API Documentation

API Documentation

March 8, 2015

Contents

1 Module PowerMateEventHandler

1.1 Functions

find_device(*dev_dir='/dev/input/'*)

Finds and returns the device in dev_dir

If the user does not have permission to access a device in dev_dir, an OSError Exception will be raised.

OSErrors are printed to stderr. These will likely happen if the user does not have permission to all devices. If the function retrns None with the device plugged in, check the permissions on the device. (There's probably a better way to do this - check the devices before attempting to open them - but that will have to wait for the moment.)

Return Value

An evdev.InputDevice. None if the device is not found.

(type=evdev.InputDevice or None)

event_time_in_ms(*event*)

Parameters

`event:` the event to get the time for

(type=evdev.InputEvent)

Return Value

The time in ms the event occurred (as an int)

(type=int)

Note: Does this by converting the event microseconds (event.usec) to seconds (multiply by 1000000), adding the event seconds (event.sec), converting to ms (multiply by 1000), then casting to an int.

get_uinput(*dev*)

Parameters

`dev:` An evdev.InputDevice for the PowerMate (see find_device)

(type=evdev.InputDevice:)

Return Value

An evdev.UInput for the device. This can be used to write to the device (to change the led brightness).

(type=evdev.UInput)

1.2 Variables

Name	Description
BUTTON_PUSHED	**Value:** `256`
KNOB_TURNED	**Value:** `7`
POSITIVE	**Value:** `1`

continued on next page

Name	Description
NEGATIVE	**Value:** `-1`
time_down	**Value:** `0`
led_brightness	**Value:** `100`
flash_duration	**Value:** `0.15`
___package___	**Value:** `None`

1.3 Class ConsolidatedEventCode

object
enum.Enum
PowerMateEventHandler.ConsolidatedEventCode

SINGLE_CLICK = 0 DOUBLE_CLICK = SINGLE_CLICK + 1 LONG_CLICK = DOUBLE_CLICK + 1 RIGHT_TURN = LONG_CLICK + 1 LEFT_TURN = RIGHT_TURN + 1

1.3.1 Methods

___format___(*self*, *format_spec*)

default object formatter

Overrides: object.___format___ extit(inherited documentation)

___reduce_ex___(*self*, *proto*)

helper for pickle

Overrides: object.___reduce_ex___ extit(inherited documentation)

___repr___(*self*)

repr(x)

Overrides: object.___repr___ extit(inherited documentation)

___str___(*self*)

str(x)

Overrides: object.___str___ extit(inherited documentation)

Inherited from enum.Enum

___dir___(), ___eq___(), ___ge___(), ___gt___(), ___hash___(), ___le___(), ___lt___(), ___ne___(), ___new___()

Inherited from object(Section ??)

___delattr___(), ___getattribute___(), ___init___(), ___reduce___(), ___setattr___(), ___sizeof___(), ___subclasshook___()

1.3.2 Properties

Name	Description
Inherited from object (Section **??***)* ___class___	

1.3.3 Class Variables

Name	Description
SINGLE_CLICK	**Value:** `0`
DOUBLE_CLICK	**Value:** `SINGLE_CLICK+ 1`
LONG_CLICK	**Value:** `DOUBLE_CLICK+ 1`
RIGHT_TURN	**Value:** `LONG_CLICK+ 1`
LEFT_TURN	**Value:** `RIGHT_TURN+ 1`
_member_map_	**Value:** `OrderedDict([('SINGLE_CLICK', <ConsolidatedEventCode.SING...`
_member_names_	**Value:** `['SINGLE_CLICK', 'DOUBLE_CLICK', 'LONG_CLICK', 'RIGHT_TUR...`
_value2member_map_	**Value:** `{0: <ConsolidatedEventCode.SINGLE_CLICK: 0>, 1: <Consolid...`
Inherited from enum.Enum name, value	

1.4 Class PowerMateEventHandler

1.4.1 Methods

___init___(*self*, *brightness*=255, *read_delay*=None, *turn_delay*=0, *long_press_time*=0.5, *double_click_time*=0.3, *dev_dir*='/dev/input/')

Find the PowerMateDevice, and get set up to start reading from it and writing to it.

If the device is not found (can happen if the device is not plugged in, or the user does not have permissions to it) a DeviceNotFound Exception will be raised.

Parameters

`brightness`: The inital brightness of the led in the base.

(type=int)

`read_delay`: Timeout when waiting for the device to be readable. Having a time out allows the threads to be joinable without waiting for another event. None (default) means to wait indefinitely for the device to be readable. This will probably yield the best performance, but means the thread will not stop after a call to stop() until a new event is triggered.

Having this configurable was intendted to allow the reading of events to be stoppable (i.e to keep from blocking the thread indefinitely). It was made tunable to allow good performance on fast CPUs, but not hog resources on slower machines.

Setting delay to None will cause the thread to block indefinitely.

(type=double)

`turn_delay`: Time in ms between consolidated turns.

(type=double)

`long_press_time`: time (in s) the button must be held to register a long press

(type=double)

`double_click_time`: time (in s) the button must be pressed again after a single press to register as a double

(type=double)

`dev_dir`: The directory in which to look for the device.

(type=str)

set_led_brightness(*self*, *brightness*)

Sets the led in the base to the specified brightness. The valid range is 0-255, where 0 is off. Anything less than 0 will be treated as zero, anything greater than 255 will be treated as 255.

flash_led(*self*, *num_flashes*=`2`, *brightness*=`100`, *duration*=`0.15`, *sleep*=`0.15`)

Convenience function to flash the led in the base. After the flashes, the brightness will be reset to whatever it was when this function was called.

Parameters

`num_flashes`: number times to flash

(type=int)

`brightness`: the brightness of the flashes (range defined by set_led_brightness)

(type=int)

`duration`: length of each flash in seconds (decimals accepted)

(type=double)

`sleep`: time between each flash in seconds (decimals accepted)

(type=double)

start(*self*, *raw_only*=`False`)

Begin capturing/queueing events. Once this has been run, get_next() can be used to start pulling events off the queue.

stop(*self*)

Stop the capture/queuing of events.

get_next(*self*, *block*=`True`, *timeout*=`None`)

Pull the next consolidated event off the queue, and return it.

Parameters

`block:` block until next is available

(type=bool)

`timeout:` block for this long

(type=double)

Return Value

If start was run with rawOnly=True, an evdev.events.InputEvent; Otherwise, a ConsolidatedEventCode. In either case, None if there is not an event ready and block is False, or timeout is reached.

(type=evdev.events.InputEvent, ConsolidatedEventCode, or None)

Note: block and timeout are passed directly to queue.get(). If block is TRUE, the thread will block for timeout seconds for the next event. If timeout is None, it will wait indefinitely. If block is False, an event will be grabbed only if one is ready immediately.

set_turn_delay(*self*, *delay*)

Set the delay between when consolidated events will be registered.

In an effort to reduce spam from a failry sensative device, this variable was created. If multiple turn events come in, the first will register a consolidated event, and those that come in within the delay time will be ignored. Once the delay threshold has been reached, another consolidated event will be registered.

Parameters

`delay:` time in ms between turn events.

(type=double)

set_read_delay(*self*, *delay*)

This was intendted to allow the reading of events to be stoppable (i.e to keep from blocking the thread indefinitely). It was made tunable to allow good performance on fast CPUs, but not hog resources on slower machines.

Setting delay to None will cause the thread to block indefinitely. This will probably yield the best performance, but means the thread will not stop after a call to stop() until a new event is triggered.

Parameters

`delay`: Time in seconds to wait for the device to be readable.

(type=double)

set_double_click_time(*self*, *time*)

Parameters

`time`: (in s) the button must be pressed again after a single press to register as a double

(type=double)

set_long_click_time(*self*, *time*)

Parameters

`time`: (in s) the button must be held to register a long press

(type=double)

Index