Move.Me Network Protocol

Table of Contents

1 Move.Me Overview

The Move.Me server is a software application for the PlayStation®3 that opens an easier path to developing applications on the PlayStation®Move platform.

The PlayStation®Move is a combination of the PlayStation®3 system, the PlayStation®Eye camera, and the PlayStation®Move motion controller. The sphere at the end of the motion controller allows the camera to pinpoint your movement and position within the room.

Figure 1 PlayStation®Move Motion Controller

Move.Me hooks up your PC to the PlayStation®Move platform: with the Move.Me server you can use the PlayStation®Move motion controller as an input device to supply sensor data to your PC application.

The Move.Me server resides on a PlayStation®3, but Move.Me allows you to use the rich controls and processing power of the platform without the need for access to the PlayStation®3 SDK, an NDA, or Game developer license.

Move.Me provides a streamlined development system to support academic, student, and research use, lending itself particularly to in-house and prototype applications. We are excited about the possibilities for the PlayStation®Move platform to enrich projects as diverse as:

- Human-computer interaction
- Motor skills rehabilitation
- Research into game and user-interface design
- Research into augmented reality
- Interactive multimedia
- Non-commercial game development
- Other non-game applications

Samples and libraries

Move.Me sample programs are available in C# and C from:

http://code.google.com/p/moveme

C# Samples

The C# samples include a PSMoveSharp library that handles the network communication and can be reused by other C# applications. The C# samples also include an Augmented Reality demo, a diagnostic application, and a mouse driver allowing you to move the cursor by pointing the PlayStation®Move motion controller at the screen as a laser pointer.

C Samples

The C samples include a library that handles network communication and can be reused by other C applications. These samples work both under Windows and Linux.

Reference Materials

This documentation set includes the following Move.Me materials:

- *Move.Me Network Protocols* — This document. Describes the interface between the Move.Me system and the PC application, including the command set, the data packet contents, and other information you'll need to develop augmented reality applications.
- *Move.Me User's Guide* — Describes the user interface of the Move.Me server, used to configure the connection between the PlayStation®3 and the PC and to follow the status of the camera and motion controller.

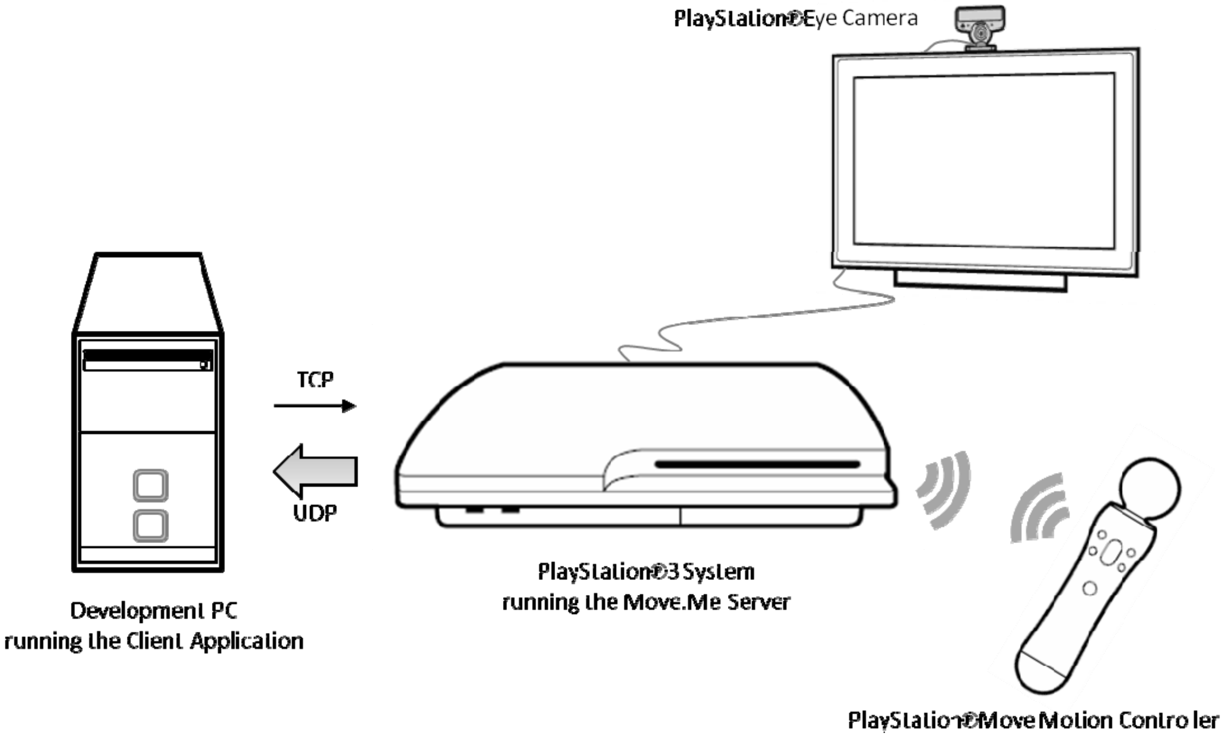
2 Network Model and Data Packets

The Move.Me server accepts connections over TCP on port 7899. The server supports up to four simultaneous connections from clients. Once a client has connected over TCP, it sends a the initialization command to the server along with a UDP port for the server to use when sending data to the client.

After the client sends the initialization command, the server sends the state of all of the motion controllers and navigation controllers at a regular interval. This interval is configurable by the client by sending the DELAY_CHANGE command.

The TCP connection between the client and server is only used when the client wants to send a command to the server. The server will never send data to the client over the TCP channel.

Figure 2 Move.Me Client/Server Environment


PlayStation®Eye Camera
TCP
UDP
Development PC
running the Client Application
PlayStation®3 System
running the Move.Me Server
PlayStation®Move Motion Controller

Data Endian Format

All data sent from Move.Me server is in big endian format and all sent to the Move.Me server must be in big endian format.

UDP Data Packets

UDP data packets sent from the Move.Me server contain a fixed-size header followed by a variable-sized payload. The header consist of the fields listed in Table 1:

Table 1 UDP Data Packet Header

Name	Byte Offset	Byte Size	Meaning
Magic	0	4	A magic number: 0xff0000dd
Server version	4	4	The server version: 0x1
Payload code	8	4	A code indicating the contents of the payload

Name	Byte Offset	Byte Size	Meaning
Payload length	12	4	The length of the payload in bytes
Packet index	16	4	An ever increasing packet index number

The server can send three different payload types:

- Standard state packet, which has the payload code of 0x1
- Camera frame slice, which has the payload code of 0x2.
- Camera frame state, which has a payload code of 0x3.

The packet index can be used to connect payloads that are received across different packets. For example, a camera frame slice payload and a camera frame state payload are from the same point in time if their packet indexes are the same. This is important when it comes to augmented reality applications where you must be using the inertial state from the time that the image was taken.

See “Packet Layouts” for details.

Camera Frame Slice Packets

The image from the camera is sent from the Move.Me server in horizontal “slices”. The number of slices is configurable using the SET_SLICES command. The number of slices is a global configuration item that affects all attached clients. Each camera frame slice packet has a 60 KB buffer used to store image data for that slice. The image data is sent in JPG format. Each slice packet contains all of the JPG data for that slice.

The resolution of the camera is 640 x 480 pixels per inch. If, for example, the server was configured to slice the camera frame into four slices, the first slice would contain the first 120 rows of the image, the second slice would contain rows 120 to 240, and so on.

Each camera frame packet includes the slice number and the number of slices that frame is being sent over in (for example, slice 2 of 4). This allows the client application to be able to determine where to draw each slice.

Image slices from the same camera frame will all share the same packet index.

To have a complete camera frame, the client must use a buffering mechanism. Assuming a 640 x 480 bitmap image containing the completed image, the client would also have temporary buffers for each of the image slices. After collecting all of the image slices for a given camera frame, these are atomically copied over the 640 x 480 completed image.

Camera Frame State Packet

Augmented reality applications display the video feed of the player and render something into the players hand. To do this properly, the client program must have a complete camera frame and a motion controller state packet (inertial sensors, position, and so on) that was computed at the time the camera captured the frame. If you have the state of the controller slightly before or after the frame of video was captured you won’t be able to seamlessly display, for example, a sword in the player’s hand.

The Move.Me server provides two kinds of state packets:

- Standard state packet, payload code 0x1, which includes the most recent data from the motion controller. In most cases, this state will be ahead of (more recent than) the video.
- Camera frame state packet, payload code 0x3, which is from the exact moment in time that the camera took the picture that the application will display.

Each state packet contains the same controller data, including the position, orientation, and acceleration. However, the standard state packet is sampled at 180 Hz while the camera frame state packet is sampled at 60 Hz (same as video).

3 Client Commands

The client application can send many different commands to the server. For example, the client can ask the server to pause sending data packets and then later send the command to resume sending the data packets. Each client command contains a fixed size header identifying the client command followed by a variable sized payload associated with the command. The header consist of the fields listed in Table 2:

Table 2 Client Command Header

Name	Byte Offset	Byte Size	Meaning
Client request	0	4	A magic number identifying the request
Payload length	4	4	The length of the payload in bytes

There is no reply from the server after a client request. For most requests, the application can determine if the request was successful by looking at the next state packet. During development, you can check for success by looking at the on-screen server log available in the Move.Me user interface. For example, if the client requests a particular PlayStation®3 Move motion controller be calibrated, the on-screen log will show that the request was made and the result of it. In addition, the next state packet received from the server will have the new status code for that motion controller.

Client Command Payload Types

There are five different client command payload structures. The following define the data types and layout.

Table 3 Client Command Payload Types

Integer client command payload (a)

Name	Byte Offset	Byte Size	Type
payload	0	4	Unsigned 32-bit integer

Rumble client command payload (b)

Name	Byte Offset	Byte Size	Type
gem_num	0	4	Unsigned 32-bit integer
Rumble	4	4	Unsigned 32-bit integer

Force RGB client command payload (c)

Name	Byte Offset	Byte Size	Type
gem_num	0	4	Unsigned 32-bit integer
r	4	4	IEEE-754 single precision float
g	8	4	IEEE-754 single precision float
b	12	4	IEEE-754 single precision float

Set tracking hues client command payload (d)

Name	Byte Offset	Byte Size	Type
gem0_hue	0	4	Unsigned 32-bit integer
gem1_hue	4	4	Unsigned 32-bit integer
gem2_hue	8	4	Unsigned 32-bit integer
gem3_hue	12	4	Unsigned 32-bit integer

Prepare camera command payload (e)

Name	Byte Offset	Byte Size	Type
max_exposure	0	4	Unsigned 32-bit integer
image_quality	4	4	IEEE-754 single precision float

Client Commands

This section defines each client command, including the command description, the client request code, the payload type, and a description of what should be stored in the payload section. The command names are for reading convenience only: the client request code is used to call the command. The payload types are described in "Client Command Payload Types".

Table 4 summarizes the client commands; the rest of the chapter describes the commands in more detail.

Table 4 Client Command Summary

Command Short Name	Client Request Code	Payload Type	Description
INIT	0x0	(a)	Initialize UDP data communications
PAUSE STATE	0x1	n/a	Pause standard state packet communications
RESUME STATE	0x2	n/a	Resume standard state packet communications
SET STATE DELAY	0x3	(a)	Change the delay between standard state packets
CONFIG CAMERA	0x4	(e)	Configure the PS Eye camera
CALIBRATE	0x5	(a)	Calibrate a PlayStation®Move motion controller
LASER LEFT PLANE	0x7	(a)	Set laser pointer left plane position
LASER RIGHT PLANE	0x8	(a)	Set laser pointer right plane position
LASER BOTTOM PLANE	0x9	(a)	Set laser pointer bottom plane position
LASER TOP PLANE	0x10	(a)	Set laser pointer top plane position
TURNON LASER	0x11	(a)	Enable laser pointer tracking
TURNOFF LASER	0x12	(a)	Disable laser pointer tracking
RESET	0x13	(a)	Reset a PlayStation®Move motion controller
POSITION LEFT PLANE	0x14	(a)	Set position pointer left plane position
POSITION RIGHT PLANE	0x15	(a)	Set position pointer right plane position
POSITION BOTTOM PLANE	0x16	(a)	Set position pointer bottom plane position
POSITION TOP PLANE	0x17	(a)	Set position pointer top plane position
TURNON POSITION	0x18	(a)	Enable position pointer tracking
TURNOFF POSITION	0x19	(a)	Disable position pointer tracking
SET SPHERE COLOR	0x20	(c)	Force the sphere to an explicit R,G,B color
SET RUMBLE	0x21	(b)	Adjust the rumble of a PlayStation®Move motion controller
SET SPHERE TRACK COLOR	0x22	(d)	Change the tracking hues of all PlayStation®Move motion controllers
SET FRAME DELAY	0x23	(a)	Change the delay between camera frame packets
SET SLICES	0x24	(a)	Configure the number of horizontal slices each camera frame is sent in
PAUSE FRAME	0x25	n/a	Pause camera frame packet communications
RESUME FRAME	0x26	n/a	Resume camera frame packet communications

INIT

Initialize UDP data communications.

Command Code

`0x0`

Payload Type/Content

Integer client command payload (a)

The payload contents is the UDP port that the client wants the server to send to.

PAUSE STATE

Pause standard state packet communications.

Command Code

`0x1`

Payload Type/Content

There is no payload.

RESUME STATE

Resume standard state packet communications.

Command Code

`0x2`

Payload Type/Content

There is no payload.

SET STATE DELAY

Change the delay between standard state packets.

Command Code

`0x3`

Payload Type/Content

Integer client command payload (a)

The delay is the number of milliseconds between state packets.

CONFIG CAMERA

Configure the PlayStation®Eye camera.

Command Code

`0x4`

Payload Type/Content

Prepare camera command payload (e)

The payload consists of:

max_exposure	The number of image rows of exposure time. The range is from 40 to 511. The longer the exposure time means decreased image noise but increased motion blur, which has a negative effect on sphere tracking.
image_quality	An image quality control knob ranging from 0.0 to 1.0.

CALIBRATE

Calibrate a PlayStation®Move motion controller.

Command Code

`0x5`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). Initiates calibration of specified motion controller, controller should be pointed at camera for best performance.

LASER LEFT PLANE

Set laser pointer left plane position.

Command Code

`0x7`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). The motion controller should be pointed at the left side of the display. Specifies the left side of the laser pointer box.

LASER RIGHT PLANE

Set laser pointer right plane position.

Command Code

`0x8`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). The motion controller should be pointed at the right side of the display. Specifies the right side of the laser pointer box.

LASER BOTTOM PLANE

Set laser pointer bottom plane position.

Command Code

`0x9`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). The motion controller should be pointed at the bottom side of the display. Specifies the bottom side of the laser pointer box.

LASER TOP PLANE

Set laser pointer top plane position.

Command Code

`0x10`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). The motion controller should be pointed at the top side of the display. Specifies the top side of the laser pointer box.

TURNON LASER

Enable laser pointer tracking.

Command Code

`0x11`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). Enables laser pointer coordinate tracking for specified Move.

TURNOFF LASER

Disable laser pointer tracking.

Command Code

`0x12`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). Disables laser pointer coordinate tracking for specified Move.

RESET

Reset a PlayStation®Move motion controller.

Command Code

`0x13`

Payload Type/Content

Integer client command payload (a)

The payload is the PlayStation®Move motion controller number (0-3). Resets the given motion controller.

POSITION LEFT PLANE

Set position pointer left plane position.

Command Code

`0x14`

Payload Type/Content

Integer client command payload (a)

See laser pointer client commands. This pointer only uses the PlayStation®Move motion controller’s position to calculate pointer coordinates. The best real world analogy for a position pointer is a paint brush.

POSITION RIGHT PLANE

Set position pointer right plane position.

Command Code

`0x15`

Payload Type/Content

Integer client command payload (a)

See laser pointer client commands. This pointer only uses the PlayStation®Move motion controller's position to calculate pointer coordinates. The best analogy for a position pointer is a paint brush.

POSITION BOTTOM PLANE

Set position pointer bottom plane position.

Command Code

`0x16`

Payload Type/Content

Integer client command payload (a)

See laser pointer client commands. This pointer only uses the PlayStation®Move motion controller's position to calculate pointer coordinates. The best analogy for a position pointer is a paint brush.

POSITION TOP PLANE

Set position pointer top plane position.

Command Code

`0x17`

Payload Type/Content

Integer client command payload (a)

See laser pointer client commands. This pointer only uses the PlayStation®Move motion controller's position to calculate pointer coordinates. The world analogy for a position pointer is a paint brush.

TURNON POSITION

Enable position pointer tracking.

Command Code

`0x18`

Payload Type/Content

Integer client command payload (a)

See laser pointer client commands. This pointer only uses the PlayStation®Move motion controller's position to calculate pointer coordinates. The best analogy for a position pointer is a paint brush.

TURNOFF POSITION

Disable position pointer tracking.

Command Code

`0x19`

Payload Type/Content

Integer client command payload (a)

See laser pointer client commands. This pointer only uses the PlayStation®Move motion controller's position to calculate pointer coordinates. The best analogy for a position pointer is a paint brush.

SET SPHERE COLOR

Force the sphere to an explicit R,G,B color.

Command Code

`0x20`

Payload Type/Content

Force RGB client command payload (c)

The payload is the PlayStation®Move motion controller number (0-3). Followed by the red, green and blue color specified from 0.0-1.0 floating point. When a sphere has been forced to a specific RGB color, sphere tracking is disabled. This has the largest impact on position tracking.

SET RUMBLE

Adjust the rumble of a PlayStation®Move motion controller.

Command Code

`0x21`

Payload Type/Content

Rumble client command payload (b)

The payload is the PlayStation®Move motion controller number (0-3). Followed by the rumble value from 0 (off) to 255 (full on). When the rumble is on, the inertial sensors inside the motion controller are affected and lose some precision.

SET SPHERE TRACK COLOR

Change the tracking hues of all PlayStation®Move motion controllers.

Command Code

`0x22`

Payload Type/Content

Set tracking hues client command payload (d)

The payload is the tracking hue for each of the PlayStation®Move motion controllers. All of the sphere tracking hues must set together. Each hue ranges from 0-359. The hues are only requests, Move.Me may move the hues apart from each other so that they can all be tracked. If the hue doesn't matter, it can be specified as '`4<<24`'. If a sphere shouldn't be tracked it can be specified as '`8<<24`'.

SET FRAME DELAY

Change the delay between camera frame packets.

Command Code

`0x23`

Payload Type/Content

Integer client command payload (a)

The payload is the number of milliseconds between each camera frame packet. This ranges from 16-255 ms.

SET SLICES

Configure the number of horizontal slices in which each camera frame is sent.

Command Code

```
0x24
```

Payload Type/Content

Integer client command payload (a)

The payload is the number of horizontal slices each camera frame is sent in. Each slice is sent in a separate packet. This ranges from one to seven slices. Typically, no more than two slices are needed.

PAUSE FRAME

Pause camera frame packet communications.

Command Code

```
0x25
```

Payload Type/Content

No payload.

RESUME FRAME

Resume camera frame packet communications.

Command Code

```
0x26
```

Payload Type/Content

No payload.

4 Packet Layouts

The following tables describe the packet layouts corresponding to the UDP data packets sent from the Move.Me server to the client in response to specific client commands.

Table 5 Constants

Constant	Value
PSMOVE_PACKET_MAGIC	0xff0000dd
PSMOVE_PACKET_CODE_STANDARD_STATE	0x1
PSMOVE_PACKET_CODE_CAMERA_FRAME_SLICE	0x2
PSMOVE_PACKET_CODE_CAMERA_FRAME_STATE	0x3
PSMOVE_SERVER_MAX_CONS	4
PSMOVE_SERVER_MAX_NAVS	7
IMAGE_BUFFER_SIZE	61440
CAMERA_FRAME_SPLIT_FORMAT_JPG	0x1
MAXIMUM_CAMERA_FRAME_SLICES	7
CELL_PAD_MAX_CODES	64

Table 6 Type Definitions

TypeDef	Value
unsigned char	u8_t
unsigned short	u16_t
unsigned int	u32_t
int	i32_t
unsigned long long	u64_t
float	float4[4]

Table 7 Packet Payloads

Name	Type	Offset	Size	Contents
PSMoveServerPacketHeader	struct		20	Header describing the packet
magic	u32_t	0	4	0xff0000dd
move_me_server_version	u32_t	4	4	1
payload_code	u32_t	8	4	Code describing payload
payload_length	u32_t	12	4	Number of bytes in payload (these bytes are immediately after the header)
packet_index	u32_t	16	4	An ever increasing counter over time
PSMoveConnectionConfig	struct		12	Client configuration state
ms_delay_between_standard_packets	u32_t	0	4	Number of milliseconds between state packets
ms_delay_between_camera_frame_packets	u32_t	4	4	Number of milliseconds between camera frame slice packets
camera_frame_packet_paused	u32_t	8	4	Camera frame paused
PSNavPadInfo	struct		28	Connection state of all navigation controllers
port_status[PSMOVE_SERVER_MAX_NAVS]	u32_t	0	28	Bit0 is whether or not controller connected. Bit1 if something changed
PSNavPadData	struct		132	Button data for each connected navigation controller
len	i32_t	0	4	Length of button data
button[CELL_PAD_MAX_CODES]	u16_t	4	128	Button bitmask
PSMovePadData	struct		4	
digitalbuttons	u16_t	0	2	Digital button on motion controller bitmask
analog_T	u16_t	2	2	Analog T value

Name	Type	Offset	Size	Contents
PSMoveState	`struct`		176	State of a motion controller
pos	`float4`	0	16	Position of sphere
vel	`float4`	16	16	Velocity of Sphere
accel	`float4`	32	16	Acceleration of sphere
quat	`float4`	48	16	Orientation of motion controller (quaternion)
angvel	`float4`	64	16	Angular velocity
angaccel	`float4`	80	16	Angular acceleration
handle_pos	`float4`	96	16	Handle position
handle_vel	`float4`	112	16	Handle velocity
handle_accel	`float4`	128	16	Handle acceleration
pad	`PSMovePadData`	144	4	Digital buttons and analog T button data
timestamp	`i64_t`	152	8	Timestamp
temperature	`float`	160	4	Temperature of controller
camera_pitch_angle	`float`	164	4	Current camera pitch angle
tracking_flags	`u32_t`	168	4	Tracking flags
PSMoveImageState	`struct`		48	
frame_timestamp	`i64_t`	0	8	Frame time stamp
timestamp	`i64_t`	8	8	Timestamp for when the sphere was actually imaged; includes exposure time adjustments
u	`float`	16	4	Horizontal pixel center of sphere
v	`float`	20	4	Vertical pixel center of sphere
r	`float`	24	4	Radius of sphere in pixels
projectionx	`float`	28	4	Normalized horizontal projection of sphere position
projectiony	`float`	32	4	Normalized vertical projection of sphere position
distance	`float`	36	4	Distance from camera origin to the sphere
visible	`u8_t`	37	1	Whether sphere was visible on the camera
r_valid	`u8_t`	38	1	Whether or not r was calculated this frame. If 0, r and distance are old.
PSMoveCameraState	`struct`		20	
exposure	`int`	0	4	Camera exposure setting (in image rows)
exposure_time	`float`	4	4	Camera exposure setting (in seconds)
gain	`float`	8	4	Gain (1.0 to 4.0)
pitch_angle	`float`	12	4	Camera pitch angle used for state computation
pitch_angle_estimate	`float`	16	4	Current camera pitch angle estimate
PSMoveSphereState	`struct`		20	
tracking	`u32_t`	0	4	Is tracking enabled?
tracking_hue	`u32_t`	4	4	Tracking hue
r	`float`	8	4	Red component of sphere LED
g	`float`	12	4	Green component of sphere LED
b	`float`	16	4	Blue component of sphere LED

Name	Type	Offset	Size	Contents
PSMovePointerState	`struct`		12	
valid	`u32_t`	0	4	Whether or not normalized_x, y is valid
normalized_x	`float`	4	4	Normalized laser pointer coordinates in X
normalized_y	`float`	8	4	Normalized laser pointer coordinates in Y
PSMovePositionPointerState	`struct`		12	
valid	`u32_t`	0	4	Whether or not normalized_x, y is valid
normalized_x	`float`	4	4	Normalized position pointer coordinates in X
normalized_y	`float`	8	4	Normalized position pointer coordinates in Y
PSMoveStatus	`struct`		16	
connected	`u32_t`	0	4	Is motion controller connected?
code	`u32_t`	4	4	
flags	`u64_t`	8	8	Motion controller status flags
PSMoveServerPacket	`struct`			Entire packet for payload_code == PSMOVE_PACKET_CODE_STANDARD_STATE and payload_code == PSMOVE_PACKET_CODE_-CAMERA_FRAME_STATE
header	`PSMoveServerPacketHeader`	0	20	
server_config	`PSMoveServerConfig`	20	8	
client_config	`PSMoveConnectionConfig`	28	12	
status[PSMOVE_SERVER_MAX_CONS]	`PSMoveStatus`	40	64	
state[PSMOVE_SERVER_MAX_CONS]	`PSMoveState`	104	704	
image_state[PSMOVE_SERVER_MAX_CONS]	`PSMoveImageState`	808	192	
pointer_state[PSMOVE_SERVER_MAX_CONS]	`PSMovePointerState`	1000	48	
pad_info	`PSNavPadInfo`	1048	28	
pad_data[PSMOVE_SERVER_MAX_NAVS]	`PSNavPadData`	1076	924	
sphere_state[PSMOVE_SERVER_MAX_CONS]	`PSMoveSphereState`	2000	80	
camera_state	`PSMoveCameraState`	2080	20	
position_pointer_state[PSMOVE_SERVER_MAX_CONS]	`PSMovePositionPointerState`	2100	48	