SVG Graphics

in the PostgreSQL Documentation

A proposal how to generate SVG files and handle them in the long-term

Purpose and state of this document: The document is part of the discussions around the efforts to enrich the PostgreSQL documentation with graphical elements. Because this discussions are not yet finished, we expect to see modifications and extensions over time. During the realization phase, we must integrate some of its parts into the regular documentation and/or into the wiki.

Date: November 21, 2018

Editor: Jürgen Purtz (juergen@purtz.de)

Actual Situation

Since about 2010 we have seen efforts to enrich the PostgreSQL documentation with graphical elements. Unfortunately until today we haven't reached a consensus about tools and long-term handling of the source of such graphics.

But most of the contributors agree to use SVG as the source language. SVG offers great advantages in general:

- It is standardized and this process evolves.
- It is implemented in most browsers and other viewers.
- It renders to devices of different physical sizes (as its name suggests).
- It is available on a wide range of operation systems. But despite of the standardisation the implementations renders in may cases to different results for the elaborated features.
- There are many tools which generate SVG in a comfortable way.
- SVG is integrated into Docbook.

Additionally, we can easily integrate SVG files into our tool chain of generating the documentation. Since version 11 our documentation source code is in XML format and the tools are able to handle SVG which is referenced by such files.

In the following we presume that our graphics will be generated out of SVG. The open questions are:

- How to generate SVG?
- How to handle SVG files in the long-term?
- Shall we avoid some parts of SVG to achieve portability?

Tools: Problems and Requirements

During the discussions people proposed the use of different tools. At the end we had a huge bunch of all well known and some nice products. You can divide them into two groups. A: generate SVG out of interactive actions (Inkscape, svgedit, …) and B: generate SVG out of another source language (ascii2svg, markdeep, dot, ...).

Problems with A:

- The generated SVG is extremely verbose and unreadable.
- They tend to integrate ‘high level’ SVG features with the result, that portability suffers.

Problems with B:

- In most cases they are designed for special purposes like UML diagrams or graphs. The huge potentials of SVG get lost.

Our requirements in respect to the resulting SVG are:

- short, compact, easy readable
- diff-able
- conforming to SVG 1.1
- restriction to such features, which are portable across most viewers and platforms
- homogeneity across all graphics by honouring a style guide (fonts, colors, sizes, ...)

Proposed Proceeding

The actual ‘tool zoo’ consists of a huge bunch of products without a clear favourite in our community. And it’s possible that within the next years new products will overrun the old ones. Our lengthy discussion has shown that it’s likely that we will reach no agreement in a manageable future concerning a SVG tool. Therefore we shall avoid a recommendation in favour of a certain product. Everybody shall use the tool of his choice – but has to deliver a clean SVG file. It’s our responsibility to describe and install methods which guarantees that all requirements for the delivered SVG are meet. The tools their-selves as well as their proprietary representation of the graphical objects are out of scope of our community. We shall focus ourselves to the pure SVG.

Some of the tools are able to generate simple SVG without loosing their powerful additional features. For each such tool we need a description how to do this.

An example: Inkscape generates by default SVG files in the format ‘Inkscape SVG’. In this format all listed disadvantages occurs. But it’s possible to serialize the graphical objects into a ‘Plain SVG’ or an ‘Optimized SVG’ format. If one parametrises Inkscape in a certain way, ‘Optimized SVG’ meets most of our requirements. In this case elaborated concepts like connectors or groups get lost. But the author has the chance to store two files, one in ‘Inkscape SVG’ for his further work and one in ‘Optimized SVG’ format for the community. The author doesn’t loose any information or feature and the community gets a small but working SVG file which renders in the same way as the big one.

This small SVG file becomes part of our documentation’s source, optionally the big one is also stored somewhere in our repository. The small one is authoritative for all further actions: generating documentation, git, diff, further editing with any other tool. If - at a later stage - someone uses the same tool as the original author for a modification of the graphic, he may start with the big file. But only the then generated new small one is used within our system.

In a second step we shall externalize all common elements like style information (font, size, color, …) or complex SVG elements (UML elements like an actor, diverse arrow symbols, discs, ...) into separate files to guarantee their homogeneous usage.

How to create SVG for the PostgreSQL Documentation

Using Inkscape

Inkscape supports different file formats to serialize its graphics. ‘Inkscape SVG’ is the product specific format which stores the complete information about the actual graphic. Therefore it creates a huge, non-portable file. In contrast ‘Optimized SVG’ creates a small, incomplete file containing only that information parts which are necessary for final rendering. This format even performs optimizations by collecting common definitions into higher level elements. The PostgreSQL documentation uses this format (naturally everybody is free to use and store for one graphic both formats in different files).

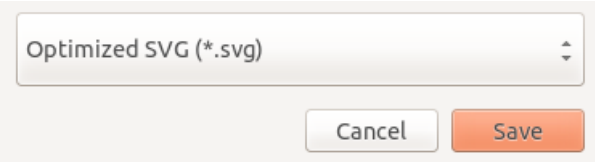
There are two sets of parameters which influences the generation of SVG files.

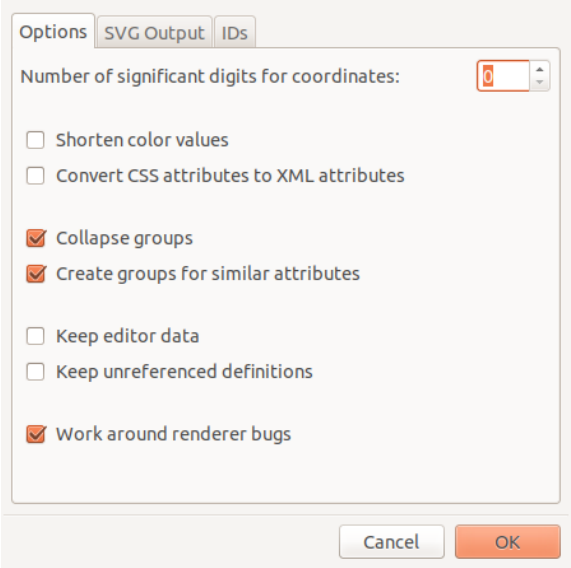
First, you can choose diverse parameters concerning the actual graphic, e.g. it’s size, default measurement in px, pt, em or something else, scaling factor, metadata, grid on/off, and many more. To support a consistent usage of those parameters across our graphics, they can be stored at a central place. Whenever you create a new graphic, they are read from `$HOME/.config/inkscape/templates/default.svg`. (Technically this is the default template for all new graphics. If you want to keep the previous behaviour for other tasks, he shall create a file postgres.svg in the template directory and create new files with the command: `File / New from Template / postgres`.) Here is the recommended content:

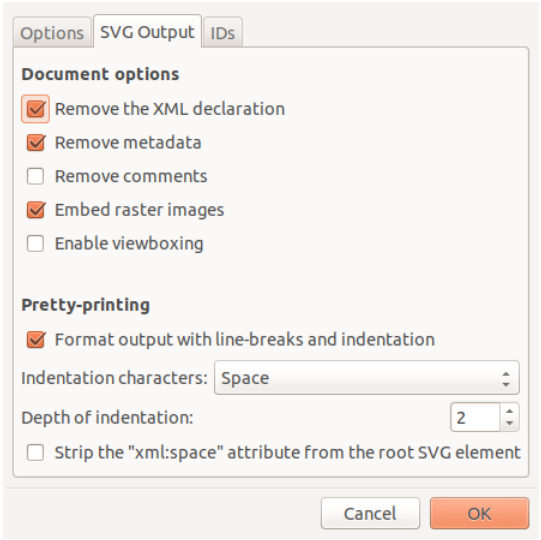
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
  xmlns="http://www.w3.org/2000/svg"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
  xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cc="http://web.resource.org/cc/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  version="1.1"
  width="580"
  height="280"
  viewBox="0 0 580 280">
  <defs id="defs1" />

  <sodipodi:namedview
    id="base"
    pagecolor="#ffffff"
    bordercolor="#666666"
    borderopacity="1.0"
```

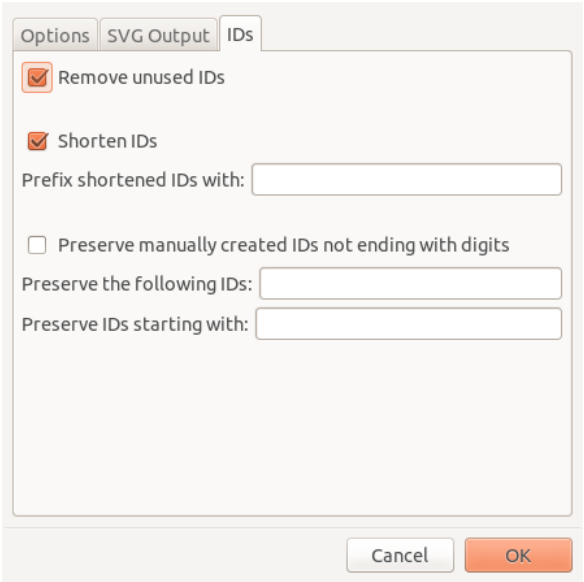
```
    borderlayer="false"
    showborder="true"
    showgrid="true"
    inkscape:document-units="px"
    inkscape:pageopacity="0.0"
    inkscape:pageshadow="2"
    inkscape:showpageshadow="false"
    inkscape:cx="290"
    inkscape:cy="140"
    inkscape:zoom="1.5"
    inkscape:window-maximized="false" >
    <inkscape:grid type="xygrid" id="grid1" />
  </sodipodi:namedview>

  <metadata>
    <rdf:RDF>
      <cc:Work rdf:about="">
        <dc:format>image/svg+xml</dc:format>
        <dc:type rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
      </cc:Work>
    </rdf:RDF>
  </metadata>

  <!-- border and background surrounding the complete graphic -->
  <rect x="1" y="1" width="99.4%" height="99.4%" rx="1%"
        fill="rgb(232, 247, 230)" stroke="rgb(132, 197, 120)"/>

</svg>
```

Second, you can choose how to generate the 'Optimized SVG' format. Select the options after `File / Save As` as shown below:









Hints:

- Use grid! This leads to simple integer values for positions: `Edit / Properties / Interface / Grids / Rectangular Grid / Major grid line every 5 px` (restart necessary).
- Use the same values for `viewbox` and `viewport (=width/height)`. This helps to to understand the plain SVG File. The `viewbox` may be scaled easily by subsequent processes like fop to adopt all sizes in a consistent way.
- By default newly created documents contain a border (rectangle with a grey background) as a clear separator between pure text, tables, and graphic. Shift it away from the page and reposition it to its original place after you have finished your work. This helps to select other objects. Don't use position 0/0 or a size of 100% for this rectangle because it leads to invisibility of the right/left and/or top/bottom lines in some other rendering systems.
- Control the generated 'Optimized SVG' file.
  - Sometimes there are a lot of identical and therefore unnecessary `<marker>` elements (Inkscape 0.92). Purge them manually.
  - Sometimes there is a `scale(0)` attribute which leads to invisibility of its element. Change it to something like `scale(0.5)`.
  - Within `<rect>` there are sometimes `rx="0"` or `ry="0"` attributes. They have no effect, you can safely remove them.
  - Within the `style` attribute there are sometimes `stroke-width:1px` strings. They have no effect because it is the default, you can safely remove them.
  - It is good practise to rearrange the SVG elements in accordance to a logical sequence. The sequence created by Inkscape follows the order in which you had inserted the elements during your interactive work. But be careful: The order determines the z-direction (later elements hide previous elements if they render to the same x/y position).

Using Xfig

ToDo

Using svgedit

ToDo

Using Gimp

ToDo

Using LibreOffice Draw

ToDo

Using Dia (unsupported since 2011?)

ToDo

Style Guide for SVG Graphics within PostgreSQL's Documentation

SVG files must comply to SVG 1.1. . This can be assured by tests at the site: https://validator.w3.org/

Voluntarily avoided SVG elements

Suspect elements [todo]

Forbidden elements [todo]

More SVG definitions

The preferred dimensions for measurements are **px**. Solely use numbers without fraction (integers). If you use **em**, the size of characters are rendered differently in HTML and PDF.

Definitions concerning the complete graphic [todo]

size, background, border, ...

Fonts & sizes [todo – in accordance with the existing HTML and PDF generation]

which font-family: 'Open Sans', verdana, sans-serif, ... ???

Colors [todo]

which colors?

color-syntax: rgb(255, 255, 255) or #ffffff or color names ?

One common Style Guide for HTML, PDF and SVG?

Attributes for element <imagedata> (Docbook)

width/depth in % (= the new viewport) defines a size relative to the browsers actual window size. When zooming into or out of the HTML page, the absolute size of the SVG graphic and its elements keeps unchanged whereas the absolute size of the surrounding elements like HTML text change.

contentwidth/contentdepth in % (= the new viewbox) and scale define a resizing of the original SVG graphic. When zooming into or out of the HTML page, the SVG graphic changes in the same ratio as the surrounding elements. This leads to a consistent visual effect, but: character sizes differs from the size of the surrounding characters (if you choose values different from 100%).

Problems

- There is a conceptual problem with single-page HTML output. The SVG files keep separate and actually we don't know how to include them into the generated single file `postgres.html`. This is independent form the decision to use a separate directory for SVG files. The problem will also arise if docbook files and SVG files reside in the same directory.

  As long as someone refers to the original file in the sgml directory the problem is not seen because the (external) svg files can be reached. It occurs, if he moves the file to a different location or download it without a correlate handling of the svg files in the svg directory.
-

Open Issues

- What font-family shall we use? (‘Open Sans’, verdana, sans-serif, ...)
-
-

What’s next?

- Externalize font definitions
- Externalize arrows
- Create often used complex objects
-