Installation

1  From a binary package 3
2  From source 5
3  Specifying header locations 7
4  Quick Start 9
   4.1 Listing accessible event devices 9
   4.2 Reading events from a device 9
   4.3 Accessing event codes 10
   4.4 Listing and monitoring input devices 10
5  Tutorial 11
   5.1 Listing accessible event devices 11
   5.2 Listing device capabilities 11
   5.3 Listing device capabilities (devices with absolute axes) 12
   5.4 Getting and setting LED states 12
   5.5 Getting currently active keys 12
   5.6 Reading events 12
   5.7 Reading events (using asyncio) 13
   5.8 Reading events from multiple devices (using select) 13
   5.9 Reading events from multiple devices (using selectors) 14
   5.10 Reading events from multiple devices (using asyncio) 14
   5.11 Accessing evdev constants 15
   5.12 Getting exclusive access to a device 15
   5.13 Associating classes with event types 15
   5.14 Injecting input 16
   5.15 Injecting events (using a context manager) 16
   5.16 Specifying uinput device options 16
   5.17 Create uinput device with capabilities of another device 17
   5.18 Create uinput device capable of receiving FF-effects 17
   5.19 Injecting an FF-event into first FF-capable device found 18
6  API Reference 19
   6.1 events 19
   6.2 eventio 20
   6.3 eventio_async 22
This package provides bindings to the generic input event interface in Linux. The *evdev* interface serves the purpose of passing events generated in the kernel directly to userspace through character devices that are typically located in `/dev/input/`.

This package also comes with bindings to *uinput*, the userspace input subsystem. *Uinput* allows userspace programs to create and handle input devices that can inject events directly into the input subsystem.

In other words, *python-evdev* allows you to read and write input events on Linux. An event can be a key or button press, a mouse movement or a tap on a touchscreen.
From a binary package

Python-evdev has been packaged for the following GNU/Linux distributions:
Consult the documentation of your OS package manager for installation instructions.
The latest stable version of `python-evdev` can be installed from pypi, provided that you have gcc/clang, pip and the Python and Linux development headers installed on your system. Installing them is distribution specific and typically falls in one of the following categories:

On a Debian compatible OS:

```bash
$ apt-get install python-dev python-pip gcc
$ apt-get install linux-headers-$(uname -r)
```

On a Redhat compatible OS:

```bash
$ yum install python-devel python-pip gcc
$ yum install kernel-headers-$(uname -r)
```

On Arch Linux and derivatives:

```bash
$ pacman -S core/linux-api-headers python-pip gcc
```

Once all dependencies are available, you may install `python-evdev` using pip:

```bash
$ sudo pip install evdev  # available globally
$ pip install --user evdev  # available to the current user
```
Specifying header locations

By default, the setup script will look for the \file{input.h} and \file{input-event-codes.h}\footnote{\file{input-event-codes.h} is found only in more recent kernel versions.} header files /usr/include/linux.

You may use the \texttt{--evdev-headers} option to the \texttt{build_ext} \texttt{setuptools} command to specify the location of these header files. It accepts one or more colon-separated paths. For example:

```bash
$ python setup.py build_ext \n   --evdev-headers buildroot/input.h:buildroot/input-event-codes.h \n   --include-dirs buildroot/ \n   install # or any other command (e.g. develop, bdist, bdist_wheel)
```
CHAPTER 4

Quick Start

4.1 Listing accessible event devices

```python
>>> import evdev

>>> devices = [evdev.InputDevice(path) for path in evdev.list_devices()]

>>> for device in devices:
...     print(device.path, device.name, device.phys)
/dev/input/event1  USB Keyboard  usb-0000:00:12.1-2/input0
/dev/input/event0  USB Optical Mouse  usb-0000:00:12.0-2/input0
```

**Note:** If you do not see any devices, ensure that your user is in the correct group (typically `input`) to have read/write access.

4.2 Reading events from a device

```python
>>> import evdev

>>> device = evdev.InputDevice('/dev/input/event1')

>>> print(device)
device /dev/input/event1, name "USB Keyboard", phys "usb-0000:00:12.1-2/input0"

>>> for event in device.read_loop():
...     if event.type == evdev.ecodes.EV_KEY:
...         print(evdev.categorize(event))
... # pressing 'a' and holding 'space'
key event at 1337016188.396030, 30 (KEY_A), down
key event at 1337016188.492033, 30 (KEY_A), up
key event at 1337016189.772129, 57 (KEY_SPACE), down
```

(continues on next page)
4.3 Accessing event codes

The `evdev.ecodes` module provides reverse and forward mappings between the names and values of the event subsystem constants.

```python
>>> from evdev import ecodes

>>> ecodes.KEY_A
... 30
>>> ecodes.ecodes['KEY_A']
... 30
>>> ecodes.KEY[30]
... 'KEY_A'
>>> ecodes.bytype[ecodes.EV_KEY][30]
... 'KEY_A'

# A single value may correspond to multiple event codes.

>>> ecodes.KEY[152]
... ['KEY_COFFEE', 'KEY_SCREENLOCK']
```

4.4 Listing and monitoring input devices

The `python-evdev` package also comes with a small command-line program for listing and monitoring input devices:

```
$ python -m evdev.evtest
```
5.1 Listing accessible event devices

```python
>>> import evdev

>>> devices = [evdev.InputDevice(path) for path in evdev.list_devices()]

>>> for device in devices:
...    print(device.path, device.name, device.phys)
/dev/input/event1  Dell Dell USB Keyboard  usb-0000:00:12.1-2/input0
/dev/input/event0  Dell USB Optical Mouse  usb-0000:00:12.0-2/input0
```

5.2 Listing device capabilities

```python
>>> import evdev

>>> device = evdev.InputDevice('/dev/input/event0')

>>> print(device)
/dev/input/event0, name "Dell USB Optical Mouse", phys "usb-0000:00:12.0-2/
˓
/input0"

>>> device.capabilities()
...
... { 0: [0, 1, 2], 1: [272, 273, 274, 275], 2: [0, 1, 6, 8], 4: [4] }

>>> device.capabilities( verbose=True)
...
... { (EV_SYN, 0): [('SYN_REPORT', 0), ('SYN_CONFIG', 1), ('SYN_MT_REPORT', 2)],
...   (EV_KEY, 1): [('BTN_MOUSE', 272), ('BTN_RIGHT', 273), ('BTN_MIDDLE', 274), ('BTN_SIDE', 275)], ... }
5.3 Listing device capabilities (devices with absolute axes)

```python
>>> import evdev

>>> device = evdev.InputDevice('/dev/input/event7')
>>> print(device)
device /dev/input/event7, name "Wacom Bamboo 2FG 4x5 Finger", phys ""

>>> device.capabilities()
...
... { 1: [272, 273, 277, 278, 325, 330, 333],
...  3: [(0, AbsInfo(min=0, max=15360, fuzz=128, flat=0)),
...     (1, AbsInfo(min=0, max=10240, fuzz=128, flat=0))]

>>> device.capabilities(Verbose=True)
...
... { ('EV_KEY', 1): [('BTN_MOUSE', 272), ('BTN_RIGHT', 273), ...],
...  ('EV_ABS', 3): [('ABS_X', 0), AbsInfo(min=0, max=15360, fuzz=128, flat=0)),
...    ('ABS_Y', 1), AbsInfo(min=0, max=10240, fuzz=128, flat=0))

>>> device.capabilities(absinfo=False)
...
... { 1: [272, 273, 277, 278, 325, 330, 333],
...  3: [0, 1, 47, 53, 54, 57]}
```

5.4 Getting and setting LED states

```python
>>> dev.leds(Verbose=True)
...
[('LED_NUML', 0), ('LED_CAPSL', 1)]

>>> dev.leds()
...
[0, 1]

>>> dev.set_led(ecodes.LED_NUML, 1)  # enable numlock

>>> dev.set_led(ecodes.LED_NUML, 0)  # disable numlock
```

5.5 Getting currently active keys

```python
>>> dev.active_keys(Verbose=True)
...
[('KEY_3', 4), ('KEY_LEFTSHIFT', 42)]

>>> dev.active_keys()
...
[4, 42]
```

5.6 Reading events

Reading events from a single device in an endless loop.

```python
>>> from evdev import InputDevice, categorize, ecodes

>>> dev = InputDevice('/dev/input/event1')

>>> print(dev)
```
5.7 Reading events (using asyncio)

Note: This requires Python 3.5+ for the async/await keywords.

5.8 Reading events from multiple devices (using select)
5.9 Reading events from multiple devices (using selectors)

This can also be achieved using the `selectors` module in Python 3.4:

```python
from evdev import InputDevice
from selectors import DefaultSelector, EVENT_READ

selector = selectors.DefaultSelector()

mouse = evdev.InputDevice('/dev/input/event1')
keybd = evdev.InputDevice('/dev/input/event2')

# This works because InputDevice has a `fileno()` method.
selector.register(mouse, selectors.EVENT_READ)
selector.register(keybd, selectors.EVENT_READ)

while True:
    for key, mask in selector.select():
        device = key.fileobj
        for event in device.read():
            print(event)
```

5.10 Reading events from multiple devices (using asyncio)

Yet another possibility is the `asyncio` module from Python 3.4:

```python
import asyncio, evdev
def print_events(device):
    @asyncio.coroutine
    def device_reader():
        try:
            events = yield from device.async_read()
            for event in events:
                print(device.path, evdev.categorize(event), sep=':')
        finally:
            device.close()  # close device

mouse = evdev.InputDevice('/dev/input/eventX')
keybd = evdev.InputDevice('/dev/input/eventY')

asyncio.async(print_events(mouse))
asyncio.async(print_events(keybd))
loop = asyncio.get_event_loop()
loop.run_forever()
```
Since Python 3.5, the `async/await` syntax makes this even simpler:

```python
import asyncio, evdev

mouse = evdev.InputDevice('/dev/input/event4')
keybd = evdev.InputDevice('/dev/input/event5')

async def print_events(device):
    async for event in device.async_read_loop():
        print(device.path, evdev.categorize(event), sep=': ')

for device in mouse, keybd:
    asyncio.ensure_future(print_events(device))

loop = asyncio.get_event_loop()
loop.run_forever()
```

### 5.11 Accessing evdev constants

```python
>>> from evdev import ecodes
... ecodes.KEY_A, ecodes.ecodes['KEY_A']
... (30, 30)
... ecodes.KEY[30]
... 'KEY_A'
... ecodes.bytype[ecodes.EV_KEY][30]
... 'KEY_A'
... ecodes.KEY[152] # a single value may correspond to multiple codes
... ['KEY_COFFEE', 'KEY_SCREENLOCK']
```

### 5.12 Getting exclusive access to a device

```python
>>> dev.grab()   # become the sole recipient of all incoming input events
>>> dev.ungrab()
```

This functionality is also available as a context manager.

```python
>>> with dev.grab_context():
...     pass
```

### 5.13 Associating classes with event types

```python
>>> from evdev import categorize, event_factory, ecodes

>>> class SynEvent(object):
...     def __init__(self, event):
...         ...

>>> event_factory[ecodes.EV_SYN] = SynEvent
```

See `events` for more information.
5.14 Injecting input

```python
>>> from evdev import UInput, ecodes as e

>>> ui = UInput()

>>> # accepts only KEY_* events by default
>>> ui.write(e.EV_KEY, e.KEY_A, 1)  # KEY_A down
>>> ui.write(e.EV_KEY, e.KEY_A, 0)  # KEY_A up

>>> ui.syn()

>>> ui.close()
```

5.15 Injecting events (using a context manager)

```python
>>> ev = InputEvent(1334414993, 274296, ecodes.EV_KEY, ecodes.KEY_A, 1)

>>> with UInput() as ui:
...     ui.write_event(ev)
...     ui.syn()
```

5.16 Specifying uinput device options

```python
>>> from evdev import UInput, AbsInfo, ecodes as e

>>> cap = {
...     e.EV_KEY : [e.KEY_A, e.KEY_B],
...     e.EV_ABS : [  
...         (e.ABS_X, AbsInfo(value=0, min=0, max=255, fuzz=0, flat=0, resolution=0)),  
...         (e.ABS_Y, AbsInfo(0, 0, 255, 0, 0, 0)),  
...         (e.ABS_MT_POSITION_X, (0, 128, 255, 0)) ]

>>> ui = UInput(cap, name='example-device', version=0x3)

>>> print(ui)
name "example-device", bus "BUS_USB", vendor "0001", product "0001", version "0003"
event types: EV_KEY EV_ABS EV_SYN

>>> print(ui.capabilities())
{0: [0, 1, 3],
 1: [30, 48],
 3: [(0, AbsInfo(value=0, min=0, max=0, fuzz=255, flat=0, resolution=0)),
     (1, AbsInfo(value=0, min=0, max=0, fuzz=255, flat=0, resolution=0)),
     (53, AbsInfo(value=0, min=255, max=255, fuzz=128, flat=0, resolution=0))]

>>> # move mouse cursor
>>> ui.write(e.EV_ABS, e.ABS_X, 20)
>>> ui.write(e.EV_ABS, e.ABS_Y, 20)
>>> ui.syn()
```
5.17 Create `uinput` device with capabilities of another device

```python
>>> from evdev import UInput, InputDevice

>>> mouse = InputDevice('/dev/input/event1')
>>> keybd = '/dev/input/event2'

>>> ui = UInput.from_device(mouse, keybd, name='keyboard-mouse-device')
```

5.18 Create `uinput` device capable of receiving FF-effects

```python
import asyncio
from evdev import UInput, categorize, ecodes

cap = {
    ecodes.EV_FF: [ecodes.FF_RUMBLE],
    ecodes.EV_KEY: [ecodes.KEY_A, ecodes.KEY_B]
}

ui = UInput(cap, name='test-controller', version=0x3)

async def print_events(device):
    async for event in device.async_read_loop():
        print(categorize(event))

        # Wait for an EV_UINPUT event that will signal us that an
        # effect upload/erase operation is in progress.
        if event.type != ecodes.EV_UINPUT:
            pass

        if event.code == ecodes.UI_FF_UPLOAD:
            upload = device.begin_upload(event.value)
            upload.retval = 0

            print(f'
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
5.19 Injecting an FF-event into first FF-capable device found

```python
from evdev import ecodes, InputDevice, ff

# Find first EV_FF capable event device (that we have permissions to use).
for name in evdev.list_devices():
    dev = InputDevice(name)
    if ecodes.EV_FF in dev.capabilities():
        break

rumble = ff.Rumble(strong_magnitude=0x0000, weak_magnitude=0xffff)
effect_type = ff.EffectType(ff_rumble_effect=rumble)
duration_ms = 1000

effect = ff.Effect(
    ecodes.FF_RUMBLE, -1, 0,
    ff.Trigger(0, 0),
    ff.Replay(duration_ms, 0),
    ff.EffectType(ff_rumble_effect=rumble)
)

repeat_count = 1
effect_id = dev.upload_effect(effect)
dev.write(e.EV_FF, effect_id, repeat_count)
dev.erase_effect(effect_id)
```
CHAPTER 6

API Reference

6.1 events

This module provides the `InputEvent` class, which closely resembles the `input_event` struct defined in `linux/input.h`:

```c
struct input_event {
    struct timeval time;
    __u16 type;
    __u16 code;
    __s32 value;
};
```

This module also defines several `InputEvent` sub-classes that know more about the different types of events (key, abs, rel etc). The `event_factory` dictionary maps event types to these classes.

Assuming you use the `evdev.util.categorize()` function to categorize events according to their type, adding or replacing a class for a specific event type becomes a matter of modifying `event_factory`.

All classes in this module have reasonable `str()` and `repr()` methods:

```python
>>> print(event)
    event at 1337197425.477827, code 04, type 04, val 458792
>>> print(repr(event))
    InputEvent(1337197425L, 477827L, 4, 4, 458792L)

>>> print(key_event)
    key event at 1337197425.477835, 28 (KEY_ENTER), up
>>> print(repr(key_event))
    KeyEvent(InputEvent(1337197425L, 477835L, 1, 28, 0L))
```

```python

class evdev.events.InputEvent (sec, usec, type, code, value)
    A generic input event.
```
**evdev documentation, Release 1.3.0**

```python
sec
   Time in seconds since epoch at which event occurred.
usec
   Microsecond portion of the timestamp.
type
   Event type - one of `evdev.EV_*`.
code
   Event code related to the event type.
value
   Event value related to the event type.
timestamp()
   Return event timestamp as a float.
```

```python
class evdev.events.KeyEvent(event, allow_unknown=False)
   An event generated by a keyboard, button or other key-like devices.
   key_up = 0
   key_down = 1
   key_hold = 2
   scancode
   keystate
   keycode
   event
      Reference to an `InputEvent` instance.
```

```python
class evdev.events.RelEvent(event)
   A relative axis event (e.g moving the mouse 5 units to the left).
   event
      Reference to an `InputEvent` instance.
```

```python
class evdev.events.AbsEvent(event)
   An absolute axis event (e.g the coordinates of a tap on a touchscreen).
   event
      Reference to an `InputEvent` instance.
```

```python
class evdev.events.SynEvent(event)
   A synchronization event. Synchronization events are used as markers to separate event. Used as markers to separate events. Events may be separated in time or in space, such as with the multitouch protocol.
   event
      Reference to an `InputEvent` instance.
```

```python
evdev.events.event_factory = {0: <class 'evdev.events.SynEvent'>, 1: <class 'evdev.events.KeyEvent'>, 2: <class 'evdev.events.RelEvent'>, 3: <class 'evdev.events.AbsEvent'>}
```

### 6.2 eventio

```python
class evdev.eventio.EventIO
   Base class for reading and writing input events.
```
This class is used by InputDevice and UInput.

- **On**, InputDevice it used for reading user-generated events (e.g. key presses, mouse movements) and writing feedback events (e.g. leds, beeps).
- **On**, UInput it used for writing user-generated events (e.g. key presses, mouse movements) and reading feedback events (e.g. leds, beeps).

**fileno()**

Return the file descriptor to the open event device. This makes it possible to pass instances directly to select.select() and asyncore.file_dispatcher.

**read_loop()**

Enter an endless select.select() loop that yields input events.

**read_one()**

Read and return a single input event as an instance of InputEvent.

Return None if there are no pending input events.

**read()**

Read multiple input events from device. Return a generator object that yields InputEvent instances. Raises BlockingIOError if there are no available events at the moment.

**need_write()**

Decorator that raises EvdevError if there is no write access to the input device.

**write_event (event)**

Inject an input event into the input subsystem. Events are queued until a synchronization event is received.

**Parameters**

- **event** (InputEvent) – InputEvent instance or an object with an event attribute (KeyEvent, RelEvent etc).

**Example**

```python
>>> ev = InputEvent(1334414993, 274296, ecodes.EV_KEY, ecodes.KEY_A, 1)
>>> ui.write_event(ev)
```

**write (etype, code, value)**

Inject an input event into the input subsystem. Events are queued until a synchronization event is received.

**Parameters**

- **etype** – event type (e.g. EV_KEY).
- **code** – event code (e.g. KEY_A).
- **value** – event value (e.g. 0 1 2 - depends on event type).

**Example**

```python
>>> ui.write(e.EV_KEY, e.KEY_A, 1) # key A - down
>>> ui.write(e.EV_KEY, e.KEY_A, 0) # key A - up
```

**close()**

__weakref__

list of weak references to the object (if defined)
6.3 eventio_async

class evdev.eventio_async.EventIO

    async_read_one()
    Asyncio coroutine to read and return a single input event as an instance of InputEvent.

    async_read()
    Asyncio coroutine to read multiple input events from device. Return a generator object that yields InputEvent instances.

    async_read_loop()
    Return an iterator that yields input events. This iterator is compatible with the async for syntax.

    close()

6.4 device

class evdev.device.AbsInfo
    Absolute axis information.
    A namedtuple used for storing absolute axis information - corresponds to the input_absinfo struct:

    value
        Latest reported value for the axis.

    min
        Specifies minimum value for the axis.

    max
        Specifies maximum value for the axis.

    fuzz
        Specifies fuzz value that is used to filter noise from the event stream.

    flat
        Values that are within this value will be discarded by joydev interface and reported as 0 instead.

    resolution
        Specifies resolution for the values reported for the axis. Resolution for main axes (ABS_X, ABS_Y, ABS_Z) is reported in units per millimeter (units/mm), resolution for rotational axes (ABS_RX, ABS_RY, ABS_RZ) is reported in units per radian.

Note: The input core does not clamp reported values to the [minimum, maximum] limits, such task is left to userspace.

class evdev.device.KbdInfo
    Keyboard repeat rate.

    repeat
        Keyboard repeat rate in characters per second.

    delay
        Amount of time that a key must be depressed before it will start to repeat (in milliseconds).

class evdev.device.DeviceInfo
bustype
vendor
product
version
class evdev.device.InputDevice(dev)
   A linux input device from which input events can be read.
   __init__(dev)
   Parameters dev(str|bytes|PathLike) – Path to input device

path
   Path to input device.
fd
   A non-blocking file descriptor to the device file.
info
   ADeviceInfo instance.
name
   The name of the event device.
phys
   The physical topology of the device.
uniq
   The unique address of the device.
version
   The evdev protocol version.
ff_effects_count
   The number of force feedback effects the device can keep in its memory.
capabilities(verbose=False, absinfo=True)
   Return the event types that this device supports as a mapping of supported event types to lists of handled
   event codes.

Example

>>> device.capabilities()
{ 1: [272, 273, 274],
  2: [0, 1, 6, 8] }

If verbose is True, event codes and types will be resolved to their names.

{ ('EV_KEY', 1): [('BTN_MOUSE', 272),
                 ('BTN_RIGHT', 273),
                 ('BTN_MIDDLE', 273)],
 ('EV_REL', 2): [('REL_X', 0),
                 ('REL_Y', 1),
                 ('REL_HWHEEL', 6),
                 ('REL_WHEEL', 8)] }

Unknown codes or types will be resolved to '?'.
If `absinfo` is `True`, the list of capabilities will also include absolute axis information in the form of `AbsInfo` instances:

```python
{ 3: [(0, AbsInfo(min=0, max=255, fuzz=0, flat=0)),
    (1, AbsInfo(min=0, max=255, fuzz=0, flat=0)) ]}
```

Combined with `verbose` the above becomes:

```python
{ ('EV_ABS', 3): [( ('ABS_X', 0), AbsInfo(min=0, max=255, fuzz=0, flat=0)),
    ( ('ABS_Y', 1), AbsInfo(min=0, max=255, fuzz=0, flat=0)) ]}
```

### input_props (verbose=False)
Get device properties and quirks.

**Example**

```python
>>> device.input_props()
[0, 5]
```

If `verbose` is `True`, input properties are resolved to their names. Unknown codes are resolved to `'?`:

```python
[['INPUT_PROP_POINTER', 0], ['INPUT_PROP_POINTING_STICK', 5]]
```

### leds (verbose=False)
Return currently set LED keys.

**Example**

```python
>>> device.leds()
[0, 1, 8, 9]
```

If `verbose` is `True`, event codes are resolved to their names. Unknown codes are resolved to `'?`:

```python
[['LED_NUML', 0], ['LED_CAPSL', 1], ['LED_MISC', 8], ['LED_MAIL', 9]]
```

### set_led (led_num, value)
Set the state of the selected LED.

**Example**

```python
>>> device.set_led(ecodes.LED_NUML, 1)
```

### __eq__(other)
Two devices are equal if their `info` attributes are equal.

### __ne__(other)
Return `self!=value`.

### __fspath__()

### close()

### grab()
Grab input device using `EVIOCGRAB` - other applications will be unable to receive events until the device is released. Only one process can hold a `EVIOCGRAB` on a device.
Warning: Grabbing an already grabbed device will raise an IOError.

ungrab()  
Release device if it has been already grabbed (uses EVIOCGRAB).

Warning: Releasing an already released device will raise an IOError('Invalid argument').

grab_context()  
A context manager for the duration of which only the current process will be able to receive events from the device.

upload_effect(effect)  
Upload a force feedback effect to a force feedback device.

erase_effect(ff_id)  
Erase a force effect from a force feedback device. This also stops the effect.

repeat  
Get or set the keyboard repeat rate (in characters per minute) and delay (in milliseconds).

active_keys(VERBOSE=False)  
Return currently active keys.

Example

```python
>>> device.active_keys()
[1, 42]
```

If VERBOSE is True, key codes are resolved to their verbose names. Unknown codes are resolved to '?'. For example:

```python
[('KEY_ESC', 1), ('KEY_LEFTSHIFT', 42)]
```

fn

absinfo(axis_num)  
Return current AbsInfo for input device axis

Parameters axis_num(int) – EV_ABS keycode (example ecodes.ABS_X)

Example

```python
>>> device.absinfo(ecodes.ABS_X)
AbsInfo(value=1501, min=-32768, max=32767, fuzz=0, flat=128, resolution=0)
```

__hash__ = None

set_absinfo(axis_num, value=None, min=None, max=None, fuzz=None, flat=None, resolution=None)  
Update AbsInfo values. Only specified values will be overwritten.

Parameters axis_num(int) – EV_ABS keycode (example ecodes.ABS_X)
Example

```py
>>> device.set_absinfo(ecodes.ABS_X, min=-2000, max=2000)
```

You can also unpack AbsInfo tuple that will overwrite all values

```py
>>> device.set_absinfo(ecodes.ABS_Y, *AbsInfo(0, -2000, 2000, 0, 15, 0))
```

6.5 uinput

class evdev.uinput.UInput:

```py
class evdev.uinput.UInput(events=None, name='py-evdev-uinput', vendor=1, product=1, version=1, bustype=3, devnode='/dev/uinput', phys='py-evdev-uinput', input_props=None)
```

A userland input device and that can inject input events into the linux input subsystem.

classmethod from_device(*devices, **kwargs):

Create an UInput device with the capabilities of one or more input devices.

Parameters:

- `devices (InputDevice|str)` – Varargs of InputDevice instances or paths to input devices.
- `filtered_types (Tuple[event type codes])` – Event types to exclude from the capabilities of the uinput device.
- `**kwargs` – Keyword arguments to UInput constructor (i.e. name, vendor etc.).

__init__:

```py
__init__ (events=None, name='py-evdev-uinput', vendor=1, product=1, version=1, bustype=3, devnode='/dev/uinput', phys='py-evdev-uinput', input_props=None)
```

Parameters:

- `events (dict)` – Dictionary of event types mapping to lists of event codes. The event types and codes that the uinput device will be able to inject - defaults to all key codes.
- `name` – The name of the input device.
- `vendor` – Vendor identifier.
- `product` – Product identifier.
- `bustype` – Bustype identifier.
- `phys` – Physical path.
- `input_props` – Input properties and quirks.

**Note:** If you do not specify any events, the uinput device will be able to inject only KEY_* and BTN_* event codes.

name

Uinput device name.

vendor

Device vendor identifier.
product
   Device product identifier.

version
   Device version identifier.

bustype
   Device bustype - e.g. BUS_USB.

phys = None
   Uinput device physical path.

devnode
   Uinput device node - e.g. /dev/uinput/.

fd
   Write-only, non-blocking file descriptor to the uinput device node.

device
   An InputDevice instance for the fake input device. None if the device cannot be opened for reading and writing.

syn()
   Inject a SYN_REPORT event into the input subsystem. Events queued by write() will be fired. If possible, events will be merged into an ‘atomic’ event.

capabilities (verbose=False, absinfo=True)
   See capabilities.

6.6 util

evdev.util.list_devices (input_device_dir=’/dev/input’)
   List readable character devices in input_device_dir.

evdev.util.is_device (fn)
   Check if fn is a readable and writable character device.

evdev.util.categorize (event)
   Categorize an event according to its type.

   The event_factory dictionary maps event types to sub-classes of InputEvent. If the event cannot be categorized, it is returned unmodified.

evdev.util.resolve_ecodes_dict (typecodemap, unknown=’?’)
   Resolve event codes and types to their verbose names.

   Parameters

   • typecodemap – mapping of event types to lists of event codes.

   • unknown – symbol to which unknown types or codes will be resolved.

Example

```python
>>> resolve_ecodes_dict({ 1: [272, 273, 274] })
{ ('EV_KEY', 1): [('BTN_MOUSE', 272),
    ('BTN_RIGHT', 273),
    ('BTN_MIDDLE', 274)] }
```
If `typecodemap` contains absolute axis info (instances of `AbsInfo`) the result would look like:

```python
>>> resolve_ecodes_dict([{ 3: [(0, AbsInfo(...))] }],
                          { ('EV_ABS', 3L): [(('ABS_X', 0L), AbsInfo(...))] })
```

`evdev.util.resolve_ecodes(ecode_dict, ecode_list, unknown='?')`

Resolve event codes and types to their verbose names.

**Example**

```python
>>> resolve_ecodes([272, 273, 274])
[('BTN_MOUSE', 272), ('BTN_RIGHT', 273), ('BTN_MIDDLE', 274)]
```

### 6.7 ecodes

This module exposes the integer constants defined in `linux/input.h` and `linux/input-event-codes.h`.

Exposed constants:

```python
KEY, ABS, REL, SW, MSC, LED, BTN, REP, SND, ID, EV,
BUS, SYN, FF, FF_STATUS, INPUT_PROP
```

This module also provides reverse and forward mappings of the names and values of the above mentioned constants:

```python
>>> evdev.ecodes.KEY_A
30

>>> evdev.ecodes.ecodes['KEY_A']
30

>>> evdev.ecodes.KEY[30]
'KEY_A'

>>> evdev.ecodes.REL[0]
'REL_X'

>>> evdev.ecodes.EV[evdev.ecodes.EV_KEY]
'EV_KEY'

>>> evdev.ecodes.bytype[evdev.ecodes.EV_REL][0]
'REL_X'
```

Keep in mind that values in reverse mappings may point to one or more event codes. For example:

```python
>>> evdev.ecodes.FF[80]
['FF_EFFECT_MIN', 'FF_RUMBLE']

>>> evdev.ecodes.FF[81]
'FF_PERIODIC'
```

`evdev.ecodes.keys {0: 'KEY_RESERVED', 1: 'KEY_ESC', 2: 'KEY_1', ...} (`

Keys are a combination of all BTN and KEY codes.

`evdev.ecodes.ecodes {'KEY_END': 107, 'FF_RUMBLE': 80, 'KEY_KPDOT': 83, 'KEY_CNT': 768, ...} `

Mapping of names to values.
evdev.ecodes.bytype { 0: 'SYN_REPORT', 1: 'SYN_CONFIG', 2: 'SYN_MT_REPORT', 3: 'SYN_USBDATA', ...}

Mapping of event types to other value/name mappings.
Python-evdev exposes most of the more common interfaces defined in the evdev subsystem. Reading and injecting events is well supported and has been tested with nearly all event types.

The basic functionality for reading and uploading force-feedback events is there, but it has not been exercised sufficiently. A major shortcoming of the uinput wrapper is that it does not support force-feedback devices at all (see issue #23).

Some characters, such as `:` (colon), cannot be easily injected (see issue #7). Translating them into UInput events would require knowing the kernel keyboard translation table, which is beyond the scope of python-evdev. Please look into the following projects if you need more complete or convenient input injection support.

- python-uinput
- uinput-mapper
- PyUserInput (cross-platform, works on the display server level)
- pygame (cross-platform)
8.1 1.3.0 (Jan 12, 2020)

- Fix build on 32bit arches with 64bit time_t
- Add functionality to query device properties. See `InputDevice.input_props` and the `input_props` argument to `Uinput`.
- `KeyEvent` received an `allow_unknown` constructor argument, which determines what will happen when an event code cannot be mapped to a keycode. The default and behavior so far has been to raise `KeyError`. If set to `True`, the keycode will be set to the event code formatted as a hex number.
- Add `InputDevice.set_absinfo()` and `InputDevice.absinfo()`.
- Instruct the asyncio event loop to stop monitoring the fd of the input device when the device is closed.

8.2 1.2.0 (Apr 7, 2019)

- Add UInput support for the resolution parameter in `AbsInfo`. This brings support for the new method of uinput device setup, which was introduced in Linux 4.5 (thanks to @LinusCDE).
- Vendor and product identifiers can be greater or equal to 0x8000 (thanks @ivaradi).

8.3 1.1.2 (Sep 1, 2018)

- Fix installation on kernels <= 4.4.
- Fix uinput creation ignoring absinfo settings.
8.4 1.1.0 (Aug 27, 2018)

- Add support for handling force-feedback effect uploads (many thanks to @ndreys).
- Fix typo preventing ff effects that need left coefficients from working.

8.5 1.0.0 (Jun 02, 2018)

- Prevent Uinput device creation raising Objects/longobject.c:415: bad argument to internal function when a non-complete AbsInfo structure is passed. All missing AbsInfo fields are set to 0.
- Fix Uinput device creation raising KeyError when a capability filtered by default is not present.
- The InputDevice.fn attribute was deprecated in favor of InputDevice.path. Using the former will show a DeprecationWarning, but would otherwise continue working as before.
- Fix InputDevice comparison raising AttributeError due to a non-existant path attribute.
- Fix asyncio support in Python 3.5+.
- Uploading FF effect now works both on Python 2.7 and Python 3+.
- Remove the asyncore example from the tutorial.

8.6 0.8.1 (Mar 24, 2018)

- Fix Python 2 compatibility issue in with Uinput.from_device.
- Fix minor evdev.evtest formatting issue.

8.7 0.8.0 (Mar 22, 2018)

- Fix InputDevice comparison on Python 2.
- The device path is now considered when comparing two devices.
- Fix UInput.from_device not correctly merging the capabilities of selected devices.
- The list of excluded event types in UInput.from_device is now configurable. For example:

  ```
  UInput.from_device(dev, filtered_types=(EV_SYN, EV_FF))
  ```

  In addition, ecodes.EV_FF is now excluded by default.
- Add a context manager for grabbing access to a device - InputDevice.grab_context. For example:

  ```
  with dev.grab_context():
  
  pass
  ```

- Add the InputDeviceuniq attribute, which contains the unique identifier of the device. As with phys, this attribute may be empty (i.e. ")". 
8.8 0.7.0 (Jun 16, 2017)

- `InputDevice` now accepts objects that support the path protocol. For example:

```
pth = pathlib.Path('/dev/input/event0')
dev = evdev.InputDevice(pth)
```

- Support path protocol in `InputDevice`. This means that `InputDevice` instances can be passed to callers that expect a `os.PathLike` object.

- Exceptions raised during `InputDevice.async_read()` (and similar) are now handled properly (i.e. an exception is set on the returned future instead of leaking that exception into the event loop) (Fixes #67).

8.9 0.6.4 (Oct 07, 2016)

- Exclude `ecodes.c` from source distribution (Fixes #63).

8.10 0.6.3 (Oct 06, 2016)

- Add the `UInput.from_device` class method, which allows uinput device to be created with the capabilities of one or more existing input devices:

```
ui = UInput.from_device('/dev/input1', '/dev/input2', **constructor_kwargs)
```

- Add the `build_ecodes` distutils command, which generates the `ecodes.c` extension module. The new way of overwriting the evdev header locations is:

```
python setup.py build \
   build_ecodes --evdev-headers path/input.h:path/input-event-codes.h \
   build_ext --include-dirs path/ \
   install
```

The `build*` and `install` commands no longer have to be part of the same command-line (i.e. running `install` will reuse the outputs of the last `build`).

8.11 0.6.1 (Jun 04, 2016)

- Disable tty echoing while evtest is running.

- Allow evtest to listen to more than one devices.

- The `setup.py` script now allows the location of the input header files to be overwritten. For example:

```
python setup.py build_ext \
   --evdev-headers path/input.h:path/input-event-codes.h \
   --include-dirs path/ \
   install
```
8.12 0.6.0 (Feb 14, 2016)

• Asyncio and async/await support (many thanks to @paulo-raca).
• Add the ability to set the phys property of uinput devices (thanks @paulo-raca).
• Add a generic InputDevice.set() method (thanks @paulo-raca).
• Distribute the evtest script along with evdev.
• Fix issue with generating ecodes.c in recent kernels (>= 4.4.0).
• Fix absinfo item indexes in UIinput.uinput_create() (thanks @forsenonlhaimaisentito).
• More robust comparison of InputDevice objects (thanks @isia).

8.13 0.5.0 (Jun 16, 2015)

• Write access to the input device is no longer mandatory. Evdev will first try to open the device for reading and writing and fallback to read-only. Methods that require write access (e.g. set_led()) will raise EvdevError if the device is open only for reading.

8.14 0.4.7 (Oct 07, 2014)

• Fallback to distutils if setuptools is not available.

8.15 0.4.6 (Oct 07, 2014)

• Rework documentation and docstrings once more.
• Fix install on Python 3.4 (works around issue21121).
• Fix ioctl() requested buffer size (thanks Jakub Wojciech Klama).

8.16 0.4.5 (Jul 06, 2014)

• Add method for returning a list of the currently active keys - InputDevice.active_keys() (thanks @spasche).
• Fix a potential buffer overflow in ioctl_capabilities() (thanks @spasche).

8.17 0.4.4 (Jun 04, 2014)

• Calling InputDevice.read_one() should always return None, when there is nothing to be read, even in case of a EAGAIN errno (thanks JPP).
8.18 0.4.3 (Dec 19, 2013)

- Silence **OSError** in destructor (thanks @polyphemus).
- Make `InputDevice.close()` work in cases in which stdin (fd 0) has been closed (thanks @polyphemus).

8.19 0.4.2 (Dec 13, 2013)

- Rework documentation and docstrings.
- Call `InputDevice.close()` from `InputDevice.__del__()`.

8.20 0.4.1 (Jul 24, 2013)

- Fix reference counting in `InputDevice.device_read()`, `InputDevice.device_read_many()` and `ioctl_capabilities()`.

8.21 0.4.0 (Jul 01, 2013)

- Add **FF_*** and **FF_STATUS** codes to `ecodes()` (thanks @bgilbert).
- Reverse event code mappings (**ecodes.{KEY,FF,REL,ABS}** and etc.) will now map to a list of codes, whenever a value corresponds to multiple codes:

```
>>> ecodes.KEY[152]
... ['KEY_COFFEE', 'KEY_SCREENLOCK']
>>> ecodes.KEY[30]
... 'KEY_A'
```

- Set the state of a LED through `InputDevice.set_led()` (thanks @accek).
- Open `InputDevice.fd` in **O_RDWR** mode from now on.
- Fix segfault in `InputDevice.device_read_many()` (thanks @bgilbert).

8.22 0.3.3 (May 29, 2013)

- Raise **IOError** from `InputDevice.device_read()` and `InputDevice.device_read_many()` when `InputDevice.read()` fails.
- Several stability and style changes (thank you debian code reviewers).

8.23 0.3.2 (Apr 05, 2013)

- Fix vendor id and product id order in `DeviceInfo()` (thanks @kived).
8.24 0.3.1 (Nov 23, 2012)

- `InputDevice.read()` will return an empty tuple if the device has nothing to offer (instead of segfaulting).
- Exclude unnecessary package data in sdist and bdist.

8.25 0.3.0 (Nov 06, 2012)

- Add ability to set/get auto-repeat settings with `EVIOC{SG}REP`.
- Add `InputDevice.version()` - the value of `EVIOCGVERSION`.
- Add `InputDevice.read_loop()`.
- Add `InputDevice.grab()` and `InputDevice.ungrab()` - exposes `EVIOCGRAB`.
- Add `InputDevice.leds()` - exposes `EVIOCGLED`.
- Replace `DeviceInfo` class with a namedtuple.
- Prevent `InputDevice.read_one()` from skipping events.
- Rename `AbsData` to `AbsInfo` (as in `struct input_absinfo`).

8.26 0.2.0 (Aug 22, 2012)

- Add the ability to set arbitrary device capabilities on uinput devices (defaults to all `EV_KEY` ecodes).
- Add `UInput.device` which is an open `InputDevice` to the input device that uinput ‘spawns’.
- Add `UInput.capabilities()` which is just a shortcut to `UInput.device.capabilities()`.
- Rename `UInput.write()` to `UInput.write_event()`.
- Add a simpler `UInput.write(type, code, value)()` method.
- Make all `UInput()` constructor arguments optional (default device name is now py-evdev-uinput).
- Add the ability to set `absmin`, `absmax`, `absfuzz` and `absflat` when specifying the uinput device’s capabilities.
- Remove the `nophys` argument - if a device fails the `EVIOCGPHYS` ioctl, phys will equal the empty string.
- Make `InputDevice.capabilities()` perform a `EVIOCGABS` ioctl for devices that support `EV_ABS` and return that info wrapped in an `AbsData` namedtuple.
- Split `ioctl_devinfo` into `ioctl_devinfo` and `ioctl_capabilities`.
- Split `UInput.uinput_open()` to `UInput.uinput_open()` and `UInput.uinput_create()`.
- Add more uinput usage examples and documentation.
- Rewrite uinput tests.
- Remove `mouserel` and `mouseabs` from `UInput`.
- Tie the sphinx version and release to the distutils version.
- Set ‘methods-before-attributes’ sorting in the docs.
- Remove `KEY_CNT` and `KEY_MAX` from `ecodes.keys()`.
8.27 0.1.1 (May 18, 2012)

- Add `events.keys`, which is a combination of all `BTN_` and `KEY_` event codes.
- `ecodes.c` was not generated when installing through `pip`.

8.28 0.1.0 (May 17, 2012)

*Initial Release*
This package is released under the terms of the Revised BSD License.
evdev.device, 22
evdev.ecodes, 28
evdev.eventio, 20
evdev.eventio_async, 22
evdev.events, 19
evdev.uinput, 26
evdev.util, 27
Symbols
__eq__() (evdev.device.InputDevice method), 24
__fspath__() (evdev.device.InputDevice method), 24
__hash__() (evdev.device.InputDevice attribute), 25
__init__() (evdev.device.InputDevice method), 23
__init__() (evdev.uinput.UInput method), 26
__ne__() (evdev.device.InputDevice method), 24
__weakref__ (evdev.eventio.EventIO attribute), 21

A
AbsEvent (class in evdev.events), 20
AbsInfo (class in evdev.device), 22
absinfo() (evdev.device.InputDevice method), 25
active_keys() (evdev.device.InputDevice method), 25
async_read() (evdev.eventio.AsyncEventIO method), 22
async_read_loop() (evdev.eventio.AsyncEventIO method), 22
async_read_one() (evdev.eventio.AsyncEventIO method), 22

B
bustype (evdev.device.DeviceInfo attribute), 22
bustype (evdev.uinput.UInput attribute), 27
bytype (in module evdev.ecodes), 28

capabilities() (evdev.device.InputDevice method), 23
capabilities() (evdev.uinput.UInput method), 27
categorize() (in module evdev.util), 27
close() (evdev.device.InputDevice method), 24
close() (evdev.uinput.AsyncEventIO method), 21
close() (evdev.uinput.UInput method), 27
code (evdev.events.InputEvent attribute), 20
code (evdev.uinput.UInput attribute), 27
device (evdev.uinput.UInput attribute), 27
DeviceInfo (class in evdev.device), 22
devnode (evdev.uinput.UInput attribute), 27

e
ecodes (in module evdev.ecodes), 28
erase_effect() (evdev.device.InputDevice method), 25
evdev.device (module), 22
evdev.ecodes (module), 28
evdev.eventio (module), 20
evdev.eventio_async (module), 22
evdev.events (module), 19
evdev.uinput (module), 26
evdev.util (module), 27
event (evdev.events.AbsEvent attribute), 20
event (evdev.events.KeyEvent attribute), 20
event (evdev.events.RelEvent attribute), 20
event (evdev.events.SynEvent attribute), 20
event_factory (in module evdev.events), 20
EventIO (class in evdev.eventio), 20
EventIO (class in evdev.eventio_async), 22

F
fd (evdev.device.InputDevice attribute), 23
fd (evdev.uinput.UInput attribute), 27
ff_effects_count (evdev.device.InputDevice attribute), 23
fileno() (evdev.eventio.EventIO method), 21
flat (evdev.device.AbsInfo attribute), 22
fn (evdev.device.InputDevice attribute), 22
from_device() (evdev.uinput.UInput class method), 26
fuzz (evdev.device.AbsInfo attribute), 22

G
grab() (evdev.device.InputDevice method), 24
grab_context() (evdev.device.InputDevice method), 25
### Index

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>info (evdev.device.InputDevice attribute), 23</td>
</tr>
<tr>
<td></td>
<td>input_props() (evdev.device.InputDevice method), 24</td>
</tr>
<tr>
<td></td>
<td>InputDevice (class in evdev.device), 23</td>
</tr>
<tr>
<td></td>
<td>InputEvent (class in evdev.events), 19</td>
</tr>
<tr>
<td></td>
<td>is_device() (in module evdev.util), 27</td>
</tr>
<tr>
<td>K</td>
<td>KbdInfo (class in evdev.device), 22</td>
</tr>
<tr>
<td></td>
<td>key_down (evdev.events.KeyEvent attribute), 20</td>
</tr>
<tr>
<td></td>
<td>key_hold (evdev.events.KeyEvent attribute), 20</td>
</tr>
<tr>
<td></td>
<td>key_up (evdev.events.KeyEvent attribute), 20</td>
</tr>
<tr>
<td></td>
<td>keycode (evdev.events.KeyEvent attribute), 20</td>
</tr>
<tr>
<td></td>
<td>KeyEvent (class in evdev.events), 20</td>
</tr>
<tr>
<td></td>
<td>keys (in module evdev.ecodes), 28</td>
</tr>
<tr>
<td></td>
<td>keystate (evdev.events.KeyEvent attribute), 20</td>
</tr>
<tr>
<td>L</td>
<td>leds() (evdev.device.InputDevice method), 24</td>
</tr>
<tr>
<td></td>
<td>list_devices() (in module evdev.util), 27</td>
</tr>
<tr>
<td>M</td>
<td>max (evdev.device.AbsInfo attribute), 22</td>
</tr>
<tr>
<td></td>
<td>min (evdev.device.AbsInfo attribute), 22</td>
</tr>
<tr>
<td>N</td>
<td>name (evdev.device.InputDevice attribute), 23</td>
</tr>
<tr>
<td></td>
<td>name (evdev.uinput.UInput attribute), 26</td>
</tr>
<tr>
<td></td>
<td>need_write() (evdev.eventio.EventIO method), 21</td>
</tr>
<tr>
<td>P</td>
<td>path (evdev.device.InputDevice attribute), 23</td>
</tr>
<tr>
<td></td>
<td>phys (evdev.device.InputDevice attribute), 23</td>
</tr>
<tr>
<td></td>
<td>phys (evdev.uinput.UInput attribute), 27</td>
</tr>
<tr>
<td></td>
<td>product (evdev.device.DeviceInfo attribute), 23</td>
</tr>
<tr>
<td></td>
<td>product (evdev.uinput.UInput attribute), 26</td>
</tr>
<tr>
<td>R</td>
<td>read() (evdev.eventio.EventIO method), 21</td>
</tr>
<tr>
<td></td>
<td>read_loop() (evdev.eventio.EventIO method), 21</td>
</tr>
<tr>
<td></td>
<td>read_one() (evdev.eventio.EventIO method), 21</td>
</tr>
<tr>
<td></td>
<td>RelEvent (class in evdev.events), 20</td>
</tr>
<tr>
<td></td>
<td>repeat (evdev.device.InputDevice attribute), 25</td>
</tr>
<tr>
<td></td>
<td>repeat (evdev.device.KbdInfo attribute), 22</td>
</tr>
<tr>
<td></td>
<td>resolution (evdev.device.AbsInfo attribute), 22</td>
</tr>
<tr>
<td></td>
<td>resolve_ecodes() (in module evdev.util), 28</td>
</tr>
<tr>
<td></td>
<td>resolve_ecodes_dict() (in module evdev.util), 27</td>
</tr>
<tr>
<td>S</td>
<td>scancode (evdev.events.KeyEvent attribute), 20</td>
</tr>
<tr>
<td></td>
<td>sec (evdev.events.InputEvent attribute), 19</td>
</tr>
<tr>
<td></td>
<td>set_absinfo() (evdev.device.InputDevice method), 25</td>
</tr>
<tr>
<td></td>
<td>set_led() (evdev.device.InputDevice method), 24</td>
</tr>
<tr>
<td></td>
<td>syn() (evdev.uinput.UInput method), 27</td>
</tr>
<tr>
<td></td>
<td>SynEvent (class in evdev.events), 20</td>
</tr>
<tr>
<td>T</td>
<td>timestamp() (evdev.events.InputEvent method), 20</td>
</tr>
<tr>
<td></td>
<td>type (evdev.events.InputEvent attribute), 20</td>
</tr>
<tr>
<td>U</td>
<td>UInput (class in evdev.uinput), 26</td>
</tr>
<tr>
<td></td>
<td>ungrab() (evdev.device.InputDevice method), 25</td>
</tr>
<tr>
<td></td>
<td>uniq (evdev.device.InputDevice attribute), 23</td>
</tr>
<tr>
<td></td>
<td>upload_effect() (evdev.device.InputDevice method), 25</td>
</tr>
<tr>
<td></td>
<td>usec (evdev.events.InputEvent attribute), 20</td>
</tr>
<tr>
<td>V</td>
<td>value (evdev.device.AbsInfo attribute), 22</td>
</tr>
<tr>
<td></td>
<td>value (evdev.events.InputEvent attribute), 20</td>
</tr>
<tr>
<td></td>
<td>vendor (evdev.device.DeviceInfo attribute), 23</td>
</tr>
<tr>
<td></td>
<td>vendor (evdev.uinput.UInput attribute), 26</td>
</tr>
<tr>
<td></td>
<td>version (evdev.device.DeviceInfo attribute), 23</td>
</tr>
<tr>
<td></td>
<td>version (evdev.device.InputDevice attribute), 23</td>
</tr>
<tr>
<td></td>
<td>version (evdev.uinput.UInput attribute), 27</td>
</tr>
<tr>
<td>W</td>
<td>write() (evdev.eventio.EventIO method), 21</td>
</tr>
<tr>
<td></td>
<td>write_event() (evdev.eventio.EventIO method), 21</td>
</tr>
</tbody>
</table>