dekstop-notifier

Release 3.4.2

Sam Schott

Nov 22, 2022

BACKGROUND

This documentation provides a short introduction to `desktop-notifier` and an API reference for the main module and platform implementations.

CHAPTER

ONE

PLATFORM SUPPORT

Some platforms may not support all options. For instance, some Linux desktop environments may not support notifications with buttons. macOS and Windows do not support manually setting the app icon or name. Instead, both are always determined by the application which uses the Library. This can be Python itself, when used interactively, or a frozen app bundle when packaged with PyInstaller or similar solutions.

The table below gives an overview over supported functionality for different platforms. Please refer to the platform documentation for more detailed information:

- macOS / iOS: UNUserNotificationCenter
- Linux: org.freedesktop.Notifications
- Windows: Toast Notifications

Note: Windows support is still experimental and may have some rough edges.

Option	Description	Linux	macOS/iOS	Windows
app_name	The application name to display	✓	–[1]	–[1]
app_icon	The icon shown with the notification	✓	–[1]	–[1]
title	The notification title	✓	✓	✓
message	The notification message	✓	✓	✓
urgency	Determines persistence and appearance	✓	✓[2]	✓
buttons	One or more buttons with callbacks	✓[3]	✓[4]	✓
reply_field	An interactive reply field	–	✓	✓
on_clicked	A callback to invoke on click	✓[3]	✓	✓
on_dismissed	A callback to invoke on dismissal	✓[3]	✓	✓
sound	Play a default sound with the notification	✓[3]	✓	✓
thread	An identifier to group notifications together	–	✓	✓
attachment	File attachment, e.g., an image	✓[5]	✓[5]	✓[5]
timeout	Duration until notification auto-dismissal	✓[3]	–	–

[1] App name and icon on macOS and Windows are automatically determined by the calling application.
[2] Only on macOS 12 and later.
[3] May be ignored by some notification servers, depending on the desktop environment.
[4] Only a single button is supported by our implementation for macOS 10.13 and lower.
[5] Limitations on file types exist for each platform.

1.1 Callbacks

MacOS, Windows and almost all Linux notification servers support executing a callback when the notification is clicked. Note the requirements on a running event loop to handle callbacks in Python.

1.2 Urgency

The notification urgency may influence how a notification is displayed. For instance, in Gnome, notifications of critical urgency will remain visible until closed by the user and their buttons will always be expanded.

This is currently not supported on macOS where critical notifications would require a special entitlement issued by Apple.

1.3 Buttons

Our implementation for macOS 10.13 and lower supports only a single button. On iOS and macOS 10.14+, we support an unlimited number of buttons.

Linux desktop environments and notification servers may or may not support a varying number of buttons. Gnome desktops typically support up to three buttons.

Windows supports up to 5 buttons.

When an implementation or a platform supports only a limited number of buttons, any additional buttons specified in the notification request will be silently ignored.

1.4 Attachments

MacOS 10.14+ and iOS support attaching a local file to the notification and may show a preview of the file. Allowed file types are:

- An audio file up to 5 MB: AIFF, WAV, MP3, or MPEG4
- An image file up to 10 MB: JPEG, GIF, or PNG
- A video file up to 50 MB: MPEG, MPEG2, MPEG4, or AVI

On macOS, only previews of image attachments will be shown. iOS will show previews of all of the above attachment types and allows long-pressing the notification to show the full attachment. The notification will still be shown if the attachment cannot be loaded.

Windows supports images only. The image URI must be a web url or point to a resource bundled with the app.

Linux notification servers may support attaching a secondary image to the notification, shown in addition to the app icon. Where this is not supported, the app icon will be replaced by a thumbnail of the image. This is currently the case for Gnome.

CHAPTER

TWO

EVENT LOOP INTEGRATION

Using the asynchronous API is highly recommended to prevent multiple milliseconds of blocking IO from DBus or Cocoa APIs. In addition, execution of callbacks requires a running event loop. On Linux, an asyncio event loop will be sufficient but macOS requires a running CFRunLoop.

You can use rubicon-objc to integrate a Core Foundation CFRunLoop with asyncio:

```
import asyncio
from rubicon.objc.eventloop import EventLoopPolicy

# Install the event loop policy
asyncio.set_event_loop_policy(EventLoopPolicy())

# Get an event loop, and run it!
loop = asyncio.get_event_loop()
loop.run_forever()
```

Desktop-notifier itself uses Rubicon Objective-C to interface with Cocoa APIs so you will not be adding a new dependency. A full example integrating with the CFRunLoop is given in examples folder. Please refer to the Rubicon Objective-C docs for more information.

Likewise, you can integrate the asyncio event loop with a Gtk main loop on Gnome using gbulb. This is not required for full functionality but may be convenient when developing a Gtk app.

CHAPTER

THREE

CONTRIBUTING

Thank you for your interest in contributing!

If you would like to contribute more than a simple bug fix, please open an issue first to discuss potential changes before implementing them.

3.1 Code

To start, install maestral with the `dev` extra to get all dependencies required for development:

```
pip3 install desktop-notifier[dev]
```

This will install packages to check and enforce the code style, use pre-commit hooks and bump the current version.

Code is formatted with black. Coding style is checked with flake8. Type hints, PEP484, are checked with mypy.

You can check the format, coding style, and type hints at the same time by running the provided pre-commit hook from the git directory:

```
pre-commit run -a
```

You can also install the provided pre-commit hook to run checks on every commit. This will however significantly slow down commits. An introduction to pre-commit commit hooks is given at https://pre-commit.com.

3.2 Tests

There are currently no tests. Contributions are very welcome!

CHAPTER

FOUR

DESKTOP_NOTIFIER.MAIN

This module handles desktop notifications and supports multiple backends, depending on the platform.

4.1 Module Contents

4.1.1 Classes

`Urgency`	Enumeration of notification levels
`Button`	A button for interactive notifications
`ReplyField`	A reply field for interactive notifications
`Notification`	A desktop notification
`DesktopNotifier`	Cross-platform desktop notification emitter

class desktop_notifier.main.**DesktopNotifier**(*app_name='Python'*, *app_icon=PYTHON_ICON_PATH*, *notification_limit=None*)

Cross-platform desktop notification emitter

Uses different backends depending on the platform version and available services. All implementations will dispatch notifications without an event loop but will require a running event loop to execute callbacks when the end user interacts with a notification. On Linux, a asyncio event loop is required. On macOS, a CFRunLoop *in the main thread* is required. Packages such as `rubicon.objc` can be used to integrate asyncio with a CFRunLoop.

Parameters

- **app_name** (*str*) – Name to identify the application in the notification center. On Linux, this should correspond to the application name in a desktop entry. On macOS, this argument is ignored and the app is identified by the bundle ID of the sending program (e.g., Python).
- **app_icon** (*Union[pathlib.Path, str, None]*) – Default icon to use for notifications. This should be either a URI string, a `pathlib.Path` path, or a name in a freedesktop.org-compliant icon theme. If None, the icon of the calling application will be used if it can be determined. On macOS, this argument is ignored and the app icon is identified by the bundle ID of the sending program (e.g., Python).
- **notification_limit** (*Optional[int]*) – Maximum number of notifications to keep in the system's notification center. This may be ignored by some implementations.

property app_name: str

The application name

Return type

str

property app_icon: Optional[str]

The application icon: a URI for a local file or an icon name.

Return type
Optional[str]

property current_notifications: List[*desktop_notifier.base.Notification*]

A list of all currently displayed notifications for this app

Return type
List[*desktop_notifier.base.Notification*]

_run_coro_sync(*coro*)

Runs the given coroutine and returns the result synchronously. This is used as a wrapper to conveniently convert the async API calls to synchronous ones.

Parameters
coro (*Coroutine[None, None, T]*) –

Return type
T

async request_authorisation()

Requests authorisation to send user notifications. This will be automatically called for you when sending a notification for the first time. It also can be called manually to request authorisation in advance.

On some platforms such as macOS and iOS, a prompt will be shown to the user when this method is called for the first time. This method does nothing on platforms where user authorisation is not required.

Returns
Whether authorisation has been granted.

Return type
bool

async has_authorisation()

Returns whether we have authorisation to send notifications.

Return type
bool

async send(*title*, *message*, *urgency=Urgency.Normal*, *icon=None*, *buttons=()*, *reply_field=None*, *on_clicked=None*, *on_dismissed=None*, *attachment=None*, *sound=False*, *thread=None*, *timeout=-1*)

Sends a desktop notification.

Some arguments may be ignored, depending on the backend.

This method will always return a `base.Notification` instance and will not raise an exception when scheduling the notification fails. If the notification was scheduled successfully, its `identifier` will be set to the platform's native notification identifier. Otherwise, the `identifier` will be `None`.

Note that even a successfully scheduled notification may not be displayed to the user, depending on their notification center settings (for instance if "do not disturb" is enabled on macOS).

Parameters

- **title** (*str*) – Notification title.
- **message** (*str*) – Notification message.

- **urgency** (desktop_notifier.base.Urgency) – Notification level: low, normal or critical. This may be interpreted differently by some implementations, for instance causing the notification to remain visible for longer, or may be ignored.
- **icon** (*Union[pathlib.Path, str, None]*) – Optional URI string, pathlib.Path or icon name to use. If given, this will replace the icon specified by *app_icon*. Will be ignored on macOS.
- **buttons** (*Sequence[desktop_notifier.base.Button]*) – A list of buttons with callbacks for the notification.
- **reply_field** (*Optional[desktop_notifier.base.ReplyField]*) – Optional reply field to show with the notification. Can be used for instance in chat apps.
- **on_clicked** (*Optional[Callable[[], Any]]*) – Optional callback to call when the notification is clicked. The callback will be called without any arguments. This is ignored by some implementations.
- **on_dismissed** (*Optional[Callable[[], Any]]*) – Optional callback to call when the notification is dismissed. The callback will be called without any arguments. This is ignored by some implementations.
- **attachment** (*Union[pathlib.Path, str, None]*) – Optional URI string or pathlib.Path for an attachment to the notification such as an image, movie, or audio file. A preview of this may be displayed together with the notification. Different platforms and Linux notification servers support different types of attachments. Please consult the platform support section of the documentation.
- **sound** (*bool*) – Whether to play a sound when the notification is shown. The platform's default sound will be used, where available.
- **thread** (*Optional[str]*) – An identifier to group related notifications together. This is ignored on Linux.
- **timeout** (*int*) – The duration (in seconds) for which the notification is shown unless dismissed. Only supported on Linux. Default is -1 which implies OS-specified.

Returns
The scheduled notification instance.

Return type
desktop_notifier.base.Notification

send_sync(*title*, *message*, *urgency=Urgency.Normal*, *icon=None*, *buttons=()*, *reply_field=None*, *on_clicked=None*, *on_dismissed=None*, *attachment=None*, *sound=False*, *thread=None*, *timeout=-1*)

Synchronous call of *send()*, for use without an asyncio event loop.

Returns
The scheduled notification instance.

Parameters

- **title** (*str*) –
- **message** (*str*) –
- **urgency** (desktop_notifier.base.Urgency) –
- **icon** (*Union[pathlib.Path, str, None]*) –
- **buttons** (*Sequence[desktop_notifier.base.Button]*) –
- **reply_field** (*Optional[desktop_notifier.base.ReplyField]*) –

- **on_clicked** (*Optional[Callable[[], Any]]*) –
- **on_dismissed** (*Optional[Callable[[], Any]]*) –
- **attachment** (*Union[pathlib.Path, str, None]*) –
- **sound** (*bool*) –
- **thread** (*Optional[str]*) –
- **timeout** (*int*) –

Return type
desktop_notifier.base.Notification

async clear(*notification*)

Removes the given notification from the notification center.

Parameters
notification (desktop_notifier.base.Notification) – Notification to clear.

Return type
None

async clear_all()

Removes all currently displayed notifications for this app from the notification center.

Return type
None

CHAPTER

FIVE

DESKTOP_NOTIFIER.BASE

This module defines base classes for desktop notifications. All platform implementations must inherit from *DesktopNotifierBase*.

5.1 Module Contents

5.1.1 Classes

Urgency	Enumeration of notification levels
Button	A button for interactive notifications
ReplyField	A reply field for interactive notifications
Notification	A desktop notification
DesktopNotifierBase	Base class for desktop notifier implementations

5.1.2 Functions

resource_path(package, resource)	

5.1.3 Attributes

logger	
PYTHON_ICON_PATH	

desktop_notifier.base.**resource_path**(*package*, *resource*)

Parameters

- **package** (*str*) –
- **resource** (*str*) –

Return type
ContextManager[pathlib.Path]

desktop_notifier.base.**logger**

desktop_notifier.base.**PYTHON_ICON_PATH**

exception desktop_notifier.base.**AuthorisationError**

Bases: Exception

Raised when we are not authorised to send notifications

class desktop_notifier.base.**Urgency**

Bases: enum.Enum

Enumeration of notification levels

The interpretation and visuals will depend on the platform.

Critical = critical

For critical errors.

Normal = normal

Default platform notification level.

Low = low

Low priority notification.

class desktop_notifier.base.**Button**(*title*, *on_pressed=None*)

A button for interactive notifications

Parameters

- **title** (*str*) – The button title.
- **on_pressed** (*Optional[Callable[[], Any]]*) – Callback to invoke when the button is pressed. This is called without any arguments.

class desktop_notifier.base.**ReplyField**(*title='Reply'*, *button_title='Send'*, *on_replied=None*)

A reply field for interactive notifications

Parameters

- **title** (*str*) – A title for the field itself. On macOS, this will be the title of a button to show the field.
- **button_title** (*str*) – The title of the button to send the reply.
- **on_replied** (*Optional[Callable[[str], Any]]*) – Callback to invoke when the button is pressed. This is called without any arguments.

class desktop_notifier.base.**Notification**(*title*, *message*, *urgency=Urgency.Normal*, *icon=None*, *buttons=()*, *reply_field=None*, *on_clicked=None*, *on_dismissed=None*, *attachment=None*, *sound=False*, *thread=None*, *timeout=-1*)

A desktop notification

Parameters

- **title** (*str*) – Notification title.
- **message** (*str*) – Notification message.
- **urgency** (Urgency) – Notification level: low, normal or critical.
- **icon** (*Optional[str]*) – URI for an icon to use for the notification or icon name.

- **buttons** (*Sequence[Button]*) – A list of buttons for the notification.
- **reply_field** (*Optional[ReplyField]*) – An optional reply field/
- **on_clicked** (*Optional[Callable[[], Any]]*) – Callback to call when the notification is clicked. The callback will be called without any arguments.
- **on_dismissed** (*Optional[Callable[[], Any]]*) – Callback to call when the notification is dismissed. The callback will be called without any arguments.
- **sound** (*bool*) – Whether to play a sound when the notification is shown.
- **thread** (*Optional[str]*) – An identifier to group related notifications together.
- **timeout** (*int*) – Duration for which the notification in shown.
- **attachment** (*Optional[str]*) –

Attachment
URI for an attachment to the notification.

property identifier: Union[str, int, None]

An platform identifier which gets assigned to the notification after it was sent. This may be a str or int.

Return type
Union[str, int, None]

class desktop_notifier.base.**DesktopNotifierBase**(*app_name='Python'*, *app_icon=None*, *notification_limit=None*)

Base class for desktop notifier implementations

Parameters

- **app_name** (*str*) – Name to identify the application in the notification center.
- **app_icon** (*Optional[str]*) – Default icon to use for notifications.
- **notification_limit** (*Optional[int]*) – Maximum number of notifications to keep in the system's notification center.

property current_notifications: List[*Notification*]

A list of all notifications which currently displayed in the notification center

Return type
List[*Notification*]

abstract async request_authorisation()

Request authorisation to send notifications.

Returns
Whether authorisation has been granted.

Return type
bool

abstract async has_authorisation()

Returns whether we have authorisation to send notifications.

Return type
bool

async send(*notification*)

Sends a desktop notification. Some arguments may be ignored, depending on the implementation. This is a wrapper method which mostly performs housekeeping of notifications ID and calls *_send()* to actually schedule the notification. Platform implementations must implement *_send()*.

Parameters
notification (Notification) – Notification to send.

Return type
None

_clear_notification_from_cache(*notification*)

Removes the notification from our cache. Should be called by backends when the notification is closed.

Parameters
notification (Notification) –

Return type
None

abstract async _send(*notification*, *notification_to_replace*)

Method to send a notification via the platform. This should be implemented by subclasses.

Implementations must raise an exception when the notification could not be delivered. If the notification could be delivered but not fully as intended, e.g., because associated resources could not be loaded, implementations should emit a log message of level warning.

Parameters

- **notification** (Notification) – Notification to send.
- **notification_to_replace** (*Optional*[Notification]) – Notification to replace, if any.

Returns
The platform's ID for the scheduled notification.

Return type
Union[str, int]

async clear(*notification*)

Removes the given notification from the notification center. This is a wrapper method which mostly performs housekeeping of notifications ID and calls *_clear()* to actually clear the notification. Platform implementations must implement *_clear()*.

Parameters
notification (Notification) – Notification to clear.

Return type
None

abstract async _clear(*notification*)

Removes the given notification from the notification center. Should be implemented by subclasses.

Parameters
notification (Notification) – Notification to clear.

Return type
None

async clear_all()

Clears all notifications from the notification center. This is a wrapper method which mostly performs housekeeping of notifications ID and calls *_clear_all()* to actually clear the notifications. Platform implementations must implement *_clear_all()*.

Return type
None

abstract async _clear_all()

Clears all notifications from the notification center. Should be implemented by subclasses.

Return type
None

CHAPTER

SIX

GETTING STARTED

Basic usage only requires the user to specify a notification title and message:

```
from desktop_notifier import DesktopNotifier

notifier = DesktopNotifier()
n = notifier.send_sync(title="Hello world!", message="Notification body")
```

By default, "Python" will be used as the app name for all notifications, but you can also manually specify an app name and icon. Advanced usage also allows setting different notification options, such as notification urgency, buttons, callbacks, etc:

```
from desktop_notifier import DesktopNotifier, Urgency, Button, ReplyField

notifier = DesktopNotifier(
    app_name="Sample App",
    icon="file:///path/to/icon.png",
    notification_limit=10
)

async def main():
    await notify.send(
      title="Julius Caesar",
      message="Et tu, Brute?",
      urgency=Urgency.Critical,
      buttons=[
        Button(
          title="Mark as read",
          on_pressed=lambda: print("Marked as read")),
      ],
      reply_field=ReplyField(
        title="Reply",
        button_title="Send",
        on_replied=lambda text: print("Brutus replied:", text),
      ),
      on_clicked=lambda: print("Notification clicked"),
      on_dismissed=lambda: print("Notification dismissed"),
      sound=True,
    )

asyncio.run(main())
```

Note that some platforms may not support all options. Any options or configuration which are not supported by the platform will be silently ignored. Please refer to *Platform support* for more information.

In addition to sending notifications, `desktop_notifier.main.DesktopNotifier` also provides methods to clear notifications from the platform's notification center and to request and verify user permissions to send notifications where this is required by the platform. Please refer to the API docs for the evolving functionality.

CHAPTER

SEVEN

NOTES ON MACOS

On macOS 10.14 and higher, the implementation uses the `UNUserNotificationCenter` instead of the deprecated `NSUserNotificationCenter`. `UNUserNotificationCenter` restricts sending desktop notifications to signed executables. This means that notifications will only work if the Python executable or bundled app has been signed. Note that the installer from python.org provides a properly signed Python framework but **homebrew does not** (manually signing the executable installed by homebrew _should_ work as well).

If you freeze your code with PyInstaller or a similar package, you must sign the resulting app bundle for notifications to work. An ad-hoc signature will be sufficient but signing with an Apple developer certificate is recommended for distribution.

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

D

H

I

L

M

N

P

R

S

U