blighty Documentation

Release 2.0.0

Gabriele N. Tornetta

Nov 22, 2018

Contents:

1 The `blighty` project **3**

2 Usage **5**

3 A clock widget **7**

4 Issues **9**

5 The brag corner **11**
5.1 blighty package 11

Python Module Index **19**

- genindex
- modindex
- search

CHAPTER 1

The blighty project

This is the documentation for blighty, a Python package for the creation of widgets for the Linux desktop. The idea is to replicate the wonders of conky, but with Python support instead of Lua.

CHAPTER 2

Usage

Using `blighty` is very simple and should come quite natural to you, especially if you already have experience with Cairo from conky.

All the you need to do is extend one of the `Canvas` classes provided by the package (e.g. *`blighty.x11.canvas.Canvas`*) and implement the `on_draw` method.

Start by reading through the documentation of the *`blighty.x11`* module and then make your way to the examples folder from the GitHub repository.

CHAPTER 3

A clock widget

Here is the example of a simple clock widget:

```
from blighty import CanvasGravity, brush
from blighty.x11 import Canvas, start_event_loop

import datetime

from math import pi as PI


class Clock(Canvas):
    def on_button_pressed(self, button, state, x, y):
        self.dispose()

    @brush
    def hand(ctx, angle, length, thickness):
        ctx.save()
        ctx.set_source_rgba(1, 1, 1, 1)
        ctx.set_line_width(thickness)
        ctx.rotate(angle)
        ctx.move_to(0, length * .2)
        ctx.line_to(0, -length)
        ctx.stroke()
        ctx.restore()

    def on_draw(self, ctx):
        now = datetime.datetime.now()

        ctx.translate(self.width >> 1, self.height >> 1)

        ctx.hand(
            angle = now.second / 30 * PI,
            length = (self.height >> 1) * .9,
            thickness = 1
```

(continues on next page)

(continued from previous page)

```
        )

        mins = now.minute + now.second / 60
        ctx.hand(
            angle = mins / 30 * PI,
            length = (self.height >> 1) * .8,
            thickness = 3
        )

        hours = (now.hour % 12) + mins / 60
        ctx.hand(
            angle = hours / 6 * PI,
            length = (self.height >> 1) * .5,
            thickness = 6
        )


if __name__ == "__main__":
    clock = Clock(0, 0, 400, 400, gravity = CanvasGravity.CENTER)
    clock.show()
    start_event_loop()
```

CHAPTER 4

Issues

If you find any issues with `blighty`, or for a list of all the currently known and open issues, please visit https://github.com/P403n1x87/blighty/issues.

CHAPTER 5

The brag corner

The `blighty` project was founded by Gabriele Tornetta in 2018.

5.1 blighty package

5.1.1 Module content

This module contains the common objects and types for the different kind of canvases provided by `blighty`.

class `blighty.`**CanvasGravity**

Window gravity control type.

The positioning of a canvas on the screen is controlled by its gravity. By default, a window is positioned in a coordinate system where the origin is located in the top-left corner of the screen, with the *x* axis running horizontally from left to right, and the *y* from top to bottom. To change the location of the origin, use one of the following values.

CENTER = 5

EAST = 6

NORTH = 2

NORTH_EAST = 3

NORTH_WEST = 1

SOUTH = 8

SOUTH_EAST = 9

SOUTH_WEST = 7

STATIC = 10

WEST = 4

class blighty.**CanvasType**

The Canvas type.

The canvas types enumerated in this Python type reflect the same window types that one can request to the window manager via the Extended Window Manager Hints.

- NORMAL is a normal top-level window.
- DESKTOP is a window drawn directly on the desktop.
- DOCK indicates a dock or panel window that will usually stay on top of other windows.
- UNDECORATED is a type of window that behaves as a toolbar. As such, it is undecorated.

DESKTOP = 1

DOCK = 2

NORMAL = 0

UNDECORATED = 3

5.1.2 Subpackages

blighty.gtk package

Module contents

Submodules

blighty.gtk.canvas module

blighty.x11 package

Module contents

This module provides support for creating X11 canvases. If you are trying to replicate conky's behaviour, the API offered by this module is the closest to it.

Submodules

blighty.x11.canvas module

Description

This module provides the *Canvas* class for the creation of X11 canvases.

The *Canvas* class is, in Java terminoly, *abstract* and should not be instantiated directly. Instead, applications should define their own subclasses of the *Canvas* and implement the on_draw() method, which gets called periodically to perform the required draw operations using pycairo.

Once created, an instance of a subclass of *Canvas* can be shown on screen by calling the show() method. This starts drawing the canvas on screen by calling the *on_draw* callback at regular intervals in time. Events can be handled by starting the event loop with blighty.x11.start_event_loop(), as described in more details in the *Event handling* section.

Creating a canvas

Canvases are created by simply subclassing the *Canvas* class and implementing the `on_draw()` callback.

The *Canvas* constructor (i.e. the `__new__()` magic method) takes the following arguments:

Argument	Description
x *y* *width* *height*	These arguments describe the basic geometry of the canvas. The *x* and *y* coordinates are relative to the `gravity` argument (see below). The *width* and *height* arguments give the canvas size in pixels.
interval	The time interval between calls to the `on_draw()` callback, in *milliseconds*. **Default value**: 1000 (i.e. 1 second)
window_type	The type of window to create. The possible choices are enumerated in the `blighty.CanvasType` type and are named after the equivalent _NET_WM_WINDOW_TYPE hints for the window manager. This is analogous to conky's `own_window_type` configuration setting. **Default value**: `CanvasType.DESKTOP`
gravity	Defines the coordinate system for the canvas relative to the screen. The allowed values are enumerated in the `blighty.CanvasGravity` type. This is the equivalent of the conky `alignment` configuration setting. For example, the value `CanvasGravity.SOUTH_EAST` indicates that the canvas should be positioned relative to the bottom-right corner of the screen. **Default value**: `CanvasGravity.NORTH_WEST`
sticky	Whether the window should *stick* to the desktop and hence be visible in all workspaces. **Default value**: `True`
keep_below	Whether the window should stay below any other window on the screen. **Default value**: `True`
skip_taskbar	Whether the window should not have an entry in the taskbar. **Default value**: `True`
skip_pager	Whether the window should not appear in the pager. **Default value**: `True`

Note that the interval can be changed dynamically by setting the `interval` attribute on the canvas object directly after it has been created.

If you want to distribute your subclasses of *Canvas*, we recommend that you create a static method `build` that returns an instance of the subclass, with some of the argumets set to a predefined values. This is useful if you want to distribute widgets with, e.g., a predefined size, as a Python module.

Showing the canvas

When a canvas is created, it is not immediately shown to screen. To map it to screen and start the draw cycle one has to call the `show()` method explicitly.

If you need to pass data to the canvas, you might want to do that before calling this method, since presumably the `on_draw()` callback, which will start to be called, makes use of it.

Finally, you must start the main event loop with `blighty.x11.start_event_loop()` to start drawing on the canvases, and in case that they should handle input events, like mouse button clicks or key presses. Note however that execution in the current thread will halt at this call, until it returns after a call to `blighty.x11.stop_event_loop()`.

For more details on how to handle events with your X11 canvases, see the section *Event handling* below.

Disposing of a canvas

If you want to programmatically dispose of a canvas, you can call the `dispose()` method. This doesn't destroy the canvas immediately, but sends a delete request to the main event loop instead. This is the preffered way of getting rid of a canvas when you are running the event loop. You can also use the `destroy()` method directly, which destroys the canvas immediately. However this is not thread safe and should not be called in the `on_draw()` callback when running the event loop.

Event handling

A feature that distinguishes blighty from conky is that it allows you to handle simple user input on the canvases. Currently, X11 canvases support two events: mouse button and key press events.

Mouse button events can be handled by implementing the `on_button_pressed()` callback in the subclass of *Canvas*. The signature is the following:

```
def on_button_pressed(self, button, state, x, y):
```

and the semantics of the arguments is the same as the `XButtonEvent`[1].

To handle key presses, implement the `on_key_pressed` callback with the following signature:

```
def on_key_pressed(self, keysym, state):
```

The `state` argument has the same semantics as in the `on_button_pressed()` case, while the `keysym` is described, e,g, in the Keyboard Econding section of the Xlib guide.

A simple example

Here is a simple example that shows all the above concepts in action:

```
from blighty import CanvasGravity
from blighty.x11 import Canvas, start_event_loop

class MyCanvas(Canvas):
    @staticmethod
    def build(x, y):
        return MyCanvas(x, y, 200, 200, gravity = CanvasGravity.NORTH)

    def on_button_pressed(self, button, state, x, y):
        if button == 1:  # Left mouse button pressed
            self.dispose()

    def on_draw(self, ctx):
        ctx.set_source_rgb(1, 0, 0)
        ctx.rectangle(0, 0, ctx.canvas.width >> 1, ctx.canvas.height >> 1)
        ctx.fill()

if __name__ == "__main__":
    # Instantiate the canvas
```

(continues on next page)

[1] https://tronche.com/gui/x/xlib/events/keyboard-pointer/keyboard-pointer.html

(continued from previous page)

```
    canvas = MyCanvas.build()

    # Map it on screen
    canvas.show()

    # Start the event loop
    start_event_loop()
```

Extra features

The *Canvas* class comes with some handy extra features that can help with common patterns, thus sparing you to have to type boilerplate code.

Brushes

Brushes are a way to rebind methods from your subclass of *Canvas* to the Cairo context. Consider the following example:

```
from random import random as r

class RectCanvas(blighty.x11.Canvas):
    def rect(self, ctx, width, height):
        ctx.set_source_rgb(*[r() for _ in range(3)])
        ctx.rectangle(0, 0, width, height)
        ctx.fill()

    def on_draw(self, ctx):
        for i in range(4):
            self.rect(ctx, self.width >> i, self.height >> i)
```

The method `rect` is defined under the class `RectCanvas` for convenience. However, from a logical point of view, it would make more sense for this method to belong to `ctx`, since the general pattern of these helper methods requires that we pass `ctx` as one of the arguments.

If one prefixes the `rect` method with `draw_` then it turns into an *implicit brush*. The `on_draw()` callback is called with the `ctx` argument being an instance of `ExtendedContext`. The `draw_rect` brush is then available from `ctx` as a bound method. The sample code above can then be refactored as:

```
from random import random as r

class RectCanvas(blighty.x11.Canvas):
    def draw_rect(ctx, width, height):
        ctx.set_source_rgb(*[r() for _ in range(3)])
        ctx.rectangle(0, 0, width, height)
        ctx.fill()

    def on_draw(self, ctx):
        for i in range(4):
            ctx.rect(self.width >> i, self.height >> i)
```

Notice how `draw_rect` now takes less arguments, and how the first one is `ctx`, the (extended) Cairo context.

If you do not wish to prefix your methods with `draw_`, you can use the `blighty.brush()` decorator instead to create an *explicit brush*. The code would then look like this:

```
from blighty import brush
from random import random as r

class RectCanvas(blighty.x11.Canvas):
    @brush
    def rect(ctx, width, height):
        ctx.set_source_rgb(*[r() for _ in range(3)])
        ctx.rectangle(0, 0, width, height)
        ctx.fill()

    def on_draw(self, ctx):
        for i in range(4):
            ctx.rect(self.width >> i, self.height >> i)
```

Text alignment

A common task is writing text on a canvas. With Cairo, text alignment usually requires the same pattern: get the text extents and compute the new position. To help with that, *Canvas* objects come with a pre-defined `write_text()` brush. Please refer to the API documentation below for usage details.

Grid

When designing a canvas from scrach, it is hard to guess at positions without any guiding lines. To help with precise placement, every *Canvas* object comes with a `draw_grid` brush that creates a rectangular grid on the canvas. The spacing between the lines is set to 50 pixels by default (assuming that the scale hasn't been changed before). This can be adjusted by passing the new spacing along the two directions as arguments. Please refer to the API documentation below for more details.

References

Module API

class blighty.x11.canvas.**Canvas**(**args*, ***kwargs*)

Bases: x11.BaseCanvas

X11 Canvas object.

This class is meant to be used as a superclass and should not be instantiated directly. Subclasses should implement the *on_draw()* callback, which is invoked every time the canvas needs to be redrawn. Redraws happen at regular intervals in time, as specified by the `interval` attribute (also passed as an argument via the constructor).

destroy()

Destroy the canvas.

This method is not thread-safe. Use the *dispose()* method instead.

dispose()

Mark the canvas as ready to be destroyed to free up resources.

draw_grid(*x=50*, *y=50*)

Draw a grid on the canvas [**implicit brush**].

This implicit brush method is intended to help with determining the location of points on the canvas during development.

Parameters

- **x** (*int*) – The horizontal spacing between lines.
- **y** (*int*) – The vertical spacing between lines.

get_size()
Get the canvas size.

Returns the 2-tuple of width and height in pixels.

Return type tuple

height
The canvas height. *Read-only*.

interval
The refresh interval, in milliseconds.

move()
Move the canvas to new coordinates.

The *x* and *y* coordinates are relative to the canvas gravity.

on_draw(*ctx*)
Draw callback.

Once the *show()* method is called on a *Canvas* object, this method gets called at regular intervals of time to perform the draw operation. Every subclass of *Canvas* must implement this method.

show()
Map the canvas to screen and set it ready for drawing.

width
The canvas width. *Read-only*.

write_text(*x*, *y*, *text*, *align=3*)
Write aligned text [**explicit brush**].

This explicit brush method helps write aligned text on the canvas. The *x* and *y* coordinates are relative to the specified *alignment*. By default, this is `blighty.TextAlign.TOP_LEFT`, meaning that the text will be left-aligned and on top of the horizontal line that passes through *y* on the vertical axis. In terms of the point *(x,y)* on the Canvas, the text will develop in the NE direction.

The return value is the text extents, in case that some further draw operations depend on the space required by the text to be drawn on the canvas.

Note that font face and size need to be set on the Cairo context prior to a call to this method.

Parameters

- **x** (*int*) – The horizontal coordinate.
- **y** (*int*) – The vertical coordinate.
- **text** (*str*) – The text to write.
- **align** (*int*) – The text alignment. Detaulf is `TextAlign.TOP_LEFT`.

Returns The same return value as `cairo.text_extents`.

Return type tuple

x
The canvas *x* coordinate. *Read-only*.

y
The canvas *y* coordinate. *Read-only*.

Python Module Index

b

blighty, 11
blighty.x11, 12
blighty.x11.canvas, 12

Index

B

blighty (module), 11
blighty.x11 (module), 12
blighty.x11.canvas (module), 12

C

Canvas (class in blighty.x11.canvas), 16
CanvasGravity (class in blighty), 11
CanvasType (class in blighty), 11
CENTER (blighty.CanvasGravity attribute), 11

D

DESKTOP (blighty.CanvasType attribute), 12
destroy() (blighty.x11.canvas.Canvas method), 16
dispose() (blighty.x11.canvas.Canvas method), 16
DOCK (blighty.CanvasType attribute), 12
draw_grid() (blighty.x11.canvas.Canvas method), 16

E

EAST (blighty.CanvasGravity attribute), 11

G

get_size() (blighty.x11.canvas.Canvas method), 17

H

height (blighty.x11.canvas.Canvas attribute), 17

I

interval (blighty.x11.canvas.Canvas attribute), 17

M

move() (blighty.x11.canvas.Canvas method), 17

N

NORMAL (blighty.CanvasType attribute), 12
NORTH (blighty.CanvasGravity attribute), 11
NORTH_EAST (blighty.CanvasGravity attribute), 11
NORTH_WEST (blighty.CanvasGravity attribute), 11

O

on_draw() (blighty.x11.canvas.Canvas method), 17

S

show() (blighty.x11.canvas.Canvas method), 17
SOUTH (blighty.CanvasGravity attribute), 11
SOUTH_EAST (blighty.CanvasGravity attribute), 11
SOUTH_WEST (blighty.CanvasGravity attribute), 11
STATIC (blighty.CanvasGravity attribute), 11

U

UNDECORATED (blighty.CanvasType attribute), 12

W

WEST (blighty.CanvasGravity attribute), 11
width (blighty.x11.canvas.Canvas attribute), 17
write_text() (blighty.x11.canvas.Canvas method), 17

X

x (blighty.x11.canvas.Canvas attribute), 17

Y

y (blighty.x11.canvas.Canvas attribute), 18