svgwrite Documentation

Release 1.4.3

Manfred Moitzi <mozman@gmx.at>

Jul 14, 2022

Contents

Welcome! This is the documentation for svgwrite 1.4.3, last updated Jul 14, 2022.

This package is in maintenance mode, no new features will be added, there will be no change of behavior, just bugfixes will be merged.

If you are looking for a SVG Documentation beyond the official W3C papers, go to O'Reilly commons: http://commons.oreilly.com/wiki/index.php/SVG_Essentials, or (german) http://www.selfsvg.info .

CHAPTER 1

Overview

As the name *svgwrite* implies, *svgwrite* creates new SVG drawings, it does not read/import existing drawings, but you can always include other SVG drawings by the <*image*> entity.

svgwrite has a debugging feature, activated by `svgwrite.Drawing(debug=True)`. This feature is meant to find SVG errors produced by your code while developing. This validation algorithm is not optimized and therefore very slow for big SVG files. That will not change in the future. And because it is slow DON'T use it in production code!

If *svgwrite* without debugging is still too slow, you have to use the *lxml* package without *svgwrite* as wrapper. That is the ugly truth since *svgwrite* is just a wrapper around *xml.etree.ElementTree*. If you learn the ElementTree API and the SVG elements and attributes, you do not need *svgwrite*.

1.1 SVG Elements

Important: Use the **factory-methods** of the class **Drawing** to create new objects. (This is necessary to support validation for different SVG versions.) All **factory-methods** have the original SVG Elementname (e.g. Drawing.a(...), Drawing.g(...), Drawing.symbol(...), Drawing.line(...))

a short example:

```
dwg = svgwrite.Drawing()
link = dwg.add(dwg.a("http://link.to/internet"))
square = link.add(dwg.rect((0, 0), (1, 1), fill='blue'))
```

1.1.1 Structural Elements

Drawing, *SVG*, *Group*, *Defs*, *Symbol*, *Marker*, *Use*, *Hyperlink*

1.1.2 Graphical Elements

Line, *Rect*, *Circle*, *Ellipse*, *Polyline*, *Polygon*, *Path*

1.1.3 Text Objects

Text, *TSpan*, *TRef*, *TextPath*, *TextArea*,

1.1.4 Paint Server

LinearGradient, *RadialGradient*, *Pattern*,

1.1.5 Masking

Mask, *ClipPath*

1.1.6 Animation

Set, *Animate*, *AnimateColor*, *AnimateMotion*, *AnimateTransform*

1.1.7 Filter Effects

Filter

1.2 Mixins

ViewBox, *Transform*, *XLink*, *Presentation*, *MediaGroup*, *Markers*, *Clipping*,

1.3 Common Attributes

1.3.1 Core Attributes

W3C Reference: http://www.w3.org/TR/SVG11/intro.html#TermCoreAttribute

The core attributes are those attributes that can be specified on any SVG element.

W3C Direct Links

- id
- xml:base
- xml:lang
- xml:space

1.3.2 Conditional Processing Attributes

W3C Reference: http://www.w3.org/TR/SVG11/intro.html#TermConditionalProcessingAttribute

A conditional processing attribute is one that controls whether or not the element on which it appears is processed. Most elements, but not all, may have conditional processing attributes specified on them.

W3C Direct Links

- requiredExtensions
- requiredFeatures
- systemLanguage

1.3.3 Document Event Attributes

W3C Reference: http://www.w3.org/TR/SVG11/intro.html#TermDocumentEventAttribute

A document event attribute is an event attribute that specifies script to run for a particular document-wide event.

W3C Direct Links

- onabort
- onerror
- onresize
- onscroll
- onunload
- onzoom

1.3.4 Graphical Event Attributes

W3C Reference: http://www.w3.org/TR/SVG11/intro.html#TermGraphicalEventAttribute

A graphical event attribute is an event attribute that specifies script to run for a particular user interaction event.

W3C Direct Links

- onactivate
- onclick
- onfocusin
- onfocusout
- onload
- onmousedown
- onmousemove
- onmouseover

- onmouseout
- onmouseup

1.3.5 Presentation Attributes

W3C Reference: http://www.w3.org/TR/SVG11/intro.html#TermPresentationAttribute

An XML attribute on an SVG element which specifies a value for a given property for that element.

W3C Direct Links

- alignment-baseline
- baseline-shift
- clip
- clip-path
- clip-rule
- color
- color-interpolation
- color-interpolation-filters
- color-profile
- color-rendering
- cursor
- direction
- display
- dominant-baseline
- enable-background
- fill
- fill-opacity
- fill-rule
- filter
- flood-color
- flood-opacity
- font-family
- font-size
- font-size-adjust
- font-strech
- font-style
- font-variant
- font-weight

- glyph-orientation-horizontal
- glyph-orientation-vertical
- image-rendering
- kerning
- letter-spacing
- lighting-color
- marker-end
- marker-mid
- marker-start
- mask
- opacity
- overflow
- pointer-events
- shape-rendering
- stop-color
- stop-opacity
- stroke
- stroke-dasharray
- stroke-dashoffset
- stroke-linecap
- stroke-linejoin
- stroke-miterlimit
- stroke-opacity
- stroke-width
- text-anchor
- text-decoration
- text-rendering
- unicode-bidi
- visibility
- word-spacing
- writing-mode

1.3.6 XLink Attributes

W3C Reference: http://www.w3.org/TR/SVG11/intro.html#TermXLinkAttributes

The XLink attributes are the seven attributes defined in the XML Linking Language specification XLINK , which are used on various SVG elements that can reference resources. The most import XLink attribute is *xlink:href*, whose definition can be found on each element that allows it.

W3C Direct Links

- xlink:href
- xlink:type
- xlink:role
- xlink:arcrole
- xlink:title
- xlink:show
- xlink:acutate

1.4 Basic Data Types

W3C: http://www.w3.org/TR/SVG11/types.html

You can always use python-types (*int*, *float*) for length, coordinate or angle values, for length and coordinates the default unit is `px`, for angles the default unit is `deg`, or you can use a string including a unit (e.g. `100in`, `1.5cm`, `3.141529rad`).

Examples:

```
Drawing(height=100, width=100) # drawing area of 100px x 100px
Drawing(height='10cm', width='20cm') # drawing area of 10cm x 20cm
```

1.5 Numbers

Numbers can be intergers or floats, also in scientific notation:

Note: *tiny profile*: numbers must **not** have more than 4 decimal digits in the fractional part of their decimal expansion and must be in the range -32,767.9999 to +32,767.9999, inclusive

---

Examples:

- 10, -23, 0
- 73.1234, -0.002, .154, -.897, +13.2, 0000.123
- 1.24E+2, 1.24e+2,1E0, -.0E-1

1.6 Angles

The <angle> unit identifier is optional. If not provided, the angle value is assumed to be in degrees.

unit	identifier	description
deg	angle in degrees	(full circle is 360deg)
grad	angle in grads	(full circle is 400grad)
rad	angle in radians	(full circle is 2*PI)

1.7 Length

A <*length*> is a distance measurement, given as a number along with a unit, the unit identifiers must be in lower case. The meaning of a percentage length value depends on the attribute for which the percentage length value has been specified.

Two common cases are:

1. when a percentage length value represents a percentage of the viewport *width* or *height*, and
2. when a percentage length value represents a percentage of the bounding box *width* or *height* on a given object.

1.8 Coordinates

A <*coordinate*> is a length in the user coordinate system that is the given distance from the origin of the user coordinate system along the relevant axis (the x-axis for X coordinates, the y-axis for Y coordinates). Its syntax is the same as that for <*length*>.

1.9 Units

W3C: http://www.w3.org/TR/SVG11/coords.html#Units

When a coordinate or length value is a number without a unit identifier (e.g., "25"), then the given coordinate or length is assumed to be in user units (i.e., a value in the current user coordinate system).

Absolute units identifiers are only recommended for the *width* and the *height* on and situations where the content contains no transformations and it is desirable to specify values relative to the device pixel grid or to a particular real world unit size.

Note: *tiny profile*: no usage of units except for the *width* and *height* attributes of the Drawing object.

---

unit	identifier description
px	one px unit is defined to be equal to one user unit
em	font-size (actual font height)
ex	x-height (height of letter 'x' of actual font)
pt	point "1pt" equals "1.25px" (and therefore 1.25 user units)
pc	pica "1pc" equals "15px" (and therefore 15 user units)
mm	millimeter "1mm" would be "3.543307px" (3.543307 user units)
cm	centimeter "1cm" equals "35.43307px" (and therefore 35.43307 user units)
in	inch "1in" equals "90px" (and therefore 90 user units)

CHAPTER 2

SVG References

- W3C (en): http://www.w3.org/Graphics/SVG/
- W3C SVG 1.1 (en): http://www.w3.org/TR/SVG11/
- W3C SVG Tiny 1.2 (en): http://www.w3.org/TR/SVGMobile12/
- W3C SVGMobile (1.1 tiny und basic) (de): http://www.schumacher-netz.de/TR/2003/REC-SVGMobile-20030114-de.html
- SVG on Wikibooks (en): http://en.wikibooks.org/wiki/SVG
- SVG on Wikibooks (de): http://de.wikibooks.org/wiki/Svg
- SVG Authoring Guidelines: http://jwatt.org/svg/authoring/ by Jonathan Watt

CHAPTER 3

Additional SVG Documentation

- O'Reilly commons: http://commons.oreilly.com/wiki/index.php/SVG_Essentials
- SelfSVG (de): http://www.selfsvg.info

CHAPTER 4

SVG Implementation Status

- Firefox: http://www.mozilla.org/projects/svg/status.html
- Opera:
  - Elements: http://www.opera.com/docs/specs/opera95/svg/elements.xml
  - Attributes: http://www.opera.com/docs/specs/opera95/svg/attributes.xml
  - SVG-CSS: http://www.opera.com/docs/specs/opera95/svg/cssproperties.xml
  - SVG-DOM: http://www.opera.com/docs/specs/opera95/svg/dominterfaces.xml
- Webkit: http://webkit.org/projects/svg/status.xml

CHAPTER 5

svgwrite module

A Python library to create SVG drawings.

SVG is a language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composed into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects.

SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declarative (i.e., by embedding SVG animation elements in SVG content) or via scripting.

See also:

http://www.w3.org/TR/SVG11/intro.html#AboutSVG

a simple example:

```
import svgwrite

dwg = svgwrite.Drawing('test.svg', profile='tiny')
dwg.add(dwg.line((0, 0), (10, 0), stroke=svgwrite.rgb(10, 10, 16, '%')))
dwg.add(dwg.text('Test', insert=(0, 0.2)))
dwg.save()
```

5.1 SVG Version

You can only create two types of SVG drawings:

- *SVG 1.2 Tiny Profile*, use Drawing(profile= `'tiny'`)
- *SVG 1.1 Full Profile*, use Drawing(profile= `'full'`)

CHAPTER 6

utils module

svgwrite.utils.**rgb**(*r=0*, *g=0*, *b=0*, *mode='RGB'*)

Convert **r**, **g**, **b** values to a *string*.

Parameters

- **r** – red part
- **g** – green part
- **b** – blue part
- **mode** (*string*) – `'RGB | %'`

Return type string

mode	Description
`'RGB'`	returns a rgb-string format: `'rgb(r, g, b)'`
`'%'`	returns percent-values as rgb-string format: `'rgb(r%, g%, b%)'`

svgwrite.utils.**iterflatlist**(*values*)

Flatten nested *values*, returns an *iterator*.

svgwrite.utils.**strlist**(*values*, *seperator=', '*)

Concatenate **values** with **sepertator**, *None* values will be excluded.

Parameters **values** – *iterable* object

Returns *string*

svgwrite.utils.**get_unit**(*coordinate*)

Get the *unit* identifier of **coordinate**, if **coordinate** has a valid *unit* identifier appended, else returns *None*.

svgwrite.utils.**split_coordinate**(*coordinate*)

Split coordinate into <*number*> and 'unit' identifier.

Returns <2-tuple> (number, unit-identifier) or (number, None) if no unit-identifier is present or coordinate is an int or float.

svgwrite.utils.**split_angle**(*angle*)
Split angle into <*number*> and <*angle*> identifier.

Returns <2-tuple> (number, angle-identifier) or (number, None) if no angle-identifier is present or angle is an int or float.

svgwrite.utils.**rect_top_left_corner**(*insert*, *size*, *pos='top-left'*)
Calculate top-left corner of a rectangle.

insert and **size** must have the same units.

Parameters

- **insert** (*2-tuple*) – insert point
- **size** (*2-tuple*) – (width, height)
- **pos** (*string*) – insert position `'vert-horiz'`

Returns `'top-left'` corner of the rect

Return type 2-tuple

pos	valid values
vert	`'top | middle | bottom'`
horiz	`'left'|'center'|'right'`

svgwrite.utils.**pretty_xml**(*xml_string*, *indent=2*)
Create human readable XML string.

Parameters **xml_string** – input xml string without line breaks and indentation

Indent int how much to indent, by default 2 spaces

Returns xml_string with linebreaks and indentation

CHAPTER 7

BaseElement

class svgwrite.base.**BaseElement**(***extra*)

The **BaseElement** is the root for all SVG elements. The SVG attributes are stored in **attribs**, and the SVG subelements are stored in **elements**.

BaseElement.**__init__**(***extra*)

Parameters **extra** – extra SVG attributes (keyword arguments)

- add trailing '_' to reserved keywords: `'class_'`, `'from_'`
- replace inner '-' by '_': `'stroke_width'`

SVG attribute names will be checked, if **debug** is *True*.

workaround for removed **attribs** parameter in Version 0.2.2:

```
# replace
element = BaseElement(attribs=adict)

#by
element = BaseElement()
element.update(adict)
```

7.1 Attributes

BaseElement.**attribs**

dict of SVG attributes

BaseElement.**elements**

list of SVG subelements

7.2 Methods

BaseElement.**add**(*element*)
Add an SVG element as subelement.

Parameters **element** – append this SVG element

Returns the added element

BaseElement.**tostring**()
Get the XML representation as unicode *string*.

Returns unicode XML string of this object and all its subelements

BaseElement.**get_xml**()
Get the XML representation as *ElementTree* object.

Returns XML *ElementTree* of this object and all its subelements

BaseElement.**get_id**()
Get the object *id* string, if the object does not have an *id*, a new *id* will be created.

Returns *string*

BaseElement.**get_iri**()
Get the *IRI* reference string of the object. (i.e., `'#id'`).

Returns *string*

BaseElement.**get_funciri**()
Get the *FuncIRI* reference string of the object. (i.e. `'url(#id)'`).

Returns *string*

BaseElement.**update**(*attribs*)
Update SVG Attributes from *dict* attribs.

Rules for keys:

1. trailing '_' will be removed (`'class_'` -> `'class'`)
2. inner '_' will be replaced by '-' (`'stroke_width'` -> `'stroke-width'`)

BaseElement.**__getitem__**(*key*)
Get SVG attribute by *key*.

Parameters **key** (*string*) – SVG attribute name

Returns SVG attribute value

BaseElement.**__setitem__**(*key*, *value*)
Set SVG attribute by *key* to *value*.

Parameters

- **key** (*string*) – SVG attribute name
- **value** (*object*) – SVG attribute value

BaseElement.**set_desc**(*title=None*, *desc=None*)
Insert a **title** and/or a **desc** element as first subelement.

BaseElement.**set_metadata**(*xmldata*)

Parameters **xmldata** – an xml.etree.ElementTree - Element() object.

set/get SVG attributes:

```
element['attribute'] = value
value = element['attribute']

attribs = {
  'class' = 'css-class',
  'stroke' = 'black',
}
element.update(attribs)
```

7.3 Common SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Document Event Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*
- *XLink Attributes*

CHAPTER 8

Drawing

The *Drawing* object is the overall container for all SVG elements. It provides the methods to store the drawing into a file or a file-like object. If you want to use stylesheets, the reference links to this stylesheets were also stored (*add_stylesheet*) in the *Drawing* object.

set/get SVG attributes:

```
element['attribute'] = value
value = element['attribute']
```

The Drawing object also includes a defs section, add elements to the defs section by:

```
drawing.defs.add(element)
```

class svgwrite.drawing.**Drawing**(*filename='noname.svg'*, *size=('100%'*, *'100%')*, ***extra*)

This is the SVG drawing represented by the top level *svg* element.

A drawing consists of any number of SVG elements contained within the drawing element, stored in the *elements* attribute.

A drawing can range from an empty drawing (i.e., no content inside of the drawing), to a very simple drawing containing a single SVG element such as a *rect*, to a complex, deeply nested collection of container elements and graphics elements.

Drawing.**__init__**(*filename='noname.svg'*, *size=('100%'*, *'100%')*, ***extra*)

Parameters

- **filename** (*string*) – filesystem filename valid for `open()`
- **size** (*2-tuple*) – width, height
- **extra** (*keywords*) – additional svg-attributes for the *SVG* object

Important (and not SVG Attributes) **extra** parameters:

Parameters

- **profile** (*string*) – `'tiny | full'` - define the SVG baseProfile

- **debug** (*bool*) – switch validation on/off

8.1 Attributes

`Drawing.`**`filename`**
string should be valid for `open()`.

`Drawing.`**`defs`**
SVG defs section - as *Defs* object.

8.2 Methods

`Drawing.`**`add`**(*element*)
Add an SVG element as subelement.

Parameters **element** – append this SVG element

Returns the added element

`Drawing.`**`write`**(*fileobj*, *pretty=False*, *indent=2*)
Write XML string to *fileobj*.

Parameters

- **fileobj** – a file-like object
- **pretty** – True for easy readable output
- **indent** – how much to indent if pretty is enabled, by default 2 spaces

Python 3.x - set encoding at the open command:

```
open('filename', 'w', encoding='utf-8')
```

`Drawing.`**`save`**(*pretty=False*, *indent=2*)
Write the XML string to *self.filename*.

Parameters

- **pretty** – True for easy readable output
- **indent** – how much to indent if pretty is enabled, by default 2 spaces

`Drawing.`**`saveas`**(*filename*, *pretty=False*, *indent=2*)
Write the XML string to *filename*.

Parameters

- **filename** (*string*) – filesystem filename valid for `open()`
- **pretty** – True for easy readable output
- **indent** – how much to indent if pretty is enabled, by default 2 spaces

`Drawing.`**`add_stylesheet`**(*href*, *title*, *alternate='no'*, *media='screen'*)
Add a stylesheet reference.

Parameters

- **href** (*string*) – link to stylesheet <URI>

- **title** (*string*) – name of stylesheet
- **alternate** (*string*) – 'yes'|'no'
- **media** (*string*) – 'all | aureal | braille | embossed | handheld | print | projection | screen | tty | tv'

Drawing.**get_xml**()
Get the XML representation as *ElementTree* object.

Returns XML *ElementTree* of this object and all its subelements

Drawing.**tostring**()
Get the XML representation as unicode *string*. If you embed the SVG object into a XHTML page, you have to link to the CSS files (if you use CSS classes) in the header section of the surrounding XHTML page.

Returns unicode XML string of this object and all its subelements

8.3 Factory Methods

Drawing.**line**(*start=(0, 0)*, *end=(0, 0)*, ***extra*)
Create a *svgwrite.shapes.Line* object.

Drawing.**rect**(*insert=(0, 0)*, *size=(1, 1)*, *rx=None*, *ry=None*, ***extra*)
Create a *svgwrite.shapes.Rect* object.

Drawing.**circle**(*center=(0, 0)*, *r=1*, ***extra*)
Create a *svgwrite.shapes.Circle* object.

Drawing.**ellipse**(*center=(0, 0)*, *r=(1, 1)*, ***extra*)
Create a *svgwrite.shapes.Ellipse* object.

Drawing.**polyline**(*points=[]*, ***extra*)
Create a *svgwrite.shapes.Polyline* object.

Drawing.**polygon**(*points=[]*, ***extra*)
Create a *svgwrite.shapes.Polygon* object.

Drawing.**text**(*text*, *insert=None*, *x=[]*, *y=[]*, *dx=[]*, *dy=[]*, *rotate=[]*, ***extra*)
Create a *svgwrite.text.Text* object.

Drawing.**tspan**(*text*, *insert=None*, *x=[]*, *y=[]*, *dx=[]*, *dy=[]*, *rotate=[]*, ***extra*)
Create a *svgwrite.text.TSpan* object.

Drawing.**tref**(*element*, ***extra*)
Create a *svgwrite.text.TRef* object.

Drawing.**textPath**(*path*, *text*, *startOffset=None*, *method='align'*, *spacing='exact'*, ***extra*)
Create a *svgwrite.text.TextPath* object.

Drawing.**textArea**(*text=None*, *insert=None*, *size=None*, ***extra*)
Create a *svgwrite.text.TextArea* object.

Drawing.**path**(*d=None*, ***extra*)
Create a *svgwrite.path.Path* object.

Drawing.**image**(*href*, *insert=None*, *size=None*, ***extra*)
Create a *svgwrite.image.Image* object.

Drawing.**g**(***extra*)
Create a *svgwrite.container.Group* object.

Drawing.**symbol**(***extra*)
Create a *svgwrite.container.Symbol* object.

Drawing.**svg**(*insert=None*, *size=None*, ***extra*)
Create a *svgwrite.container.SVG* object.

Drawing.**use**(*href*, *insert=None*, *size=None*, ***extra*)
Create a *svgwrite.container.Use* object.

Drawing.**a**(*href*, *target='_blank'*, ***extra*)
Create a *svgwrite.container.Hyperlink* object.

Drawing.**marker**(*insert=None*, *size=None*, *orient=None*, ***extra*)
Create a *svgwrite.container.Marker* object.

Drawing.**script**(*href=None*, *content=''*, ***extra*)
Create a *svgwrite.container.Script* object.

Drawing.**style**(*content=''*, ***extra*)
Create a *svgwrite.container.Style* object.

Drawing.**linearGradient**(*start=None*, *end=None*, *inherit=None*, ***extra*)
Create a *svgwrite.gradients.LinearGradient* object.

Drawing.**radialGradient**(*center=None*, *r=None*, *focal=None*, *inherit=None*, ***extra*)
Create a *svgwrite.gradients.RadialGradient* object.

Drawing.**mask**(*start=None*, *size=None*, ***extra*)
Create a *svgwrite.masking.Mask* object.

Drawing.**clipPath**(***extra*)
Create a *svgwrite.masking.ClipPath* object.

Drawing.**set**(*element=None*, ***extra*)
Create a *svgwrite.animate.Set* object.

Drawing.**animate**(*element=None*, ***extra*)
Create a *svgwrite.animate.Animate* object.

Drawing.**animateColor**(*element=None*, ***extra*)
Create a *svgwrite.animate.AnimateColor* object.

Drawing.**animateMotion**(*element=None*, ***extra*)
Create a *svgwrite.animate.AnimateMotion* object.

Drawing.**animateTransform**(*transform*, *element=None*, ***extra*)
Create a *svgwrite.animate.AnimateTransform* object.

Drawing.**filter**(*start=None*, *size=None*, *resolution=None*, *inherit=None*, ***extra*)
Create a *svgwrite.filters.Filter* object. (Filter Primitives are created by **factory-methods** of the class **Filter**)

8.4 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.container.Symbol*
- *svgwrite.container.SVG*
- *svgwrite.mixins.Transform*

- *svgwrite.mixins.ViewBox*
- *svgwrite.mixins.Presentation*
- *svgwrite.mixins.Clipping*
- svgwrite.elementfactory.ElementFactory

CHAPTER 9

SVG

class svgwrite.container.**SVG**(*insert=None*, *size=None*, ***extra*)

A SVG document fragment consists of any number of SVG elements contained within an **svg** element.

An SVG document fragment can range from an empty fragment (i.e., no content inside of the **svg** element), to a very simple SVG document fragment containing a single SVG graphics element such as a **rect**, to a complex, deeply nested collection of container elements and graphics elements.

See also:

http://www.w3.org/TR/SVG11/struct.html#SVGElement

SVG.**__init__**(*insert=None*, *size=None*, ***extra*)

Parameters

- **insert** (*2-tuple*) – insert position (**x**, **y**)
- **size** (*2-tuple*) – (**width**, **height**)
- **extra** – additional SVG attributes as keyword-arguments

SVG.**embed_stylesheet**(*content*)

Add <style> tag to the defs section.

Parameters **content** – style sheet content as string

Returns *Style* object

SVG.**embed_font**(*name*, *filename*)

Embed font as base64 encoded data from font file.

Parameters

- **name** – font name
- **filename** – file name of local stored font

SVG.**embed_google_web_font**(*name*, *uri*)

Embed font as base64 encoded data acquired from google fonts.

Parameters

- **name** – font name
- **uri** – google fonts request uri like 'http://fonts.googleapis.com/css?family=Indie+Flower'

9.1 Attributes

SVG.**defs**

Defs container for referenced elements

adding SVG elements to *defs*:

```
svgobject.defs.add(element)
```

9.2 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.container.Symbol*
- *svgwrite.container.SVG*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.ViewBox*
- *svgwrite.mixins.Presentation*

9.3 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element
- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element
- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*
- **transform** – use *svgwrite.mixins.Transform* interface
- **x** – <*coordinate*> – **insert** parameter

  (Has no meaning or effect on *Drawing* .)

  The x-axis coordinate of one corner of the rectangular region into which an embedded **svg** element is placed.

  Default is `'0'`.
- **y** – <*coordinate*> – **insert** parameter

  (Has no meaning or effect on *Drawing* .)

  The y-axis coordinate of one corner of the rectangular region into which an embedded **svg** element is placed.

  Default is `'0'`.

- **width** – <*length*> – **size** parameter

  For outermost **svg** elements (*Drawing*), the intrinsic width of the SVG document fragment. For embedded **svg** elements, the width of the rectangular region into which the **svg** element is placed.

  A negative value is an error. A value of zero disables rendering of the element.

  Default is `'100%'`.

- **height** – <*length*> – **size** parameter

  For outermost **svg** elements (*Drawing*), the intrinsic height of the SVG document fragment. For embedded **svg** elements, the height of the rectangular region into which the **svg** element is placed.

  A negative value is an error. A value of zero disables rendering of the element.

  Default is `'100%'`.

- **viewBox** – *svgwrite.mixins.ViewBox* interface
- **preserveAspectRatio** – *svgwrite.mixins.ViewBox* interface
- **zoomAndPan** – `'disable | magnify'`

  Default is `'magnify'`.

Note: do not set or change following SVG attributes: version, baseProfile, contentScriptType, contentStyleType

9.4 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Document Event Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 10

Group

class svgwrite.container.**Group**(***extra*)

The **Group** (SVG **g**) element is a container element for grouping together related graphics elements.

Grouping constructs, when used in conjunction with the **desc** and **title** elements, provide information about document structure and semantics. Documents that are rich in structure may be rendered graphically, as speech, or as braille, and thus promote accessibility.

A group of elements, as well as individual objects, can be given a name using the **id** attribute. Named groups are needed for several purposes such as animation and re-usable objects.

See also:

http://www.w3.org/TR/SVG11/struct.html#GElement

10.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

10.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **transform** – use `svgwrite.mixins.Transform` interface

10.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 11

Defs

class svgwrite.container.**Defs**(***extra*)

The **defs** element is a container element for referenced elements. For understandability and accessibility reasons, it is recommended that, whenever possible, referenced elements be defined inside of a **defs**.

See also:

http://www.w3.org/TR/SVG11/struct.html#DefsElement

11.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.container.Group*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

11.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **transform** – use *svgwrite.mixins.Transform* interface

11.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 12

Symbol

class svgwrite.container.**Symbol**(***extra*)

The **symbol** element is used to define graphical template objects which can be instantiated by a **use** element. The use of **symbol** elements for graphics that are used multiple times in the same document adds structure and semantics. Documents that are rich in structure may be rendered graphically, as speech, or as braille, and thus promote accessibility.

See also:

http://www.w3.org/TR/SVG11/struct.html#SymbolElement

12.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.ViewBox*
- *svgwrite.mixins.Presentation*
- *svgwrite.mixins.Clipping*

12.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **viewBox** – use *svgwrite.mixins.ViewBox* interface
- **preserveAspectRatio** – use *svgwrite.mixins.ViewBox* interface

12.3 Standard SVG Attributes

- *Core Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 13

Marker

class svgwrite.container.**Marker**(*insert=None*, *size=None*, *orient=None*, ***extra*)

The **marker** element defines the graphics that is to be used for drawing arrowheads or polymarkers on a given **path**, **line**, **polyline** or **polygon** element.

Add Marker definitions to a **defs** section, preferred to the **defs** section of the **main drawing**.

See also:

http://www.w3.org/TR/SVG11/painting.html#MarkerElement

example:

```
dwg = svgwrite.Drawing()
# create a new marker object
marker = dwg.marker(insert=(5,5), size=(10,10))

# red point as marker
marker.add(dwg.circle((5, 5), r=5, fill='red'))

# add marker to defs section of the drawing
dwg.defs.add(marker)

# create a new line object
line = dwg.add(dwg.polyline(
    [(10, 10), (50, 20), (70, 50), (100, 30)],
    stroke='black', fill='none'))

# set marker (start, mid and end markers are the same)
line.set_markers(marker)

# or set markers direct as SVG Attributes 'marker-start', 'marker-mid',
# 'marker-end' or 'marker' if all markers are the same.
line['marker'] = marker.get_funciri()

# NEW in v1.1.11
```

(continues on next page)

(continued from previous page)

```
# set individually markers, to just set the end marker set other markers to None or␣
↪False:
line.set_markers((None, False, marker))
```

Marker.**__init__**(*insert=None*, *size=None*, *orient=None*, ***extra*)

Parameters

- **insert** (*2-tuple*) – reference point (**refX**, **refY**)
- **size** (*2-tuple*) – (**markerWidth**, **markerHeight**)
- **orient** – `'auto'` | *angle*
- **extra** – additional SVG attributes as keyword-arguments

13.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.ViewBox*
- *svgwrite.mixins.Presentation*

13.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **viewBox** – use *svgwrite.mixins.ViewBox* interface

- **preserveAspectRatio** – use *svgwrite.mixins.ViewBox* interface

- **markerUnits** – `'strokeWidth|userSpaceOnUse'`

  Defines the coordinate system for attributes **markerWidth**, **markerHeight** and the contents of the **marker**.

  If markerUnits – `'strokeWidth'`, **markerWidth**, **markerHeight** and the contents of the **marker** represent values in a coordinate system which has a single unit equal the size in user units of the current stroke width in place for the graphic object referencing the marker.

  If markerUnits – `'userSpaceOnUse'`, **markerWidth**, **markerHeight** and the contents of the **marker** represent values in the current user coordinate system in place for the graphic object referencing the marker (i.e., the user coordinate system for the element referencing the **marker** element via a **marker**, **marker-start**, **marker-mid** or **marker-end** property).

- **refX** – <*coordinate*> – **insert** parameter

  The x-axis coordinate of the reference point which is to be aligned exactly at the marker position. The coordinate is defined in the coordinate system after application of the **viewBox** and **preserveAspectRatio** attributes. (default = "0")

- **refY** – <*coordinate*> – **insert** parameter

  The y-axis coordinate of the reference point which is to be aligned exactly at the marker position. The coordinate is defined in the coordinate system after application of the **viewBox** and **preserveAspectRatio** attributes. (default = "0")

- **markerWidth** – <*length*> – **size** parameter

  Represents the width of the viewport into which the marker is to be fitted when it is rendered. (default = "3")

- **markerHeight** – <*length*> – **size** parameter

  Represents the height of the viewport into which the marker is to be fitted when it is rendered. A value of zero disables rendering of the element. (default = "3")

- **orient** – `'auto'` | <*angle*> – **orient** parameter

  Indicates how the marker is rotated. (SVG default = "0", but for `__init__()` `'auto'` is the default value)

  See also:

  http://www.w3.org/TR/SVG11/painting.html#OrientAttribute

13.3 Standard SVG Attributes

- *Core Attributes*
- *Presentation Attributes*

CHAPTER 14

Use

class svgwrite.container.**Use**(*href*, *insert=None*, *size=None*, ***extra*)

The **use** element references another element and indicates that the graphical contents of that element is included/drawn at that given point in the document.

Link to objects by href = `'#object-id'` or use the object itself as href-argument, if the given element has no **id** attribute it gets an automatic generated id.

See also:

http://www.w3.org/TR/SVG11/struct.html#UseElement

Use.**__init__**(*href*, *insert=None*, *size=None*, ***extra*)

Parameters

- **href** (*string*) – object link (id-string) or an object with an id-attribute
- **insert** (*2-tuple*) – insert point (**x**, **y**)
- **size** (*2-tuple*) – (**width**, **height**)
- **extra** – additional SVG attributes as keyword-arguments

14.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.XLink*
- *svgwrite.mixins.Presentation*

14.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **x** – <*coordinate*> – **insert** parameter

  The x-axis coordinate of one corner of the rectangular region into which the referenced element is placed.

  Default is `'0'`.

- **y** – <*coordinate*> – **insert** parameter

  The y-axis coordinate of one corner of the rectangular region into which the referenced element is placed.

  Default is `'0'`.

- **width** – <*length*> – **size** parameter

  The width of the rectangular region into which the referenced element is placed. A negative value is an error. A value of zero disables rendering of this element.

  Default is `'100%'`.

- **height** – <*length*> – **size** parameter

  The height of the rectangular region into which the referenced element is placed. A negative value is an error. A value of zero disables rendering of this element.

  Default is `'100%'`.

- **transform** – *svgwrite.mixins.Transform* interface

- **xlink:href** – *string* – **href** parameter

  set on __init__(href)

14.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*
- *XLink Attributes*

CHAPTER 15

Hyperlink

The **Hyperlink** class represents the SVG **a** element.

class svgwrite.container.**Hyperlink**(*href*, *target='_blank'*, ***extra*)

The **a** element indicate links (also known as Hyperlinks or Web links).

The remote resource (the destination for the link) is defined by a *<URI>* specified by the XLink **xlink:href** attribute. The remote resource may be any Web resource (e.g., an image, a video clip, a sound bite, a program, another SVG document, an HTML document, an element within the current document, an element within a different document, etc.). By activating these links (by clicking with the mouse, through keyboard input, voice commands, etc.), users may visit these resources.

A **Hyperlink** is defined for each separate rendered element contained within the **Hyperlink** class; add sublements as usual with the *add* method.

See also:

http://www.w3.org/TR/SVG11/linking.html#AElement

Hyperlink.**__init__**(*href*, *target='_blank'*, ***extra*)

Parameters

- **href** (*string*) – hyperlink to the target resource
- **target** (*string*) – '_blank|_replace|_self|_parent|_top|<XML-name>'
- **extra** – additional SVG attributes as keyword-arguments

15.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

15.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **transform** – use *svgwrite.mixins.Transform* interface

- **xlink:href** – *string* – **href** parameter

- **xlink:show** – `'new|replace'`

  use the **target** attribute

- **xlink:acuate** – `'onRequest'`

  This attribute provides documentation to XLink-aware processors that an application should traverse from the starting resource to the ending resource only on a post-loading event triggered for the purpose of traversal.

- **target** – *string* – **target** parameter

  This attribute specifies the name or portion of the target window, frame, pane, tab, or other relevant presentation context (e.g., an HTML or XHTML frame, iframe, or object element) into which a document is to be opened when the link is activated.

  - `_replace`: The current SVG image is replaced by the linked content in the same rectangular area in the same frame as the current SVG image.
  - **`_self`: The current SVG image is replaced by the linked content** in the same frame as the current SVG image. This is the lacuna value, if the target attribute is not specified.
  - `_parent`: The immediate frameset parent of the SVG image is replaced by the linked content.
  - `_top`: The content of the full window or tab, including any frames, is replaced by the linked content
  - `_blank`: A new un-named window or tab is requested for the display of the linked content. If this fails, the result is the same as `_top`
  - `<XML-Name>`: Specifies the name of the frame, pane, or other relevant presentation context for display of the linked content. If this already exists, it is re-used, replacing the existing content. If it does not exist, it is created (the same as `_blank`, except that it now has a name).

15.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*
- *XLink Attributes*

CHAPTER 16

Script

The *script* element indicate links to a client-side language.

class svgwrite.container.**Script**(*href=None*, *content=''*, ***extra*)

The **script** element indicate links to a client-side language. This is normally a (also known as Hyperlinks or Web links).

The remote resource (the source of the script) is defined by a <*URI*> specified by the XLink **xlink:href** attribute. The remote resource must be a text-file that contains the script contents. This script can be used within the SVG file by catching events or adding the mouseover/mousedown/ mouseup elements to the markup.

See also:

http://www.w3.org/TR/SVG/script.html

Script.**__init__**(*href=None*, *content=''*, ***extra*)

Parameters

- **href** (*string*) – hyperlink to the target resource or *None* if using *content*
- **content** (*string*) – script content
- **extra** – additional attributes as keyword-arguments

Use *href* **or** *content*, but not both at the same time.

Script.**append**(*content*)

Append content to the existing element-content.

Best place for the *script* element is the *defs* attribute of the *Drawing* class:

```
drawing.defs.add(drawing.script('script-content'))
```

16.1 Parent Classes

- *svgwrite.base.BaseElement*

16.2 SVG Attributes

- **type** – *string*

  Identifies the scripting language for the given *script* element. The value content-type specifies a media type, per MIME. If a *type* is not provided, the value of **contentScriptType** on the **svg** element shall be used, which in turn defaults to `'application/ecmascript'`. If a *script* element falls outside of the outermost svg element and the *type* is not provided, the *type* must default to `'application/ecmascript'`

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **xlink:href** – *string* – **href** parameter

- **xlink:show** – `'new|replace'`

- **xlink:acuate** – `'onRequest'`

  This attribute provides documentation to XLink-aware processors that an application should traverse from the starting resource to the ending resource only on a post-loading event triggered for the purpose of traversal.

16.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*
- *XLink Attributes*

CHAPTER 17

Style

Internal Stylesheets

class svgwrite.container.**Style**(*content=''*, ***extra*)

The *style* element allows style sheets to be embedded directly within SVG content. SVG's *style* element has the same attributes as the corresponding element in HTML.

See also:

http://www.w3.org/TR/SVG/styling.html#StyleElement

Style.**__init__**(*content=''*, ***extra*)

Parameters **content** (*string*) – stylesheet content

Style.**append**(*content*)

Append content to the existing element-content.

Best place for the *style* element is the *defs* attribute of the *Drawing* class:

```
drawing.defs.add(drawing.style('stylesheet-content'))
```

17.1 Parent Classes

- *svgwrite.base.BaseElement*

17.2 SVG Attributes

- **type** – *string*

  default is `'text/css'`

- **title** – *string*

  (For compatibility with HTML 4.) This attribute specifies an advisory title for the 'style' element.

- **media** – *string*

  This attribute specifies the intended destination medium for style information. It may be a single media descriptor or a comma-separated list. The default value for this attribute is `'all'`.

17.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*
- *XLink Attributes*

CHAPTER 18

Path

class svgwrite.path.**Path**(*d=None*, ***extra*)

The <path> element represent the outline of a shape which can be filled, stroked, used as a clipping path, or any combination of the three.

See also:

http://www.w3.org/TR/SVG11/paths.html#PathElement

Path.**__init__**(*d=None*, ***extra*)

Parameters

- **d** (*iterable*) – *coordinates*, *length* and *commands*
- **attribs** (*dict*) – additional SVG attributes
- **extra** – additional SVG attributes as keyword-arguments

18.1 Attributes

commands

list – the command and coordinate stack

18.2 Methods

Path.**push**(**elements*)

Push commands and coordinates onto the command stack.

Parameters **elements** (*iterable*) – *coordinates*, *length* and *commands*

Path.**push_arc**(*target*, *rotation*, *r*, *large_arc=True*, *angle_dir='+'*, *absolute=False*)

Helper function for the elliptical-arc command.

see SVG-Reference: http://www.w3.org/TR/SVG11/paths.html#PathData

Parameters

- **target** (*2-tuple*) – *coordinate* of the arc end point
- **rotation** (*number*) – x-axis-rotation of the ellipse in degrees
- **r** (*number|2-tuple*) – radii rx, ry when r is a *2-tuple* or rx=ry=r if r is a *number*
- **large_arc** (*bool*) – draw the arc sweep of greater than or equal to 180 degrees (**large-arc-flag**)
- **angle_dir** – '+|-' '+' means the arc will be drawn in a "positive-angle" direction (**sweep-flag**)
- **absolute** (*bool*) – indicates that target *coordinates* are absolute else they are relative to the current point

18.3 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*
- *svgwrite.mixins.Markers*

18.4 Path Commands

See also:

http://www.w3.org/TR/SVG11/paths.html#PathData

Uppercase commands indicates absolute coordinates, lowercase commands indicates relative coordinates

- **horizontal-line 'h', 'H' x+**

  Draws a horizontal line from the current point (cpx, cpy) to (x, cpy).

- **vertical-line 'v', 'V' y+**

  Draws a vertical line from the current point (cpx, cpy) to (cpx, y).

- **line 'l', 'L' (x y)+**

  Draw a line from the current point to the given (x,y) coordinate.

- **moveto 'm', 'M' (x y)+**

  Start a new sub-path at the given (x,y) coordinate. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands. Hence, implicit lineto commands will be relative if the moveto is relative, and absolute if the moveto is absolute. If a relative moveto (m) appears as the first element of the path, then it is treated as a pair of absolute coordinates. In this case, subsequent pairs of coordinates are treated as relative even though the initial moveto is interpreted as an absolute moveto.

- **cubic-bezier-curve 'c', 'C' (x1 y1 x2 y2 x y)+**

  Draws a cubic Bézier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve.

- **smooth-cubic-bezier-curve 's', 'S' (x2 y2 x y)+**

  Draws a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the second control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not an C, c, S or s, assume the first control point is coincident with the current point.) (x2,y2) is the second control point (i.e., the control point at the end of the curve).

- **quadratic-bezier-curve 'q', 'Q' (x1 y1 x y)+**

  Draws a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point.

- **smooth-quadratic-bezier-curve 't', 'T' (x y)+**

  Draws a quadratic Bézier curve from the current point to (x,y). The control point is assumed to be the reflection of the control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not a Q, q, T or t, assume the control point is coincident with the current point.)

- **elliptical-arc 'a', 'A' (rx ry x-axis-rotation large-arc-flag sweep-flag x y)+**

  Draws an elliptical arc from the current point to (x, y). The size and orientation of the ellipse are defined by two radii (rx, ry) and an x-axis-rotation, which indicates how the ellipse as a whole is rotated relative to the current coordinate system. The center (cx, cy) of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. large-arc-flag and sweep-flag contribute to the automatic calculations and help determine how the arc is drawn.

- **'z', 'Z'**

  close current subpath

18.5 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **transform** – use `svgwrite.mixins.Transform` methods

- **pathLength** – <*number*>

  the *pathLength* attribute can be used to provide the author's computation of the total length of the path so that the user agent can scale distance-along-a-path computations by the ratio of 'pathLength' to the user agent's own computed value for total path length. A "moveto" operation within a 'path' element is defined to have zero length.

- **d** – *string*

  The definition of the outline of a shape, use push-method to add commands and coordinates

18.6 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*

- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 19

Line

class svgwrite.shapes.**Line**(*start=(0, 0)*, *end=(0, 0)*, ***extra*)

The **line** element defines a line segment that starts at one point and ends at another.

See also:

http://www.w3.org/TR/SVG11/shapes.html#LineElement

Line.**__init__**(*start=(0, 0)*, *end=(0, 0)*, ***extra*)

Parameters

- **start** (*2-tuple*) – start point (**x1**, **y1**)
- **end** (*2-tuple*) – end point (**x2**, **y2**)
- **extra** – additional SVG attributes as keyword-arguments

19.1 SVG Attributes

- **x1** – <*coordinate*> – **start** parameter
- **y1** – <*coordinate*> – **start** parameter
- **x2** – <*coordinate*> – **end** parameter
- **y2** – <*coordinate*> – **end** parameter

19.2 Common SVG Attributes

These are the common SVG Attributes for Line, Rect, Circle, Ellipse, Poliyline and Polygon.

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **transform** – use *svgwrite.mixins.Transform* interface

19.3 Common Standard SVG Attributes

These are the common Standard SVG Attributes for Line, Rect, Circle, Ellipse, Poliyline and Polygon.

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

19.4 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*
- *svgwrite.mixins.Markers*

CHAPTER 20

Rect

class svgwrite.shapes.**Rect**(*insert=(0, 0)*, *size=(1, 1)*, *rx=None*, *ry=None*, ***extra*)

The **rect** element defines a rectangle which is axis-aligned with the current user coordinate system. Rounded rectangles can be achieved by setting appropriate values for attributes **rx** and **ry**.

See also:

http://www.w3.org/TR/SVG11/shapes.html#RectElement

Rect.**__init__**(*insert=(0, 0)*, *size=(1, 1)*, *rx=None*, *ry=None*, ***extra*)

Parameters

- **insert** (*2-tuple*) – insert point (**x**, **y**), left-upper point
- **size** (*2-tuple*) – (**width**, **height**)
- **rx** (*<length>*) – corner x-radius
- **ry** (*<length>*) – corner y-radius
- **extra** – additional SVG attributes as keyword-arguments

20.1 SVG Attributes

- **x** – <*coordinate*> – **insert** parameter

  The x-axis coordinate of the side of the rectangle which has the smaller x-axis coordinate value

- **y** – <*coordinate*> – **insert** parameter

  The y-axis coordinate of the side of the rectangle which has the smaller y-axis coordinate value

- **width** – <*length*> – **size** parameter
- **height** – <*length*> – **size** parameter

- **rx** – <*length*> – **rx** parameter

  For rounded rectangles, the y-axis radius of the ellipse used to round off the corners of the rectangle.

- **ry** – <*length*> – **ry** parameter

  For rounded rectangles, the y-axis radius of the ellipse used to round off the corners of the rectangle.

20.2 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

CHAPTER 21

Circle

class svgwrite.shapes.**Circle**(*center=(0, 0)*, *r=1*, ***extra*)

The **circle** element defines a circle based on a center point and a radius.

See also:

http://www.w3.org/TR/SVG11/shapes.html#CircleElement

Circle.**__init__**(*center=(0, 0)*, *r=1*, ***extra*)

Parameters

- **center** (*2-tuple*) – circle center point (**cx**, **cy**)
- **r** (*length*) – circle-radius **r**
- **extra** – additional SVG attributes as keyword-arguments

21.1 SVG Attributes

- **cx** – <*coordinate*> – **center** parameter

  The x-axis coordinate of the center of the circle.

- **cy** – <*coordinate*> – **center** parameter

  The y-axis coordinate of the center of the circle.

- **r** – <*length*> – **r** parameter

  The radius of the circle.

21.2 Parent Classes

- *svgwrite.base.BaseElement*

- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

CHAPTER 22

Ellipse

class svgwrite.shapes.**Ellipse**(*center=(0, 0)*, *r=(1, 1)*, ***extra*)

The **ellipse** element defines an ellipse which is axis-aligned with the current user coordinate system based on a center point and two radii.

See also:

http://www.w3.org/TR/SVG11/shapes.html#EllipseElement

Ellipse.**__init__**(*center=(0, 0)*, *r=(1, 1)*, ***extra*)

Parameters

- **center** (*2-tuple*) – ellipse center point (**cx**, **cy**)
- **r** (*2-tuple*) – ellipse radii (**rx**, **ry**)
- **extra** – additional SVG attributes as keyword-arguments

22.1 SVG Attributes

- **cx** – <*coordinate*> – **center** parameter

  The x-axis coordinate of the center of the ellipse.

- **cy** – <*coordinate*> – **center** parameter

  The y-axis coordinate of the center of the ellipse.

- **rx** – <*length*> – **r** parameter

  The x-axis radius of the ellipse.

- **ry** – <*length*> – **r** parameter

  The y-axis radius of the ellipse.

22.2 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

CHAPTER 23

Polyline

class svgwrite.shapes.**Polyline**(*points=[]*, ***extra*)

The **polyline** element defines a set of connected straight line segments. Typically, **polyline** elements define open shapes.

See also:

http://www.w3.org/TR/SVG11/shapes.html#PolylineElement

Polyline.**__init__**(*points=[]*, ***extra*)

Parameters

- **points** (*iterable*) – *iterable* of points (points are *2-tuples*)
- **extra** – additional SVG attributes as keyword-arguments

23.1 Attributes

Polyline.**points**

list of points, a point is a *2-tuple* (x, y): x, y = <*number*>

23.2 SVG Attributes

- **points** – *list* of points – **points** parameter

  The points that make up the polyline. All coordinate values are in the **user coordinate system** (no units allowed).

How to append points:

```
Polyline.points.append( point )
Polyline.points.extend( [point1, point2, point3, ...] )
```

23.3 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*
- *svgwrite.mixins.Markers*

CHAPTER 24

Polygon

class svgwrite.shapes.**Polygon**(*points=[]*, ***extra*)

The **polygon** element defines a closed shape consisting of a set of connected straight line segments.

Same as *Polyline* but closed.

See also:

http://www.w3.org/TR/SVG11/shapes.html#PolygonElement

24.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*
- *svgwrite.mixins.Markers*

CHAPTER 25

Basic Shapes Examples

```
import svgwrite
from svgwrite import cm, mm


def basic_shapes(name):
    dwg = svgwrite.Drawing(filename=name, debug=True)
    hlines = dwg.add(dwg.g(id='hlines', stroke='green'))
    for y in range(20):
        hlines.add(dwg.line(start=(2*cm, (2+y)*cm), end=(18*cm, (2+y)*cm)))
    vlines = dwg.add(dwg.g(id='vline', stroke='blue'))
    for x in range(17):
        vlines.add(dwg.line(start=((2+x)*cm, 2*cm), end=((2+x)*cm, 21*cm)))
    shapes = dwg.add(dwg.g(id='shapes', fill='red'))

    # set presentation attributes at object creation as SVG-Attributes
    circle = dwg.circle(center=(15*cm, 8*cm), r='2.5cm', stroke='blue', stroke_
↪width=3)
    circle['class'] = 'class1 class2'
    shapes.add(circle)

    # override the 'fill' attribute of the parent group 'shapes'
    shapes.add(dwg.rect(insert=(5*cm, 5*cm), size=(45*mm, 45*mm),
                        fill='blue', stroke='red', stroke_width=3))

    # or set presentation attributes by helper functions of the Presentation-Mixin
    ellipse = shapes.add(dwg.ellipse(center=(10*cm, 15*cm), r=('5cm', '10mm')))
    ellipse.fill('green', opacity=0.5).stroke('black', width=5).dasharray([20, 20])
    dwg.save()


if __name__ == '__main__':
    basic_shapes('basic_shapes.svg')
```

25.1 basic_shapes.svg

CHAPTER 26

Image

class svgwrite.image.**Image**(*href*, *insert=None*, *size=None*, ***extra*)

The **image** element indicates that the contents of a complete file are to be rendered into a given rectangle within the current user coordinate system. The **image** element can refer to raster image files such as PNG or JPEG or to files with MIME type of "image/svg+xml".

26.1 Methods

Image.**__init__**(*href*, *insert=None*, *size=None*, ***extra*)

Parameters

- **href** (*string*) – hyperlink to the image resource
- **insert** (*2-tuple*) – insert point (**x**, **y**)
- **size** (*2-tuple*) – (**width**, **height**)
- **attribs** (*dict*) – additional SVG attributes
- **extra** – additional SVG attributes as keyword-arguments

Image.**stretch**()

Stretch viewBox in x and y direction to fill viewport, does not preserve aspect ratio.

Image.**fit**(*horiz='center'*, *vert='middle'*, *scale='meet'*)

Set the preserveAspectRatio attribute.

Parameters

- **horiz** (*string*) – horizontal alignment `'left'|'center'|'right'`
- **vert** (*string*) – vertical alignment `'top'|'middle'|'bottom'`
- **scale** (*string*) – scale method `'meet'|'slice'`

Scale methods	Description
`meet`	preserve aspect ration and zoom to limits of viewBox
`slice`	preserve aspect ration and viewBox touch viewport on all bounds, viewBox will extend beyond the bounds of the viewport

26.2 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Clipping*

26.3 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **transform** – use *svgwrite.mixins.Transform* interface

- **x** – <*coordinate*> – **insert** parameter

  The x-axis coordinate of one corner of the rectangular region into which the referenced document is placed.

  Default is `'0'`.

- **y** – <*coordinate*> – **insert** parameter

  The y-axis coordinate of one corner of the rectangular region into which the referenced document is placed.

  Default is `'0'`.

- **width** – <*length*> – **size** parameter

  The width of the rectangular region into which the referenced document is placed. A negative value is an error. A value of zero disables rendering of the element.

- **height** – <*length*> – **size** parameter

  The height of the rectangular region into which the referenced document is placed. A negative value is an error. A value of zero disables rendering of the element.

- **xlink:href** – *string* – **href** parameter

  A IRI reference to the image resource.

- **preserveAspectRatio** – `'[defer] <align> [<meetOrSlice>]'`

26.4 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*
- *XLink Attributes*

CHAPTER 27

Text

Text that is to be rendered as part of an SVG document fragment is specified using the **text** element. The characters to be drawn are expressed as XML character data inside the **text** element.

class svgwrite.text.**Text**(*text*, *insert=None*, *x=None*, *y=None*, *dx=None*, *dy=None*, *rotate=None*, ***extra*)

The **Text** element defines a graphics element consisting of text. The characters to be drawn are expressed as XML character data inside the **Text** element.

See also:

http://www.w3.org/TR/SVG11/text.html#TextElement

Refer to **TSpan** SVG Attributes

27.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.text.TSpan*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

CHAPTER 28

TSpan

class svgwrite.text.**TSpan**(*text*, *insert=None*, *x=None*, *y=None*, *dx=None*, *dy=None*, *rotate=None*, ***extra*)

Within a **Text** element, text and font properties and the current text position can be adjusted with absolute or relative coordinate values by using the **TSpan** element. The characters to be drawn are expressed as XML character data inside the **TSpan** element.

See also:

http://www.w3.org/TR/SVG11/text.html#TSpanElement

TSpan.**__init__**(*text*, *insert=None*, *x=None*, *y=None*, *dx=None*, *dy=None*, *rotate=None*, ***extra*)

Parameters

- **text** (*string*) – **tspan** content
- **insert** (*2-tuple*) – The **insert** parameter is the absolute insert point of the text, don't use this parameter in combination with the **x** or the **y** parameter.
- **x** (*list*) – list of absolute x-axis values for characters
- **y** (*list*) – list of absolute y-axis values for characters
- **dx** (*list*) – list of relative x-axis values for characters
- **dy** (*list*) – list of relative y-axis values for characters
- **rotate** (*list*) – list of rotation-values for characters (in degrees)

28.1 Attributes

TSpan.**text**

stores the text value.

28.2 Methods

TSpan.**__init__**(*text*, *insert=None*, *x=None*, *y=None*, *dx=None*, *dy=None*, *rotate=None*, ***extra*)

Parameters

- **text** (*string*) – **tspan** content
- **insert** (*2-tuple*) – The **insert** parameter is the absolute insert point of the text, don't use this parameter in combination with the **x** or the **y** parameter.
- **x** (*list*) – list of absolute x-axis values for characters
- **y** (*list*) – list of absolute y-axis values for characters
- **dx** (*list*) – list of relative x-axis values for characters
- **dy** (*list*) – list of relative y-axis values for characters
- **rotate** (*list*) – list of rotation-values for characters (in degrees)

28.3 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Presentation*

28.4 SVG Attributes

- **x** – <*coordinate*+>

  If a single <*coordinate*> is provided, then the value represents the new absolute X coordinate for the current text position for rendering the glyphs that correspond to the first character within this element or any of its descendants.

  If *list* of n <*coordinates*> is provided, then the values represent new absolute X coordinates for the current text position for rendering the glyphs corresponding to each of the first n characters within this element or any of its descendants.

  If more <*coordinates*> are provided than characters, then the extra <*coordinates*> will have no effect on glyph positioning.

  If more characters exist than <*coordinates*> , then for each of these extra characters:

  (a) if an ancestor **Text** or **TSpan** element specifies an absolute X coordinate for the given character via an **x** attribute, then that absolute X coordinate is used as the starting X coordinate for that character (nearest ancestor has precedence), else

  (b) the starting X coordinate for rendering the glyphs corresponding to the given character is the X coordinate of the resulting current text position from the most recently rendered glyph for the current **Text** element.

  If the attribute is not specified:

  (a) if an ancestor **Text** or **TSpan** element specifies an absolute X coordinate for a given character via an **x** attribute, then that absolute X coordinate is used (nearest ancestor has precedence), else

(b) the starting X coordinate for rendering the glyphs corresponding to a given character is the X coordinate of the resulting current text position from the most recently rendered glyph for the current **Text** element.

- **y** – <*coordinate+*>

  The corresponding list of absolute Y coordinates for the glyphs corresponding to the characters within this element. The processing rules for the **y** attribute parallel the processing rules for the **x** attribute.

- **dx** – <*length+*>

  If a single <length> is provided, this value represents the new relative X coordinate for the current text position for rendering the glyphs corresponding to the first character within this element or any of its descendants. The current text position is shifted along the x-axis of the current user coordinate system by <*length*> before the first character's glyphs are rendered.

  If a *list* of n <*length*> is provided, then the values represent incremental shifts along the x-axis for the current text position before rendering the glyphs corresponding to the first n characters within this element or any of its descendants. Thus, before the glyphs are rendered corresponding to each character, the current text position resulting from drawing the glyphs for the previous character within the current **Text** element is shifted along the X axis of the current user coordinate system by <*length*> .

  If more <*lengths*> are provided than characters, then any extra <*lengths*> will have no effect on glyph positioning.

  If more characters exist than <length>s, then for each of these extra characters:

  (a) if an ancestor **Text** or **TSpan** element specifies a relative X coordinate for the given character via a **dx** attribute, then the current text position is shifted along the x-axis of the current user coordinate system by that amount (nearest ancestor has precedence), else

  (b) no extra shift along the x-axis occurs.

  If the attribute is not specified:

  (a) if an ancestor **Text** or **TSpan** element specifies a relative X coordinate for a given character via a **dx** attribute, then the current text position is shifted along the x-axis of the current user coordinate system by that amount (nearest ancestor has precedence), else

  (b) no extra shift along the x-axis occurs.

- **dy** – <*length+*>

  The corresponding list of relative Y coordinates for the characters within the **tspan** element. The processing rules for the **dy** attribute parallel the processing rules for the **dx** attribute.

- **rotate** – <*angle+*>

  The supplemental rotation about the current text position that will be applied to all of the glyphs corresponding to each character within this element.

  If a *list* of <*numbers*> is provided, then the first <*number*> represents the supplemental rotation for the glyphs corresponding to the first character within this element or any of its descendants, the second <number> represents the supplemental rotation for the glyphs that correspond to the second character, and so on.

  If more <*numbers*> are provided than there are characters, then the extra <*numbers*> will be ignored.

  If more characters are provided than <*numbers*>, then for each of these extra characters the rotation value specified by the last number must be used.

  If the attribute is not specified and if an ancestor **Text** or **TSpan** element specifies a supplemental rotation for a given character via a **rotate** attribute, then the given supplemental rotation is applied to

the given character (nearest ancestor has precedence). If there are more characters than <*numbers*> specified in the ancestor's **rotate** attribute, then for each of these extra characters the rotation value specified by the last number must be used.

This supplemental rotation has no impact on the rules by which current text position is modified as glyphs get rendered and is supplemental to any rotation due to text on a path and to **glyph-orientation-horizontal** or **glyph-orientation-vertical**.

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **textLength** – <*length*>

  The purpose of this attribute is to allow the author to achieve exact alignment, in visual rendering order after any bidirectional reordering, for the first and last rendered glyphs that correspond to this element; thus, for the last rendered character (in visual rendering order after any bidirectional reordering), any supplemental inter-character spacing beyond normal glyph advances are ignored (in most cases) when the user agent determines the appropriate amount to expand/compress the text string to fit within a length of **textLength**.

- **lengthAdjust** – `'spacing | spacingAndGlyphs'`

  Indicates the type of adjustments which the user agent shall make to make the rendered length of the text match the value specified on the **textLength** attribute.

  - `'spacing'` indicates that only the advance values are adjusted. The glyphs themselves are not stretched or compressed.
  - `'spacingAndGlyphs'` indicates that the advance values are adjusted and the glyphs themselves stretched or compressed in one axis (i.e., a direction parallel to the inline-progression-direction).

  If the attribute is not specified, the effect is as a value of *spacing* were specified.

28.5 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 29

TRef

class svgwrite.text.**TRef**(*element*, ***extra*)

The textual content for a **Text** can be either character data directly embedded within the <text> element or the character data content of a referenced element, where the referencing is specified with a **TRef** element.

See also:

http://www.w3.org/TR/SVG11/text.html#TRefElement

TRef.**__init__**(*element*, ***extra*)

Parameters **element** – create a reference this element, if element is a *string* its the **id** name of the referenced element, if element is a *BaseElement* the **id** SVG Attribute is used to create the reference.

TRef.**set_href**(*element*)

Create a reference to **element**.

Parameters **element** – if element is a *string* its the **id** name of the referenced element, if element is a **BaseElement** class the **id** SVG Attribute is used to create the reference.

29.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*
- *svgwrite.mixins.Presentation*

29.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **xlink:href** – <*string*> A IRI reference to an element whose character data content shall be used as character data for this **tref** element.

29.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 30

TextPath

class svgwrite.text.**TextPath**(*path*, *text*, *startOffset=None*, *method='align'*, *spacing='exact'*, ***extra*)

In addition to text drawn in a straight line, SVG also includes the ability to place text along the shape of a **path** element. To specify that a block of text is to be rendered along the shape of a **path**, include the given text within a **textPath** element which includes an **xlink:href** attribute with a IRI reference to a **path** element.

See also:

http://www.w3.org/TR/SVG11/text.html#TextPathElement

30.1 Methods

TextPath.**__init__**(*path*, *text*, *startOffset=None*, *method='align'*, *spacing='exact'*, ***extra*)

Parameters

- **path** – link to **path**, **id** string or **Path** object
- **text** (*string*) – **textPath** content
- **startOffset** (*number*) – text starts with offset from begin of path.
- **method** (*string*) – `align|stretch`
- **spacing** (*string*) – `exact|auto`

30.2 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*
- *svgwrite.mixins.Presentation*

30.3 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **xlink:href** – *string*

  A IRI reference to an element whose character data content shall be used as character data for this **TRef** element.

- **startOffset** – <*length*>

  An offset from the start of the **path** for the initial current text position, calculated using the user agent's distance along the path algorithm. Value as percentage or distance along the path measured in the current user coordinate system.

- **method** – `'align | stretch'`

  Indicates the method by which text should be rendered along the path.

- **spacing** – `'auto | exact'`

  Indicates how the user agent should determine the spacing between glyphs that are to be rendered along a path.

30.4 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Graphical Event Attributes*
- *Presentation Attributes*

CHAPTER 31

TextArea

class svgwrite.text.**TextArea**(*text=None*, *insert=None*, *size=None*, ***extra*)

At this time **textArea** is only available for SVG 1.2 Tiny profile.

The **textArea** element allows simplistic wrapping of text content within a given region. The *tiny* profile of SVG specifies a single rectangular region. Other profiles may allow a sequence of arbitrary shapes.

Text wrapping via the **textArea** element is available as a lightweight and convenient facility for simple text wrapping where a complete box model layout engine is not required.

The layout of wrapped text is user agent dependent; thus, content developers need to be aware that there might be different results, particularly with regard to where line breaks occur.

The TextArea class wraps every text added by write() or writeline() as **tspan** element.

See also:

http://www.w3.org/TR/SVGMobile12/text.html#TextAreaElement

31.1 Methods

TextArea.**write**(*text*, ***extra*)

Add text as **tspan** elements, with extra-params for the **tspan** element.

Use the '\n' character for line breaks.

31.2 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.Transform*
- *svgwrite.mixins.Presentation*

31.3 SVG Attributes

- **x** – <*coordinate*>

  The x-axis coordinate of one corner of the rectangular region into which the text content will be placed. The lacuna value is `'0'`.

- **y** – <*coordinate*>

  The y-axis coordinate of one corner of the rectangular region into which the text content will be placed. The lacuna value is `'0'`.

- **width** – `'auto'` | <*coordinate*>

  The width of the rectangular region into which the text content will be placed. A value of `'auto'` indicates that the width of the rectangular region is infinite. The lacuna value is `'auto'`.

- **height** – `'auto'` | <*coordinate*>

  The height of the rectangular region into which the text content will be placed. A value of `'auto'` indicates that the height of the rectangular region is infinite. The lacuna value is `'auto'`.

- **editable** – `'auto`| simple'`

  This attribute indicates whether the text can be edited. See the definition of the 'editable' attribute.

- **focusable** – `'true | false | auto'`

Value	Description
`'true'`	The element is keyboard-aware and must be treated as any other UI component that can get focus.
`'false'`	The element is not focusable.
`'auto'`	The lacuna value. Equivalent to `'false'`

  Exception: see http://www.w3.org/TR/SVGMobile12/interact.html#focusable-attr

- **line-increment** – `'auto | inherit'` | <*number*>

  The **line-increment** property provides limited control over the size of each line in the block-progression-direction. This property applies to the **textArea** element, and to child elements of the **textArea** element. The **line-increment** property must not have any effect when used on an element which is not, or does not have as an ancestor, a **textArea** element.

- **text-align** – `'start | end | center | inherit'`

  Alignment in the inline progression direction in flowing text is provided by the text-align property.

- **display-align** – `'auto | before | center | after | inherit'`

  The **display-align** property specifies the alignment, in the block-progression-direction, of the text content of the **textArea** element.

Value	Description
`'auto'`	For SVG, auto is equivalent to before.
`'before'`	The before-edge of the first line is aligned with the before-edge of the first region.
`'center'`	The lines are centered in the block-progression-direction.
`'after'`	The after-edge of the last line is aligned with the after-edge of the last region.

  Layout rules: see http://www.w3.org/TR/SVGMobile12/text.html#TextAreaElement

CHAPTER 32

LinearGradient

Gradients consist of continuously smooth color transitions along a vector from one color to another, possibly followed by additional transitions along the same vector to other colors. SVG provides for two types of gradients: linear gradients and radial gradients.

See also:

- http://www.w3.org/TR/SVG11/pservers.html#Gradients
- http://www.w3.org/TR/SVG11/pservers.html#LinearGradients

class svgwrite.gradients.**LinearGradient**(*start=None*, *end=None*, *inherit=None*, ***extra*)
Linear gradients are defined by a SVG <linearGradient> element.

32.1 Methods

LinearGradient.**__init__**(*start=None*, *end=None*, *inherit=None*, ***extra*)

Parameters

- **start** (*2-tuple*) – start point of the gradient (**x1**, **y1**)
- **end** (*2-tuple*) – end point of the gradient (**x2**, **y2**)
- **inherit** – gradient inherits properties from *inherit* see: **xlink:href**

LinearGradient.**add_stop_color**(*offset=None*, *color=None*, *opacity=None*)
Adds a stop-color to the gradient.

Parameters

- **offset** – is either a <number> (usually ranging from 0 to 1) or a <*percentage*> (usually ranging from 0% to 100%) which indicates where the gradient stop is placed. Represents a location along the gradient vector. For radial gradients, it represents a percentage distance from (fx,fy) to the edge of the outermost/largest circle.
- **color** – indicates what color to use at that gradient stop

- **opacity** – defines the opacity of a given gradient stop

`LinearGradient.`**`get_paint_server`**`(`*default='none'*`)`
Returns the <FuncIRI> of the gradient.

32.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **gradientUnits** – `'userSpaceOnUse | objectBoundingBox'`

  Defines the coordinate system for attributes **x1**, **y1**, **x2** and **y2**.

  See also:

  http://www.w3.org/TR/SVG11/pservers.html#LinearGradientElementGradientUnitsAttribute

- **gradientTransform** – <*transform-list*>

  Use the `-svgwrite.mixins.Transform` interface to set transformations.

  Contains the definition of an optional additional transformation from the gradient coordinate system onto the target coordinate system (i.e., userSpaceOnUse or objectBoundingBox). This allows for things such as skewing the gradient. This additional transformation matrix is post-multiplied to (i.e., inserted to the right of) any previously defined transformations, including the implicit transformation necessary to convert from object bounding box units to user space.

- **x1** – <*coordinate*> – **start** parameter

  x1, **y1**, **x2** and **y2** define a gradient vector for the linear gradient. This gradient vector provides starting and ending points onto which the gradient stops are mapped. The values of **x1**, **y1**, **x2** and **y2** can be either numbers or percentages.

  default is `'0%'`

- **y1** – <*coordinate*> – **start** parameter

  See **x1**. Default is `'0%'`

- **x2** – <*coordinate*> – **end** parameter

  See **x1**. Default is `'100%'`

- **y2** – <*coordinate*> – **end** parameter

  See **x1**. Default is `'0%'`

- **spreadMethod** – `'pad | reflect | repeat'`

  Indicates what happens if the gradient starts or ends inside the bounds of the target rectangle. Possible values are: `'pad'`, which says to use the terminal colors of the gradient to fill the remainder of the target region, `'reflect'`, which says to reflect the gradient pattern start-to-end, end-to-start, start-to-end, etc. continuously until the target rectangle is filled, and `'repeat'`, which says to repeat the gradient pattern start-to-end, start-to-end, start-to-end, etc. continuously until the target region is filled.

default is `'pad'`

- **xlink:href** – <*iri*> – **inherit** parameter

  A URI reference to a different *LinearGradient* or *RadialGradient* element within the current SVG document fragment. Any *LinearGradient* attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has no defined gradient stops, and the referenced element does (possibly due to its own **xlink:href** attribute), then this element inherits the gradient stop from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attribute or gradient stops due to its own **xlink:href** attribute, then the current element can inherit those attributes or gradient stops.

CHAPTER 33

RadialGradient

See also:

http://www.w3.org/TR/SVG11/pservers.html#RadialGradients

class svgwrite.gradients.**RadialGradient**(*center=None*, *r=None*, *focal=None*, *inherit=None*, ***extra*)

Radial gradients are defined by a SVG <radialGradient> element.

33.1 Methods

RadialGradient.**__init__**(*center=None*, *r=None*, *focal=None*, *inherit=None*, ***extra*)

Parameters

- **center** (*2-tuple*) – center point for the gradient (**cx**, **cy**)
- **r** – radius for the gradient
- **focal** (*2-tuple*) – focal point for the radial gradient (**fx**, **fy**)
- **inherit** – gradient inherits properties from *inherit* see: **xlink:href**

RadialGradient.**add_stop_color**(*offset=None*, *color=None*, *opacity=None*)

Adds a stop-color to the gradient.

Parameters

- **offset** – is either a <number> (usually ranging from 0 to 1) or a <*percentage*> (usually ranging from 0% to 100%) which indicates where the gradient stop is placed. Represents a location along the gradient vector. For radial gradients, it represents a percentage distance from (fx,fy) to the edge of the outermost/largest circle.
- **color** – indicates what color to use at that gradient stop
- **opacity** – defines the opacity of a given gradient stop

RadialGradient.**get_paint_server**(*default='none'*)
Returns the <FuncIRI> of the gradient.

33.2 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **gradientUnits** – `'userSpaceOnUse | objectBoundingBox'`

  Defines the coordinate system for attributes **cx**, **cy**, **r**, **fx** and **fy**.

  See also:

  http://www.w3.org/TR/SVG11/pservers.html#RadialGradientElementGradientUnitsAttribute

- **cx** – <*coordinate*> – **center** parameter

  cx, **cy** and **r** define the largest (i.e., outermost) circle for the radial gradient. The gradient will be drawn such that the 100% gradient stop is mapped to the perimeter of this largest (i.e., outermost) circle.

  default is `'50%'`

- **cy** – <*coordinate*> – **center** parameter

  See **cx**. Default is `'50%'`.

- **r** – <*length*> – **r** parameter

  See **cx**.

  A value of zero will cause the area to be painted as a single color using the color and opacity of the last gradient stop.

  Default is `'50%'`.

- **fx** – <*coordinate*> – **focal** parameter

  fx and **fy** define the focal point for the radial gradient. The gradient will be drawn such that the 0% gradient stop is mapped to (fx, fy). If attribute **fx** is not specified, **fx** will coincide with the presentational value of **cx** for the element whether the value for **cx** was inherited or not. If the element references an element that specifies a value for **fx**, then the value of 'fx' is inherited from the referenced element.

- **fy** – <*coordinate*> – **focal** parameter

  See **fx**. If attribute **fy** is not specified, **fy** will coincide with the presentational vlaue of **cy** for the element whether the value for **cy** was inherited or not. If the element references an element that specifies a value for **fy**, then the value of **fy** is inherited from the referenced element.

- **gradientTransform** – <*transform-list*>

  Use the `-svgwrite.mixins.Transform` interface to set transformations.

  See *`LinearGradient`*

- **spreadMethod** – `'pad | reflect | repeat'`

  See *LinearGradient*

- **xlink:href** – <*iri*> – **inherit** parameter

  See *LinearGradient*

CHAPTER 34

Pattern

class svgwrite.pattern.**Pattern**(*insert=None*, *size=None*, *inherit=None*, ***extra*)

A pattern is used to fill or stroke an object using a pre-defined graphic object which can be replicated ("tiled") at fixed intervals in x and y to cover the areas to be painted. Patterns are defined using a *pattern* element and then referenced by properties *fill* and *stroke* on a given graphics element to indicate that the given element shall be filled or stroked with the referenced pattern.

See also:

http://www.w3.org/TR/SVG11/pservers.html#PatternElement

34.1 Methods

Pattern.**__init__**(*insert=None*, *size=None*, *inherit=None*, ***extra*)

Parameters

- **insert** (*2-tuple*) – base point of the pattern (**x**, **y**)
- **size** (*2-tuple*) – size of the pattern (**width**, **height**)
- **inherit** – pattern inherits properties from *inherit* see: **xlink:href**

Pattern.**add**()

Add **element** to the pattern content.

The contents of the **pattern** are relative to a new coordinate system. If there is a **viewBox** attribute, then the new coordinate system is fitted into the region defined by the **x**, **y**, **width**, **height** and **patternUnits** attributes on the **pattern** element using the standard rules for **viewBox** and **preserveAspectRatio**. If there is no **viewBox** attribute, then the new coordinate system has its origin at (x, y), where x is established by the **x** attribute on the **pattern** element, and y is established by the **y** attribute on the 'pattern' element. Thus, in the following example:

```
<pattern x="10" y="10" width="20" height="20">
  <rect x="5" y="5" width="10" height="10"/>
</pattern>
```

or as *svgwrite* calls:

```
# dwg is the main svg drawing
pattern = dwg.pattern(insert=(10, 10), size=(20, 20))
pattern.add(dwg.rect(insert=(5, 5), size=(10, 10))
```

the rectangle has its top/left located 5 units to the right and 5 units down from the origin of the pattern tile.

34.2 SVG Attributes

- **patternUnits** – `'userSpaceOnUse | objectBoundingBox'`

  Defines the coordinate system for attributes **x**, **y**, **width** and **height**.

  If patternUnits= `'userSpaceOnUse'` , **x** , **y**, **width** and **height** represent values in the coordinate system that results from taking the current user coordinate system in place at the time when the **pattern** element is referenced (i.e., the user coordinate system for the element referencing the **pattern** element via a **fill** or **stroke** property) and then applying the transform specified by attribute **patternTransform**.

  If patternUnits= `'objectBoundingBox'` , the user coordinate system for attributes **x**, **y**, **width** and **height** is established using the bounding box of the element to which the pattern is applied (see Object bounding box units) and then applying the transform specified by attribute **patternTransform**.

  Default is `'objectBoundingBox'`.

- **patternContentUnits** – `'userSpaceOnUse | objectBoundingBox'`

  Defines the coordinate system for the contents of the **pattern**. Note that this attribute has no effect if attribute **viewBox** is specified.

  If patternContentUnits= `'userSpaceOnUse'` , the user coordinate system for the contents of the **pattern** element is the coordinate system that results from taking the current user coordinate system in place at the time when the **pattern** element is referenced (i.e., the user coordinate system for the element referencing the **pattern** element via a **fill** or **stroke** property) and then applying the transform specified by attribute **patternTransform**.

  If patternContentUnits= `'objectBoundingBox'` , the user coordinate system for the contents of the **pattern** element is established using the bounding box of the element to which the pattern is applied (see Object bounding box units) and then applying the transform specified by attribute **patternTransform**.

  Default is `'userSpaceOnUse'`.

- **patternTransform** – <*transform-list*>

  Use the *Transform* interface to set transformations.

  Contains the definition of an optional additional transformation from the pattern coordinate system onto the target coordinate system (i.e., `'userSpaceOnUse'` or `'objectBoundingBox'`). This allows for things such as skewing the pattern tiles. This additional transformation matrix is post-multiplied to (i.e., inserted to the right of) any previously defined transformations, including the implicit transformation necessary to convert from object bounding box units to user space.

- **x** – <*coordinate*> – **insert** parameter

  x, **y**, **width** and **height** indicate how the pattern tiles are placed and spaced. These attributes represent coordinates and values in the coordinate space specified by the combination of attributes **patternUnits** and **patternTransform**.

  Default is `'0'`.

- **y** – <*coordinate*> – **center** parameter

  See **x**.

  Default is `'0'`.

- **width** – <*length*> – **size** parameter

  See **x**.

  A negative value is an error. A value of zero disables rendering of the element (i.e., no paint is applied).

  Default is `'0'`.

- **height** – <*length*> – **size** parameter

  See **x**.

  A negative value is an error. A value of zero disables rendering of the element (i.e., no paint is applied).

  Default is `'0'`.

- **xlink:href** – *string* – **inherit** parameter

  A URI reference to a different **pattern** element within the current SVG document fragment. Any attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has no children, and the referenced element does (possibly due to its own **xlink:href** attribute), then this element inherits the children from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attributes or children due to its own **xlink:href** attribute, then the current element can inherit those attributes or children.

- **preserveAspectRatio** – `'[defer] <align> [<meetOrSlice>]'`

  Use the *ViewBox* interface to set **viewbox** and **preserveAspectRatio**.

CHAPTER 35

SolidColor

The *solidColor* element is a paint server that provides a single color with opacity. It can be referenced like the other paint servers (i.e. gradients).

class svgwrite.solidcolor.**SolidColor**(*color='currentColor'*, *opacity=None*, ***extra*)

The *solidColor* element is a paint server that provides a single color with opacity. It can be referenced like the other paint servers (i.e. gradients). The *color* parameter specifies the color that shall be used for this *solidColor* element. The keyword `"currentColor"` can be specified in the same manner as within a <paint> specification for the *fill* and *stroke* properties. The *opacity* parameter defines the opacity of the *solidColor*.

See also:

https://www.w3.org/TR/SVGTiny12/painting.html#SolidColorElement

35.1 Methods

SolidColor.**__init__**(*color='currentColor'*, *opacity=None*, ***extra*)

Parameters

- **color** – solid color like the other paint servers (i.e. gradients).
- **opacity** (*float*) – opacity of the solid color in the range *0.0* (fully transparent) to *1.0* (fully opaque)

35.2 SVG Attributes

- **solid-color** – `'currentColor | <color> | inherit'` (__init__() parameter *color*)

  The *solid-color* attribute specifies the color that shall be used for this *solidColor* element. The keyword `"currentColor"` can be specified in the same manner as within a <*paint*> specification for the *fill* and *stroke* properties.

- **solid-opacity** – `'<opacity-value> | inherit'` (__init__() parameter *opacity*)

The *solid-opacity* parameter defines the opacity of the *solidColor*. Any values outside the range *0.0* (fully transparent) to *1.0* (fully opaque) must be clamped to this range.

CHAPTER 36

ClipPath

class svgwrite.masking.**ClipPath**(***extra*)

The clipping path restricts the region to which paint can be applied. Conceptually, any parts of the drawing that lie outside of the region bounded by the currently active clipping path are not drawn. A clipping path can be thought of as a mask wherein those pixels outside the clipping path are black with an alpha value of zero and those pixels inside the clipping path are white with an alpha value of one (with the possible exception of anti-aliasing along the edge of the silhouette).

A **clipPath** element can contain **path** elements, **text** elements, basic shapes (such as **circle**) or a **use** element. If a **use** element is a child of a **clipPath** element, it must directly reference **path**, **text** or basic shape elements. Indirect references are an error.

Adding clipping elements to `ClipPath`:

```
dwg = svgwrite.Drawing()
clip_path = dwg.defs.add(dwg.clipPath())
clip_path.add(dwg.circle((100, 100), 50))
```

See also:

http://www.w3.org/TR/SVG11/masking.html#ClippingPaths

36.1 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **transform** – use *svgwrite.mixins.Transform* methods
- **clipPathUnits** – `'userSpaceOnUse | objectBoundingBox'`

  Defines the coordinate system for the contents of the **clipPath**.

  If clipPathUnits = `'userSpaceOnUse'` , the contents of the **clipPath** represent values in the current user coordinate system in place at the time when the **clipPath** element is referenced (i.e., the user coordinate system for the element referencing the **clipPath** element via the **clip-path** property).

  If clipPathUnits = `'objectBoundingBox'` , then the user coordinate system for the contents of the **clipPath** element is established using the bounding box of the element to which the clipping path is applied.

  Default is `'userSpaceOnUse'`

36.2 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Presentation Attributes*

CHAPTER 37

Mask

class svgwrite.masking.**Mask**(*start=None*, *size=None*, ***extra*)

In SVG, you can specify that any other graphics object or **g** element can be used as an alpha mask for compositing the current object into the background.

A **mask** can contain any graphical elements or container elements such as a **g**.

See also:

http://www.w3.org/TR/SVG11/masking.html#Masking

37.1 SVG Attributes

- **class** – *string*

  assigns one or more css-class-names to an element

- **style** – *string*

  allows per-element css-style rules to be specified directly on a given element

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **maskUnits** – `'userSpaceOnUse | objectBoundingBox'`

  Defines the coordinate system for attributes **x**, **y**, **width** and **height**.

  If maskUnits = `'userSpaceOnUse'` , **x**, **y**, **width** and **height** represent values in the current user coordinate system in place at the time when the **mask** element is referenced (i.e., the user coordinate system for the element referencing the **mask** element via the **mask** property).

  If maskUnits = `'objectBoundingBox'` , **x**, **y**, **width** and **height** represent fractions or percentages of the bounding box of the element to which the mask is applied.

  Default is `'objectBoundingBox'`.

- **maskContentUnits** – `'userSpaceOnUse | objectBoundingBox'`

  Defines the coordinate system for the contents of the **mask**.

  If maskContentUnits = `'userSpaceOnUse'` , the user coordinate system for the contents of the **mask** element is the current user coordinate system in place at the time when the **mask** element is referenced (i.e., the user coordinate system for the element referencing the **mask** element via the **mask** property).

  If maskContentUnits = `'objectBoundingBox'` , the user coordinate system for the contents of the **mask** is established using the bounding box of the element to which the mask is applied.

  Default is `'userSpaceOnUse'`.

- **x** – <*coordinate*> – **start** parameter

  The x-axis coordinate of one corner of the rectangle for the largest possible offscreen buffer. Note that the clipping path used to render any graphics within the mask will consist of the intersection of the current clipping path associated with the given object and the rectangle defined by **x**, **y**, **width** and **height**.

  Default is `'-10%'`.

- **y** – <*coordinate*> – **start** parameter

  The y-axis coordinate of one corner of the rectangle for the largest possible offscreen buffer.

  Default is `'-10%'`.

- **width** – <*length*> – **size** parameter

  The width of the largest possible offscreen buffer. Note that the clipping path used to render any graphics within the mask will consist of the intersection of the current clipping path associated with the given object and the rectangle defined by **x**, **y**, **width** and **height**.

  Default is `'120%'`.

- **height** – <*length*> – **size** parameter

  The height of the largest possible offscreen buffer.

  Default is `'120%'`.

37.2 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *Presentation Attributes*

CHAPTER 38

animate module

Because the Web is a dynamic medium, SVG supports the ability to change vector graphics over time.

See also:

http://www.w3.org/TR/SVG11/animate.html

CHAPTER 39

Set

class svgwrite.animate.**Set**(*href=None*, ***extra*)

The **set** element provides a simple means of just setting the value of an attribute for a specified duration. It supports all attribute types, including those that cannot reasonably be interpolated, such as string and boolean values. The **set** element is non-additive. The additive and accumulate attributes are not allowed, and will be ignored if specified.

See also:

http://www.w3.org/TR/SVG11/animate.html#SetElement

39.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*

39.2 Methods

Set.**__init__**(*href=None*, ***extra*)

Set constructor.

Parameters **href** – target svg element, if **href** is not *None*; else the target SVG Element is the parent SVG Element.

Animate.**set_href**(*element*)

Parameters **element** – set target svg element to *element*

Animate.**set_target**(*attributeName*, *attributeType=None*)

Set animation attributes *attributeName* and *attributeType*.

Animate.**set_event**(*onbegin=None*, *onend=None*, *onrepeat=None*, *onload=None*)

Set animation attributes *onbegin*, *onend*, *onrepeat* and *onload*.

Animate.**set_timing**(*begin=None*, *end=None*, *dur=None*, *min=None*, *max=None*, *restart=None*, *repeatCount=None*, *repeatDur=None*)
Set animation attributes *begin*, *end*, *dur*, *min*, *max*, *restart*, *repeatCount* and *repeatDur*.

Animate.**freeze**()
Freeze the animation effect. (see also *fill*)

39.3 SVG Animation Attributes

- onbegin, onend, onrepeat, onload (*Animation Event Attributes*)
- attributeType, attributeName (*Animation Target Attributes*)
- begin, dur, end, min, max, restart, repeatCount, repeatDur, fill (*Animation Timing Attributes*)

39.4 SVG Attributes

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **to** – <*value*> Specifies the value for the attribute during the duration of the **set** element. The argument value must match the attribute type.

39.5 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *XLink Attributes*

CHAPTER 40

AnimateMotion

class svgwrite.animate.**AnimateMotion**(*path=None*, *href=None*, ***extra*)

The **animateMotion** element causes a referenced element to move along a motion path.

See also:

http://www.w3.org/TR/SVG11/animate.html#AnimateMotionElement

40.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*
- *svgwrite.animate.Set*

40.2 Methods

AnimateMotion.**__init__**(*path=None*, *href=None*, ***extra*)

Parameters

- **path** – the motion path
- **href** – target svg element, if **href** is not *None*; else the target SVG Element is the parent SVG Element.

AnimateMotion.**set_value**(*path=None*, *calcMode=None*, *keyPoints=None*, *rotate=None*)

Set animation attributes *path*, *calcMode*, *keyPoints* and *rotate*.

40.3 SVG Animation Attributes

- onbegin, onend, onrepeat, onload (*Animation Event Attributes*)
- begin, dur, end, min, max, restart, repeatCount, repeatDur, fill (*Animation Timing Attributes*)
- calcMode, values, keyTimes, keySplines, from, to, by (*Animation Value Attributes*)
- additive, accumulate (*Animation Addition Attributes*)

40.4 SVG Attributes

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **calcMode** – `'discrete | linear | paced | spline'`

  Specifies the interpolation mode for the animation.

- **path** – <*path-data*> – **path** parameter

  The motion path, expressed in the same format and interpreted the same way as the *d* attribute on the **Path** element. The effect of a motion path animation is to add a supplemental transformation matrix onto the CTM for the referenced object which causes a translation along the x- and y-axes of the current user coordinate system by the computed X and Y values computed over time.

- **keyPoints** – <*list-of-numbers*>

  keyPoints takes a semicolon-separated list of floating point values between 0 and 1 and indicates how far along the motion path the object shall move at the moment in time specified by corresponding **keyTimes** value. Distance calculations use the user agent's distance along the path algorithm. Each progress value in the list corresponds to a value in the **keyTimes** attribute list.

  If a list of **keyPoints** is specified, there must be exactly as many values in the **keyPoints** list as in the **keyTimes** list.

  If there are any errors in the **keyPoints** specification (bad values, too many or too few values), then the document is in error.

- **rotate** – <*number*> | `'auto'` | `'auto-reverse'`

  The **rotate** attribute post-multiplies a supplemental transformation matrix onto the CTM of the target element to apply a rotation transformation about the origin of the current user coordinate system. The rotation transformation is applied after the supplemental translation transformation that is computed due to the **path** attribute.

  - `'auto'`

    Indicates that the object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path.

  - `'auto-reverse'`

    Indicates that the object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path plus 180 degrees.

  - <*number*>

    Indicates that the target element has a constant rotation transformation applied to it, where the rotation angle is the specified number of degrees.

    Default value is `'0'`.

- **origin** – `'default'`

  The **origin** attribute is defined in the SMIL Animation specification

40.5 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *XLink Attributes*

CHAPTER 41

Animate

class svgwrite.animate.**Animate**(*attributeName=None*, *values=None*, *href=None*, ***extra*)

The **animate** element allows scalar attributes and properties to be assigned different values over time .

See also:

http://www.w3.org/TR/SVG11/animate.html#AnimateElement

41.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*
- *svgwrite.animate.Set*

41.2 Methods

Animate.**__init__**(*attributeName=None*, *values=None*, *href=None*, ***extra*)

Parameters

- **attributeName** – name of the SVG Attribute to animate
- **values** – interpolation values, *string* as <*semicolon-list*> or a python *list*
- **href** – target svg element, if **href** is not *None*; else the target SVG Element is the parent SVG Element.

Animate.**set_value**(*values*, *calcMode=None*, *keyTimes=None*, *keySplines=None*, *from_=None*, *to=None*, *by=None*)

Set animation attributes *values*, *calcMode*, *keyTimes*, *keySplines*, *from*, *to* and *by*.

41.3 SVG Animation Attributes

- onbegin, onend, onrepeat, onload (*Animation Event Attributes*)
- attributeType, attributeName (*Animation Target Attributes*)
- begin, dur, end, min, max, restart, repeatCount, repeatDur, fill (*Animation Timing Attributes*)
- calcMode, values, keyTimes, keySplines, from, to, by (*Animation Value Attributes*)
- additive, accumulate (*Animation Addition Attributes*)

41.4 SVG Attributes

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

41.5 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *XLink Attributes*

CHAPTER 42

AnimateColor

class svgwrite.animate.**AnimateColor**(*attributeName=None*, *values=None*, *href=None*, ***extra*)

The **animateColor** element specifies a color transformation over time.

See also:

http://www.w3.org/TR/SVG11/animate.html#AnimateColorElement

The **from**, **by** and **to** attributes take color values, where each color value is expressed using the following syntax (the same syntax as used in SVG's properties that can take color values):

<color> <icccolor>?

The **values** attribute for the **animateColor** element consists of a semicolon-separated list of color values, with each color value expressed in the above syntax.

Out of range color values can be provided, but user agent processing will be implementation dependent. User agents should clamp color values to allow color range values as late as possible, but note that system differences might preclude consistent behavior across different systems.

The **color-interpolation** property applies to color interpolations that result from **animateColor** animations.

42.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*
- *svgwrite.animate.Animate*

42.2 SVG Animation Attributes

- onbegin, onend, onrepeat, onload (*Animation Event Attributes*)

- attributeType, attributeName (*Animation Target Attributes*)
- begin, dur, end, min, max, restart, repeatCount, repeatDur, fill (*Animation Timing Attributes*)
- calcMode, values, keyTimes, keySplines, from, to, by (*Animation Value Attributes*)
- additive, accumulate (*Animation Addition Attributes*)

42.3 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *XLink Attributes*

CHAPTER 43

AnimateTransform

class svgwrite.animate.**AnimateTransform**(*transform*, *element=None*, ***extra*)

The **animateTransform** element animates a transformation attribute on a target element, thereby allowing animations to control translation, scaling, rotation and/or skewing.

See also:

http://www.w3.org/TR/SVG11/animate.html#AnimateTransformElement

43.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*
- *svgwrite.animate.Animate*

43.2 Methods

AnimateTransform.**__init__**(*transform*, *element=None*, ***extra*)

Parameters

- **element** – target svg element, if element is not *None*; else the target svg element is the parent svg element.
- **transform** (*string*) – `'translate | scale | rotate | skewX | skewY'`

43.3 SVG Animation Attributes

- onbegin, onend, onrepeat, onload (*Animation Event Attributes*)

- attributeType, attributeName (*Animation Target Attributes*)
- begin, dur, end, min, max, restart, repeatCount, repeatDur, fill (*Animation Timing Attributes*)
- calcMode, values, keyTimes, keySplines, from, to, by (*Animation Value Attributes*)
- additive, accumulate (*Animation Addition Attributes*)

43.4 SVG Attributes

- **externalResourcesRequired** – *bool*

  False: if document rendering can proceed even if external resources are unavailable else: *True*

- **type** – `'translate | scale | rotate | skewX | skewY'`

  Indicates the type of transformation which is to have its values change over time. If the attribute is not specified, then the effect is as if a value of **translate** were specified.

The **from**, **by** and **to** attributes take a value expressed using the same syntax that is available for the given transformation type:

- For a type = `'translate'`, each individual value is expressed as <*tx*> *[,<ty>]*.
- For a type = `'scale'`, each individual value is expressed as <*sx*> *[,<sy>]*.
- For a type = `'rotate'`, each individual value is expressed as <*rotate-angle*> *[<cx> <cy>]*.
- For a type = `'skewX'` and type = `'skewY'`, each individual value is expressed as <*skew-angle*>.

43.5 Standard SVG Attributes

- *Core Attributes*
- *Conditional Processing Attributes*
- *XLink Attributes*

CHAPTER 44

SVG Animation Attributes

44.1 Animation Events Attributes

44.1.1 onbegin

See also:

http://www.w3.org/TR/SVG11/script.html#OnBeginEventAttribute

onbegin = <*anything*>

Specifies some script to execute when "bubbling" or "at target" phase listeners for the corresponding event are fired on the element the attribute is specified on. See the Complete list of support events to determine which event each of these event attributes corresponds to.

Complete list of support events: http://www.w3.org/TR/SVG11/interact.html#SVGEvents

44.1.2 onend

onend = <*anything*>

Complete list of support events: http://www.w3.org/TR/SVG11/interact.html#SVGEvents

See also:

http://www.w3.org/TR/SVG11/script.html#OnEndEventAttribute

44.1.3 onrepeat

onrepeat = <*anything*>

Complete list of support events: http://www.w3.org/TR/SVG11/interact.html#SVGEvents

See also:

http://www.w3.org/TR/SVG11/script.html#OnRepeatEventAttribute

44.1.4 onload

onload = <*anything*>

Specifies some script to execute when "bubbling" or "at target" phase listeners for the SVGLoad event are fired on the element the attribute is specified on.

See also:

http://www.w3.org/TR/SVG11/script.html#OnLoadEventAttribute

44.2 Animation Target Attributes

44.2.1 attributeType

Specifies the namespace in which the target attribute and its associated values are defined. The attribute value is one of the following (values are case-sensitive):

value	description
`CSS`	This specifies that the value of **attributeName** is the name of a CSS property defined as animatable in this specification.
`XML`	This specifies that the value of **attributeName** is the name of an XML attribute defined in the default XML namespace for the target element. If the value for **attributeName** has an XMLNS prefix, the implementation must use the associated namespace as defined in the scope of the target element. The attribute must be defined as animatable in this specification.
`auto`	The implementation should match the **attributeName** to an attribute for the target element. The implementation must first search through the list of CSS properties for a matching property name, and if none is found, search the default XML namespace for the element.

The default value is `'auto'`.

See also:

http://www.w3.org/TR/SVG11/animate.html#AttributeTypeAttribute

44.2.2 attributeName

attributeName = <*attributeName*>

Specifies the name of the target attribute. An XMLNS prefix may be used to indicate the XML namespace for the attribute. The prefix will be interpreted in the scope of the current (i.e., the referencing) animation element.

See also:

http://www.w3.org/TR/SVG11/animate.html#AttributeNameAttribute

44.3 Animation Timing Attributes

44.3.1 begin

begin = <*begin-value-list*>

Defines when the element should begin (i.e. become active).

The attribute value is a semicolon separated list of values.

See also:

http://www.w3.org/TR/SVG11/animate.html#BeginAttribute

44.3.2 dur

dur = <*Clock-value*> | `'media | indefinite'`

Specifies the simple duration.

The attribute value can be one of the following:

value	description
<*Clock-value*>	Specifies the length of the simple duration in presentation time. Value must be greater than 0.
`media`	Specifies the simple duration as the intrinsic media duration. This is only valid for elements that define media.(For SVG's animation elements, if `'media'` is specified, the attribute will be ignored.)
`indefinite`	Specifies the simple duration as indefinite.

If the animation does not have a **dur** attribute, the simple duration is indefinite. Note that interpolation will not work if the simple duration is indefinite (although this may still be useful for **set** elements).

See also:

http://www.w3.org/TR/SVG11/animate.html#DurAttribute

44.3.3 end

end = <*end-value-list*>

Defines an end value for the animation that can constrain the active duration.

The attribute value is a semicolon separated list of values.

A value of `'indefinite'` specifies that the end of the animation will be determined by an endElement method call (the animation DOM methods are described in DOM interfaces).

See also:

http://www.w3.org/TR/SVG11/animate.html#EndAttribute

44.3.4 min

min = <*Clock-value*> | `'media'`

Specifies the minimum value of the active duration.

value	description
<*Clock-value*>	Specifies the length of the minimum value of the active duration, measured in local time. Value must be greater than 0.
`'media'`	Specifies the minimum value of the active duration as the intrinsic media duration. This is only valid for elements that define media. (For SVG's animation elements, if `'media'` is specified, the attribute will be ignored.)

The default value for **min** is `'0'`. This does not constrain the active duration at all.

See also:

http://www.w3.org/TR/SVG11/animate.html#MinAttribute

44.3.5 max

max = <*Clock-value*> | `'media'`

Specifies the maximum value of the active duration.

value	description
<*Clock-value*>	Specifies the length of the maximum value of the active duration, measured in local time. Value must be greater than 0.
`'media'`	Specifies the maximum value of the active duration as the intrinsic media duration. This is only valid for elements that define media. (For SVG's animation elements, if `'media'` is specified, the attribute will be ignored.)

There is no default value for **max**. This does not constrain the active duration at all.

See also:

http://www.w3.org/TR/SVG11/animate.html#MaxAttribute

44.3.6 restart

restart = `'always | whenNotActive | never'`

value	description
`'always'`	The animation can be restarted at any time. This is the default value.
`'whenNotActive'`	The animation can only be restarted when it is not active (i.e. after the active end). Attempts to restart the animation during its active duration are ignored.
`'never'`	The element cannot be restarted for the remainder of the current simple duration of the parent time container. (In the case of SVG, since the parent time container is the SVG document fragment, then the animation cannot be restarted for the remainder of the document duration.)

See also:

http://www.w3.org/TR/SVG11/animate.html#RestartAttribute

44.3.7 repeatCount

repeatCount = <*number*> | `'indefinite'`

Specifies the number of iterations of the animation function. It can have the following attribute values:

value	description
<number>	This is a (base 10) "floating point" numeric value that specifies the number of iterations. It can include partial iterations expressed as fraction values. A fractional value describes a portion of the simple duration. Values must be greater than 0.
`'indefinite'`	The animation is defined to repeat indefinitely (i.e. until the document ends).

See also:

http://www.w3.org/TR/SVG11/animate.html#RepeatCountAttribute

44.3.8 repeatDur

repeatDur = <*Clock-value*> | `'indefinite'`

Specifies the total duration for repeat. It can have the following attribute values:

value	description
<*Clock-value*>	Specifies the duration in presentation time to repeat the animation function f(t).
`'indefinite'`	The animation is defined to repeat indefinitely (i.e. until the document ends).

See also:

http://www.w3.org/TR/SVG11/animate.html#RepeatDurAttribute

44.3.9 fill

fill = `'freeze | remove'`

This attribute can have the following values:

value	description
`'freeze'`	The animation effect F(t) is defined to freeze the effect value at the last value of the active duration. The animation effect is "frozen" for the remainder of the document duration (or until the animation is restarted - see SMIL Animation: Restarting animation).
`'remove'`	The animation effect is removed (no longer applied) when the active duration of the animation is over. After the active end of the animation, the animation no longer affects the target (unless the animation is restarted - see SMIL Animation: Restarting animation). This is the default value.

See also:

http://www.w3.org/TR/SVG11/animate.html#FillAttribute

44.4 Animation Value Attributes

44.4.1 calcMode

calcMode = `'discrete | linear | paced | spline'`

Specifies the interpolation mode for the animation. This can take any of the following values. The default mode is `'linear'`, however if the attribute does not support linear interpolation (e.g. for strings), the **calcMode** attribute is ignored and discrete interpolation is used.

value	description
`'discrete'`	This specifies that the animation function will jump from one value to the next without any interpolation.
`'linear'`	Simple linear interpolation between values is used to calculate the animation function. Except for **animateMotion**, this is the default **calcMode**.
`'paced'`	Defines interpolation to produce an even pace of change across the animation. This is only supported for values that define a linear numeric range, and for which some notion of "distance" between points can be calculated (e.g. position, width, height, etc.). If `'paced'` is specified, any **keyTimes** or **keySplines** will be ignored. For **animateMotion**, this is the default **calcMode**.
`'spline'`	Interpolates from one value in the **values** list to the next according to a time function defined by a cubic Bézier spline. The points of the spline are defined in the **keyTimes** attribute, and the control points for each interval are defined in the **keySplines** attribute.

See also:

http://www.w3.org/TR/SVG11/animate.html#CalcModeAttribute

44.4.2 values

values = <*list*>

A semicolon-separated list of one or more values. Vector-valued attributes are supported using the vector syntax of the **attributeType** domain. Per the SMIL specification, leading and trailing white space, and white space before and after semicolon separators, is allowed and will be ignored.

See also:

http://www.w3.org/TR/SVG11/animate.html#ValuesAttribute

44.4.3 keyTimes

keyTimes = <*list*>

A semicolon-separated list of time values used to control the pacing of the animation. Each time in the list corresponds to a value in the **values** attribute list, and defines when the value is used in the animation function. Each time value in the **keyTimes** list is specified as a floating point value between 0 and 1 (inclusive), representing a proportional offset into the simple duration of the animation element.

If a list of **keyTimes** is specified, there must be exactly as many values in the **keyTimes** list as in the **values** list.

Each successive time value must be greater than or equal to the preceding time value.

The **keyTimes** list semantics depends upon the interpolation mode:

- For linear and spline animation, the first time value in the list must be 0, and the last time value in the list must be 1. The key time associated with each value defines when the value is set; values are interpolated between the key times.
- For discrete animation, the first time value in the list must be 0. The time associated with each value defines when the value is set; the animation function uses that value until the next time defined in **keyTimes**.

If the interpolation mode is `'paced'`, the **keyTimes** attribute is ignored.

If there are any errors in the **keyTimes** specification (bad values, too many or too few values), the document fragment is in error.

If the simple duration is indefinite, any **keyTimes** specification will be ignored.

See also:

http://www.w3.org/TR/SVG11/animate.html#KeyTimesAttribute

44.4.4 keySplines

keySplines = <*list*>

A set of Bézier control points associated with the **keyTimes** list, defining a cubic Bézier function that controls interval pacing. The attribute value is a semicolon-separated list of control point descriptions. Each control point description is a set of four values: x1 y1 x2 y2, describing the Bézier control points for one time segment. Note: SMIL allows these values to be separated either by commas with optional whitespace, or by whitespace alone. The 'keyTimes' values that define the associated segment are the Bézier "anchor points", and the 'keySplines' values are the control points. Thus, there must be one fewer sets of control points than there are **keyTimes**.

The values must all be in the range 0 to 1.

This attribute is ignored unless the **calcMode** is set to `'spline'`.

If there are any errors in the **keySplines** specification (bad values, too many or too few values), the document fragment is in error.

See also:

http://www.w3.org/TR/SVG11/animate.html#KeySplinesAttribute

44.4.5 from

from = <*value*>

Specifies the starting value of the animation.

See also:

http://www.w3.org/TR/SVG11/animate.html#FromAttribute

44.4.6 to

to = <*value*>

Specifies the ending value of the animation.

See also:

http://www.w3.org/TR/SVG11/animate.html#ToAttribute

44.4.7 by

by = <*value*>

Specifies a relative offset value for the animation.

See also:

http://www.w3.org/TR/SVG11/animate.html#ByAttribute

44.5 Animation Addition Attributes

44.5.1 additive

additive = `'replace | sum'`

Controls whether or not the animation is additive.

value	description
`'sum'`	Specifies that the animation will add to the underlying value of the attribute and other lower priority animations.
`'replace'`	Specifies that the animation will override the underlying value of the attribute and other lower priority animations. This is the default, however the behavior is also affected by the animation value attributes **by** and **to**, as described in SMIL Animation: How from, to and by attributes affect additive behavior.

See also:

http://www.w3.org/TR/SVG11/animate.html#AdditiveAttribute

44.5.2 accumulate

accumulate = `'none | sum'`

Controls whether or not the animation is cumulative.

value	description
`'sum'`	Specifies that each repeat iteration after the first builds upon the last value of the previous iteration.
`'none'`	Specifies that repeat iterations are not cumulative. This is the default.

This attribute is ignored if the target attribute value does not support addition, or if the animation element does not repeat.

Cumulative animation is not defined for “to animation”.

This attribute will be ignored if the animation function is specified with only the **to** attribute.

See also:

http://www.w3.org/TR/SVG11/animate.html#AccumulateAttribute

CHAPTER 45

Introduction

This chapter describes SVG's declarative filter effects feature set, which when combined with the 2D power of SVG can describe much of the common artwork on the Web in such a way that client-side generation and alteration can be performed easily. In addition, the ability to apply filter effects to SVG graphics elements and container elements helps to maintain the semantic structure of the document, instead of resorting to images which aside from generally being a fixed resolution tend to obscure the original semantics of the elements they replace. This is especially true for effects applied to text.

Filter effects are defined by **filter** elements. To apply a filter effect to a **graphics element** or a **container element**, you set the value of the **filter** property on the given element such that it references the filter effect.

See also:

http://www.w3.org/TR/SVG11/filters.html#Introduction

CHAPTER 46

Filter Element

class svgwrite.filters.**Filter**(*start=None*, *size=None*, *resolution=None*, *inherit=None*, ***extra*)

The filter element is a container element for filter primitives, and also a **factory** for filter primitives.

See also:

http://www.w3.org/TR/SVG11/filters.html#FilterElement

46.1 Parent Classes

- *svgwrite.base.BaseElement*
- *svgwrite.mixins.XLink*
- *svgwrite.mixins.Presentation*

46.2 Methods

Filter.**__init__**(*start=None*, *size=None*, *resolution=None*, *inherit=None*, ***extra*)

Parameters

- **start** (*2-tuple*) – defines the start point of the filter effects region (**x**, **y**)
- **size** (*2-tuple*) – defines the size of the filter effects region (**width**, **height**)
- **resolution** – takes the form `'x-pixels [y-pixels]'`, and indicates the width and height of the intermediate images in pixels.
- **inherit** – inherits properties from Filter *inherit* see: **xlink:href**

Filter.**feBlend**(*in_*, *start=None*, *size=None*, ***extra*)

create and add a *feBlend Filter Element*

`Filter.`**`feColorMatrix`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feColorMatrix Filter Element*

`Filter.`**`feComponentTransfer`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feComponentTransfer Filter Element*

`Filter.`**`feComposite`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feComposite Filter Element*

`Filter.`**`feConvolveMatrix`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feConvolveMatrix Filter Element*

`Filter.`**`feDiffuseLighting`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feDiffuseLighting Filter Element*

`Filter.`**`feDisplacementMap`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feDisplacementMap Filter Element*

`Filter.`**`feFlood`**(*start=None*, *size=None*, ***extra*)
create and add a *feFlood Filter Element*

`Filter.`**`feGaussianBlur`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feGaussianBlur Filter Element*

`Filter.`**`feImage`**(*href*, *start=None*, *size=None*, ***extra*)
create and add a *feImage Filter Element*

`Filter.`**`feMerge`**(*start=None*, *size=None*, ***extra*)
create and add a *feMerge Filter Element*

`Filter.`**`feMorphology`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feMorphology Filter Element*

`Filter.`**`feOffset`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feOffset Filter Element*

`Filter.`**`feSpecularLighting`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feSpecularLighting Filter Element*

`Filter.`**`feTile`**(*in_*, *start=None*, *size=None*, ***extra*)
create and add a *feTile Filter Element*

`Filter.`**`feTurbulence`**(*start=None*, *size=None*, ***extra*)
create and add a *feTurbulence Filter Element*

46.3 SVG Attributes

- **filterUnits** – `'userSpaceOnUse | objectBoundingBox'`

  See Filter effects region.

- **primitiveUnits** – `'userSpaceOnUse | objectBoundingBox'` Specifies the coordinate system for the various length values within the filter primitives and for the attributes that define the filter primitive subregion.

  If **primitiveUnits** = `'userSpaceOnUse'`, any length values within the filter definitions represent values in the current user coordinate system in place at the time when the **filter** element is referenced (i.e., the user coordinate system for the element referencing the **filter** element via a **filter** property).

  If **primitiveUnits** = `'objectBoundingBox'`, then any length values within the filter definitions represent fractions or percentages of the bounding box on the referencing element (see Object bounding box units). Note

that if only one number was specified in a <*number-optional-number*> value this number is expanded out before the **primitiveUnits** computation takes place.

If attribute **primitiveUnits** is not specified, then the effect is as if a value of `'userSpaceOnUse'` were specified.

- **x** – <*coordinate*> – **start** parameter

  See Filter effects region.

- **y** – <*coordinate*> – **start** parameter

  See Filter effects region.

- **width** – <*length*> – **size** parameter

  See Filter effects region.

- **height** – <*length*> – **size** parameter

  See Filter effects region.

- **filterRes** – <*number-optional-number*> – **resolution** parameter

  See Filter effects region.

- **xlink:href** – <*iri*> – **inherit** parameter

  A IRI reference to another **filter** element within the current SVG document fragment. Any attributes which are defined on the referenced **filter** element which are not defined on this element are inherited by this element.

46.4 Standard SVG Attributes

- *Core Attributes*
- *Presentation Attributes*
- *XLink Attributes*

46.5 Example

Source: https://secure.wikimedia.org/wikibooks/de/wiki/SVG/_Effekte#Urfilter_fePointLight.2C_Punktlichtquelle

```
    import sys
    from pathlib import Path
    sys.path.insert(0, str(Path(__file__).resolve().parent.parent))

import svgwrite
dwg = svgwrite.Drawing("fePointLight.svg")

filtr = dwg.defs.add(
    dwg.filter(id="DL", start=(0, 0), size=(500, 500),
               filterUnits="userSpaceOnUse"))
diffuse_lighting = filtr.feDiffuseLighting(
    start=(0, 0), size=(500, 500),
    surfaceScale=10,
    diffuseConstant=1,
    kernelUnitLength=1,
    lighting_color="#f8f")
```

(continues on next page)

(continued from previous page)

```
point_light = diffuse_lighting.fePointLight( (500, 250, 250) )
point_light.add(
    dwg.animate('x',
        values=(0,100,500,100,0),
        dur='30s',
        repeatDur='indefinite'))
point_light.add(
    dwg.animate('y',
        values=(0,500,400,-100,0),
        dur='31s',
        repeatDur='indefinite'))
point_light.add(
    dwg.animate('z',
        values=(0,1000,500,-100,0),
        dur='37s',
        repeatDur='indefinite'))
dwg.save()
```

and the XML result (with manual reformatting):

```
<?xml version="1.0" encoding="utf-8" ?>
<svg baseProfile="full" height="100%" version="1.1" width="100%"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:xlink="http://www.w3.org/1999/xlink">
    <defs>
        <filter id="DL" filterUnits="userSpaceOnUse"
          x="0" y="0" width="500" height="500" >
            <feDiffuseLighting diffuseConstant="1"
              x="0" y="0" width="500" height="500"
              in="SourceGraphic"
              kernelUnitLength="1"
              lighting-color="#f8f"
              surfaceScale="10">
                <fePointLight x="500" y="250" z="250">
                    <animate attributeName="x"
                      dur="30s"
                      repeatDur="indefinite"
                      values="0;100;500;100;0" />
                    <animate attributeName="y"
                      dur="31s"
                      repeatDur="indefinite"
                      values="0;500;400;-100;0" />
                    <animate attributeName="z"
                      dur="37s"
                      repeatDur="indefinite"
                      values="0;1000;500;-100;0" />
                </fePointLight>
            </feDiffuseLighting>
        </filter>
    </defs>
</svg>
```

CHAPTER 47

Filter Primitives Overview

See also:

http://www.w3.org/TR/SVG11/filters.html#FilterPrimitivesOverview

Unless otherwise stated, all image filters operate on premultiplied RGBA samples. Filters which work more naturally on non-premultiplied data (feColorMatrix and feComponentTransfer) will temporarily undo and redo premultiplication as specified. All raster effect filtering operations take 1 to N input RGBA images, additional attributes as parameters, and produce a single output RGBA image.

The RGBA result from each filter primitive will be clamped into the allowable ranges for colors and opacity values. Thus, for example, the result from a given filter primitive will have any negative color values or opacity values adjusted up to color/opacity of zero.

The color space in which a particular filter primitive performs its operations is determined by the value of property **color-interpolation-filters** on the given filter primitive. A different property, **color-interpolation** determines the color space for other color operations. Because these two properties have different initial values (**color-interpolation-filters** has an initial value of linearRGB whereas **color-interpolation** has an initial value of sRGB), in some cases to achieve certain results (e.g., when coordinating gradient interpolation with a filtering operation) it will be necessary to explicitly set **color-interpolation** to linearRGB or **color-interpolation-filters** to sRGB on particular elements. Note that the examples below do not explicitly set either **color-interpolation** or **color-interpolation-filters**, so the initial values for these properties apply to the examples.

CHAPTER 48

Common SVG Attributes for Filter Primitives

With the exception of the **in** attribute, all of the following attributes are available on all filter primitive elements:

- **x** – <*coordinate*>

  The minimum x coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

- **y** – <*coordinate*>

  The minimum y coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

- **width** – <*length*>

  The width of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

- **height** – <*length*>

  The height of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

- **result** – <*filter-primitive-reference*>

  Assigned name for this filter primitive. If supplied, then graphics that result from processing this filter primitive can be referenced by an **in** attribute on a subsequent filter primitive within the same **filter** element. If no value is provided, the output will only be available for re-use as the implicit input into the next filter primitive if that filter primitive provides no value for its **in** attribute.

- **in** – `'SourceGraphic | SourceAlpha | BackgroundImage | BackgroundAlpha | FillPaint | StrokePaint'` | <*filter-primitive-reference*>

  Identifies input for the given filter primitive. The value can be either one of six keywords or can be a string which matches a previous **result** attribute value within the same **filter** element. If no value is provided and this is the first filter primitive, then this filter primitive will use `'SourceGraphic'` as its input. If no value is provided and this is a subsequent filter primitive, then this filter primitive will use the result from the previous filter primitive as its input.

- **SourceGraphic**

  This keyword represents the graphics elements that were the original input into the **filter** element. For raster effects filter primitives, the graphics elements will be rasterized into an initially clear RGBA raster in image space. Pixels left untouched by the original graphic will be left clear. The image is specified to be rendered in linear RGBA pixels. The alpha channel of this image captures any anti-aliasing specified by SVG. (Since the raster is linear, the alpha channel of this image will represent the exact percent coverage of each pixel.)

- **SourceAlpha**

  This keyword represents the graphics elements that were the original input into the **filter** element. SourceAlpha has all of the same rules as SourceGraphic except that only the alpha channel is used. The input image is an RGBA image consisting of implicitly black color values for the RGB channels, but whose alpha channel is the same as SourceGraphic. If this option is used, then some implementations might need to rasterize the graphics elements in order to extract the alpha channel.

- **BackgroundImage**

  This keyword represents an image snapshot of the canvas under the filter region at the time that the **filter** element was invoked. See Accessing the background image.

- **BackgroundAlpha**

  Same as BackgroundImage except only the alpha channel is used. See SourceAlpha and Accessing the background image.

- **FillPaint**

  This keyword represents the value of the **fill** property on the target element for the filter effect. The FillPaint image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the *"paint"* itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.

- **StrokePaint**

  This keyword represents the value of the **stroke** property on the target element for the filter effect. The StrokePaint image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the *"paint"* itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.

CHAPTER 49

feBlend Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feBlendElement

This filter composites two objects together using commonly used imaging software blending modes. It performs a pixel-wise combination of two input images.

For common properties see: *Filter Primitives Overview*

49.1 SVG Attributes

- **mode** – `'normal | multiply | screen | darken | lighten'`

  One of the image blending modes. If attribute **mode** is not specified, then the effect is as if a value of `'normal'` were specified.

  see also: http://www.w3.org/TR/SVG11/filters.html#feBlendModeAttribute

- **in** – (see *in* attribute)
- **in2** – (see *in* attribute)

  The second input image to the blending operation. This attribute can take on the same values as the **in** attribute.

CHAPTER 50

feColorMatrix Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feColorMatrixElement

For common properties see: *Filter Primitives Overview*

50.1 SVG Attributes

- **in** – (see *in* attribute)
- **type** – `'matrix | saturate | hueRotate | luminanceToAlpha'`

  Indicates the type of matrix operation. The keyword **matrix** indicates that a full 5x4 matrix of values will be provided. The other keywords represent convenience shortcuts to allow commonly used color operations to be performed without specifying a complete matrix. If attribute **type** is not specified, then the effect is as if a value of matrix were specified.
- **values** = *list of <number>s*

  The contents of **values** depends on the value of attribute **type** see: http://www.w3.org/TR/SVG11/filters.html#feColorMatrixValuesAttribute

CHAPTER 51

feComponentTransfer Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feComponentTransferElement

This filter primitive performs component-wise remapping of data:

```
R' = feFuncR( R )
G' = feFuncG( G )
B' = feFuncB( B )
A' = feFuncA( A )
```

for every pixel. It allows operations like brightness adjustment, contrast adjustment, color balance or thresholding.

The calculations are performed on non-premultiplied color values. If the input graphics consists of premultiplied color values, those values are automatically converted into non-premultiplied color values for this operation. (Note that the undoing and redoing of the premultiplication can be avoided if feFuncA is the identity transform and all alpha values on the source graphic are set to 1.)

For common properties see: *Filter Primitives Overview*

51.1 SVG Attributes

- **in** – (see *in* attribute)

51.2 Methods

feFuncR(*type_*, ***extra*)
create and add a transfer function for the **red** component of the input graphic

feFuncG(*type_*, ***extra*)
create and add a transfer function for the **green** component of the input graphic

feFuncB(*type_*, ***extra*)
create and add a transfer function for the **blue** component of the input graphic

feFuncA(*type_*, ***extra*)
create and add a transfer function for the **alpha** component of the input graphic

51.2.1 Parameters for feFuncX() Methods

- **type** – `'identity | table | discrete | linear | gamma'`

  see: http://www.w3.org/TR/SVG11/filters.html#feComponentTransferTypeAttribute

- **tableValues** – *(list of <number>s)*

  When **type** = `'table'`, the list of *<number>s* v0,v1,... vn, separated by white space and/or a comma, which define the lookup table. An empty list results in an identity transfer function.

- **slope** – *<number>*

  When **type** = `'linear'`, the slope of the linear function.

- **intercept** – *<number>*

  When **type** = `'linear'`, the intercept of the linear function.

- **amplitude** – *<number>*

  When **type** = `'gamma'`, the amplitude of the gamma function.

- **exponent** – *<number>*

  When **type** = `'gamma'`, the exponent of the gamma function.

- **offset** – *<number>*

  When **type** = `'gamma'`, the offset of the gamma function.

CHAPTER 52

feComposite Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feCompositeElement

This filter performs the combination of the two input images pixel-wise in image space using one of the Porter-Duff compositing operations: over, in, atop, out, xor. Additionally, a component-wise arithmetic operation (with the result clamped between [0..1]) can be applied.

The arithmetic operation is useful for combining the output from the **feDiffuseLighting** and **feSpecularLighting** filters with texture data. It is also useful for implementing dissolve. If the arithmetic operation is chosen, each result pixel is computed using the following formula:

```
result = k1*i1*i2 + k2*i1 + k3*i2 + k4
```

For this filter primitive, the extent of the resulting image might grow as described in the section that describes the filter primitive subregion.

For common properties see: *Filter Primitives Overview*

52.1 SVG Attributes

- **in** – (see *in* attribute)
- **operator** – `'over | in | out | atop | xor | arithmetic`

  The compositing operation that is to be performed. All of the **operator** types except arithmetic match the correspond operation as described in [PORTERDUFF]. The arithmetic operator is described above. If attribute **operator** is not specified, then the effect is as if a value of `'over'` were specified.

- **k1**, **k2**, **k3**, **k4** – <*number*>

  Only applicable if **operator** = `'arithmetic'`. If the attribute is not specified, the effect is as if a value of 0 were specified.

- **in2** – (see *in* attribute)

The second input image to the compositing operation. This attribute can take on the same values as the **in** attribute.

CHAPTER 53

feConvolveMatrix Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feConvolveMatrixElement

feConvolveMatrix applies a matrix convolution filter effect. A convolution combines pixels in the input image with neighboring pixels to produce a resulting image. A wide variety of imaging operations can be achieved through convolutions, including blurring, edge detection, sharpening, embossing and beveling.

For common properties see: *Filter Primitives Overview*

53.1 SVG Attributes

- **in** – (see *in* attribute)
- **order** – <*number-optional-number*>

  Indicates the number of cells in each dimension for **kernelMatrix**. The values provided must be <*integer*>*'s greater than zero. The first number, '*<*orderX*>, indicates the number of columns in the matrix. The second number, <*orderY*>, indicates the number of rows in the matrix. If <*orderY*> is not provided, it defaults to <orderX>.

  If the attribute is not specified, the effect is as if a value of 3 were specified.

- **kernelMatrix** – <*list of numbers*>

  The list of <*number*>*'s that make up the kernel matrix for the convolution. Values are separated by space characters and/or a comma. The number of entries in the list must equal '*<*orderX*> times <*orderY*>.

- **divisor** – <*number*>

  After applying the **kernelMatrix** to the input image to yield a number, that number is divided by **divisor** to yield the final destination color value. The default value is the sum of all values in **kernelMatrix**, with the exception that if the sum is zero, then the divisor is set to 1.

- **bias** = <*number*>

After applying the **kernelMatrix** to the input image to yield a number and applying the **divisor**, the **bias** attribute is added to each component.

- **targetX** – <*integer*>

  Determines the positioning in X of the convolution matrix relative to a given target pixel in the input image. The leftmost column of the matrix is column number zero. The value must be such that: 0 <= targetX < orderX. By default, the convolution matrix is centered in X over each pixel of the input image (i.e., targetX = floor (orderX / 2)).

- **targetY – <*integer*>** Determines the positioning in Y of the convolution matrix relative to a given target pixel in the input image. The topmost row of the matrix is row number zero. The value must be such that: 0 <= targetY < orderY. By default, the convolution matrix is centered in Y over each pixel of the input image (i.e., targetY = floor (orderY / 2)).

- **edgeMode** – `'duplicate | wrap | none'`

  Determines how to extend the input image as necessary with color values so that the matrix operations can be applied when the kernel is positioned at or near the edge of the input image.

  - `'duplicate'` indicates that the input image is extended along each of its borders as necessary by duplicating the color values at the given edge of the input image.
  - `'wrap'` indicates that the input image is extended by taking the color values from the opposite edge of the image.
  - `'none'` indicates that the input image is extended with pixel values of zero for R, G, B and A.

  If attribute **edgeMode** is not specified, then the effect is as if a value of `'duplicate'` were specified.

- **kernelUnitLength** – <*number-optional-number*>

  The first number is the <*dx*> value. The second number is the <*dy*> value. If the <*dy*> value is not specified, it defaults to the same value as <*dx*>. Indicates the intended distance in current filter units (i.e., units as determined by the value of attribute **primitiveUnits**) between successive columns and rows, respectively, in the **kernelMatrix**. By specifying value(s) for **kernelUnitLength**, the kernel becomes defined in a scalable, abstract coordinate system. If **kernelUnitLength** is not specified, the default value is one pixel in the offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable. For some level of consistency across display media and user agents, it is necessary that a value be provided for at least one of **filterRes** and **kernelUnitLength**. In some implementations, the most consistent results and the fastest performance will be achieved if the pixel grid of the temporary offscreen images aligns with the pixel grid of the kernel. A negative or zero value is an error.

- **preserveAlpha** – `'false | true'`

  A value of false indicates that the convolution will apply to all channels, including the alpha channel.

  A value of true indicates that the convolution will only apply to the color channels. In this case, the filter will temporarily unpremultiply the color component values, apply the kernel, and then re-premultiply at the end.

  If **preserveAlpha** is not specified, then the effect is as if a value of `'false'` were specified.

CHAPTER 54

feDiffuseLighting Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feDiffuseLightingElement

This filter primitive lights an image using the alpha channel as a bump map. The resulting image is an RGBA opaque image based on the light color with alpha = 1.0 everywhere. The lighting calculation follows the standard diffuse component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map.

The light map produced by this filter primitive can be combined with a texture image using the multiply term of the arithmetic **feComposite** compositing method. Multiple light sources can be simulated by adding several of these light maps together before applying it to the texture image.

For common properties see: *Filter Primitives Overview*

54.1 Methods

`feDiffuseLighting.`**`feDistantLight`**(*azimuth=0*, *elevation=0*, ***extra*)

create and add a light source: *feDistantLight Filter Element*

`feDiffuseLighting.`**`fePointLight`**(*source=(0, 0, 0)*, ***extra*)

Parameters **`source`** – source 3D point (**x**, **y**, **z**)

create and add a light source: *fePointLight Filter Element*

`feDiffuseLighting.`**`feSpotLight`**(*source=(0, 0, 0)*, *target=(0, 0, 0)*, ***extra*)

Parameters

- **`source`** – source 3D point (**x**, **y**, **z**)
- **`target`** – target 3D point (**pointsAtX**, **pointsAtY**, **pointsAtZ**)

create and add a light source: *feSpotLight Filter Element*

54.2 SVG Attributes

- **in** – (see *in* attribute)
- **surfaceScale** – <*number*>

  height of surface when Ain = 1.

  If the attribute is not specified, then the effect is as if a value of 1 were specified.

- **diffuseConstant** – <*number*>

  kd in Phong lighting model. In SVG, this can be any non-negative number.

  If the attribute is not specified, then the effect is as if a value of 1 were specified.

- **kernelUnitLength** – <*number-optional-number*>

  The first number is the <*dx*> value. The second number is the <*dy*> value. If the <*dy*> value is not specified, it defaults to the same value as <*dx*>. Indicates the intended distance in current filter units (i.e., units as determined by the value of attribute **primitiveUnits**) between successive columns and rows, respectively, in the **kernelMatrix**. By specifying value(s) for **kernelUnitLength**, the kernel becomes defined in a scalable, abstract coordinate system. If **kernelUnitLength** is not specified, the default value is one pixel in the offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable. For some level of consistency across display media and user agents, it is necessary that a value be provided for at least one of **filterRes** and **kernelUnitLength**. In some implementations, the most consistent results and the fastest performance will be achieved if the pixel grid of the temporary offscreen images aligns with the pixel grid of the kernel. A negative or zero value is an error.

- **lighting-color** – `'currentColor'` | <*color*> [<*icccolor*>] | `'inherit'`

  The **lighting-color** property defines the color of the light source for filter primitives **feDiffuseLighting** and **feSpecularLighting**.

CHAPTER 55

feDisplacementMap Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feDisplacementMapElement

This filter primitive uses the pixels values from the image from **in2** to spatially displace the image from **in**.

This filter can have arbitrary non-localized effect on the input which might require substantial buffering in the processing pipeline. However with this formulation, any intermediate buffering needs can be determined by scale which represents the maximum range of displacement in either x or y.

When applying this filter, the source pixel location will often lie between several source pixels. In this case it is recommended that high quality viewers apply an interpolent on the surrounding pixels, for example bilinear or bicubic, rather than simply selecting the nearest source pixel. Depending on the speed of the available interpolents, this choice may be affected by the **image-rendering** property setting.

The **color-interpolation-filters** property only applies to the **in2** source image and does not apply to the **in** source image. The 'in' source image must remain in its current color space.

For common properties see: *Filter Primitives Overview*

55.1 SVG Attributes

- **in** – (see *in* attribute)
- **in2** – (see *in* attribute)

  The second input image, which is used to displace the pixels in the image from attribute **in**. This attribute can take on the same values as the **in** attribute.

- **scale** – <*number*>

  Displacement scale factor. The amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **filter** element.

  When the value of this attribute is `'0'`, this operation has no effect on the source image.

If the attribute is not specified, then the effect is as if a value of `'0'` were specified.

- **xChannelSelector** – `'R | G | B | A'`

  Indicates which channel from **in2** to use to displace the pixels in **in** along the x-axis. If attribute **xChannelSelector** is not specified, then the effect is as if a value of `'A'` were specified.

- **yChannelSelector** – `'R | G | B | A'`

  Indicates which channel from **in2** to use to displace the pixels in **in** along the y-axis. If attribute **yChannelSelector** is not specified, then the effect is as if a value of `'A'` were specified.

CHAPTER 56

feFlood Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feFloodElement

This filter primitive creates a rectangle filled with the color and opacity values from properties **flood-color** and **flood-opacity**. The rectangle is as large as the filter primitive subregion established by the **x**, **y**, **width** and **height** attributes on the **feFlood** element.

For common properties see: *Filter Primitives Overview*

56.1 SVG Attributes

- **flood-color** – `'currentColor'|`<*color*>` [`<*icccolor*>`]|' inherit'`

  initial value is `'black'`

- **flood-opacity** – <*opacity-value*>`|'inherit'`

  initial value is `'1'`

CHAPTER 57

feGaussianBlur Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feGaussianBlurElement

This filter primitive performs a Gaussian blur on the input image.

Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, `'SourceAlpha'`. The implementation may notice this and optimize the single channel case. If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

For common properties see: *Filter Primitives Overview*

57.1 SVG Attributes

- **in** – (see *in* attribute)
- **stdDeviation** – <*number-optional-number*>

  The standard deviation for the blur operation. If two <*number*>*s* are provided, the first number represents a standard deviation value along the x-axis of the coordinate system established by attribute **primitiveUnits** on the **filter** element. The second value represents a standard deviation in Y. If one number is provided, then that value is used for both X and Y.

  A negative value is an error. A value of zero disables the effect of the given filter primitive (i.e., the result is the filter input image).

  If the attribute is not specified, then the effect is as if a value of `'0'` were specified.

CHAPTER 58

feImage Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feImageElement

This filter primitive refers to a graphic external to this filter element, which is loaded or rendered into an RGBA raster and becomes the result of the filter primitive.

This filter primitive can refer to an external image or can be a reference to another piece of SVG. It produces an image similar to the built-in image source `'SourceGraphic'` except that the graphic comes from an external source.

If the **xlink:href** references a stand-alone image resource such as a JPEG, PNG or SVG file, then the image resource is rendered according to the behavior of the **image** element; otherwise, the referenced resource is rendered according to the behavior of the **use** element. In either case, the current user coordinate system depends on the value of attribute **primitiveUnits** on the **filter** element. The processing of the **preserveAspectRatio** attribute on the **feImage** element is identical to that of the **image** element.

When the referenced image must be resampled to match the device coordinate system, it is recommended that high quality viewers make use of appropriate interpolation techniques, for example bilinear or bicubic. Depending on the speed of the available interpolents, this choice may be affected by the **image-rendering** property setting.

For common properties see: *Filter Primitives Overview*

58.1 SVG Attributes

- **xlink:href** – *<iri>*

  A IRI reference to the image source.

- **preserveAspectRatio** – `'[defer] <align> [<meetOrSlice>]'`

  If attribute **preserveAspectRatio** is not specified, then the effect is as if a value of `'xMidYMid'` meet were specified.

CHAPTER 59

feMerge Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feMergeElement

This filter primitive composites input image layers on top of each other using the over operator with *Input1* (corresponding to the first **feMergeNode** child element) on the bottom and the last specified input, *InputN* (corresponding to the last **feMergeNode** child element), on top.

Many effects produce a number of intermediate layers in order to create the final output image. This filter allows us to collapse those into a single image. Although this could be done by using n-1 Composite-filters, it is more convenient to have this common operation available in this form, and offers the implementation some additional flexibility.

Each **feMerge** element can have any number of **feMergeNode** subelements, each of which has an **in** attribute.

The canonical implementation of feMerge is to render the entire effect into one RGBA layer, and then render the resulting layer on the output device. In certain cases (in particular if the output device itself is a continuous tone device), and since merging is associative, it might be a sufficient approximation to evaluate the effect one layer at a time and render each layer individually onto the output device bottom to top.

If the topmost image input is `'SourceGraphic'` and this **feMerge** is the last filter primitive in the filter, the implementation is encouraged to render the layers up to that point, and then render the SourceGraphic directly from its vector description on top.

For common properties see: *Filter Primitives Overview*

`Filter.`**`feMerge`**(*layernames*, ***extra*)

> **Parameters** **`layernames`** (*`list`*) – layernames as *strings*
>
> Create a **feMerge** filter, containing several **feMergeNode** subelements, with the input sources specified by **layernames**.

59.1 Methods

`feMerge.`**`feMergeNode`**(*layernames*)

Parameters **layernames** (*list*) – layernames as *strings*

Add several **feMergeNode** subelements, with the input sources specified by **layernames**.

CHAPTER 60

feMorphology Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feMorphologyElement

This filter primitive performs "fattening" or "thinning" of artwork. It is particularly useful for fattening or thinning an alpha channel.

The dilation (or erosion) kernel is a rectangle with a width of 2*x-radius and a height of 2*y-radius. In dilation, the output pixel is the individual component-wise maximum of the corresponding R,G,B,A values in the input image's kernel rectangle. In erosion, the output pixel is the individual component-wise minimum of the corresponding R,G,B,A values in the input image's kernel rectangle.

Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, `'SourceAlpha'`. In that case, the implementation might want to optimize the single channel case.

If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

Because **feMorphology** operates on premultipied color values, it will always result in color values less than or equal to the alpha channel.

For common properties see: *Filter Primitives Overview*

60.1 SVG Attributes

- **in** – (see *in* attribute)
- **operator** – `'erode | dilate'`

  A keyword indicating whether to erode (i.e., thin) or dilate (fatten) the source graphic. If attribute **operator** is not specified, then the effect is as if a value of `'erode'` were specified.

- **radius** – <*number-optional-number*>

  The radius (or radii) for the operation. If two <*number*>*s* are provided, the first number represents a x-radius and the second value represents a y-radius. If one number is provided, then that value is used

for both X and Y. The values are in the coordinate system established by attribute **primitiveUnits** on the **filter** element.

A negative value is an error. A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).

If the attribute is not specified, then the effect is as if a value of `'0'` were specified.

CHAPTER 61

feOffset Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feOffsetElement

This filter primitive offsets the input image relative to its current position in the image space by the specified vector.

This is important for effects like drop shadows.

When applying this filter, the destination location may be offset by a fraction of a pixel in device space. In this case a high quality viewer should make use of appropriate interpolation techniques, for example bilinear or bicubic. This is especially recommended for dynamic viewers where this interpolation provides visually smoother movement of images. For static viewers this is less of a concern. Close attention should be made to the **image-rendering** property setting to determine the authors intent.

For common properties see: *Filter Primitives Overview*

61.1 SVG Attributes

- **in** – (see *in* attribute)
- **dx** – <*number*>

  The amount to offset the input graphic along the x-axis. The offset amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **filter** element.

  If the attribute is not specified, then the effect is as if a value of `'0'` were specified.

- **dy** – <*number*>

  The amount to offset the input graphic along the y-axis. The offset amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **filter** element.

  If the attribute is not specified, then the effect is as if a value of `'0'` were specified.

CHAPTER 62

feSpecularLighting Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feSpecularLightingElement

This filter primitive lights a source graphic using the alpha channel as a bump map. The resulting image is an RGBA image based on the light color. The lighting calculation follows the standard specular component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map. The result of the lighting calculation is added. The filter primitive assumes that the viewer is at infinity in the z direction (i.e., the unit vector in the eye direction is (0,0,1) everywhere).

This filter primitive produces an image which contains the specular reflection part of the lighting calculation. Such a map is intended to be combined with a texture using the add term of the arithmetic **feComposite** method. Multiple light sources can be simulated by adding several of these light maps before applying it to the texture image.

Unlike the **feDiffuseLighting**, the **feSpecularLighting** filter produces a non-opaque image. This is due to the fact that the specular result is meant to be added to the textured image. The alpha channel of the result is the max of the color components, so that where the specular light is zero, no additional coverage is added to the image and a fully white highlight will add opacity.

The **feDiffuseLighting** and **feSpecularLighting** filters will often be applied together. An implementation may detect this and calculate both maps in one pass, instead of two.

For common properties see: *Filter Primitives Overview*

62.1 Methods

`feSpecularLighting.`**`feDistantLight`**(*azimuth=0*, *elevation=0*, ***extra*)

create and add a light source: *feDistantLight Filter Element*

`feSpecularLighting.`**`fePointLight`**(*source=(0, 0, 0)*, ***extra*)

Parameters **`source`** – source 3D point (**x**, **y**, **z**)

create and add a light source: *fePointLight Filter Element*

feSpecularLighting.**feSpotLight**(*source=(0, 0, 0)*, *target=(0, 0, 0)*, ***extra*)

Parameters

- **source** – source 3D point (**x**, **y**, **z**)
- **target** – target 3D point (**pointsAtX**, **pointsAtY**, **pointsAtZ**)

create and add a light source: *feSpotLight Filter Element*

62.2 SVG Attributes

- **in** – (see *in* attribute)
- **surfaceScale** – <*number*>

  height of surface when Ain = 1.

  If the attribute is not specified, then the effect is as if a value of `'1'` were specified.
- **specularConstant** – <*number*>

  ks in Phong lighting model. In SVG, this can be any non-negative number.

  If the attribute is not specified, then the effect is as if a value of `'1'` were specified.
- **specularExponent** – <*number*>

  Exponent for specular term, larger is more "shiny". Range 1.0 to 128.0.

  If the attribute is not specified, then the effect is as if a value of `'1'` were specified.
- **kernelUnitLength** – <*number-optional-number*>

  The first number is the <*dx*> value. The second number is the <*dy*> value. If the <*dy*> value is not specified, it defaults to the same value as <*dx*>. Indicates the intended distance in current filter units (i.e., units as determined by the value of attribute **primitiveUnits**) between successive columns and rows, respectively, in the **kernelMatrix**. By specifying value(s) for **kernelUnitLength**, the kernel becomes defined in a scalable, abstract coordinate system. If **kernelUnitLength** is not specified, the default value is one pixel in the offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable. For some level of consistency across display media and user agents, it is necessary that a value be provided for at least one of **filterRes** and **kernelUnitLength**. In some implementations, the most consistent results and the fastest performance will be achieved if the pixel grid of the temporary offscreen images aligns with the pixel grid of the kernel. A negative or zero value is an error.
- **lighting-color** – `'currentColor'` | <*color*> [<*icccolor*>] | `'inherit'`

  The **lighting-color** property defines the color of the light source for filter primitives **feDiffuseLighting** and **feSpecularLighting**.

CHAPTER 63

feTile Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feTileElement

This filter primitive fills a target rectangle with a repeated, tiled pattern of an input image. The target rectangle is as large as the filter primitive subregion established by the **x**, **y**, **width** and **height** attributes on the **feTile** element.

Typically, the input image has been defined with its own filter primitive subregion in order to define a reference tile. **feTile** replicates the reference tile in both X and Y to completely fill the target rectangle. The top/left corner of each given tile is at location (x+i*width,y+j*height), where (x,y) represents the top/left of the input image's filter primitive subregion, width and height represent the width and height of the input image's filter primitive subregion, and i and j can be any integer value. In most cases, the input image will have a smaller filter primitive subregion than the **feTile** in order to achieve a repeated pattern effect.

Implementers must take appropriate measures in constructing the tiled image to avoid artifacts between tiles, particularly in situations where the user to device transform includes shear and/or rotation. Unless care is taken, interpolation can lead to edge pixels in the tile having opacity values lower or higher than expected due to the interaction of painting adjacent tiles which each have partial overlap with particular pixels.

For common properties see: *Filter Primitives Overview*

63.1 SVG Attributes

- **in** – (see *in* attribute)

CHAPTER 64

feTurbulence Filter Element

See also:

http://www.w3.org/TR/SVG11/filters.html#feTurbulenceElement

This filter primitive creates an image using the Perlin turbulence function. It allows the synthesis of artificial textures like clouds or marble. For a detailed description the of the Perlin turbulence function, see "Texturing and Modeling", Ebert et al, AP Professional, 1994. The resulting image will fill the entire filter primitive subregion for this filter primitive.

For common properties see: *Filter Primitives Overview*

64.1 SVG Attributes

- **in** – (see *in* attribute)
- **baseFrequency** – <*number-optional-number*>

  The base frequency (frequencies) parameter(s) for the noise function. If two <*number*>*s* are provided, the first number represents a base frequency in the X direction and the second value represents a base frequency in the Y direction. If one number is provided, then that value is used for both X and Y.

  A negative value for base frequency is an error.

  If the attribute is not specified, then the effect is as if a value of `'0'` were specified.

- **numOctaves** – <*integer*>

  The numOctaves parameter for the noise function.

  If the attribute is not specified, then the effect is as if a value of `'1'` were specified.

- **seed** – <*number*>

  The starting number for the pseudo random number generator.

If the attribute is not specified, then the effect is as if a value of `'0'` were specified. When the seed number is handed over to the algorithm above it must first be truncated, i.e. rounded to the closest integer value towards zero.

- **stitchTiles** – `'stitch | noStitch'`
  - `'noStitch'` – no attempt it made to achieve smooth transitions at the border of tiles which contain a turbulence function. Sometimes the result will show clear discontinuities at the tile borders.
  - `'stitch'` – then the user agent will automatically adjust baseFrequency-x and baseFrequency-y values such that the **feTurbulence** node's width and height (i.e., the width and height of the current subregion) contains an integral number of the Perlin tile width and height for the first octave. The baseFrequency will be adjusted up or down depending on which way has the smallest relative (not absolute) change as follows: Given the frequency, calculate lowFreq=floor(width*frequency)/width and hiFreq=ceil(width*frequency)/width. If frequency/lowFreq < hiFreq/frequency then use lowFreq, else use hiFreq. While generating turbulence values, generate lattice vectors as normal for Perlin Noise, except for those lattice points that lie on the right or bottom edges of the active area (the size of the resulting tile). In those cases, copy the lattice vector from the opposite edge of the active area.

  If attribute **stitchTiles** is not specified, then the effect is as if a value of `'noStitch'` were specified.

- **type** – `'fractalNoise | turbulence'`

  Indicates whether the filter primitive should perform a noise or turbulence function. If attribute **type** is not specified, then the effect is as if a value of `'turbulence'` were specified.

CHAPTER 65

feDistantLight Filter Element

The light source **feDistantLight** is a child element of the filter primitives *feDiffuseLighting* or *feSpecularLighting*, create and add this object with the method `feDistantLight()` of the filter primitives **feDiffuseLighting** or **feSpecularLighting**.

See also:

http://www.w3.org/TR/SVG11/filters.html#feDistantLightElement

65.1 SVG Attributes

- **azimuth** – <*number*>

  Direction angle for the light source on the XY plane (clockwise), in degrees.

  Default is `'0'`

- **elevation** – <*number*>

  Direction angle for the light source on the YZ plane, in degrees.

  Default is `'0'`

CHAPTER 66

fePointLight Filter Element

The light source **fePointLight** is a child element of the filter primitives *feDiffuseLighting* or *feSpecularLighting*, create and add this object with the method `fePointLight()` of the filter primitives **feDiffuseLighting** or **feSpecularLighting**.

The light source **feDistantLight** is a child element of the filter primitives **feDiffuseLighting** or **feSpecularLighting**.

See also:

http://www.w3.org/TR/SVG11/filters.html#fePointLightElement

66.1 SVG Attributes

- **x** – <*number*> – **source** parameter

  X location for the light source in the coordinate system established by attribute **primitiveUnits** on the **filter** element.

  Default is `'0'`

- **y – <*number*> – source parameter** Y location for the light source in the coordinate system established by attribute **primitiveUnits** on the **filter** element.

  Default is `'0'`

- **z** – <*number*>– **source** parameter

  Z location for the light source in the coordinate system established by attribute **primitiveUnits** on the 'filter' element, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X and Y.

  Default is `'0'`

CHAPTER 67

feSpotLight Filter Element

The light source **feSpotLight** is a child element of the filter primitives *feDiffuseLighting* or *feSpecularLighting*, create and add this object with the method `feSpotLight()` of the filter primitives **feDiffuseLighting** or **feSpecularLighting**.

See also:

http://www.w3.org/TR/SVG11/filters.html#feSpotLightElement

67.1 SVG Attributes

- **x**, **y**, **z** – see *fePointLight Filter Element*
- **pointsAtX** – <*number*> – **target** parameter

  X location in the coordinate system established by attribute **primitiveUnits** on the **filter** element of the point at which the light source is pointing.

  Default is `'0'`
- **pointsAtY** – <*number*> – **target** parameter

  Y location in the coordinate system established by attribute **primitiveUnits** on the **filter** element of the point at which the light source is pointing.

  Default is `'0'`
- **pointsAtZ** – <*number*> – **target** parameter

  Z location in the coordinate system established by attribute **primitiveUnits** on the **filter** element of the point at which the light source is pointing, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X and Y.

  Default is `'0'`
- **specularExponent** – <*number*>

Exponent value controlling the focus for the light source.

Default is `'1'`

- **limitingConeAngle** – <*number*>

  A limiting cone which restricts the region where the light is projected. No light is projected outside the cone. **limitingConeAngle** represents the angle in degrees between the spot light axis (i.e. the axis between the light source and the point to which it is pointing at) and the spot light cone. User agents should apply a smoothing technique such as anti-aliasing at the boundary of the cone.

  If no value is specified, then no limiting cone will be applied.

CHAPTER 68

ViewBox Mixin

class svgwrite.mixins.**ViewBox**

The **ViewBox** mixin provides the ability to specify that a given set of graphics stretch to fit a particular container element.

The value of the **viewBox** attribute is a list of four numbers **min-x**, **min-y**, **width** and **height**, separated by whitespace and/or a comma, which specify a rectangle in **user space** which should be mapped to the bounds of the viewport established by the given element, taking into account attribute **preserveAspectRatio**.

ViewBox.**viewbox**(*minx=0*, *miny=0*, *width=0*, *height=0*)

Specify a rectangle in **user space** (no units allowed) which should be mapped to the bounds of the viewport established by the given element.

Parameters

- **minx** (*number*) – left border of the viewBox
- **miny** (*number*) – top border of the viewBox
- **width** (*number*) – width of the viewBox
- **height** (*number*) – height of the viewBox

ViewBox.**stretch**()

Stretch viewBox in x and y direction to fill viewport, does not preserve aspect ratio.

ViewBox.**fit**(*horiz='center'*, *vert='middle'*, *scale='meet'*)

Set the **preserveAspectRatio** attribute.

Parameters

- **horiz** (*string*) – horizontal alignment `'left | center | right'`
- **vert** (*string*) – vertical alignment `'top | middle | bottom'`
- **scale** (*string*) – scale method `'meet | slice'`

Scale methods	Description
`'meet'`	preserve aspect ration and zoom to limits of viewBox
`'slice'`	preserve aspect ration and viewBox touch viewport on all bounds, viewBox will extend beyond the bounds of the viewport

CHAPTER 69

Transform Mixin

class svgwrite.mixins.**Transform**

The **Transform** mixin operates on the **transform** attribute. The value of the **transform** attribute is a <*transform-list*>, which is defined as a list of transform definitions, which are applied in the order provided. The individual transform definitions are separated by whitespace and/or a comma. All coordinates are **user space coordinates**.

Transform.**translate**(*tx*, *ty=None*)

Specifies a translation by **tx** and **ty**. If **ty** is not provided, it is assumed to be zero.

Parameters

- **tx** (*number*) – user coordinate - no units allowed
- **ty** (*number*) – user coordinate - no units allowed

Transform.**rotate**(*angle*, *center=None*)

Specifies a rotation by **angle** degrees about a given point. If optional parameter **center** are not supplied, the rotate is about the origin of the current user coordinate system.

Parameters

- **angle** (*number*) – rotate-angle in degrees
- **center** (*2-tuple*) – rotate-center as user coordinate - no units allowed

Transform.**skewX**(*angle*)

Specifies a skew transformation along the x-axis.

Parameters **angle** (*number*) – skew-angle in degrees, no units allowed

Transform.**skewY**(*angle*)

Specifies a skew transformation along the y-axis.

Parameters **angle** (*number*) – skew-angle in degrees, no units allowed

Transform.**scale**(*sx*, *sy=None*)

Specifies a scale operation by **sx** and **sy**. If **sy** is not provided, it is assumed to be equal to **sx**.

Parameters

- **sx** (*number*) – scalar factor x-axis, no units allowed

- **sy** (*number*) – scalar factor y-axis, no units allowed

CHAPTER 70

XLink Mixin

class svgwrite.mixins.**XLink**
XLink mixin

XLink.**set_href**(*element*)
Create a reference to **element**.

Parameters **element** – if element is a *string* its the **id** name of the referenced element, if element is a **BaseElement** class the **id** SVG Attribute is used to create the reference.

XLink.**set_xlink**(*title=None*, *show=None*, *role=None*, *arcrole=None*)
Set XLink attributes (for *href* use *set_href()*).

Set **xlink:actuate** and **xlink:type** by the index operator:

```
element['xlink:type'] = 'simple'
element['xlink:actuate'] = 'onLoad'
```

CHAPTER 71

Presentation Mixin

class svgwrite.mixins.**Presentation**
Helper methods to set presentation attributes.

Presentation.**fill**(*color=None*, *rule=None*, *opacity=None*)
Set SVG Properties **fill**, **fill-rule** and **fill-opacity**.

See also:

- http://www.w3.org/TR/SVG11/painting.html#FillProperty
- http://www.w3.org/TR/SVG11/painting.html#FillRuleProperty
- http://www.w3.org/TR/SVG11/painting.html#FillOpacityProperty

Presentation.**stroke**(*color=None*, *width=None*, *opacity=None*, *linecap=None*, *linejoin=None*, *miterlimit=None*)
Set SVG Properties **stroke**, **stroke-width**, **stroke-opacity**, **stroke-linecap** and **stroke-miterlimit**.

See also:

- http://www.w3.org/TR/SVG11/painting.html#StrokeProperty
- http://www.w3.org/TR/SVG11/painting.html#StrokeWidthProperty
- http://www.w3.org/TR/SVG11/painting.html#StrokeOpacityProperty
- http://www.w3.org/TR/SVG11/painting.html#StrokeLinecapProperty
- http://www.w3.org/TR/SVG11/painting.html#StrokeMiterlimitProperty

Presentation.**dasharray**(*dasharray=None*, *offset=None*)
Set SVG Properties **stroke-dashoffset** and **stroke-dasharray**.

Where *dasharray* specify the lengths of alternating dashes and gaps as <list> of <int> or <float> values or a <string> of comma and/or white space separated <lengths> or <percentages>. (e.g. as <list> dasharray=[1, 0.5] or as <string> dasharray='1 0.5')

See also:

- http://www.w3.org/TR/SVG11/painting.html#StrokeDasharrayProperty

- http://www.w3.org/TR/SVG11/painting.html#StrokeDashoffsetProperty

CHAPTER 72

MediaGroup Mixin

SVG Tiny 1.2

valid for SVG Elements: animation, audio, desc, image, metadata, title, video

class svgwrite.mixins.**MediaGroup**
Helper methods to set media group attributes.

MediaGroup.**viewport_fill**(*color=None*, *opacity=None*)
Set SVG Properties **viewport-fill** and **viewport-fill-opacity**.

See also:

- http://www.w3.org/TR/SVGMobile12/painting.html#viewport-fill-property
- http://www.w3.org/TR/SVGMobile12/painting.html#viewport-fill-opacity-property

CHAPTER 73

Markers Mixin

class svgwrite.mixins.**Markers**
Helper methods to set marker attributes.

Markers.**set_markers**(*markers*)
Set markers for line elements (line, polygon, polyline, path) to values specified by *markers*.

- if *markers* is a 3-tuple:
  - attribute 'marker-start' = markers[0]
  - attribute 'marker-mid' = markers[1]
  - attribute 'marker-end' = markers[2]
- *markers* is a *string* or a *Marker* class:
  - attribute 'marker' = *FuncIRI* of markers

See also:

- http://www.w3.org/TR/SVG11/painting.html#MarkerProperty
- http://www.w3.org/TR/SVG11/painting.html#MarkerStartProperty
- http://www.w3.org/TR/SVG11/painting.html#MarkerMidProperty
- http://www.w3.org/TR/SVG11/painting.html#MarkerEndProperty

CHAPTER 74

Clipping Mixin

class svgwrite.mixins.**Clipping**

Clipping.**clip_rect**(*top='auto'*, *right='auto'*, *bottom='auto'*, *left='auto'*)
Set SVG Property **clip**.

See also:

http://www.w3.org/TR/SVG11/masking.html#OverflowAndClipProperties

CHAPTER 75

Inkscape Extension

This extension adds support for layers in inkscape. A layer in inkscape is an extended *group* container with additional *label* and *locked* attributes. Inkscape supports nested layers.

First import the inkscape extension:

```
import svgwrite
from svgwrite.extensions import Inkscape


dwg = svgwrite.Drawing('test-inkscape-extension.svg', profile='full', size=(640, 480))
```

You have to activate the extension for each drawing, because additional XML name spaces are required:

```
inkscape = Inkscape(dwg)
```

Create a new layer, all attributes that are supported by the *group* container are also allowed:

```
top_layer = inkscape.layer(label="Top LAYER 1", locked=True)
```

Add new layer as top level layer to the SVG drawing:

```
dwg.add(top_layer)
```

Create new elements and add them to a layer:

```
line = dwg.line((100, 100), (300, 100), stroke=svgwrite.rgb(10, 10, 16, '%'), stroke_
↪width=10)
top_layer.add(line)

text = dwg.text('Test', insert=(100, 100), font_size=100, fill='red')
top_layer.add(text)
```

Create another layer and add them as nested layer to “Top LAYER 1”:

```
nested_layer = inkscape.layer(label="Nested LAYER 2", locked=False)
top_layer.add(nested_layer)

text = dwg.text('Test2', insert=(100, 200), font_size=100, fill='blue')
nested_layer.add(text)

dwg.save()
```

CHAPTER 76

Indices and tables

- genindex
- modindex
- search

CHAPTER 77

Document License

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

P