svgutils Documentation

Release 0.1

Bartosz Telenczuk

Oct 25, 2017

Contents

1 Contents **1**
1.1 Tutorials 1
1.2 Reference 8

2 Indices and tables **21**

Bibliography **23**

Python Module Index **25**

CHAPTER 1

Contents

Tutorials

Creating plublication-quality figures

Matplotlib is a decent Python library for creating publication-quality plots which offers a multitude of different plot types. However, one limitation of `matplotlib` is that creating complex layouts can be at times complicated. Therefore, post-processing of plots is usually done in some other vector graphics editor such as inkscape or Adobe Illustrator. The typical workflow is as following:

1. Import and analyse data in Python
2. Create figures in `matplotlib`
3. Export figures to PDF/SVG
4. Import figures to vector-graphics editor
5. Arrange and edit figures manually
6. Export the figure to PDF

As you probably see, the typical workflow is quite complicated. To make things worse you may need to repeat the process several times, when, for example, you want to include more data into the analysis. This includes manual editing and arranging the figure, which is obviously time consuming. Therefore it makes sense to try and automate the process. Here, I will describe an automatic workflow which completely resides on Python tools.

1. *Create plots*

   First you need to create nice matplotlib-based plots you would like to compose your figure from. You may download the scripts I will use in the example from github repository: anscombe.py and sigmoid_fit.py.

2. *Export to SVG*

   A nice feature of matplotlib is that it allows to export figure to Scalable Vector Graphics (SVG) which is an open

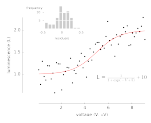


Fig. 1.1: `sigmoid_fit.py`

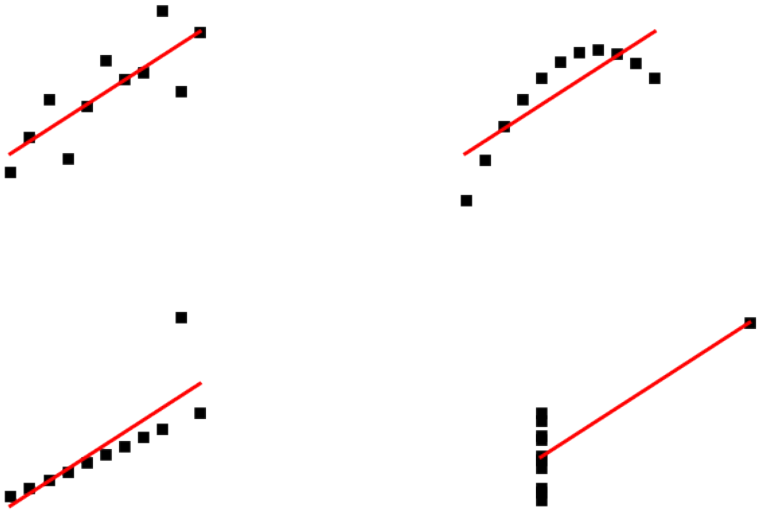


Fig. 1.2: `anscombe.py`

vector format[1] understood by many applications (such as Inkscape, Adobe Illustrator or even web browsers). Not going too much into details, I will only say that SVG files are text files with special predefined tags (much alike HTML tags). You may try to open one of them in a text editor to find out what I mean.

3. *Arrange plots into composite figures*

   Now, we would like to combine both plots into one figure and add some annotations (such as one-letter labels: A,B, etc.). To this end, I will use a small Python package I wrote with this purpose `svgutils`. It is written completely in Python and uses only standard libraries. You may download it from github.

   The basic operations are similar to what you would do in a vector graphics editor, but instead of using a mouse you will do some scripting (I am sure you love it as much as I do). It may take some more time at the beginning, but with the advantage that you will not have to repeat the process when, for some reason, you need to modify the plots you generated with matplotlib (to add more data or modify the parameters of your analysis, just to name a few reasons).

   An example script is shown and explained below:

```
import svgutils.transform as sg
import sys

#create new SVG figure
fig = sg.SVGFigure("16cm", "6.5cm")

# load matpotlib-generated figures
fig1 = sg.fromfile('sigmoid_fit.svg')
fig2 = sg.fromfile('anscombe.svg')

# get the plot objects
plot1 = fig1.getroot()
plot2 = fig2.getroot()
plot2.moveto(280, 0, scale=0.5)

# add text labels
txt1 = sg.TextElement(25,20, "A", size=12, weight="bold")
txt2 = sg.TextElement(305,20, "B", size=12, weight="bold")

# append plots and labels to figure
fig.append([plot1, plot2])
fig.append([txt1, txt2])

# save generated SVG files
fig.save("fig_final.svg")
```

4. *Convert to PDF/PNG*

   After running the script, you may convert the output file to a format of your choice. To this end, you can use `inkscape` which can produce PNG and PDF files from SVG source. You can do that directly from command line without the need of opening the whole application:

```
inkscape --export-pdf=fig_final.pdf fig_final.svg
inkscape --export-png=fig_final.png fig_final.svg
```

   And here is the final result:

Now, whenever you need to re-do the plots you can simply re-run the above scripts. You can also automate the process

[1] In case you do not know it, a vector format in contrast to other (raster) formats such as PNG, JPEG does not represent graphics as individual pixels, but rather as modifiable objects (lines, circles, points etc.). They usually offer better qualitiy for publication plots (PDF files are one of them) and are also editable.

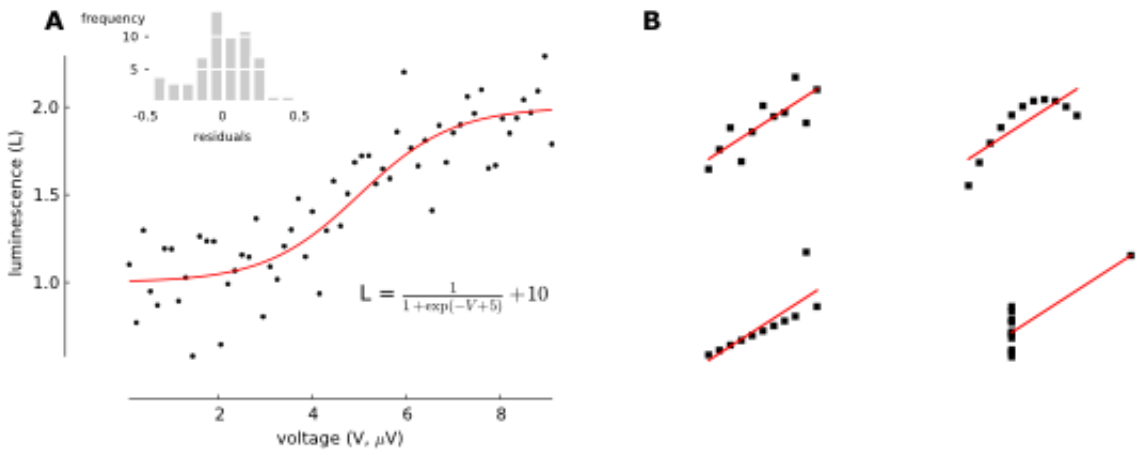


Fig. 1.3: Final publication-ready figure.

by means of a build system, such as GNU `make` or similar. This part will be covered in some of the next tutorials from the series.

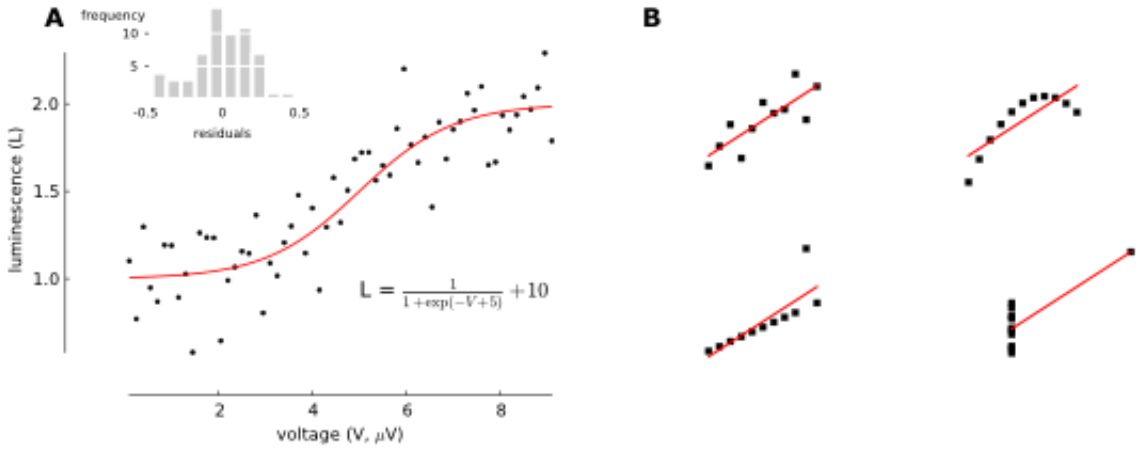
Good luck and happy plotting!

PS If you have a better/alternative method for creating your publication plots, I would be very interested in learning about it. Please comment or mail me!

Composing multi-panel figures

As I already explained in the previous tutorial, creating figures programmatically has many advantages. However, obtaining a complex layout only by scripting can be very time consuming and even distressing. Therefore, the possible gains can be crippled by the time spent tweaking the programs to obtain optimal results and under time pressure many of us resort to visual editors. One way to alleviate the problem is to use a library with little boilerplate code and which simplifies the common tasks (such as inserting a new panel and adjusting its position). That's why I introduced the compose module, which is a wrapper around the low-level API described in *Creating plublication-quality figures*.

Let's take the example from the previous tutorial



To obtain this nicely-formatted final figure we needed a *considerable* amount of code. The same effect could be achieved in `compose` with fewer lines of code:

```
#!/usr/bin/env python
#coding=utf-8

from svgutils.compose import *

Figure("16cm", "6.5cm",
        Panel(
              SVG("sigmoid_fit.svg"),
              Text("A", 25, 20, size=12, weight='bold')
             ),
        Panel(
              SVG("anscombe.svg").scale(0.5),
              Text("B", 25, 20, size=12, weight='bold')
             ).move(280, 0)
        ).save("fig_final_compose.svg")
```

The `compose` module offers the same functionality as the `transform`, but rather than being based on procedural description of the figure it attempts declarative approach. The code defining the figure mimics a hierarchical structure typical of most figures: A figure contains multiple panels; these panels can in turn contain several graphical elements such as text, markers or other (sub-)panels.

Defining a figure

Before we start we need to import the definitions from `svgutils.compose` module:

```
from svgutils.compose import *
```

In *compose* the top-most element is the `Figure()` object. To create a figure we need to specify its size (width and height) and its contents. For example, to create a figure consisting of a single imported SVG file we might write:

```
Figure("16cm", "6.5cm",
       SVG("sigmoid_fit.svg")
       )
```

This will create a 16-by-6.5 cm figure with showing the `sigmoid_fit.svg` file. Note that the dimensions can be defined together with units supported by SVG (so far "px" and "cm" are implemented). If no units are defined it defaults to "px". `SVG()` is another object from `compose` module, which simply parses and pastes the content of a SVG file into the figure.

The `Figure()` object also defines several methods; the `save()` method saves the figure in a SVG file:

Listing 1.1: `Figure preview`

```
Figure("16cm", "6.5cm",
       SVG("sigmoid_fit.svg")
       ).save("fig1.svg")
```

Adding annotations

The simple example of previous section is superfluous, because it does not modify the `sigmoid_fit.svg` file apart from changing its size. Let us try then overlaying some text on top of the figure. In `compose` we can add text using `Text()` object:

Listing 1.2: `Figure preview`

```
Figure("16cm", "6.5cm",
        Text("A", 25, 20),
        SVG("sigmoid_fit.svg")
       )
```

In addition to the text itself we defined the x and y coordinates of the text element in pixel units. We can also add additional style arguments – to increase the font size and change to bold letters we can use:

Listing 1.3: `Figure preview`

```
Figure("16cm", "6.5cm",
        Text("A", 25, 20, size=12, weight='bold'),
        SVG("sigmoid_fit.svg")
       )
```

Arranging multiple elements

We can combine multiple SVG drawings by simply listing them inside the `Figure()` object:

Listing 1.4: `Figure preview`

```
Figure("16cm", "6.5cm",
       SVG("sigmoid_fit.svg"),
       SVG("anscombe.svg")
       )
```

The problem with this figure is that the drawings will overlap and become quite unreadable. To avoid it we have to move figure elements. To do that automatically you can use `tile()` method of `Figure()`, which arranges the elements on a regular two-dimensional grid. For example, to arrange the two SVG elements in a single row we might use:

Listing 1.5: `Figure preview`

```
 Figure("16cm", "6.5cm",
        SVG("sigmoid_fit.svg"),
        SVG("anscombe.svg")
        ).tile(2, 1)
```

The second figure (`anscombe.svg`) does not fit entirely in the figure so we have to scale it down. For this aim each element of the Figure exposes a `scale()` method, which takes the scaling factor as its sole argument:

Listing 1.6: `Figure preview`

```
Figure("16cm", "6.5cm",
       SVG("sigmoid_fit.svg"),
       SVG("anscombe.svg").scale(0.5)
       ).tile(2, 1)
```

For more control over the final figure layout we can position the individual elements using their `move()` method:

Listing 1.7: `Figure preview`

```
Figure("16cm", "6.5cm",
       SVG("sigmoid_fit.svg"),
       SVG("anscombe.svg").move(280, 0)
       )
```

This will move the `ansombe.svg` 280 px horizontally. Methods can be also chained:

Listing 1.8: `Figure preview`

```
Figure("16cm", "6.5cm",
       SVG("sigmoid_fit.svg"),
       SVG("anscombe.svg").scale(0.5)
                          .move(280, 0)
       )
```

It's often difficult to arrange the figures correctly and it can involve mundane going back and fro between the code and generated SVG file. To ease the process `compose` offers several helper objects: The `Grid()` object generates a grid of horizontal and vertical lines labelled with their position in pixel units. To add it simply list `Grid()` as one of `Figure()` elements:

Listing 1.9: `Figure preview`

```
Figure("16cm", "6.5cm",
       SVG("sigmoid_fit.svg"),
       SVG("anscombe.svg").scale(0.5)
                          .move(280, 0),
       Grid(20, 20)
       )
```

The two parameters of `Grid()` define the spacing between the vertical and horizontal lines, respectively. You can use the lines and numerical labels to quickly estimate the required vertical and horizontal shifts of the figure elements.

Grouping elements into panels

Figures prepared for publications often consist of sub-panels, which can contain multiple elements such as graphs, legends and annotations (text, arrows etc.). Although it is possible to list all these elements separately in the `Figure()` object, it's more convenient to work with all elements belonging to a single panel as an entire group. In `compose` one can group the elements into panels using `Panel()` object:

Listing 1.10: `Figure preview`

```
Figure("16cm", "6.5cm",
       Panel(
          Text("A", 25, 20),
          SVG("sigmoid_fit.svg")
          ),
       Panel(
          Text("B", 25, 20).move(280, 0),
          SVG("anscombe.svg").scale(0.5)
                             .move(280, 0)
          )
       )
```

`Panel()` just like a `Figure()` object takes a list of elements such as text objects or SVG drawings. However, in

contrast to `Figure()` it does not allow to define the size and does not offer `save()` method. The two `Panel()` objects of this example contain each a text element and a SVG file.

In this example the `Panel()` object serve no other role than grouping elements that refer to a single panel – it may enhance the readability of the code generating the figure, but it does not simplify the task of creating the figure. In the second `Panel()` we apply twice the method `move()` to position both the text element and the SVG. The advantage of `Panel()` is that we can apply such transforms to the entire panel:

Listing 1.11: `Figure preview`

```
Figure("16cm", "6.5cm",
       Panel(
          Text("A", 25, 20),
          SVG("sigmoid_fit.svg")
          ),
       Panel(
          Text("B", 25, 20),
          SVG("anscombe.svg").scale(0.5)
          ).move(280, 0)
       )
```

This way we simplified the code, but also the change allows for easier arrangement of the panels. An additional advantage is that the `tile()` method will automatically arrange the entire panels not the individual elements.

Reference

`transform` – basic SVG transformations

This module implements low-level API allowing to open and manipulate SVG files. An example use is described in the *Creating plublication-quality figures* tutorial.

class svgutils.transform.**FigureElement**(*xml_element*, *defs=None*)
Base class representing single figure element

Methods

copy()	Make a copy of the element
find_id(element_id)	Find element by its id.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

copy()
Make a copy of the element

find_id(*element_id*)
Find element by its id.

Parameters **element_id** : str

ID of the element to find

Returns FigureElement

one of the children element with the given ID.

moveto(*x*, *y*, *scale=1*)
Move and scale element.

Parameters **x, y** : float

displacement in x and y coordinates in user units ('px').

scale : float

scaling factor. To scale down scale < 1, scale up scale > 1. For no scaling scale = 1.

rotate(*angle*, *x=0*, *y=0*)
Rotate element by given angle around given pivot.

Parameters **angle** : float

rotation angle in degrees

x, y : float

pivot coordinates in user coordinate system (defaults to top-left corner of the figure)

scale_xy(*x=0*, *y=None*)

Scale element separately across the two axes x and y. If y is not provided, it is assumed equal to x (according to the W3 specification).

Parameters **x** : float

x-axis scaling factor. To scale down x < 1, scale up x > 1.

y : (optional) float

y-axis scaling factor. To scale down y < 1, scale up y > 1.

skew(*x=0*, *y=0*)
Skew the element by x and y degrees Convenience function which calls skew_x and skew_y

Parameters **x,y** : float, float

skew angle in degrees (default 0)

If an x/y angle is given as zero degrees, that transformation is omitted.

skew_x(*x*)
Skew element along the x-axis by the given angle.

Parameters **x** : float

x-axis skew angle in degrees

skew_y(*y*)
Skew element along the y-axis by the given angle.

Parameters **y** : float

y-axis skew angle in degrees

tostr()
String representation of the element

class svgutils.transform.**GroupElement**(*element_list*, *attrib=None*)
Group element.

Container for other elements. Corresponds to SVG <g> tag.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find element by its id.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.transform.**ImageElement**(*stream*, *width*, *height*, *format='png'*)
Inline image element.

Correspoonds to SVG <image> tag. Image data encoded as base64 string.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find element by its id.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.transform.**LineElement**(*points*, *width=1*, *color='black'*)
Line element.

Corresponds to SVG <path> tag. It handles only piecewise straight segments

Methods

copy()	Make a copy of the element
find_id(element_id)	Find element by its id.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees

Continued on next page

Table 1.4 – continued from previous page

skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.transform.**SVGFigure**(*width=None*, *height=None*)

SVG Figure.

It setups standalone SVG tree. It corresponds to SVG <svg> tag.

Attributes

height	Figure height
width	Figure width

Methods

append(element)	Append new element to the SVG figure
find_id(element_id)	Find elements with the given ID
get_size()	Get figure size
getroot()	Return the root element of the figure.
save(fname)	Save figure to a file
set_size(size)	Set figure size
to_str()	Returns a string of the SVG figure.

append(*element*)

Append new element to the SVG figure

find_id(*element_id*)

Find elements with the given ID

get_size()

Get figure size

getroot()

Return the root element of the figure.

The root element is a group of elements after stripping the toplevel <svg> tag.

Returns GroupElement

All elements of the figure without the <svg> tag.

height

Figure height

save(*fname*)

Save figure to a file

set_size(*size*)

Set figure size

to_str()

Returns a string of the SVG figure.

width
Figure width

class svgutils.transform.**TextElement**(*x*, *y*, *text*, *size=8*, *font='Verdana'*, *weight='normal'*, *letterspacing=0*, *anchor='start'*, *color='black'*)

Text element.

Corresponds to SVG <text> tag.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find element by its id.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

svgutils.transform.**from_mpl**(*fig*, *savefig_kw*)

Create a SVG figure from a matplotlib figure.

Parameters **fig** : matplotlib.Figure instance

savefig_kw : dict

keyword arguments to be passed to matplotlib's *savefig*

Returns SVGFigure

newly created *SVGFigure* initialised with the string content.

Examples

If you want to overlay the figure on another SVG, you may want to pass the *transparent* option:

```
>>> from svgutils import transform
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> line, = plt.plot([1,2])
>>> svgfig = transform.from_mpl(fig,
...                savefig_kw=dict(transparent=True))
>>> svgfig.getroot()
<svgutils.transform.GroupElement object at ...>
```

svgutils.transform.**fromfile**(*fname*)

Open SVG figure from file.

Parameters **fname** : str

name of the SVG file

Returns SVGFigure

newly created *SVGFigure* initialised with the file content

svgutils.transform.**fromstring**(*text*)
Create a SVG figure from a string.

Parameters **text** : str

string representing the SVG content. Must be valid SVG.

Returns SVGFigure

newly created *SVGFigure* initialised with the string content.

compose – easy figure composing

compose module is a wrapper on top of *svgutils.transform* that simplifies composing SVG figures. Here is a short example of how a figure could be constructed:

```
Figure( "10cm", "5cm",
        SVG('svg_logo.svg').scale(0.2),
        Image(120, 120, 'lion.jpeg').move(120, 0)
      ).save('test.svg')
```

SVG definitions designed for easy SVG composing

Features:

- allow for wildcard import
- defines a mini language for SVG composing
- short but readable names
- easy nesting
- method chaining
- **no boilerplate code (reading files, extracting objects from svg,** transversing XML tree)
- universal methods applicable to all element types
- dont have to learn python

class svgutils.compose.**Element**(*xml_element*, *defs=None*)
Base class for new SVG elements.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

find_id(*element_id*)
Find a single element with the given ID.

Parameters **element_id** : str

ID of the element to find

Returns found element

find_ids(*element_ids*)
Find elements with given IDs.

Parameters **element_ids** : list of strings

list of IDs to find

Returns a new *Panel* object which contains all the found elements.

move(*x*, *y*)
Move the element by x, y.

Parameters **x,y** : int, str

amount of horizontal and vertical shift

Notes

The x, y can be given with a unit (for example, "3px", "5cm"). If no unit is given the user unit is assumed ("px"). In SVG all units are defined in relation to the user unit *[R114]*.

scale(*factor*)
Scale SVG element.

Parameters **factor** : float

The scaling factor.

Factor > 1 scales up, factor < 1 scales down.

class svgutils.compose.**Figure**(*width*, *height*, **svgelements*)
Main figure class.

This should be always the top class of all the generated SVG figures.

Parameters **width, height** : float or str

Figure size. If unit is not given, user units (px) are assumed.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
save(fname)	Save figure to SVG file.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.

Continued on next page

Table 1.9 – continued from previous page

skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tile(ncols, nrows)	Automatically tile the panels of the figure.
tostr()	String representation of the element

save (*fname*)
Save figure to SVG file.

Parameters **fname** : str

Full path to file.

tile (*ncols*, *nrows*)
Automatically tile the panels of the figure.

This will re-arranged all elements of the figure (first in the hierarchy) so that they will uniformly cover the figure area.

Parameters **ncols, nrows** : type

The number of columns and rows to arange the elements into.

Notes

ncols * nrows must be larger or equal to number of elements, otherwise some elements will go outside the figure borders.

class svgutils.compose.**Grid**(*dx*, *dy*, *size=8*)
Line grid with coordinate labels to facilitate placement of new elements.

Parameters **dx** : float

Spacing between the vertical lines.

dy : float

Spacing between horizontal lines.

size : float or str

Font size of the labels.

Notes

This element is mainly useful for manual placement of the elements.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.

Continued on next page

Table 1.10 – continued from previous page

rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.compose.**Image**(*width*, *height*, *fname*)

Raster or vector image

Parameters **width** : float

height : float

image dimensions

fname : str

full path to the file

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.compose.**Line**(*points*, *width=1*, *color='black'*)

Line element connecting given points.

Parameters **points** : sequence of tuples

List of point x,y coordinates.

width : float, optional

Line width.

color : str, optional

Line color. Any of the HTML/CSS color definitions are allowed.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.compose.**Panel**(**svgelements*)

Figure panel.

Panel is a group of elements that can be transformed together. Usually it relates to a labeled figure panel.

Parameters **svgelements** : objects derving from Element class

one or more elements that compose the panel

Notes

The grouped elements need to be properly arranged in scale and position.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.compose.**SVG**(*fname*)

SVG from file.

Parameters **fname** : str

full path to the file

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.compose.**Text**(*text*, *x=None*, *y=None*, ***kwargs*)

Text element.

Parameters **text** : str

content

x, y : float or str

Text position. If unit is not given it will assume user units (px).

size : float, optional

Font size.

weight : str, optional

Font weight. It can be one of: normal, bold, bolder or lighter.

font : str, optional

Font family.

Methods

copy()	Make a copy of the element
find_id(element_id)	Find a single element with the given ID.
find_ids(element_ids)	Find elements with given IDs.
move(x, y)	Move the element by x, y.
moveto(x, y[, scale])	Move and scale element.
rotate(angle[, x, y])	Rotate element by given angle around given pivot.
scale(factor)	Scale SVG element.
scale_xy([x, y])	Scale element separately across the two axes x and y.
skew([x, y])	Skew the element by x and y degrees
skew_x(x)	Skew element along the x-axis by the given angle.
skew_y(y)	Skew element along the y-axis by the given angle.
tostr()	String representation of the element

class svgutils.compose.**Unit**(*measure*)

Implementaiton of SVG units and conversions between them.

Parameters **measure** : str

value with unit (for example, '2cm')

Methods

to(unit)	Convert to a given unit.

to(*unit*)
Convert to a given unit.

Parameters **unit** : str

Name of the unit to convert to.

Returns **u** : Unit

new Unit object with the requested unit and computed value.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Bibliography

[R114] W3C SVG specification: https://www.w3.org/TR/SVG/coords.html#Units

Python Module Index

S

svgutils.compose, 13
svgutils.transform, 8

Index

A

append() (svgutils.transform.SVGFigure method), 11

C

copy() (svgutils.transform.FigureElement method), 8

E

Element (class in svgutils.compose), 13

F

Figure (class in svgutils.compose), 14
FigureElement (class in svgutils.transform), 8
find_id() (svgutils.compose.Element method), 14
find_id() (svgutils.transform.FigureElement method), 8
find_id() (svgutils.transform.SVGFigure method), 11
find_ids() (svgutils.compose.Element method), 14
from_mpl() (in module svgutils.transform), 12
fromfile() (in module svgutils.transform), 12
fromstring() (in module svgutils.transform), 12

G

get_size() (svgutils.transform.SVGFigure method), 11
getroot() (svgutils.transform.SVGFigure method), 11
Grid (class in svgutils.compose), 15
GroupElement (class in svgutils.transform), 9

H

height (svgutils.transform.SVGFigure attribute), 11

I

Image (class in svgutils.compose), 16
ImageElement (class in svgutils.transform), 10

L

Line (class in svgutils.compose), 16
LineElement (class in svgutils.transform), 10

M

move() (svgutils.compose.Element method), 14
moveto() (svgutils.transform.FigureElement method), 9

P

Panel (class in svgutils.compose), 17

R

rotate() (svgutils.transform.FigureElement method), 9

S

save() (svgutils.compose.Figure method), 15
save() (svgutils.transform.SVGFigure method), 11
scale() (svgutils.compose.Element method), 14
scale_xy() (svgutils.transform.FigureElement method), 9
set_size() (svgutils.transform.SVGFigure method), 11
skew() (svgutils.transform.FigureElement method), 9
skew_x() (svgutils.transform.FigureElement method), 9
skew_y() (svgutils.transform.FigureElement method), 9
SVG (class in svgutils.compose), 17
SVGFigure (class in svgutils.transform), 11
svgutils.compose (module), 13
svgutils.transform (module), 8

T

Text (class in svgutils.compose), 18
TextElement (class in svgutils.transform), 12
tile() (svgutils.compose.Figure method), 15
to() (svgutils.compose.Unit method), 19
to_str() (svgutils.transform.SVGFigure method), 11
tostr() (svgutils.transform.FigureElement method), 9

U

Unit (class in svgutils.compose), 18

W

width (svgutils.transform.SVGFigure attribute), 11