

Controller Service Reference

© 2014 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

Table of Contents

Constants	3
Basic Button State Constants.....	4
SCE_CTRL_INTERCEPTED.....	6
Controller Mode Constants	7
Wireless Controller Connection State Constants	8
Error Codes.....	9
Datatypes.....	10
SceCtrlData	11
SceCtrlRapidFireRule	12
Datatypes for the Wireless Controller.....	13
SceCtrlData2	14
SceCtrlWirelessControllerInfo	16
Mode Setting Functions.....	17
sceCtrlSetSamplingMode	18
sceCtrlGetSamplingMode	19
Data Obtaining Functions.....	20
sceCtrlPeekBufferPositive, sceCtrlPeekBufferNegative.....	21
sceCtrlReadBufferPositive, sceCtrlReadBufferNegative	23
Data Obtaining Functions for the Wireless Controller.....	25
sceCtrlPeekBufferPositive2, sceCtrlPeekBufferNegative2	26
sceCtrlReadBufferPositive2, sceCtrlReadBufferNegative2	28
Debug Support Functions	30
sceCtrlClearRapidFire.....	31
sceCtrlSetRapidFire	32
Information Obtaining Functions for the Wireless Controller.....	34
sceCtrlIsMultiControllerSupported	35
sceCtrlGetWirelessControllerInfo.....	36

Constants

000004892117

Basic Button State Constants

Basic button state constants

Definition

```
#include <ctrl.h>
#define Value (Number)
```

Value	(Number)	Description
SCE_CTRL_SELECT	(1<<0)	SELECT button/SHARE button
SCE_CTRL_L3	(1<<1)	L3 button
SCE_CTRL_R3	(1<<2)	R3 button
SCE_CTRL_START	(1<<3)	START button/OPTIONS button
SCE_CTRL_UP	(1<<4)	Up button
SCE_CTRL_Lup		
SCE_CTRL_RIGHT	(1<<5)	Right button
SCE_CTRL_Lright		
SCE_CTRL_DOWN	(1<<6)	Down button
SCE_CTRL_Ldown		
SCE_CTRL_LEFT	(1<<7)	Left button
SCE_CTRL_Lleft		
SCE_CTRL_L	(1<<8)	L button
SCE_CTRL_R	(1<<9)	R button
SCE_CTRL_L2	(1<<8)	L2 button
SCE_CTRL_R2	(1<<9)	R2 button
SCE_CTRL_L1	(1<<10)	L1 button
SCE_CTRL_R1	(1<<11)	R1 button
SCE_CTRL_TRIANGLE	(1<<12)	Triangle button
SCE_CTRL_Rup		
SCE_CTRL_CIRCLE	(1<<13)	Circle button
SCE_CTRL_Rright		
SCE_CTRL_CROSS	(1<<14)	Cross button
SCE_CTRL_Rdown		
SCE_CTRL_SQUARE	(1<<15)	Square button
SCE_CTRL_Rleft		

Description

These constants indicate the bit positions representing the pressed/ not pressed state of each basic button to be stored in the *buttons* member of the *SceCtrlData*/*SceCtrlData2* structure, which stores controller state information.

When using *sceCtrlPeekBufferPositive()*, *sceCtrlPeekBufferNegative()*, *sceCtrlReadBufferPositive()* or *sceCtrlReadBufferNegative()* to obtain *SceCtrlData*, *SCE_CTRL_L* and *SCE_CTRL_R* will be contained in the *buttons* member. Note that *SCE_CTRL_L1*, *SCE_CTRL_R1*, *SCE_CTRL_L2*, *SCE_CTRL_R2*, *SCE_CTRL_L3*, and *SCE_CTRL_R3* will not be contained in the *buttons* member.

When using *sceCtrlPeekBufferPositive2()*, *sceCtrlPeekBufferNegative2()*, *sceCtrlReadBufferPositive2()* or *sceCtrlReadBufferNegative2()* to obtain *SceCtrlData2*, *SCE_CTRL_L1*, *SCE_CTRL_R1*, *SCE_CTRL_L2*, *SCE_CTRL_R2*, *SCE_CTRL_L3*, and *SCE_CTRL_R3* will be contained in the *buttons* member. Note that *SCE_CTRL_L* and *SCE_CTRL_R* will not be contained in the *buttons* member.

SCE CONFIDENTIAL

See Also

SceCtrlData, SceCtrlData2

000004892117

SCE CONFIDENTIAL

SCE_CTRL_INTERCEPTED

Controller intercepted constant

Definition

```
#include <ctrl.h>
#define SCE_CTRL_INTERCEPTED (1<<16)
```

Description

This constant indicates the bit position representing controller input interception to be stored in the *buttons* member of the *SceCtrlData*/*SceCtrlData2* structure, which stores controller state information.

See Also

SceCtrlData, *SceCtrlData2*

Controller Mode Constants

Controller mode constants

Definition

```
#include <ctrl.h>
#define SCE_CTRL_MODE_DIGITALONLY 0
#define SCE_CTRL_MODE_DIGITALANALOG 1
#define SCE_CTRL_MODE_DIGITALANALOG_WIDE 2
```

Description

These constants are used when setting the controller mode with `sceCtrlSetSamplingMode()`.

`SCE_CTRL_MODE_DIGITALONLY` is a constant that indicates the buttons only mode in which only buttons are used and the analog sticks are not.

The `SCE_CTRL_MODE_DIGITALANALOG` is a constant that represents the buttons/analog sticks mode with analog sticks operating in the normal mode.

`SCE_CTRL_MODE_DIGITALANALOG_WIDE` is a constant that indicates the buttons/analog sticks mode with analog sticks operating in the wide mode.

For details on the operation modes of analog sticks, refer to the "Operation Modes of Analog Sticks" section in the "Controller Service Overview" document.

See Also

`sceCtrlSetSamplingMode()`, `sceCtrlGetSamplingMode()`

Wireless Controller Connection State Constants

Wireless controller connection state constants

Definition

```
#include <ctrl.h>
#define SCE_CTRL_WIRELESS_INFO_NOT_CONNECTED    0
#define SCE_CTRL_WIRELESS_INFO_CONNECTED       1
```

Description

These constants are used when obtaining information of the wireless controller with `sceCtrlGetWirelessControllerInfo()`.

`SCE_CTRL_WIRELESS_INFO_NOT_CONNECTED` represents that a wireless controller is not connected.

`SCE_CTRL_WIRELESS_INFO_CONNECTED` represents that a wireless controller is connected.

See Also

`sceCtrlGetWirelessControllerInfo()`

SCE CONFIDENTIAL

Error Codes

Error codes

Definition

Value	(Number)	Description
SCE_CTRL_ERROR_INVALID_ARG	0x80340001	Invalid argument specified
SCE_CTRL_ERROR_PRIV_REQUIRED	0x80340002	Access with invalid privilege
SCE_CTRL_ERROR_NO_DEVICE	0x80340020	Specified device does not exist
SCE_CTRL_ERROR_NOT_SUPPORTED	0x80340021	Not supported
SCE_CTRL_ERROR_INVALID_MODE	0x80340022	Invalid mode
SCE_CTRL_ERROR_FATAL	0x803400FF	Fatal error occurred

Datatypes

000004892117

SceCtrlData

Controller state information

Definition

```
#include <ctrl.h>
typedef struct SceCtrlData {
    SceUInt64 timeStamp;
    SceUInt32 buttons;
    SceUInt8 lx;
    SceUInt8 ly;
    SceUInt8 rx;
    SceUInt8 ry;
    SceUInt8 rsrv[16];
} SceCtrlData;
```

Members

<i>timeStamp</i>	Time stamp in the Controller Service (process time: μ sec)
<i>buttons</i>	Button state information *Includes SCE_CTRL_L and SCE_CTRL_R (excludes SCE_CTRL_L1, SCE_CTRL_R1, SCE_CTRL_L2, SCE_CTRL_R2, SCE_CTRL_L3, and SCE_CTRL_R3)
<i>lx</i>	Left stick X axis (left 0x00 to right 0xff)
<i>ly</i>	Left stick Y axis (top 0x00 to bottom 0xff)
<i>rx</i>	Right stick X axis (left 0x00 to right 0xff)
<i>ry</i>	Right stick Y axis (top 0x00 to bottom 0xff)
<i>rsrv</i>	Reserved area

Description

This structure is for obtaining controller state information.

The time stamp of the time (process time) at which sampling was performed and data was obtained is placed in the *timeStamp* member.

32-bit button state information is placed in the *buttons* member. Using the basic button state constants and SCE_CTRL_INTERCEPTED, an application can determine the pressed/not pressed state of each button, etc.

The analog stick coordinate data, with 0x80 taken to be the center, is placed in the *lx*, *ly*, *rx* and *ry* members.

See Also

Basic Button State Constants, SCE_CTRL_INTERCEPTED, *sceCtrlPeekBufferPositive()*, *sceCtrlPeekBufferNegative()*, *sceCtrlReadBufferPositive()*, *sceCtrlReadBufferNegative()*

SceCtrlRapidFireRule

Set button rapid-fire rules

Definition

```
#include <ctrl.h>
typedef struct SceCtrlRapidFireRule {
    SceUInt32 uiMask;
    SceUInt32 uiTrigger;
    SceUInt32 uiTarget;
    SceUInt32 uiDelay;
    SceUInt32 uiMake;
    SceUInt32 uiBreak;
} SceCtrlRapidFireRule;
```

Members

<i>uiMask</i>	Comparison mask of the button operation for rapid-fire trigger
<i>uiTrigger</i>	Button operation for rapid-fire trigger
<i>uiTarget</i>	Button for which rapid-fire input is performed
<i>uiDelay</i>	Dead time of rapid-fire trigger (sampling count)
<i>uiMake</i>	Button press time (sampling count)
<i>uiBreak</i>	Button release time (sampling count)

Description

This structure is used as the argument for setting the rules for the button rapid-fire functionality with `sceCtrlSetRapidFire()` mainly for debugging.

For details, refer to `sceCtrlSetRapidFire()`.

See Also

`sceCtrlSetRapidFire()`

Datatypes for the Wireless Controller

SCE CONFIDENTIAL

SceCtrlData2

Controller state information

Definition

```
#include <ctrl.h>
typedef struct SceCtrlData2 {
    SceUInt64 timeStamp;
    SceUInt32 buttons;
    SceUInt8 lx;
    SceUInt8 ly;
    SceUInt8 rx;
    SceUInt8 ry;
    SceUInt8 up;
    SceUInt8 right;
    SceUInt8 down;
    SceUInt8 left;
    SceUInt8 l2;
    SceUInt8 r2;
    SceUInt8 l1;
    SceUInt8 r1;
    SceUInt8 triangle;
    SceUInt8 circle;
    SceUInt8 cross;
    SceUInt8 square;
    SceUInt8 rsrv[4];
} SceCtrlData2;
```

Members

<i>timeStamp</i>	Time stamp in the Controller Service (process time: μ sec)
<i>buttons</i>	Button state information *Includes SCE_CTRL_L1, SCE_CTRL_R1, SCE_CTRL_L2, SCE_CTRL_R2, SCE_CTRL_L3, and SCE_CTRL_R3 (excludes SCE_CTRL_L and SCE_CTRL_R)
<i>lx</i>	Left stick X axis (left 0x00 to right 0xff)
<i>ly</i>	Left stick Y axis (top 0x00 to bottom 0xff)
<i>rx</i>	Right stick X axis (left 0x00 to right 0xff)
<i>ry</i>	Right stick Y axis (top 0x00 to bottom 0xff)
<i>up</i>	Pressure sensitivity value of the up key (disabled)
<i>right</i>	Pressure sensitivity value of the right key (disabled)
<i>down</i>	Pressure sensitivity value of the down key (disabled)
<i>left</i>	Pressure sensitivity value of the left key (disabled)
<i>l2</i>	Pressure sensitivity value of the L2 button
<i>r2</i>	Pressure sensitivity value of the R2 button
<i>l1</i>	Pressure sensitivity value of the L1 button (disabled)
<i>r1</i>	Pressure sensitivity value of the R1 button (disabled)
<i>triangle</i>	Pressure sensitivity value of the triangle button (disabled)
<i>circle</i>	Pressure sensitivity value of the circle button (disabled)
<i>cross</i>	Pressure sensitivity value of the cross button (disabled)
<i>square</i>	Pressure sensitivity value of the square button (disabled)
<i>rsrv</i>	Reserved area

Description

This structure is for obtaining controller state information.

The *timeStamp* member stores the timestamp (process time) at which sampling was performed and data was obtained.

The *buttons* member stores 32-bit button state information. An application can use basic button state constants and `SCE_CTRL_INTERCEPTED` to determine the pressed/not pressed state of each button, etc.

The *lx*, *ly*, *rx*, and *ry* members store coordinate data of the analog sticks with 0x80 as the center.

The *l2* and *r2* members store pressure sensitive values from 0x00 to 0xFF.

Values for the *up*, *right*, *down*, *left*, *l1*, *r1*, *triangle*, *circle*, *cross*, and *square* are disabled. Obtain each button's state from the *buttons* member.

See Also

Basic Button State Constants, `SCE_CTRL_INTERCEPTED`, `sceCtrlPeekBufferPositive2()`, `sceCtrlPeekBufferNegative2()`, `sceCtrlReadBufferPositive2()`, `sceCtrlReadBufferNegative2()`

SceCtrlWirelessControllerInfo

Wireless controller information

Definition

```
#include <ctrl.h>
#define SCE_CTRL_MAX_WIRELESS_NUM 4
typedef struct SceCtrlWirelessControllerInfo {
    SceBool connected[SCE_CTRL_MAX_WIRELESS_NUM];
    SceInt32 reserved[12];
} SceCtrlWirelessControllerInfo;
```

Members

connected Connection state
reserved Reserved area

Description

This structure is for storing wireless controller information.

For the *connected* member, the connection state of the wireless controller will be stored.

Values will be stored in an array as follows for *connected*.

Member	Description
<i>connected</i> [0]	Connection state of the Controller Number 1 wireless controller
<i>connected</i> [1]	Connection state of the Controller Number 2 wireless controller
<i>connected</i> [2]	Connection state of the Controller Number 3 wireless controller
<i>connected</i> [3]	Connection state of the Controller Number 4 wireless controller

The maximum number of wireless controllers that can be connected is four.

See Also

`sceCtrlGetWirelessControllerInfo()`

Mode Setting Functions

SCE CONFIDENTIAL

sceCtrlSetSamplingMode

Setting the controller mode

Definition

```
#include <ctrl.h>
int sceCtrlSetSamplingMode (
    SceUInt32 uiMode
);
```

Arguments

uiMode Controller mode

Return value

Returns the old mode value which had been specified up to that point for normal termination.
If an error occurs, a negative value is returned.
For details on the error, refer to "Error Codes."

Description

This function sets the controller mode.

Set the controller mode constant to the *uiMode* argument.

The default is buttons only mode (SCE_CTRL_MODE_DIGITALONLY).

See Also

`sceCtrlGetSamplingMode()`, Controller Mode Constants

SCE CONFIDENTIAL

sceCtrlGetSamplingMode

Get controller mode

Definition

```
#include <ctrl.h>
int sceCtrlGetSamplingMode (
    SceUInt32 *puiMode
);
```

Arguments

puiMode Pointer to the SceUInt32 variable which is to receive the controller mode

Return value

Returns the controller mode setting value that is currently set for normal termination.
If an error occurs, a negative value is returned.
For details on the error, refer to "Error Codes."

Description

This function obtains the controller mode setting value that is currently set.
The setting can be changed with the `sceCtrlSetSamplingMode()` function.

See Also

`sceCtrlSetSamplingMode()`, Controller Mode Constants

Data Obtaining Functions

000004892117

sceCtrlPeekBufferPositive, sceCtrlPeekBufferNegative

Obtain controller state information by polling

Definition

```
#include <ctrl.h>
int sceCtrlPeekBufferPositive (
    SceInt32 port,
    SceCtrlData *pData,
    SceInt32 nBufs
);

#include <ctrl.h>
int sceCtrlPeekBufferNegative (
    SceInt32 port,
    SceCtrlData *pData,
    SceInt32 nBufs
);
```

Arguments

<i>port</i>	Type of data to obtain
<i>pData</i>	Buffers to receive controller data
<i>nBufs</i>	Number of buffers to receive controller data (1 to 64)

Return value

Stores the obtained controller data in *pData* and returns the number of obtained data for normal termination. The value is in the range of 1 to *nBufs*.

If an error occurs, a negative value is returned.

For details on the error, refer to "Error Codes."

Description

These functions obtain controller state information.

Each bit of the *buttons* member of the *SceCtrlData* structure to be stored in the *pData* argument will be as follows when the corresponding button is pressed.

- *sceCtrlPeekBufferPositive()*: 1 (positive logic)
- *sceCtrlPeekBufferNegative()*: 0 (negative logic)

The *buttons* member of the *SceCtrlData* structure will contain *SCE_CTRL_L* and *SCE_CTRL_R* (it will not contain *SCE_CTRL_L1*, *SCE_CTRL_R1*, *SCE_CTRL_L2*, *SCE_CTRL_R2*, *SCE_CTRL_L3*, and *SCE_CTRL_R3*).

The most recent 64 sets of controller information are stored in buffers of the Controller Service and *sceCtrlPeekBufferPositive()/sceCtrlPeekBufferNegative()* obtains the controller information of the number specified in the *nBufs* argument from the newest buffers by polling.

Buffer contents are updated by an interrupt every time the controller is sampled. Because of this, depending on the timing at which

sceCtrlPeekBufferPositive()/sceCtrlPeekBufferNegative() is called, the data that is obtained may differ in terms of being data before/after buffer updates (usually carried out at the VSYNC period).

When using this function for PlayStation®Vita, specify 0 to *port*.

When using this function for PlayStation®TV, specify one of the following values to *port*.

<i>port</i>	Description
0	Obtains merged data of the Controller Number 1 wireless controller, BD remote control, and a TV remote control from HDMI CEC remote passthrough.
1	Obtains data of the Controller Number 1 wireless controller.
2	Obtains data of the Controller Number 2 wireless controller.
3	Obtains data of the Controller Number 3 wireless controller.
4	Obtains data of the Controller Number 4 wireless controller.

Data obtained when *port*=0 is the merged value of controller and remote control inputs. It is recommended that 1 be specified to *port* when obtaining information of a single wireless controller.

Notes

The similar functions of `sceCtrlReadBufferPositive()` and `sceCtrlReadBufferNegative()` obtain controller state information by blocking; when calling either of these functions earlier than the sampling interval, the function will block the thread to obtain the most recent data.

See Also

`SceCtrlData`

sceCtrlReadBufferPositive, sceCtrlReadBufferNegative

Obtain controller state information by blocking

Definition

```
#include <ctrl.h>
int sceCtrlReadBufferPositive (
    SceInt32 port,
    SceCtrlData *pData,
    SceInt32 nBufs
);

#include <ctrl.h>
int sceCtrlReadBufferNegative (
    SceInt32 port,
    SceCtrlData *pData,
    SceInt32 nBufs
);
```

Arguments

<i>port</i>	Type of data to obtain
<i>pData</i>	Buffers to receive controller data
<i>nBufs</i>	Number of buffers to receive controller data (1 to 64)

Return value

Stores the obtained controller data in *pData* and returns the number of obtained data for normal termination. The value is in the range of 1 to *nBufs*.

If an error occurs, a negative value is returned.

For details on the error, refer to "Error Codes."

Description

These functions obtain controller state information.

Each bit of the *buttons* member of the *SceCtrlData* structure to be stored in the *pData* argument will be as follows when the corresponding button is pressed.

- *sceCtrlReadBufferPositive()*: 1 (positive logic)
- *sceCtrlReadBufferNegative()*: 0 (negative logic)

The *buttons* member of the *SceCtrlData* structure will contain SCE_CTRL_L and SCE_CTRL_R (it will not contain SCE_CTRL_L1, SCE_CTRL_R1, SCE_CTRL_L2, SCE_CTRL_R2, SCE_CTRL_L3, and SCE_CTRL_R3).

The Controller Service holds the most recent 64 sets of controller state information at maximum in its buffers. *sceCtrlReadBufferPositive()/sceCtrlReadBufferNegative()* obtains the controller information of the number specified in the *nBufs* argument the position of the current read pointer and then updates the read pointer..

If the controller was not sampled even once during the period from the last call of the `sceCtrlReadBufferPositive()`/`sceCtrlReadBufferNegative()` function until the current call of the `sceCtrlReadBufferPositive()`/`sceCtrlReadBufferNegative()` function, the thread is blocked in the `sceCtrlReadBufferPositive()`/`sceCtrlReadBufferNegative()` function until the controller is sampled again. Consequently, the return value will always be 1 or greater, and the most recent data will be obtained.

In addition, when the `sceCtrlReadBufferPositive()`/`sceCtrlReadBufferNegative()` function cannot be called at each specified sampling interval due to a processing lag (for example), set the value of `nBuf` to 2 or greater; based on whether or not the return value is greater than 1, it will be possible to determine whether or not there was a processing lag.

When using this function for PlayStation®Vita, specify 0 to `port`.

When using this function for PlayStation®TV, specify one of the following values to `port`.

<code>port</code>	Description
0	Obtains merged data of the Controller Number 1 wireless controller, BD remote control, and a TV remote control from HDMI CEC remote passthrough.
1	Obtains data of the Controller Number 1 wireless controller.
2	Obtains data of the Controller Number 2 wireless controller.
3	Obtains data of the Controller Number 3 wireless controller.
4	Obtains data of the Controller Number 4 wireless controller.

Data obtained when `port=0` is the merged value of controller and remote control inputs. It is recommended that 1 be specified to `port` when obtaining information of a single wireless controller.

Notes

The similar functions of `sceCtrlPeekBufferPositive()` and `sceCtrlPeekBufferNegative()` obtain controller state information by snooping; when calling either of these functions earlier than the sampling interval, the function will not block the thread and the last sampling result will be repeatedly obtained.

See Also

`SceCtrlData`

Data Obtaining Functions for the Wireless Controller

SCE CONFIDENTIAL

sceCtrlPeekBufferPositive2, sceCtrlPeekBufferNegative2

Obtain controller state information by polling

Definition

```
#include <ctrl.h>
int sceCtrlPeekBufferPositive2 (
    SceInt32 port,
    SceCtrlData2 *pData,
    SceInt32 nBufs
);

#include <ctrl.h>
int sceCtrlPeekBufferNegative2 (
    SceInt32 port,
    SceCtrlData2 *pData,
    SceInt32 nBufs
);
```

Arguments

<i>port</i>	Type of data to obtain
<i>pData</i>	Buffers to receive controller data
<i>nBufs</i>	Number of buffers to receive controller data (1 to 64)

Return Values

Stores the obtained controller data in *pData* and returns the number of obtained data for normal termination. The value is in the range of 1 to *nBufs*.

If an error occurs, a negative value is returned.

For details on the error, refer to "Error Codes."

Description

These functions obtain controller state information.

Each bit of the *buttons* member of the *SceCtrlData2* structure to be stored in the *pData* argument will be as follows when the corresponding button is pressed.

- *sceCtrlPeekBufferPositive2*(): 1 (positive logic)
- *sceCtrlPeekBufferNegative2*(): 0 (negative logic)

The *buttons* member of the *SceCtrlData2* structure will contain SCE_CTRL_L1, SCE_CTRL_R1, SCE_CTRL_L2, SCE_CTRL_R2, SCE_CTRL_L3, and SCE_CTRL_R3 (it will not contain SCE_CTRL_L and SCE_CTRL_R).

The Controller Service holds the most recent 64 sets of controller state information at maximum in its buffers. *sceCtrlPeekBufferPositive2*()/*sceCtrlPeekBufferNegative2*() obtains the controller information of the number specified in the *nBufs* argument from the newest buffers by polling.

Buffer contents are updated by an interrupt every time the controller is sampled. Because of this, depending on the timing at which

`sceCtrlPeekBufferPositive2()` / `sceCtrlPeekBufferNegative2()` is called, the data that is obtained may differ in terms of being data before/after buffer updates (usually carried out at the VSYNC period).

When using this function for PlayStation®Vita, specify 0 to *port*.

When using this function for PlayStation®TV, specify one of the following values to *port*.

<i>port</i>	Description
0	Obtains merged data of the Controller Number 1 wireless controller, BD remote control, and a TV remote control from HDMI CEC remote passthrough.
1	Obtains data of the Controller Number 1 wireless controller.
2	Obtains data of the Controller Number 2 wireless controller.
3	Obtains data of the Controller Number 3 wireless controller.
4	Obtains data of the Controller Number 4 wireless controller.

Data obtained when *port*=0 is the merged value of controller and remote control inputs. It is recommended that 1 be specified to *port* when obtaining information of a single wireless controller.

Notes

The similar functions of `sceCtrlReadBufferPositive2()` and `sceCtrlReadBufferNegative2()` obtain controller state information by blocking; when calling either of these functions earlier than the sampling interval, the function will block the thread to obtain the most recent data.

See Also

`SceCtrlData2`

sceCtrlReadBufferPositive2, sceCtrlReadBufferNegative2

Obtain controller state information by blocking

Definition

```
#include <ctrl.h>
int sceCtrlReadBufferPositive2 (
    SceInt32 port,
    SceCtrlData2 *pData,
    SceInt32 nBufs
);

#include <ctrl.h>
int sceCtrlReadBufferNegative2 (
    SceInt32 port,
    SceCtrlData2 *pData,
    SceInt32 nBufs
);
```

Arguments

<i>port</i>	Type of data to obtain
<i>pData</i>	Buffers to receive controller data
<i>nBufs</i>	Number of buffers to receive controller data (1 to 64)

Return Values

Stores the obtained controller data in *pData* and returns the number of obtained data for normal termination. The value is in the range of 1 to *nBufs*.

If an error occurs, a negative value is returned.

For details on the error, refer to "Error Codes."

Description

These functions obtain controller state information.

Each bit of the *buttons* member of the *SceCtrlData2* structure to be stored in the *pData* argument will be as follows when the corresponding button is pressed.

- *sceCtrlReadBufferPositive2()*: 1 (positive logic)
- *sceCtrlReadBufferNegative2()*: 0 (negative logic)

The *buttons* member of the *SceCtrlData2* structure will contain SCE_CTRL_L1, SCE_CTRL_R1, SCE_CTRL_L2, SCE_CTRL_R2, SCE_CTRL_L3, and SCE_CTRL_R3 (it will not contain SCE_CTRL_L and SCE_CTRL_R).

The most recent 64 sets of controller information are stored in buffers of the Controller Service and *sceCtrlReadBufferPositive2()*/*sceCtrlReadBufferNegative2()* obtains the controller information of the number specified in the *nBufs* argument from the position of the current read pointer and then updates the read pointer.

If the controller was not sampled even once during the period from the last call of the `sceCtrlReadBufferPositive2()`/`sceCtrlReadBufferNegative2()` function until the current call of the `sceCtrlReadBufferPositive2()`/`sceCtrlReadBufferNegative2()` function, the thread is blocked within the

`sceCtrlReadBufferPositive2()`/`sceCtrlReadBufferNegative2()` function until the controller is sampled again. Consequently, the return value will always be 1 or greater, and the most recent data will be obtained.

In addition, when the `sceCtrlReadBufferPositive2()`/`sceCtrlReadBufferNegative2()` function cannot be called at the set sampling interval due to a processing lag (for example), set the value of `nBuf` to 2 or greater; based on whether or not the return value is greater than 1, it will be possible to determine whether or not there was a processing lag.

When using this function for PlayStation®Vita, specify 0 to `port`.

When using this function for PlayStation®TV, specify one of the following values to `port`.

<code>port</code>	Description
0	Obtains merged data of the Controller Number 1 wireless controller, BD remote control, and a TV remote control from HDMI CEC remote passthrough.
1	Obtains data of the Controller Number 1 wireless controller.
2	Obtains data of the Controller Number 2 wireless controller.
3	Obtains data of the Controller Number 3 wireless controller.
4	Obtains data of the Controller Number 4 wireless controller.

Data obtained when `port=0` is the merged value of controller and remote control inputs. It is recommended that 1 be specified to `port` when obtaining information of a single wireless controller.

Notes

The similar functions of `sceCtrlPeekBufferPositive2()` and `sceCtrlPeekBufferNegative2()` obtain controller state information by snooping; when calling either of these functions earlier than the sampling interval, the function will not block the thread and the last sampling result will be repeatedly obtained.

See Also

`SceCtrlData2`

Debug Support Functions

SCE CONFIDENTIAL

sceCtrlClearRapidFire

Clear rapid-fire settings

Definition

```
#include <ctrl.h>
int sceCtrlClearRapidFire (
    SceInt32 port,
    SceInt32 idx
);
```

Arguments

<i>port</i>	Specify 0.
<i>idx</i>	Rule index for button rapid-fire (0 to 15)

Return value

Returns `SCE_OK (0)` for normal termination.
If an error occurs, a negative value is returned.
For details on the error, refer to "Error Codes."

Description

This function clears the button rapid-fire settings which were set using `sceCtrlSetRapidFire()`.
In the *idx* argument, select the rule index to clear.

See Also

`sceCtrlSetRapidFire()`

SCE CONFIDENTIAL

sceCtrlSetRapidFire

Set button rapid-fire rules

Definition

```
#include <ctrl.h>
int sceCtrlSetRapidFire (
    SceInt32 port,
    SceInt32 idx,
    const SceCtrlRapidFireRule* pRule
);
```

Arguments

<i>port</i>	Specify 0.
<i>idx</i>	Rule index for button rapid-fire (0 to 15)
<i>pRule</i>	Rapid-fire trigger rules

Return value

Returns `SCE_OK (0)` for normal termination.
 If an error occurs, a negative value is returned.
 For details on the error, refer to the section "Error Codes."

Description

This function sets the rules for the button rapid-fire functionality.

It will work on the retail units as well, but is primarily intended for debugging.

Place a rule index number from 0 to 15 in the *idx* argument. Up to 16 patterns can be stored as rules.

The button rapid-fire rules are set with the structure specified in the *pRule* argument.

For the set rules, when the value obtained by masking the button input with *pRule->uiMask* matches *pRule->uiTrigger*, the button specified with *pRule->uiTarget* is placed in the rapid-fire state. *pRule->uiMask*, *pRule->uiTrigger* and *pRule->uiTarget* all specify the logic in positive logic. The rapid-fire start timing is specified with the *pRule->uiDelay* argument, and the rapid-fire cycle is specified with *pRule->uiMake* and *pRule->uiBreak*.

Sample Settings

```

SceCtrlRapidFireRule rule1, rule2;

//rule1
rule1.uiMask    = SCE_CTRL_CIRCLE | SCE_CTRL_L; // Mask circle and L buttons
rule1.uiTrigger = SCE_CTRL_CIRCLE;             // If the circle button is in
the pressed state
rule1.uiTarget  = SCE_CTRL_SQUARE;              // Using the square button,
rule1.uiDelay   = 10;                          // after a 10-cycle delay,
rule1.uiMake    = 1;                          // with a (1+1)-cycle period:
ON→OFF→
rule1.uiBreak   = 1;                          //

//Set rule 1 to index 0
sceCtrlSetRapidFire(0,          // port=0
                    0,          // idx = 0
                    &rule1); // rule1

//rule2
rule2.uiMask    = SCE_CTRL_TRIANGLE; // Mask triangle button
rule2.uiTrigger = SCE_CTRL_TRIANGLE; // If the triangle button is in the pressed
state
rule2.uiTarget  = SCE_CTRL_TRIANGLE; // Using the triangle button,
rule2.uiDelay   = 1;                  // after a 1-cycle delay,
rule2.uiMake    = 2;                  // with a (2+1) cycle period: ON→ON→OFF →
rule2.uiBreak   = 1;                  //

//Set rule 2 to index 1
sceCtrlSetRapidFire(0,          //port=0
                    1,          //idx =1
                    &rule2); //rule2

```

Notes

In this function, an error (SCE_CTRL_ERROR_INVALID_ARG) will occur if any one of the *uiDelay*, *uiMake* and *uiBreak* members of the *SceCtrlRapidFireRule* structure is set to 64 or greater. Because the maximum number of data buffers in the Controller Service is 64, setting rules exceeding this number of buffers is not allowed.

Furthermore, it is also prohibited to set 0 in all of *uiDelay*, *uiMake* and *uiBreak* because such setting does not make sense.

See Also

```
sceCtrlClearRapidFire()
```

Information Obtaining Functions for the Wireless Controller

SCE CONFIDENTIAL

sceCtrlsMultiControllerSupported

Get multiple wireless controller support state

Definition

```
#include <ctrl.h>
SceBool sceCtrlsMultiControllerSupported(void);
```

Arguments

None

Return Values

Returns true when multiple wireless controllers are supported and false when not supported.

Description

This function obtains whether or not the environment supports multiple wireless controllers.

SCE CONFIDENTIAL

sceCtrlGetWirelessControllerInfo

Get wireless controller information

Definition

```
#include <ctrl.h>
int sceCtrlGetWirelessControllerInfo (
    SceCtrlWirelessControllerInfo* pInfo
);
```

Arguments

pInfo Buffer to receive wireless controller information

Return Values

Returns SCE_OK(0) for normal termination.

If an error occurs, a negative value is returned.

For details on the error, refer to the section "Error Codes."

Description

This function obtains information of the wireless controller.

For details on wireless controller information, refer to the description for `SceCtrlWirelessControllerInfo`.

See Also

`SceCtrlWirelessControllerInfo`, Wireless Controller Connection State Constants