Sgvizler: A JavaScript Wrapper for Easy Visualization of SPARQL Result Sets

Martin G. Skjæveland(✉)

Department of Informatics, University of Oslo, Oslo, Norway
martige@ifi.uio.no

Abstract. Sgvizler is a small JavaScript wrapper for visualization of SPARQL results sets. It integrates well with HTML web pages by letting the user specify SPARQL SELECT queries directly into designated HTML elements, which are rendered to contain the specified visualization type on page load or on function call. Sgvizler supports a vast number of visualization types, most notably all of the major charts available in the Google Chart Tools, but also by allowing users to easily modify and extend the set of rendering functions, e.g., specified using direct DOM manipulation or external JavaScript visualization tool-kits. Sgvizler is compatible with all modern web browsers.

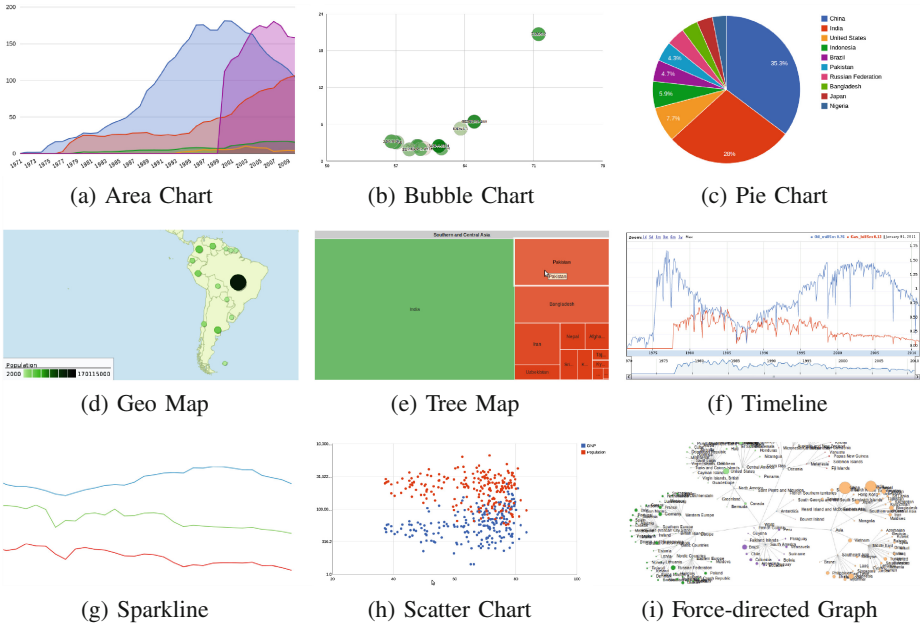
1 Introduction

The prominent way of displaying data residing in a SPARQL endpoint to a human user is to list the results of a DESCRIBE query for a given resource in a table. While tables are good for rendering heterogeneous data, needless to say, they do not convey the meaning of, e.g., geographical, temporal or quantitative data as well as respectively maps, timelines or different charts, such as pie charts and bar charts, do. There are different approaches to leverage this problem: The Data-gov Wiki project uses XSLT to convert the XML result of SPARQL SELECT queries into a format edible by the Query interface of the Google Chart Tools [1], which is used to render the data into charts and maps [2]. SPARQL Web Pages [3] uses a special-purpose vocabulary to specify how web pages are to be built from SPARQL query instructions. Spark [4] is a JavaScript library for processing SPARQL SELECT queries put directly into the HTML markup, which may be rendered into a list, a table, a line chart or a pie chart, or into other representations using custom-made visualization functions. Sgvizler [5], the topic of this paper, is a JavaScript wrapper of SPARQL result set visualization combining many of the ideas of the above-mentioned approaches into a powerful, easy-to-use and cross-browser visualization tool.

2 Overview of Sgvizler

Sgvizler is a JavaScript wrapper of SPARQL result set visualization, like Spark. What makes Sgvizler special is the ease with which it lets one integrate the visualization of SPARQL SELECT query result sets directly into web pages, *combined*

E. Simperl et al. (Eds.): ESWC 2012 Satellite Events, LNCS 7540, pp. 361–365, 2015.
DOI: 10.1007/978-3-662-46641-4_27



(a) Area Chart (b) Bubble Chart (c) Pie Chart

(d) Geo Map (e) Tree Map (f) Timeline

(g) Sparkline (h) Scatter Chart (i) Force-directed Graph

Fig. 1. Sgvizler chart examples.

with the large number of visualization types it supports and its compatibility of different origin SPARQL endpoint querying for all major modern browsers and most SPARQL endpoints. In addition, Sgvizler is built to be easily extensible by having a clear and simple API for adding user-defined rendering functions. A few samples of the available visualization types are found in Figs. 1 and 2, and Example 1 shows how simply visualizations can be added into web pages by the use of Sgvizler.

Example 1. The snippet below shows the source code of an HTML element which renders into the chart in Fig. 2 on page load. Input data to Sgvizler is set using HTML5 compatible `data-` prefixed attributes. The endpoint address and SPARQL query is given by the attributes `data-sgvizler-endpoint` and `data-sgvizler-query`,[1] respectively. The chart type to draw (`sMap`) is specified with `data-sgvizler-chart`, and additional options to the rendering function is listed in `data-sgvizler-chart-options`. The output format of the SPARQL endpoint is specified with `data-sgvizler-format` (permissible values are `xml`, `json`, `jsonp`) and `data-sgvizler-loglevel="2"` sets the level of feedback given to the user while preparing the chart.

```
1  <div id="id1" data-sgvizler-endpoint="http://dbpedia.org/sparql"
2        data-sgvizler-query="SELECT ?lat ?long ?name ?text ?url ?image
3         { ?url a dbpo:AdministrativeRegion ; dct:subject dbp:Rogaland ;
```

[1] The query is simplified to save space; this and other live examples are found at [5].

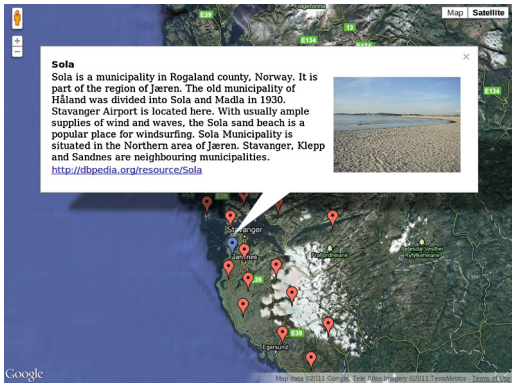


Fig. 2. An `sMap` of the municipalities in Rogaland, Norway—with a fact bubble about Sola.

```
4               rdfs:label ?name ; geo:lat ?lat ; geo:long ?long .
5               rdfs:comment ?text ; dbpo:thumbnail ?image }"
6       data-sgvizler-chart="sMap"
7       data-sgvizler-chart-options="dataMode=markers|showTip=true"
8       data-sgvizler-format="jsonp"
9       data-sgvizler-loglevel="2"
10      style="width: 800px; height: 600px;"></div>
```

Sgvizler is lightweight: 31 KB in minified condition and 6 KB minified and gzipped. It relies on external libraries for visualization, endpoint communication and DOM manipulation. Currently, the following chart types and rendering functions are available: quantitative data charts (line chart, area chart, column chart, bar chart, bubble chart, scatter chart, sparkline, pie chart, candlestick chart, motion chart, gauge), hierarchical data charts (tree map, org chart), geographical visualizations (maps, geo chart, geo map), graphs, lists, tables and a generic text rendering function. It is released under an MIT License, and the complete source code, documentation, examples, issues are available at [5].

2.1 How Does It Work?

There are three intended ways of using Sgvizler: adding queries directly into HTML markup—as in Example 1, issuing a query and visualization options using an HTML form, or by direct JavaScript function call. For this explanation we will assume the first. On page load all HTML elements designated for Sgvizler visualization are collected and processed asynchronously. Each query is sent, together with the format the query results should be returned as—either XML or JSON, to the specified endpoint using jQuery's `ajax()` function [6]. The returned result set, expected by Sgvizler to be of one of the formats described by W3C [7,8], is parsed into a Google DataTable object. The DataTable object is then paired with drawing options and passed to the specified rendering function which fills the HTML element where the query was collected with its visualization results.

The DataTable class serves as input parameter type for all of the chart functions in Google's Chart Tools. This means that all these charts are readily available for Sgvizler visualization. Additionally, Sgvizler is designed such that any user-defined rendering functions must take a DataTable object as input. This makes it possible to use the same code for handling all visualization functions and easy to register new functions. Also, a DataTable object is equipped with many convenient functions for editing and querying its contents,[2] making it a helpful object to render into new representations.

The `sMap` function used in Example 1 is an example of a simple user-defined function. It extends the native Google Map function from three arguments to six, making it easier to create HTML formatted fact bubbles for points on the map, as shown in Fig. 2. The Force-directed Graph function (see Fig. 1i) is a different example, created entirely by use of the JavaScript visualization library D3 [9].

2.2 SPARQL Query Design

When designing SPARQL queries for visualization by Sgvizler, the order of the columns in the result set, i.e., the order of the variables in the `SELECT` block, is crucial. As indicated by the variable names in the query found in Example 1,[3] the `sMap` function expects the two first columns in the result set to contain respectively the latitude and longitude values of points to plot on the map. The remaining result set columns for `sMap` set respectively the heading, the body text, a clickable link and the link to an image to place in the fact bubble which appears when the point on the map is selected. Different rendering functions have different requirements on data input format. The data format for each function is described on the Sgvizler homepage [5].

2.3 Browser and Endpoint Compatibility

Disregarding SPARQL endpoint communication, Sgvizler has the same web browser compatibility as the external JavaScript libraries it uses. For jQuery and the Google Chart Tools this means compatibility with all reasonably new web browsers.[4] The compatibility of endpoint communication is a more complex matter. In general JavaScript has to abide by the *same origin policy*, a security measure which, for the purpose of Sgvizler, means that it cannot retrieve data from a SPARQL endpoint at a different domain, subdomain or port than where the script lives. This is a precaution that surely does not fit well with the idea of distributing data with SPARQL endpoints. However, *Cross-Origin Resource*

[2] See http://code.google.com/apis/chart/interactive/docs/reference.html.

[3] Even though the names of the variables in the example indicate their contents, the actual names are not important for the visualization function.

[4] A list of jQuery supported browsers are available at [6]. Google Chart Tools' information on the subject is: "Charts are rendered using HTML5/SVG technology to provide cross-browser compatibility (including VML for older IE versions) and cross platform portability to iPhones, iPads and Android" [1].

Sharing (CORS) [10] is a specification that aims to safely allow such requests and is supported by most modern browsers—the notable exception being Opera. In order for CORS to work the endpoint server must be *CORS-enabled*, meaning essentially that its header response must include a list of domains (or a wildcard $*$) with which it allows CORS communication. The specification is currently a W3C Working Draft, and not all SPARQL endpoints are CORS-enabled.

A second way to circumvent the same origin policy is to use *JSONP*. Data retrieved as JSONP is returned as a function call on the data in JSON format, thus exploiting that the HTML `<script>` tag is not required to respect the same origin policy. This, of course, requires that the endpoint can return data in JSONP format; luckily many do. Sgvizler supports receiving data in JSONP format. It has successfully been tested on CORS-enabled endpoints with output formats XML [7] and JSON [8] for the browsers Firefox 3.6, Chrome 12, Internet Explorer 8 and Safari 5.1, and, as expected, it does not work for Opera 11.51. On endpoints which return JSONP Sgvizler has been successfully tested also on Opera 11.51, including all the above-mentioned browsers. For same origin requests, endpoint communication is no longer a compatibility issue, thus in such cases Sgvizler works for all browsers compatible with the external JavaScript libraries used.

3 Future Work

The following future work items have been identified. *Technical issues:* Reduce page load time with more parallelization of tasks and selective library import. Improve Sgvizler's "external" API. *More graph visualizations:* RDF data naturally lends itself especially well for graph visualizations. However, the only graph visualization function available in Sgvizler is currently Force-directed Graph and it is in early development. *Linked Data tool integration:* Integrate Sgvizler with popular Linked Data SPARQL frontends. *Vocabulary sensitivity:* Make Sgvizler able to suggest good visualizations based on the vocabulary used by the dataset.

References

1. Google Chart Tools. http://code.google.com/apis/chart/
2. Guang Zheng, J., Ding, L.: How to render SPARQL results using Google visualization API. December 2011. http://iw.rpi.edu/wiki/How_to_render_SPARQL_results_using_Google_Visualization_API
3. Knublauch, H.: SPARQL Web Pages. http://uispin.org/
4. Vrandečić, D., Harth. A.: Spark. http://km.aifb.kit.edu/sites/spark/
5. Skjæveland, M.G.: Sgvizler. http://code.google.com/p/sgvizler/
6. jquery. http://jquery.com/
7. Beckett, D., Broekstra, J., (eds.): SPARQL Query Results XML Format. W3C Recommendation, W3C (2008). http://www.w3.org/TR/rdf-sparql-XMLres/
8. Grant Clark, K., Feigenbaum, L., Torres, E., (eds.): Serializing SPARQL Query Results in JSON. W3C Working Group Note, W3C (2008). http://www.w3.org/TR/rdf-sparql-json-res/
9. Bostock, M.: D3.js. http://mbostock.github.com/d3/
10. van Kesteren, A., (ed.): Cross-Origin Resource Sharing. W3C Working Draft, W3C, July 2010. http://www.w3.org/TR/cors/