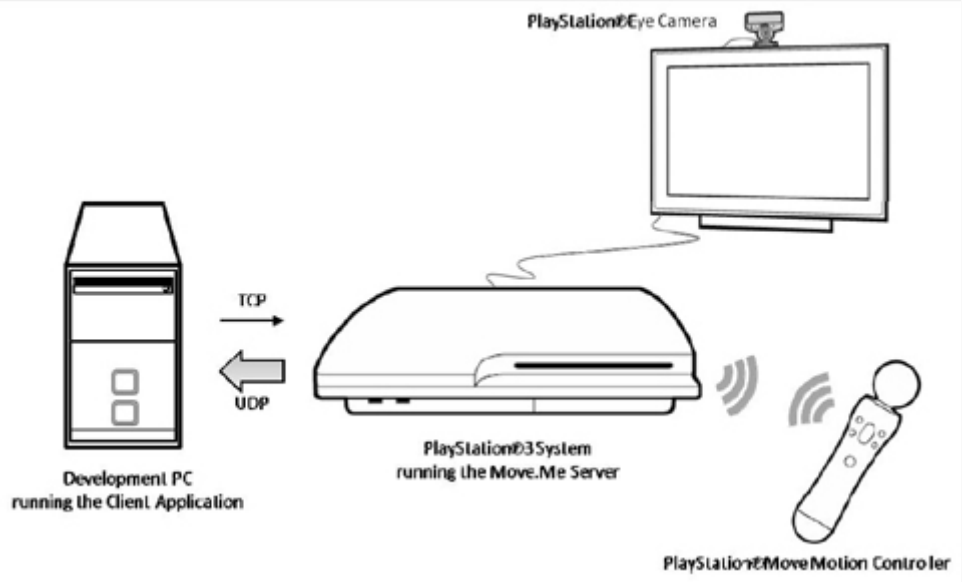
PlayStation Move Wrapper for Move.me

1. Installing the Move.me SDK

In order to use PS Move for development, you need to install Move.me SDK on PlayStation3. Move.me SDK is supported by Sony, and can be accessed by PlayStation Network.
Follow this link. to learn more about Move.me SDK.



the network protocol of Move.me

Launch Move.me Server

NOTE: Before running your PS Move game, you MUST run Move.me on PS3.

1. Connect PlayStation Eye camera and PS Move to PS3 and make sure PS3 is connected to internet.
2. Launch Move.me under the **GAME** section.
3. Once the instruction of Move.me finishes, press any button on Move controller to enter.
4. If you see IP address and port on the upper left corner, Move.me is ready.
5. Do **NOT** press PS button on the Move controller when you are running your game, Move.me will not work in the menu mode.

Calibration, Track and Reset

- **Calibration and Track** - Press MOVE button, and the ball will flash and then glow. Stand still until the ball glows, otherwise calibration will fail. After calibration, you can see a sword in Move.me SDK for the first Move. For other Moves, you can see virtual Move controllers.

- **Reset** - Press SELLECT button to reset.

Integrating with Unity

All you need to do is to import the PSMove package, and run the **PSMoveBasicScene**. Type in the IP address and port number given Move.me, and hit connect, you can see data coming from PS3 will show on your screen.

IMPORTANT:

- **Turn OFF the Windows Firewall if your build version cannot get data from PS3.**
- **Please make sure your Move controller is charged.**
- **For new Move controller, please connect to PS3 by USB cable and press PS button to pair.**

2. Bringing the Move.me SDK in to Unity

Install the Unity Package

The pre-built Unity package includes all of the scripts required to start running your game. To install, simply import the package, and you all of the required assets will appear in your project.

The following items are included within the package:

Scenes

- **PSMoveBasicScene** - This scene shows you what data you can get from and send to PS Move, and how to build an avatar for the Move controller in your world. Use this to get a feel for what the PS Move is capable of. It also shows you how to prepare your GameObjects

Scripts

- **PSMoveExample** - This is the script that shows the basic use of PSMoveInput and PSMoveForUnity in the PSMoveBasicScene.

- **PSEyeImageDisplay** - This is the script that shows how to display camera image.

- **PSMoveWrapper/PSMoveInput** - This is the script you need to access to for all interactions with PS Move: connect/disconnect, access data and send command. You may use it anywhere in your script just as using *Input* in Unity.

- **PSMoveWrapper/MoveController**- This is the script defined all data and method related to Move controller.

- **PSMoveWrapper/NavController**- This is the script defined all data and method related to Navigation controller.

- **PSMoveWrapper/PSMoveForUnity** - This is the script that pulls out most of the features from the Move.me and brings them for use within Unity. It also contains most of the update

logic required for using the PS Move. You need to attach this script to a GameObject, and it will persist between scenes because of the DontDestroyOnLoad call in this script.

- **PSMoveWrapperUtil** - The scripts in this folder are for internal use, including network and other utility functions.

3. Using the Scripts

After importing the asset package, you have the PSMoveBasicScene to test and get familiar with the PSMove. You can use this as your base scene, or start a new scene and drag the **PSController** prefab into the scene.

PSMoveInput

The Move.me SDK currently can support up to 4 Move controllers and up to 7 Navigation controllers. Thus **MoveControllers** field has 4 elements, representing Move controller No.0 - No.3, and**NavControllers** field has 7 elements, representing Navigation controller No.0 - No.6.

MoveController[] MoveControllers

An array of data of move controllers, see **MoveController** for details.

NavController[] NavControllers

An array of data of navigation controllers, see **NavController** for details

int moveCount

how many Move controllers are connected and calibrated.

int NavCount

how many Navigation controllers are connected.

bool IsConnected

whether the PSMoveWrapper is connected to the Move.me server on PS3.

bool IsCameraResume

whether to receive the camera image stream.

PSMoveSharp.PSMoveSharpState State

the complete state information of Move.me Server and all Move controllers. Use it when you need additional information which is not provided by PSMoveInput.

PSMoveSharp.PSMoveSharpCameraFrameState CameraFrameState

the complete state information of PS Eye camera. Use it when you need additional information which is not provided by PSMoveInput.

bool onlineMode

whether the wrapper should connect to PS3. If it is false, the **Connect** function will **NOT** connect to PS3 even it is called. This prevents possible interference with

other games testing on Move.me. It is also suggested to use this value to determine whether to activate your hot keys/simulator.

IMPORTANT: The default value of **onlineMode** is true.

void Connect(string address, int port)

connect to Move.me server.

void Disconnect()

the same as "Disconnect(true)". Pause camera stream, reset all move controllers and disconnect to Move.me server.

void Disconnect(bool isCleanUp)

If you just want to disconnect to Move.me server, call "Disconnect(false)". **NOT RECOMMENDED**.

isCleanUp whether to clean up stuff before disconnection.

void CameraFrameResume()

the same as "CameraFrameResume(8)".

void CameraFrameResume(int sliceNum)

Resume camera stream and set slice num. Camera stream is initially paused, you need to call CameraFrameResume() to get camera image.

sliceNum the number of slices for sending camera image, range from 1 to 8.

void SetCameraFrameSlices(int sliceNum)

Set slice num.

sliceNum the number of slices for sending camera image, range from 1 to 8.

void CameraFramePause()

Pause camera image stream.

Color32[] GetCameraImage()

The image should be 640x480, please check the length before using it. Also I know it is a weird issue, but after Move.me started, please calibrate at least once to make sure the image stream working smoothly. Once you have done this, it will work fine.

Returns the complete camera image, null if not connected or camera stream is paused.

void TrackAll()

Set all Move controllers with default color and track with corresponding hue. Use this method when you do not care about the color of all Move controllers.

void ResetAll()

Reset the Move controllers.

int[] GetTrackingHues()

Get tracking hues of all Move controllers.

MoveController

int Num

the Num of this controller, range from 0-3.

bool Connected

whether a single Move controller is connected and calibrated. **NOTE: WE SAY A MOVE CONTROLLER IS CONNECTED AFTER IT IS SUCCESSFULLY CALIBRATED.**

int RumbleLevel

the rumble scale for Move controller, range from 0 to 19. This field should read-only. If you want to set rumble level, call "SetRumble" fucntion.

MoveData Data

data of the move controller as following:

Vector3 Position

the position of the ball.

Vector3 Velocity

velocity of the ball.

Vector3 Acceleration

acceleration of the ball.

Vector3 Orientation

the orientation of the wand in Euler angles.

Vector3 QOrientation

the orientation of the wand in Quaternion.

Vector3 AngularVelocity

the angular velocity of the wand.

Vector3 AngularAcceleration

the angular acceleration of the wand.

Vector3 HandlePosition

the position of the handle.

Vector3 HandleVelocity

the velocity of the handle.

Vector3 HandleAcceleration

the acceleration of the handle.

MoveButton Buttons

the current state of buttons.

MoveButton PrevButtons

the previous state of buttons.

bool GetButtons(MoveButton requestButtons)

check if the buttons are pressed down. It supports button combination. For example,

GetButton(MoveButton.Circle | MoveButton.Cross)

checks if Circle button AND Cross button are both pressed down.

bool GetButtonsAny(MoveButton requestButtons)

check if any of the buttons are pressed down. It supports button combination. For example,

GetButtonAny(MoveButton.Circle | MoveButton.Cross)

checks if either Circle button OR Cross button is pressed down.

bool GetButtonsAny()

check if any button is pressed down. The same as *GetButtonAny(MoveButton.All)*

bool GetButtonsUp(MoveButton requestButtons)

check if the button combination are pressed this frame.

bool GetButtonsDown(MoveButton requestButtons)

check if the button combination are released this frame.

int ValueT

the scale of button TRIGGER, range from 0 to 255. 0 means fully released and 255 means fully pressed.

int thresholdT

whether the button TRIGGER can be treated as "pressed". It changes the status of button TRIGGER from a range of int to a bool. The default value is 250. This field is **modifiable**.

Color SphereColor

the light color of the ball of Move controller.

bool IsTracking

whether the PS Eye camera is tracking the ball of Move controller.

int TrackingHue

the hue that PS Eye camera is tracking for Move controller, range from 0 to 359.

Vector2 SpherePixelPosition

the pixel position of the ball of move controller on camera image.

float SpherePixelRadius

the pixel radius of the ball of move controller on camera image.

Vector2 SphereProjectionPosition

the normalized position of the ball of move controller on camera image.

float SphereDistance

the actual distance of move controller to the origin of the camera (in meters/Unity

units).

bool SphereVisible

whether the camera can see the ball of move controller.

bool SphereRadiusValid

whether sphereRadius/sphereDistance of this frame is valid.

void Calibrate()

Calibrate the Move controller. The ball will **NOT** glow after calibration. You need to call any one of the "SetColor" methods to make it glow, and the "Track" methods to track.

void AutoTrack()

Let PS3 pick color and track the selected Move controller. Use this method when you do not care about the color of the ball.

void SetColor(Color color)

The PS Eye camera will **NOT** automatically track after calling SetColor(). If you change the color of a tracking Move controller, the tracking will be lost. You need to call any one of the "Track" methods to track. The minimum step for color is 0.2f for any RGB value, the alpha value is not used.

color the color to set. Set the color of Move controller's ball.

void SetTrackingHue(int hue)

Set the tracking hue of Move controller. The hue should fit the ball's color to enable tracking.

hue the hue to track.

void SetColorAndTrack(Color color)

The combination of "SetColor" and "SetTrackingHue".

color the color to set and to track.

void CalibrateAndTrack()

the same as "CalibrateAndTrack(0.8f)"

void CalibrateAndTrack(Color color)

the same as "CalibrateAndTrack(color, 0.8f)"

void CalibrateAndTrack(float time)

The combination of "Calibrate" and "AutoTrack". Since calibration take time, tracking should be delayed a certain amount of seconds. 0.8f seems appropriate after some tests.

time delay time for tracking after calibration.

CalibrateAndTrack(Color color, float time)

The combination of "Calibrate" and "SetColorAndTrack".

color the color to set and to track.

time delay time for tracking after calibration.

void Reset()

Reset the Move controller. The ball will not glow and it need to re-calibrate.

void SetRumbleint level)

To set the rumble of Move controller. 0 is not rumble, 19 is maximum rumble.

level the scale of rumble, range from 0 - 19.

NavController

int Num

the Num of this controller, range from 0-6.

bool Connected

whether a single Navigation controller is connected.

NavData Data

data of the navigation controller as following:

Vector2 ValueAnalog

scale of navigation analog, range from -128 to 127.

NavButton Buttons

the current state of buttons.

NavButton PrevButtons

the previous state of buttons.

bool GetButtons(NavButton requestButtons)

check if the buttons are pressed down. It supports button combination. For example,

GetButton(NavButton.Circle | NavButton.Cross)

checks if Circle button AND Cross button are both pressed down.

bool GetButtonsAny(NavButton requestButtons)

check if any of the buttons are pressed down. It supports button combination. For example,

GetButtonAny(NavButton.Circle | NavButton.Cross)

checks if either Circle button OR Cross button is pressed down.

bool GetButtonsAny()

check if any button is pressed down. The same as *GetButtonAny(NavButton.All)*

bool GetButtonsUp(NavButton requestButtons)

check if the button combination are pressed this frame.

bool GetButtonsDown(NavButton requestButtons)

check if the button combination are released this frame.

int ValueL2

the scale of button TRIGGER, range from 0 to 255. 0 means fully released and 255 means fully pressed.

int thresholdL2

whether the button L2 can be treated as "pressed". It changes the status of button L2 from a range of int to a bool. The default value is 250. This field is **modifiable**.

PSMoveForUnity

bool onlineMode

This value will be set as initial value of **PSMoveInput.onlineMode**. Changing this value in runtime will NOT have effect.

bool enableDefaultInGameCalibrate

whether you can calibrate and reset by pressing MOVE button and RESET button via PSMoveWrapper.

PSMoveExample

string ipAddress

the IP address of PS3. You can see the IP address after you launch Move.me server on PS3.

string port

the port of PS3. You can see the port after you launch the Move.me server on PS3. The default value is 7899.

GameObject gem

act as the avatar for the ball of Move controller in the scene.

GameObject handle

act as the avatar for the handle of Move controller in the scene.

isMirror

whether the avatar for Move controller in the scene is its mirror image. The default value is TRUE.

zOffset

the start z position in the scene. This variable is used only when isMirror == FALSE. You can imagine that the PS Eye camera's Z postion in the scene is zOffset. The default value is 20.

PSEyeImageDisplay

This script shows you how to get camera image. To display, just drag the script to an gameObject with renderer (Cube or Plane for instance).