pystray

Release 0.19.5

Sep 17, 2023

Contents

1 Table of contents **3**
- 1.1 Creating a *system tray icon* . . . 3
- 1.2 Integrating with other frameworks . . . 6
- 1.3 Selecting a backend . . . 7
- 1.4 Frequently asked question . . . 7
- 1.5 Reference . . . 8

Index **13**

This library allows you to create a *system tray icon*.

It allows specifying an icon, a title and a callback for when the icon is activated. The icon and title can be changed after the icon has been created, and the visibility of the icon can be toggled.

CHAPTER 1

Table of contents

1.1 Creating a *system tray icon*

In order to create a *system tray icon*, the class `pystray.Icon` is used:

```
import pystray

from PIL import Image, ImageDraw


def create_image(width, height, color1, color2):
    # Generate an image and draw a pattern
    image = Image.new('RGB', (width, height), color1)
    dc = ImageDraw.Draw(image)
    dc.rectangle(
        (width // 2, 0, width, height // 2),
        fill=color2)
    dc.rectangle(
        (0, height // 2, width // 2, height),
        fill=color2)

    return image


# In order for the icon to be displayed, you must provide an icon
icon = pystray.Icon(
    'test name',
    icon=create_image(64, 64, 'black', 'white'))


# To finally show you icon, call run
icon.run()
```

The call to `pystray.Icon.run()` is blocking, and it must be performed from the main thread of the application. The reason for this is that the *system tray icon* implementation for *OSX* will fail unless called from the main thread,

and it also requires the application runloop to be running. `pystray.Icon.run()` will start the runloop.

If you only target *Windows*, calling `run()` from a thread other than the main thread is safe.

The `run()` method accepts an optional argument: `setup`, a callable.

The `setup` function will be run in a separate thread once the *system tray icon* is ready. The icon does not wait for it to complete, so you may put any code that would follow the call to `pystray.Icon.run()` in it.

The call to `pystray.Icon.run()` will not complete until `stop()` is called.

1.1.1 Getting input from the *system tray icon*

In order to receive notifications about user interaction with the icon, a popup menu can be added with the `menu` constructor argument.

This must be an instance of `pystray.Menu`. Please see the reference for more information about the format.

It will be displayed when the right-hand button has been pressed on the icon on *Windows*, and when the icon has been clicked on other platforms. Menus are not supported on *X*.

Menus also support a default item. On *Windows*, and *X*, this item will be activated when the user clicks on the icon using the primary button. On other platforms it will be activated if the menu contains no visible entries; it does not have to be visible.

All properties of menu items, except for the callback, can be dynamically calculated by supplying callables instead of values to the menu item constructor. The properties are recalculated every time the icon is clicked or any menu item is activated.

If the dynamic properties change because of an external event, you must ensure that `Icon.update_menu` is called. This is required since not all supported platforms allow for the menu to be generated when displayed.

1.1.2 Creating the menu

This is not supported on Xorg; please check Icon.HAS_MENU at runtime for support on the current platform.

A menu can be attached to a system tray icon by passing an instance of *pystray.Menu* as the `menu` keyword argument.

A menu consists of a list of menu items, optionally separated by menu separators.

Separators are intended to group menu items into logical groups. They will not be displayed as the first and last visible item, and adjacent separators will be hidden.

A menu item has several attributes:

text* and *action The menu item text and its associated action.

These are the only required attributes. Please see *submenu* below for alternate interpretations of *action*.

checked Whether the menu item is checked.

This can be one of three values:

`False` The item is decorated with an unchecked check box.

`True` The item is decorated with a checked check box.

`None` There is no hint that the item is checkable.

If you want this to actually be togglable, you must pass a callable that returns the current state:

```
from pystray import Icon as icon, Menu as menu, MenuItem as item

state = False

def on_clicked(icon, item):
    global state
    state = not item.checked

# Update the state in `on_clicked` and return the new state in
# a `checked` callable
icon('test', create_image(), menu=menu(
    item(
        'Checkable',
        on_clicked,
        checked=lambda item: state))).run()
```

radio *This is not supported on macOS; please check Icon.HAS_MENU_RADIO at runtime for support on the current platform.*

Whether this is a radio button.

This is used only if `checked` is `True` or `False`, and only has a visual meaning. The menu has no concept of radio button groups:

```
from pystray import Icon as icon, Menu as menu, MenuItem as item

state = 0

def set_state(v):
    def inner(icon, item):
        global state
        state = v
    return inner

def get_state(v):
    def inner(item):
        return state == v
    return inner

# Let the menu items be a callable returning a sequence of menu
# items to allow the menu to grow
icon('test', create_image(), menu=menu(lambda: (
    item(
        'State %d' % i,
        set_state(i),
        checked=get_state(i),
        radio=True)
    for i in range(max(5, state + 2))))).run()
```

default *This is not supported on Darwin and using AppIndicator; please check Icon.HAS_DEFAULT at runtime for support on the current platform.*

Whether this is the default item.

It is drawn in a distinguished style and will be activated as the default item on platforms that support default actions. On *X*, this is the only action available.

visible Whether the menu item is visible.

enabled Whether the menu item is enabled. Disabled menu items are displayed, but are greyed out and cannot be activated.

submenu The submenu, if any, that is attached to this menu item. Either a submenu or an action can be passed as the second argument to the constructor.

The submenu must be an instance of `Menu`:

```
from pystray import Icon as icon, Menu as menu, MenuItem as item

icon('test', create_image(), menu=menu(
    item(
        'With submenu',
        menu(
            item(
                'Submenu item 1',
                lambda icon, item: 1),
            item(
                'Submenu item 2',
                lambda icon, item: 2))))).run()
```

Once created, menus and menu items cannot be modified. All attributes except for the menu item callbacks can however be set to callables returning the current value. This also applies to the sequence of menu items belonging to a menu: this can be a callable returning the current sequence.

1.1.3 Displaying notifications

This is not supported on macOS and Xorg; please check Icon.HAS_NOTIFICATION at runtime for support on the current platform.

To display a system notification, use *pystray.Icon.notify()*:

```
from pystray import Icon as icon, Menu as menu, MenuItem as item

icon('test', create_image(), menu=menu(
    item(
        'With submenu',
        menu(
            item(
                'Show message',
                lambda icon, item: icon.notify('Hello World!')),
            item(
                'Submenu item 2',
                lambda icon, item: icon.remove_notification()))))).run()
```

1.2 Integrating with other frameworks

The *pystray* `run` method is blocking, and must be called from the main thread to maintain platform independence. This is troublesome when attempting to use frameworks with an event loop, since they may also require running in the main thread.

For this case you can use `run_detached`. This allows you to setup the icon and then pass control to the framework. Please see the documentation for more information.

1.3 Selecting a backend

pystray aims to provide a unified *API* for all supported platforms. In some cases, however, that is not entirely possible.

This library supports a number of backends. On *macOS* and *Windows*, the operating system has system tray icons built-in, so the default backends should be used, but on *Linux* you may have to make a decision depending on your needs.

By setting the environment variable `PYSTRAY_BACKEND` to one of the strings in the next section, the automatic selection is turned off.

1.3.1 Supported backends

appindicator This is one of the backends available on *Linux*, and is the preferred choice. All *pystray* features except for a menu default action are supported, and if the *appindicator* library is installed on the system and the desktop environment supports it, the icon is guaranteed to be displayed.

If the *appindicator* library is not available on the system, a fallback on *ayatana-appindicator* is attempted.

darwin This is the default backend when running on *macOS*. All *pystray* features are available.

gtk This is one of the backends available on *Linux*, and is prioritised above the *XOrg* backend. It uses *GTK* as underlying library. All *pystray* features are available, but it may not actually result in a visible icon: when running a *gnome-shell* session, an third party plugin is required to display legacy tray icons.

win32 This is the default backend when running on *Windows*. All *pystray* features are available.

xorg This is one of the backends available on *Linux*. It is used as a fallback when no other backend can be loaded. It does not support any menu functionality except for a default action.

1.4 Frequently asked question

1.4.1 How do I use *pystray* in a *virtualenv* on *Linux*?

On *Linux*, runtime introspection data is required to use the *AppIndicator* backend, and the *GTK* backend may not be fully functional without installing desktop environment extensions. The *XOrg* backend will work, but it provides limited functionality.

In order to use the *AppIndicator* backend, you may install the package `PyGObject`. No wheel is provided, so the package must be built locally. On *Debian* derivatives, such as *Ubuntu*, the following packages, in addition to compilers and *pkg-config*, must be installed:

- `libcairo-dev`
- `libgirepository1.0-dev`

1.4.2 I am trying to integrate with a framework, but `run_detached` does not work

The `run_detached` method is used to allow a different framework to drive the main loop. This requires that the framework uses the same kind of mainloop.

On *Windows* and *macOS*, this will be the case if you use the platform GUI toolkits. On *Linux*, the situation is a bit more complicated. Generally, the *xorg* backend will work with any toolkit, as long as you run it in an *X* session and not under *Wayland*. The *GTK* and *AppIndicator* backends will work if your toolkit is based on *GObject*.

However, `run_detached` is strictly necessary only on *macOS*. For other platforms, it is possible to just launch the icon mainloop in a thread:

```
import pystray
import threading
import some_toolkit

# Create the icon
icon = pystray.Icon(
    'test name',
    icon=create_icon())

# Run the icon mainloop in a separate thread
threading.Thread(target=icon.run).start()

# Run the toolkit mainlon in the main thread
some_toolkit.mainloop()
```

1.5 Reference

class pystray.**Icon**(*name*, *icon=None*, *title=None*, *menu=None*, ***kwargs*)

A representation of a system tray icon.

The icon is initially hidden. Set *visible* to `True` to show it.

Parameters

- **name** (*str*) – The name of the icon. This is used by the system to identify the icon.
- **icon** – The icon to use. If this is specified, it must be a `PIL.Image.Image` instance.
- **title** (*str*) – A short title for the icon.
- **menu** – A menu to use as popup menu. This can be either an instance of *Menu* or an iterable, which will be interpreted as arguments to the *Menu* constructor, or `None`, which disables the menu.

  The behaviour of the menu depends on the platform. Only one action is guaranteed to be invokable: the first menu item whose *default* attribute is set.

  Some platforms allow both menu interaction and a special way of activating the default action, some platform allow only either an invisible menu with a default entry as special action or a full menu with no special way to activate the default item, and some platforms do not support a menu at all.
- **kwargs** – Any non-standard platform dependent options. These should be prefixed with the platform name thus: `appindicator_`, `darwin_`, `gtk_`, `win32_` or `xorg_`.

  Supported values are:

  darwin_nsapplication An `NSApplication` instance used to run the event loop. If this is not specified, the shared application will be used.

HAS_DEFAULT_ACTION = True

Whether this particular implementation has a default action that can be invoked in a special way, such as clicking on the icon.

HAS_MENU = True

Whether this particular implementation supports menus.

HAS_MENU_RADIO = True
Whether this particular implementation supports displaying mutually exclusive menu items using the *MenuItem.radio* attribute.

HAS_NOTIFICATION = True
Whether this particular implementation supports displaying a notification.

SETUP_THREAD_TIMEOUT = 5.0
The timeout, in secods, before giving up on waiting for the setup thread when stopping the icon.

icon
The current icon.

Setting this to a falsy value will hide the icon. Setting this to an image while the icon is hidden has no effect until the icon is shown.

menu
The menu.

Setting this to a falsy value will disable the menu.

name
The name passed to the constructor.

notify(*message*, *title=None*)
Displays a notification.

The notification will generally be visible until *remove_notification()* is called.

The class field *HAS_NOTIFICATION* indicates whether this feature is supported on the current platform.

Parameters

- **message** (*str*) – The message of the notification.
- **title** (*str*) – The title of the notification. This will be replaced with *title* if `None`.

remove_notification()
Remove a notification.

run(*setup=None*)
Enters the loop handling events for the icon.

This method is blocking until *stop()* is called. It *must* be called from the main thread.

Parameters **setup** (*callable*) – An optional callback to execute in a separate thread once the loop has started. It is passed the icon as its sole argument.

Please note that this function is started in a thread, and when the icon is stopped, an attempt to join this thread is made, so stopping the icon may be blocking for up to `SETUP_THREAD_TIMEOUT` seconds if the function is not well-behaved.

If not specified, a simple setup function setting *visible* to `True` is used. If you specify a custom setup function, you must explicitly set this attribute.

run_detached(*setup=None*)
Prepares for running the loop handling events detached.

This allows integrating *pystray* with other libraries requiring a mainloop. Call this method before entering the mainloop of the other library.

Depending on the backend used, calling this method may require special preparations:

macOS Pass an instance of `NSApplication` retrieved from the library with which you are integrating as the argument `darwin_nsapplication`. This will allow this library to integrate with the main loop.

Parameters **setup** (*callable*) – An optional callback to execute in a separate thread once the loop has started. It is passed the icon as its sole argument.

If not specified, a simple setup function setting *visible* to `True` is used. If you specify a custom setup function, you must explicitly set this attribute.

Raises **NotImplementedError** – if this is not implemented for the preparations taken

stop()
Stops the loop handling events for the icon.

If the icon is not running, calling this method has no effect.

title
The current icon title.

update_menu()
Updates the menu.

If the properties of the menu descriptor are dynamic, that is, any are defined by callables and not constants, and the return values of these callables change by actions other than the menu item activation callbacks, calling this function is required to keep the menu in sync.

This is required since not all supported platforms allow the menu to be generated when shown.

For simple use cases where menu changes are triggered by interaction with the menu, this method is not necessary.

visible
Whether the icon is currently visible.

Raises **ValueError** – if set to `True` and no icon image has been set

class pystray.**Menu**(**items*)
A description of a menu.

A menu description is immutable.

It is created with a sequence of `Menu.Item` instances, or a single callable which must return a generator for the menu items.

First, non-visible menu items are removed from the list, then any instances of *SEPARATOR* occurring at the head or tail of the item list are removed, and any consecutive separators are reduced to one.

SEPARATOR = <pystray._base.MenuItem object>
A representation of a simple separator

items
All menu items.

visible
Whether this menu is visible.

class pystray.**MenuItem**(*text*, *action*, *checked=None*, *radio=False*, *default=False*, *visible=True*, *enabled=True*)
A single menu item.

A menu item is immutable.

It has a text and an action. The action is either a callable of a menu. It is callable; when called, the activation callback is called.

The *visible* attribute is provided to make menu creation easier; all menu items with this value set to `False` will be discarded when a *Menu* is constructed.

checked
Whether this item is checked.

This can be either `True`, which implies that the item is checkable and checked, `False`, which implies that the item is checkable but not checked, and `None` for uncheckable items.

Depending on platform, uncheckable items may be rendered differently from unchecked items.

default
Whether this is the default menu item.

enabled
Whether this menu item is enabled.

radio
Whether this item is a radio button.

This is only used for checkable items. It is always set to `False` for uncheckable items.

submenu
The submenu used by this menu item, or `None`.

text
The menu item text.

visible
Whether this menu item is visible.

If the action for this menu item is a menu, that also has to be visible for this property to be `True`.

- genindex

Index

C

checked (*pystray.MenuItem attribute*), 11

D

default (*pystray.MenuItem attribute*), 11

E

enabled (*pystray.MenuItem attribute*), 11

H

HAS_DEFAULT_ACTION (*pystray.Icon attribute*), 8
HAS_MENU (*pystray.Icon attribute*), 8
HAS_MENU_RADIO (*pystray.Icon attribute*), 8
HAS_NOTIFICATION (*pystray.Icon attribute*), 9

I

Icon (*class in pystray*), 8
icon (*pystray.Icon attribute*), 9
items (*pystray.Menu attribute*), 10

M

Menu (*class in pystray*), 10
menu (*pystray.Icon attribute*), 9
MenuItem (*class in pystray*), 10

N

name (*pystray.Icon attribute*), 9
notify() (*pystray.Icon method*), 9

R

radio (*pystray.MenuItem attribute*), 11
remove_notification() (*pystray.Icon method*), 9
run() (*pystray.Icon method*), 9
run_detached() (*pystray.Icon method*), 9

S

SEPARATOR (*pystray.Menu attribute*), 10
SETUP_THREAD_TIMEOUT (*pystray.Icon attribute*), 9
stop() (*pystray.Icon method*), 10
submenu (*pystray.MenuItem attribute*), 11

T

text (*pystray.MenuItem attribute*), 11
title (*pystray.Icon attribute*), 10

U

update_menu() (*pystray.Icon method*), 10

V

visible (*pystray.Icon attribute*), 10
visible (*pystray.Menu attribute*), 10
visible (*pystray.MenuItem attribute*), 11