macpy Documentation

Release 0.1.0b

Tomas Ravinskas

Feb 07, 2022

Contents:

This package provides easy keyboard/pointer/window management macro creation and GUI automation for python versions 2.7 and 3.4+. Currently it works on Windows and Linux (both under X and with limited functionallity under Wayland). Among it's features are:

- Low level hooks for keyboard, pointer events
- A hook for window creation, destruction and focus change
- Support for registering hotkeys and hotstrings
- Simulating keyboard/pointer events
- Providing platform independent definition/mapping of keys/buttons
- Listing open windows
- Managing open windows
- And more!

Note: Window management functionallity is not available under Wayland.

More, keyboard and pointer functions require root access under Wayland.

CHAPTER 1

Interfaces

macpy.**record**(*record_type*, *stop_key=None*, *timer=None*)
Record events of record_type and return a list.

Parameters

- **record_type** (RecordType) – The type of events to record.
- **stop_key** (Key) – The key or button which will end the recording. If stop_key is None the recording will go on until timer runs out.
- **timer** (*float*) – The duration of recording session. If timer is None the session will go on until specified key is pressed.

Returns A list of recorded events.

Return type [*Event*]

macpy.**replay**(*event_list*, *delay=0*)
Replay events from a sequence.

Parameters

- **event_list** (*[Event]*) – A sequence of events.
- **delay** (*float*) – The seconds to wait between each event (or pair).

1.1 Keyboard

class macpy.**Keyboard**
Keyboard interface object.

Allows simulating keyboard input as well as reading data from connected physical keyboards.

close()
Close opened resources and cleanly exit mainloop.

Call this method when you are done with this object.

get_key_state(*key*)
Check whether the key is pressed or released.

Parameters **key** (Key) – The key to check.

Returns Current state of the key.

Return type *KeyState*

install_keyboard_hook(*callback*, *grab=False*)
Installs a low level hook that sends all keyboard input to the callback.

Callback must take a single event argument.

For event definition see *KeyboardEvent*.

Parameters

- **callback** (*Callable*) – A callable which receives events.
- **grab** (*bool*) – If grab is True events are consumed and not passed through to other applications. .. note:

```
Even if grab is :obj:`True`, synthetic events are still
allowed on Windows.

Under wayland this option does nothing.
```

uninstall_keyboard_hook()
Uninstall keyboard hook and stop hook's loop.

You don't have to explicitly call this method, calling *close()* automatically removes hook if it's installed.

init_hotkeys()
Initialize hotkey loop.

You need to call this method once before using any hotkey related methods.

uninit_hotkeys()
Deinitialize hotkey loop.

You don't have to explicitly call this method, calling *close()* automatically stop hotkey loop if it's started.

register_hotkey(*key*, *modifiers*, *callback*)
Register a key combination that once pressed triggers callback.

Note: It is currently not possible to define hotkeys which trigger only with e.g. KEY_LEFTSHIFT. Left/right keys are automatically converted to generic modifier e.g. KEY_SHIFT.

Parameters

- **key** (Key) – The key which triggers callback.
- **modifiers** (*[Key, ...]*) – Iterable of modifier keys that also need to be pressed. Valid modifiers are KEY_SHIFT, KEY_CTRL, KEY_ALT and KEY_META.
- **callback** (*Callable*) – Callable which will be called with *HotKey* object as a single argument.

Returns A hotkey object.

Return type *HotKey*

unregister_hotkey(*hotkey*)
Unegister a previously registered hotkey.

Parameters **hotkey** (HotKey) – A hotkey object as returned by e.g. *register_hotkey()*.

register_hotstring(*string*, *triggers*, *callback*)
Register a string that once typed will trigger callback.

If triggers are empty, the string will trigger as soon as it's typed. Otherwise it will only trigger if it's followed by one of the triggers.

Keyboard hook needs to be installed for hotstrings to work. Otherwise this method raises RuntimeError.

Parameters

- **string** (*str*) – The string that will trigger the callback.
- **triggers** (*(str)*) – Iterable of characters that will be checked for after the string.
- **callback** (*Callable*) – A callable that will be called with *HotString* as a single argument.

Returns A hotstring object.

Return type *HotString*

Raises RuntimeError

unregister_hotstring(*hotstring*)
Unregister a previously registered hotstring.

Parameters **hotstring** (HotString) – A hotstring object as returned by e.g. *register_hotstring()*.

keypress(*key*, *state=None*)
Simulate a key press/release event.

Parameters

- **key** (Key) – Key to simulate.
- **state** (KeyState) – The state to simulate. If state is None (default), both key press and release are simulated.

type(*string*)
Type a given string.

Depending on underlying implementation and current platform this may be more efficient then using *keypress()*.

Parameters **string** (*str*) – String to type.

1.2 Pointer

class macpy.**Pointer**
Pointer interface object.

Allows simulating pointer input as well as reading data from connected physical pointing devices.

close()
Close opened resources and cleanly exit mainloop.

Call this method when you are done with this object.

install_pointer_hook(*callback*, *grab=False*)
Installs a low level hook that sends all pointer events to the callback.

Callback must take a single event argument.

For event definitions see *PointerEventMotion*, *PointerEventButton* and *PointerEventAxis*.

Parameters

- **callback** (*Callable*) – A callable which will receive pointer events.
- **grab** (*bool*) – If grab is True events are consumed and not passed through to other applications. .. note:

```
Even if grab is :obj:`True`, synthetic events are still
allowed on Windows.

Under wayland this option does nothing.
```

uninstall_pointer_hook()
Uninstalls pointer hook and stops hook's loop.

You don't have to explicitly call this method. Calling *close()* will remove the hook automatically if it's installed.

warp(*x*, *y*, *relative=False*)
Warp pointer to the given location on screen.

Pointer cannot be warped beyond the bounds of the virtual screen.

Parameters

- **x** (*int*) – X coordinate.
- **y** (*int*) – Y coordinate.
- **relative** (*bool*) – Whether given coordinates are absolute or relative to current pointer position.

scroll(*axis*, *value*)
Simulate mouse scroll wheel along the given axis.

Note: value is platform dependent, so the same value may result in different amount scrolled depending on current platform.

Parameters

- **axis** (PointerAxis) – The axis along which to scroll.
- **value** (*int*) – The amount which to scroll. See Note.

click(*key*, *state=None*)
Simulate a mouse click.

Parameters

- **key** (Key) – A button to click.
- **state** (KeyState) – The state to simulate. If state is None both button press and release are simulated.

get_button_state(*button*)
Check whether the button is pressed or released.

Parameters **button** (Key) – The button to check.

Returns Current state of the key.

Return type *KeyState*

position
Current position of the pointer on screen.

Returns

A namedtuple where first member is the x coordinate and the second - y coordinate, in pixels.

Return type tuple

1.3 Window

class macpy.**Window**(*window*)
Window interface object.

Allows manipulating windows on supported platforms: activating, minimizing, closing, moving, etc.

Rather than instanciating this class directly, use one of the class methods, e.g. *get_active()*.

title
Visible window title. Might be None if window is already closed.

Type str

wm_class
Window class. Might be None if window is already closed.

Type str

pid
PID of the process to which this window belongs. Might be None if window is closed or if window does not set this property.

Type int

classmethod install_window_hook(*callback*)
Hook window creation, destruction and focus change.

Callback is called with *WindowEvent* as a single argument.

Parameters **callback** (*Callable*) – A callable to receive events.

Raises NotImplementedError

classmethod uninstall_window_hook()
Remove window hook.

Since hook runs in a separate thread, you should call this method once you are done for a clean exit.

Raises NotImplementedError

classmethod list_windows()
Return a tuple of currently open window objects.

Returns A tuple of currently open windows.

Return type (*Window*, ...)

Raises NotImplementedError

classmethod get_active()
Return currently focused window.

Returns A window object.

Return type *Window*

Raises NotImplementedError

classmethod get_under_pointer()
Return the window that is currently under pointer.

Returns A window object.

Return type *Window*

Raises NotImplementedError

classmethod get_by_class(*wm_class*)
Return the first window whose *wm_class* matches wm_class.

Parameters **wm_class** (*str*) – Window class to match.

Returns A window object.

Return type *Window*

Raises NotImplementedError

classmethod get_by_title(*title*)
Return the first window whose *title* matches title.

Parameters **title** (*str*) – Partial window title to match.

Returns A window object.

Return type *Window*

Raises NotImplementedError

state
This window's state.

Returns Window state.

Return type *WindowState*

position
This window's position on screen in pixels.

Returns A namedtuple of x and y coordinates.

Return type (int, int)

size
This window's size in pixels.

Returns A namedtuple of width and height.

Return type (int, int)

activate()
Activate this window.

restore()
Restore this window.

minimize()
Minimize this window.

maximize()
Maximize this window.

resize(*width*, *height*)
Resize this window to the given width and height in pixels.

Parameters

- **width** (*int*) – New width.
- **height** (*int*) – New height.

move(*x*, *y*)
Move this window to the given screen x and y coordinates in pixels.

Parameters

- **x** (*int*) – New position along x axis.
- **y** (*int*) – New position along y axis.

close()
Request this window to close.

If there are unsaved data, the window may refuse to close.

force_close()
Forcibly close this window.

send_event(*event*)
Send an input event dirrectly to this window, regardless of whether it has input focus.

Valid input events are *KeyboardEvent*, *PointerEventMotion*, *PointerEventButton* and *PointerEventAxis*.

Note: For events that contain coordinates, these coordinates are always relative to this window.

Parameters **event** (Event) – Event to send.

CHAPTER 2

Events

class macpy.event.**Event**
Base class for all macpy events.

time
Event timestamp. This does not translate to concrete time but timestamps of later events are guaranteed to be greater than timestamps of earlier events.

Type float

2.1 Pointer

class macpy.event.**PointerEventMotion**(*x*, *y*, *modifiers*)
Event representing pointer movement on screen.

position
A namedtuple containing x and y coordinates of pointer on screen.

Type tuple

modifiers
A namedtuple containing modifier state at the time of this event.

Type tuple

__init__(*x*, *y*, *modifiers*)
Event representing pointer motion.

Parameters

- **x** (*int*) – Pointer position on x axis in pixels.
- **y** (*int*) – Pointer position on y axis in pixels.
- **modifiers** (*dict*) – Modifier key state at the time of this event.

class macpy.event.**PointerEventButton**(*x*, *y*, *button*, *state*, *modifiers*)
Event representing button events on connected pointing devices.

button
Button that was pressed/released.

Type *Key*

state
Whether button was pressed or released.

Type *KeyState*

modifiers
A namedtuple containing modifier state at the time of this event.

Type tuple

__init__(*x*, *y*, *button*, *state*, *modifiers*)
Event representing button press/release.

Parameters

- **x** (*int*) – Pointer position on x axis in pixels.
- **y** (*int*) – Pointer position on y axis in pixels.
- **button** (Key) – Button that was pressed/released.
- **state** (KeyState) – Whether the button was pressed or released.
- **modifiers** (*dict*) – Modifier key state at the time of this event.

class macpy.event.**PointerEventAxis**(*x*, *y*, *value*, *axis*, *modifiers*)
Event representing scrolling.

value
The amount scrolled. This is platform dependent.

Type float

axis
The axis along which scrolling ocured.

Type *PointerAxis*

modifiers
A namedtuple containing modifier state at the time of this event.

Type tuple

__init__(*x*, *y*, *value*, *axis*, *modifiers*)
Event representing scrolling.

Parameters

- **x** (*int*) – Pointer position on x axis in pixels.
- **y** (*int*) – Pointer position on y axis in pixels.
- **value** (*int*) – The amount scrolled, exact interpretation of this value is platform-specific.
- **axis** (PointerAxis) – The axis along which to scroll.
- **modifiers** (*dict*) – Modifier key state at the time of this event.

2.2 Keyboard

class macpy.event.**KeyboardEvent**(*key*, *state*, *char*, *modifiers*, *locks*)
Event representing key press/release on connected keyboards.

key
The key that was pressed/released.

Type *Key*

state
Whether the key was pressed or released.

Type *KeyState*

char
The character produced by this key event if any.

Type str

modifiers
A namedtuple containing modifier state at the time of this event.

Type tuple

locks
A namedtuple containing lock key state at the time of this event.

Type tuple

__init__(*key*, *state*, *char*, *modifiers*, *locks*)
Event representing key press/release.

Parameters

- **key** (Key) – The key that will be pressed/released.
- **state** (KeyState) – Whether the key will be pressed or released.
- **char** (*str*) – The character that will be typed. Currently this is ignored, you can set it to None.
- **modifiers** (*dict*) – Modifier key state at the time of this event.
- **locks** (*dict*) – Lock key state at the time of this event.

class macpy.event.**HotKey**(*key*, *modifiers*)
A hotkey object.

Hotkey object are hashable and compare equal regardless of timestamps.

key
A key that triggered this event.

Type *Key*

modifiers
A frozenset of modifier keys that were also pressed.

Type frozenset

class macpy.event.**HotString**(*string*, *triggers*, *trigger=None*)
A hotstring object.

Hotstring objects are hashable and compare equal regardless of timestamps and the current trigger.

string
The string that needs to be typed to trigger this hotstring.

Type str

triggers
The trigger keys that need to be typed after the string. This frozenset may be empty.

Type frozenset

trigger
The trigger that triggered this hotstring. May be None.

Type str

2.3 Window

class macpy.event.**WindowEvent**(*window*, *event_type*)
Event representing window creation, destruction and focus change.

window
The window that was created/destroyed/focused.

Type *Window*

type
The action that was taken on the window.

Type *WindowEventType*

CHAPTER 3

Enumerations

class macpy.**RecordType**
An enumeration specifying which events to record.

KEYBOARD
Record keyboard events only.

POINTER
Record pointer events only.

BOTH
Record both keyboard and pointer events.

class macpy.key.**Key**
An enumeration describing platform independent keys/buttons.

While members of this enum behave the same on every platform, not every platform defines every key. For complete list of keys/buttons defined on your platform see `input.h` on Linux and Virtual Keycodes on Windows.

Members of this enum are also valid `tuple` where first member is a Linux event code and second member is a Windows virtual keycode. These can also be accessed as attributes *ec* and *vk* respectively.

ec
A Linux event code that is this enum member.

Returns A Linux event code.

Return type EventCode

vk
A Windows virtual keycode that is this enum member.

Returns A Windows virtual keycode.

Return type VirtualKeycode

class macpy.key.**KeyState**
An enumeration describing whether the key/button is pressed or released.

This enum implements `__bool__()`, so if the key is pressed it will be `True` and `False` otherwise.

class macpy.event.**PointerAxis**
An enumeration describing pointer scrolling axis.

class macpy.event.**WindowState**
An enumeration describing window state.

class macpy.event.**WindowEventType**
An enumeration describing whether window was created, destroyed or focused.

Python Module Index

m

Index

Symbols

A

B

C

E

F

G

H

I

K

L

M