7TDMI as the processor core and is equipped with an input/output unit in addition to 128K-bytes of flash memory, 2K-bytes of RAM, and a 32x32-dot monochrome LCD. The PocketStation has functions for saving program files (PDA applications) and executing saved program files by mapping them to an ARM7 virtual address space according to similar procedures as those that are used for PS2 memory cards that are inserted in the memory card slot of the PlayStation 2.

Besides being able to access the PDA as a memory card from the PlayStation 2, PDA-specific information can be read and written, LED lamps and other devices on the PDA can be controlled, and PDA applications can be executed. However, access to the PocketStation by a title application that does not own a PDA application is prohibited.

Hardware Features

The main features of the PDA are shown below.

Table 5-2

Item	Description
Processor	ARM7TDMI
Memory	RAM: 2KB, flash memory: 128 KB
Display units	32x32-dot monochrome LCD display, LEDs
Input devices	5 buttons
Infrared I/O	IrDA mode, remote control mode
Sound	10-bit DAC, internal speaker
Power supply	Internal battery (CR2032), or supplied from PlayStation 2
Power conservation	Operation clock setting function, sleep function for each device
Other	Internal serial number for individual identification

Features of the PDA when it functions as a memory card are shown below.

Table 5-3

Item	Description
Capacity	120K bytes when formatted (accessed in units of 128-byte sectors)
Communication format	Synchronous serial communications using the controller port
Access speed	Maximum speed for continuous reads: Approx. 10K bytes/sec. Cannot be accessed for 20 ms after writing 1 sector.
Other	Hot pluggable (can be inserted and removed without turning off the PlayStation 2) Guaranteed for 100,000 writes

File Creation Rules

Follow the file creation rules presented below when creating files on the PocketStation.

Due to operational regulations, the only files that a PlayStation 2 title application can create on a PocketStation are PDA applications (PocketStation program files). The explanations presented below are basically for the creation of PDA application files by PlayStation 2 titles. Any points mentioned for other files should be considered as reference information.

Filename

The filename, which is based on the product model number of the title application (model number of the first disk if the title application consists of multiple disks), takes the form: "key code" + "product model number" + "arbitrary character string (at most eight characters)."

The key code, which represents a region, is associated with a product model number as follows.

Table 5-4

Key Code	Product	Model No. Object
BI	SLPS-xxxxx	SCEI-oriented license title
BA	SLUS-xxxxx	SCEA-oriented license title
BE	SLES-xxxxx	SCEE-oriented license title

The hyphen (0x2d) in the fifth character of the product model number in the PDA application file name must be replaced with an uppercase "P" (0x50) so that the name becomes SLPSPxxxxx.

The arbitrary character string in the final portion, which consists of at most eight characters, can be freely set. However, the asterisk "*" (0x2a), question mark "?" (0x3f), and null (0x00) cannot be used. When multiple files are created by a single title, use this portion to distinguish among them.

Set a null (0x00) terminator at the end of the filename.

An example is shown below.

Example: When the product model number is SPS-10000, the filename is as follows.

BISLPSP10000PDA

File Header

Use the memory card extended file header when creating a PDA application. Place the following header at the beginning of each file.

Table 5-5

Offset	Size	Contents	Description
0	2	Magic	Always "SC"
2	1	Type Either 0x11, 0x12, or 0x13	PlayStation Number of memory card management screen icon image data values
3	1	Block count	Number of blocks occupied by file
4	64	Document name	Shift-JIS code, up to 32 characters
68	12	Padding	All 0x00
80	2	PDA file list icon image data count	Number of animation count (=n1) displayed when listing files
82	4	File type	"MCX0" or "CRD0"
86	1	Game selection icon page count	Number of animation pages (=n2) in game selection screen
87	1	User-defined device driver entry count	Total number of user-defined device drivers (=n3)
88	4	Reserved	All 0x00
92	4	Program starting address	Absolute address with beginning of file header at 0x02000000
96	32	CLUT	CLUT entries x 16
128	128	PlayStation memory card management screen icon image 1	Required 16 x 16 x 4 bits
(*)	128	PlayStation memory card management screen icon image 2	Only for Type=0x12 or 0x13 16 x 16 x 4 bits
(*)	128	PlayStation memory card management screen icon image 3	Only for Type=0x13 16 x 16 x 4 bits

Offset	Size	Contents	Description
(*)	128**n4*	Device driver entry table	4 bytes x 2 for each device driver (*n4*=(*n3*+15)/16)
(*)	128**n1*	PDA file list icon image data	32 x 32 x 1 bit x *n1*
(*)	128**n5*	Game selection icon entry table	4 bytes x 2 per page of animation (*n5* = (*n2*+15)/16)
(*)	128**n6*	Game selection icon image data	32 x 32 x 1 bit x *n6*. *n6* is the total number of frames per page of game selection icons

(*) Although this varies according to the data contents, icon image data must be aligned with a 128-byte boundary.

Each item is explained in detail below. (Items related to icons are explained together separately.)

Document Name

The document name is the name displayed on the PlayStation 2 memory card management screen. The only characters that can be used are shift-JIS code non-kanji and Level 1 kanji. However, 0x84bf to 0x889e cannot be used. The document name can contain at most 32 full-width characters. If there are less than 32 characters, place a null (0x00) at the end.

File Type

Specify "MCX0" as the file type. This indicates that the file is a PDA application.

Other file types are the default file type "CRD0", which indicates a data file that is supported by the PDA. All other file types indicate data files that are not supported by the PDA.

Program Starting Address

For a PDA application, the file including the header is mapped to virtual addresses 0x02000000 and higher, and execution begins from the address specified here. Therefore, for this address, specify the value obtained by adding 0x02000000 to the offset of the entry point within the file.

No error handling is performed if a non-existent address is specified.

CLUT

This is a color lookup table that is applied to the PlayStation memory card management screen icon images. Each CLUT entry has 5 bits of color information each for RGB as follows.

CLUT entry = (B[4:0] << 10) | (G[4:0] << 5) | R[4:0]

User-Defined Device Driver

A device driver is a PDA-side subroutine for calling special PDA functions from the PlayStation 2. If a device driver number is specified when sceMcxDevRead() or sceMcxDevWrite() is called, control transitions to the corresponding device driver by the callback function of the PDA kernel. For details, refer to "Kernel Service Overview: Communicating with the PlayStation: Device Entry Callbacks" in the "PDA Specification."

For the user-defined device driver entry count (n3), specify the total number of user-defined device drivers. The minimum value is 0, in which case there is no device driver entry table.

For the device driver entry table, specify the data length of the fixed part and the entry point of each device driver in the following format.

Table 5-6

Offset	Size	Contents
0	4	Data length of fixed part for device 128
4	4	Entry point of data passing subroutine for device 128
8	4	Data length of fixed part for device 129
12	4	Entry point of data passing subroutine for device 128l
.....		
(*n3*-1)*8	4	Data length of fixed part for device *n3* + 127
(*n3*-1)*8+4	4	Entry point of data passing subroutine for device *n3* + 127

Note: The numbers 128 and higher are assigned to user-defined device drivers.

The numeric value of each entry point, like the program starting address, is the address that was expanded in memory with the beginning of the file header assigned to 0x02000000. The size of the device entry table is determined in units of 128-bytes. If there are more than 16 drivers, allocate a supplemental table after 128 bytes and use it to store entries 136, 137, ...

Icons

There are three types of icons used by the PDA. These are referred to as PlayStation memory card management screen icons, PDA file list icons, and game selection icons. These are each stored in a predetermined format in the file header. The method of using each type of icon and the corresponding data format are described in detail below.

Icon Types and Features

The features of each of the three types of icons are shown below. The entries enclosed in brackets [] are the related items of the header file.

Table 5-7

Icon Type	Pixel Size	Color	Number of Pages	Number of Frames	Frame Rate (fps)
PlayStation memory card management icon	16 x 16	16 colors out of 32,768	1	1 to 3 [Type]	2 or 3
PDA file list icon	32 x 32	Monochrome	1(0)	0 or more [*n1*]	6
Game selection icon	32 x 32	Monochrome	1 or more [*n2*]	1 or more [total *n6*]	Specified for each page

PlayStation Memory Card Management Screen Icon

A PlayStation memory card management file icon is displayed on the PlayStation 2 memory card management screen. All PDA files have these kinds of icons, not just PDA application files. The image data, which has a size of 16 x 16 pixels and provides 16 colors according to a 4-bit CLUT, is animated at 2 fps when there are 2 frames (Type=0x12) or at 3 fps when there are 3 frames (Type=0x13).

PDA File List Icon

A PDA file list icon is used when listing the files that are stored in the PDA on the LCD display of the PDA. The image data, which has 32 x 32 monochrome pixels, is animated at 6 fps for an arbitrary number of frames.

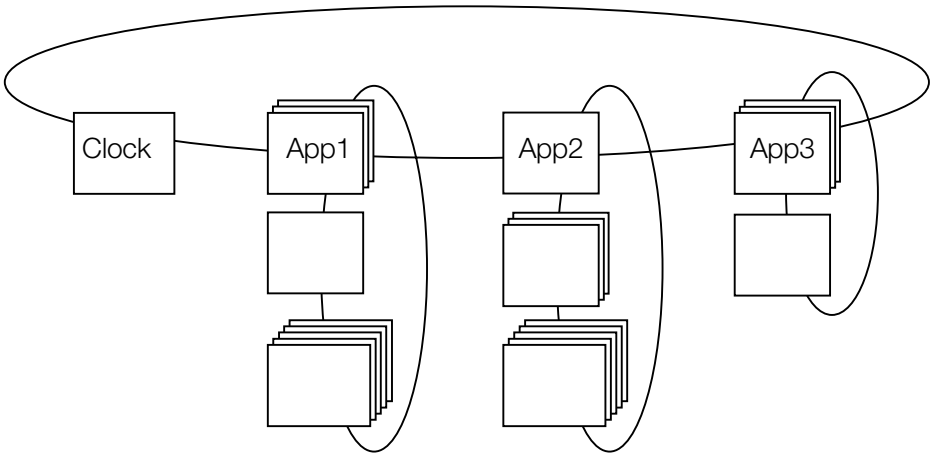
If the number of frames for a PDA application file is 0 (no image data), the first page of the game selection icon is used in place of the PDA file list icon.

Setting 0 for the number of frames for a data file that is supported by the PDA (file type = "CRD0") is prohibited. Such a file always has a PDA file list icon. Also, a data file that is not supported by the PDA does not have a PDA file list icon, and the PlayStation memory card management screen icon is displayed for such a file in place of the PDA file list icon.

Game Selection Icon

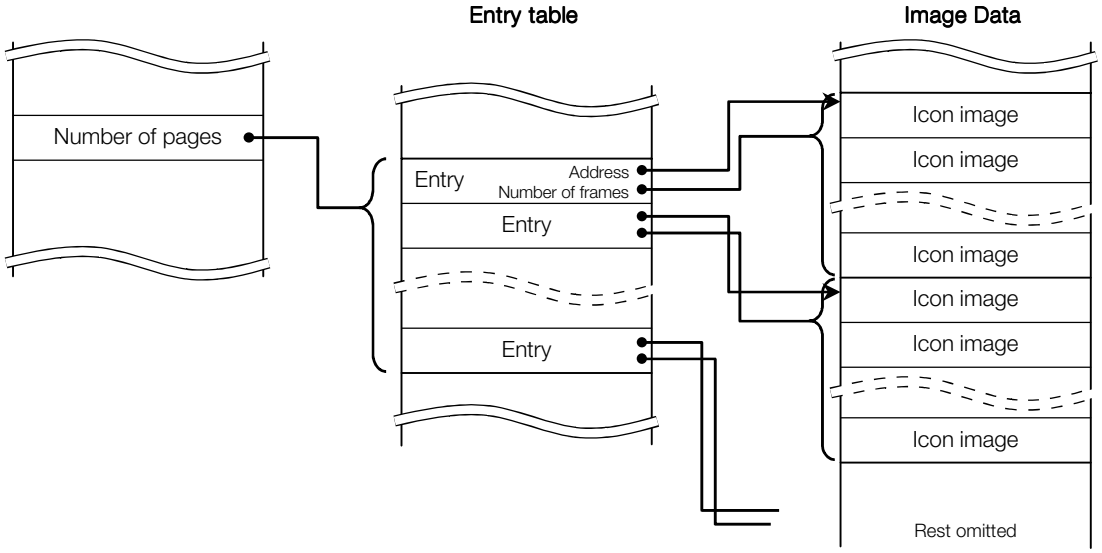
A game selection icon is used by the "game selection function" to display the PDA application files that are stored in the PDA. The image data, which has 32 x 32 monochrome pixels, can have multiple pages. Therefore, it can be used, for example, with the first page for the game title animation, the second page for a game summary animation, and the third page for credits animation. The number of frames and the frame rate of each page can be set independently.

Figure 5-1: Game Selection Function



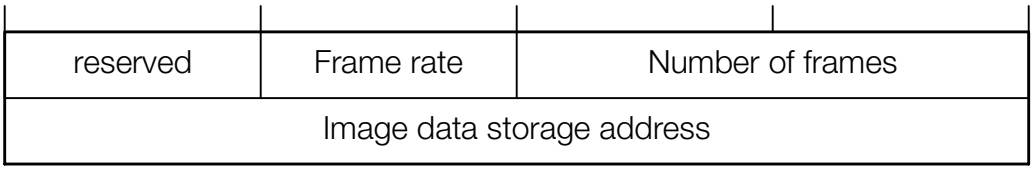
The image data is packed and arranged sequentially beginning with the first frame of the first page, and the number of frames per page, frame rate, and image data position are specified according to the entry table.

Figure 5-2: Game Selection Icon Data Structure



The entry structure is as follows.

Figure 5-3: Game Selection Icon Entry Table Data Structure



For the frame rate, specify the display interval per frame in terms of 1/30-second units. That is, if 30 is specified, 1 frame is displayed every second; if 10 is specified, 3 frames are displayed every second; if 5 is specified, 6 frames are displayed every second; and so on.

For the image data storage address, specify the starting address of the icon image data of the first frame of that page. This address, which is the address that was expanded in memory with the beginning of the file header assigned to 0x02000000, must be aligned on a 128-byte boundary.

Overview of Procedure for Using the PDA

This section describes the processing procedure performed in a program when the PDA is used. The initialization processing differs depending on whether or not the multitap is supported.

Verification of PDA Insertion or Removal

The libmcx library has no function for verifying that the PDA has been inserted or removed. Use the sceMcGetInfo() function of the libmc library.

Initialization (When a Multitap is Not Used)

1. **Load PDA driver (IOP module)**

   Use the sceSifLoadModule() function to load sio2man.irx, mcxman.irx, and mcxserv.irx sequentially in the order listed here. padman.irx, mcman.irx, and mcserv.irx can be loaded either before or after mcxserv.irx and mcxman.irx as long as they are loaded after sio2man.irx.

2. **Initialize memory card driver**

   Call the sceMcxInit() function.

Initialization (When a Multitap is Used)

1. **Load PDA driver (IOP module)**

   Use the sceSifLoadModule() function to load sio2man.irx, mtapman.irx, mcxman.irx, and mcxserv.irx in the order listed here. padman.irx, mcman.irx, and mcserv.irx can be loaded either before or after mcxserv.irx and mcxman.irx as long as they are loaded after mtapman.irx.

2. **Initialize multitap driver**

   Call the sceMtapInit() function

3. **Declare start of multitap slot use**

   Call sceMtapPortOpen(2) or sceMtapPortOpen(3) to declare that memory card insertion slot 1 or 2 of the PlayStation 2 console can be accessed via the multitap.

4. **Initialize memory card driver**

   Call the sceMcxInit() function.

5. **Verify multitap connection**

   Call sceMtapGetConnection(2) or sceMtapGetConnection(3) to verify that the multitap is connected (this can be verified at any time, not only during initialization).

6. **Verify the number of available slots**

   If you know that the multitap is connected, call sceMcGetSlotMax(0) or sceMcGetSlotMax(1) to find the maximum number of slots that the multitap is equipped with. The slot numbers that are accepted by various functions of the libmcx library range from 0 to (maximum number of slots - 1).

Register Processing and Wait for its Completion

Like most memory card control functions, when a PDA library function is called, its processing is only registered, and the PDA waits for the processing to end by using sceMcxSync() to perform polling. No other PDA control processing can be registered while the PDA is waiting. A program can take either of the following forms.

```
[Synchronous waiting]

sceMcxGetInfo ... (  ,  ,  );
sceMcxSync(0,  ,  );

[Asynchronous waiting]

sceMcxGetInfo ... (  ,  ,  );
while(!sceMcxSync(1,  ,  )) {
   /* Other processing */
}
```

Procedure for Saving a PDA Application

The procedure for saving a PDA application in a PDA is almost the same as the procedure for saving a file on a PS2 memory card. However, processing for preventing the use of alternate sectors and processing for setting the PDA application flag are required.

An alternate sector is a separate area that is used in place of a memory card area that cannot be used due to a physical defect. A check for the existence of defects is performed in advance when a memory card is formatted. When an attempt is made to access a defective area, processing is performed to automatically access the alternate sector and read it instead. Using an alternate sector is for a normal data file is not a problem. However, for a PDA application, since the file image is mapped to an ARM7 virtual address space for execution, if a portion is written to an alternate sector, the program code that originally was supposed to be contiguous will have been divided and will no longer be able to be executed. To prevent the use of alternate sectors, call the sceMcxShowTrans() function before writing the file. After the file has been written, you can call the sceMcxHideTrans() function to permit the use of alternate sectors again.

The PDA application flag is defined as a file attribute. After the file is written, use the sceMcSetFileInfo() function of the libmc library to set the sceMcFileAttrPDAExec bit of the file attribute to 1.

The procedure for saving a PDA application is summarized as follows.

1. sceMcxShowTrans(port, slot, 1, TimeOut);
2. Open file("filename");
3. Write file
4. Close file
5. sceMcxHideTrans(port, slot);
6. sceMcSetFileInfo(port, slot, "filename", info, sceMcFileInfoAttr);

Precaution When Formatting

The sceMcFormat() function of the libmc library can be used to format a PDA. However, make sure that no transfer-in-progress indication is displayed at the time.

If the sceMcxShowTrans() function is used to display a transfer-in-progress indication, access to the alternate sector on the PDA will be prohibited until the sceMcxHideTrans() function is used to delete the display. As a result, no alternate sector initialization can be performed, and formatting will fail.

Chapter 6: Multitap Library

Library Overview

libmtap is the library used to control the multitap.

Using libmtap together with the controller library or the memory card library makes it possible to communicate with controllers and PS2 memory cards connected to a multitap.

Related Files

The following files are required for libmtap.

Table 6-1

Category	Filename
Library file	libmtap.a
Header file	libmtap.h
Module file	mtapman.irx
	sio2man.irx
	padman.irx, mcman.irx, mcserv.irx

libmtap is consists of an EE library and an IOP module. To use libmtap, the EE program should load mtapman.irx with sceSifLoadModule(). The loading sequence relative to other associated modules is important. This will be described in detail later.

To link to the EE library, include -lmtap in the link option of the Makefile.

Usage

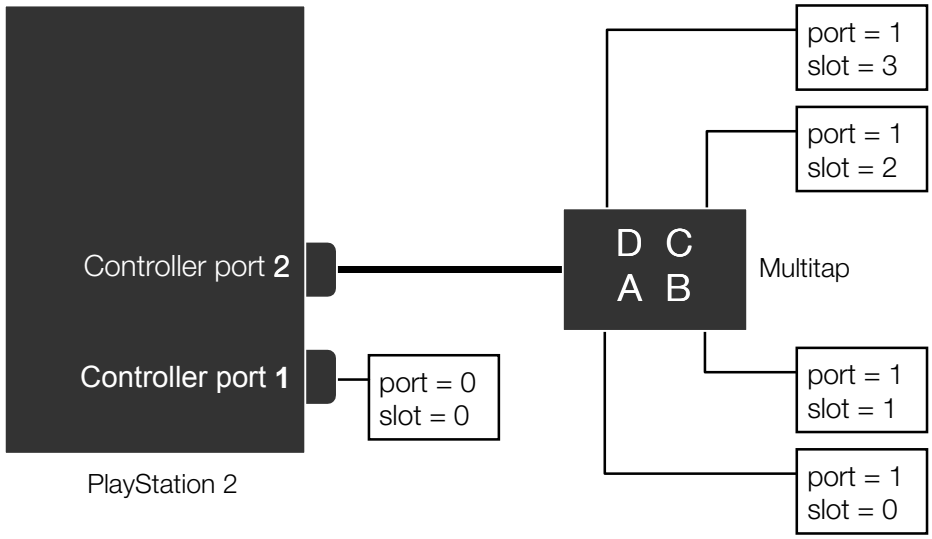
Ports and Slots

To access controllers/PS2 memory cards connected to a multitap, the port and slot arguments of every libpad/libmc function are used to specify the controller/PS2 memory card. port specifies a controller terminal/memory card slot on the PlayStation 2 console. slot specifies a controller terminal/memory card slot on the multitap.

To access a controller/PS2 memory card connected directly to the PlayStation 2 console, specify the corresponding port number in port, and set slot to 0.

To access a controller/PS2 memory card connected through a multitap, specify the appropriate values for both port and slot. See the example, below.

Figure 6-1

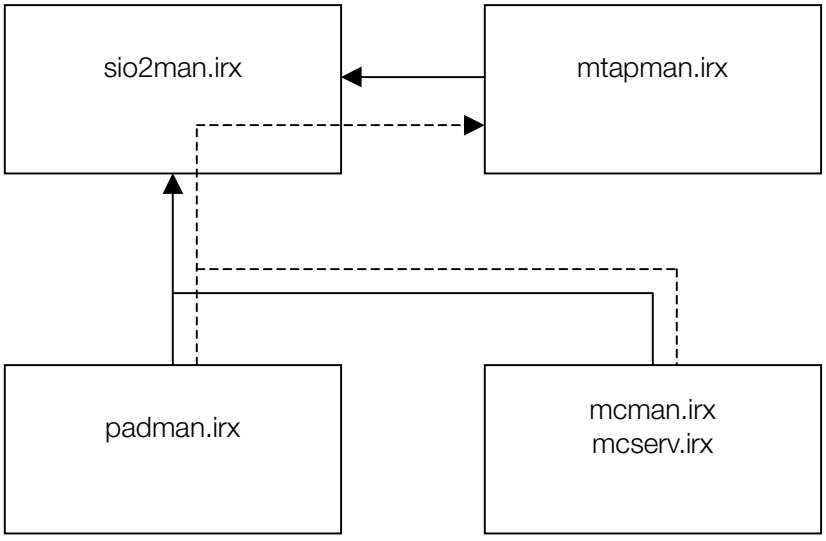


IOP Module Dependencies and Loading Sequence

The IOP modules associated with controller and memory card communications are sio2man.irx, padman.irx, mcman.irx, mcserv.irx and mtapman.irx.

padman.irx, mcman.irx, and mcserv.irx use functions provided by mtapman.irx to communicate with devices on a multitap. The mtapman.irx functions are accessed indirectly using sio2man.irx. Even if mtapman.irx has not been loaded, these modules will still operate properly.

Figure 6-2



Consequently, applications that do not support multitaps do not need to load mtapman.irx.

If an application does provide multitap support, the following modules must be loaded in this sequence.

1. sio2man.irx
2. mtapman.irx
3. padman.irx and mcman.irx / mcserv.irx

The loading sequence of padman.irx and mcman.irx / mcserv.irx is not important.

The loading sequence of mcman.irx and mcserv.irx should follow what is described in the documentation for the memory card library.

Using the Library

The procedure for using libmtap and libpad / libmc to access controllers and PS2 memory cards connected to a multitap is as follows:

1. **Initialize libmtap**

   Call sceMtapInit() to initialize the multitap library.

2. **Initialize libpad / libmc**

   Initialize the controller library / memory card library.

3. **Open the multitap port**

   Call sceMtapPortOpen() to open the port where the multitap should be connected. The opened port will be checked once per frame for the presence of the multitap.

4. **Confirm the number of slots**

   Call scePadGetSlotMax() of the controller library to confirm the number of slots on the multitap connected to the opened port.

5. **Open the controller port / Check memory card status**

   Call scePadPortOpen() to begin communications with the controller. For PS2 memory cards, call sceMcGetInfo() to confirm the PS2 memory card connection state.

Changing the Thread Priorities

The priorities of the threads of the mtapman.irx IOP multitap driver can be changed. This can be done in one of two ways:

- Specify thread priorities at module load time
- Change thread priorities during execution

To specify thread priorities at module load time, pass the priorities as arguments to sceSifLoadModule(). The mtapman.irx module has two module priorities--the one specified first is the priority of the main thread, and the second is the priority of the SIF interface module. An example of how parameters can be specified at module load time is shown below.

```
char arg[] = "thpri=20,46";
sceSifLoadModule( "host0:/usr/local/sce/iop/modules/mtapman.irx",
                                    strlen( arg ) + 1, arg );
```

To change thread priorities during execution, call sceMtapChangeThreadPriority().

In this case, both of the two module priorities will be changed at the same time. For more information, please refer to the function reference.

Notes

Opening Ports and System Overhead

If both 1P and 2P are opened with sceMtapPortOpen(), and scePadPortOpen() is also called for each slot, eight controller ports will be opened. Communication processing takes place for opened controller ports regardless of whether or not controllers are present. Since communication processing is performed asynchronously there is no IOP overhead, however, since the serial interface will be set busy, communication processing with another controller or PS2 memory card will also be affected.

Since this effect seems particularly noticeable for the PS2 memory card, you should call the scePadPortClose() function for ports which do not have a controller connected to reduce the likelihood that PS2 memory card access speed will be affected.

Notes on RPC Re-entry

libmtap functions all execute sceSifBindRpc() / sceSifCallRpc() in WAIT state. Thus, if multiple threads are used, RPC re-entry should be avoided.

Also, interrupts should not be disabled and functions should not be called from within interrupt handlers.

RPC re-entry is discussed in the "SIF System" document.

Chapter 7: Controller Library

Library Overview

libpad is the library used to manage the controllers on the EE. The main features of the library are:

- Getting button data
- Changing the controller mode
- Operating the actuators (vibration)

When used with the multitap library (libmtap), libpad can also manage controllers connected through a multitap.

The current version supports the following PlayStation 2/PlayStation controllers:

- Controller (digital) (ID=4)
- DUALSHOCK, DUALSHOCK 2 (ID=7)
- NeGcon controller (ID=2)
- Analog joystick (ID=5)
- Namco gun controller (ID=6)
- Jog controller, Fishing controller (extended ID)

Support for other controllers will be provided and we plan to support almost all of the PlayStation 2/PlayStation controllers in the future.

Related Files

The following files are needed to use libpad.

Table 7-1

Category	Filename
Library file	libpad.a
Header file	libpad.h
IOP Module files	sio2man.irx
	padman.irx
	mtapman.irx (for multitap support)

The IOP modules should be loaded at startup by the EE program using sceSifLoadModule().

To link to libpad, add "-lpad" to Makefile link options.

Sample Programs

The following libpad sample programs are provided for reference.

- **sce/ee/sample/pad/basic**
  Shows how to use basic functions of the controller library.
- **sce/ee/sample/pad/dual2**
  Shows how to use the pressure-sensitivity feature of the DUALSHOCK 2

- **sce/ee/sample/pad/mtap**
  Shows how to use the multitap library to communicate with up to four controllers on a multitap.
- **sce/ee/sample/pad/gun**
  Shows how to use the gun controller.

Usage

Basic Usage

The basic procedure for operating controllers using libpad is as follows.

1. **Load IOP modules**
   Call sceSifLoadModule() to load sio2man.irx, then padman.irx.
2. **Initialize the library**
   Call scePadInit() to initialize the controller library.
3. **Open the controller port**
   Call scePadPortOpen() to declare the controller port to use. Specify a port number, a slot number (normally 0), and a pointer to a work buffer are in the arguments.
   The work buffer is used for DMA transfers from the IOP. Due to cache issues, it must be 64-byte aligned.
4. **Get button information**
   The library begins communicating with the controller automatically when the controller port is opened. When the controller is in its normal state, controller button information can be retrieved using scePadRead(). Use scePadGetState() to determine whether or not the controller is in its normal state.
5. **Exit operations**
   When it is no longer necessary to communicate with the controller, call scePadPortClose() to close all controller ports, and then call scePadEnd().

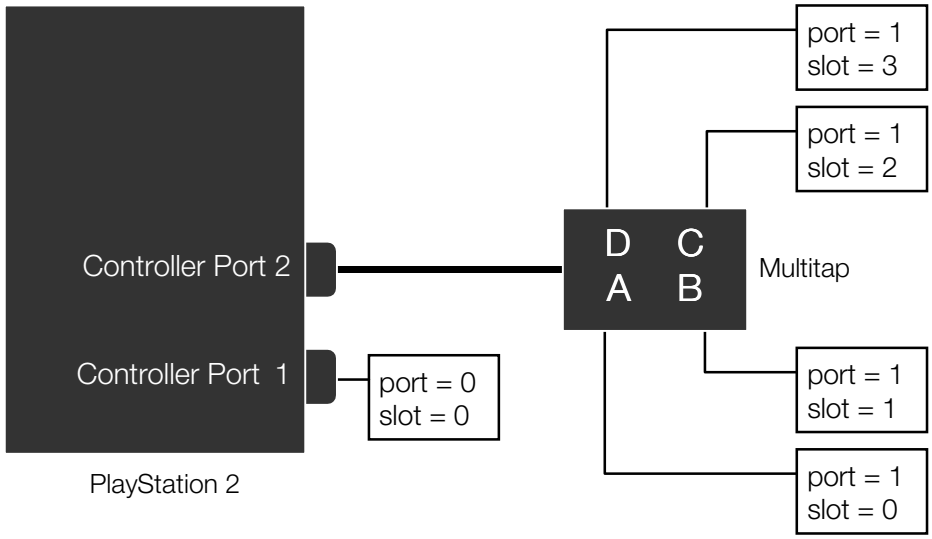
Ports/Slots

Many of the libpad functions use two arguments, port and slot, to specify the target controller. port is used to specify the controller terminal on the PlayStation 2 console, while slot is used to specify a controller terminal on the multitap.

To communicate with a controller connected directly to the PlayStation 2 console, specify the corresponding port number in port, and set slot to 0. To communicate with a controller connected through a multitap, specify the appropriate values for both port and slot.

The figure below shows the port number and slot number used when a multitap is connected to controller terminal 2 on the PlayStation 2 console.

Figure 7-1



Sample Source Code

```
main()
        {
            unsigned char rdata[32];
            u_long128 pad_dma_buff[scePadDmaBufferMax]
        __attribute__((aligned(64)));

            scePadInit();
            scePadPortOpen(0, 0, pad_dma_buff);
            while(1) {
                scePadRead(0,0,rdata);
                .
                .
                .
            }
            scePadPortClose(0,0);
            scePadEnd();
        }
```

Using Multitaps

The multitap library (libmtap) must be used together with libpad to use the PlayStation 2 multitap. Please see the documentation for the multitap library for detailed information on how this is done.

When using a multitap, the slot argument provided in many of the libpad functions is used. Controllers connected to a multitap can be used by initializing the multitap library and specifying the slot number for the slot argument.

Using the Pressure-sensitivity Mode of the DUALSHOCK 2

The DUALSHOCK 2 is a controller in which pressure-sensitivity features have been added to the DUALSHOCK. To use the pressure-sensitivity features of the DUALSHOCK 2, the following procedure needs to be performed to enter pressure-sensitivity mode.

1. Switch to analog mode with scePadSetMainMode().
2. Call scePadInfoPressMode() to check that the controller is the DUALSHOCK 2.
3. Switch to pressure-sensitivity mode by calling scePadEnterPressMode().

In pressure-sensitivity mode, pressure level information is added to the button information of the controller. Apart from the added pressure level data, DUALSHOCK 2 controller information is identical to that of the DUALSHOCK. The DUALSHOCK 2 controller mode ID is 0x79.

Pressure levels are represented as 8-bit values for each button. The values become higher when the button is pressed further. A 0 indicates that the button is not pressed. If the button is pressed, the pressure is registered in any of 255 levels from 1 to 255.

Use scePadExitPressMode() to exit pressure-sensitivity mode.

If the ANALOG mode switch on the controller is pressed in pressure-sensitivity mode (assuming it is not locked), the mode will switch to digital mode. Note that if the ANALOG mode switch is pressed again, the mode will switch to DUALSHOCK analog mode rather than to pressure-sensitivity mode. Switching to pressure-sensitivity mode requires calling scePadEnterPressMode() again.

Actuator Operations

1. Set up alignment information using scePadSetActAlign().
2. Send the vibration value to the controller using scePadSetActDirect().

   The relationship between the vibration value and rotation speed is shown below.

Table 7-2

small motor	0 = stop, 1 = rotate
large motor	0-255 Rotation is faster for higher values

Changing the Thread Priority

The priority of threads used by padman can be either High or Low. The default values are High = 20, Low = 46.

Table 7-3

Type	Priority	Operation
High	20	Communications with controller
Low	46	Other operations

Thread priorities can be set to values other than the defaults by using the third argument of sceSifLoadModule() when loading padman.irx.

The following is an example where settings of High = 32 and Low = 34 are used.

```
char* mes = "thpri=32,34";
sceSifLoadModule( "host0:/usr/local/sce/iop/modules/padman.irx",
strlen(mes)+1, mes );
```

Thread priorities can also be specified in a similar manner for sio2man.irx.

Note: There are no plans to add the ability to change thread priorities dynamically for either padman.irx or sio2man.irx.

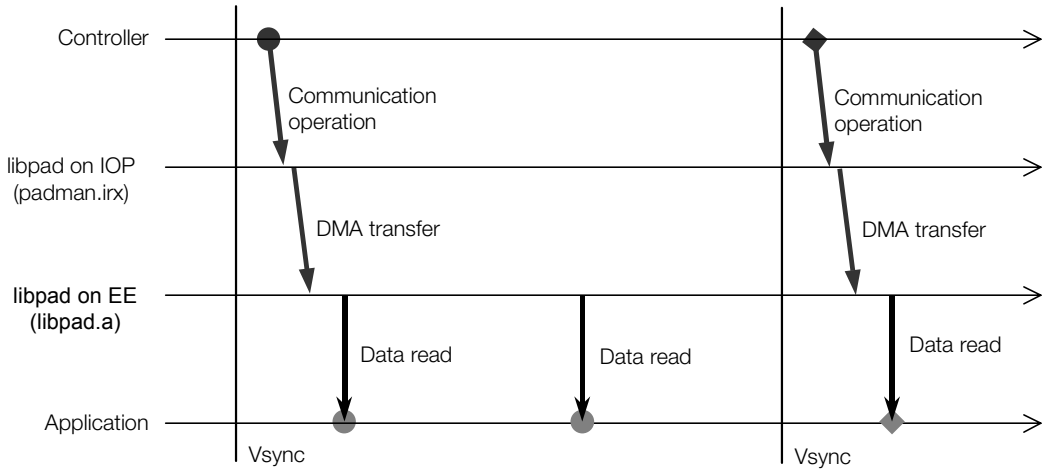
Library Operations

Read Timing for Controller Data

The main operation performed by the controller library - communication with controllers - is performed automatically by padman.irx on the IOP. Communication takes place once per frame via the VBlank interrupt. The communication results are DMA transferred to the main memory of the EE each time and can be retrieved using a function such as scePadRead().

In other words, an application running on the EE can call a libpad function at any time to read controller data, but this data will indicate a past state up to 1 Vsync earlier.

Figure 7-2



There are plans to extend features so that applications can dictate the timing at which communication takes place with the controller. However, at least a one-frame interval will be required between communications with a controller.

Asynchronous Operations

The following functions are implemented as asynchronous functions since they require multiple frames' worth of time for controller-side operations, e.g., scePadSetMainMode(), which sets up the controller mode.

Table 7-4

Function name	Description
ScePadSetMainMode	Switch/lock controller mode
ScePadSetActAlign	Set up contents of actuator parameters
ScePadEnterPressMode	Switch to pressure-sensitivity mode
ScePadExitPressMode	Exit from pressure-sensitivity mode

These functions return immediately after they are called. The completion of the operation should be checked using scePadGetState() or scePadGetReqState().

Notes

Operating Environment

- Libpad has only been tested on the DTL-T10000 development tool. It will not work on other development units (e.g., the EB-2000).
- Libpad cannot be used in conjunction with the pad/pads drivers used by the sceOpen() function in the fileio.irx module. These drivers should not be executed. However, if these drivers are used, padman.irx should not be loaded.

Gun Controllers

When creating applications that support gun controllers, be aware of IOP overhead. Due to its characteristics, gun controllers must complete communication within a VBLANK interval. Depending on overhead caused by threads running on the IOP, processing for the controller module (padman.irx) can be delayed, preventing communications from completing within a VBLANK.

This may result in incorrect coordinate information being obtained. Furthermore, multiple gun controllers cannot be connected to a multitap. To use a gun controller, it must be connected to either 1P/2P of the PS2 or 1-A/2-A of a multitap.

Notes Regarding RPC Re-entry

Libpad functions use the SIF RPC. If multiple threads are used, RPC re-entry must be avoided. Refer to the discussion on RPC re-entry in the "SIF System" document.

RPC WAIT Functions

In the following functions, sceSifBindRpc() / sceSifCallRpc() is executed in WAIT state. Besides taking note of RPC re-entry, make sure that interrupts are disabled and that these functions are not called from within an interrupt handler.

- scePadEnd()
- scePadEnterPressMode()
- scePadExitPressMode()
- scePadGetSlotMax()
- scePadInfoPressMode()
- scePadInit()
- scePadPortClose()
- scePadPortOpen()
- scePadSetActAlign()
- scePadSetMainMode()

Non-RPC Functions

The following functions do not use sceSifBindRpc() / sceSifCallRpc(). These can be used without concern for RPC re-entry issues (this does not mean the functions themselves are re-entrant).

- scePadGetReqState()
- scePadGetState()
- scePadInfoAct()
- scePadInfoComb()

- scePadInfoMode()
- scePadRead()
- scePadReqIntToStr()
- scePadSetActDirect()
- scePadStateIntToStr()

Output Messages

"padman: DMA Busy"

padman may output a "padman: DMA Busy" message.

This may happen in one of the following two cases:

- **SIFDMA (or the entire DMAC) is not responding.**

  This can happen on rare occasions if, for example, Dn_CHCR.STR is lowered during a drawing operation so that DMA stops.

  This can be avoided by using D_ENABLEW to stop all DMA before manipulating the Dn_CHCR.STR bit, then resuming DMA after lowering Dn_CHCR.STR.

- **The EE has stopped**

  If the EE processor encounters a fatal error and halts, SIFDMA from the IOP will not be processed, leading to an error message.

  Also, on rare occasions a CTRL-C from dsedb can cause an error message to be displayed even when the program is stopped.

"PADMAN: * VBLANK OVERLAP ***"**

padman may output a "PADMAN: *** VBLANK OVERLAP ***" message.

This is displayed if communication with a controller could not be complete within a one-frame interval.

More specifically, this error message is displayed if the padman thread could not run due to the overhead from other threads, thus preventing operations from being completed within one frame.

In this case, padman will continue processing to the next frame. This means that the frame in which the error was displayed will not have updated button information for the EE, and the previous button information will be returned to scePadRead() instead.

For titles where controller response has a significant effect on the playability of the game, the overhead from threads operating at a higher priority than padman should be reduced so that they do not drag down padman performance.

"padman: Over Consumpt Max [port][slot]"

This message is displayed when the actuator vibration request to the controller that is connected to the indicated port and slot exceeds the permissible power supply current of the PlayStation 2 console.

For information about the permissible power supply current rating, see "Limitations of analog controllers."

Limitations of Analog Controllers

The following limitations apply when using the analog controller.

Stick center position error

Although the analog controller stick will naturally return to the center position when the stick is released, the stick may actually return to a position offset from true center, depending on where it was released. An application that uses stick position information to determine stick release must take the center position error into account. For DUALSHOCK and DUALSHOCK2, the guaranteed range for returning to the center when the stick is released is 80h +/- 25h.

Number of actuators that can be used simultaneously

The analog controller has a vibration function, however, the number of actuators that can vibrate simultaneously is restricted. This is due to the limitation on power supply current provided by the PlayStation 2 console. When the vibration function is used, the actuators must be controlled according to the following rules.

Actuators must be turned ON such that the total current drain does not exceed 60 units. For actuator 2, a value other than "0x00" is treated as ON. The current drain for each actuator is as follows.

Table 7-5: Current drain for each actuator

Controller	Actuator Type	Current Drain
DUALSHOCK or DUALSHOCK2	Actuator 1	10 units
DUALSHOCK or DUALSHOCK2	Actuator 2	20 units

If by chance, a controller is operated such that the total current drain would exceed 60 units, a limiter internal to the library is activated and the actuator that would cause the current drain to exceed 60 units will not work.

Algorithm for limiting actuator operation due to current drain

Actuator priority is as follows.

1. Port number order
2. Actuator number order

Example 1: When all of the controllers connected to ports 00, 01, 02, and 03 issue operation requests for actuator 1 (20 units)

The current drain is added in port number order. When the resulting current drain exceeds 60 units, the operation requested for that actuator is rejected. Therefore, the actuators for ports 00, 01, and 02 will operate, but the actuator for port 03 will not work.

Example 2: When the following operation is requested

- Port 00 requests actuators 1 and 2 (10 and 20 units)
- Port 01 requests actuator 2 (20 units)
- Port 02 requests actuator 2 (20 units)
- Port 03 requests actuator 1 (10 units)

Since the total current drain for ports 00 and 01 is 50 units, when port 02's request is received, the current drain will exceed 60 units. Therefore, port 02's request is rejected. However, when an attempt is made to add port 03's request next, it will be accepted since the total current drain will still be less than 60 units.

Chapter 8: Controller Library 2

Library Overview

Overview

libpad2 is a library for managing controllers that are used on the EE. When the current libpad is extended in the future, the IOP module size is expected to grow. As a result, libpad2 is implemented in a way such that the module is divided into smaller parts.

The current version of libpad2 supports the (digital) controller, and the DUALSHOCK and DUALSHOCK 2 controllers. In the future, we plan to support all PlayStation 2 / PlayStation controllers.

At present, the multitap is not supported.

Related Files

The following files are required to use libpad2.

Table 8-1

Category	Filname
Library files	libdbc.a
	libpad2.a
Header files	libdbc.h
	libpad2.h
IOP module files	sio2man.irx (Release 2.4 or later)
	sio2d.irx
	dbcman.irx
	Driver modules (***.irx)

IOP modules must be loaded using sceSifLoadModule() when the EE-side program is started. For the driver modules, select and load those modules that are required by the application. The current driver modules are shown below. The following table indicates the compatibility with each controller.

Table 8-2

Module Name	Controller (Digital)	DUALSHOCK	DUALSHOCK 2
ds1u.irx	*1	*1	*2
ds2u.irx	*1	*1	*1
dgco.irx	*1	*3	*3
ds1o.irx	*4	*1	*2
ds2o.irx	*4	*4	*1

*1: Supported *2: DUALSHOCK mode compatible *3: Controller (digital) mode compatible
*4: Not compatible

ds1u.irx and ds2u.irx support the controller (digital), DUALSHOCK, and DUALSHOCK 2. If a DUALSHOCK 2 is inserted, ds2u.irx will automatically switch to pressure-sensitive mode, but ds1u.irx will operate in DUALSHOCK mode. Also, if a controller (digital) or DUALSHOCK is inserted, these modules will operate in controller (digital) mode and DUALSHOCK mode, respectively. In contrast, dgco.irx, ds1o.irx, and ds2o.irx will only operate in specific modes.

dgco.irx will only operate in controller (digital) mode. This permits an application that does not use vibration or pressure-sensitive information to conserve IOP memory by loading dgco.irx.

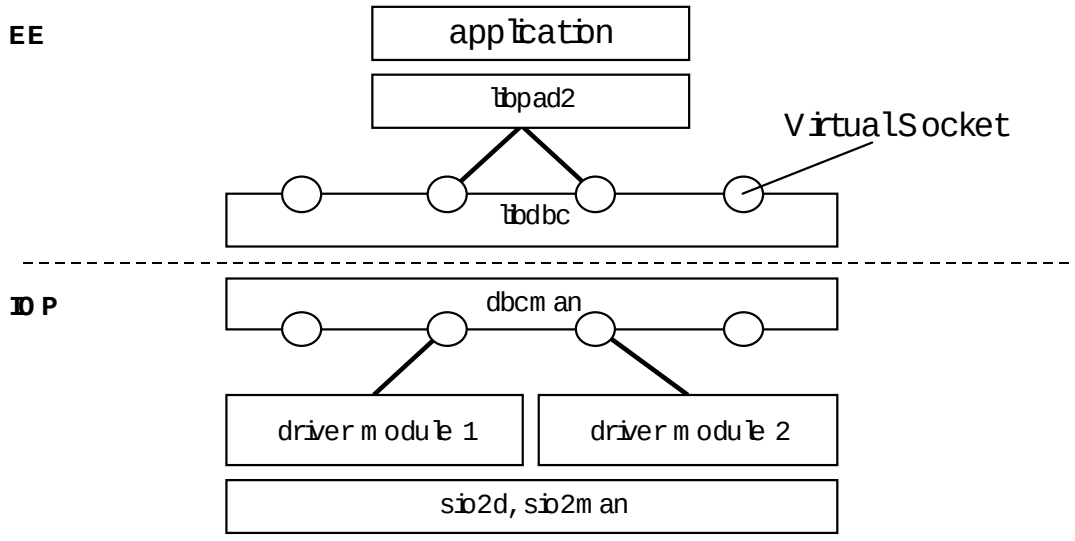
Also, ds1o.irx and ds2o.irx will operate only in DUALSHOCK and DUALSHOCK 2 modes, respectively. By loading ds1o.irx if the application requires only vibration or analog stick information or loading ds2o.irx if the application requires pressure-sensitive information, controllers other than those which are required can be excluded, according to the driver level.

The driver modules can be loaded in this manner to match the needs of the application. In addition, since the number of controllers to be used will vary with the application, driver modules must be loaded only for the number of controllers that will actually be used.

Library Structure

The following figure shows the structure of the library.

Figure 8-1



libdbc is a library for managing inter-SIF communication sockets. libpad2 uses virtual sockets provided by libdbc to communicate with each driver module. The sio2man and sio2d modules for managing sio2 communication ports are also located between the controller and each device driver. The driver modules enable communication between the actual controller and sio2 via these modules.

Sample Programs

The following is provided as a sample program for libpad2. Refer to it as needed.

- **sce/ee/sample/pad2/basic**
  This sample shows basic use of controller library 2.

Usage

Basic Procedure

The basic procedure for using libpad2 to manage controllers is as follows.

1. **Load IOP modules**

   Use sceSifLoadModule() to load sio2man.irx, sio2d.irx, and dbcman.irx sequentially in order. Load the appropriate driver modules as required by the application.

2. **Initialize libraries**

   Call sceDbcInit() and scePad2Init() to initialize libdbc and libpad2.

3. **Create virtual sockets**

   Call scePad2CreateSocket() to create virtual sockets to be used. If you wish to specify a controller port or other information for the sockets, set values for optional conditions in the parameter structure.

   A work buffer, which is an area that is used for DMA transfers from the IOP, must be allocated with 64-byte alignment with respect to the cache.

4. **Get button information**

   When the device driver is loaded and that device driver's controller connection is recognized, a connection is made to the virtual socket that is present in dbc. Then, libpad2 uses that virtual socket to communicate with the controller. If the controller is in a stable state, scePad2Read() can be used to get controller button information. The scePad2GetState() function can be used to check whether or not the controller is in a stable state.

5. **Termination processing**

   When communication with the controller is no longer necessary, call scePad2DeleteSocket() to delete each virtual socket. Then, call scePad2End() to perform termination processing.

Sample Source Code

```
main()
{
    int socket_number;
    unsigned char rdata[32];
    unsigned char profile[10];
    u_long128 pad_buff[SCE_PAD2_DMA_BUFFER_MAX]
__attribute__((aligned(64)));

sceDbcInit();
scePad2Init();
    socket_number = scePad2CreateSocket( NULL, pad_buff );
    while(1) {
        scePad2ButtonProfile( socket_number, profile );
        scePad2Read( socket_number, rdata );
        .
        .
        .
    }
    scePad2DeleteSocket( socket_number );
    scePad2End();
    sceDbcEnd();
}
```

Profiles

Concerning Profiles

A profile lists the features of a controller. That is, it indicates all the detailed information about a connected controller such as the number and types of buttons. Button data is also generated based on the profile. The user can easily learn the state of a specific button by checking the profile and button data.

Profile and Button Data Format

Each bit of profile data describes an individual feature. If the controller has that feature, the corresponding bit is set to 1. If it does not have that feature, the corresponding bit is set to 0. The resulting bit pattern is the controller's profile data.

Button data is arranged so that the button data corresponding to the bits that are 1 in the profile will be pre-packed in order of increasing bit numbers of profile data. The button profile and the profile and button data structures for the controller (digital) and DUALSHOCK 2 are shown below.

Button Profile

The button profile consists of the following four bytes of data (we plan to extend this in the future) with button types assigned to each bit.

Table 8-3

Byte	Bit	Feature	Data size (bits)
0	0	SELECT	1
	1	L3	1
	2	R3	1
	3	START	1
	4	up-arrow	1
	5	right-arrow	1
	6	down-arrow	1
	7	left-arrow	1
1	8	L2	1
	9	R2	1
	10	L1	1
	11	R1	1
	12	Triangle	1
	13	Circle	1
	14	X	1
	15	Square	1
2	16	Stick Rx (analog)	8
	17	Stick Ry (analog)	8
	18	Stick Lx (analog)	8
	19	Stick Ly (analog)	8
	20	right-arrow (analog)	8
	21	left-arrow (analog)	8

Byte	Bit	Feature	Data size (bits)
	22	up-arrow (analog)	8
	23	down-arrow (analog)	8
3	24	Triangle (analog)	8
	25	Circle (analog)	8
	26	Cross (analog)	8
	27	Square (analog)	8
	28	L1 (analog)	8
	29	R1 (analog)	8
	30	L2 (analog)	8
	31	R2 (analog)	8

(Subsequent bytes are undefined)

For Controller (Digital)

Table 8-4: Profile Data

Byte	3	2	1	0
Bit	31 to 24	23 to 16	15 to 8	7 to 0
Profile	00000000	00000000	11111111	11111001

Table 8-5: Button Data

	B7	B6	B5	B4	B3	B2	B1	B0
0Byte	R2	L2	left-arrow	down-arrow	right-arrow	up-arrow	STR	SEL
1Byte			Square	X	Circle	Triangle	R1	L1

STR: Start SEL: Select

For DUALSHOCK 2

Table 8-6: Profile Data

Byte	3	2	1	0
Bit	31 to 24	23 to 16	15 to 8	7 to 0
Profile	11111111	11111111	11111111	11111111

Button Data

Table 8-7

	7bit	6bit	5bit	4bit	3bit	2bit	1bit	0bit
0byte	left-arrow	down-arrow	right-arrow	up-arrow	R3	L3	STR	SEL
1byte	square	X	circle	triangle	R1	L1	R2	L2
2 byte	Stick Rx (analog)							
3 byte	Stick Ry (analog)							
4 byte	Stick Lx (analog)							
5 byte	Stick Ly (analog)							
6 byte	right-arrow (analog)							
7 byte	left-arrow (analog)							
8 byte	up-arrow (analog)							
9 byte	down-arrow (analog)							
10 byte	triangle (analog)							
11 byte	circle (analog)							
12 byte	X (analog)							
13 byte	square (analog)							
14 byte	L1 (analog)							
15 byte	R1 (analog)							
16 byte	L2 (analog)							
17 byte	R2 (analog)							

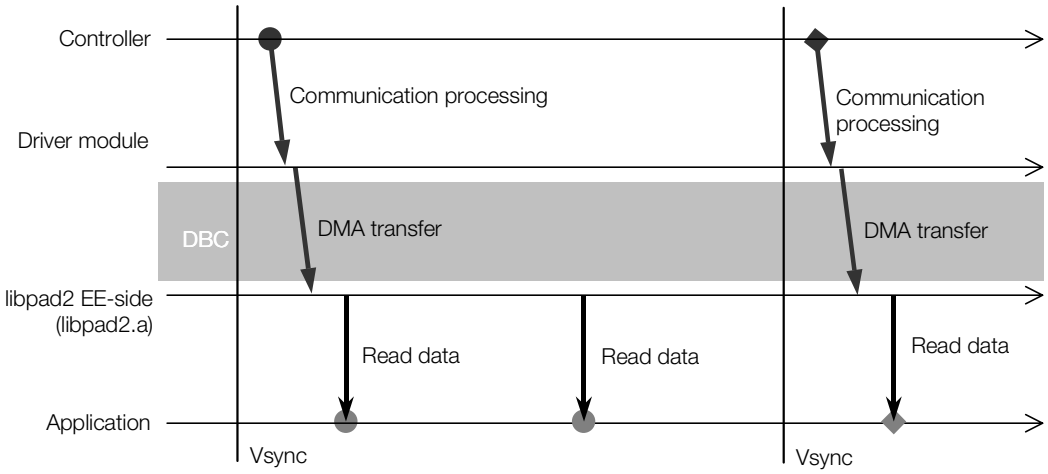
STR:Start SEL:Select

Library Operation

Controller Data Read Timing

The principal processing of the controller library is communication with the controller which is automatically performed by a driver module on the IOP. Communication is performed once per frame using a VBlank interrupt. The result of that communication is DMA transferred each VBlank to EE main memory and it can be obtained using a function such as scePad2Read(). That is, an application that runs on the EE can read the controller data by calling a libpad2 function at arbitrary times, and that data represents the state no earlier than one previous Vsync.

Figure 8-2



Chapter 9: USB Keyboard Library

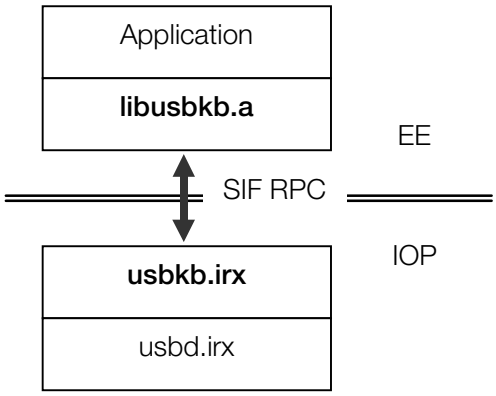
Library Overview

The USB keyboard library, which controls the USB keyboard from an EE application, consists of the EE-side archive file libusbkb.a and the IOP module usbkb.irx. (This library cannot be used from an IOP application.)

Use of this library is not compulsory. An independently developed library may be used for USB keyboard control.

Note: usbkb.irx unloading is supported from release 2.4.

Figure 9-1: Hierarchical Structure of USB Keyboard Library



Supported Device Models

The conditions for keyboards that are supported by the USB keyboard library are as follows:

- Compliant with USB Specification 1.0 or 1.1
- Key arrangement for 101, 104, 106, or 109 specifications of PC-AT compatible computers or Macintosh computers

Note that some keyboards will not work even if these conditions are satisfied, so be sure to confirm operation.

Related Files

The following files are required to use the USB keyboard library.

sce/ee/include/libusbkb.h	(EE external header)
sce/ee/lib/libusbkb.a	(EE library file)
sce/iop/modules/usbkb.irx	(USB keyboard driver)
sce/iop/modules/usbd.irx	(USB driver)

Sample Programs

The following samples are provided as programs that use the USB keyboard library.

- **sce/ee/sample/usb/libusbkb/*** (EE sample programs)

Using the USB Keyboard Library

Main Functions of the USB Keyboard Library

The main functions of the USB keyboard library are shown below.

Abstraction of USB Keyboard Data (character input)

Normally, the USB keyboard returns the key code of all keys that are pressed, as follows:

1. Press a => Key code: a
2. Now press b => Key code: a b
3. Now press c => Key code: a b c

In the USB keyboard library, the following type of abstraction can be performed to simplify character input:

1. Press a => Key code: a
2. Now press b => Key code: b
3. Now press c => Key code: c

Since abstraction is performed in this way, an application can handle the code that is returned by the library as is.

This abstraction function works only when read mode is set to character input mode. Read mode can be set using the sceUsbKbSetReadMode() function.

Key Code Conversion

The key code returned by the USB keyboard is a device code from the original USB specification. The USB keyboard library converts the device code to ASCII while taking into account the state of the Shift and CapsLock keys. The conversion function can also be disabled if necessary.

Key Repeat

Since the USB keyboard does not perform key repeat operations at the hardware level, the USB keyboard library supports key repeat via software.

LED Illumination Control

The NumLock, CapsLock, and ScrollLock LEDs that are installed on the keyboard automatically illuminate or extinguish when their corresponding keys are pressed. The automatic control can be disabled as necessary, and the lighting of these LEDs can be controlled by the application.

The COMPOSE-LED and KANA-LED are currently not subject to this automatic control.

Basic Usage Procedure

The basic procedure for using the USB keyboard library is given below.

1. **Load IOP modules**
   Load and start usbd.irx and usbkb.irx on the IOP.
2. **Initialize library**
   Call sceUsbKbInit() to perform initialization.

3. **Get connection information**

   Call sceUsbKbGetInfo() to get USB keyboard connection information.

4. **Set up keyboard**

   If a newly connected keyboard is found, call sceUsbKbSetLEDStatus(), sceUsbKbSetLEDMode(), sceUsbKbSetRepeat(), sceUsbKbSetCodeType(), sceUsbKbSetArrangement() and sceUsbKbSetReadMode() to set various modes related to that keyboard.
   If necessary, clear the data that has accumulated in the ring buffer using sceUsbKbClearRbuf().

5. **Read keyboard data**

   Call sceUsbKbRead() every 1 to 2 VSync to read keyboard data.

   If a read error (disconnection) occurs during the processing in step 5, return to step 3 and perform that step again. To support dynamically connected keyboards, it may be necessary to always monitor the connection information using sceUsbKbGetInfo().

   The various APIs of the USB keyboard library are not thread-safe. To operate the keyboard from multiple threads, use semaphores or some other means to perform exclusive control.

Starting Up usbkb.irx

usbkb.irx Syntax

Syntax

usbkb.irx [keybd=1-127] [debug=0|1] [lmode=AUTOLOAD|TESTLOAD]

Options

keybd Specifies the maximum number of keyboard connections. The default is 2.

debug Specifies the data contents that are output by the printf statement for debugging.

0: Title and fatal errors (default)

1: Packets sent from the USB keyboard

lmode This is an option that is used by the USB module autoloader (usbmload.irx) and normally should not be set. For details, refer to the USB module autoloader document.

Startup Procedure

Generally, this module should be started up as follows from the EE.

```
#define USBKEYBD_ARG “keybd=8\0debug=1”
char *option = USBKEYBD_ARG;
while(sceSifLoadModule(“host0:/usr/local/sce/iop/modules/usbkb.irx”,
sizeof(USBKEYBD_ARG)+1,option) < 0)
{
printf(“Can’t load module usbkb\n”);
}
```

Key Code Conversion Function

The USB keyboard library has a function that converts the device key code that is returned by the USB keyboard to ASCII while taking into account the state of the Shift key and the like. Details about this key code conversion function are explained below.

Key Code Definitions

The USB keyboard library handles key codes as u_short values whose bit positions are defined as shown below.

Table 9-1

Bit Position	Contents
bits 0-7	Code (ASCII code or device key code)
bits 8-13	(reserved)
bit 14	Numeric key pad flag 0 Normal key 1 Numeric key pad key
bit 15	Conversion flag 0 ASCII code 1 Device key code of the USB specification

Bits 0 to 7 contain either the converted ASCII code or raw device key code, and bit 15 indicates which of these two codes is used.

Bit masks for the numeric key pad flag and conversion flag are defined as follows in libusbkb.h.

```
#define USBKB_RAWDAT 0x8000 /*Code that cannot be converted to ASCII code*/
#define USBKB_KEYPAD 0x4000 /*Key-pad code*/
```

Original USB Specification Key Codes

For original USB specification key codes (raw key codes), refer to page 35 in “HID Usage Tables Document 1.1,” which is available at http://www.usb.org/developers/devclass_docs.html#approved.

Key Codes That Cannot be Converted

Regardless of the conversion mode setting, for a key code that has no corresponding ASCII code, bit 15 is set to 1 and the device key code from the original USB specification is placed in bits 0 to 7.

Some of the key codes that cannot be converted are defined as follows in libusbkb.h.

```
#define USBKEYC_NO_EVENT         0x00
#define USBKEYC_E_ROLLOVER       0x01
#define USBKEYC_E_POSTFAIL       0x02
#define USBKEYC_E_UNDEF          0x03
#define USBKEYC_ESCAPE           0x29
#define USBKEYC_106_KANJI        0x35   /* Half-width/full-width kanji */
#define USBKEYC_CAPS_LOCK        0x39
#define USBKEYC_F1               0x3a
#define USBKEYC_F2               0x3b
#define USBKEYC_F3               0x3c
#define USBKEYC_F4               0x3d
#define USBKEYC_F5               0x3e
#define USBKEYC_F6               0x3f
#define USBKEYC_F7               0x40
#define USBKEYC_F8               0x41
#define USBKEYC_F9               0x42
#define USBKEYC_F10              0x43
#define USBKEYC_F11              0x44
#define USBKEYC_F12              0x45
#define USBKEYC_PRINTSCREEN      0x46
#define USBKEYC_SCROLL_LOCK      0x47
#define USBKEYC_PAUSE            0x48
#define USBKEYC_INSERT           0x49
```

```
#define USBKEYC_HOME            0x4a
#define USBKEYC_PAGE_UP         0x4b
#define USBKEYC_DELETE          0x4c
#define USBKEYC_END             0x4d
#define USBKEYC_PAGE_DOWN       0x4e
#define USBKEYC_RIGHT_ARROW     0x4f
#define USBKEYC_LEFT_ARROW      0x50
#define USBKEYC_DOWN_ARROW      0x51
#define USBKEYC_UP_ARROW        0x52
#define USBKEYC_NUM_LOCK        0x53
#define USBKEYC_APPLICATION     0x65   /* Application key */
#define USBKEYC_KANA            0x88   /* Katakana, hiragana, romaji */
#define USBKEYC_HENKAN          0x8a   /* Previous candidate, convert, all
                                          candidates */
#define USBKEYC_MUHENKAN        0x8b   /* No conversion */
```

Typical Raw Codes

When CODETYPE_RAW is set using sceUsbKbSetCodeType(), it is necessary to deal with the raw code that is returned by the USB device. The following are typical raw codes, and are defined in "libusbkb.h".

Use this information together with the information in the above section, "Key Codes That Cannot be Converted."

```
#define USBKEYC_A          0x04
#define USBKEYC_B          0x05
#define USBKEYC_C          0x06
#define USBKEYC_D          0x07
#define USBKEYC_E          0x08
#define USBKEYC_F          0x09
#define USBKEYC_G          0x0A
#define USBKEYC_H          0x0B
#define USBKEYC_I          0x0C
#define USBKEYC_J          0x0D
#define USBKEYC_K          0x0E
#define USBKEYC_L          0x0F
#define USBKEYC_M          0x10
#define USBKEYC_N          0x11
#define USBKEYC_O          0x12
#define USBKEYC_P          0x13
#define USBKEYC_Q          0x14
#define USBKEYC_R          0x15
#define USBKEYC_S          0x16
#define USBKEYC_T          0x17
#define USBKEYC_U          0x18
#define USBKEYC_V          0x19
#define USBKEYC_W          0x1A
#define USBKEYC_X          0x1B
#define USBKEYC_Y          0x1C
#define USBKEYC_Z          0x1D
#define USBKEYC_1          0x1E
#define USBKEYC_2          0x1F
#define USBKEYC_3          0x20
#define USBKEYC_4          0x21
#define USBKEYC_5          0x22
#define USBKEYC_6          0x23
#define USBKEYC_7          0x24
#define USBKEYC_8          0x25
#define USBKEYC_9          0x26
#define USBKEYC_0          0x27
```

```
#define USBKEYC_ENTER       0x28
#define USBKEYC_ESC         0x29
#define USBKEYC_BS          0x2A
#define USBKEYC_TAB         0x2B
#define USBKEYC_SPACE       0x2C
#define USBKEYC_MINUS       0x2D
#define USBKEYC_EQUAL_101             0x2E  /* = and + */
#define USBKEYC_ACCENT_CIRCONFLEX_106 0x2E  /* ^ and ~ */
#define USBKEYC_LEFT_BRACKET_101      0x2F  /* [ */
#define USBKEYC_ATMARK_106            0x2F  /* @ */
#define USBKEYC_RIGHT_BRACKET_101     0x30  /* ] */
#define USBKEYC_LEFT_BRACKET_106      0x30  /* [ */
#define USBKEYC_BACKSLASH_101         0x31  /* \ and | */
#define USBKEYC_RIGHT_BRACKET_106     0x32  /* ] */
#define USBKEYC_SEMICOLON             0x33  /* ; */
#define USBKEYC_QUOTATION_101         0x34  /* ' and " */
#define USBKEYC_COLON_106             0x34  /* : and * */
#define USBKEYC_COMMA       0x36
#define USBKEYC_PERIOD      0x37
#define USBKEYC_SLASH       0x38
#define USBKEYC_CAPS_LOCK   0x39
#define USBKEYC_KPAD_NUMLOCK   0x53
#define USBKEYC_KPAD_SLASH     0x54
#define USBKEYC_KPAD_ASTERISK  0x55
#define USBKEYC_KPAD_MINUS     0x56
#define USBKEYC_KPAD_PLUS      0x57
#define USBKEYC_KPAD_ENTER     0x58
#define USBKEYC_KPAD_1         0x59
#define USBKEYC_KPAD_2         0x5A
#define USBKEYC_KPAD_3         0x5B
#define USBKEYC_KPAD_4         0x5C
#define USBKEYC_KPAD_5         0x5D
#define USBKEYC_KPAD_6         0x5E
#define USBKEYC_KPAD_7         0x5F
#define USBKEYC_KPAD_8         0x60
#define USBKEYC_KPAD_9         0x61
#define USBKEYC_KPAD_0         0x62
#define USBKEYC_KPAD_PERIOD    0x63
```

KEY OFF Code

The key code with numeric value zero, which is called the KEY OFF code, is generated in the following situations.

- When only a Modifier Key (Shift, Ctrl, Alt, or Win) is pressed
- When a key is released

It is not possible to determine the key that was released from the KEY OFF code.

Exceptional Key Code Conversions

The ASCII code that is obtained as the result of a key code conversion basically is the character that is printed on the key top. However, there are several keys for which exceptional conversions are performed.

- **\ key**

  The \ (backslash) key of the 106-key keyboard is not converted in order to distinguish it from the “Yen” key. However, if this key is pressed together with the Shift key, it is converted to “_” (underscore).

- **Keys that are converted to control codes**

Tab, Enter, Backspace are converted to “\t”, “\n”, and “\b”.

- **Numeric key pad**

  For a key other than the NumLock key, the key code that is obtained is changed depending on the NumLock state.

  If NumLock is ON, each key is converted to the ASCII code of the corresponding numeric digit or symbol, and USBKB_KEYPAD, which indicates the numeric key pad, is set to 1. If NumLock is OFF, the code is converted only for the following keys.

Table 9-2

Key	Obtained Key Code
Home	USBKEYC_HOME | USBKB_KEYPAD | USBKB_RAWDAT
End	USBKEYC_END | USBKB_KEYPAD | USBKB_RAWDAT
PageUp	USBKEYC_PAGE_UP | USBKB_KEYPAD | USBKB_RAWDAT
PageDown	USBKEYC_PAGE_DOWN | USBKB_KEYPAD | USBKB_RAWDAT
Insert	USBKEYC_INSERT | USBKB_KEYPAD | USBKB_RAWDAT
Delete	USBKEYC_DELETE | USBKB_KEYPAD | USBKB_RAWDAT
Up arrow	USBKEYC_UP_ARROW | USBKB_KEYPAD | USBKB_RAWDAT
Down arrow	USBKEYC_DOWN_ARROW | USBKB_KEYPAD | USBKB_RAWDAT
Right arrow	USBKEYC_RIGHT_ARROW | USBKB_KEYPAD | USBKB_RAWDAT
Left arrow	USBKEYC_LEFT_ARROW | USBKB_KEYPAD | USBKB_RAWDAT

Keyboard No.

The USB keyboard library manages more than one connected USB keyboard using the keyboard No.

The keyboard No. can be a value from 0 to (maximum number that can be connected - 1).

Basically, the keyboard No. is determined by the enumeration order. Although this is also said to be the order in which the USB keyboards were connected, it is not really that simple, as described below.

When more than one USB device is connected before usbkb.irx is started up

Since the devices are enumerated in random order according to the hardware specification, the keyboard No. of each keyboard is indeterminate.

When a separate keyboard is connected to a port to which a keyboard had been previously connected

The keyboard No. will be the same as the No. that was assigned to the previous keyboard.

However, when there are more physical ports than the maximum possible number of connections, the No. will not necessarily be the same. An example of this situation is described below.

1. Assume that the maximum possible number of connections is 2. That is, only 0 or 1 can be used as a keyboard No.
2. Connect a 4-port hub to the root hub (the hub that is built into the PS2).
3. Connect keyboard A to port 1 of the 4-port hub. The keyboard will be assigned No. 0.
4. Connect keyboard B to port 2 of the 4-port hub. The keyboard will be assigned No. 1. At this time, the connections and keyboard numbers are as follows.

   Hub port 1 = Keyboard No. 0 (keyboard A)

   Hub port 2 = Keyboard No. 1 (keyboard B)

Hub port 3 = None

5. Remove keyboard A from the hub.
6. Connect keyboard A to port 3 of the 4-port hub. At this time, since the only free keyboard No. is 0, keyboard No. 0 is assigned to port 3. At this time, the connections and keyboard numbers are as follows.

   Hub port 1 = None

   Hub port 2 = Keyboard No. 1 (keyboard B)

   Hub port 3 = Keyboard No. 0 (keyboard A)
7. Remove keyboard B from the hub.

   Connect keyboard B to port 1 of the 4-port hub. At this time, since the only free keyboard No. is 1, keyboard No. 1 is assigned to port 1. At this time, the connections and keyboard numbers are as follows.

   Hub port 1 = Keyboard No. 1 (keyboard B)

   Hub port 2 = None

   Hub port 3 = Keyboard No. 0 (keyboard A)

As can be seen, even if a keyboard is connected to the same port, it will not necessarily have the same keyboard No.

This problem can be avoided by setting the maximum possible number of connections to a sufficiently large value.

For example, if the maximum possible number of connections is set to 32, the number of physical ports cannot actually be larger than this number.

Precautions When Using the Keyboard as a Keypad

Network game specifications may require a situation in which a keyboard is to be used for chatting in conversational mode and the same keyboard is to be used for motion in motion mode.

Since this library provides two data reading modes, namely character input mode and packet mode, this situation can be handled (see the sceUsbKbSetReadMode() function in usbkb_rf.doc).

However, when the keyboard is used as a keypad, please realize that a USB keyboard is not very good at handling simultaneous key presses.

Although simultaneous key presses of up to two keys can be detected accurately for a USB keyboard, a RollOverError sometimes occurs for a key press combination of three simultaneous keys.

The problem is that the specification that indicates which combination of simultaneous key presses will cause a RollOverError differs for each USB keyboard.

Therefore, be sure to check the key arrangement specification very carefully.

Chapter 10: Vibration Library

Library Overview

libvib is a library for controlling the vibration of a controller. It differs from libpad2 in that libpad2 is mainly in charge of managing controller information, whereas libvib does not provide that functionality. Normally, libvib should be used together with libpad2.

Related Files

The following files are required to use libvib.

Table 10-1

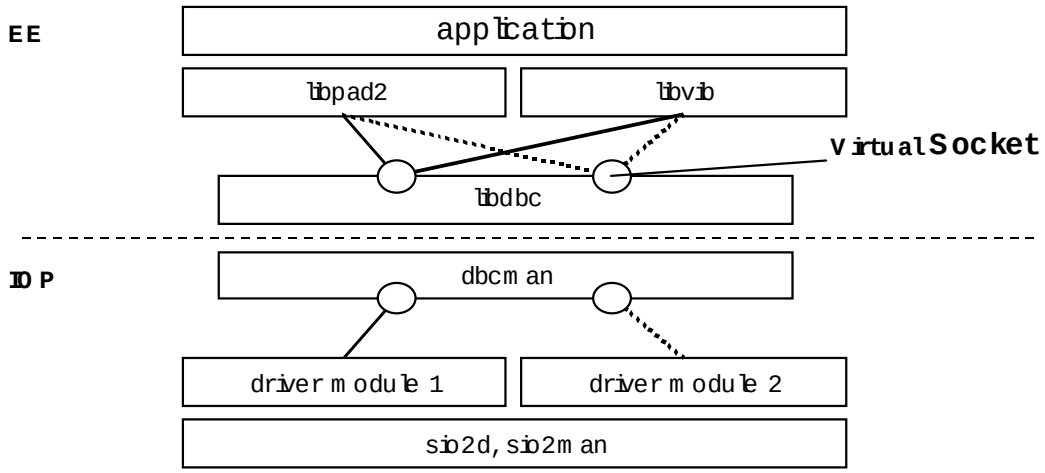
Category	Filename
Library files	libdbc.a
	libpad2.a
	libvib.a
Header files	libdbc.h
	libpad2.h
	libvib.h
IOP module files	sio2man.irx (Release 2.4 and later)
	sio2d.irx
	dbcman.irx
	Various driver modules (***.irx)

IOP modules must be loaded using sceSifLoadModule() when the EE-side program is started up. For driver modules, select and load the modules that are required by the application. Currently, libvib supports the DUALSHOCK and DUALSHOCK 2 driver modules. For more information about driver modules, refer to the libpad2 Overview.

Library Configuration

The following figure shows the configuration of the library

Figure 10-1



libvib has been positioned at the same level as libpad2. Vibration commands are sent to the specified device using the socket number. The socket number is generated by libpad2, and the application uses it in specifications to libvib.

Sample Programs

The following libvib sample program is provided for reference.

- **sce/ee/sample/pad2/vib**
  This program shows the basic use of the vibration library.

Usage

Basic Procedure

The basic procedure for using libvib to vibrate the controller is as follows.

1. **Load IOP modules**
   Use sceSifLoadModule() to sequentially load sio2man.irx, sio2d.irx, and dbcman.irx in order. Load the appropriate driver modules as required by the application.
2. **Initialize libraries**
   Call sceDbcInit() and scePad2Init() to initialize libdbc and libpad2, respectively.
3. **Create virtual sockets**
   Call scePad2CreateSocket() to create the virtual sockets to be used. To specify controller ports to be linked to the sockets, set values for optional conditions in the parameter structure.
   A work buffer, which is an area that is used for DMA transfers from the IOP, must be allocated with 64-byte alignment with respect to the cache.
4. **Set parameters for actuators**
   When the drivers are subsequently loaded and the controllers are in a stable state, use sceVibSetActParam() to send the vibration profile and vibration data. The bits for actuators for which the state is to be changed should be set to 1 in the vibration profile, and the parameters for those actuators are stored so they are pre-packed in the vibration data. Later, the device driver receives this data making the controller vibrate.
   To check whether or not the controller is in a stable state, use scePad2GetState().
5. **Termination processing**
   When communication with the controllers is no longer necessary, call scePad2DeleteSocket() to delete each virtual socket. Then, call scePad2End() to perform termination processing.

Sample Source Code

```
main()
{
    int socket_number;
    unsigned char out_data;
    unsigned char out_profile;
    u_long128 pad_buff[SCE_PAD2_DMA_BUFFER_MAX]
__attribute__((aligned(64)));
```

```
sceDbcInit();
scePad2Init();
    socket_number = scePad2CreateSocket( NULL, pad_buff );
    while(1)
     {
       if( scePad2GetButtonInfo( socket_number, rdata,
SCE_PAD2_UP ) )
     {
            out_profile = 0x01;
            out_data = 0x01;
            sceVibSetActParam( socket_number, 1, &out_profile,
                                              1, &out_data );
       }
     .
     .
     .
     .
    }
    scePad2DeleteSocket( socket_number );
    scePad2End();
    sceDbcEnd();
}
```

Profile

Concerning Profiles

A profile lists the features of a controller. That is, it indicates all the detailed information about the controller such as the number and types of actuators of a connected controller.

Profile Data Format

Each bit of profile data describes an individual feature. If the controller has that feature, the corresponding bit is set to 1. If it does not have that feature, the corresponding bit is set to 0. The resulting bit pattern is the controller's profile data.

Vibration data is arranged so that the actuator data corresponding to the bits that are 1 in the vibration profile will be pre-packed in order of increasing vibration profile bit numbers.

Vibration Profile

The vibration profile consists of the following one byte of data (we plan to extend this in the future) with actuator types assigned to each bit.

Table 10-2

Byte	Bit	Feature	Data Size (bits)
0	0	Small motor	1
	1	Large motor	8
	2	(Subsequent	
	3	bits are undefined)	
	4		
	5		
	6		
	7		

Vibration Data

(Sample send data) Setting a value for the large motor

Table 10-3: Profile Data

Byte	0
Bit	7 to 0
Bit Pattern	00000010

Table 10-4: Vibration Data

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Large motor set value							

(Sample send data 2) Simultaneously setting values for the large and small motors

Table 10-5: Profile Data

Byte	0
Bit	7 to 0
Bit Pattern	00000011

Table 10-6: Vibration Data

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Large motor set value Lower 7 bits							Small motor set value
1								Large motor set value Lower 1 bit

Chapter 11: PlayStation File System Overview

Library Overview

The pfs module is used to manage the PFS (PlayStation File System).

The PFS is a filesystem for the PlayStation 2, designed to maximize performance while keeping memory utilization to a minimum. It has external specifications like hierarchical directories and file modes similar to that of the Unix filesystem. It also has a feature that enables multiple partitions to be treated as a single filesystem. It has a ROBUST mode that keeps the danger of data corruption to a minimum. Moreover, even when data corruption occurs, the data can be quickly and precisely recovered through the use of journalling.

All operations for the pfs module are performed via the I/O manager. Device names are "pfs0:" / "pfs1:" / "pfs2:" ... and the numbers can be specified when the filesystems are mounted.

Related Files

The following files are required to use the pfs module.

EE

Table 11-1

Category	Filename
Header file	sifdev.h
Library file	libkernl.a

IOP

Table 11-2

Category	Filename
Header files	stdio.h
	dirent.h
	errno.h
	sys/file.h
	sys/ioctl.h
	sys/mount.h
	sys/stat.h
Library file	iop.ilb
Module files	dev9.irx
	atad.irx
	hdd.irx
	pfs.irx

Modules

Dependency Relationships Between Modules

To use the pfs module, the following three modules must be loaded sequentially prior to pfs.irx and should be made resident in memory.

dev9.irx	Device module for which card is connected
atad.irx	Hard disk drive driver module
hdd.irx	Hard disk drive and partition management module

If an error occurs during initialization of these modules, then they will not be resident in memory. When loading the pfs module, ensure that these modules are resident by using LoadStartModule() on the IOP and by using sceSifLoadStartModule() on the EE.

Module Arguments

-m Number of Simultaneous Mounts

Specifies the number of filesystems that can be mounted simultaneously. In the current version, one mount consumes 324 bytes of memory.

-n Number of Buffer Caches

Specifies the number of buffer caches to be used by the driver. In the current version, each buffer consumes 1052 bytes of memory. By default, 8 buffers are used. Increasing the number of buffers may result in an increase in the speed of operation of the module.

The minimum number of buffers that can be specified is 9 and the maximum number for the current version is 127. Specifying a value of less than 8 results in 8 getting specified.

-o Number of Files

Specifies the number of files that can be opened simultaneously. In the current version, each open consumes 560 bytes of memory. Each open also requires 1 ~ 2 buffer caches.

PFS Filesystem Features

Structure

64-bit Filesystem

PFS is a 64-bit filesystem. The unit of a single read / write / seek operation is, however, 32 bits.

Upper Limit on Size

The maximum size of one filesystem is 2T bytes. The maximum file size is also 2T bytes.

Logical Volumes

For effective utilization of the disk, PFS has a structure that enables multiple physical partitions to be treated as a single logical volume (filesystem). Even after building a filesystem, its size can be changed by adding partitions. The newly added partitions are recognized automatically when the filesystem is mounted.

Directories

PFS contains a hierarchical directory-based structure, similar to a conventional filesystem. The size of a directory is of variable length and the number of files that can be placed in a directory is unlimited. However, since there is no directory lookup cache, performance will go down if a large number of files are placed in a single directory. As much as possible, do not place a large number of files in a single directory.

Filenames

Any characters can be used in a filename except '/'. The maximum length of a filename is 255 characters and the maximum length of a path name is 1024 characters.

File Modes

PFS allows execution rights / writing rights / reading rights to be established for each of the 3 classes of owner / group / other, as file access modes (permissions), as in a conventional Unix system.

In the current PlayStation 2 system, there is no such concept as "user." Therefore, in the current library, the user ID (uid), which represents the owner of the file, and the group ID (gid), which represents the group to which the user belongs, are handled as constants (0xffff). For files created by the system, uid / gid are both 0.

The concept of user may be introduced in the future. Mode must be specified at the time of creating a file or directory.

Links

PFS supports only symbolic links. It does not support hard links.

Extended Attributes

Extended attributes using "key/value" can be set up in each file similar to the way MIME types are handled. The length of strings used for key and value is limited to 255 characters. There is no restriction on the characters that can be used.

In the current version, however, the extended attributes area is limited to 1024 bytes. Because of this, in the worst case, it may be that at most 2 entries can be set up.

Performance

Memory Efficiency

PFS is designed to reduce memory usage as much as possible and to get maximum performance out of a smaller memory utilization. Moreover, the amount of buffer cache used by the filesystem can be changed by means of an option when the module is loaded.

File Location

In PFS, the speed of file reads is considered the most important consideration from among the various aspects of performance. When files are placed on disk, a continuous area is allocated as much as possible, so as to enable high-speed reading. Due to this, file write speed is relatively slower but only when an area is being reserved.

If it is desired to improve file write speed, the area should be reserved in advance.

Setting Zone Size

In PFS, the minimum unit for building file fragments (this corresponds to a cluster in FAT or a block in ext2, etc.) is called a zone. The size of a zone is variable and can be selected from 2K / 4K / 8K ... 128K bytes when the filesystem is formatted.

Safety

Journalling

PFS has a recording process called journalling that enables recovery of the filesystem in case the filesystem has become corrupted due to a system crash. When an operation such as file creation is performed, it is necessary to write filesystem configuration information such as data related to directories or inodes (this is known as metadata) to the hard disk drive, besides the actual file contents. If the system crashes during a series of writes, the metadata will become inconsistent and it may not be possible to read the file contents properly.

Even in a conventional filesystem, the filesystem can be recovered and inconsistencies detected by checking all the metadata using the filesystem check utility. This, however, takes a very long time and complete recovery may not be possible.

In PFS, a journal area is provided separately and the metadata is written after changes are recorded in this area. Even if the system crashes in the middle of an operation, the filesystem can be recovered quickly and completely on the basis of journal data.

Moreover, PFS performs journalling only for metadata. The file data is written directly to the disk and is not kept, even in the filesystem's buffer cache.

ROBUST Mode

A mode called ROBUST mode that keeps the danger of data corruption to a minimum, can be selected when the filesystem is mounted.

When mounted in normal mode, metadata is updated when the buffer cache is full of data to be updated, when the Sync function is called, or when the filesystem is unmounted. For example, when a new file is created, the metadata may not be updated at the point after the open function returns. If the system were to crash at this time, the file that should have been created may be lost.

When mounting is performed in ROBUST mode, whenever a process for updating the filesystem is performed, the metadata is updated. For example, it is guaranteed that on returning from the open function, the metadata wiil be updated as if the Sync function had been called.

Note that when ROBUST mode is specified, performance is reduced.

Functions Other Than File Management

In addition to normal file management, the following functions are provided.

- Allocating / freeing areas
- Obtaining region size
- Obtaining the number of empty zones
- Batch closing of all files
- Extended attribute operations

For details, refer to the function reference.

Notes

Conditions for Maintaining Performance

Reading and writing the hard disk drive in units of 512 bytes is an effective way to improve speed. Performance of hard disk drive reading and writing is lowered in the following cases.

- If the file pointer is not a multiple of 512
- If read and write sizes are not multiples of 512
- If read/write buffers are not 64-byte aligned on the EE

When it is difficult to use 512-byte units, try to read and write in units of 64 bytes on the EE, but at least in 4-byte units. Performance improves when reading and writing is performed in somewhat larger units (16K - 128K).

Read/Write Speed Not Guaranteed

Programs that perform hard disk drive reads and writes should be created so that they do not depend on disk access speed.

Speed will vary depending on the type of hard disk drive, and furthermore, read/write speed varies greatly depending on the location of files. There is no guarantee that read/write speed at the time of manufacture will be reproduced in the end user's environment, so be sure that the program is independent of the read/write speed of the disk.

Power OFF Process

When powering OFF the PlayStation 2 console, be sure to perform hard disk drive power off processing using CD(DVD)-ROM library functions because these functions support the EXPANSION BAY type of PlayStation 2 hard disk drive.

1. Detect interrupt processing with sceCdPOffCallback().
2. Close files etc.
3. Power OFF dev9.
4. Power OFF the PlayStation 2 console with sceCdPoweroff().

For an external type of PlayStation 2 hard disk drive, the sceCdPOffCallback() function cannot be used to detect the interrupt.

In 3., dev9 can be powered off by using the DDIOC_OFF devctl command of the dev9 library directly and by using the HDIOC_DEV9OFF command of the hard disk library. However, when the hard disk drive is used, the hard disk library command must be used.

Also, when cdvdfsv.irx is unloaded, libcdvd functions will no longer be available from the ee.

After cdvdfsv.irx is unloaded, be sure to use scePowerOffHandler() in 1. and to use CDIOC_POWEROFF of sceDevctl() in 4.

For details, refer to the hard disk library samples (ee/sample/hdd/basic), CD(DVD)-ROM library reference, network library overview (inet), and standard I/O function reference.

Error Processing

The most significant 16th bit of errno represents a device-dependent error. All the errors returned by pfs are defined in errno.h but currently, a device-dependent error is returned in only one case.

This error occurs when a bad sector is detected in user file data during a file read. It is not an ordinary EIO error and returns -(EIO|0x10000) instead.

To deal with this error, do not run the filesystem check utility. Either overwrite the file or erase it completely.