

4:evdev

evdev - Generic Linux input driver

Contents [1 SYNOPSIS](#) [2 DESCRIPTION](#) [3 SUPPORTED HARDWARE](#) [4 CONFIGURATION DETAILS](#) [5 BASIC CONFIGURATIONS](#) [6 ADVANCED OPTIONS](#) [7 DEVICE SPECIFICATION](#) [8 RELATIVE AXIS CONFIGURATION](#) [9 ABSOLUTE AXIS CONFIGURATION](#) [10 BUTTON CONFIGURATION](#) [11 KEYBOARD CONFIGURATION](#) [11.1 RELATED](#) [11.2 CATEGORY](#)

SYNOPSIS

```
Section "InputDevice"
    Identifier "devname"
    Driver "evdev"
    Option "Device" "devpath"
    ...
EndSection
```

DESCRIPTION

evdev is an Xorg input driver for Linux's generic event devices. It therefore supports all input devices that the kernel knows about, including most mice and keyboards.

The **evdev** driver can serve as both a pointer and a keyboard input device, and may be used as both the core keyboard and the core pointer. Multiple input devices are supported by multiple instances of this driver, with one

Load directive for evdev in the Module section of your xorg.conf for each input device that will use this driver.

SUPPORTED HARDWARE

In general, any input device that the kernel has a driver for can be accessed through the **evdev** driver. See the Linux kernel documentation for a complete list.

CONFIGURATION DETAILS

Please refer to [xorg.conf\(5\)](#) for general configuration details and for options that can be used with all input drivers. This section only covers configuration details specific to this driver.

BASIC CONFIGURATIONS

Most users of this driver will probably be quite happy with the following for all QWERTY keyboards:

```
Section "InputDevice"
    Identifier "keyboard"
    Driver "evdev"
    Option "evBits" "+1"
    Option "keyBits" "~1-255 ~352-511"
    Option "Pass" "3"
    ...
EndSection
```

And the following for all mice:

```
Section "InputDevice"
    Identifier "mouse"
    Driver "evdev"
    Option "evBits" "+1-2"
    Option "keyBits" "~272-287"
    Option "relBits" "~0-2 ~6 ~8"
    Option "Pass" "3"
    ...
EndSection
```

To understand what those Bits options do, or for more complex configurations, please see **ADVANCED OPTIONS** below.

ADVANCED OPTIONS DEVICE SPECIFICATION

For this section you'll want to have knowledge of [glob\(7\)](#) and our evil BIT MATCHING SPECIFICATION **stuff**.

The following driver **Options** control what devices are accepted:

Option "Device" "*string*"

Specifies the device node through which the device can be accessed. At this time ONLY /dev/input/event<N>, where <N> is an integer, are matched against this field.

This option uses globbing.

Please note that use of this option is strongly discouraged.

Option "Name" "*string*"

Specifies the device name for the device you wish to use. The device name is generally the only consistent identifier for devices that are commonly unplugged and plugged back into different ports.

A list of currently plugged in devices and associated device names can be obtained by typing "cat

`/proc/bus/input/devices`", the "Name" field is the value you want for this option.

This option uses globbing.

Option "Phys" "string"

Specifies the device phys string for the device you wish to use.

The phys string is generally consistent to the USB port a device is plugged into.

A list of currently plugged in devices and associated device names can be obtained by typing "cat

`/proc/bus/input/devices`", the "Phys" field is the value you want for this option.

This option uses globbing.

Option "<map>Bits" "bit specifier"

Specifies device capability bits which must be set, possibly set, or unset.

<map>Bits: Where map is one of ev, key, rel, abs, msc, led, snd, or ff.

The bit specifier format is a string consisting of +<n>, -<n>, and ~<n> space separated specifiers, where

<n> is a positive integer or integer range. (The latter given in the format of 2-6.)

+ specifies bits which must be set.

- specifies bits which must not be set.

~ is a little more complex, it specifies that at least one of the bits given with ~ for the field in question must be set, but it doesn't matter how many or which of the bits. (It is actually the most useful of the 3 specifiers.)

As an example '+0 +3 -1-2 ~5-10', requires bits 0 and 3 be set, bits 1 and 2 to not be set, and at least one bit in the range of 5 to 10 be set.

An annoyingly formatted set of bitmasks for your devices can be obtained by typing "cat `/proc/bus/input/devices`", and `/usr/include/linux/input.h` should contain the defines which declare what bits are what for each field.

Option "bustype" "integer"

Specifies the bus ID for the device you wish to use.

This is either 0 (the default, matches anything), or the Bus=<n> field in `/proc/bus/input/devices` for your device.

This value depends on what type of bus your device is connected to.

Option "vendor" "integer"

Specifies the vendor ID for the device you wish to use.

This is either 0 (the default, matches anything), or the Vendor=<n> field in `/proc/bus/input/devices` for your device.

This value should remain constant barring perhaps firmware updates to the device itself.

Option "version" "integer"

Specifies the version for the device you wish to use.

This is either 0 (the default, matches anything), or the
Version=<n> field in **/proc/bus/input/devices** for
your device.

This value should remain constant barring perhaps firmware updates
to the device itself.

Option "product" "integer"

Specifies the product ID for the device you wish to use.

This is either 0 (the default, matches anything), or the
Product=<n> field in **/proc/bus/input/devices** for
your device.

This value should remain constant barring perhaps firmware updates
to the device itself.

Option "Pass" "integer"

Specifies the order in which evdev will scan for devices.

This is in the range of 0 to 3, and is used for the case where more
then one evdev inputsection matches
the same device.

An inputsection with a lower pass number will always beat out one
with a higher pass number. Order when

both sections are the same number is undefined.

The default is 0.