# Use VEC/Infinity USB Foot Pedal as a Keyboard Under Linux

A. Cynic

Version , Thu Aug 30 16:57:51 MDT 2018

I own a 3-switch USB foot pedal that looks like this:


This is how I got it working under Linux so that I can assign any keysym to each button which I can then bind to functions in my window manager or programs as I see fit.

(It is unintuitive to me that the left pedal is marked FWD and the right REW, but it doesn't matter since we will map them to whatever function we want.)

I'm unsure of its ancestry, but the label on the back of my pedal says "Infinity IN-USB-1" and I've seen newer pedals online called IN-USB-2 which have a different case but as far as I can tell provide an identical interface (same idVendor and idProduct, same scancodes). The In-USB-2 pedals, by VEC Electronics, also

seem to be the same as some of the X-keys pedals sold by P.I. Engineering. So this guide should apply to any of them, and a similar process will likely work to get other USB foot pedals setup.

For more direct, less detailed instructions on getting your VEC footpedal to work, see the Parlatype documentation on footpedals.

# evdev

When I plug my pedal in, `dmesg` reports the following information:

```
[81924.859319] usb 10-1: new low-speed USB device
number 25 using uhci_hcd
[81925.261424] usb 10-1: New USB device found,
idVendor=05f3, idProduct=00ff
[81925.261427] usb 10-1: New USB device strings:
Mfr=1, Product=2, SerialNumber=0
[81925.261429] usb 10-1: Product: VEC USB Footpedal
[81925.261431] usb 10-1: Manufacturer: VEC
[81925.275747] input: VEC  VEC USB Footpedal as
/devices/pci0000:00/0000:00:1d.2/usb10/10-1/10-
1:1.0/0003:05F3:00FF.0022/input/input41
[81925.276084] hid-generic 0003:05F3:00FF.0022:
input,hiddev0,hidraw4: USB HID v1.00 Device [VEC  VEC
USB Footpedal] on usb-0000:00:1d.2-1/input0
```

This is a good sign. Already we see the vendor ID (05F3) and product ID (00FF), which we will use to match the device in udev rules, and udev recognizes it as “VEC USB Footpedal”. The

kernel's HID driver seems to recognize it as an input device and exposes its events as a character device via evdev.

Sure enough, running `evtest` (in debian: `apt install evtest`) sees it and shows the `/dev/input/eventX` node it was assigned:

```
...
/dev/input/event4:      VEC  VEC USB Footpedal
...
```

In my case it is `event4` (this will differ depending on how many input devices you have plugged in), and choosing `4` from the `evtest` prompt then pressing the foot pedal switches shows that it is detecting the events!

*Output of `$ evtest` after pressing and releasing each pedal from left to right*

```
Event: time 1535305153.576554, type 4 (EV_MSC), code 4
(MSC_SCAN), value 90001
Event: time 1535305153.576554, type 1 (EV_KEY), code
256 (BTN_0), value 1
Event: time 1535305153.576554, --------------
SYN_REPORT ------------
Event: time 1535305153.696549, type 4 (EV_MSC), code 4
(MSC_SCAN), value 90001
Event: time 1535305153.696549, type 1 (EV_KEY), code
256 (BTN_0), value 0
Event: time 1535305153.696549, --------------
SYN_REPORT ------------
Event: time 1535305154.224555, type 4 (EV_MSC), code 4
(MSC_SCAN), value 90002
Event: time 1535305154.224555, type 1 (EV_KEY), code
```

```
257 (BTN_1), value 1
Event: time 1535305154.224555, --------------
SYN_REPORT ------------
Event: time 1535305154.352584, type 4 (EV_MSC), code 4
(MSC_SCAN), value 90002
Event: time 1535305154.352584, type 1 (EV_KEY), code
257 (BTN_1), value 0
Event: time 1535305154.352584, --------------
SYN_REPORT ------------
Event: time 1535305155.992658, type 4 (EV_MSC), code 4
(MSC_SCAN), value 90003
Event: time 1535305155.992658, type 1 (EV_KEY), code
258 (BTN_2), value 1
Event: time 1535305155.992658, --------------
SYN_REPORT ------------
Event: time 1535305156.136669, type 4 (EV_MSC), code 4
(MSC_SCAN), value 90003
Event: time 1535305156.136669, type 1 (EV_KEY), code
258 (BTN_2), value 0
Event: time 1535305156.136669, --------------
SYN_REPORT ------------
```

From this output we can see the scancodes sent by each pedal switch, and the default keycodes they are translated to by the kernel (by default they act like the left, middle, and right mouse buttons).

*Table 1. Summary of default scancodes and keycodes sent by the pedal*

| Pedal | Scancode (hex) | Keycode |
|---|---|---|
| Left | 90001 | `BTN_0` (left mouse button) |

| Pedal | Scancode (hex) | Keycode |
|---|---|---|
| Center | 90002 | `BTN_1` (middle mouse button) |
| Rightt | 90003 | `BTN_2` (right mouse button) |

So it seems to work out-of-the-box, with only two issues:

1. Even though the kernel sees the pedal, it is invisible to Xorg and the programs I actually want to use with it.
2. Having it act like mouse buttons is not very useful; I want to re-map its keysyms to something I can use as hotkeys.

# Xorg

Pressing the pedals has no apparent effect under X. Running `xev` and pushing the switches also has no effect. It is not included in the output of `libinput list-devices` — Xorg is simply not seeing the footpedal as in input device.

One slight frustration is that manually running `udevadm trigger` allows X to see the pedal, but after unplugging and plugging it back in results in it once again being ignored.

The problem is that udev is not tagging the pedal as a keyboard or a mouse when it is added during the hotplug event, so X is ignoring it. This can be seen by running the `udevadm info` tool:

*Output of* `$ udevadm info /dev/input/event`

```
udevadm info /dev/input/event4
P: /devices/pci0000:00/0000:00:1d.2/usb10/10-1/10-
```

```
1:1.0/0003:05F3:00FF.0022/input/input41/event4
N: input/event4
S: input/by-id/usb-VEC_VEC_USB_Footpedal-event-if00
S: input/by-path/pci-0000:00:1d.2-usb-0:1:1.0-event
E: DEVLINKS=/dev/input/by-path/pci-0000:00:1d.2-usb-
0:1:1.0-event /dev/input/by-id/usb-
VEC_VEC_USB_Footpedal-event-if00
E: DEVNAME=/dev/input/event4
E: DEVPATH=/devices/pci0000:00/0000:00:1d.2/usb10/10-
1/10-1:1.0/0003:05F3:00FF.0022/input/input41/event4
E: ID_BUS=usb
E: ID_INPUT=1
E: ID_MODEL=VEC_USB_Footpedal
E: ID_MODEL_ENC=VEC\x20USB\x20Footpedal
E: ID_MODEL_ID=00ff
E: ID_PATH=pci-0000:00:1d.2-usb-0:1:1.0
E: ID_PATH_TAG=pci-0000_00_1d_2-usb-0_1_1_0
E: ID_REVISION=0100
E: ID_SERIAL=VEC_VEC_USB_Footpedal
E: ID_TYPE=hid
E: ID_USB_DRIVER=usbhid
E: ID_USB_INTERFACES=:030000:
E: ID_USB_INTERFACE_NUM=00
E: ID_VENDOR=VEC
E: ID_VENDOR_ENC=VEC\x20
E: ID_VENDOR_ID=05f3
E: LIBINPUT_DEVICE_GROUP=3/5f3/ff:usb-0000:00:1d.2-1
E: MAJOR=13
E: MINOR=68
E: SUBSYSTEM=input
E: USEC_INITIALIZED=81926725812
```

The device has the `ID_INPUT` tag, but no `ID_INPUT_KEYBOARD` or `ID_INPUT_MOUSE` tag which Xorg's evdev/libinput drivers look for.

(For some details on how udev decides what is a keyboard or mouse, see Matt Fischer's "How Does udev Know What's a Keyboard or Mouse?") Luckily the fix is a simple one-line udev rule file (thanks to Parlatype developer Gabor Karsay for providing this solution in Parlatype Issue 28):

*/etc/udev/rules.d/10-vec-usb-footpedal.rules*

```
ACTION=="add|change", KERNEL=="event[0-9]*",
ATTRS{idVendor}=="05f3", ATTRS{idProduct}=="00ff",
ENV{ID_INPUT_KEYBOARD}="1"
```

Create a file called `/etc/udev/rules.d/10-vec-usb-footpedal.rules` containing that line. No need to run any command to refresh udev, the new file should automatically be detected. Now whenever the foot pedal is plugged in, it will be given the `ID_INPUT_KEYBOARD=1` tag and Xorg/libinput will use it as an input device!

# Remapping keysyms with udev hwdb

Having three extra mouse buttons to use with my foot is not very useful to me. We will fix this with a udev hwdb file.

Create a file under `/etc/udev/hwdb.d/` (I put mine in `/etc/udev/hwdb.d/60-usb-footpedal.hwdb`) containing these lines:

*/etc/udev/hwdb.d/60-usb-footpedal.hwdb*

```
evdev:input:b*v05F3p00FF*
 KEYBOARD_KEY_90001=f14
 KEYBOARD_KEY_90002=f15
 KEYBOARD_KEY_90003=f16
```

This syntax is good for udev version 220 and later.

This time we do need to inform the system to update the binary hwdb file:

```
$ sudo systemd-hwdb update
```

And then unplug and repug the device.

The first line of the hwdb file matches our device (vendor 05F3, product 00FF), and the subsequent lines map a scancode (hex) to a keycode. I chose the F14, F15, and F16 function keys, but a list of available keycodes is defined in /usr/include/linux/input-event-codes.h; to use the names #defined in that file as hwdb keycodes, simply convert them to lowercase and remove the `key_` prefix.

The comments at the top of the system-wide configuration file contain some documentation: /lib/udev/hwdb.d/60-keyboard.hwdb And for more generic instructions on using udev to remap keys, see Kim Jongyul's "Linux keymapping with udev hwdb."

The default (pc+us) xkb keyboard layout on my computer maps `F14`, `F15`, and `F16` to the `XF86Launch5`, `XF86Launch6`, and `XF86Launch7` keysyms, respectively. Using those keysyms, each pedal switch can be mapped as a hotkey in your desktop or window manager.

# Use new function keys with Vim

Unfortunately not all programs can bind to arbitrary keysyms like the XF86* keys. A list of keys available to vim, for example, can be found at :h t_ku.

One solution is to use `xmodmap` to map the keycodes to the actual `F14`, `F15`, and `F16` keysyms. Use `xev` to find out what keycode X is seeing for each pedal. Here's example output from the left pedal:

```
KeyPress event, serial 34, synthetic NO, window
0x400001,
    root 0x439, subw 0x0, time 154723303, (340,452),
root:(1051,943),
    state 0x10, keycode 192 (keysym 0x1008ff45,
XF86Launch5), same_screen YES,
    XLookupString gives 0 bytes:
    XmbLookupString gives 0 bytes:
    XFilterEvent returns: False
```

The thing to note is the "keycode" is 192. Then to remap the keysyms:

```
$ xmodmap -e "keycode 192 = F14"
$ xmodmap -e "keycode 193 = F15"
$ xmodmap -e "keycode 194 = F16"
```

I put these commands in my `.xsession` file so they run every time I log in to X. Key autorepeating can be supressed (so that you can leave your foot on the pedal without it triggering more key presses) with the `xset` command:

```
$ xset -r 193
```

The disadvantage of simply placing `xmodmap`/`xset` commands in the `.xsession` file is that they need to be re-run every time the pedal is unplugged. One solution would be to use a udev rule to run a script containing those `xmodmap` commands every time the pedal is plugged back in.

Both gvim and neovim now see the function keys and you can map them as normal:

```
inoremap <F15> Middle pedal
```

But depending on your terminal settings, vim in the console probably still does not recognize the keys. To fix it, set vim's keycodes to whatever escape sequence your terminal sends for each pedal (in insert mode type Ctrl-Q then hit the pedal).

I have these lines in my .vimrc file, which allows me to map `F14` et al. in vim (gvim and neovim ignore these settings):

```
" Map higher F keys to codes sent by libvte
" The escape codes are literal:
" type Ctrl-q then press the footpedal switch
set <F14>=^[[26~
set <F15>=^[[28~
set <F16>=^[[29~
```

Success at last!

For some background on terminal escape codes for function keys see Phil Gold's "Terminal Function Key Escape Codes"

For anyone planning on transcribing audio in neovim, I recently came across this plugin for controlling the playback of audio: gallcaras/transcribe.nvim

# Other options

A Python program called footcontroller by rolfofsacony reads events from the evdev device (`/dev/input/eventX`) and can be configured to issue key strokes (via xdotool) or run scripts in response to stepping on the pedals. The program allows for configuring several sets of commands and switching between them (so you can use your foot pedal for different purposes: transcription, controlling your media player, to launch apps, to exit vim, etc.) The user manual for the program includes several examples and instructions for switching between command sets using hotkeys. Footcontroller can be used in lieu of the xmodmap step above.

In addition to the evdev device, the kernel's HID driver also

creates a hiddev node (`/dev/usb/hiddevX`) which provides a more raw interface to the pedal state. Here is a C program which reads the HID node to handle pedal events; it includes a sample program to control mplayer using the pedal: infinity-pedal driver

Because the pedal is an HID device, it should be possible to write a cross-platform program to read it using something like HIDAPI

# Mac OS X

P.I. Engineering provides software that works with their devices (including the IN-USB-1 and IN-USB-2 foot pedals): Software for X-keys.

Most impressive is a GUI program called ControllerMate by OrderedBytes, which provides a powerful graphical scripting language for triggering actions from input devices. Best of all is they offer a free version especially for X-keys devices which works with our pedal.