

ReportLab Graphics and SVG Integration

Programming and reusing
production quality vector graphics

Status report of work in progress

Dinu C. Gherman, dinu@reportlab.com

European Python and Zope Conference

Charleroi, Belgium, June 2002



Overview

- Motivation
- ReportLab Graphics
- Scalable Vector Graphics
- Bridging RLG and SVG
- Sample code and output
- Sample uses in applications
- Roadmap
- References



Motivation

- Main goal: create and reuse SVG data
- Reuse vector graphics in ReportLab documents
- Create charts/maps dynamically from data
- Reuse manually created "clip art"
- Use SVG-aware editors, e.g. Illustrator, Sketch
- Use XML-aware editors, e.g. OmniGraffle, Create
- Print "skeleton" SVG files (Mark Millikan)

RLG - ReportLab Graphics

- Simple design
- Easy to use
- Programmatic API
- Pure Python (but PIL/libart for bitmaps)
- Abstract vector graphics
- Concrete renderers
- Attribute validation
- Shapes, widgets, properties, transforms
- Lower-left origin, Y-axis points *up*

Shapes

- Rect, Circle, Ellipse, Line, Polygon, PolyLine, Path, String, (Image)
- Group, Drawing (containers)
- Sample:

```
from reportlab.graphics import shapes  
from reportlab.lib import colors
```

```
rect = shapes.Rect(0, 0, 100, 50)  
rect.strokeColor = colors.red  
rect.fillColor = colors.blue
```

Properties

- Provide high configurability
- Allow low user-level OO-knowledge
- Intended use by setting properties
- Subclassing for non-standard use
- Runtime attribute validation:

```
>>> from reportlab.graphics import shapes
>>> from reportlab.lib import colors
>>> rect = shapes.Rect(0, 0, 100, 50)
>>> rect.fillColor = colors.red
>>> rect.filColor = colors.red    ### typo!
Traceback (most recent call last):
  [...]
AttributeError: Illegal attribute 'filColor' in class Rect
>>> rect.fillColor = 42    ### wrong type!
Traceback (most recent call last):
  [...]
AttributeError: Illegal assignment of '42' to 'fillColor'
in class Rect ]]>
```

Widgets

- Collections of customised reusable shapes
- Domain-specific graphics libraries
- Sample:

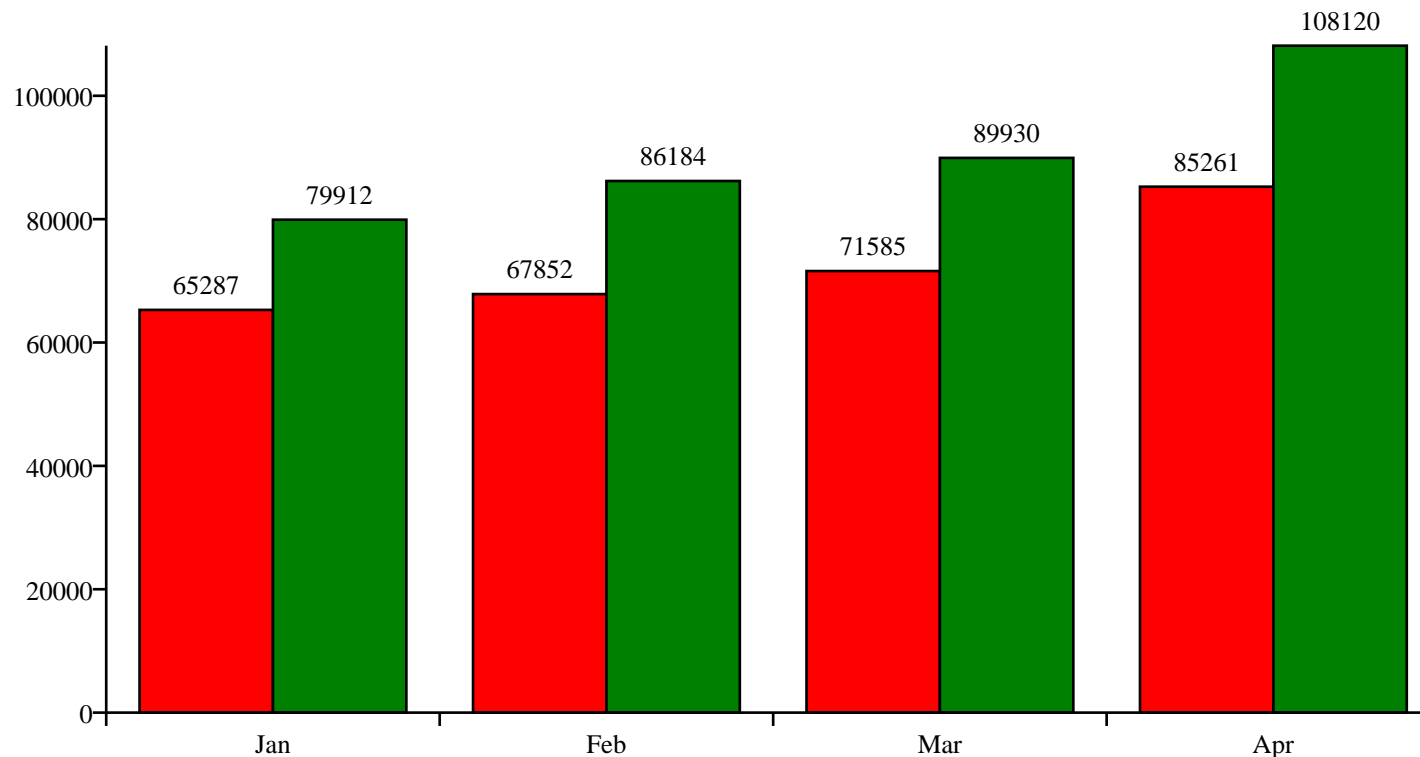
```
from reportlab.graphics.widgets \  
    import signsandsymbols as sas  
from reportlab.lib.colors \  
    import chartreuse
```

```
smiley = sas.SmileyFace()  
smiley.fillColor = chartreuse  
smiley.size = 100
```



Widget sample: bar chart

- HTTP access statistics for reportlab.com
- Mandatory: "base" properties (data, overall size)
- Optional: "additional" properties (range, step, ...)
- Optional: "style" properties (colors, fonts, sizes, ...)



Widget sample: bar chart cont'd

```
from reportlab.graphics.charts.barcharts \
    import VerticalBarChart

bc = VerticalBarChart()
bc.x, bc.y = 0, 0
bc.height, bc.width = 250, 500
# bc.barWidth = 20
# bc.barSpacing = 3
# bc.groupSpacing = 10
bc.data = data
bc.strokeColor = None
bc.barLabelFormat = '%d'
bc.barLabels.nudge = 10
bc.categoryAxis.categoryNames = cats
# bc.categoryAxis.labels.boxAnchor = 'n'
```

Renderers

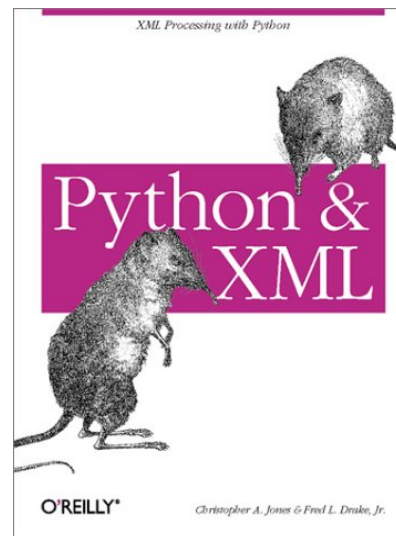
- Output filters for RLG
- Need to handle only shapes
- Sample code:

```
from reportlab.graphics import renderPDF
from reportlab.graphics import renderPS
from reportlab.graphics import renderPM

d = makeDrawing()
renderPS.drawToFile(d, "drawing.eps")
renderPDF.drawToFile(d, "drawing.pdf")
renderPM.drawToFile(d, "drawing.jpg", "JPG")
```

SVG - Scalable Vector Graphics

- W3C recommendation ("standard")
- Current version: 1.0 (new: 1.1 for "mobile gadgets")
- Vector graphics for the internet
- Based on XML, XSL, CSS, XLink, ...
- Competes with Flash MX, MS VML (does it?)
- Recommended reading (see also: References):



SVG Features

- Complex design
- Basic shapes
- Gradients and patterns
- Text and fonts
- Raster effects ("filters")
- Clipping, masking, compositing
- Animation
- Interaction
- Hyperlinking
- Scripting (JavaScript)
- Groups with "inherited" attributes
- Transforms
- Upper-left origin, Y-axis points *down*

Hello World in SVG

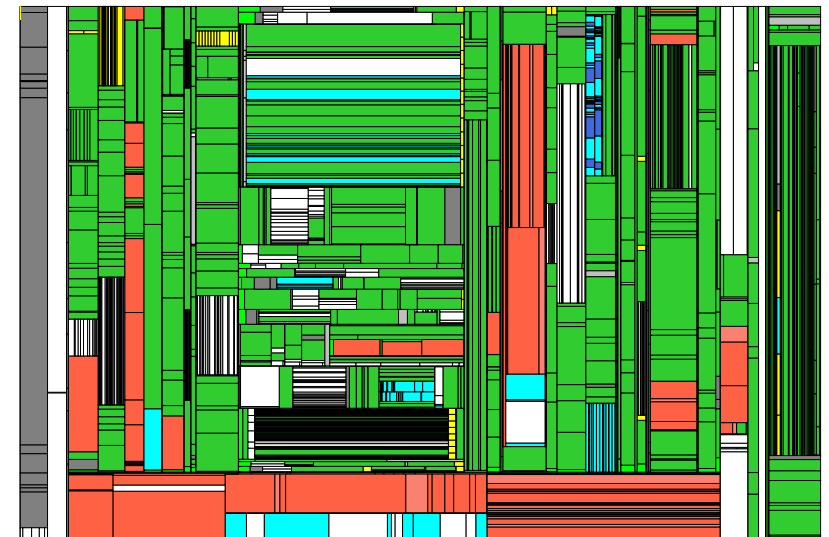
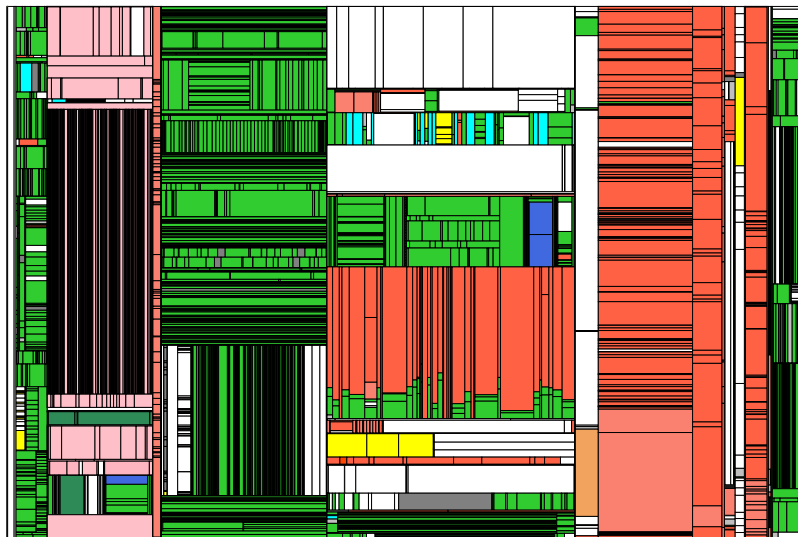
```
<?xml version="1.0" encoding="iso-8859-1"?>  
  
<svg width="400" height="200">  
  
  <rect x="0" y="0" width="400" height="200"  
    stroke-width="5"  
    stroke="black"  
    fill="aqua"/>  
  
  <text x="50" y="50"  
    fill="blue"  
    font-size="36"  
    font-family="Helvetica">Hello World</text>  
  
</svg>
```



Hello World

RLG to SVG

- Conceptually simple
- Implemented as module renderSVG
- Uses Python 2.x DOM (minidom), on purpose!
- Basically converts shapes and attributes
- Handles some differences between RLG and SVG
- Sample tree-maps of two major Python code bases (?):



SVG to RLG

- Conceptually challenging
- Will always handle only an SVG subset
- RLG groups have no properties
- Uses 2.x minidom (backlinks)
- Resolves inherited SVG attributes via DOM backlinks
- Could also use just SAX, (more work)
- Might use PyRXP later for speed (no backlinks, yet)
- Packaging as reportlab module not yet clear

Sample code: SVG to PNG "tool"

```
#!/usr/bin/env python

"svg2png.py"

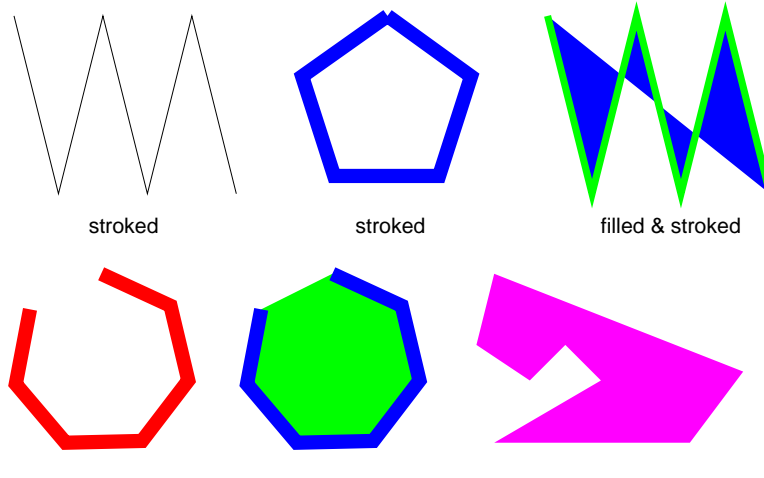
import sys
from os.path import splitext
from reportlab.graphics import renderPM
from svglib import svg2rlg

inPath = sys.argv[1]
drawing = svg2rlg(inPath)
outPath = splitext(inPath)[0] + '.png'
renderPM.drawToFile(drawing, outPath, 'PNG')
```

Sample PDF results (SVG test suite)

- Polylines
- Quadratic Bézier curves

Basic polylines.

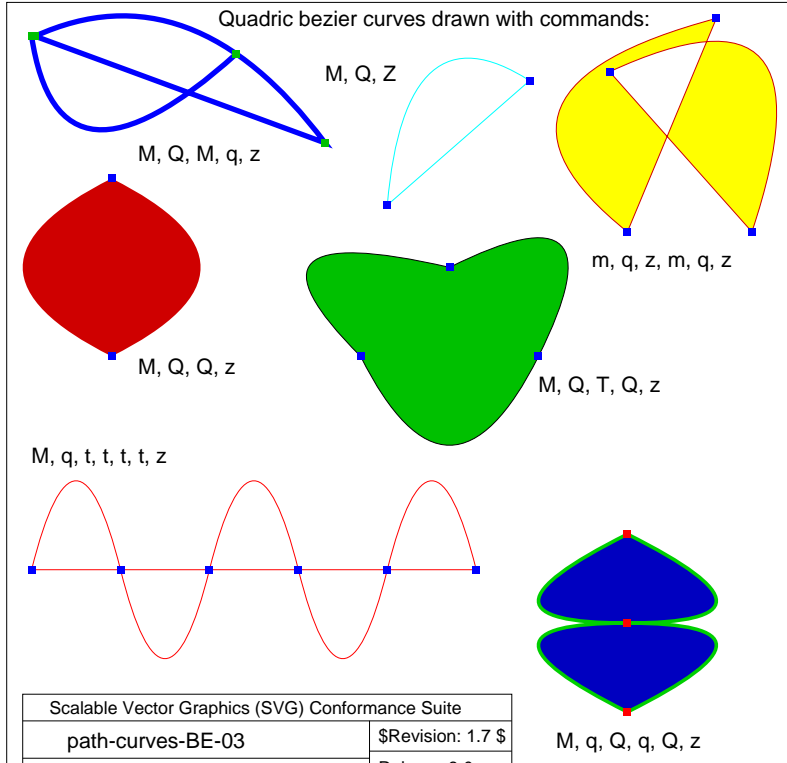


stroked stroked filled & stroked

stroked filled & stroked filled

Scalable Vector Graphics (SVG) Conformance Suite	
shapes-polyline-BE-06	\$Revision: 1.6 \$
Copyright 2000 W3C. All Rights Reserved.	Release 3.0

Quadratic bezier curves drawn with commands:



M, Q, Z

M, Q, M, q, z

m, q, z, m, q, z

M, Q, Q, z

M, Q, T, Q, z

M, q, t, t, t, t, z

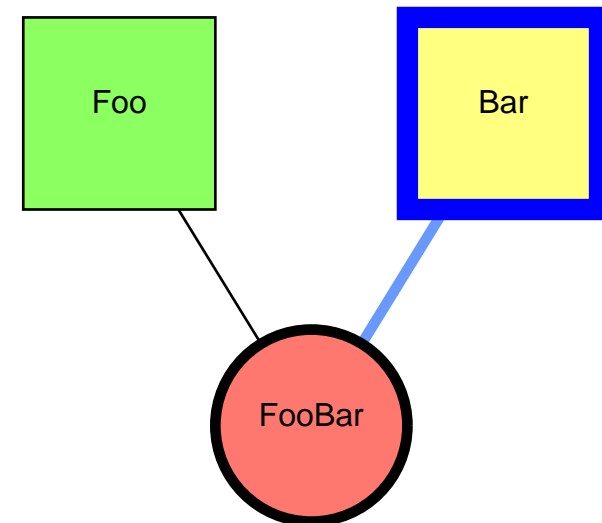
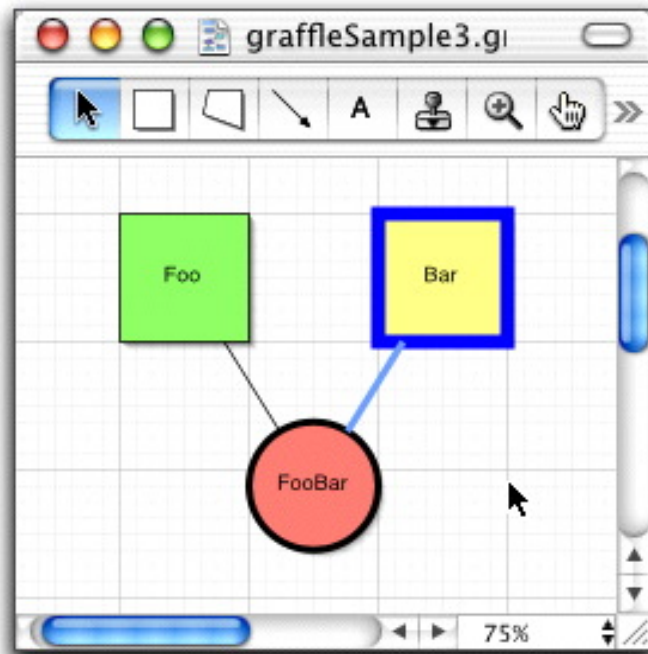
M, q, Q, q, Q, z

Scalable Vector Graphics (SVG) Conformance Suite	
path-curves-BE-03	\$Revision: 1.7 \$
Copyright 2000 W3C. All Rights Reserved.	Release 3.0

Applications

OmniGraffle

- Writes XML (Apple's PropertyList dialect)
- Very simple structure, undocumented semantics
- Received two Apple design awards in 2002



OmniGraffle

- PropertyList DTD:

```
<!ENTITY %plistObject
          "(array | data | date | dict | real |
           integer | string | true | false )">
<!ELEMENT plist      %plistObject;>
<!ATTLIST plist      version CDATA "0.9">

<!-- Collections -->
<!ELEMENT array      (%plistObject;)*>
<!ELEMENT dict       (key, %plistObject;)*>
<!ELEMENT key        (#PCDATA)>

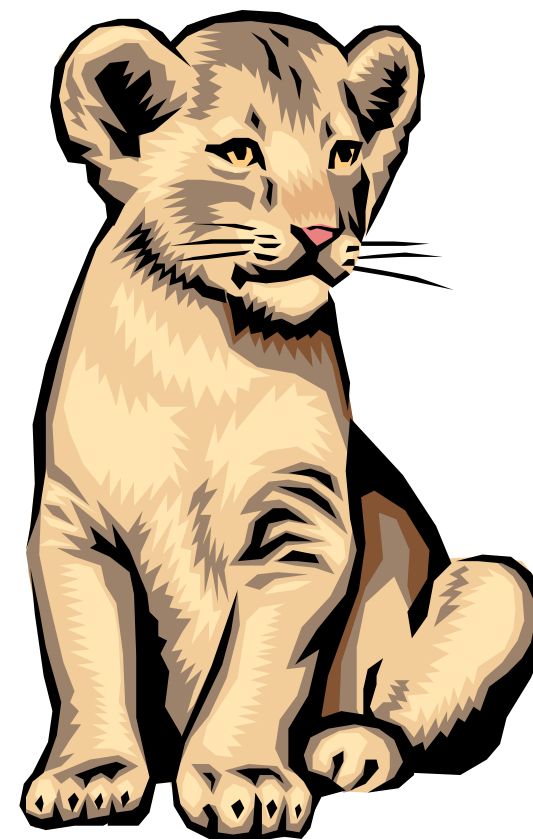
<!-- Primitive types -->
<!ELEMENT string     (#PCDATA)>
<!ELEMENT data       (#PCDATA)>
<!ELEMENT date       (#PCDATA)>
<!ELEMENT true       EMPTY>
<!ELEMENT false      EMPTY>

<!-- Numerical primitives -->
<!ELEMENT real       (#PCDATA)>
<!ELEMENT integer    (#PCDATA)>
```

PythonPoint

- Pure Python presentation application
- Creates presentations like this one
- Converts PythonPoint XML to PDF
- Part of Open Source ReportLab toolkit
- Customised parser, maybe soon PyRXP
- SVG import in PythonPoint XML code:

```
<customshape  
  module="custom"  
  class="SvgDrawing"  
  initargs="(0,0,100,None,'lion.svg')"/>
```



Wrapper code for PythonPoint

```
import svglib
```

```
class SvgDrawing:
```

```
    "SVG Wrapper for PythonPoint."
```

```
def __init__(self, x,y,width,height,path):
```

```
    assert (width,height) != (None,None)
```

```
    self.x = x
```

```
    self.y = y
```

```
    self.width = width
```

```
    self.height = height
```

```
    self.drawing = svglib.svg2rlg(path)
```

Wrapper code for PythonPoint (cont'd)

```
def drawOn(self, canvas):
    d = self.drawing
    x, y = self.x, self.y
    w, h = self.width, self.height

    # maintain aspect ratio
    if h == None:
        h = w * d.height/d.width
    elif w == None:
        w = h * d.width/d.height

    d.scale(w/d.width, h/d.height)
    d.drawOn(canvas, x, y)
```

Roadmap

Done (hopefully)

- Shapes (except cubic Bézier paths) and simple text
- Groups, transforms

Must-Do

- Fix bugs
- Add more tests
- Support more attributes
- Improve error reporting
- Add RLG support (Image, Path additions)
- Hope for Jython XML bug fixes (`minidom.parse()`)

Should-Do

- Hyperlinking, Stylesheets, Clipping, Markers

Could-Do

- SVG output from PythonPoint
- Gradients, Transparency, Unicode
- General-purpose mapping between PDF and SVG

Don't-Do

- Use other backends like Tkinter, wxPython(?)
- Interaction, animation (other backends? GIFs?)
- Filters (PIL?!)
- "Real" extended SVG objects (maybe later)

References

- **Dinu Gherman's svglib**,
<http://python.net/~gherman/#svglib>
- **Raph Levien's SVG/Gnome pages/samples**,
<http://www.levien.com/svg>
- **ReportLab**, <http://www.reportlab.com>
- **Adobe**, <http://www.adobe.com/svg/>
- **W3C**, <http://www.w3.org/Graphics/SVG/>
- **OmniGroup**,
<http://www.omnigroup.com/applications/omnigraffle/>
- **SVG Essentials** by J. David Eisenberg,
<http://www.oreilly.com/catalog/svgess/>
- **Python and XML** by Christopher A. Jones and Fred L. Drake, Jr., <http://www.oreilly.com/catalog/pythonxml/>