PS Move API Documentation

Release 4.0.12

Thomas Perl

Dec 19, 2020

Contents

CHAPTER 1

Building PS Move API from source

Note that the PS Move API Github Releases page contains prebuilt library downloads, you might prefer to use those.

1.1 Building on macOS 10.14

You need to install the following requirements:

- Homebrew

Install the dependencies:

```
bash -e -x scripts/install_dependencies.sh
```

To build from source, you can use the build script:

```
bash -e -x scripts/macos/build-macos
```

1.2 Building on Ubuntu 18.04

Install the dependencies:

```
bash -e -x scripts/install_dependencies.sh
```

To build from source, you can use the build script:

```
bash -e -x scripts/linux/build-debian
```

1.3 Building on Windows (Visual Studio)

You need to install the following requirements:

- Visual Studio Community 2017 or newer
- CMake 3.9.1 or newer
- Git

1.3.1 Automatic build

A build script is provided which will take care of the build for you. The script will generate the Visual Studio solution and build everything in Debug and Release.

Run the batch files in the "Developer Command Prompt for VS 2017" (you will find it in the start menu folder "Visual Studio 2017", for VS 2013/2015 the menu items should be named similarly) from the PS Move API checkout folder.

For Visual Studio 2017 and 64-bit use:

```
call scripts/visualc/build_msvc.bat 2017 x64
```

For Visual Studio 2017 and 32-bit use:

```
call scripts/visualc/build_msvc.bat 2017 x86
```

The resulting binaries will be placed in `build-x86/` (for the 32-bit build) and `build-x64/` (for the 64-bit build).

1.4 Cross-Compiling for Windows (MinGW)

To cross-compile for Windows on Ubuntu:

```
sudo apt-get install mingw-w64 cmake
```

To build manually without the tracker:

```
mkdir build-win32
cd build-win32
cmake \
    -DCMAKE_TOOLCHAIN_FILE=../cmake/i686-w64-mingw32.toolchain \
    -DPSMOVE_BUILD_TRACKER=OFF \
    ..

mkdir build-win64
cd build-win64
cmake \
    -DCMAKE_TOOLCHAIN_FILE=../cmake/x86_64-w64-mingw32.toolchain \
    -DPSMOVE_BUILD_TRACKER=OFF \
    ..
```

Or use the ready-made build script:

```
bash -x scripts/mingw64/cross-compile x86_64-w64-mingw32
bash -x scripts/mingw64/cross-compile i686-w64-mingw32
```

1.5 Building for the Pocket C.H.I.P

PS Move API now supports the Pocket C.H.I.P, an embedded Linux computer running a Debian-based operating system. The device has built-in Bluetooth, WIFI, a standard-sized USB port and a 3.5mm headphone jack, making it suitable for portable PS Move applications.

To build on a Pocket C.H.I.P, `ssh` into your device (or use the Terminal) and then build the release tarball:

```
bash -e -x scripts/pocketchip/install_dependencies.sh
bash -e -x scripts/pocketchip/build.sh
```

1.5.1 Installation and Configuration

In order to be able to use the PS Move Motion Controllers without `root` access, you need to install an udev rules file on your C.H.I.P:

```
sudo cp contrib/99-psmove.rules /etc/udev/rules.d/
```

Also, not all kernels ship with the required `hidraw` support, you can check if your kernel does by running the following command after bootup:

```
dmesg | grep hidraw
```

A kernel with hidraw will print something like the following:

```
[    1.265000] hidraw: raw HID events driver (C) Jiri Kosina
```

If your kernel does not have hidraw support, you should install the newest Firmware for your Pocket C.H.I.P, and make sure to install all updates via `apt`.

1.6 Python bindings

Python bindings (among others) are built using SWIG. So make sure you have that installed. CMake will let you know if SWIG could not be found in the initial configure step. Look in CMake's output in the section "Language bindings".

Also required is the Python library (`libpython-dev` on Linux). If you have multiple versions of Python installed (most likely some 2.x and 3.x) chances are CMake decides to use the wrong one. Again, look in CMake's output in the section "Language bindings" which version of the Python library CMake is using for the build. Make sure it matches the version you want to run your Python scripts with later. They must be the same!

If CMake does not choose the correct version right away, use the option `PSMOVE_PYTHON_VERSION` to set the desired one. Usually it is sufficient to set this to either 2 or 3 (for Python 2 and 3, respectively), but minor versions are also supported. So you could choose between building for Python 2.6 and 2.7. If you are running CMake from the command line set the version like so:

```
cmake .. -DPSMOVE_PYTHON_VERSION=2
```

Check CMake's output to verify that the correct version is now found; some flavor of Python 2 in this example. If CMake still uses the wrong one, try removing all the files CMake generated in the `build` directory and run again.

1.6.1 Testing the build

A lot of Python example scripts are provided in the `examples/python/` directory. They are laid out so that when you build the library (and its Python bindings) in the customary `build` folder in the PSMove API checkout, the Python examples should find the modules without needing to install anything. We suggest you start with `always.py` which you can directly call from within the `build` directory like so:

```
python ../examples/python/always.py
```

This script does not require Bluetooth and should thus provide an easy way to test the Python bindings. Simply connect your Move controller via USB and run the script as shown above. If that is working, continue with `pair.py` to set everything up for using Bluetooth.

CHAPTER 2

Pairing the Controller to your PC

The PS Move connects via two methods:

- **USB**: Used for Bluetooth pairing; can set rumble and LEDs, cannot read sensors
- **Bluetooth**: Used for wireless connectivity; can set rumble, LEDs and read sensors

2.1 Bluetooth pairing

The `psmove pair` utility is used for Bluetooth pairing – it will write the Bluetooth host address of the computer it's running on to PS Move controllers connected via USB.

It optionally takes a single command-line parameter that is an alternative Bluetooth host address. For example, if you want to pair your PS Move controller to your phone, but it does not have USB Host Mode, you can use this on your PC:

```
./psmove pair aa:bb:cc:dd:ee:ff
```

Where `aa:bb:cc:dd:ee:ff` is the Bluetooth host address of your phone. Note that depending on the phone, you also need to run pairing code there.

Depending on the operating system, you might need to run the utility as Administrator (Windows), enter your password (OS X) or run using `sudo` (Linux) to let the utility modify the system Bluetooth settings and whitelist the PS Move for connection.

Note: You only need to pair the controller to your PC once, from then on it will always try to connect to your PC. Only when you connect your controller to a PS3 or pair with another PC will you have to re-do the pairing process on your computer.

2.2 Connecting via Bluetooth

Unplug the USB cable and press the PS button on the controller. The red status LED will start blinking and should eventually remain lit to indicate a working Bluetooth connection. If it continues blinking, it might not be paired via Bluetooth, or - if you can see the connection on your computer - the battery is low and needs charging via USB or a charger. To verify the connection, check the Bluetooth devices list of your computer and see if there is an entry for "Motion Controller".

On recent versions of OS X, a dialog might pop up asking for a PIN. Close it and pair the controller again using `psmove pair`. After that, it should connect successfully.

2.3 Troubleshooting

Here are some advanced tips if you can't get pairing working out of the box:

2.3.1 Mac OS X

If `psmove pair` doesn't work or you get a PIN prompt when you press the PS button on your controller, follow these steps:

- Right after you run `psmove pair` write down the adress you find after "controller address:" in the form "aa:bb:cc:dd:ee:ff"
- Disable Bluetooth (or the modifications that follow won't work)
- Wait for the Bluetooth process to quit (`pgrep blued`); repeat until nothing prints anymore (e.g. the process "blued" has quit) - this can take up to a minute
- Authorize the controller's MAC address: `sudo defaults write /Library/Preferences/com.apple.Bluetooth HIDDevices -array-add "<aa-bb-cc-dd-ee-ff>"` (where <aa-bb-cc-dd-ee-ff> is the MAC address you wrote down at 8.2 but with hyphens for separators)
- Enable Bluetooth again then press the PS button on the controller. The PIN request should not pop up this time and the Move should now appear in the bluetooth devices as "Motion Controller".

Starting with macOS Sierra, at most 2 Move controllers can be connected via Bluetooth at the same time when using the built-in Bluetooth adapter. This is due to some internal changes Apple made and did work on the same hardware prior to updating to Sierra. If you need more controllers at the same time, you can use an external Bluetooth adapter.

2.3.2 Ubuntu Linux

If you wish to access the PSMove controller via USB or Bluetooth without requiring root-level permissions then it is necessary to copy the `contrib/99-psmove.rules` file to `/etc/udev/rules.d/`:

```
sudo cp contrib/99-psmove.rules /etc/udev/rules.d/
sudo udevadm trigger
```

2.3.3 Windows

You might have to try pairing multiple times for the Bluetooth connection to work on Windows. Also, be sure to use the Microsoft Bluetooth Stack and do not install any third party drivers (e.g. MotionInJoy) that would interfere with proper operation of PS Move API on Windows.

2.3.4 Pocket C.H.I.P

During testing, we found that pairing a PS Move via USB works fine in a Pocket C.H.I.P with its battery. On a C.H.I.P (the board, without the battery, just connected via a MicroUSB charger) we noticed that sometimes the PS Move seems to draw too much power from USB, causing the C.H.I.P to reboot. In this case, you can pair your controller with the C.H.I.P in the Pocket C.H.I.P shell and then remove the C.H.I.P and just power it up and connect via Bluetooth only.

If you do not have a Pocket C.H.I.P, and just a C.H.I.P, you can pair the controller on your workstation (Linux, macOS, Windows) and then register the controller using `psmove register`.

CHAPTER 3

Tracking and Camera

This page contains information about the camera integration, especially as it relates to tracking and the different variants for different OSes.

3.1 PSEYEDriver

This is the recommended way of interfacing with the PS Eye camera on Windows and OS X.

3.2 CL Eye Driver

For some situations, on Windows this might give better performance than the PS Eye Driver. You just install the CL Eye Driver normally and then use the camera via OpenCV.

3.3 iSight exposure locking

On Mac OS X, we have problems getting the PS Eye to work reliably with OpenCV. Because of that, it makes sense to use other cameras like the iSight camera built into most (all?) MacBook (Pro) computers and iMacs. Unfortunately, there is no API for controlling the exposure directly. It is possible to lock the exposure to a given value so that the exposure isn't continously corrected.

How it works normally (continuous autoexposure):

1. The camera device determines a good first exposure when started
2. While frames are captured, the exposure is continuously modified (e.g. when you hold the controller near the iSight, the exposure is automatically lowered to accommodate the brighter camera image)

How it works with locked exposure:

1. The camera device determines a good first exposure when started

2. The exposure set after opening the device will stay the same

So what we need to do is the following:

1. Light up the controller to a very bright color
2. Move the controller directly in front of the camera (the sphere can actually touch the iSight camera and "block its sight")
3. Open the camera device (the first exposure is determined here)
4. As soon as the camera device is opened (and the exposure set), lock the exposure so that it doesn't change anymore
5. Move the controller away from the camera, proceed with normal initialization procedure (blinking calibration, etc..)

The call to locking has been integrated into the tracker library, the exposure will be automatically locked when the tracker library is started.

The exposure seems to be locked until the next reboot in my experiments, but the exposure is determined every time the camera device is opened (but will stay locked after opening the device).

Because of this, the calibration procedure on Mac OS X is a bit different to other platforms, because we have to light up the controller, put it in front of the iSight, open the camera device, wait for a good exposure, and only then continue with normal calibration. This has been implemented in `examples/c/test_tracker.c` - the console output will tell you what to do:

1. Start the application
2. Move the controller that lights up close to the iSight
3. Press the Move button on this controller
4. Keep the controller close to the iSight while the camera is opened
5. Move the controller away from the iSight and press the Move button

Thanks to Raphaël de Courville who had the initial idea for the exposure locking, did some experimentation with it and provided an first draft of the implementation that is now integrated in simplified form in the library.

CHAPTER 4

Networking (Move Daemon)

PS Move API contains a feature that allows you to connect more than 7 PS Move controllers to a single instance, even on systems that do not allow multiple Bluetooth adapters. The way this works is by exposing the connected controllers via IP (UDP).

This feature has been used by e.g. Edgar Rice Soiree to run a game with 20 move controllers distributed over 3 PCs.

4.1 moved2_hosts.txt

You need to place a file "moved2_hosts.txt" into the data directory of PS Move API (usually ~/.psmoveapi/, in Windows %APPDATA%/.psmoveapi/) with one IP address per line that points to a moved instance.

Example moved2_hosts.txt contents:

```
192.168.0.2
```

This would try to use the moved running on 192.168.0.2. You can have multiple IPs listed there (one per line), the servers will be tried in order.

If you add * on a single line, this will enable local network auto-discovery, meaning that any running Move Daemon instances can be auto-discovered via UDP broadcast.

4.2 API Usage

After that, you can use the PS Move API just as you normally would, and after your locally-connected controllers, controllers connected to remote machines will be used (psmove_count_connected() and psmove_connect_by_id() will take the remote controllers into account).

If you want to only use remotely-connected controllers (ignoring any controllers that are connected locally), or if you want to use moved running on localhost (where it will already provide the connected controllers to you), you should add the following API call at the beginning of your application to ensure that locally-connected devices are not used via hidapi:

```
psmove_set_remote_config(PSMove_OnlyRemote);
```

Similarly, you can disable remote controllers via the following API call (in that case, no connections to moved are carred out, only hidapi is used):

```
psmove_set_remote_config(PSMove_OnlyLocal);
```

CHAPTER 5

Configuration

5.1 Environment Variables

Some aspects of the tracker can be configured at runtime using environment variables. The following environment variables are currently recognized by the tracker (with examples for the Linux/Unix bash):

PSMOVE_TRACKER_CAMERA If set, this is the camera that will be used when using `psmove_tracker_new()` instead of using auto-detection.

Example:

```
export PSMOVE_TRACKER_CAMERA=2  # Will use the 3rd camera
```

PSMOVE_TRACKER_FILENAME If set, this will use a video file to playback instead of capturing from a camera. Any camera settings are ignored.

Example:

```
export PSMOVE_TRACKER_FILENAME=demo.avi # Will play demo.avi
```

PSMOVE_TRACKER_ROI_SIZE If set, this controls the size of the biggest (initial) ROI that will be used to track the controller. Bigger means slower in general, but recovery from tracking loss might be faster.

Example:

```
export PSMOVE_TRACKER_ROI_SIZE=200
```

PSMOVE_TRACKER_WIDTH, PSMOVE_TRACKER_HEIGHT If set, these variables control the desired size of the camera picture.

Example:

```
export PSMOVE_TRACKER_WIDTH=1280
export PSMOVE_TRACKER_HEIGHT=720
```

CHAPTER 6

Navigation Controller

While the "PS Move Navigation Controller" (CECH-ZCS1E) is marketed together with the Move Controller, in reality it's just a special variant of the SixAxis / DualShock 3 controller with its own PID.

- Vendor ID: **0x054c** (Sony Corp.)
- Product ID: **0x042f** (PlayStation Move navigation controller)

As such, the Navigation Controller does not need a special library for reading, processing and interpreting input events. After pairing, the controller can be used just like a normal gamepad device via the operating system game controller APIs.

6.1 Pairing

The `psmove` command-line utility now comes with a new sub-command `pair-nav`, which will use `sixpair` to pair a Navigation Controller (as well as a Sixaxis, DS3 and DS4) via USB. `sixpair` comes from the sixad package, but has been modified to read the Bluetooth Host address from the PS Move API libraries (for cross-platform support):

```
psmove pair-nav
```

Just like with the normal pairing function, one can pass an alternative Bluetooth host address to the pairing tool:

```
psmove pair-nav aa:bb:cc:dd:ee:ff
```

6.2 Testing

The easiest way to test the controller is via the `jstest` tool, which comes in the `joystick` package on Debian-based systems:

```
sudo apt install joystick
```

Assuming the Navigation Controller is the only joystick-style device connected, you can test it with:

```
jstest /dev/input/js0
```

There is an example file `examples/c/test_navcon.cpp` that shows how to use SDL2 to read the navigation controller (this is also built as a program if `PSMOVE_BUILD_NAVCON_TEST` is enabled in CMake):

```
test_navcon
```

6.3 Axes and Buttons

These values are also exposed in `psmove.h` as `enum PSNav_Button` and `enum PSNav_Axis` for your convenience. However, you are expected to bring your own Joystick-reading library (e.g. SDL2).

- Analog stick
  - Axis 0 (horizontal, "X")
  - Axis 1 (vertical, "Y")
  - Button 7 when pressed ("L3")
- Shoulder button ("L1"): Button 4
- Analog trigger ("L2"):
  - Axis 2 ("Z")
  - Button 5 when pressed (even just slightly)
- Cross button: Button 0
- Circle button: Button 1
- D-Pad Up: Button 8
- D-Pad Down: Button 9
- D-Pad Left: Button 10
- D-Pad Right: Button 11
- PS button: Button 6

6.4 Caveats

On Linux, all inputs work via both USB and Bluetooth. When connecting via USB, you might need to press the PS button for the controller to start reporting.

On macOS, inputs only work over Bluetooth, not via USB. Also, the button assignment is different on macOS (but `psmove.h` contains definitions for both platforms and will pick the right ones). As a special case, the analog value of the trigger isn't available on macOS at the moment.

This is not tested at all on Windows, contributions welcome.

CHAPTER 7

Mailing List

For questions, please read the archives of the PS Move Mailing List. If you cannot find an answer to your question in the archives, send an e-mail:

https://groups.google.com/forum/#!aboutgroup/psmove