The Javascript GraphDB Connector

Table of Content:

Introduction

This document describes the Javascript Client for the sones GraphDB(v2.0) database.

The Javascript client gives you the ability to establish a connection to the database, send GQL-Queries and parse the result. It is also very easy to use and to include in existing projects.
Everyone who wants to use the sones GraphDB as data basis in ANY web application, needs a suitable data transmission interface between the visualization layer and the persistence layer. This is the precisely purpose of the Javascript GraphDB(v2.0) Client. It performs all tasks to deliver and parse the graph out of the database into a query result object. Of course, this library executes this tasks as an Ajax request. The Client uses the already known REST Port of the data storage. To set up a connection, there is only the host and the credentials necessary (credentials don't have to deliver with the script). In this way it is possible to create any numbers of clients to talk to any numbers of GraphDB instances.
Every GQL-Query gets a result. The Javascript GraphDB Client is able to parse the "QueryResult" and represents the graph in vertex-/ , single-edge and hyper-edgeobjects. So it is very easy to fetch the requested information.
Please mind the "Same – Origin – Policy" (SOP) of the Browsers. This restriction prevents cross – domain requests. This is obviously a problem, because of the different service ports (Http: Port 80, GraphDB REST: may Port 9975). Therefore you have to set up your server or use a proxy. Both ways will be described in this document.
Beyond this, the client is part of the Community of our GraphDB(v2.0) and you can take a look at the operation of the library and add or manipulate functionality. All code is located at http://www.github.com/sones.

Feature List

There are a lot of features already implemented; these include the REST client which handles the communication to the service and the provision of the credentials (not urgent necessary). Furthermore there is the result parser, which iterates over the result document and returns known objects to hold a propertygraph in a Javascript environment. Additional, there are the feature to handle some metadata about the request process and the actual result.
The initial release also contains some missing features, like traverse methods or anonymous functions. Probably these functions will be added in later releases.

Here a short summary of the features :

Features:

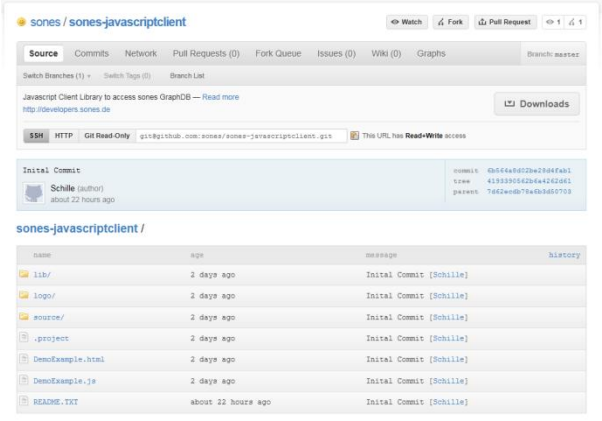
- possibility to easily send GQL-Queries to the service
- parsing methods to create a QueryResult out of the XML-Response
- API to handle vertices, edges and some result meta data
- take one client to one instance of sones GraphDB Service
- simple connection to a GraphDB REST service based on given URI and credentials (not urgent necessary)

Furthermore there are dependences to the jquery framework and some plugins.
These are: jquery-1.5.2.js (current version at the date of release), jquery.base64.js and jquery.urlencode.js (for URL encoding of the query string)

First Steps

Source of supply

Since the Javascript GraphDB client was released, it is reachable at github. You can download it for free at: https://github.com/sones/sones-javascriptclient . Just click on "Download" and get the source. You can choose between *.zip or *.tar.gz. After you extracted the code, you are able to use the client. But, before you run your first test, you have to set up your environment.



Set up your environment

The problem is: The Javascript client makes a request to the REST service of the GraphDB. This service runs on a specified port (default is 9975). But your web page is delivered on a different one (e.g. port 80). Even in the case that both, your web service and GraphDB runs on the same machine, the browser permits the access to the REST service of the database, because of the “Same – Origin – Policy” (see: http://en.wikipedia.org/wiki/Same_origin_policy). Well, even another port represents another origin! So there are some ways in order to bypass the “Same – Origin – Policy” problem.

Set up a Http Server

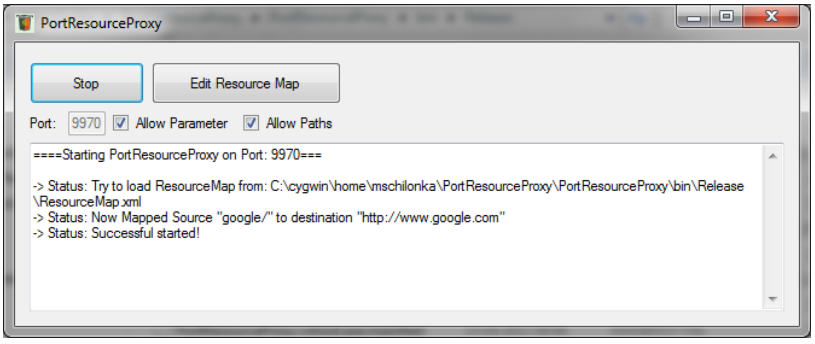
As commonly known Http Server, let’s take a glance at Apache Httpd (see: http://httpd.apache.org/).
Actually, you always set up a small proxy. The Apache Httpd has already a useful mechanism for our problem (see: http://httpd.apache.org/docs/current/mod/mod_proxy.html). You just need to add some lines in the configuration file.
First, open your configuration file located in …\apache\conf\httpd.conf with your favorite texteditor. Then you have to add these lines in the correct way:

RewriteEngine On
ProxyRequests off
<Proxy>
Order deny,allow
Allow from all
</Proxy>
ProxyPass /gql http://localhost:9975/gql
ProxyPassReverse http://localhost:9975/gql /gql
RewriteRule http://localhost:9975/gql ^/gql

With this configuration you can reach the REST interface of the GraphDB through requesting the pattern: http://localhost:”your_port”/gql. The server will translate this request to the address of http://localhost:9975/gql which is the Community Edition default. Therefore you are able to fetch the response of your GraphDB(v2.0) instance. Of course, you have to adapt these settings to your needs.

Set up a simple Proxy

If you don’t want / could / should change the configuration of your Http Server there is also a second way in order to proxy the Ajax request of the Javascript client. There is a small Proxy available also at github: https://github.com/Schille/PortResourceProxy. With this tool, you can easily create http pattern to proxy your GQL request to another origin, for instance the REST interface of the GraphDB.



This Proxy is suitable for development purposes. For further information see the link’s destination and the manual.

The ResourceMap configuration that matches to our needs should be:

```
<Proxy>
  <ResourceMap>
    <Path>
      <source>gql</source>
      <destination>http://localhost:9975/gql</destination>
    </Path>
  </ResourceMap>
</Proxy>
```

May you changed the default port, so you have to adopt it! Don't forget to "Allow Parameter". The PortResourceProxy isn't interested in the port which was configured in the Javascript GraphDBClient.

In both ways the request of the Javascript client goes along to: http://ORIGIN/gql. So you have to handle this request to reach the expected destination at the GraphDB REST interface. Keep in mind to handle the correct port to the client!

Embed into projects

The only folder you really need is the "source". There you find the "sones2-javascriptclient.js". Just copy the source folder into your project and start using the client. Include the client to your html page as it is always done.

```
<script  type="text/javascript" src="source/sones2-javascriptclient.js"></script>
```

Additional, there is a useful function named "*addDependencies()".* This function can automatically add all needed dependencies. Properly you have to edit this function to fit your environment:

```
//Add dependencies
function addDependencies() {
     addJavascript('lib/jquery/jquery-1.5.2.js');
     addJavascript('lib/jquery/jquery.base64.js');
};
```

Important: Before the first test, make sure there is a GraphDB(v2.0) instance running. Without running a database it is impossible to reach a query result, of course. Furthermore it is important, that you handled the Same – Origin – Policy problem.

Usage

Well, you got and included the source. Now it's time to look at the usage.
Since you included the sones-javascriptclient.js it is possible to create an instance – object of "GraphDBClient".
There are two procedures in order to use the client.

The first one in to call the doQuery(gql) function, which takes a GQL Statement and creates a default GraphDBClient prototype.

```
function doQuery(gql){

     var Client = new GraphDBClient('localhost','test','test',9970);
     var QueryResponse = Client.Query(gql);
     delete Client;
     return QueryResponse;
};
```

You see, the parameter of the GraphDB “constructor” – function are already set (username/password ‘test’ is the default value of the Community Edition(v2.0) of sones GraphDB).
As return – value of this function, you get a “QueryResponse”. But this function is not suitable if you want to create more than one GraphDBClients, may because you have more than one GraphDB instances running.
The second way is to implement a function which creates a

```
var Client = new GraphDBClient(HOST,USERNAME,PASSWORD,PORT);
```

You have to put a host (e.g. localhost), a username (not necessary), a password (also not necessary) and perhaps you put a port (not necessary – see: “Set up your environment”)

The host information and credentials will be stored in the object, to reuse the connection configuration. Of course, you create for each GraphDB one instance in order to have a representation of the existing systems.

Now, let’s fire some GQL-Statements. It works like this:

```
var QueryResponse = Client.Query('CREATE VERTEX Car');
```

In this Way, you can talk to the database, using Query(myGQLQuery);
The return value is an Object of “QueryResult”. This query result contains all information about the last query. It is really useful to evaluate the query status, whether it was successful or faulty! The most important functions of QueryResult are:

- getAllFetchedVertices() – returns an array of vertices from the result
- getQueryMeta() – returns a QueryMeta Object, containing the query information

There are some other functions to evaluate the result in the returned objects.
As you have seen, it at the is really easy to use the Javascript GraphDB client. For visualization examples, see the Demo.

Walkthrough the Demo

What’s the Demo?

The Demo consists only one HTML page and a handful javascript files like: jquery dependencies, sones2-javascriptclient.js and the DemoExample.js which consist the logic of the Demo. It is a simple Ajax powered page. The Demo uses the Community Edition of the GraphDB, because all parameters are already set. To get it working you have to set up your environment correctly and include all needed jquery frameworks (if you use the `addDependencies()` function – mind to set up them to)! If not, it would not work.

The configuration (ResourceMap.xml) for the PortResourceProxy should be:

```
<?xml version="1.0" encoding="utf-8" ?>
<Proxy>
  <ResourceMap>
    <Path>
      <source>gql</source>
      <destination>http://localhost:9975/gql</destination>
    </Path>
    <Path>
      <source>sones-javascriptclient</source>
      <destination>http://localhost/sones2-javascriptclient/DemoExample.html</destination>
    </Path>
    <Path>
      <source>DemoExample.js</source>
      <destination>http://localhost/sones2-javascriptclient/DemoExample.js</destination>
    </Path>
    <Path>
      <source>source/</source>
      <destination>http://localhost/sones2-javascriptclient/source/</destination>
    </Path>
    <Path>
      <source>logo/</source>
      <destination>http://localhost/sones2-javascriptclient/logo/</destination>
    </Path>
    <Path>
      <source>lib/</source>
      <destination>http://localhost/sones2-javascriptclient/lib/</destination>
    </Path>
  </ResourceMap>
</Proxy>
```

The DemoExample has to be provided by any server on port 80 (in this example). So everything would be proxyed.

It is important to reach all of the Javascript files. Take a glance at the PortResourceProxy's console to get further information about unreachable requests!

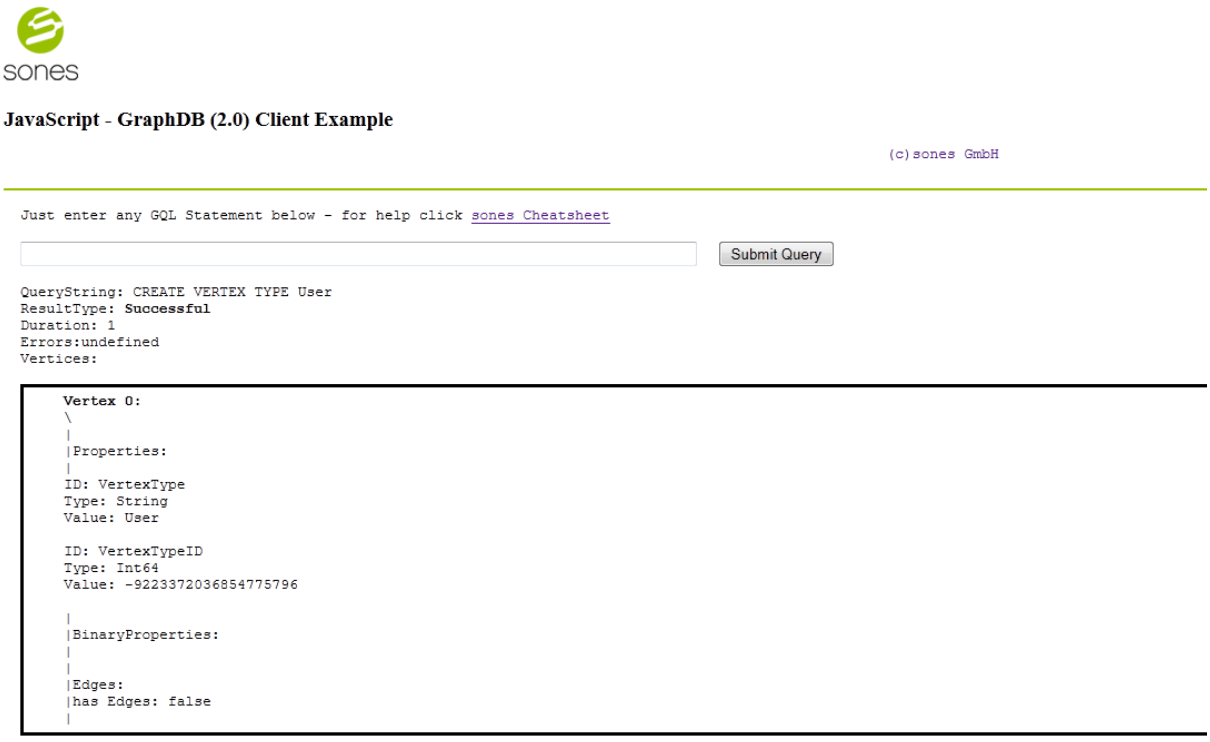
Now let's shift attention to the project:

```
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
   <script type="text/javascript" src="DemoExample.js"></script>
    <script type="text/javascript" src="source/sones-javascriptclient.js"></script>
   <script type="text/javascript">
         addDependencies();
   </script>
   <title>DemoExample</title>
</head>
```

As you see, the `addDependencies()` function will be executed. In the Demo you have two simple inputs:

```
<input type="text" id="gql" size="100" style="margin:15px auto auto 15px"/>
<input type="button" value="Submit Query" name="btnSubmit"
onclick="DemoExample(document.getElementById('gql').value);" style="margin:15px auto auto
15px">
```

In this way, you can fire some test GQL statements by passing them into the textfield. Please mind the DemoExample.js



A way to visualize the “QueryResult”

The DemoExample calls the `DemoExample(gql){}` which handles the request.

```
function DemoExample(gql){
    var result = doQuery(gql)
    var div = document.getElementById("output");
```

…

It gets the “div” output and renders the whole html by themselves. If you are interested in the output, you can take a glance at the “DemoExample.js”.

Get involved at github

The complete Community Edition (Open Source Edition) of our sones GraphDB(v2.0) is provided at: http://www.github.com/sones. At this location, there are the database and some other projects available.