Controller Service Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Purpose and Characteristics

The Controller Service is a service for reading button and analog stick data from the PlayStation®Vita system and controllers, such as, the wireless controller.

Note
Wherever a distinction is not required in this document, the term "wireless controller" will be used in reference to the SIXAXIS™ wireless controller, the DUALSHOCK®3 wireless controller and the DUALSHOCK®4 wireless controller.

Note
Wherever a distinction is not required in this document, the term "analog sticks" will be used in reference to the left stick and right stick of PlayStation®Vita and of the wireless controller.

Main Features

- Obtaining data of controller buttons and analog sticks
- Emulation of button rapid-fire state

Embedding into a Program

The following files are required in order to use the Controller Service.

Filename	Description
libSceCtrl_stub.a	Stub library file
ctrl.h	Header file

Include ctrl.h in the source program. Various header files will be automatically included as well.

Upon building the program, link libSceCtrl_stub.a.

Sample Programs

Sample programs are provided in the following directory as examples of programs that use the Controller Service.

sample_code/input_output_devices/api_ctrl/basic

This sample shows the basic procedure for using the Controller Service.

2 Usage Procedure

Basic Usage Procedure (If Calls Can Be Made Properly at the VSYNC Period)

The Controller Service performs sampling of controller data at the VSYNC period.

Because of this, the application can correctly obtain data queued in the Controller Service by calling the `sceCtrlReadBufferPositive()` function at the VSYNC period.

Note
The `sceCtrlReadBufferPositive()` function obtains data in positive logic, where each bit corresponding to a pressed button will be 1.
To obtain data in negative logic, where each bit corresponding to a pressed button will be 0, use `sceCtrlReadBufferNegative()` instead of `sceCtrlReadBufferPositive()`.

(1) Initialization

The Controller Service runs as a default module. Explicit initialization is not necessary.

(2) Setting the Controller Mode

The mode in which the Controller Service performs sampling is set using the `sceCtrlSetSamplingMode()` function.

Controller modes are as follows.

Controller Mode	Description
Buttons only mode (`SCE_CTRL_MODE_DIGITALONLY`)	Only button data is read
Buttons/analog sticks mode (`SCE_CTRL_MODE_DIGITALANALOG`)	Both button and analog stick data are read (normal mode)
Buttons/analog sticks wide mode (`SCE_CTRL_MODE_DIGITALANALOG_WIDE`)	Both button and analog stick data are read (wide mode)

The default is the buttons only mode

If analog sticks aren't used, the buttons only mode will reduce power consumption, if only slightly, in comparison to the modes using both buttons and analog sticks.

Regarding the normal mode and wide mode, refer to the "Operation Modes of Analog Sticks" section in the "3 Precautions" chapter.

(3) Obtaining Data from the Controller Service

Controller state information can be obtained by blocking with the `sceCtrlReadBufferPositive()` function.

```
SceCtrlData ct;
int res;

res = sceCtrlReadBufferPositive(0, &ct, 1);
```

Controller state information obtained using the `sceCtrlReadBufferPositive()` function is stored in the `SceCtrlData` structure. Using the basic button state constants such as `SCE_CTRL_CIRCLE`, it is possible to obtain data on which buttons were pressed.

```
int IsCircleButtonPressed(void)
{
   SceCtrlData ct;
   int res;

   res = sceCtrlReadBufferPositive(0, &ct, 1);
   if (res < 0) {
        return (res);
   }

if ((ct.buttons & SCE_CTRL_CIRCLE)!=0) {
        //The circle button has been pressed
        return (1);
  } else {
        //The circle button has not been pressed
        return (0);
  }
}
```

If Calls May Be Made at Periods of Two VSYNCs or More

The `sceCtrlReadBufferPositive()` function may not be called at the ideal VSYNC period upon occurrence of a processing lag in the application.

If this is a possibility, obtain controller data as follows.

(1) Initialization

The Controller Service runs as a default module. Explicit initialization is not necessary.

(2) Setting the Controller Mode

The mode in which the Controller Service performs sampling is set using the `sceCtrlSetSamplingMode()` function.

Controller modes are as follows.

Controller Mode	Description
Buttons only mode (`SCE_CTRL_MODE_DIGITALONLY`)	Only button data is read
Buttons/analog sticks mode (`SCE_CTRL_MODE_DIGITALANALOG`)	Both button and analog stick data are read (normal mode)
Buttons/analog sticks wide mode (`SCE_CTRL_MODE_DIGITALANALOG_WIDE`)	Both button and analog stick data are read (wide mode)

The default is the buttons only mode.

If analog sticks aren't used, the buttons only mode will reduce power consumption, if only slightly, in comparison to the modes using both buttons and analog sticks.

Regarding the normal mode and wide mode, refer to the "Operation Modes of Analog Sticks" section in the "3 Precautions" chapter.

(3) Obtaining Data from the Controller Service

When calling the `sceCtrlReadBufferPositive()` function, specify multiple buffers in the argument.

For example, if a maximum of six periods' worth of data is to be obtained, specify 6 for the number of buffers to receive data. Doing this will make it possible to obtain, counting back from the newest data being buffered by the Controller Service, as many as six sets of controller state information.

The number of data actually obtained can be confirmed with the return value of the `sceCtrlReadBufferPositive()` function. If a data obtaining function is being called at the VSYNC period, the return value will be 1; if the return value is 2 or more, this means there was a processing lag.

```
SceCtrlData ct[6];
int res;

res = sceCtrlReadBufferPositive(0, ct, 6);
```

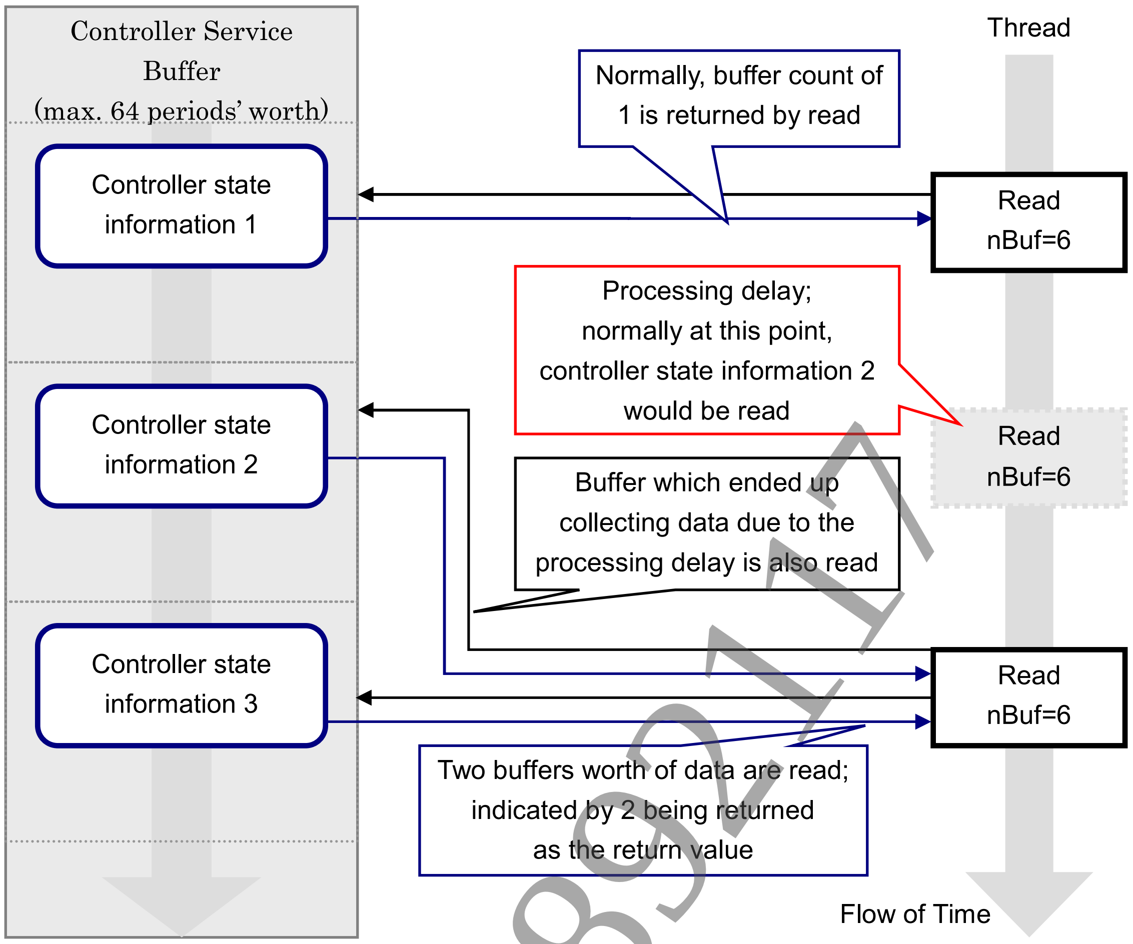
You can prevent missing key inputs by referencing past data as follows.

```
int IsCircleButtonPressed(void)
{
   SceCtrlData ct[6];
   unsigned int btn;
   int res;
   int i;

   res = sceCtrlReadBufferPositive(0, &ct, 6);
   if (res < 0) {
     return (res);
   }

   btn = 0;
   for (i=0; i<res; i++) {
      btn |= ct[i].buttons;
   }
if ((btn & SCE_CTRL_CIRCLE)!=0) {
      //The circle button has been pressed
       return (1);
  } else {
      //The circle button has not been pressed
      return (0);
  }
}
```

Within the Controller Service, up to a maximum of 64 buffers (periods' worth) of data are held per controller. In other words, the maximum number of buffers which can be specified by the `sceCtrlReadBufferPositive()` function is likewise 64.


Controller Service Buffer (max. 64 periods' worth)
Thread
Normally, buffer count of 1 is returned by read
Controller state information 1
Read nBuf=6
Processing delay; normally at this point, controller state information 2 would be read
Controller state information 2
Read nBuf=6
Buffer which ended up collecting data due to the processing delay is also read
Controller state information 3
Read nBuf=6
Two buffers worth of data are read; indicated by 2 being returned as the return value
Flow of Time

Usage Procedure of the Wireless Controller

When using a wireless controller, the `sceCtrlReadBufferPositive()` function is replaced by the `sceCtrlReadBufferPositive2()` function, for example, as the data obtaining function. Basic usage of the Controller Service is the same as when using the buttons and analog sticks of the PlayStation®Vita system. When using the wireless controller, implement your application in the same manner as described in the "Basic Usage Procedure (If Calls Can Be Made Properly at the VSYNC Period)" section and the "If Calls May Be Made at Periods of Two VSYNCs or More" section, and change the APIs used as follows.

PlayStation®Vita	Wireless Controller
`SceCtrlData`	`SceCtrlData2`
`sceCtrlPeekBufferPositive()`	`sceCtrlPeekBufferPositive2()`
`sceCtrlPeekBufferNegative()`	`sceCtrlPeekBufferNegative2()`
`sceCtrlReadBufferPositive()`	`sceCtrlReadBufferPositive2()`
`sceCtrlReadBufferNegative()`	`sceCtrlReadBufferNegative2()`

Use of Multiple Wireless Controllers

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

When using multiple wireless controllers, check that your environment supports multiple controllers. Call `sceCtrlIsMultiControllerSupported()`; if the return value is true, multiple wireless controllers can be used.

Call data obtaining functions per controller. For example, specify 1 to the *`port`* argument of `sceCtrlReadBufferPositive2()`, `sceCtrlReadBufferNegative2()`, `sceCtrlPeekBufferPositive2()`, and `sceCtrlPeekBufferNegative2()` to obtain data of Controller Number 1; specify 2 to the *`port`* argument of these functions to obtain data of Controller Number 2.

Moreover, the connected/disconnected state of the wireless controller can be checked with the *`connected`* member of the `SceCtrlWirelessControllerInfo` structure obtained with `sceCtrlGetWirelessControllerInfo()`. Note that if the state of the wireless controller specified to the *`port`* argument of a function such as `sceCtrlReadBufferPositive2()` is disconnected, the obtained data will be for a state of no operation.

Evaluating Press of the L Button and R Button on the PlayStation®Vita System

To evaluate whether or not the L button or R button of the PlayStation®Vita system is pressed, use the following data obtaining functions

```
sceCtrlPeekBufferPositive(),sceCtrlPeekBufferNegative()

sceCtrlReadBufferPositive(),sceCtrlReadBufferNegative()
```

Implement with the assumption that `SCE_CTRL_L` and `SCE_CTRL_R` will return to the *`buttons`* member of the obtained `SceCtrlData` structure.

On the other hand, do not implement with the assumption that `SCE_CTRL_L1`, `SCE_CTRL_R1`, `SCE_CTRL_L2`, `SCE_CTRL_R2`, `SCE_CTRL_L3`, and `SCE_CTRL_R3` will return.

Evaluating Press of the L1, R1, L2, R2, L3, and R3 Buttons on the Wireless Controller

To evaluate whether or not the L1 button, R1 button, L2 button, R2 button, L3 button, or R3 button on the wireless controller is pressed, use the following data obtaining functions for the wireless controller.

```
sceCtrlPeekBufferPositive2(),sceCtrlPeekBufferNegative2()

sceCtrlReadBufferPositive2(),sceCtrlReadBufferNegative2()
```

Implement with the assumption that `SCE_CTRL_L1`, `SCE_CTRL_R1`, `SCE_CTRL_L2`, `SCE_CTRL_R2`, `SCE_CTRL_L3`, and `SCE_CTRL_R3` will return to the *`buttons`* member of the obtained `SceCtrlData2` structure.

On the other hand, do not implement with the assumption that `SCE_CTRL_L` and `SCE_CTRL_R` will return.

Differences in Behavior of Functions for the PlayStation®Vita System and the Wireless Controller

Function behavior differs when using buttons of the PlayStation®Vita system and of the wireless controller.

PlayStation®Vita System

The behavior of each function when using buttons on the PlayStation®Vita system are as follows.

API	*port*	Description

`sceCtrlReadBufferPositive()`
`sceCtrlReadBufferNegative()`
`sceCtrlPeekBufferPositive()`
`sceCtrlPeekBufferNegative()`
	Only 0 is valid	Values of the L button and R button can be obtained as they are

`sceCtrlReadBufferPositive2()`
`sceCtrlReadBufferNegative2()`
`sceCtrlPeekBufferPositive2()`
`sceCtrlPeekBufferNegative2()`
	Only 0 is valid	The input value of the L button is converted to and obtained as a value of the L1 button.
		The input value of the R button is converted to and obtained as a value of the R1 button.

*The `SCE_CTRL_ERROR_INVALID_ARG` error code will return if 1 or 2 is specified to *port*.

Wireless Controller

The behavior of each function when using the wireless controller is as follows

API	*port*	Description

`sceCtrlReadBufferPositive()`
`sceCtrlReadBufferNegative()`
`sceCtrlPeekBufferPositive()`
`sceCtrlPeekBufferNegative()`
	0, 1, 2, 3 or 4 can be specified	The input value of the L1 button is converted to and obtained as a value of the L button.
		The input value of the R1 button is converted to and obtained as a value of the R button.
		Values of the L2 button, R2 button, L3 button, and R3 button cannot be obtained.

`sceCtrlReadBufferPositive2()`
`sceCtrlReadBufferNegative2()`
`sceCtrlPeekBufferPositive2()`
`sceCtrlPeekBufferNegative2()`
	0, 1, 2, 3 or 4 can be specified	Values of the L1, R1, L2, R2, L3, and R3 buttons can be obtained as they are

Regarding PlayStation®TV, the type of data that can be obtained differs by the value specified to the *port* argument of each of the above functions.

port	Description
0	Obtain merged data of the Controller Number 1 wireless controller, BD remote control, and a TV remote control from HDMI CEC remote passthrough
1	Obtain data of Controller Number 1 wireless controller
2	Obtain data of Controller Number 2 wireless controller
3	Obtain data of Controller Number 3 wireless controller
4	Obtain data of Controller Number 4 wireless controller

It is recommended that 1 be specified to *port* when obtaining information of a single wireless controller.

Using the Appropriate Data Obtaining Function

As mentioned earlier, the behaviors of `sceCtrlReadBufferPositive()` and `sceCtrlReadBufferPositive2()`, `sceCtrlReadBufferNegative()` and `sceCtrlReadBufferNegative2()`, `sceCtrlPeekBufferPositive()` and `sceCtrlPeekBufferPositive2()`, as well as `sceCtrlPeekBufferNegative()` and `sceCtrlPeekBufferNegative2()` differ; use the function appropriate to your purpose.

Moreover, note the difference in data contained in the *`buttons`* member of the obtained `SceCtrlData`/`SceCtrlData2` structure (`SCE_CTRL_L`, `SCE_CTRL_R` and `SCE_CTRL_L1`; `SCE_CTRL_R1`, `SCE_CTRL_L2`, `SCE_CTRL_R2`, `SCE_CTRL_L3`, and `SCE_CTRL_R3` sections).

Example

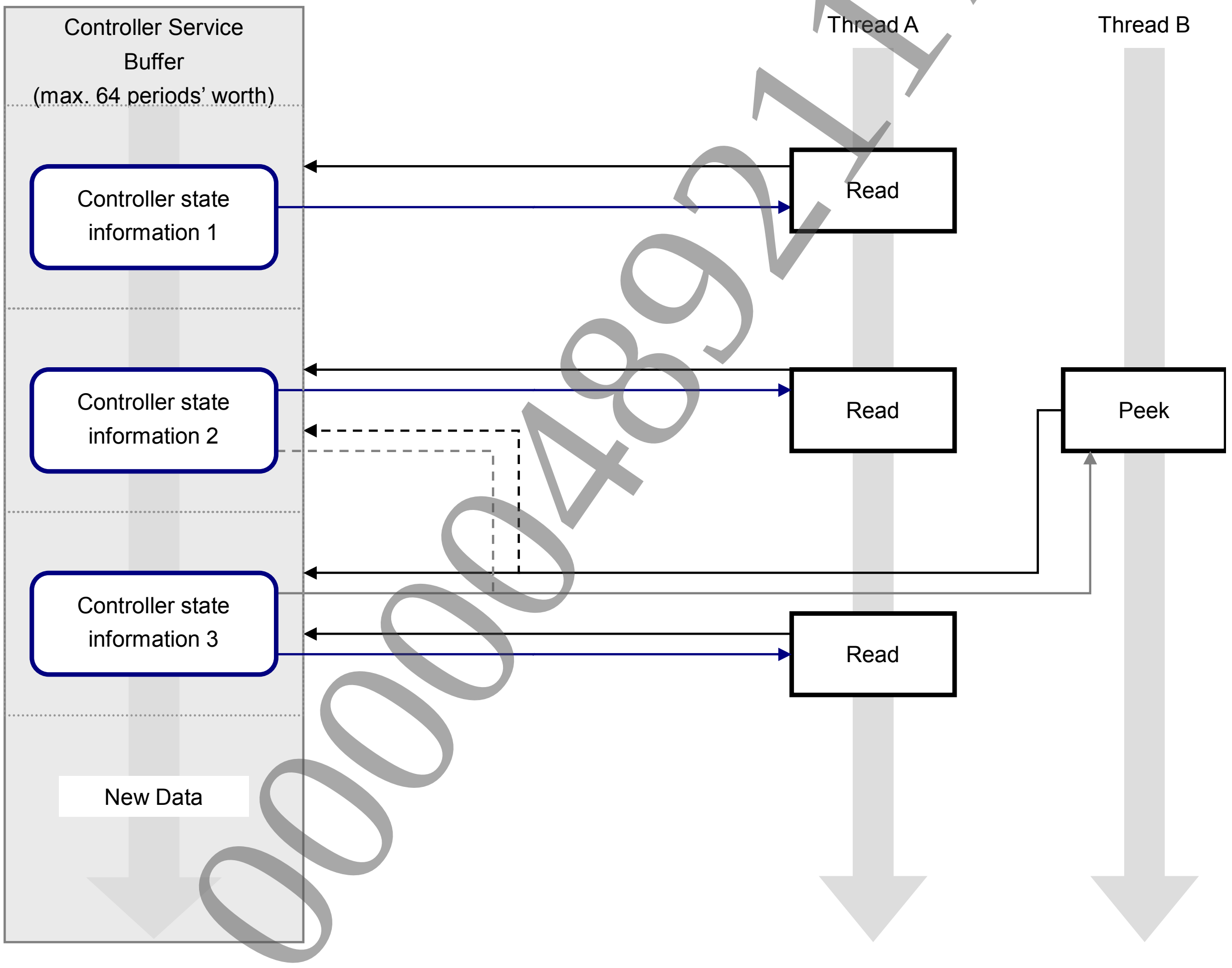
- When using buttons on the PlayStation®Vita system
  Use `sceCtrlReadBufferPositive()` with 0 specified to *`port`*.
- When using one wireless controller
  Use `sceCtrlReadBufferPositive2()` with 1 specified to *`port`*.
- When using multiple wireless controllers
  Controller Number 1: use `sceCtrlReadBufferPositive2()` with 1 specified to *`port`*.
  Controller Number 2: use `sceCtrlReadBufferPositive2()` with 2 specified to *`port`*.
  Controller Number 3: use `sceCtrlReadBufferPositive2()` with 3 specified to *`port`*.
  Controller Number 4: use `sceCtrlReadBufferPositive2()` with 4 specified to *`port`*.

3 Precautions

Buffer Polling

When the `sceCtrlReadBufferPositive()` function is called, the controller state information which had not yet been read since the last time the function was called are read from the buffer in the Controller Service. In addition, since this buffer is a process global resource, inconsistencies will occur if the `sceCtrlReadBufferPositive()` function is called from multiple threads within the same process at the same time.

Call the `sceCtrlReadBufferPositive()` function from only one thread, and if the controller state information is needed by another thread, use the `sceCtrlPeekBufferPositive()` function. By using the `sceCtrlPeekBufferPositive()` function, it is possible to snoop the data without initializing the buffer.


Controller Service
Buffer
(max. 64 periods' worth)
Controller state information 1
Controller state information 2
Controller state information 3
New Data
Thread A
Thread B
Read
Read
Peek
Read

However, note that since the `sceCtrlPeekBufferPositive()` function references the contents of the buffer at the moment that the function is called, due to variations in interrupt load it may not be possible to obtain data at a consistent interval even if the controller is being sampled every VBLANK period. This means that, in the diagram above, it is unclear whether the peek by thread B will obtain controller state information 2 or controller state information 3.

Note
`sceCtrlReadBufferPositive()`/`sceCtrlPeekBufferPositive()` obtain data in positive logic, where the bits corresponding to pressed buttons in the button state information will be 1. To obtain data in negative logic where the bits corresponding to pressed buttons will be 0, use `sceCtrlReadBufferNegative()`/`sceCtrlPeekBufferNegative()` instead of `sceCtrlReadBufferPositive()`/`sceCtrlPeekBufferPositive()`.

Operation Modes of Analog Sticks

Two modes, normal and wide, can be specified as the modes for operating the analog sticks.

The relationship between the two modes is defined as follows.

Normal Mode

This is the compatibility mode with the existing PSP™ (PlayStation®Portable), and in this mode, the range of motion is adjusted so that the maximum and minimum values of the X and Y axes correspond to a motion range that is approximately 66% of the motion range of analog sticks.

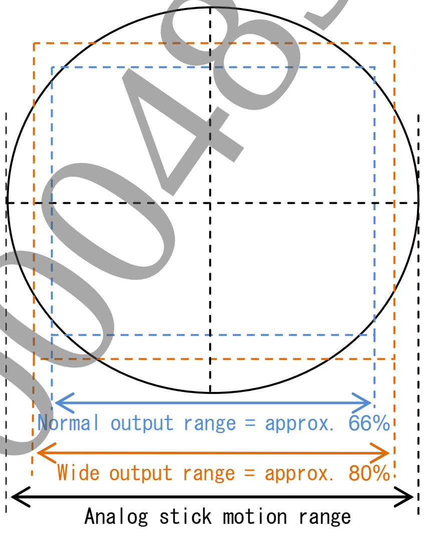
Wide Mode

In this mode, the range of motion is adjusted so that the maximum and minimum values of the X and Y axes correspond to a motion range that is approximately 80% that of the motion range of analog sticks.

Use `sceCtrlSetSamplingMode()` to set the operation mode.

To set the normal mode, specify `SCE_CTRL_MODE_DIGITALANALOG` to the *`uiMode`* argument.

To set the wide mode, specify `SCE_CTRL_MODE_DIGITALANALOG_WIDE` to the *`uiMode`* argument.


Normal output range = approx. 66%
Wide output range = approx. 80%
Analog stick motion range

Range of Values for the Analog Stick

The analog stick values can range from 0x00 to 0xFF on both the horizontal and vertical axes. In the horizontal direction, 0x00 is output when at the left limit, and 0xFF is output when at the right limit; in the vertical direction, 0x00 is output when at the top limit, and 0xFF is output when at the bottom limit. In both directions, the center value is 0x80. However, there can be a margin of error of as much as about ±0x20, so in order to detect the neutral state it is necessary to set up a dead zone of ±0x20 (i.e., to ignore values from 0x60 to 0xA0).

Two different modes, the normal mode and the wide mode, can be specified as the modes for operating the analog sticks, and in both of these modes, when the user moves along the entire range of motion in either the horizontal or vertical direction, full-scale values of 0x00 to 0xFF are guaranteed to be output. However, the ability for both the horizontal and vertical directions to perform full-scale output at the same time is not guaranteed. This means that, for example, when the user moves to the extreme lower right corner of the analog stick, there is no guarantee that 0xFF will be outputted as the value of both X and Y at the same time. Do not expect the analog stick to be formed such that, when viewed from above, the area of movement traces out a perfect square.

INTERCEPTED State

Input data from the controller may be intercepted when the system software or another application runs preferentially.

In this situation, the `SCE_CTRL_INTERCEPTED` bit is set in the *`buttons`* member of the `SceCtrlData` structure, and applications can detect that the input data from the controller is being intercepted by referring the `SCE_CTRL_INTERCEPTED` bit. Regarding the analog stick, however, the value continues to be returned even if `SCE_CTRL_INTERCEPTED` is set.

Touchscreen Pointer of PlayStation®TV

While using the touchscreen pointer of PlayStation®TV, state information of the wireless controller can be obtained as state information of the touch panel using the Touch Service. Note that, during this time, state information of the wireless controller cannot be obtained using the Controller Service.

Button Assignments of Accessibility

When the user changes the settings of the Settings application's **Accessibility** > **Button Assignments**, the controller data that can be obtained from the following data obtaining functions will be values pursuant to the settings of **Button Assignments**.

```
sceCtrlPeekBufferPositive(),sceCtrlPeekBufferPositive2(),
sceCtrlPeekBufferNegative(),sceCtrlPeekBufferNegative2(),
sceCtrlReadBufferPositive(),sceCtrlReadBufferPositive2(),
sceCtrlReadBufferNegative(),sceCtrlReadBufferNegative2()
```