OmniCanvas Documentation

Release 0.2.2

Sam Ireland

December 15, 2016

Contents

OmniCanvas is a Python canvas which supports basic two-dimensional graphics, and outputs to various graphics formats - currently just SVG.

CHAPTER 1

Example

```
>>> import omnicanvas
>>> canvas = omnicanvas.Canvas(700, 400)
>>> canvas.add_rectangle(10, 20, 300, 200, fill_color="#CC0000")
>>> canvas.graphics()[0]
<Rectangle 300×200 at (10,20)>
>>> canvas.save("example.svg")
```

See the the examples page for more examples, or the full API for a full listing of features.

CHAPTER 2

Table of Contents

2.1 Installing

2.1.1 pip

OmniCanvas can be installed using pip:

```
$ pip3 install omnicanvas
```

OmniCanvas is written for Python 3, and does not support Python 2.

If you get permission errors, try using `sudo`:

```
$ sudo pip3 install omnicanvas
```

2.1.2 Development

The repository for OmniCanvas, containing the most recent iteration, can be found here. To clone the OmniCanvas repository directly from there, use:

```
git clone git://github.com/samirelanduk/omnicanvas.git
```

2.1.3 Requirements

OmniCanvas currently has no external dependencies, and is pure Python.

2.2 Overview

2.2.1 The Canvas

The *Canvas* class is the first thing you will need to create - it is the base upon which everything else is created. If you've ever used HTML or GUI canvases, they work in much the same way.

```
>>> import omnicanvas
>>> canvas = omnicanvas.Canvas(600, 400)
>>> canvas.width()
600
>>> canvas.height()
400
```

Canvases need width and height information, in pixels. You can also specify the background color:

```
>>> canvas = omnicanvas.Canvas(600, 400, background_color="#0000DD")
>>> canvas.background_color()
'#0000DD'
```

2.2.2 Graphics

Canvases contain a list of graphics (which all inherit from class *Graphic*), and it is these objects which make up the image.

Currently, there are the following types.

Rectangles

Rectangle objects are... rectangles. Boxes, defined with the coordinates of their top-left corner, and width and height:

```
>>> canvas.add_rectangle(10, 10, 300, 200)
>>> canvas.graphics()[0]
<Rectangle 300×200 at (10,10)>
>>> canvas.graphics()[0].width()
300
```

Rectangles inherit from *ShapeGraphic* which means they have a fill color and opacity:

```
>>> canvas.add_rectangle(10, 10, 300, 200, fill_color="#3344EE", opacity=0.5)
>>> canvas.graphics()[-1].fill_color()
'#3344EE'
>>> canvas.graphics()[-1].opacity()
0.5
```

They *also*, like all classes which inherit from class *Graphic*, have properties relating to their lines:

```
>>> canvas.add_rectangle(10, 10, 300, 200, line_width=2, line_style="--")
>>> canvas.graphics()[-1].line_width()
2
>>> canvas.graphics()[-1].line_style()
'--'
>>> canvas.graphics()[-1].line_color("#FFFF00")
>>> canvas.graphics()[-1].line_color()
'#FFFF00'
```

Lines

Line objects are even more straightforward. They are lines defined by a start coordinate and an end coordinate:

```
>>> canvas.add_line(10, 10, 90, 90)
>>> canvas.x1()
10
>>> canvas.y1()
10
>>> canvas.x2()
90
>>> canvas.y2()
90
```

Lines inherit directly from *Graphic* and have the same properties relating to line width etc. as above.

Polygons

Polygon objects are two-dimensional shapes with an arbitrary number of points. These are given as a sequence of coordinates:

```
>>> canvas.add_polygon(60, 60, 90, 120, 30, 120) # Creates a triangle
>>> canvas.graphics()[-1].coordinates()
(60, 60, 90, 120, 30, 120)
>>> canvas.graphics()[-1].coordinates(xy_pairs=True)
((60, 60), (90, 120), (30, 120))
```

You must supply an even number of points, and there must be at least three vertices.

Otherwise they behave much like Rectangles - they inherit from *ShapeGraphic* and *Graphic* and so have the above properties relating to fill and border.

Text

Text objects are used to hold text. Unlike other Graphics, their default `fill_color` is black, not white, and their default `line_width` is 0, not 1.

```
>>> canvas.add_text(50, 50, "OmniCanvas is sexy", font_size=32)
>>> canvas.graphics()[-1].text()
'OmniCanvas is sexy'
>>> canvas.graphics()[-1].font_size()
32
>>> canvas.graphics()[-1].fill_color()
'#000000'
>>> canvas.graphics()[-1].line_width()
0
```

The coordinate given by default will be the centre of the text. This can be changed by specifying the desired horizontal and vertical alignment:

```
>>> canvas.add_text(50, 50, "X", vertical_align="top", horizontal_align="left")
```

Polylines

These are very similar to *Polygon*, except the last vertex is not joined to the first one, and so they have no interior space. They are just lines with an arbitrary number of vertices.

They behave very similarly to Polygons:

```
>>> canvas.add_polyline(60, 60, 90, 120, 30, 120)
>>> canvas.graphics()[-1].coordinates()
(60, 60, 90, 120, 30, 120)
>>> canvas.graphics()[-1].coordinates(xy_pairs=True)
((60, 60), (90, 120), (30, 120))
```

2.2.3 Graphic Retrieval

All of the above graphic adding methods will return the graphic they have just added, if you need a reference to it later.

```
>>> rectangle = canvas.add_rectangle(10, 10, 300, 200)
>>> rectangle
<Rectangle 300×200 at (10,10)>
```

Additionally, all graphics can be given names, which can then be used to retrieve them from the Canvas using two methods - *get_graphic_by_name()* and *get_graphics_by_name()*:

```
>>> canvas.add_line(10, 10, 90, 90, name="Line 1")
>>> canvas.add_line(20, 10, 90, 90, name="Line 2")
>>> canvas.add_line(10, 20, 90, 90, name="A Line")
>>> canvas.add_line(20, 20, 90, 90, name="A Line")
>>> canvas.get_graphic_by_name("Line 1")
<Line (10,10) to (90,70)>
>>> canvas.get_graphic_by_name("Line 2")
<Line (20,10) to (90,70)>
>>> canvas.get_graphic_by_name("A Line")
<Line (10,20) to (90,70)>
>>> canvas.get_graphics_by_name("A Line")
[<Line (10,20) to (90,70)>, <Line (20,20) to (90,70)>]
```

2.2.4 Outputs

Once the canvas has been decorated with whatever Graphics you see fit, it can be saved to file:

```
>>> canvas.save("example.svg")
```

Most browsers will have no trouble displaying SVG files once created.

If you want to get the text of the SVG directly, you can use the `to_svg()` method of canvases, which will return this raw text.

2.3 Examples

2.3.1 The UK Union Flag

The following code will produce an SVG file of the British flag:

```
import math
import omnicanvas

def create_union_flag(height):
    # The union flag is twice as wide as it is high
    canvas = omnicanvas.Canvas(height * 2, height, background_color="#000066")

    #This is the length of the diagonal of the flag, with Pythagoras
    diagonal_length = math.sqrt((height ** 2) + ((height * 2) ** 2))

    # This is the angle of the diagonal strips from the horizontal
    # tan(θ) = opposite / adjacent, so θ = atan(opposite / adjacent)
    diagonal_angle = math.degrees(math.atan((height / 2) / height))

    # Add The diagonal white strips
    canvas.add_rectangle(
     height - (height * 0.1),
     (height / 2) - (diagonal_length / 2),
```

```
    height * 0.2,
    diagonal_length,
    line_width=0,
    rotation=(
     height, height / 2, 270 + diagonal_angle
    )
   )
   canvas.add_rectangle(
    height - (height * 0.1),
    (height / 2) - (diagonal_length / 2),
    height * 0.2,
    diagonal_length,
    line_width=0,
    rotation=(
     height, height / 2, 90 - diagonal_angle
    )
   )

   # Add diagonal red strips - these'll be partly covered by the white cross
   canvas.add_rectangle(
    height - (height / 15),
    (height / 2) - (diagonal_length / 2),
    height / 15,
    diagonal_length / 2,
    line_width=0,
    fill_color="#CC0000",
    rotation=(
     height, height / 2, 90 - diagonal_angle
    )
   )
   canvas.add_rectangle(
    height - (height / 15),
    (height / 2) - (diagonal_length / 2),
    height / 15,
    diagonal_length / 2,
    line_width=0,
    fill_color="#CC0000",
    rotation=(
     height, height / 2, 270 - diagonal_angle
    )
   )
   canvas.add_rectangle(
    height - (height / 15),
    (height / 2) - (diagonal_length / 2),
    height / 15,
    diagonal_length / 2,
    line_width=0,
    fill_color="#CC0000",
    rotation=(
     height, height / 2, 270 + diagonal_angle
    )
   )
   canvas.add_rectangle(
    height - (height / 15),
    (height / 2) - (diagonal_length / 2),
    height / 15,
    diagonal_length / 2,
    line_width=0,
```

```
     fill_color="#CC0000",
     rotation=(
      height, height / 2, 90 + diagonal_angle
     )
    )

    # Add the white cross
    canvas.add_rectangle(
     height - (height / 6),
     0,
     height / 3,
     height,
     line_width=0
    )
    canvas.add_rectangle(
     0,
     (height / 2) - (height / 6),
     height * 2,
     height / 3,
     line_width=0
    )

    # Add the red cross
    canvas.add_rectangle(
     height - (height / 10),
     0,
     height / 5,
     height,
     line_width=0,
     fill_color="#CC0000",
    )
    canvas.add_rectangle(
     0,
     (height / 2) - (height / 10),
     height * 2,
     height / 5,
     line_width=0,
     fill_color="#CC0000",
    )

    return canvas

# Create a flag of height 360px (and so width 720px)
create_union_flag(360).save("ukflag.svg")
```

2.4 Full API

2.4.1 `omnicanvas.canvas` (The canvas)

This module contains the main Canvas class.

class omnicanvas.canvas.**Canvas**(*width*, *height*, *background_color=None*)

A backdrop on which other *Graphic* objects are painted.

Parameters

- **width** – The canvas's width in pixels.
- **height** – The canvas's height in pixels.
- **background_color** – The canvas's background colour - the default is white

width(*width=None*)
The canvas's width in pixels. Passing a value will update the width property.

Parameters **width** – If given, the canvas's width will be set to this.

Return type int

height(*height=None*)
The canvas's height in pixels. Passing a value will update the height property.

Parameters **height** – If given, the canvas's height will be set to this.

Return type int

background_color(*background_color=None*)
The canvas's background colour, as a hex string. Passing a value will update the background_color property (as a hex string).

Parameters **background_color** (*str*) – If given, the canvas's background_color will be set to this.

Return type str

graphics()
A list of all the *Graphic* objects on this canvas.

Return type list

get_graphic_by_name(*name*)
Searches the canvas's *Graphic* objects and returns the first one with a matching name. Returns None if there are no matches.

Parameters **name** (*str*) – The name to search by.

Return type str

get_graphics_by_name(*name*)
Searches the canvas's *Graphic* objects and returns all the ones with a matching name. Returns an empty list if there are no matches.

Parameters **name** (*str*) – The name to search by.

Returns list of *Graphic*

add_rectangle(**args*, ***kwargs*)
Adds a *Rectangle* to the canvas.

Parameters

- **x** – The x-coordinate of the Rectangle's upper left corner.
- **y** – The y-coordinate of the Rectangle's upper left corner.
- **width** – The Rectangle's width.
- **height** – The Rectangle's height.
- **fill_color** (*str*) – The Rectangle's interior colour.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **line_width** – The width of the edge of the Rectangle in pixels.

- **line_style** (*str*) – The pattern of the edges. Acceptable values are - (default), .. (dotted) or -- (dashed).
- **line_color** (*str*) – The colour of the edge.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle).
- **data** (*dict*) – Any data to be associated with the Rectangle.

Return type *Rectangle*

add_line(**args*, ***kwargs*)
Adds a *Line* to the canvas.

Parameters

- **x1** – The x-coordinate of the Line's start point.
- **y1** – The y-coordinate of the Line's start point.
- **x2** – The x-coordinate of the Line's end point.
- **y2** – The y-coordinate of the Line's end point.
- **line_width** – The width of the Line in pixels.
- **line_style** (*str*) – The pattern of the Line. Acceptable values are - (default), .. (dotted) or -- (dashed).
- **line_color** (*str*) – The colour of the Line.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle).
- **data** (*dict*) – Any data to be associated with the Line.

Return type *Line*

add_polygon(**args*, ***kwargs*)
Adds a *Polygon* to the canvas.

Parameters

- ***points** – The alternating x and y values of the Polygon's corners.
- **fill_color** (*str*) – The Polygon's interior colour.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **line_width** – The width of the edge of the Polygon in pixels.
- **line_style** (*str*) – The pattern of the edges. Acceptable values are - (default), .. (dotted) or -- (dashed).
- **line_color** (*str*) – The colour of the edge.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle).
- **data** (*dict*) – Any data to be associated with the Polygon.

Return type *Polygon*

add_text(**args*, ***kwargs*)
Adds a *Text* to the canvas.

Parameters

- **x** – The Text's x location.
- **y** – The Text's y location.
- **text** (*str*) – The text to display.
- **font_size** – The font size of the Text when displayed.
- **horizontal_align** – The horizontal alignment of the Text. Acceptable values are `left`, `center` (default) and `right`.
- **vertical_align** – The vertical alignment of the Text. Acceptable values are `top`, `middle` (default) and `bottom`.
- **fill_color** (*str*) – Defaults to '#FFFFFF'.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **line_width** – Defaults to 0.
- **line_style** (*str*) – The line pattern. Acceptable values are `-` (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Text.

Return type *Text*

add_polyline(**args*, ***kwargs*)
Adds a *Polyline* to the canvas.

Parameters

- ***points** – The alternating x and y values of the Polyline's corners.
- **line_width** – The width of the edge of the Polyline in pixels.
- **line_style** (*str*) – The pattern of the edges. Acceptable values are `-` (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – The colour of the edge.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle).
- **data** (*dict*) – Any data to be associated with the Polyline.

Return type *Polyline*

save(*path*)
Saves the canvas to file as an SVG file.

Parameters **path** (*str*) – The location and filename to save to.

to_svg(*canvas*)
Returns the SVG text of the canvas.

Any `data` attributes of the Graphics contained will be rendered as SVG attributes.

Return type `str`

2.4.2 omnicanvas.graphics (The various Graphics)

This module contains the various Graphics objects which can be painted to the canvas.

class omnicanvas.graphics.**Graphic**(*name=None*, *line_width=1*, *line_style='-'*, *line_color='#000000'*, *rotation=(0, 0, 0)*, *data=None*)

The base class of all Graphics - it would not orindarily need to be instantiated at this abstract level.

Its properties generally relate to lines, as all Graphics have an edge of some kind.

Parameters

- **name** (*str*) – An identifable name for the Graphic.
- **line_width** – Defaults to 1.
- **line_style** (*str*) – The line pattern. Acceptable values are - (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Graphic.

name(*name=None*)

An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type `str`

line_width(*line_width=None*)

The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type `int`

line_style(*line_style=None*)

The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are - (default), `..` (dotted) or `--` (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type `str`

line_color(*line_color=None*)

The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type `str`

rotation()

The rotation applied to the Graphic. It takes the form of a three number tuple - `(pivot_x_coordinate, pivot_y_coordinate, angle)`. The angle is in degrees, and works clockwise.

Return type `tuple`

rotate(*x_pivot*, *y_pivot*, *angle*)

Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's `rotation` property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.
- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

data()

Returns any data associated with the Graphic as a `dict`. This `dict` is modifiable.

When rendered as SVG, the data will be turned into `attribute="value"` pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type `dict`

class omnicanvas.graphics.**ShapeGraphic**(**args*, *fill_color='#FFFFFF'*, *opacity=1*, ***kwargs*)

Base class: *Graphic*

Shapes are Graphics which have an interior space of some kind, which in practice is most Graphics which aren't just lines.

Its properties relate to the appearence of this interior space. It would not generally be instantiated directly.

Parameters

- **fill_color** (*str*) – Defaults to '#FFFFFF'.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **name** (*str*) – An identifable name for the Graphic.
- **line_width** – Defaults to 1.
- **line_style** (*str*) – The line pattern. Acceptable values are – (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Graphic.

fill_color(*fill_color=None*)

The colour of the shape's interior space. Passing a value will update the fill_color property. All colours must be in the form #RRGGBB.

Parameters **fill_color** (*str*) – If given, the Shape's fill_color will be set to this.

Return type `str`

opacity(*opacity=None*)

The degree of transparency of the interior space, from 0 to 1 (0 being invisible). Passing a value will update the opacity property.

Parameters **opacity** (*floar*) – If given, the Shape's opacity will be set to this.

Return type `float`

data()
Returns any data associated with the Graphic as a `dict`. This `dict` is modifiable.

When rendered as SVG, the data will be turned into `attribute="value"` pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type `dict`

line_color(*line_color=None*)
The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type `str`

line_style(*line_style=None*)
The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are – (default), `..` (dotted) or `--` (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type `str`

line_width(*line_width=None*)
The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type `int`

name(*name=None*)
An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type `str`

rotate(*x_pivot*, *y_pivot*, *angle*)
Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's `rotation` property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.
- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

rotation()
The rotation applied to the Graphic. It takes the form of a three number tuple - `(pivot_x_coordinate, pivot_y_coordinate, angle)`. The angle is in degrees, and works clockwise.

Return type `tuple`

class `omnicanvas.graphics.`**BoxGraphic**(*x*, *y*, *width*, *height*, **args*, ***kwargs*)
Base class: *ShapeGraphic*

BoxGraphics are shapes which are either a box themselves (such as *Rectangle*) or are defined by some kind of bounding box. It would not generally be instantiated directly.

Parameters

- **x** – The x-value of the top-left corner.
- **y** – The y-value of the top-left corner.
- **width** – The Box's width in pixels.
- **height** – The Box's height in pixels.
- **fill_color** (*str*) – Defaults to '#FFFFFF'.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **name** (*str*) – An identifable name for the Graphic.
- **line_width** – Defaults to 1.
- **line_style** (*str*) – The line pattern. Acceptable values are - (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Box.

x(*x=None*)

The x value of the Box's upper-left corner coordinate. Passing a value will update the x property.

Parameters **x** – If given, the Box's x value will be set to this.

Return type `float` or `int`

y(*y=None*)

The y value of the Box's upper-left corner coordinate. Passing a value will update the y property.

Parameters **y** – If given, the Box's y value will be set to this.

Return type `float` or `int`

width(*width=None*)

The width of the Box. Passing a value will update the width.

Parameters **width** – If given, the Box's width will be set to this.

Return type `float` or `int`

height(*height=None*)

The height of the Box. Passing a value will update the height.

Parameters **height** – If given, the Box's height will be set to this.

Return type `float` or `int`

data()

Returns any data associated with the Graphic as a `dict`. This `dict` is modifiable.

When rendered as SVG, the data will be turned into `attribute="value"` pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type `dict`

fill_color(*fill_color=None*)
The colour of the shape's interior space. Passing a value will update the fill_color property. All colours must be in the form #RRGGBB.

Parameters **fill_color** (*str*) – If given, the Shape's fill_color will be set to this.

Return type str

line_color(*line_color=None*)
The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type str

line_style(*line_style=None*)
The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are – (default), `..` (dotted) or `--` (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type str

line_width(*line_width=None*)
The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type int

name(*name=None*)
An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type str

opacity(*opacity=None*)
The degree of transparency of the interior space, from 0 to 1 (0 being invisible). Passing a value will update the opacity property.

Parameters **opacity** (*floar*) – If given, the Shape's opacity will be set to this.

Return type float

rotate(*x_pivot*, *y_pivot*, *angle*)
Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's `rotation` property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.
- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

rotation()
The rotation applied to the Graphic. It takes the form of a three number tuple - `(pivot_x_coordinate, pivot_y_coordinate, angle)`. The angle is in degrees, and works clockwise.

Return type `tuple`

class omnicanvas.graphics.**Rectangle**(**args*, ***kwargs*)
Base class: *BoxGraphic*

A rectangular graphic, represented as a Rectangle on screen.

Parameters

- **x** – The x-value of the top-left corner.
- **y** – The y-value of the top-left corner.
- **width** – The Rectangle's width in pixels.
- **height** – The Rectangle's height in pixels.
- **fill_color** (*str*) – Defaults to '#FFFFFF'.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **name** (*str*) – An identifable name for the Graphic.
- **line_width** – Defaults to 1.
- **line_style** (*str*) – The line pattern. Acceptable values are - (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle).
- **data** (*dict*) – Any data to be associated with the Shape.

data()
Returns any data associated with the Graphic as a `dict`. This `dict` is modifiable.

When rendered as SVG, the data will be turned into `attribute="value"` pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type `dict`

fill_color(*fill_color=None*)
The colour of the shape's interior space. Passing a value will update the fill_color property. All colours must be in the form #RRGGBB.

Parameters **fill_color** (*str*) – If given, the Shape's fill_color will be set to this.

Return type `str`

height(*height=None*)
The height of the Box. Passing a value will update the height.

Parameters **height** – If given, the Box's height will be set to this.

Return type `float` or `int`

line_color(*line_color=None*)
The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type str

line_style(*line_style=None*)
The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are – (default), `..` (dotted) or `--` (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type str

line_width(*line_width=None*)
The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type int

name(*name=None*)
An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type str

opacity(*opacity=None*)
The degree of transparency of the interior space, from 0 to 1 (0 being invisible). Passing a value will update the opacity property.

Parameters **opacity** (*floar*) – If given, the Shape's opacity will be set to this.

Return type float

rotate(*x_pivot*, *y_pivot*, *angle*)
Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's `rotation` property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.
- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

rotation()
The rotation applied to the Graphic. It takes the form of a three number tuple - `(pivot_x_coordinate, pivot_y_coordinate, angle)`. The angle is in degrees, and works clockwise.

Return type tuple

width(*width=None*)
The width of the Box. Passing a value will update the width.

Parameters **width** – If given, the Box's width will be set to this.

Return type float or int

x (*x=None*)
The x value of the Box's upper-left corner coordinate. Passing a value will update the x property.

Parameters **x** – If given, the Box's x value will be set to this.

Return type `float` or `int`

y (*y=None*)
The y value of the Box's upper-left corner coordinate. Passing a value will update the y property.

Parameters **y** – If given, the Box's y value will be set to this.

Return type `float` or `int`

class `omnicanvas.graphics.`**Line**(*x1*, *y1*, *x2*, *y2*, **args*, ***kwargs*)
Base class: *Graphic*

A straight line between two points.

Parameters

- **x1** – The x-value of the start point.
- **y1** – The y-value of the start point.
- **x2** – The x-value of the end point.
- **y2** – The y-value of the end point.
- **name** (*str*) – An identifable name for the Graphic.
- **line_width** – Defaults to 1.
- **line_style** (*str*) – The line pattern. Acceptable values are - (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Graphic.

x1 (*x1=None*)
The x value of the Line's start point coordinate. Passing a value will update the x1 property.

Parameters **x1** – If given, the Line's x1 value will be set to this.

Return type `float` or `int`

y1 (*y1=None*)
The y value of the Line's start point coordinate. Passing a value will update the y1 property.

Parameters **y1** – If given, the Line's y1 value will be set to this.

Return type `float` or `int`

x2 (*x2=None*)
The x value of the Line's end point coordinate. Passing a value will update the x2 property.

Parameters **x2** – If given, the Line's x2 value will be set to this.

Return type `float` or `int`

y2 (*y2=None*)
The y value of the Line's end point coordinate. Passing a value will update the y2 property.

Parameters **y2** – If given, the Line's y2 value will be set to this.

Return type float or int

data()
Returns any data associated with the Graphic as a dict. This dict is modifiable.

When rendered as SVG, the data will be turned into attribute="value" pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type dict

line_color(*line_color=None*)
The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type str

line_style(*line_style=None*)
The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are – (default), .. (dotted) or –– (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type str

line_width(*line_width=None*)
The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type int

name(*name=None*)
An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type str

rotate(*x_pivot*, *y_pivot*, *angle*)
Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's rotation property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.
- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

rotation()
The rotation applied to the Graphic. It takes the form of a three number tuple - (pivot_x_coordinate, pivot_y_coordinate, angle). The angle is in degrees, and works clockwise.

Return type tuple

class omnicanvas.graphics.**Polygon**(**coordinates*, ***kwargs*)
Base class: *ShapeGraphic*

Polygons are shapes with an arbitrary number of vertices.

Parameters

- ***coordinates** – The coordinates as a sequence of alternating x and y values.
- **fill_color** (*str*) – Defaults to '#FFFFFF'.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **name** (*str*) – An identifable name for the Graphic.
- **line_width** – Defaults to 1.
- **line_style** (*str*) – The line pattern. Acceptable values are – (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Polygon.

Raises

- **ValueError** – if an odd number of coordinate values are given.
- **GeometryError** – if there are fewer than three vertices.

coordinates(*xy_pairs=False*)
Returns the coordinates of the Polygon. By default they will be returned as a single `tuple` of alternating x and y values.

```
(x1, y1, x2, y2, x3, y3...)
```

However, you can optionally have it return them as a tuple of xy two-tuple pairs with the `xy_pairs` argument.

```
((x1, y1), (x2, y2), (x3, y3)...)
```

Parameters **xy_pairs** (*bool*) – if True, the coordinates will be returned as xy pairs (see above).

Return type `tuple`

add_vertex(*x*, *y*)
Adds a vertex to the Polygon. The vertex will be added at the end of the list of vertices.

Parameters

- **x** – The x-value of the new vertex's coordinate.
- **y** – The y-value of the new vertex's coordinate.

remove_vertex(*index*)
Removes a the vertex at the specified index from the Polygon.

Parameters **index** (*int*) – The index of the vertex to be removed.

Raises **GeometryError** – if removing a vertex would leave the Polygon with fewer than three vertices.

data()
Returns any data associated with the Graphic as a `dict`. This `dict` is modifiable.

When rendered as SVG, the data will be turned into `attribute="value"` pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type `dict`

fill_color(*fill_color=None*)
The colour of the shape's interior space. Passing a value will update the fill_color property. All colours must be in the form #RRGGBB.

Parameters **fill_color** (*str*) – If given, the Shape's fill_color will be set to this.

Return type `str`

line_color(*line_color=None*)
The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type `str`

line_style(*line_style=None*)
The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are – (default), `..` (dotted) or −− (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type `str`

line_width(*line_width=None*)
The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type `int`

name(*name=None*)
An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type `str`

opacity(*opacity=None*)
The degree of transparency of the interior space, from 0 to 1 (0 being invisible). Passing a value will update the opacity property.

Parameters **opacity** (*floar*) – If given, the Shape's opacity will be set to this.

Return type `float`

rotate(*x_pivot*, *y_pivot*, *angle*)
Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's `rotation` property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.

- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

rotation()
The rotation applied to the Graphic. It takes the form of a three number tuple - `(pivot_x_coordinate, pivot_y_coordinate, angle)`. The angle is in degrees, and works clockwise.

Return type `tuple`

class omnicanvas.graphics.**Text**(*x*, *y*, *text*, **args*, *font_size=18*, *fill_color='#000000'*, *line_width=0*, *horizontal_align='center'*, *vertical_align='center'*, ***kwargs*)
Base class: *ShapeGraphic*

Graphics of textual information.

Parameters

- **x** – The Text's x location.
- **y** – The Text's y location.
- **text** (*str*) – The text to display.
- **font_size** – The font size of the Text when displayed.
- **horizontal_align** – The horizontal alignment of the Text. Acceptable values are `left`, `center` (default) and `right`.
- **vertical_align** – The vertical alignment of the Text. Acceptable values are `top`, `center` (default) and `bottom`.
- **fill_color** (*str*) – Defaults to '#FFFFFF'.
- **opacity** – The degree of transparency, from 0 to 1 (0 being invisible).
- **name** (*str*) – An identifable name for the Graphic.
- **line_width** – Defaults to 0.
- **line_style** (*str*) – The line pattern. Acceptable values are - (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Text.

x(*x=None*)
The x-value of the Text's location. How this point relates to the Text's location is determined by `horizontal_align` and `vertical_align`. Passing a value will update the x property.

Parameters **x** – If given, the Text's x value will be set to this.

Return type `float` or `int`

y(*y=None*)
The y-value of the Text's location. How this point relates to the Text's location is determined by `horizontal_align` and `vertical_align`. Passing a value will update the y property.

Parameters **y** – If given, the Text's y value will be set to this.

Return type `float` or `int`

text (*text=None*)
The text that the the Graphic displays. If something other than a string is passed, the `__str__` representation of that object will be used. Passing a value will update the text property.

Parameters **text** – If given, the Text's text value will be set to this.

Return type `str`

font_size (*font_size=None*)
The text's font size, in pt. Passing a value will update the font_size.

Parameters **font_size** – If given, the font_size value will be set to this.

Return type `int` or `float`

horizontal_align (*horizontal_align=None*)
The horizontal alignment of the text. If `center` the coordinate will represent the center of the Text. If `left` the Text will be to the left of the coordinate and if `right` the Text will be to the right. Passing a value will update the horizontal_align.

Parameters **horizontal_align** (*str*) – If given, the horizontal_align value will be set to this.

Raises **ValueError** – if you try to set horizontal_align to something other than the three allowed values.

Return type str

vertical_align (*vertical_align=None*)
The vertical alignment of the text. If `center` the coordinate will represent the center of the Text. If `top` the Text will be to the abovef the coordinate and if `bottom` the Text will be below. Passing a value will update the vertical_align.

Parameters **vertical_align** (*str*) – If given, the vertical_align value will be set to this.

Raises **ValueError** – if you try to set vertical_align to something other than the three allowed values.

Return type str

data()
Returns any data associated with the Graphic as a `dict`. This `dict` is modifiable.

When rendered as SVG, the data will be turned into `attribute="value"` pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type `dict`

fill_color (*fill_color=None*)
The colour of the shape's interior space. Passing a value will update the fill_color property. All colours must be in the form #RRGGBB.

Parameters **fill_color** (*str*) – If given, the Shape's fill_color will be set to this.

Return type `str`

line_color (*line_color=None*)
The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type `str`

line_style(*line_style=None*)
The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are – (default), `..` (dotted) or −− (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type str

line_width(*line_width=None*)
The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type int

name(*name=None*)
An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type str

opacity(*opacity=None*)
The degree of transparency of the interior space, from 0 to 1 (0 being invisible). Passing a value will update the opacity property.

Parameters **opacity** (*floar*) – If given, the Shape's opacity will be set to this.

Return type float

rotate(*x_pivot*, *y_pivot*, *angle*)
Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's `rotation` property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.
- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

rotation()
The rotation applied to the Graphic. It takes the form of a three number tuple - `(pivot_x_coordinate, pivot_y_coordinate, angle)`. The angle is in degrees, and works clockwise.

Return type tuple

class omnicanvas.graphics.**Polyline**(**coordinates*, ***kwargs*)
Base class: *Graphic*

Polylines are lines with an arbitrary number of vertices. Unlike Polygons, the last vertex is not joined to the first one, and so they have no interior space.

Parameters

- ***coordinates** – The coordinates as a sequence of alternating x and y values.
- **line_width** – Defaults to 1.

- **name** (*str*) – An identifable name for the Graphic.
- **line_style** (*str*) – The line pattern. Acceptable values are – (default), `..` (dotted) or `--` (dashed).
- **line_color** (*str*) – Defaults to '#000000'.
- **rotation** (*tuple*) – Any rotation to be applied, in the format (x of rotation point, y of rotation point, angle), in degrees.
- **data** (*dict*) – Any data to be associated with the Polygon.

Raises

- **ValueError** – if an odd number of coordinate values are given.
- **GeometryError** – if there are fewer than two vertices.

coordinates(*xy_pairs=False*)

Returns the coordinates of the Polyline. By default they will be returned as a single `tuple` of alternating x and y values.

```
(x1, y1, x2, y2, x3, y3...)
```

However, you can optionally have it return them as a tuple of xy two-tuple pairs with the `xy_pairs` argument.

```
((x1, y1), (x2, y2), (x3, y3)...)
```

Parameters **xy_pairs** (*bool*) – if True, the coordinates will be returned as xy pairs (see above).

Return type `tuple`

add_vertex(*x*, *y*)

Adds a vertex to the Polyline. The vertex will be added at the end of the list of vertices.

Parameters

- **x** – The x-value of the new vertex's coordinate.
- **y** – The y-value of the new vertex's coordinate.

remove_vertex(*index*)

Removes a the vertex at the specified index from the Polyline.

Parameters **index** (*int*) – The index of the vertex to be removed.

Raises **GeometryError** – if removing a vertex would leave the Polyline with fewer than two vertices.

data()

Returns any data associated with the Graphic as a `dict`. This `dict` is modifiable.

When rendered as SVG, the data will be turned into `attribute="value"` pairs in the SVG element, and is useful in, among other things, assigning JavaScript events to individual elements.

Return type `dict`

line_color(*line_color=None*)

The colur of the Graphic's edge. Passing a value will update the line_color property. All colours must be in the form #RRGGBB.

Parameters **line_color** (*str*) – If given, the Graphic's line_color will be set to this.

Return type `str`

line_style(*line_style=None*)
The line pattern of the Graphic's edge. Passing a value will update the line_style property. Acceptable values are – (default), `..` (dotted) or −− (dashed).

Parameters **line_style** (*str*) – If given, the Graphic's line_style will be set to this.

Return type str

line_width(*line_width=None*)
The width of the Graphic's edge in pixels. Passing a value will update the line_width property.

Parameters **line_width** – If given, the Graphic's line_width will be set to this.

Return type int

name(*name=None*)
An identifable name for the Graphic. Passing a value will update the name property.

Parameters **name** – If given, the Graphic's name will be set to this.

Return type str

rotate(*x_pivot*, *y_pivot*, *angle*)
Rotates the Graphic about a point. The rotation will be clockwise, and the angle given must be between 0 and 360.

Note: rotations do not sum. If you rotate a Graphic once by 30° and then again by 100° about the same point, the end result will just be a rotation of 100° around that point, *not* 130°. This is because the method just sets the Graphic's `rotation` property to whatever is given.

Parameters

- **x_pivot** (*str*) – The x value of the pivot point.
- **y_pivot** (*str*) – The y value of the pivot point.
- **angle** (*str*) – The (clockwise) angle of rotation, in degrees.

Raises **ValueError** – if the angle is not between 0 and 360.

rotation()
The rotation applied to the Graphic. It takes the form of a three number tuple - `(pivot_x_coordinate, pivot_y_coordinate, angle)`. The angle is in degrees, and works clockwise.

Return type tuple

2.4.3 omnicanvas.colors (Tools for processing colours)

This module contains various functions usued internally to process colours, mainly for validation pruposes currently.

omnicanvas.colors.**process_color**(*color*)
Raises exceptions if a colour does not meet requirements.

Parameters **color** (*str*) – The colour to check.

Raises

- **TypeError** – if the colour is not a string.
- **ValueError** – if the colour is not valid.

omnicanvas.colors.**is_valid_color**(*color*)
Checks that a given string represents a valid hex colour.

Parameters **color** (*str*) – The color to check.

Return type bool

omnicanvas.colors.**hsl_to_rgb**(*hue*, *saturation*, *lightness*)
Takes a colour in HSL format and produces an RGB string in the form #RRGGBB.

Parameters

- **hue** – The Hue value (between 0 and 360).
- **saturation** – The Saturation value (between 0 and 100).
- **lightness** – The Lightness value (between 0 and 100).

Raises **ValueError** – if any of the three parameters are outside their bounds.

2.4.4 omnicanvas.exceptions (Custom exceptions)

OmniCanvas custom exceptions.

exception omnicanvas.exceptions.**GeometryError**
The exception raised if a shape is created that defies the laws of geometry. If this exception were not thrown the universe would be destroyed.

2.5 Changelog

2.5.1 Release 0.2.2

15 December 2016

- Fixed inconsistency in whether the central vertical alignment is 'middle' or 'center'

2.5.2 Release 0.2.1

14 December 2016

- Graphics adding methods now return a reference to the graphic just added.
- Added names to graphics, and methods for retrieving them from the canvas by name.

2.5.3 Release 0.2.0

12 December 2016

- All attributes are now method properties.
- Added function for converting HSL colours to RGB strings.
- Added Polyline Graphic.

2.5.4 Release 0.1.0

6 June 2016

- Canvas with four graphic types:
  - Rectangles
  - Lines
  - Polygons
  - Text
- SVG output

O

A

B

C

D

F

G

H

I

L

N

O

P

R

S

T

V

W

X

Y