Inthing Documentation

Release 0.1.4

Will McGugan

May 02, 2016

Contents

Inthing is a Python interface to https://www.inthing.io, a realtime events service.

Contents:

CHAPTER 1

Introduction

Inthing is a Python module for https://inthing.io.

Inthing users create streams of events, which are much like a social media timelines in that they can contain text, rich-media, comments etc., but unlike tranditional social media, Inthing streams are intended primarily for machines to post to via a easy to use API.

How a stream is used and what kind of events are posted to it is entirely up to the developer. It can be used for something as sophisticated as a motion activated camera, or as simple as a one-liner to send a text notification.

1.1 Getting Started

Use PIP to install the Python module:

```
pip install inthing
```

You may need to prefix that with `sudo` on some systems.

This will install the `inthing` Python module, and also the `inthing` command line app. Run the following from the command line to check instalation:

```
inthing -v
```

You will probably also want to create an inthing.io account, which you can do by visiting https://inthing.io. Creating an account is *not* a requirement, but will give you more control over your streams.

1.2 Demo

Here's a very simple example that generates an ascii mandlebrot set and posts it to a new Inthing stream:

```
# demo.py
"""An example of posting a text event to a stream."""

from inthing import Stream


def mandel(xsize=80, ysize=20, max_iteration=50):
    """Render an ascii mandelbrot set!"""
    chars = " .,~:;+*%@##"
    rows = []
    for pixy in xrange(ysize):
        y0 = (float(pixy) / ysize) * 2 - 1
        row = ""
        for pixx in xrange(xsize):
            x0 = (float(pixx) / xsize) * 3 - 2
            x = 0
            y = 0
            iteration = 0
            while (x * x + y * y < 4) and iteration < max_iteration:
                xtemp = x * x - y * y + x0
                y = 2 * x * y + y0
                x = xtemp
                iteration += 1
            row += chars[iteration % 10]
        rows.append(row)
    return "```\n{}\n```\n#mandlebrot".format('\n'.join(rows))


stream = Stream.new()
result = stream.text(mandel(), title="Mandelbrot Set!")
result.browse()
```

CHAPTER 2

Streams

A stream is a URL on inthing.io with a list of events. You can post to a stream with an instance of a `inthing.stream.Stream` class.

2.1 Creating Stream Objects

To create a stream object for an existing stream, pass in the URL of the stream as to the Stream constructor as follows:

```
>>> from inthing import Stream
>>> my_stream = Stream('https://www.inthing.io/will/test/')
```

Note: You can drop the `https://www.inthing.io/` part and just use `/will/test/` if you prefer.

Most streams will be protected with a password, which you can pass in with the *password* parameter:

```
>>> my_stream = Stream('https://www.inthing.io/will/test/', password="daffodil")
```

Streams also have a UUID (universally unique identifier), which you can use in place of a URL. Here's an example:

```
>>> my_stream = Stream('4d9f242c-0805-11e6-8d28-fb90f1f704e0', password="daffodil")
```

Using a UUID has the advantage that it will never change, whereas the URL might, if you change the title.

If you store the stream ID and password in the environments variables `INTHING_STREAM` and `INTHING_STREAM_PASSWORD`, you can omit the arguments from the Stream constructor entirely.

For example, in Linux and OSX, you can do the following from the terminal:

```
export INTHING_STREAM=4d9f242c-0805-11e6-8d28-fb90f1f704e0
export INTHING_STREAM_PASSWORD=daffodil
```

Then in Python, you can create a stream object as follows:

```
from inthing import Stream
my_stream = Stream()
```

Note: Environment variables also work with Windows, but the process differs from version to version. Ask Google for instructions.

2.2 Unclaimed Streams

An alternative to creating a stream object for an existing stream, is to call `inthing.Stream.new()`, which creates a new stream on the server. Here's an example:

```
>>> from inthing import Stream
>>> my_stream = Stream.new()
>>> print(my_stream.url)
```

Stream objects created in this way are *unclaimed*, in that they aren't owned by anyone. You will not see the unclaimed stream in your timeline – but you may still post events to it as normal. If you would like to *claim* the stream, you should visit the stream's page and click the Claim button. This will attach the stream to your account.

2.3 Stream Objects

Streams are relatively simple objects. For most applications you will only ever need a single Stream, which you may use to post events.

Streams have the following public attributes:

- `stream.generator` A string the identifies the entitiy creating events. The default value for this is the hostname of the computer it is running on. (read/write)
- `stream.id` The UUID of the stream. (read only)
- `stream.url` The full URL to the stream. (read only)

The `inthing.Stream.browse()` method will open a webbrowser the stream page. This does the equivelent of opening a browser and navigating to the URL in `stream.url`.

The following public methods on stream objects are devoted to posting various kinds of events:

- `inthing.Stream.code()`
- `inthing.Stream.text()`
- `inthing.Stream.image()`
- `inthing.Stream.screenshot()`

See *Events* for details.

2.4 Rate Limiting

The inthing.io server imposes a *rate limit* on requests; if you add events too rapidly, the Stream object will throw a `inthing.errors.RateLimited` exception for new events. If this happens, you can wait a while and try again.

This is just a precaution against errors in your code from overloading the server by posting too quickly. The rate limit is high enough that you are unlikely to reach it with functioning code. The server allows for *bursts* of events as long as the number of events averages out within a longer period.

The exact limits are subject to change, but as a general rule, if events are being posted too fast for an average human to read, you may be rate limited.

CHAPTER 3

Events

Inthing supports a variety of *event types*, which display different kinds of information (text, code, images etc) on your stream.

When you successfully post an event with one of the methods in `inthing.Stream`, you will get back a `inthing.Result` object, which contains a `url` attribute, and a `inthing.Result.browse()` method.

Here's an example of using the `Result` object:

```
from inthing import Stream
stream = Stream.new()
result = stream.text('my first event!')
print("opening {}".format(result.url))
result.browse()
```

See Event Types for a description of the event types.

3.1 Priorities

Events have an associated priority value which is an integer between -2 and +2 (inclusive).

Table 3.1: Priority values

Value	Meaning
+2	Urgent
+1	Important (of greater than normal signficance)
0	Normal priority (default)
-1	Informative
-2	Verbose

3.2 Markup

Most events will have a description field. You can set how the description should be displayed via the *markup* parameter.

Table 3.2: Supported markups

Markup	Meaning
text	Simple text
markdown	Markdown.
html	Simple HTML
bbcode	BBCode.

Note: Inthing.io will strip descriptions of potentially dangerous markup, such as <script> tags.

3.3 Capturing Output

Inthing can automatically *capture* output from your Python script or application. It does this by hooking in to your apps *standard output* and *standard error*, i.e. anything you print to the terminal.

Here's an example:

```
from inthing import Stream
stream = Stream.new()
with stream.capture(title="Capture all the things!") as capture:
    print('Anything you print will get captured...')
capture.browse()
```

In the above code, any print statement inside the `with` blog will be captured. The text will still appear in your terminal, but when the `with` block exists, inthing will upload a new event.

CHAPTER 4

Event Types

The following is a description of the event types you can post to a stream.

4.1 Text

A *text* event may contain text for pretty much any purpose.

The following is an example that posts text messages to a stream (Ctrl+C to exit):

```
from inthing import Stream
stream = Stream.new()
stream.browse()
while 1:
    stream.text(raw_input('type something: '))
```

You could adapt this quite easily to create a realtime chat system on the web.

The `text` method also has a `markup` parameter which sets the markup for the text. By default it is markdown, which means you can easily insert formatting. Here's an example:

```
stream.text('**Bold** and *italic*')
```

You can also set the `markup` parameter to `text`, `bbcode` or `html`. But note that Inthing.io will strip out any potentially dangerous HTML markup (so no script tags)!

See *`inthing.Stream.screenshot()`* for details.

4.2 Code

A *code* event contains source code which you can share and comment on. If you have a piece of code you are particularily proud of, you can post it to Inthing.io. It will be nicely syntax highlighted. A variety of languages are supported.

Here's how you might post source code to a stream:

```
my_stream.code('cool.py', language="python", title="I wrote cool.py")
```

See *`inthing.Stream.code()`* for details.

4.3 Image

An *image* event contains an image, typically a photo.

Here's how you would post the file `alien1.jpg`:

```
my_stream.image('./alien1.jpg', description="Alien Autopsy!")
```

See `inthing.Stream.image()` for details.

4.4 Screenshots

A *screenshot* event is a special kind of image event that contains a screenshot. Calling `inthing.Stream.screenshot()` will capture a screenshot of your desktop and add the event to your Stream.

Here's how you would upload a screenshot after 5 seconds:

> my_stream.screenshot(title="My Desktop!", delay=5)

Warning: Be careful with this event, you wouldn't want to screenshot any passwords or nuclear launch codes!

See `inthing.Stream.screenshot()` for details.

Command Line App

The inthing app is installed with the Python module and can be used to post events directly to a stream via the command line. To check if it is available, run the following:

```
inthing -v
```

You can specify the stream credentials via the environement variables `INTHING_STREAM` and `INTHING_STREAM_PASSWORD`. These can be the same values as used by the the Python *`inthing.Stream`* object. Here's an example (Linux or OSX):

```
export INTHING_STREAM=https://www.inthing.io/will/test/
export INTHING_STREAM_PASSWORD=daffodil
```

You can then one of the available subcommands supported by `inthing`. The following example posts a simple text event:

```
inthing text "Hello, World!" --title "Test Event"
```

An alternative to using environment variables is to specify the stream and password on the command line as well. Here's an example:

```
inthing text "Hello, World!" --title "Test Event" --id https://www.inthing.io/will/test/ --password
```

To see the full range of subcommands available, run the following:

```
inthing -h
```

5.1 Capturing Output

The `capture` subcommand can be used to capture output from a command and post an event to your stream. Simply *pipe* a command in to `inthing captue`. Here's an example:

```
uptime | inthing capture
```

This will add a text event containing the output of the `uptime` command.

If you add the `--browse` switch it will open your browser at the new event, or added the `--url` to just display the URL.

CHAPTER 6

Code

6.1 Inthing module

class `inthing.`**`Result`**(*url*)

Represents a successfully posted event.

The `url` attribute contains the URL to the the event's page.

`browse`()

Open a webbrowser at this event.

class `inthing.`**`Stream`**(*id=None*, *password=None*, *generator=None*, *silence_errors=False*)

Inthing Stream class interface.

This class is the main interface for posting to an Inthing Stream. It represents a single Stream, and has methods to post the various event types.

Construct a new Stream object.

Parameters

- **`id`** (*`str`*) – The ID of your stream. This may be either a UUID (mixture of letters an numbers), or the URL of the stream.
- **`password`** (*`str`*) – The stream's password, if required. It's possible for a stream to have no password, which allows for anyone to post to it.
- **`generator`** (*`str`*) – A short string to identify what is creating events.
- **`silence_errors`** (*`bool`*) – If True, ignore errors when posting events.

Return type *Stream*

`add_event`(*event*)

Add an event.

Parameters **`event`** (`Event`) – Event you want to add.

`browse`()

Open this stream in your browser.

`capture`(*title=u'Captured output'*, *description=None*, *browse=False*, *stdout=True*, *stderr=False*)

Capture stdout and stderr.

Parameters

- **`title`** (*`str`*) – Title of the captured event

- **description** (*str*) – Optional description
- **browse** (*bool*) – Open a webbrowse to new event?
- **stdout** (*bool*) – Capture stdout?
- **stderr** (*bool*) – Capture stderr?

This method returns a context manager, which will capture `print` output, and post it to a stream. Here is an example:

```
from inthing import Stream
stream = Stream.new()
with stream.capture(title="Capture Example") as capture:
    print('This output will go to a stream!')
capture.browse()
```

code (*code*, *language=None*, *description=None*, *title=u'Code'*, *markup=u'markdown'*, *priority=0*)
Add a (source) code event.

```
with open('example.py') as code_file:
    stream.code(code_file, language="Python")
```

Parameters

- **code** (*str or open object*) – Path to a file containing code, or an open file object
- **language** (*str*) – Programming language.
- **description** (*str*) – A description of the source dode.
- **title** (*str*) – Title of the Event.
- **markup** (*str*) – Markup type for the description.

Return type *Result*

image (*path*, *description=None*, *title=u'New Photo'*, *markup=u'markdown'*, *priority=0*)
Add an image event.

Parameters

- **path** (*str*) – A path to a JPG or PNG.
- **description** (*str*) – Description of the image.
- **title** (*str*) – Title of the image event.
- **markup** (*str*) – Markup used to render description.

Return type *Result*

classmethod new()
Create a new unclaimed stream.

An *unclaimed* stream functions like any other stream, but is not owned by any user, and has no password set. Anyone may post events to an unclaimed stream, but in practice they are private as long as you don't give away the ID or URL.

Rtype Stream

```
>>> my_stream = Stream.new()
>>> my_stream.text('I just create a new stream!')
>>> print(my_stream.url)
```

screenshot (*delay=0, description=None, title=u'New Screenshot', markup=u'markdown', priority=0*)
Take a screenshot and post an event.

Parameters

- **delay** (*int*) – Number of seconds to wait before taking the screenshot.
- **description** (*str*) – Description of the screenshot.
- **title** (*str*) – Title of the event.
- **markup** (*str*) – Markup of the description.

Return type *Result*

text (*text, title=u'Text', markup=u'markdown', priority=0*)
Add a text event to this stream.

```
>>> my_stream = Stream.new()
>>> result = my_stream.text('Hello, World!')
>>> result.browse()
```

Parameters

- **text** (*str*) – The text you want to associate with this event.
- **title** (*str*) – The title of the event

Return type *Result*

class inthing.**Event** (*title=u'New Event', type=u'text', priority=0, markup=u'markdown', description=None, text=None, generator=None, code_language=None*)
Contains the details of a new event.

This class provides a more finely grained way of creating events. To use, construct an Event instance and add it to a stream with *inthing.Stream.add_event()*.

Store details for a single event.

This class is used in the low level interface for creating events. You probably want to used the methods on *inthing.Stream* for a simpler interface.

Here's an example of using this class:

```
event = Event(title="Example text event", text="Hello, World!")
stream.add_event(event)
```

Parameters

- **title** (*str*) – The title of the event, shown as a header.
- **type** (*str*) – The type of the event.
- **markup** (*str*) – The markup used to render the descriptions, may be 'text', 'html', 'markdown'
- **text** (*str*) – Text associated with the event.
- **generator** (*str*) – Text regarding what generated the event.
- **code_language** (*str*) – The language used to syntax highlight code events.

add_image (*path*)
Add an image to the event.

Parameters **`path`** (*`str`*) – Path to an image file

exception `inthing.errors.`**`BadResponse`**
The server returned an unexpected response.

exception `inthing.errors.`**`ConnectivityError`**
There was an issue reaching the server.

exception `inthing.errors.`**`EventError`**
An error relating to an attempt to post an event.

exception `inthing.errors.`**`EventFail`**(*result*)
The event contained invalid information.

`print_error`()
Print error details to stderr.

exception `inthing.errors.`**`RateLimited`**
Events are being posted to fast.

exception `inthing.errors.`**`StreamError`**
Base class for inthing Stream related errors.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index