Windows-Toasts

Release 1.1.0

DatGuy

Feb 12, 2024

CONTENTS

Release v1.1.0.

Windows-Toasts is a Python library used to send toast notifications on Windows machines.

CHAPTER

ONE

WHY WINDOWS-TOASTS?

As opposed to other toast notification libraries, Windows-Toasts uses Windows SDK bindings to create and deliver notifications. This means no less-than-pretty Powershell hackyness and the like, and is in turn scalable, maintainable, and easy to use.

The other packages I've seen also don't use tests or have no active maintainers, while Windows-Toasts has decent test coverage, is fully typed and documented, and has additional features. Any issues or feature requests you put on GitHub shouldn't stand there for too long without receiving a response.

CHAPTER

TWO

CONTENTS

2.1 Getting started

2.1.1 Installing Windows-Toasts

The latest version of Windows-Toasts requires Python 3.9 or later, and supports Windows 10 and 11.

It can be installed using pip:

```
$ python -m pip install windows-toasts
```

Install from source

The stable release version will most likely include the library's latest developments, but you can also install it directly from GitHub, where the code is hosted.

First, clone the repository:

```
$ git clone https://github.com/DatGuy1/Windows-Toasts.git
```

You can then embed it in your own Python package, or install it into your site-packages:

```
$ cd Windows-Toasts
$ python -m pip install .
```

2.1.2 Quickstart

To display a toast notification:

```
# We import WindowsToaster and a toast format we want
from windows_toasts import WindowsToaster, Toast
# Prepare the toaster for bread (or your notification)
toaster = WindowsToaster('Python')
# Initialise the toast
newToast = Toast()
# Set the body of the notification
newToast.text_fields = ['Hello, World!']
# And display it!
toaster.show_toast(newToast)
```

2.2 Interactable toasts

Interactable toasts are toast notifications that lets the user interact with them, be it through different buttons or input fields.

2.2.1 Usage

We import *InteractableWindowsToaster* instead of `WindowsToaster`, but the rest is mostly the same. Here's a basic example:

```
from windows_toasts import InteractableWindowsToaster, Toast, ToastActivatedEventArgs,␣
↪ToastButton

interactableToaster = InteractableWindowsToaster('Questionnaire')
newToast = Toast(['How are you?'])

# Add two actions (buttons)
newToast.AddAction(ToastButton('Decent', 'response=decent'))
newToast.AddAction(ToastButton('Not so good', 'response=bad'))

# Display it like usual
interactableToaster.show_toast(newToast)
```

And we have buttons! We can't do much with them though, at least until we use on_activated.

```
def activated_callback(activatedEventArgs: ToastActivatedEventArgs):
    print(activatedEventArgs.arguments) # response=decent/response=bad

newToast.on_activated = activated_callback
```

Note: To make sure the activation of the toast triggeres the callback following its relegation to the action center, you must use a custom AUMID.

Input fields

Windows-Toasts also supports using text fields and selection boxes.

```
from windows_toasts import InteractableWindowsToaster, Toast, ToastInputTextBox,␣
↪ToastInputSelectionBox, ToastSelection

interactableToaster = InteractableWindowsToaster('Questionnaire')
newToast = Toast(['Please enter your details'])

# A text input field asking the user for their name
newToast.AddInput(ToastInputTextBox('name', 'Your name', 'Barack Obama'))

# Create three selections: Male, female, other, and prefer not to say
toastSelections = (ToastSelection('male', 'Male'), ToastSelection('female', 'Female'),␣
↪ToastSelection('other', 'Other'), ToastSelection('unknown', 'Prefer not to say'))
# Initialise the selection box with a caption 'What is your gender?'. The selections are␣
```

(continues on next page)

(continued from previous page)

```
↪passed in, and it defaults to 'prefer not to say.'
selectionBoxInput = ToastInputSelectionBox('gender', 'What is your gender?',␣
↪toastSelections, default_selection=toastSelections[3])
newToast.AddInput(selectionBoxInput)

# For example: {'name': 'John Smith', 'gender': 'male'}
newToast.on_activated = lambda activatedEventArgs: print(activatedEventArgs.inputs)

interactableToaster.show_toast(newToast)
```

In this case, the on_activated callback will be executed when the user presses on the notification.

Combining the two

We can combine the two and a submit button

```
from windows_toasts import InteractableWindowsToaster, Toast

interactableToaster = InteractableWindowsToaster('Questionnaire')
newToast = Toast()

newToast.text_fields = ['What\'s your name?']
newToast.AddInput(ToastInputTextBox('name', 'Your name', 'Barack Obama'))
newToast.AddAction(ToastButton('Submit', 'submit'))
newToast.on_activated = lambda activatedEventArgs: print(activatedEventArgs.input)

interactableToaster.show_toast(newToast)
```

2.2.2 Caveats

You may have noticed something weird when testing the above code. Why, when we display the toast, does it say command prompt in the top left, and have the icon for it? InteractableWindowsToaster requires an Application User Model ID (AUMID) to function properly. The package provides the command prompt as the default, and the applicationText becomes the *attribution text*.

You can choose between staying with the default command prompt AUMID, finding another one, or creating your own.

2.3 Advanced usage

What else can Windows-Toasts be used for? Since you're here, you probably already have your own idea, but here's a few examples:

2.3.1 Display an image

Lets try out displaying an image

```
from windows_toasts import Toast, ToastDisplayImage, WindowsToaster

toaster = WindowsToaster('Windows-Toasts')

newToast = Toast()
newToast.text_fields = ['<--- look, the Windows logo!']
# str or PathLike
newToast.AddImage(ToastDisplayImage.fromPath('C:/Windows/System32/@WLOGO_96x96.png'))

toaster.show_toast(newToast)
```

Note: When not using InteractableWindowsToaster you can display up to two images, and one of them must be marked as 'hero'.

2.3.2 Open a website on click

We use *windows_toasts.toast.Toast.launch_action* to open a website when the notification is pressed.

```
from windows_toasts import Toast, WindowsToaster

toaster = WindowsToaster('Rick Astley')

newToast = Toast()
newToast.text_fields = ['Hello there! You just won a thousand dollars! Click me to claim␣
↪it!']
# Inline lambda function. This could also be an actual function
newToast.launch_action = 'https://www.youtube.com/watch?v=dQw4w9WgXcQ'

# Send it
toaster.show_toast(newToast)
```

2.3.3 Play different audio

There is a list of available, out-of-the-box audio sources at *windows_toasts.toast_audio.AudioSource*. Lets play the Windows IM sound looping until the notification is dismissed/expires.

```
from windows_toasts import AudioSource, Toast, ToastAudio, WindowsToaster

toaster = WindowsToaster('Windows-Toasts')

newToast = Toast()
newToast.text_fields = ['Ding ding! Ding ding! Ding ding!']
newToast.audio = ToastAudio(AudioSource.IM, looping=True)

toaster.show_toast(newToast)
```

2.3.4 Progress bars

```
from windows_toasts import InteractableWindowsToaster, Toast, ToastProgressBar

toaster = InteractableWindowsToaster('Windows-Toasts')

# progress=None means the bar will be indeterminate
progressBar = ToastProgressBar(
    'Preparing...', 'Python 4 release', progress=None, progress_override='? millenniums␣
↪remaining'
)

newToast = Toast(progress_bar=progressBar)

toaster.show_toast(newToast)
```

2.3.5 Dynamically modifying toast content

You can dynamically modify a toast's progress bar or text field

```
import time
from windows_toasts import InteractableWindowsToaster, Toast, ToastProgressBar

toaster = InteractableWindowsToaster('Python')

newToast = Toast(['Starting.'])
progressBar = ToastProgressBar('Waiting...', progress=0)
newToast.progress_bar = progressBar

toaster.show_toast(newToast)

for i in range(1, 11):
    time.sleep(1)
    progressBar.progress += 0.1
    newToast.text_fields = [f'Stage {i}']

    toaster.update_toast(newToast)

newToast.text_fields = ['Goodbye!']

toaster.update_toast(newToast)
```

From Microsoft.com:

Since Windows 10, you could always replace a notification by sending a new toast with the same Tag and Group. So what's the difference between replacing the toast and updating the toast's data?

Table 1: Update or replace a notification

	Replacing	Updating
Position in Action Center	Moves the notification to the top of Action Center.	Leaves the notification in place within Action Center.
Modifying content	Can completely change all content/layout of the toast	Can only change progress bar and top-level text
Reappearing as popup	Can reappear as a toast popup if you leave *suppress_popup* set to false (or set to true to silently send it to Action Center)	Won't reappear as a popup; the toast's data is silently updated within Action Center
User dismissed	Regardless of whether user dismissed your previous notification, your replacement toast will always be sent	If the user dismissed your toast, the toast update will fail

2.3.6 Scheduled toasts

You can also schedule a toast to display at a specified time

```
from datetime import datetime, timedelta
from windows_toasts import WindowsToaster, Toast

toaster = WindowsToaster('Python')

displayTime = datetime.now() + timedelta(seconds=10)
newToast = Toast([f'This will pop up at {displayTime}'])

toaster.schedule_toast(newToast, displayTime)
```

2.3.7 Snoozing and dismissing

It is possible to snooze toasts and have them pop up later, as well as dismiss the toast entirely

```
from windows_toasts import InteractableWindowsToaster, Toast, ToastSystemButton,␣
↪ToastSystemButtonAction, ToastInputSelectionBox, ToastSelection

newToast = Toast(['Reminder', 'It\'s time to stretch!'])

selections = (ToastSelection('1', '1 minute'), ToastSelection('2', '2 minutes'),␣
↪ToastSelection('5', '5 minutes'))
selectionBox = ToastInputSelectionBox(
    'snoozeBox', caption='Snooze duration', selections=selections, default_
↪selection=selections[0]
)
newToast.AddInput(selectionBox)

snoozeButton = ToastSystemButton(ToastSystemButtonAction.Snooze, 'Remind Me Later',␣
↪relatedInput=selectionBox)
dismissBox = ToastSystemButton(ToastSystemButtonAction.Dismiss)
newToast.AddAction(snoozeButton)
newToast.AddAction(dismissBox)
```

(continues on next page)

(continued from previous page)

```
InteractableWindowsToaster('Python').show_toast(newToast)
```

If you do not provide a caption, Windows will automatically use the appropriate localized strings. If the *relatedInput* is None, the notification will snooze only once for a system-defined time interval. Otherwise, specifying a *ToastInputSelectionBox* allows the user to select a predefined snooze interval.

Note: Ensure the *selection_id* is a positive integer, which represents the interval in minutes.

2.3.8 ...and much more

See *windows_toasts.toast.Toast* or the tests for more modifications you can make to toast notifications.

2.4 Custom AUMIDs

Custom AUMIDs can be used to display user-defined titles and icons. When initialising *InteractableWindowsToaster*, pass the custom AUMID as notifierAUMID.

2.4.1 Installing a custom AUMID

Installing a custom AUMID allows you to use your own title, icon, and listen to activation after the notification was relegated to the action center.

The library comes with a script to implement it, `register_hkey_aumid.py`. The arguments can be understood using the –help argument. If you have the Python Scripts directory in your path, you should be able to execute them by opening the command console and simply executing `register_hkey_aumid`.

2.4.2 Using an installed AUMID

Microsoft.com has a page on finding the Application User Model ID of an installed app. Below are the ways I recommend

Using Powershell

You can use Powershell to view existing AUMIDs.

```
Get-StartApps
```

Will return a table of all applications installed for the current user, with the right row containing AUMIDs for each corresponding name.

Using the registry

1. Open registry editor
2. In the top address bar, paste `HKEY_CURRENT_USER\Software\Classes\ActivatableClasses\Package`
3. Many Microsoft product AUMIDs should be listed, among other third-party programs

2.5 Toasters

2.5.1 Classes

windows_toasts.toasters.ToastNotificationT	Invariant `TypeVar` constrained to `winrt.windows.ui.notifications.ToastNotification` and `winrt.windows.ui.notifications.ScheduledToastNotification`.
windows_toasts.toasters.BaseWindowsToaster(...)	Wrapper to simplify WinRT's ToastNotificationManager
windows_toasts.toasters.WindowsToaster(...)	Basic toaster, used to display toasts without actions and/or inputs.
windows_toasts.toasters.InteractableWindowsToaster(...)	`WindowsToaster`, but uses a AUMID to support actions.

2.5.2 Data

windows_toasts.toasters.**ToastNotificationT = TypeVar(ToastNotificationT, ToastNotification, ScheduledToastNotification)**

Type: `TypeVar`

Invariant `TypeVar` constrained to `winrt.windows.ui.notifications.ToastNotification` and `winrt.windows.ui.notifications.ScheduledToastNotification`.

2.5.3 API

class windows_toasts.toasters.**BaseWindowsToaster**(*applicationText: str*)

Wrapper to simplify WinRT's ToastNotificationManager

Parameters
applicationText – Text to display the application as

notifierAUMID: str | None

toastNotifier: winrt.windows.ui.notifications.ToastNotifier

applicationText: str

show_toast(*toast:* Toast) → None

Displays the specified toast notification. If *toast* has already been shown, it will pop up again, but make no new sections in the action center

Parameters
toast – Toast to display

update_toast(*toast:* Toast) → bool

Update the passed notification data with the new data in the clas

Parameters
toast (Toast) – Toast to update

Returns
Whether the update succeeded

schedule_toast(*toast:* Toast, *displayTime: datetime*) → None

Schedule the passed notification toast. Warning: scheduled toasts cannot be updated or activated (i.e. on_X)

Parameters

- **toast** (Toast) – Toast to display
- **displayTime** (*datetime*) – Time to display the toast on

unschedule_toast(*toast:* Toast) → None

Unschedule the passed notification toast

Raises
ToastNotFoundError: If the toast could not be found

clear_toasts() → None

Clear toasts popped by this toaster

clear_scheduled_toasts() → None

Clear all scheduled toasts set for the toaster

class windows_toasts.toasters.**WindowsToaster**(*applicationText: str*)

Basic toaster, used to display toasts without actions and/or inputs. If you need to use them, see *InteractableWindowsToaster*

Parameters
applicationText – Text to display the application as

show_toast(*toast:* Toast) → None

Displays the specified toast notification. If *toast* has already been shown, it will pop up again, but make no new sections in the action center

Parameters
toast – Toast to display

class windows_toasts.toasters.**InteractableWindowsToaster**(*applicationText: str*, *notifierAUMID: str | None = None*)

WindowsToaster, but uses a AUMID to support actions. Actions require a recognised AUMID to trigger on_activated, otherwise it triggers on_dismissed with no arguments

Parameters

- **applicationText** – Text to display the application as
- **notifierAUMID** – AUMID to use. Defaults to Command Prompt. To use a custom AUMID, see one of the scripts

2.6 Toast

2.6.1 Classes

windows_toasts.toast.ToastInput	Invariant TypeVar constrained to *windows_toasts.wrappers.ToastInputTextBox* and *windows_toasts.wrappers.ToastInputSelectionBox*.
windows_toasts.toast.Toast([text_fields, ...])	

2.6.2 Data

windows_toasts.toast.**ToastInput = TypeVar(ToastInput, ToastInputTextBox, ToastInputSelectionBox)**

Type: TypeVar

Invariant TypeVar constrained to *windows_toasts.wrappers.ToastInputTextBox* and *windows_toasts.wrappers.ToastInputSelectionBox*.

2.6.3 API

class windows_toasts.toast.**Toast**

__init__(*text_fields: list[str | None] | tuple[str | None] | set[str | None] = None*, *audio:* ToastAudio *| None = None*, *duration:* ToastDuration *= ToastDuration.Default*, *expiration_time: datetime | None = None*, *group: str | None = None*, *launch_action: str | None = None*, *progress_bar:* ToastProgressBar *| None = None*, *scenario:* ToastScenario *= ToastScenario.Default*, *suppress_popup: bool = False*, *timestamp: datetime | None = None*, *on_activated: Callable[[*ToastActivatedEventArgs*], None] | None = None*, *on_dismissed: Callable[[winrt.windows.ui.notifications.ToastDismissedEventArgs], None] | None = None*, *on_failed: Callable[[winrt.windows.ui.notifications.ToastFailedEventArgs], None] | None = None*, *actions: Iterable[*ToastButton *| *ToastSystemButton*] = ()*, *images: Iterable[*ToastDisplayImage*] = ()*, *inputs: Iterable[ToastInput] = ()*) → None

Initialise a toast

Parameters

- **actions** (*Iterable[Union[*ToastButton, ToastSystemButton*]]*) – Iterable of actions to add; see *AddAction()*
- **images** (*Iterable[*ToastDisplayImage*]*) – See *AddImage()*
- **inputs** (*Iterable[ToastInput]*) – See *AddInput()*

audio: *ToastAudio* **| None**

The custom audio configuration for the toast

duration: **Literal[ToastDuration.Default, ToastDuration.Long, ToastDuration.Short]**

ToastDuration, be it the default, short, or long

scenario: *ToastScenario*

Scenario for the toast

progress_bar: *ToastProgressBar* | None

An adjustable progress bar for the toast

timestamp: datetime | None

A custom timestamp. If you don't provide one, Windows uses the time that your notification was sent

group: str | None

A generic identifier, where you can assign groups like "wallPosts", "messages", "friendRequests", etc.

expiration_time: datetime | None

The time for the toast to expire on in the action center. If it is on-screen, nothing will happen

suppress_popup: bool

Whether to suppress the toast popup and relegate it immediately to the action center

actions: list[*ToastButton* | *ToastSystemButton*]

List of buttons to include. Implemented through *AddAction()*

images: list[*ToastDisplayImage*]

See *AddImage()*

inputs: list[ToastInput]

Text/selection input boxes

text_fields: list[str | None]

Various text fields

on_activated: Callable[[*ToastActivatedEventArgs*], None] | None

Callable to execute when the toast is clicked if basic, or a button is clicked if interactable

on_dismissed: Callable[[winrt.windows.ui.notifications.ToastDismissedEventArgs], None] | None

Callable to execute when the toast is dismissed (X is clicked or times out) if interactable

on_failed: Callable[[winrt.windows.ui.notifications.ToastFailedEventArgs], None] | None

Callable to execute when the toast fails to display

tag: str

uuid of a tag for the toast

updates: int

Number of times the toast has been updated; mostly for internal use

AddAction(*action:* ToastButton | ToastSystemButton) → None

Add an action to the action list. For example, if you're setting up a reminder, you would use 'action=remindlater&date=2020-01-20' as arguments. Maximum of five.

AddImage(*image:* ToastDisplayImage) → None

Adds an the image that will be displayed on the toast. If using WindowsToaster, a maximum of two (one as the logo and one hero) images will work.

Parameters

image – `ToastDisplayImage` to display in the toast

AddInput(*toast_input: ToastInput*) → None

Adds an input field to the notification. It will be supplied as user_input of type ValueSet in on_activated

Parameters
toast_input – *ToastInput* to display in the toast

property launch_action: str | None

Protocol to launch when the toast is clicked

clone() → *Toast*

Clone the current toast and return the new one

Returns
A deep copy of the toast

Return type
Toast

2.7 Audio

2.7.1 Classes

windows_toasts.toast_audio.AudioSource(value)	Different audios built into Windows
windows_toasts.toast_audio.ToastAudio([...])	Audio configuration in a toast

2.7.2 API

enum windows_toasts.toast_audio.**AudioSource**(*value*)

Different audios built into Windows

Valid values are as follows:

Default = <AudioSource.Default: 'Default'>

IM = <AudioSource.IM: 'IM'>

Mail = <AudioSource.Mail: 'Mail'>

Reminder = <AudioSource.Reminder: 'Reminder'>

SMS = <AudioSource.SMS: 'SMS'>

Alarm = <AudioSource.Alarm: 'Looping.Alarm'>

Alarm2 = <AudioSource.Alarm2: 'Looping.Alarm2'>

Alarm3 = <AudioSource.Alarm3: 'Looping.Alarm3'>

Alarm4 = <AudioSource.Alarm4: 'Looping.Alarm4'>

Alarm5 = <AudioSource.Alarm5: 'Looping.Alarm5'>

Alarm6 = <AudioSource.Alarm6: 'Looping.Alarm6'>

Alarm7 = <AudioSource.Alarm7: 'Looping.Alarm7'>

Alarm8 = <AudioSource.Alarm8: 'Looping.Alarm8'>

Alarm9 = <AudioSource.Alarm9: 'Looping.Alarm9'>

Alarm10 = <AudioSource.Alarm10: 'Looping.Alarm10'>

Call = <AudioSource.Call: 'Looping.Call'>

Call2 = <AudioSource.Call2: 'Looping.Call2'>

Call3 = <AudioSource.Call3: 'Looping.Call3'>

Call4 = <AudioSource.Call4: 'Looping.Call4'>

Call5 = <AudioSource.Call5: 'Looping.Call5'>

Call6 = <AudioSource.Call6: 'Looping.Call6'>

Call7 = <AudioSource.Call7: 'Looping.Call7'>

Call8 = <AudioSource.Call8: 'Looping.Call8'>

Call9 = <AudioSource.Call9: 'Looping.Call9'>

Call10 = <AudioSource.Call10: 'Looping.Call10'>

class windows_toasts.toast_audio.**ToastAudio**(*sound:* AudioSource | *str = AudioSource.Default*, *looping: bool = False*, *silent: bool = False*)

Audio configuration in a toast

Parameters

- **sound** (*Union[AudioSource, str]*) – Selected AudioSource to play
- **looping** (*bool*) – Whether the audio should loop once it ends. Stops abruptly when the notification is dismissed
- **silent** (*bool*) – Silence any audio

sound: *AudioSource* | str = 'Default'

looping: bool = False

silent: bool = False

property sound_value: str

Returns the string value of the selected sound

2.8 Wrappers

2.8.1 Classes

windows_toasts.wrappers.ToastButtonColour(value)	Possible colours for toast buttons
windows_toasts.wrappers.ToastDuration(value)	Possible values for duration to display toast for
windows_toasts.wrappers.ToastImagePosition(value)	Allowed positions for an image to be placed on a toast notification
windows_toasts.wrappers.ToastScenario(value)	Possible scenarios for the toast
windows_toasts.wrappers.ToastSystemButtonAction(value)	
windows_toasts.wrappers.ToastImage(imagePath)	Image that can be displayed in various toast elements
windows_toasts.wrappers.ToastDisplayImage(image)	Define an image that will be displayed as the icon of the toast
windows_toasts.wrappers.ToastProgressBar(status)	Progress bar to be included in a toast
windows_toasts.wrappers.ToastInputTextBox(...)	A text box that can be added in toasts for the user to enter their input
windows_toasts.wrappers.ToastSelection(...)	An item that the user can select from the drop down list
windows_toasts.wrappers.ToastInputSelectionBox(...)	A selection box control, which lets users pick from a dropdown list of options
windows_toasts.wrappers.ToastButton([...])	A button that the user can click on a toast notification
windows_toasts.wrappers.ToastSystemButton(action)	Button used to perform a system action, snooze or dismiss
windows_toasts.events.ToastActivatedEventArgs([...])	Wrapper over Windows' ToastActivatedEventArgs to fix an issue with reading user input

2.8.2 API

class windows_toasts.wrappers.**ToastImage**(*imagePath: str | PathLike*)

Image that can be displayed in various toast elements

__init__(*imagePath: str | PathLike*)

Initialise an *ToastImage* class to use in certain classes. Online images are supported only in packaged apps that have the internet capability in their manifest. Unpackaged apps don't support http images; you must download the image to your local app data, and reference it locally.

Parameters
imagePath (*Union[str, PathLike]*) – The path to an image file

Raises
InvalidImageException: If the path to an online image is supplied

path: str

The URI of the image source

enum windows_toasts.wrappers.**ToastButtonColour**(*value*)

Possible colours for toast buttons

Valid values are as follows:

Default = <ToastButtonColour.Default: ''>

Green = <ToastButtonColour.Green: 'Success'>

Red = <ToastButtonColour.Red: 'Critical'>

enum windows_toasts.wrappers.**ToastDuration**(*value*)

Possible values for duration to display toast for

Valid values are as follows:

Default: str = <ToastDuration.Default: 'Default'>

Short: str = <ToastDuration.Short: 'short'>

Long: str = <ToastDuration.Long: 'long'>

enum windows_toasts.wrappers.**ToastImagePosition**(*value*)

Allowed positions for an image to be placed on a toast notification

Valid values are as follows:

Inline = <ToastImagePosition.Inline: ''>

Hero = <ToastImagePosition.Hero: 'hero'>

AppLogo = <ToastImagePosition.AppLogo: 'appLogoOverride'>

enum windows_toasts.wrappers.**ToastScenario**(*value*)

Possible scenarios for the toast

Valid values are as follows:

Default: str = <ToastScenario.Default: ''>

Alarm: str = <ToastScenario.Alarm: 'alarm'>

Reminder: str = <ToastScenario.Reminder: 'reminder'>

IncomingCall: str = <ToastScenario.IncomingCall: 'incomingCall'>

Important: str = <ToastScenario.Important: 'urgent'>

enum windows_toasts.wrappers.**ToastSystemButtonAction**(*value*)

Valid values are as follows:

Snooze = <ToastSystemButtonAction.Snooze: 0>

Dismiss = <ToastSystemButtonAction.Dismiss: 1>

class windows_toasts.wrappers.**ToastDisplayImage**(*image: ToastImage*, *altText: str | None = None*, *position: ToastImagePosition = ToastImagePosition.Inline*, *circleCrop: bool = False*)

Define an image that will be displayed as the icon of the toast

image: *ToastImage*

An image file

altText: str | None = None

A description of the image, for users of assistive technologies

position: *ToastImagePosition* = ''

Whether to set the image as 'hero' and at the top, or as the 'logo'. Only works on *InteractableWindowsToaster*

circleCrop: bool = False

If the image is not positioned as 'hero', whether to crop the image as a circle, or leave it as is

classmethod fromPath(*imagePath: str | PathLike*, *altText: str | None = None*, *position:* ToastImagePosition *= ToastImagePosition.Inline*, *circleCrop: bool = False*) → *ToastDisplayImage*

Create a *ToastDisplayImage* object from path without having to create *ToastImage*

class windows_toasts.wrappers.**ToastProgressBar**(*status: str*, *caption: str | None = None*, *progress: float | None = 0*, *progress_override: str | None = None*)

Progress bar to be included in a toast

status: str

Status string, which is displayed underneath the progress bar on the left. This string should reflect the status of the operation, like "Downloading…" or "Installing…"

caption: str | None = None

An optional title string

progress: float | None = 0

The percentage value of the progress bar, {0..1}. Defaults to zero. If set to None, it will use an indeterminate bar

progress_override: str | None = None

Optional string to be displayed instead of the default percentage string

class windows_toasts.wrappers.**ToastInputTextBox**(*input_id: str*, *caption: str = ''*, *placeholder: str = ''*)

A text box that can be added in toasts for the user to enter their input

placeholder: str = ''

Optional placeholder for a text input box

class windows_toasts.wrappers.**ToastSelection**(*selection_id: str*, *content: str*)

An item that the user can select from the drop down list

selection_id: str

Identifier for the selection

content: str

Value for the selection to display

class windows_toasts.wrappers.**ToastInputSelectionBox**(*input_id: str*, *caption: str = ''*, *selections: Sequence[*ToastSelection*] = ()*, *default_selection:* ToastSelection | *None = None*)

A selection box control, which lets users pick from a dropdown list of options

selections: Sequence[*ToastSelection*] = ()

Sequence of selections to include in the box

default_selection: *ToastSelection* | None = None

Selection to default to. If None, the default selection will be empty

class windows_toasts.wrappers.**ToastButton**(*content: str = ''*, *arguments: str = ''*, *image:* ToastImage | *None = None*, *relatedInput:* ToastInputTextBox | ToastInputSelectionBox | *None = None*, *inContextMenu: bool = False*, *tooltip: str | None = None*, *launch: str | None = None*, *colour:* ToastButtonColour = *ToastButtonColour.Default*)

A button that the user can click on a toast notification

content: str = ''

The content displayed on the button

arguments: str = ''

String of arguments that the app will later receive if the user clicks this button

image: *ToastImage* | None = None

An image to be used as an icon for the button

relatedInput: *ToastInputTextBox* | *ToastInputSelectionBox* | None = None

An input to position the button besides

inContextMenu: bool = False

Whether to place the button in the context menu rather than the actual toast

tooltip: str | None = None

The tooltip for a button, if the button has an empty content string

launch: str | None = None

An optional protocol to launch when the button is pressed

colour: *ToastButtonColour* = ''

ToastButtonColour for the button

class windows_toasts.wrappers.**ToastSystemButton**(*action:* ToastSystemButtonAction, *content: str = ''*, *relatedInput:* ToastInputSelectionBox | *None = None*, *image:* ToastImage | *None = None*, *tooltip: str | None = None*, *colour:* ToastButtonColour = *ToastButtonColour.Default*)

Button used to perform a system action, snooze or dismiss

action: *ToastSystemButtonAction*

Action the system button should perform

content: str = ''

A custom content string. If you don't provide one, Windows will automatically use a localized string

relatedInput: *ToastInputSelectionBox* | None = None

If you want the user to select a snooze interval, set this to a ToastInputSelectionBox with the minutes as IDs

image: *ToastImage* | None = None

An image to be used as an icon for the button

tooltip: str | None = None

The tooltip for the button

colour: *ToastButtonColour* = ''

ToastButtonColour to be displayed on the button

class windows_toasts.events.**ToastActivatedEventArgs**(*arguments: str | None = None*, *inputs: dict | None = None*)

Wrapper over Windows' ToastActivatedEventArgs to fix an issue with reading user input

arguments: str | None = None

Arguments provided to *AddAction()*

inputs: dict | None = None

Inputs received when using *AddInput()*

classmethod fromWinRt(*eventArgs: winrt.system.Object*) → *ToastActivatedEventArgs*

2.9 Exceptions

2.9.1 Classes

windows_toasts.exceptions.InvalidImageException	The image provided was invalid
windows_toasts.exceptions.ToastNotFoundError	The toast could not be found

2.9.2 API

exception windows_toasts.exceptions.**InvalidImageException**

The image provided was invalid

exception windows_toasts.exceptions.**ToastNotFoundError**

The toast could not be found

exception windows_toasts.exceptions.**UnsupportedOSVersionException**

The operating system version is not supported

2.10 windows_toasts.toast_document

2.10.1 Classes

windows_toasts.toast_document.IXmlType	Invariant TypeVar constrained to winrt.windows.data.xml.dom.IXmlNode and winrt.windows.data.xml.dom.XmlElement.
windows_toasts.toast_document.ToastDocument(toast)	The XmlDocument wrapper for toasts, which applies all the attributes configured in *Toast*

2.10.2 Data

windows_toasts.toast_document.**IXmlType = TypeVar(IXmlType, IXmlNode, XmlElement)**

Type: TypeVar

Invariant TypeVar constrained to winrt.windows.data.xml.dom.IXmlNode and winrt.windows.data.xml.dom.XmlElement.

2.10.3 API

class windows_toasts.toast_document.**ToastDocument**(*toast:* Toast)

The XmlDocument wrapper for toasts, which applies all the attributes configured in *Toast*

xmlDocument: winrt.windows.data.xml.dom.XmlDocument

bindingNode: IXmlType

Binding node, as to avoid having to find it every time

static GetAttributeValue(*nodeAttribute: IXmlType*, *attributeName: str*) → str

Helper function that returns an attribute's value

Parameters

- **nodeAttribute** (*IXmlType*) – Node that has the attribute
- **attributeName** (*str*) – Name of the attribute, e.g. "duration"

Returns
The value of the attribute

Return type
str

GetElementByTagName(*tagName: str*) → IXmlType | None

Helper function to get the first element by its tag name

Parameters
tagName (*str*) – The name of the tag for the element

Return type
IXmlType

SetAttribute(*nodeAttribute: IXmlType*, *attributeName: str*, *attributeValue: str*) → None

Helper function to set an attribute to a node. <nodeAttribute attributeName="attributeValue" />

Parameters

- **nodeAttribute** (*IXmlType*) – Node to apply attributes to
- **attributeName** (*str*) – Name of the attribute, e.g. "duration"
- **attributeValue** (*str*) – Value of the attribute, e.g. "long"

SetNodeStringValue(*targetNode: IXmlType*, *newValue: str*) → None

Helper function to set the inner value of a node. <text>newValue</text>

Parameters

- **targetNode** (*IXmlType*) – Node to apply attributes to
- **newValue** (*str*) – Inner text of the node, e.g. "Hello, World!"

SetAttributionText(*attributionText: str*) → None

Set attribution text for the toast. This is used if we're using *InteractableWindowsToaster* but haven't set up our own AUMID. AttributionText on Microsoft.com

Parameters
attributionText – Attribution text to set

SetAudioAttributes(*audioConfiguration:* ToastAudio) → None

Apply audio attributes for the toast. If a loop is requested, the toast duration has to be set to long. Audio on Microsoft.com

SetTextField(*nodePosition: int*) → None

Set a simple text field. Text elements on Microsoft.com

Parameters
nodePosition – Index of the text fields of the toast type for the text to be written in

SetTextFieldStatic(*nodePosition: int*, *newValue: str*) → None

SetTextField() but static, generally used for scheduled toasts

Parameters

- **nodePosition** – Index of the text fields of the toast type for the text to be written in
- **newValue** – Content value of the text field

SetCustomTimestamp(*customTimestamp: datetime*) → None

Apply a custom timestamp to display on the toast and in the notification center. Custom timestamp on Microsoft.com

Parameters
customTimestamp (*datetime.datetime*) – The target datetime

AddImage(*displayImage:* ToastDisplayImage) → None

Add an image to display. Inline image on Microsoft.com

SetScenario(*scenario:* ToastScenario) → None

Set whether the notification should be marked as important. Important Notifications on Microsoft.com

Parameters
scenario (ToastScenario) – Scenario to mark the toast as

AddInput(*toastInput:* ToastInputTextBox | ToastInputSelectionBox) → None

Add a field for the user to input. Inputs with button bar on Microsoft.com

SetDuration(*duration:* ToastDuration) → None

Set the duration of the toast. If looping audio is enabled, it will automatically be set to long

AddAction(*action:* ToastButton | ToastSystemButton) → None

Adds a button to the toast. Only works on *InteractableWindowsToaster*

AddProgressBar() → None

Add a progress bar on your app notification to keep the user informed of the progress of operations. Progress bar on Microsoft.com

AddStaticProgressBar(*progressBar:* ToastProgressBar) → None

AddProgressBar() but static, generally used for scheduled toasts

2.11 Metadata

```
_version.__title__ = 'Windows-Toasts'

_version.__description__ = 'Python library used to send toast notifications on Windows
machines'

_version.__url__ = 'https://github.com/DatGuy1/Windows-Toasts'

_version.__version__ = '1.1.0'

_version.__author__ = 'DatGuy'

_version.__license__ = 'Apache 2.0'
```

2.12 Migrating from v0.x to v1.0.0

Version 1.0.0 comes with a large backend refactoring and simplification of existing features, along with a few new features. This guide will detail the changes and how to adapt to them.

2.12.1 Replaced winsdk requirement with modularisation

Instead of the 12 MB winsdk release, Windows-Toasts now uses a number of streamlined package to lessen install times and storage requirements.

2.12.2 Toast class simplification

Toasts no longer require a ToastType, but are rather initialised with just *windows_toasts.toast.Toast*.

In addition, all of the SetX methods have been removed in favour of directly modifying the attributes (the AddX methods remain for now). Set[Headline/Body/FirstLine/SecondLine] is now a list named *text_fields*. Instead of using `SetDuration()` and the like, just set it directly: `toast.duration = ToastDuration.Short`.

For instance,

Here is how you would configure toasts before:

```
from windows_toasts import WindowsToaster, ToastDuration

from windows_toasts import ToastText2

toast = ToastText2()

toast.SetHeadline('Hello,')
toast.SetBody('World!')

toast.SetDuration(ToastDuration.Short)
```

(continues on next page)

(continued from previous page)

```
WindowsToaster('Python').show_toast(toast)
```

Here's how you would do it now:

```
from windows_toasts import WindowsToaster, ToastDuration

from windows_toasts import Toast

toast = Toast()

toast.text_fields = ['Hello', 'World!']
# Or, directly, toast = Toast(['Hello', 'World!'])

toast.duration = ToastDuration.short

WindowsToaster('Python').show_toast(toast)
```

and here's the highlighted difference between the two:

```
from windows_toasts import WindowsToaster, ToastDuration

- from windows_toasts import ToastText2
+ from windows_toasts import Toast

- toast = ToastText2()
+ toast = Toast()

- toast.SetHeadline('Hello,')
- toast.SetBody('World!')
+ toast.text_fields = ['Hello', 'World!']

- toast.SetDuration(ToastDuration.Short)
+ toast.duration = ToastDuration.short

WindowsToaster('Python').show_toast(toast)
```

2.12.3 New features

Release v1.0.0 also arrives with a few new features

Launching through protocols

For applications that support protocols, you can now make your toasts and buttons launch that protocol directly.

```
from windows_toasts import InteractableWindowsToaster, Toast, ToastButton

protocol_toast = Toast(['Click the toast to launch google.com', 'or, alternatively'],␣
↪launch_action='https://google.com')

bing_button = ToastButton('Launch Bing', launch='https://bing.com')
```

(continues on next page)

(continued from previous page)

```
baidu_button = ToastButton('Launch Baidu', launch='https://baidu.com')

protocol_toast.AddAction(bing_button)
protocol_toast.AddAction(baidu_button)

InteractableWindowsToaster('Browser Launcher').show_toast(protocol_toast)
```

Note: Web browsers are not the only thing you can launch with protocols. Set *windows_toasts.wrappers.ToastButton.launch* to `spotify:playlist:37i9dQZEVXbMDoHDwVN2tF` to launch the Spotify client on the global Top 50, set it to `steam://friends/status/offline` to set yourself offline on the Steam client, et cetera. You can also launch files by entering their path.

Inline images

Images have been reworked, with the *windows_toasts.wrappers.ToastImagePosition* enum introducted as to make it possible to display more than two.

```
# Downloads the Python logo
import urllib.request
from pathlib import Path

# Save the image to python.png
image_url = 'https://www.python.org/static/community_logos/python-powered-h-140x182.png'
image_path = Path.cwd() / 'python.png'
urllib.request.urlretrieve(image_url, image_path)

from windows_toasts import InteractableWindowsToaster, Toast, ToastDisplayImage,␣
↪ToastImage, ToastImagePosition
toast_image_python = ToastImage(image_path)

toast_images = [
    ToastDisplayImage(toast_image_python, position=ToastImagePosition.Hero),
    ToastDisplayImage(toast_image_python, position=ToastImagePosition.AppLogo),
    ToastDisplayImage(toast_image_python, position=ToastImagePosition.Inline),
    ToastDisplayImage(toast_image_python, position=ToastImagePosition.Inline)
]
new_toast = Toast(text_fields=['Hiss!'], images=toast_images)

InteractableWindowsToaster('Python').show_toast(new_toast)
```

System actions

There is a writeup on how to use the snooze and dismiss system actions in the *Snoozing and dismissing* section

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

U

W

X