Inputs Documentation

Release 0.5

Zeth Green

Feb 09, 2019

Contents

Release v0.5 **Inputs** aims to provide cross-platform Python support for keyboards, mice and gamepads.

This site covers the usage of Inputs. To see the source code see the main project website on Github.

CHAPTER 1

The User Guide

We begin with some background information about Inputs, then focus on step-by-step instructions for getting the most out of Inputs.

1.1 Introduction

The inputs module provides an easy way for your Python program to listen for user input.

Currently supported platforms are Linux (including the Raspberry Pi and Chromebooks in developer mode), Windows and the Apple Mac.

Python versions supported are all versions of Python 3 and your granddad's Python 2.7.

Inputs is in pure Python and there are no dependencies on Raspberry Pi, Linux or Windows. On the Mac, inputs needs PyObjC which the included setup.py file will install automatically (as will pip).

1.1.1 Why Inputs?

Obviously high level graphical libraries such as PyGame and PyQT will provide user input support in a very friendly way. However, the inputs module does not require your program to use any particular graphical toolkit, or even have a monitor at all.

In the Embedded Linux, Raspberry Pi or Internet of Things type situation, it is quite common not to have an X-server installed or running.

This module may also be useful where a computer needs to run a particular application full screen but you would want to listen out in the background for a particular set of user inputs, e.g. to bring up an admin panel in a digital signage setup.

This module is a single file, so if you cannot or are not allowed to use setuptools for some reason, just copy the file inputs.py into your project.

The killer feature of inputs over other similar modules is that it is cross-platform. It normalises the event data so no matter what platform you (or your users) are on, you can write a program on your operating system and it will work the same on other operating systems.

I.e. you don't have to fill your program with if Linux do this, if Windows do that, etc.

The caveat to the above is that not all operating systems support the same subset of devices by default. See *Hardware Support* for what is currently known to work.

1.1.2 Note to Children

It is pretty easy to use any user input device library, including this one, to build a keylogger. Using this module to spy on your mum or teacher or sibling is not cool and may get you into trouble. So please do not do that. Make a game instead, games are cool.

1.2 Install

Install through pypi using pip (or your favourite trendy tool):

```
pip install inputs
```

Or download it from github:

```
git clone https://github.com/zeth/inputs.git
cd inputs
python setup.py install
```

Inputs written is in pure Python and there are no dependencies on Raspberry Pi, Linux or Windows. On the Mac, inputs needs PyObjC which the included setup.py file will install automatically (as will pip).

1.3 Windows permissions

By default Windows doesn't stop inputs. However, if you have some third-party security software you may need to white-list Python. Try it and find out.

1.4 Linux permissions

On the Raspberry Pi's Raspbian everything just works.

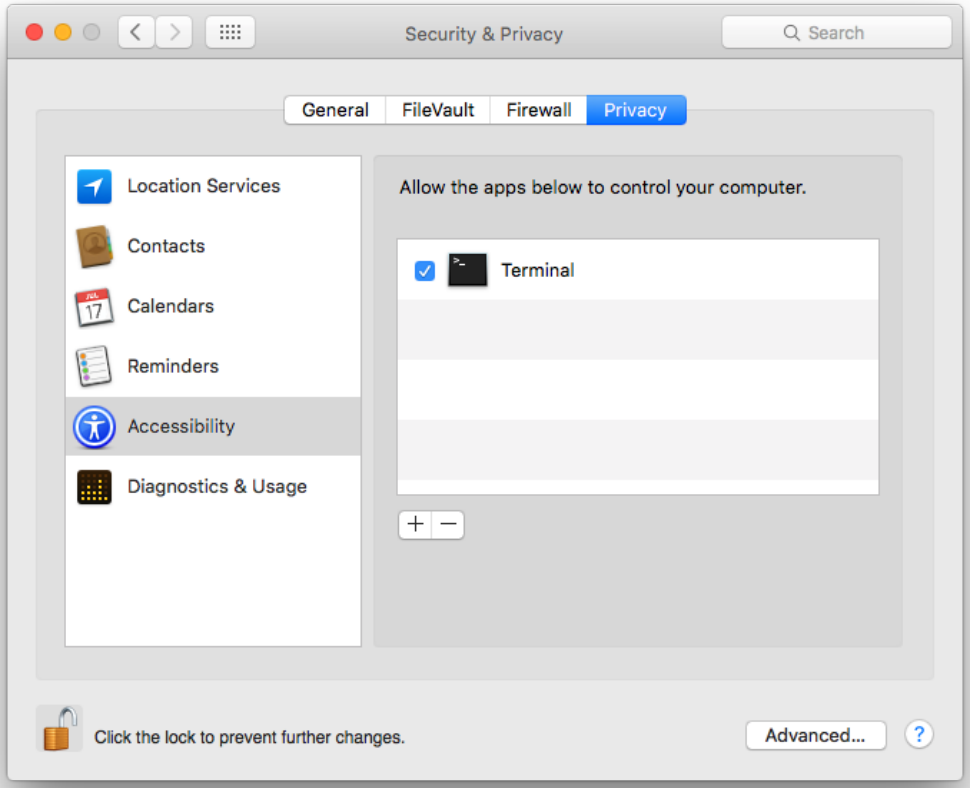
However, each Linux distribution is different. Some will work straight away, for some you need to fiddle with permissions.

Linux distributions often (quite rightly) assume that applications are installed through their package manager and given the relevant permissions to access the input devices. However, inputs.py is brand new and not yet packaged by any Linux distribution.

Therefore, if the inputs module works as root (e.g. using sudo) but not as your normal user, then you usually need to add yourself to an inputs group or similar.

1.5 Mac permissions

On the Mac, until you write a proper installer for your program, you will probably have to use the settings application to allow your program to access the input devices.



The first time you use inputs, it will not have any output, then you will either get the above settings window pop up automatically, or you will need to find your way there.

1.6 Quick Start

To access all the available input devices on the current system:

```
>>> from inputs import devices
>>> for device in devices:
...     print(device)
```

You can also access devices by type:

```
>>> devices.gamepads
>>> devices.keyboards
>>> devices.mice
>>> devices.other_devices
```

Each device object has the obvious methods and properties that you expect, stop reading now and just get playing!

If that is not high level enough, there are three basic functions that simply give you the latest events (key press, mouse movement/press or gamepad activity) from the first connected device in the category, for example:

```
>>> from inputs import get_gamepad
>>> while 1:
...     events = get_gamepad()
...     for event in events:
...         print(event.ev_type, event.code, event.state)
```

```
>>> from inputs import get_key
>>> while 1:
...     events = get_key()
...     for event in events:
...         print(event.ev_type, event.code, event.state)
```

```
>>> from inputs import get_mouse
>>> while 1:
...     events = get_mouse()
...     for event in events:
...         print(event.ev_type, event.code, event.state)
```

1.7 Advanced Information

A keyboard is represented by the Keyboard class, a mouse by the Mouse class and a gamepad by the Gamepad class. These themselves are subclasses of InputDevice.

The devices object is an instance of DeviceManager, as you can prove:

```
>>> from inputs import DeviceManager
>>> devices = DeviceManager()
```

The DeviceManager is reponsible for finding input devices on the user's system and setting up InputDevice objects.

The InputDevice objects emit instances of InputEvent. So from top down, the classes are arranged thus:

DeviceManager > InputDevice > InputEvent

So when you have a particular InputEvent instance, you can access its device and manager:

```
>>> event.device.manager
```

The event object has a property called device and the device has a property called manager.

As you can see, it is really very simple. The device manager has an attribute called codes which is giant dictionary of key, button and other codes.

1.8 Hardware Support

Support for different input devices is increasing over time. Currently tested hardware combinations are as listed below.

If one of these combinations does not work, that is likely a bug, while support for new devices is a feature request.

1.8.1 Linux

- All USB Keyboards
- All USB Mice
- Laptop built in keyboard and touchpads
- Xbox 360 Controller via USB cable
- Xbox One Controller via USB cable
- PS4 Controller via USB cable
- PS3 Controller via USB cable (press PS button if controller is not awake)
- Pi-Hut SNES Style USB Gamepad
- Wii controller

1.8.2 Chromebook (in developer mode)

All the above devices listed under Linux.

1.8.3 Raspberry Pi

All the above devices listed under Linux, plus:

Raspberry Pi Sense HAT

The microcontroller on the Raspberry Pi Sense HAT presents the joystick to the operating system as a keyboard, so find it there under keyboards. If you worry about this, you are over-thinking things.

Raspberry Pi Touch Screen

This presents as a mouse. Again please do not over think it.

1.8.4 Windows

- All USB Keyboards
- All USB Mice
- Laptop built in keyboard and touchpads
- Xbox 360 Controller via USB cable
- Xbox One Controller via USB cable

PS4 DualShock 4 Controller

Install the Windows driver from http://ds4windows.com/

Then it works perfectly. Even the touchpad works as a mouse.

1.8.5 Apple macOS High Sierra - 10.13

- Built in keyboard and touchpads
- All Apple USB keyboards and Mice

1.9 Examples

Friendly examples are provided in the inputs repository.

You can also find them below:

- devices_example.py - lists the devices that have been found on your computer.
- keyboard_example.py - shows an easy way to get keyboard events.
- mouse_example.py - shows an easy way to get mouse events.
- gamepad_example.py - shows an easy way to get gamepad events.
- vibrate_example.py - showing how to get a gamepad to vibrate.
- jstest.py - shows gamepad events in a summary view, in the style of jstest.

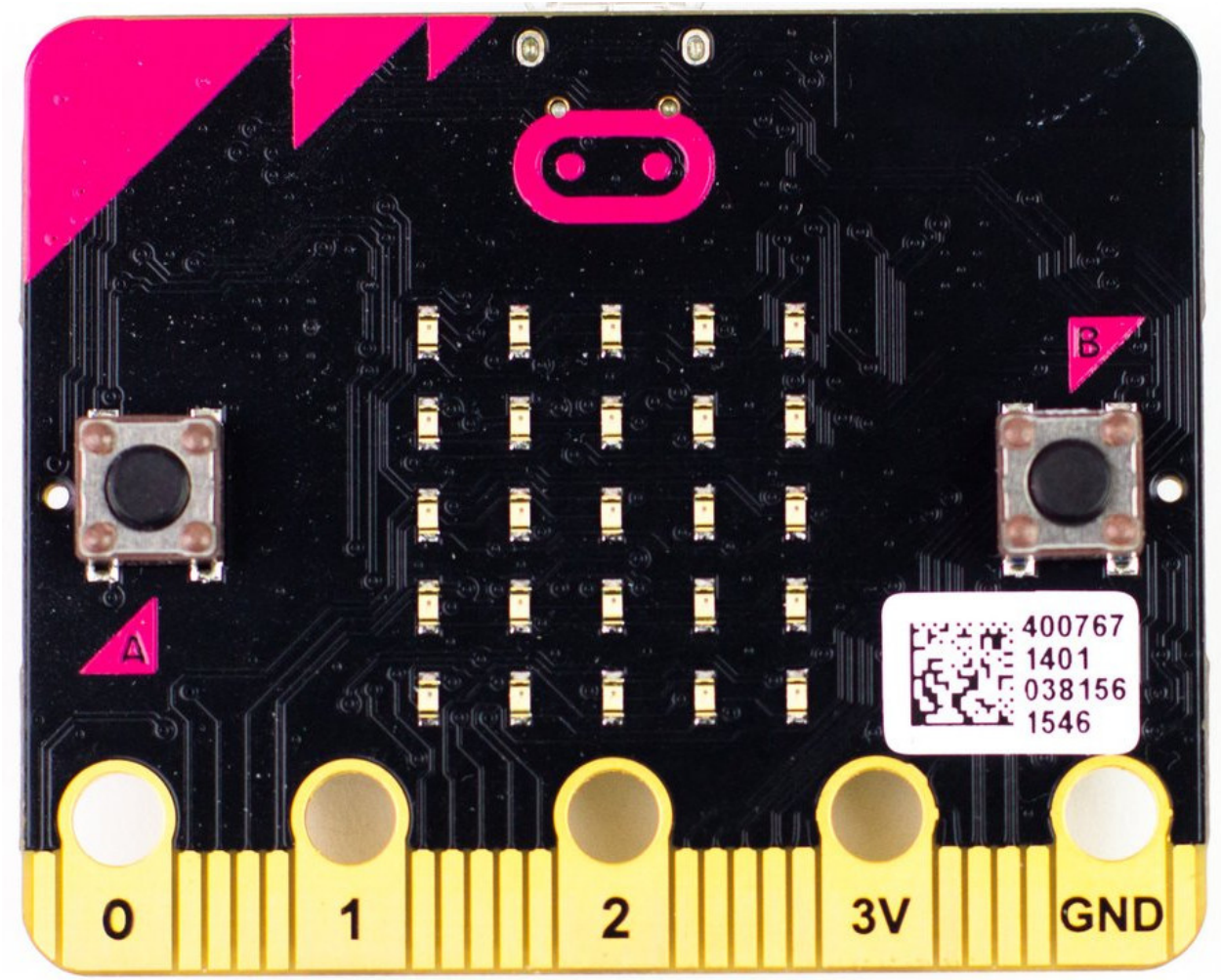
There are two examples for the BBC Microbit, see *Microbit Gamepad* for more details.

- jstest_microbit.py - shows microbit events, in the style of jstest.
- vibrate_microbit.py - an led effect to simulate vibration.

We hope to add more examples in the future.

1.10 Microbit Gamepad

The micro:bit is a tiny programmable ARM device that costs about £10-15.



1.10.1 Usage

To simulate a D-Pad, you can use the accelerometer, tilt the whole device forward, backward, left or right.

It has two press buttons labelled A and B, and three ring buttons.

To use the ring buttons, hold ground (GND) with your right hand and then press 0, 1, or 2 with your left hand.

1.10.2 Setup

You need to setup bitio. Get it from the following link and follow the instructions:

https://github.com/whaleygeek/bitio/

Basically you need to install the bitio hex file onto the microbit and put the *microbit* module into your Python path.

(Quick fix for testing is to symlink the microbit module into the same directory as the examples).

1.10.3 Usage

Plug the microbit into your computer using USB, the LED display on the Microbit should the letters IO to show that you have bitio successfully installed onto the microbit.

We start by detecting the microbit.

```
>>> import inputs
>>> inputs.devices.detect_microbit()
```

When inputs has detected the microbit, the LED display will change to show a vertical line in the middle of the screen.

You can now use the microbit like a normal gamepad:

```
>>> gamepad = inputs.devices.microbits[0]
>>> while 1:
...     events = gamepad.read()
...     for event in events:
...         print(event.ev_type, event.code, event.state)
```

1.10.4 Examples

There are two examples provided:

- jstest_microbit.py - shows microbit events, in the style of jstest.
- vibrate_microbit.py - an led effect to simulate vibration.

CHAPTER 2

Developer Information

If you want to contribute to the project, this part of the documentation is for you.

2.1 Contributor's Guide

Inputs is under active development, and contributions are more than welcome!

2.1.1 How to Help

- Run the tests and the examples on your system, provide feedback about worked and what didn't.
- Test different devices to see what happens
- Add support for more devices
- Write more examples using inputs to do fun things.

2.1.2 How to Develop

Inputs is managed using github. To get started with developing inputs, download the code from github and run the tests using the below:

```
git clone https://github.com/zeth/inputs.git
cd inputs
python setup.py test
```

What next?

1. Check for open issues or open a fresh issue to start a discussion around a bug.
2. Fork the repository on GitHub and start making your changes to a new branch.
3. Write a test which shows that the bug was fixed.

4. Send a pull request and bug the maintainer until it gets merged and published. :)

2.1.3 Get Early Feedback

A quote from a completely different Python project seems apt here:

Note: If you are contributing, do not feel the need to sit on your contribution until it is perfectly polished and complete. It helps everyone involved for you to seek feedback as early as you possibly can. Submitting an early, unfinished version of your contribution for feedback in no way prejudices your chances of getting that contribution accepted, and can save you from putting a lot of work into a contribution that is not suitable for the project.

—*Cory Benfield*

2.2 Credits

Inputs is by Zeth, all mistakes are mine.

Thanks to Dave Jones for stick.py which is not only the basis for Sense HAT stick support in this module but more importantly also taught me an easier way to parse the Evdev event format in Python:

https://github.com/RPi-Distro/python-sense-hat/blob/master/sense_hat/stick.py

https://github.com/waveform80/pisense/blob/master/pisense/stick.py

Thanks to Andy (r4dian) and Jason R. Coombs whose existing (MIT licenced) Python examples for Xbox 360 controller support on Windows helped me understand xinput greatly. Xbox 360 controller support on Windows here is based on their work:

https://github.com/r4dian/Xbox-360-Controller-for-Python

http://pydoc.net/Python/jaraco.input/1.0.1/jaraco.input.win32.xinput/