Tingbot Documentation

Release 1.2.1

Joe Rickerby

May 29, 2017

Contents

Contents:

CHAPTER 1

Tingbot

Python APIs to write apps for Tingbot.

- Documentation: http://tingbot-python.readthedocs.org/.

CHAPTER 2

Graphics

Drawing

`screen.fill(color)`

Fills the screen with the specified `color`.

The color option can be specified using a name (e.g. 'white', 'black'), or an RGB triple.

Listing 2.1: Example: fill the screen with white

```
screen.fill(color='white')
```

Listing 2.2: Example: fill the screen with red

```
screen.fill(color=(255, 0, 0))
```

`screen.text(`*string...*, *xy=...*, *color=...*, *align=...*, *font=...*, *font_size=...*, *max_width=...*, *max_lines=...*, *max_height=...*`)`

Draws text `string`.

`xy` is the position that the text will be drawn.

The align option is one of:

topleft, left, bottomleft, top, center, bottom, topright, right, bottomright

If a custom font is used, it must be included in the tingapp bundle.

Listing 2.3: Example: Write 'Hello world' in black on the screen

```
screen.text('Hello world!', color='black')
```

Listing 2.4: Example: changing the alignment

```
screen.text('Hello world!', xy=(20,20), color='black', align='topleft')
```

Listing 2.5: Example: Using a custom font

```
screen.text('Hello world!', color='black', font='Helvetica.ttf')
```

Listing 2.6: Example: Changing the text size

```
screen.text('Hello world!', color='black', font_size=50)
```

Listing 2.7: Example: Confining text to a single line

```
screen.text('Lorem ipsum dolor sit amet, consectetur adipiscing elit!', color=
↪'black', max_lines=1)
```

Listing 2.8: Example: Confining text to two lines

```
screen.text('Lorem ipsum dolor sit amet, consectetur adipiscing elit!', color=
↪'black', max_width=300, max_lines=2)
```

screen.**rectangle**(*xy=…*, *size=…*, *color=…*, *align=…*)

Draws a rectangle at position xy, with the specified size and color.

Align is one of

topleft, left, bottomleft, top, center, bottom, topright, right, bottomright,

Listing 2.9: Example: Drawing a red square

```
screen.rectangle(xy=(25,25), size=(100,100), color=(255,0,0))
```

Listing 2.10: Example: Drawing centered

```
screen.rectangle(xy=(160,120), size=(100,100), color=(255,0,0), align='center')
```

screen.**image**(*filename…*, *xy=…*, *scale=…*, *align=…*, *max_width=…*, *max_height=…*, *raise_error=True*)

Draws an image with name filename at position xy. If filename is a URL (e.g. http://example.com/cats.png) then it will attempt to download this and display it.

Images can be animated GIFs. Make sure to draw them in a loop() function to see them animate.

Scale is a number that changes the size of the image e.g. scale=2 makes the image bigger, scale=0.5 makes the image smaller. There are also special values 'fit' and 'fill', which will fit or fill the image according to `max_width` and `max_height`.

Align is one of

topleft, left, bottomleft, top, center, bottom, topright, right, bottomright

If raise_error is True then any errors encountered while opening or retrieving the image will cause an exception. If it is False, then if there is an error a "file not found" icon will be displayed instead

Listing 2.11: Example: Drawing an Image

```
screen.image('tingbot.png', xy=(25,25))
```

Listing 2.12: Example: Drawing an Image from a URL

```
screen.image('http://i.imgur.com/xbT92Gm.png')
```

`screen.line` (*start_xy=...*, *end_xy=...*, *color=...*, *width=...*)
Draws a line between `start_xy` and `end_xy`.

Screen

The screen supports all the methods above, and some extras below.

`screen.update` ()
After drawing, this method should to be called to refresh the screen. When drawing in a `draw()` or `loop()` function, this is called automatically, but when drawing in a tight loop, e.g. during a calculation, it can called manually.

Listing 2.13: Example: An app without a run loop - calling `screen.update()` manually

```
import tingbot
from tingbot import *

screen.fill(color='black')

# pump the main run loop just once to make sure the app starts
tingbot.input.EventHandler().poll()

frame_count = 0

while True:
    screen.fill(color='black')
    screen.text(frame_count)
    screen.update()
    frame_count += 1
```

`screen.brightness`
The brightness of the screen, between 0 and 100.

Listing 2.14: Example: Dimming the screen

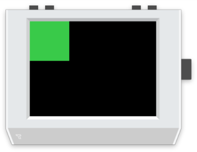
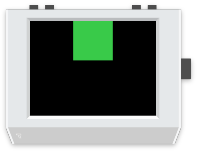
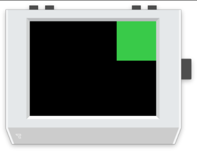
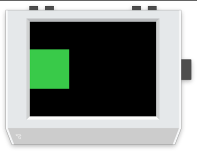
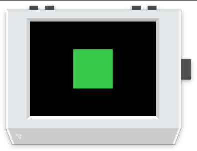
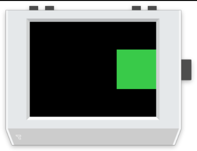
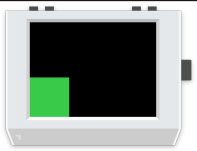
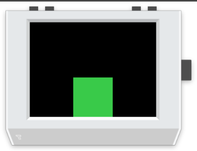
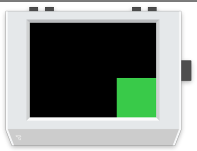
```
screen.brightness = 25
```

Listing 2.15: Example: Brightness test app

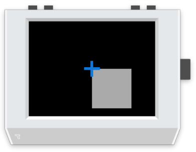
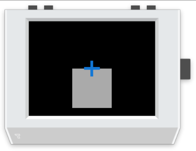
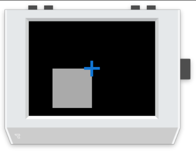
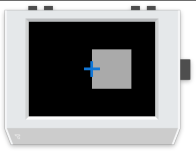
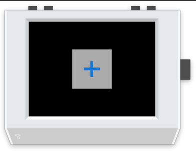
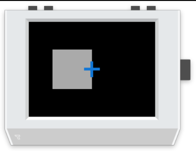
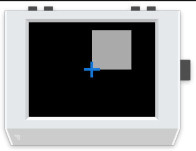
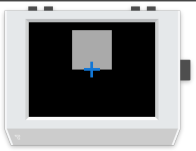
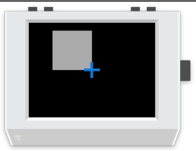
```
import tingbot
from tingbot import *

state = {'brightness': 0}

def loop():
    screen.brightness = state['brightness']

    screen.fill(color='black')
    screen.text('Brightness\n %i' % state['brightness'])

    state['brightness'] += 1
    if state['brightness'] > 100:
        state['brightness'] = 0

tingbot.run(loop)
```

The `align` option

When used without the `xy` parameter, the item is positioned relative to the screen/drawing surface.

Setting	Screenshot	Code
topleft		`screen.rectangle(color='green', align='topleft')`
top		`screen.rectangle(color='green', align='top')`
topright		`screen.rectangle(color='green', align='topright')`
left		`screen.rectangle(color='green', align='left')`
center		`screen.rectangle(color='green', align='center')`
right		`screen.rectangle(color='green', align='right')`
bottomleft		`screen.rectangle(color='green', align='bottomleft')`
bottom		`screen.rectangle(color='green', align='bottom')`
bottom-right		`screen.rectangle(color='green', align='bottomright')`

When used with the `xy` parameter, it positions the item relative to the `xy` point.

Setting	Screenshot	Code
topleft		`screen.rectangle(xy=(160, 120), align='topleft')`
top		`screen.rectangle(xy=(160, 120), align='top')`
topright		`screen.rectangle(xy=(160, 120), align='topright')`
left		`screen.rectangle(xy=(160, 120), align='left')`
center		`screen.rectangle(xy=(160, 120), align='center')`
right		`screen.rectangle(xy=(160, 120), align='right')`
bottomleft		`screen.rectangle(xy=(160, 120), align='bottomleft')`
bottom		`screen.rectangle(xy=(160, 120), align='bottom')`
bottom-right		`screen.rectangle(xy=(160, 120), align='bottomright')`

The `color` option

The color option can be either an RGB value, or predefined color name.

RGB values

RGB values (as a tuple), like `(255, 128, 0)`.

Predefined colors

We also have a set of default colors, referred to by their name, as a string.

Thanks to http://clrs.cc for the color scheme!

CHAPTER 3

Touch

Your Tingbot comes equipped with a resistive touch screen! It's easy to react to touch events.

Listing 3.1: Example: Simple drawing app

```
import tingbot
from tingbot import *

screen.fill(color='black')

@touch()
def on_touch(xy):
    screen.rectangle(xy=xy, size=(5,5), color='blue')

tingbot.run()
```

This is a simple drawing app. It uses the `@touch()` decorator to receive touch events and draws a rectangle to the screen at the same place.

`@touch`(*xy=. . .*, *size=. . .*, *align=. . .*)

This 'decorator' marks the function after it to receive touch events.

You can optionally pass an area that you're interested in, using the `xy`, `size` and `align` arguments. If you specify no area, you will receive all touch events.

The handler function can optionally take the arguments `xy` and `action`. `xy` is the location of the touch. `action` is one of 'down', 'move', 'up'.

Listing 3.2: Example: Simple Drawing app code

```
@touch()
def on_touch(xy):
    screen.rectangle(xy=xy, size=(5,5), color='blue')
```

Listing 3.3: Example: Making a button do something

```
@touch(xy=(0,0), size=(100,50), align='topleft')
def on_touch(xy, action):
    if action == 'down':
        state['screen_number'] = 2
```

CHAPTER 4

Buttons

There are four buttons on the top of the Tingbot. These can be used in programs to trigger functions in your code.

Listing 4.1: Example: Score-keeping app.

```python
import tingbot
from tingbot import *

state = {'score': 0}

@left_button.press
def on_left():
    state['score'] -= 1

@right_button.press
def on_right():
    state['score'] += 1

def loop():
    screen.fill(
        color='black')
    screen.text(
        state['score'],
        color='white')

tingbot.run(loop)
```

This is a simple counter program. Whenever the right button is pressed, the score goes up by one. On the left button, the score goes down.

`@left_button.press`

`@midleft_button.press`

`@midright_button.press`

`@right_button.`**`press`**

This 'decorator' marks the function to be called when a button is pressed.

`button` can be one of: left_button, midleft_button, midright_button, right_button.

Listing 4.2: Example: Button handler

```
@left_button.press
def on_left():
    state['score'] -= 1
```

Listing 4.3: Example: Button handler for all buttons

```
@left_button.press
@midleft_button.press
@midright_button.press
@right_button.press
def on_button():
    state['score'] -= 1
```

Only presses shorter than a second count - anything longer counts as a 'hold' event.

`@Button.`**`hold`**

This marks the function to be called when a button is held down for longer than a second.

Listing 4.4: Example: Reset button handler

```
@left_button.hold
def reset_score():
    state['score'] = 0
```

`@Button.`**`down`**

This marks the function to be called as soon as a button is pushed down. This could be the start of a 'press' or a 'hold' event.

This one is useful for games or when you want the button to be as responsive as possible.

Listing 4.5: Example: Reset button handler

```
@right_button.down
def jump():
    dude.jump()
```

`@Button.`**`up`**

This marks the function to be called when a button is released.

Listing 4.6: Example: Down/up handler pair

```
@right_button.down
def down():
    state['button_is_down'] = True

@right_button.up
def up():
    state['button_is_down'] = False
```

`@button.`**`combo`**`(`*buttons...*`)`

This marks the function to be called when some buttons are pressed at the same time.

You can give it as many buttons as you like and `combo` will call the function when all the buttons are pressed together.

Listing 4.7: Example: Combo to dim/wake the screen

```python
@button.combo(left_button, right_button)
def screen_dim():
    if screen.brightness == 100:
        screen.brightness = 0
    else:
        screen.brightness = 100
```

CHAPTER 5

Webhooks

You can push data to Tingbot using webhooks.

Here is an example that displays SMS messages using If This Then That. See our tutorial video to see how to set up IFTTT with webhooks.

```
import tingbot
from tingbot import *

screen.fill(color='black')
screen.text('Waiting...')

@webhook('demo_sms')
def on_webhook(data):
    screen.fill(color='black')
    screen.text(data, color='green')

tingbot.run()
```

@**webhook** (*webhook_name...*)

This decorator calls the marked function when a HTTP POST request is made to the URL `http://webhook.tingbot.com/`*`webhook_name`*. To avoid choosing the same name as somebody else, you can add a random string of characters to the end.

The POST data of the URL is available to the marked function as the `data` parameter. The data is limited to 1kb, and the last value that was POSTed is remembered by the server, so you can feed in relatively slow data sources.

You can use webhooks to push data to Tingbot, or to notify Tingbot of an update that happened elsewhere on the internet.

Hint: IFTTT is a great place to start for ideas for webhooks. Slack also has native support for webhooks!

CHAPTER 6

Settings

You can store local data on the tingbot. Simply use `tingbot.app.settings` as a dict. This will store any variables you like on a file in the application directory (called local_settings.json). This is stored in JSON format.

```
import tingbot

#store an item
tingbot.app.settings['favourite_colour'] = 'red'

#local_settings.json on disk now contains: {"favourite_colour":"red"}

#retrieve an item
tingbot.screen.fill(tingbot.app.settings['favourite_colour'])
```

Any item that can be stored in JSON can be used in tingbot.app.settings - so strings, ints, floats, even dicts and lists can be used.

Note: Take care when changing the insides of dicts or lists that are stored in tingbot.app.settings, as your changes will not be saved automatically.

You can force a save by calling *tingbot.app.settings.save()*

```
import tingbot

# create a sub-dictionary
tingbot.app.settings['ages'] = {'Phil': 39, 'Mabel': 73}

# local_settings.json on disk now contains: {"ages":{"Phil":39,"Mabel":73}}

tingbot.app.settings['ages']['Barry'] = 74

# warning: local_settings.json has not been updated because you haven't directly␣
↪changed tingbot.app.settings

tingbot.app.settings.save()
```

```
# now local_settings.json on disk now contains: {"ages":{"Phil":39,"Mabel":73,"Barry
↪":74}}
```

Storage

There are three settings files, that have different uses:

- `default_settings.json` When writing your app, you can put default values for your settings in this file.
- `settings.json` Somebody who's downloaded your app can create this file in Tide to fill in some settings before uploading to Tingbot. This file should be 'gitignored' so it's not shared when the app is copied, and can contain secrets like API keys or passwords.
- `local_settings.json` When code within the app sets a setting, it's stored in this file. This prevents the app from overwriting data from the previous two files. This shouldn't be copied with an app and should be 'gitignored' too.

When the first setting is accessed the app loads each file in turn, so values in 'local_settings' override those in 'settings', which override those is 'default_settings'.

CHAPTER 7

Run loop

Tingbot has an internal run loop that it uses to schedule events.

`tingbot.`**`run`**`(`*loop=None*`)`
: This function starts the run loop.

  The optional `loop` function is called every 1/30th seconds.

`@`**`every`**`(`*hours=0*, *minutes=0*, *seconds=0*`)`
: This decorator will call the function marked periodically, according to the time specified.

Listing 7.1: Example: Refreshing data every 10 minutes

```
@every(minutes=10)
def refresh_data():
    r = requests.get('http://api.openweathermap.org/data/2.5/weather?q=London,uk&
↪appid=bd82977b86bf27fb59a04b61b657fb6f')
    state['data'] = r.json()
```

`@`**`once`**`(`*hours=0*, *minutes=0*, *seconds=0*`)`
: This decorator will call the function marked once, after the duration specified.

`tingbot.RunLoop.`**`call_after`**`(`*callable*`)`
: Call function `callable` at the next possible moment from the run loop. This allows threads to communicate with the main run loop in a thread-safe fashion

CHAPTER 8

Hardware

There are several useful functions that can be used to see if hardware is connected to the tingbot.

`get_ip_address``()`
Returns the IP address of the tingbot or None if it is not connected.

`get_wifi_cell``()`
Returns a WifiCell object (or None if there is no wifi adapter).

A WifiCell object has the following attributes:

- ssid
- link_quality
- signal_level

`mouse_attached``()`
Returns True if a mouse is attached

`keyboard_attached``()`
Returns True if a keyboard is attached

`joystick_attached``()`
Returns True if a joystick is attached

CHAPTER 9

Sounds

If you plug in a USB audio device into your Tingbot, you can play sounds.

Listing 9.1: Example: Simple sounds app

```python
import tingbot
from tingbot import *

sound = Sound('car_chirp.wav')

@left_button.press
def on_press():
    sound.play()

@every(seconds=1.0/30)
def draw():
    screen.fill(color='black')
    screen.text('Press the left button to play a sound.')

tingbot.run()
```

class **`Sound`**(*filename*)

Loads a sound ready for playing. Currently WAV and OGG files are supported.

`play`(*loop=False*)

Starts the playback of the sound.

Parameters **`loop`** (*`bool`*) – Pass `True` to loop the sound until `stop()` is called.

`stop`()

Stops the sound.

CHAPTER 10

Full reference

CHAPTER 11

Indices and tables

- search

Index