PlayStation®2 IOP Library Reference
Release 2.4.3

Sif Libraries

Summary Table of Contents

About This Manual

This is the Runtime Library Release 2.4.3 version of the *PlayStation®2 IOP Library Reference - Sif Libraries* manual.

The purpose of this manual is to define all available PlayStation®2 IOP sif library structures and functions. The companion *PlayStation®2 IOP Library Overview - Sif Libraries* describes the structure and purpose of the libraries.

Changes Since Last Release

Chapter 1: Multithreaded SIF Remote Procedure Calls (for IOP)

- In the "Structure" section of the sceSifMServeData structure, an error in the *client* member type has been corrected.

Related Documentation

Library specifications for the EE can be found in the *PlayStation®2 EE Library Reference* manuals and the *PlayStation®2 EE 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: Multithreaded SIF Remote Procedure Calls (for IOP)
Table of Contents

Structures

sceSifMQueueData

RPC service function

Library	*Introduced*	*Documentation last modified*
imrpc	2.1	January 22, 2001

Structure

```
typedef struct _sifm_queue_data {
    int key;
    int active;
    int sleep;
    struct _sifm_serve_data *link;
    struct _sifm_serve_data *start;
    struct _sifm_serve_data *end;
    struct _sifm_queue_data *next;
} sceSifMQueueData;
```

The members are automatically set, so there is no need to set them in the program.

Description

This data structure is used to queue requests received by the server.

See also

sceSifMEntryLoop()

sceSifMRpcFunc

RPC service function

Library	*Introduced*	*Documentation last modified*
imrpc	2.1	January 22, 2001

Structure

typedef void * (* sceSifMRpcFunc)(

unsigned int *fno,*	fno of sceSifMCallRpc()
void **data,*	Address where received data is stored
int *size);*	Size of received data

Description

This is the RPC service function executed on the server side. It is registered by means of sceSifMEntryLoop(), and it is executed with sceSifMExecRequest() after a request is received.

The address of the data returned to the client that generated the request, is the return value of this function. The reply address and data size are specified by the client.

See also

sceSifMEntryLoop()

sceSifMServeData

RPC server data

Library	*Introduced*	*Documentation last modified*
imrpc	2.1	January 4, 2002

Structure

```
typedef struct _sifm_serve_data {
    void *func_buff;
    int size;
    void  *cfunc_buff;
    int csize;
    sceSifMClientData *client;
    void  *paddr;
    unsigned int fno;
    void  *receive;
    int rsize;
    int rmode;
    unsigned int rid;
    struct _sifm_serve_data  *link;
    struct _sifm_serve_data  *next;
    struct _sifm_queue_data  *base;
    struct _sifm_serve_entry  *sentry;
} sceSifMServeData;
```

The members are automatically set, so there is no need to set them in the program.

Description

This data structure is used to register the received data address, the client data address, etc. Every time sceSifMBindRpc() is called from a client thread, sceSifMServeData is created for each thread.

sceSifMServeEntry

RPC server entry data

Library	*Introduced*	*Documentation last modified*
imrpc	2.1	January 22, 2001

Structure

```
typedef struct _sifm_serve_entry {
    unsigned int mbxid;
    unsigned int command;
    sceSifMRpcFunc  func;
    sceSifMRpcFunc  cfunc;
    sceSifMServeData *serve_list;
    struct _sifm_serve_entry *next;
} sceSifMServeEntry;
```

The members are automatically set, so there is no need to set them in the program.

Description

This data structure is used to register service functions, IDs for identifying requests received by the server, etc.

See also

sceSifMEntryLoop()

Functions

sceSifMEntryLoop

Wait for request

Library	*Introduced*	*Documentation last modified*
imrpc	2.1	March 26, 2001

Syntax

void sceSifMEntryLoop(	
sceSifMServeEntry **se,*	Pointer to RPC server entry data
int *request,*	Request ID
sceSifMRpcFunc *func,*	Function executed when request received
sceSifMRpcFunc *cfunc);*	(Unused)

Calling conditions

Can be called by a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

For a particular service function, this function enters a loop that repeats the wait for Bind and UnBind requests from a client, executes services, and creates service threads. sceSifMEntryLoop() is necessary to implement the functionality of a server. When the function is called, the executing thread sleeps until a request from a client is received. After the request is received, the thread wakes up. If the request is a Bind, this function creates the associated service thread. If the request is an UnBind, it deletes the associated service thread.

After this processing ends, the cycle of again waiting for a request repeats.

Return value

None (no return from this function)

sceSifMInitRpc

Initialize MSIF RPC API

Library	*Introduced*	*Documentation last modified*
imrpc	2.1	March 26, 2001

Syntax

void sceSifMInitRpc(	
unsigned int *mode);*	Start-up mode (fixed at 0 in the current implementation)

Calling conditions

Can be called by a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function initializes the MSIF RPC API.

It initializes the internal variables and registers command functions for handling requests in the system buffer of the MSIF command API.

Initialization must be performed by both the server and the client.

In order to synchronize server and client, internally one side waits in the function until the other side is called.

Return value

None

Chapter 2: SIF Command
Table of Contents

Structures

sceSifCmdData

Command function registration data

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	January 27, 2000

Structure

typedef struct {	
sceSifCmdHandler *func;*	Command function.
void *data;*	Address of data passed as argument when function is executed
} sceSifCmdData;	

Description

This is the data structure used when registering a command function.

sceSifCmdHandler

Command function

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	January 27, 2000

Structure

typedef void (*sceSifCmdHandler)(

void **pkt,*	Address of copy of command packet specified by sceSifSendCmd()
void **data)*	Address of data registered together with command function by sceSifAddCmdHandler

Description

This function is executed during a DMA interrupt that occurs due to the sceSifSendCmd() function. At this time, the addresses of the sender's command packet and receiver's specified data are passed as arguments.

sceSifCmdHdr

Command packet header

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	January 27, 2000

Structure

typedef struct {	
unsigned int *psize:8;*	Size of command packet including this header (16 <= *psize* <= 112)
unsigned int *dsize:24;*	Size of accompanying data that is sent together with packet
unsigned int *daddr;*	Address of accompanying data
unsigned int *fcode;*	Number of command function that is called
unsigned int *opt;*	Data area that programmer can use
} sceSifCmdHdr;	

Description

A command packet is at most 112 bytes of data beginning with this data structure.

sceSifCmdSRData

Software register update packet

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	February 29, 2000

Structure

typedef struct {	
sceSifCmdHdr *chdr;*	Command packet header
int *rno;*	Software register number (0 to 31)
unsigned int *value;*	Setting value
} sceSifCmdSRData;	

Description

The SIF Command API system has 32 arrays (software registers), each of which has a size of 32 bits. System registers having numbers 0 to 7 are used by the system, and the remaining software registers can be used by user programs. The functions for using these software registers are also registered by default.

To set a software register on the target side, use this structure as follows. (This example sets register number 31 on the target side to the value 0xff.)

```
sceSifCmdSRData d;
d.rno = 31;
d.value = 0xff;
sceSifSendData(SIF_CMDC_SET_SREG,&d,sizeof(d),0,0,0);
```

To get or set a software register value on the local side, use the sceSifGetSreg() or sceSifSetSreg() functions.

See also

sceSifGetSreg(), sceSifSetSreg()

Functions

sceSifAddCmdHandler

Register command function

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	October 11, 2001

Syntax

#include <sifcmd.h>

void sceSifAddCmdHandler(

unsigned int *pos,*	Position in buffer for registering command function
sceSifCmdHandler *f,*	Command function to be registered
void **data***)**	Address of data that is passed to command function f

Calling conditions

Can be called from an interrupt handler

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-disabled state)

Description

Registers a function (command function) that will be called when a command packet in the buffer that was registered using the sceSifSetCmdBuffer() function is invoked.

Notes

Since command functions are executed as interrupt handlers, special care is required when programming. Refer to the “Interrupt Handler Descriptions” section of \overview\eekernel for details.

Return value

None

sceSifExitCmd

Terminate SIF Command API

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	March 26, 2001

Syntax

#include <sifcmd.h>

void sceSifExitCmd(void)

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

Removes the interrupt function that was registered by the sceSifInitCmd() function.

When the SIF Command API is used before an object transition occurs on the EE, this function must be used to remove the interrupt function, and sceSifInitCmd() must be called by the new object.

Return value

None

sceSifGetSreg

Get software register contents

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	March 26, 2001

Syntax

#include <sifcmd.h>

unsigned int sceSifGetSreg(

int *no***)**	Register number (0 to 31)

Calling conditions

Can be called from a thread

Not multi-thread safe

Description

Gets the value of a local-side software register. The initial value of software registers is zero.

Return value

Register contents

sceSifInitCmd

Initialize SIF Command API

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	March 26, 2001

Syntax

#include <sifcmd.h>

void sceSifInitCmd(void)

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

Initializes the SIF Command API. Registers an interrupt function for initializing internal variables and processing commands.

The side that calls sceSifInitCmd() first (either EE or IOP) will be synchronized with the other side. Consequently, when this function is called, one side will enter a wait state until the function is called by the other side.

Return value

None

sceSifRemoveCmdHandler

Remove command function

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	March 26, 2001

Syntax

#include <sifcmd.h>

void sceSifRemoveCmdHandler(

unsigned int *pos***)**	Position in buffer of function to be removed

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

Deletes the function that was registered at buffer position *pos*.

Return value

None

sceSifSendCmd

Send command packet

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	March 26, 2001

Syntax

#include <sifcmd.h>

unsigned int sceSifSendCmd(

unsigned int *pos,*	Position of function to be called (position registered by the sceSifAddCmdHandler() function)
void **cp,*	Command (command packet) address
int *ps,*	Command packet size in bytes (16 <= *ps* <= 112 bytes)
void **src,*	Address of additional data to be sent
void **dest,*	Address of target's additional data
int *size***)**	Size of additional data in bytes

Calling conditions

Can be called from an interrupt handler

Can be called from a thread

Multi-thread safe (must be called in an interrupt-enabled state)

Description

Uses sceSifSetDma() to send a command packet. Then calls the command function that is registered at position pos on the target side.

Successful delivery of the command to the target can be determined from the return value, and by calling the sceSifDmaStat() function.

cp, *src*, and *dest* should be placed at addresses that are aligned on 16-byte boundaries for the EE and on 4-byte boundaries for the IOP. *size* is a multiple of 16 bytes for the EE and a multiple of 4 bytes for the IOP.

For the EE, although *cp* is written back if it is in the cache, since *src* is not, it is the programmer's responsibility to make sure that it is flushed from the cache.

The isceSendCmd() function should be called within an interrupt function (for both the EE and IOP).

Notes

The *size* maximum is the DMA maximum 1Mbyte - 16bytes which can be sent at one time.

Return value

Queuing identifier used by sceSifSetDma() function

0: Queuing failed

Non-zero: Queuing identifier

sceSifSendCmdIntr

Send command packet

Library	*Introduced*	*Documentation last modified*
sifcmd	2.2	March 26, 2001

Syntax

#include <sifcmd.h>	
unsigned int sceSifSendCmdIntr (	
unsigned int *pos,*	Position of the function to call (the position registered with sceSifAddCmdHandler())
void **cp,*	Address of the command (command packet)
int *ps,*	Size of the command packet (bytes) (between 16 and 112 bytes)
void **src,*	Address of data sent together (appended data)
void **dest,*	Address of the destination sending the appended data
int *size,*	Size of the appended data (bytes)
void (**func***)(void *),**	Function that is called after sending the command
void **data***)**	Argument passed to *func*

Calling conditions

Can be called from an interrupt handler

Can be called from a thread

Multi-thread safe (must be called in an interrupt-enabled state)

Description

After sending the command, func() is called as the interrupt function. At that time, data is passed as the argument. Other than this, it is similar to sceSifSendCmd().

For interrupt-disabled areas, refer to isceSifSendCmdIntr().

At present, this function has been implemented only for the IOP.

Return value

Queuing identifier for the used sceSifSetDma()

0	Queuing failure
Other than 0	Queuing identifier

sceSifSetCmdBuffer

Register command function buffer

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	October 11, 2001

Syntax

#include <sifcmd.h>

sceSifCmdData * sceSifSetCmdBuffer(

sceSifCmdData **db,*	Starting address of buffer
int *size***)**	Size of buffer

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-disabled state)

Description

Sets a buffer for registering a function (command function) that will be invoked by the SIF Command API. Initially, no buffer is registered.

Return value

Address of buffer that had been previously registered.

sceSifSetSreg

Set software register contents

Library	*Introduced*	*Documentation last modified*
sifcmd	1.1	March 26, 2001

Syntax

#include <sifcmd.h>

unsigned int sceSifSetSreg(

int *no,*	Register number (0 to 31)
unsigned int *value***)**	Value to be set in register

Calling conditions

Can be called from a thread

Not multi-thread safe

Description

Sets the specified value in the local-side software register.

Return value

Value that was set.

Chapter 3: SIF DMA
Table of Contents

Structures

sceSifDmaData

DMA data

Library	*Introduced*	*Documentation last modified*
sdma	1.1	October 6, 2000

Structure

typedef struct {	
u_int *data;*	Address of data to be sent (must be aligned on a 16-byte boundary for the EE and on a 4-byte boundary for the IOP)
u_int *addr;*	Target address to which data is to be sent (must be aligned on a 16-byte boundary for the EE and on a 4-byte boundary for the IOP)
u_int *size;*	Data size (16-byte units for the EE and 4-byte units for the IOP)
u_int *mode;*	Note: Currently, this should only be set to 0 (mode where no interrupt is issued). Do not use SIF DMA related interrupts since they are currently used only with SIF CMD. SIF_DMA_INT_I: Interrupt after transfer ends sender) SIF_DMA_INT_O: Interrupt after transfer ends receiver) SIF_DMA_TAG : 1 qword at beginning of data may be used as TAG (can be specified only by the EE) SIF_DMA_ERT: Stop IOP DMA after transfer (can be specified only by the EE)
} sceSifDmaData;	

Description

This structure is used to specify the address of the data to be DMA transferred, the destination address, the size, and the mode.

The addresses must be aligned on a 16-byte boundary for the EE and on a 4-byte boundary for the IOP.

The size is in units of 16-bytes for the EE and in units of 4-bytes for the IOP. The maximum number of units transferred at one time (with one sceSifDmaData) is (1M - 16) bytes.

See also

sceSifSetDma()

Functions

sceSifDmaStat / isceSifDmaStat

Get queuing state

Library	*Introduced*	*Documentation last modified*
sdma	1.1	March 26, 2001

Syntax

#include <sif.h>	
int sceSifDmaStat(	
unsigned int *id***)**	Queuing identifier returned by sceSifSetDma()
int isceSifDmaStat(	
unsigned int *id***)**	Queuing identifier returned by sceSifSetDma()

Calling conditions

sceSifDmaStat	EE	Can be called from a thread
		Multi-thread safe (must be called in an interrupt-enabled state)
	IOP	Can be called from a thread
		Not multi-thread safe (must be called in an interrupt-disabled state)
isceSifDmaStat		Can be called from an interrupt handler

Description

Checks the DMA state of the specified *id*.

Notes

For the EE, isceSifSetDma() should be used within an interrupt function.

Return value

Positive (>0):	Queued and standing by
0:	DMA execution in progress
Negative (<0):	DMA completed

sceSifSetDChain / isceSifSetDChain

Set DMA channel again

Library	*Introduced*	*Documentation last modified*
sdma	1.1	March 26, 2001

Syntax

#include <sif.h>

void sceSifSetDChain(void)

void isceSifSetDChain(void)

Calling conditions

sceSifSetDChain	EE	Can be called from a thread
		Multi-thread safe (must be called in an interrupt-enabled state)
	IOP	Can be called from a thread
		Not multi-thread safe (must be called in an interrupt-disabled state)
isceSifSetDChain		Can be called from an interrupt handler

Description

When a DMA transfer cannot be received due to an interrupt or another cause, the DMA channel can be set again on the receiving side by executing this function.

For the EE, if a SIF DMA interrupt from the IOP occurs, this function must be used to set the channel again since the DMA receiving channel will be closed.

For the IOP, if SIF_DMA_ERT has been specified, this function must be used to set the channel again since the receiving channel will be similarly closed.

Notes

For the EE, isceSifSetDChain() should be called within an interrupt function.

sceSifSetDma / isceSifSetDma

Perform DMA transfer to target's memory

Library	*Introduced*	*Documentation last modified*
sdma	1.1	March 26, 2001

Syntax

#include <sif.h>	
unsigned int sceSifSetDma(	
sceSifDmaData **sdd,*	DMA contents
int *len***)**	*sdd* data count
unsigned int isceSifSetDma(	
sceSifDmaData **sdd,*	DMA contents
int *len***)**	*sdd* data count

Calling conditions

sceSifSetDma	EE	Can be called from a thread
		Multi-thread safe (must be called in an interrupt-enabled state)
	IOP	Can be called from a thread
		Not multi-thread safe (must be called in an interrupt-disabled state)
isceSifSetDma		Can be called from an interrupt handler

Description

Uses DMA to send data to the receiver's address.

Multiple data can be specified at one time by using the sceSifDmaData structure.

If a DMA transfer is already in progress, the request is queued so that the next transfer will begin after the current transfer completes.

Over time, the same value may be returned for the queuing identifier.

With the current implementation, the queue count is 32 ring buffers for the EE and 32 double buffers per side for the IOP. As a result, the DMA completion interrupt function resides in the IOP.

Notes

For the EE, isceSifSetDma() should be used within an interrupt function.

For the IOP, this function must be called when interrupts are disabled.

SPR cannot be handled in the current implementation.

Return value

Non-zero: The queuing identifier is returned

0: Queuing failed

sceSifSetDmaIntr

Library	*Introduced*	*Documentation last modified*
sdma	2.2	March 26, 2001

Syntax

#include <sif.h>

unsigned int sceSifSetDmaIntr(

sceSifDmaData **sdd,*	Contents of DMA
int *len,*	Number of *sdd* data
void (**func***)(void *),**	Function that is called after completion of DMA
void ****data*)**	Argument passed to *func*

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in interrupt-disabled state)

Description

After the completion of data transfer, func() is called as the interrupt function. At that time, *data* is passed as the argument. Other than this, it is similar to sceSifSetDma().

At present, this function has been implemented only for the IOP.

Notes

For the IOP, this function must be called from an interrupt-disabled area.

The SPR cannot be handled in the current implementation.

Return value

Other than 0	Queuing identifier
0	Queuing failure

Chapter 4: SIF Remote Procedure Call
Table of Contents

Structures

sceSifClientData

RPC client information

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	July 24, 2000

Structure

typedef struct _sif_client_data {
  struct _sif_rpc_data *rpcd;*
  unsigned int *command;*
  void **buff*;
  void **cbuff;*
  sceSifEndFunc *func;*
  void **para;*
  struct _sif_serve_data **serve;*
} sceSifClientData;

Description

This structure stores client information that was obtained by sceSifBindRpc(). Because the members are automatically set, there is no need to set them in the program. This structure is also used when a service function is called by sceSifCallRpc().

See also

sceSifBindRpc(), sceSifCallRpc(), sceSifCheckStatRpc()

sceSifEndFunc

RPC end function

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	January 27, 2000

Structure

typedef void (* sceSifEndFunc)(	
void ****data*);**	Data address passed when function is called

Description

This function is called in an interrupt area when the RPC service function ends. At this time the address of data is passed.

See also

sceSifCallRpc()

sceSifQueueData

RPC request queue data

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	July 24, 2000

Structure

```
typedef struct _sif_queue_data {
    int key;
    int active;
    struct _sif_serve_data *link;
    struct _sif_serve_data *start;
    struct _sif_serve_data *end;
    struct _sif_queue_data *next;
} sceSifQueueData;
```

Description

This structure queues requests that were received by the server. Because the members are automatically set, there is no need to set them in the program.

See also

sceSifSetRpcQueue(), sceSifRegisterRpc(), sceSifGetNextRequest(), sceSifRpcLoop()

sceSifReceiveData

RPC data receive information

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	July 24, 2000

Structure

typedef struct _sif_receive_data {

  struct _sif_rpc_datarpcd;

  void **src;*

  void **dest;*

  int *size;*

} sceSifReceiveData;

Description

This structure stores control data when data from the target is received using DMA. Because the members are automatically set, there is no need to set them in the program.

See also

sceSifCheckStatRpc(), sceSifGetOtherData()

sceSifRpcData

RPC client data header

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	July 24, 2000

Structure

typedef struct _sif_rpc_data {

void **paddr;*	Packet address
unsigned int *pid;*	Packet ID
int *tid;*	Thread ID
unsigned int *mode;*	Call mode

} sceSifRpcData;

Description

Common header data for RPC clients.

See also

sceSifClientData(), sceSifReceiveData(), sceSifCheckStatRpc()

sceSifRpcFunc

RPC service function

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	January 27, 2000

Structure

typedef void * (* sceSifRpcFunc)(	
unsigned int *fno,*	*fno* of sceSifCallRpc()
void **data,*	Address where receive data is stored
int *size***)**	Size of receive data

Description

This is an RPC service function that is executed by the server. It is registered using sceSifRegisterRpc() and is executed using sceSifExecRequest() when a request is received.

The return value of this function is the address of the data that is returned to the client that issued the request. The destination address and data size are specified by the client.

See also

sceSifRegisterRpc()

sceSifServeData

RPC server data

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	July 24, 2000

Structure

```
typedef struct _sif_serve_data {
    unsigned int command;
    sceSifRpcFunc func;
    void *buff;
    int size;
    sceSifRpcFunc cfunc;
    void *cbuff;
    int csize;
    sceSifClientData *client;
    void *paddr;
    unsigned int fno;
    void *receive;
    int rsize;
    int rmode;
    unsigned int rid;
    struct _sif_serve_data *link;
    struct _sif_serve_data *next;
    struct _sif_queue_data *base;
} sceSifServeData;
```

Description

This structure registers various pieces of information used to identify a request that was accepted by the server. This information includes the identifier, service function, and receive data address.

Because the members are automatically set, there is no need to set them in the program.

See also

sceSifExecRequest(), sceSifGetNextRequest(), sceSifRegisterRpc()

Functions

sceSifBindRpc

Search RPC service function data

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

int sceSifBindRpc(

sceSifClientData **bd,*	Pointer to structure for fetching client information
unsigned int *request,*	Request identifier
unsigned int *mode*)	Calling mode. This is usually zero. Specify the following constant when necessary: SIF_RPCM_NOWAIT: Asynchronous execution

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Gets required client information from the server to send a request. This function is needed by the client.

When this function is called, it checks whether the service function for the request identifier specified by the *request* argument has been registered on the server. If it has been registered, it returns client information in the sceSifClientData structure specified by the *bd* argument, that will subsequently be used as the calling key.

Normally, the thread that called this function is in Sleep state until there is a reply from the server. If SIF_RPCM_NOWAIT has been specified for the mode argument, control exits directly without the thread entering Sleep state. In this case, the completion of processing on the server can be confirmed by using the sceSifCheckStatRpc() function.

Notes

From release 1.4, the EE uses the internally reserved semaphore to wait for completion, instead of Sleep.

Whether or not the service function had been registered (whether Bind succeeded) can be determined by whether a non-zero value is set for the serve member of sceSifClientData. This is shown below.

```
#define BIND_ID  0x12345678
while(1){
       if (sceSifBindRpc( &cd0, BIND_ID, 0) < 0) {
              printf("bind errr\n");
              exit(-1);
       }
       if (cd0.serve != 0) break;
}
```

If requests are frequently sent from the EE to the IOP using code such as that shown above, the IOP will be practically stopped because the EE is quite fast. Therefore, a small interval should be inserted between requests.

Return value

0:	Notification to server succeeded
Negative (<0):	Execution failed

sceSifCallRpc

Call RPC service function

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

int sceSifCallRpc(

sceSifClientData **bd,*	Client information for which Bind has completed
unsigned int *fno,*	Number to be passed to the service function that is called
unsigned int *mode,*	Calling mode. This is usually zero. Specify the following constants as a mask when necessary: SIF_RPCM_NOWAIT: Asynchronous execution SIF_RPCM_NOWBDC: No cache writeback
void **send,*	Data buffer to be sent (16-byte/4-byte alignment for EE/IOP)
int *ssize,*	Data size to be sent (bytes; 16-byte/4-byte units for EE/IOP)
void **receive,*	Data buffer to be received (16-byte/4-byte alignment for EE/IOP)
int *rsize,*	Data size to be received (bytes; 16-byte/4-byte units for EE/IOP)
sceSifEndFunc **end_func,*	Function that is executed when execution ends and interrupts are disabled
void **end_para*)	Address of *end_func* parameter

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Calls a service function that has been registered. This function is used by the client.

For the *bd* argument, specify the client information structure that had previously used the sceSifBindRpc() function to complete the Bind.

ssize bytes of the data specified by the send argument are sent to the server, and the *send* and *ssize* argument values are passed as the second and third arguments to the service function. The *fno* argument value is passed as the first argument.

After the service function is executed, *rsize* bytes of the data at the address indicated by the service function's return value are sent back to the area specified by the *receive* argument.

After execution ends, the function specified by the *end_func* argument is called when interrupts are disabled.

Normally, the thread that called the sceSifCallRpc() function is in Sleep state until there is a reply from the server. If SIF_RPCM_NOWAIT has been masked in advance for the mode argument, control exits directly without the thread entering Sleep state. In this case, the completion of processing on the server can be confirmed by using the sceSifCheckStatRpc() function.

Notes

From release 1.4, the EE uses the internally reserved semaphore to wait for completion, instead of Sleep.

For the EE, although writeback is performed for send/receive data that has been loaded in the cache, if SIF_RPCM_DOWBDC has been masked in advance for the *mode* argument, writeback will not be performed.

With the current implementation, a service function cannot be reentrant. Always confirm that execution has ended before calling the next function.

The "*ssize*" and "*rsize*" maximum is the maximum DMA 1Mbyte - 16 bytes which can be sent at one time.

Since the completion processing function end_func is executed as an interrupt handler, special care is required when programming. Refer to the “Interrupt Handler Descriptions” section of \overview\eekernel for details.

Return value

0:	Notification to server succeeded
Negative (<0):	Execution failed

sceSifCheckStatRpc

Determine RPC status

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

int sceSifCheckStatRpc(

sceSifRpcData **bd***)**	Pointer to sceSifRpcData structure

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function determines the state of the sceSifBindRpc(), sceSifCallRpc(), and sceSifGetOtherData() functions. It is primarily used to determine the end of execution when the function was invoked with SIF_RPCM_NOWAIT.

For the *bd* argument, specify sceSifClientData or sceSifReceiveData by casting it to sceSifRpcData.

Return value

1: Execution in progress

0: Execution ended

sceSifExecRequest

Execute service function

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

void sceSifExecRequest(

sceSifServeData ****sd*);**	Pointer to request

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Executes the service function corresponding to a request.

This function is required for the server.

Return value

None

sceSifGetNextRequest

Get RPC request

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

sceSifServeData * sceSifGetNextRequest(

sceSifQueueData ****dp*)**	Pointer to request receive queue set by the sceSifSetRpcQueue() function

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Gets the sceSifQueueData structure, which represents the received request, from the receive queue. This function is required for the server.

If the return value is not zero, the service function will be executed when the pointer is passed to the sceSifExecRequest() function.

Return value

0:	No request
Non-zero:	Pointer to request

sceSifGetOtherData

Fetch target-side data

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

int sceSifGetOtherData(

sceSifReceiveData **bd,*	Pointer to sceSifReceiveData structure
void **src,*	Target-side data address (16-byte/4-byte alignment for EE/IOP)
void **dest,*	Transfer destination address (16-byte/4-byte alignment for EE/IOP)
int *size,*	Size of data to be transferred
unsigned int *mode***)**	Calling mode. This is usually zero. Specify the following constant as a mask when necessary: SIF_RPCM_NOWAIT: Asynchronous execution

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function transfers data from the target-side address *src* to the local-side address *dest*.

Normally, the thread that called the sceSifreceiveRpc() function is in Sleep state until there is a reply from the target. If SIF_RPCM_NOWAIT has been masked in advance for *mode*, control exits directly without the thread entering Sleep state. In this case, the completion of processing can be confirmed by using the sceSifCheckStatRpc() function.

Notes

From release 1.4, the EE uses the internally reserved semaphore to wait for completion, instead of Sleep.

Return value

0:	Notification to target side succeeded
Negative (<0):	Execution failed

sceSifInitRpc

Initialize SIF RPC API

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>	
void sceSifInitRpc(	
unsigned int *mode***)**	Startup mode (In the current implementation, this value is fixed at 0.)

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Initializes the SIF RPC API.

Registers a command function for initializing internal variables and processing requests in the system buffer of the SIF Command API.

This function must be executed on both the server and client.

Since the sceSifInitRpc() function internally calls the sceSifInitCmd() function, for synchronization purposes, one side will enter a wait state within this function until the function is called by the other side.

Return value

None

sceSifRegisterRpc

Register RPC service function in receive queue

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

void sceSifRegisterRpc(

sceSifServeData **serve,*	Pointer to structure for storing service function information
unsigned int *request,*	Request identifier
sceSifRpcFunc *func,*	Service function to be executed when request is received
void **buff,*	Data address that is argument of *func*
sceSifRpcFunc *cfunc,*	Function that is executed when interrupts are disabled and sceSifCancelRpc() is invoked
void **cbuff,*	Buffer that is argument of *cfunc*
sceSifQueueData **qd***)**	Receive queue structure for registering serve structure

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Registers the specified request identifier and service function in the receive queue structure. This function is required for the server.

The request identifier is one that is used by the sceSifBindRpc() function to search for a service function. A request identifier for which the highest-order bit (bit 31) is set to 1 cannot be specified (this kind of request identifier is for system use).

If the sceSifCallRpc() function is called from the client side, the request is entered in the receive queue by the function that was registered using the SIF Command API. After the SIF Command function that is executed as an interrupt function ends, the request is fetched from the receive queue using a normal context, and the service function func is executed. When *func* execution ends, the data at the address specified by its return value is returned to the address specified by the receive argument of the sceSifCallRpc() function. However, the size of the data that is returned is limited to the size specified by the *rsize* argument of the sceSifCallRpc() function.

With the current implementation, a service function cannot be reentrant. Always confirm that execution of the current function completes before calling the next function. Even if the function itself is reentrant, this restriction holds to reduce the queuing structure and the amount of traffic between the EE and IOP.

sceSifCancelRpc() is currently not implemented.

Return value

None

sceSifRemoveRpc

Remove RPC service functions

Library	*Introduced*	*Documentation last modified*
sifrpc	1.5	July 2, 2001

Syntax

#include <sifrpc.h>

sceSifServeData *sceSifRegisterRpc(

sceSifServeData **serve,*	Pointer to structure where service function information is stored
sceSifQueueData **qd)*	Receive queue structure which registers the *serve* structure

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Removes the request identifier and service functions from the receive queue structure.

When unloading the IOP module, the receive queue registered by this function must be removed from the queue structure.

Return value

NULL:	Failure (not registered in the receive queue)
Other than NULL:	Success

sceSifRemoveRpcQueue

Remove RPC receive queue registration

Library	*Introduced*	*Documentation last modified*
sifrpc	1.5	July 2, 2001

Syntax

#include <sifrpc.h>

sceSifQueueData *sceSifRemoveRpcQueue(

sceSifQueueData **dq*)	Receive queue structure

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Removes the receive queue of the RPC request from the RPC system.

When removing the IOP module, remove the queue structure registered in the RPC system.

Return value

NULL:	Failure (not registered)
Other than NULL:	Success

sceSifRpcLoop

Wait for request

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

void sceSifRpcLoop(

sceSifQueueData ****pd*)**	Pointer to request receive queue set by the sceSifSetRpcQueue() function

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Causes the execution thread to enter a loop in which it repeatedly waits for a request and executes a service. This function is required for the server.

When this function is called, the execution thread enters Sleep state until a request from the client is received. When a request is received, Wakeup is invoked, and the service function of the request is executed. After execution ends, the execution thread enters Sleep state again. Therefore, the request receive queue specified by *pd* must be the queue for which the thread ID was specified in the second argument, at the time it was registered by the sceSifSetRpcQueue() function.

The source code of sceSifRpcLoop() is shown below for reference.

```
void sceSifRpcLoop(sceSifQueueData *qd)
{
       sceSifServeData *rdp;
       while(1) {
              /* Get processing function */
              while ((rdp = sceSifGetNextRequest(qd))){
              /* Execute function */
                    sceSifExecRequest(rdp);
              }
       /* Sleep until next command arrives */
              SleepThread();
       }
       return;
}
```

Return value

None (control does not return from this function)

sceSifSetRpcQueue

Register RPC receive queue

Library	*Introduced*	*Documentation last modified*
sifrpc	1.1	March 26, 2001

Syntax

#include <sifrpc.h>

void sceSifSetRpcQueue(

sceSifQueueData **dq,*	Receive queue structure
int *key***)**	Thread ID. A negative number (<0) will cause a busy wait to occur.

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Registers an RPC request receive queue in the RPC system.

This function is required for the server.

Normally a thread ID is specified for the *key* argument so that Wakeup will be performed for the thread each time a request arrives from a client. If a negative number (<0) is specified for *key*, Wakeup will not be performed and a busy wait will occur.

Return value

None