Implementing the Chart Tools Datasource Protocol (V0.6)

This page describes how you can implement a service that supports the Chart Tools Datasource protocol to expose data to charts using the Query class (https://google-developers.appspot.com/chart/interactive/docs/reference#queryobjects).

Contents

Audience

This page is intended primarily for developers who will be creating their own data source without the aid of the Chart Tools Datasource Library (https://google-developers.appspot.com/chart/interactive/docs/dev/dsl_about). If you are using that or any other helper libraries, read your library's documentation first.

This page is also intended for readers interested in understanding the wire protocol used for communication between a client visualization and a data source.

If you are creating or using a visualization, you do not need to read this page.

In order to read this document, you should understand basic JSON and HTTP request syntax. You should also understand how charts work from a user's perspective.

Note: Google does not officially endorse or support any non-Google Datasources that support the Chart Tools Datasource protocol.

Overview

You can implement the Chart Tools Datasource protocol in order to become a data source provider for your own charts, or other charts. A Chart Tools Datasource exposes a URL, called a *Datasource URL*, to which a chart can send HTTP GET requests. In response, the data source returns properly formatted data that the chart can use to render the graphic on the page. This request-response protocol is known as the Google Visualization API *wire protocol*.

The data that is served by a data source can be extracted from various resources, such as a file or database. The only restriction is that you can format the data as a two dimensional table with typed columns.

As a Chart Tools Datasource, you must parse a request in a specific format, and return a response in a specific format. You can do this in one of two general ways:

- **Use one of the following helper libraries** to handle the request and response, and construct the DataTable to return. If you use one of these libraries, you need write only the code needed to make your data available to the library in the form of a table.
    - Java Datasource Library (https://google-developers.appspot.com/chart/interactive/docs/dev/dsl_intro) - Handles the request and response, creates the response table from the data you give it, and implements the Google Chart Tools SQL query language.
    - Python Datasource Library (https://google-developers.appspot.com/chart/interactive/docs/dev/gviz_api_lib) - Creates a response table generates the response syntax. Does not handle parsing the request (#requestformat) or implementing the Google Chart Tools SQL query language.

    OR

- **Write your own data source from scratch** by handling the request, constructing a DataTable, and sending the response.

How it works:

1. The data source exposes a URL, called the *datasource URL*, to which charts send an HTTP GET request.
2. The client makes an HTTP GET request with parameters that describe what format to use for returned data, an optional query string, and optional custom parameters.
3. The Datasource receives and parses the request, as described in Request Format (#requestformat).
4. The Datasource prepares the data in the format requested; typically, this is a JSON table. Response formats are covered in the section Response Format (#responseformat). The Datasource can optionally support the Visualization API query language (https://google-developers.appspot.com/chart/interactive/docs/querylanguage) that specifies filtering, sorting, and other data manipulation.

5. The Datasource creates an HTTP response that includes the serialized data and other response parameters, and sends it back to the client as described in Response Format (#responseformat)

Note: All parameters and string constant values listed in this document for requests and responses (such as `responseHandler` and "ok") are lowercase, and case-sensitive.

Minimum Requirements

These are the minimum requirements to serve as a Chart Tools Datasource:

- T**he data source should accept HTTP GET requests, and should be available to your clients.**
- The protocol can change and supports a version scheme (the current version is 0.6), so your data source should support requests using previous versions as well as the current version. You should try to support new versions as soon as they are released to avoid breaking any clients that upgrade to the newest version quickly.
- **Do not fail if unknown properties are sent as part of the request.** This is because new versions can introduce new properties that you are not aware of.
- **Parse only properties that you expect.** Although new versions might introduce new properties, do not blindly accept and use the whole request string. To protect yourself against malicious attacks, carefully parse and use only the properties that you expect.
- **Document your data source requirements carefully** if you are not coding the client charts yourself. This includes documenting the following information:
    - Any custom parameters that you accept,
    - Whether or not you can parse the Google Visualization API query language, and
    - What kind of data you return, and the structure of that data (what the rows and columns represent, and any labeling).
- **Take all standard security precautions for a site that accepts requests from unknown clients**. You can reasonably support MD5, hashing, and other security mechanisms in your parameters to authenticate requests or help secure against malicious attacks, and expect clients to be aware of your requirements and respond to them. However, be sure to document all your requirements well, if you are not coding the charts yourself. See Security Considerations (#security_considerations) below.
- **All request and response strings should be UTF-8 encoded.**

- **The most important response format is JSON.** Be sure to implement JSON first, since this is the format used by most charts. Add other response types afterward.
- You are not *required* to support the Google Visualization API query language (https://google-developers.appspot.com/chart/interactive/docs/querylanguage), but it makes your data source more useful to customers.
- You are not *required* to support requests from any and all chart types, and you can support custom parameters for custom charts. But you should return responses in the standard format described below.

Security Considerations

When designing your data source, you'll need to consider how secure your data must be. You can use a variety of security schemes for your site, from simple password access to secure cookie authentication.

XSSI (cross-site script inclusion) attacks are a risk with charts. A user might navigate to a page that holds a malicious script that then starts trying to make queries to data source URLs, using the credentials of the current user. If the user has not logged out of a site, the script will be authenticated as the current user and have permissions on that site. Using a <script src> tag the malicious script can be able to include the data source, similar to JSONP.

As an additional level of security, you might consider restricting requests to those coming from the same domain as your data source. This will considerably restrict the visibility of your data source, but if you have very sensitive data that should not be accessed from outside your domain, you should consider it. A data source that only allows requests from the same domain is called a *restricted data source*, as opposed to an *unrestricted data source*, which will accept queries from any domain. Here are some details on how to implement a restricted data source:

To ensure that a request is truly coming from within your domain, and not from an outside domain (or a browser inside the domain that's under XSRF attack (http://en.wikipedia.org/wiki/Cross-site_request_forgery)):

- Verify the presence of an "X-DataSource-Auth" header in the request. This header is defined by the Google Visualization API; you don't need to examine the contents of this header, only verify that it is there. If you're using the Google Chart Tools Datasource library (https://google-developers.appspot.com/chart/interactive/docs/dev/dsl_about), you can have the library handle this for you.

- Use cookie authentication (http://en.wikipedia.org/wiki/HTTP_cookie) to authenticate the client. There is no known way to inject custom headers to a cross-domain request while still keeping authentication cookies in place.
- Make the JavaScript unlikely to execute when included with a <script src> tag. To do this, prefix your JSON response with)]}' followed by a newline. In your client strip the prefix from the response. For XmlHttpRequest this is only possible when the request originated from the same domain.

Request Format

A client sends an HTTP GET request with several parameters, including custom elements, an optional query string, signature, and other elements. You are only responsible for parsing out the parameters described in this section, and should be careful not to handle others to avoid malicious attacks.

Be sure to have default values for optional parameters (both standard and custom), and document all your defaults in your site documentation.

Here are a few sample requests (you can see more request and response samples at the end of this document in Examples (#examples)):

Note: The following request strings, and those shown in the *Examples* section, should be url-escaped before sending.

```
Basic request, no parameters:
http://www.example.com/mydatasource

Request with the tqx parameter that contains two properties:
http://www.example.com/mydatasource?tqx=reqId:0;sig:4641982796834063168

Request with a query string:
http://www.example.com/mydatasource?tq=limit 1
```

Here are the list of all standard parameters in the request string. Note that both parameter names (such as "version") and constant string values (such as "ok", "warning", and "not_modified") are case-sensitive. The table also describes whether the parameter is required to be sent and, if sent, whether you are required to handle it.

Parameter	Required in	Data Source	Description

	Request?	Must Handle?	
tq	No	No	A query written in Google Visualization API query language (https://google-developers.appspot.com/chart/interactive/docs/querylanguage), specifying how to filter, sort, or otherwise manipulate the returned data. The string does not need to be quoted. **Example:** `http://www.example.com/mydatasource?tq=select Col1`

tqx	No	Yes	A set of colon-delimited key/value pairs for standard or custom parameters. Pairs are separated by semicolons. Here is a list of the standard parameters defined by the Visualization protocol:
			
			• **`reqId`** - [*Required in request; Data source must handle*] A numeric identifier for this request. This is used so that if a client sends multiple requests before receiving a response, the data source can identify the response with the proper request. Send this value back in the response.
			• **`version`** - [*Optional in request; Data source must handle*] The version number of the Google Visualization protocol. *Current version is 0.6.* If not sent, assume the latest version.
			• **`sig`** - [*Optional in request; Optional for data source to handle*] A hash of the **`DataTable`** sent to this client in any previous requests to this data source. This is an optimization to avoid sending identical data to a client twice. See Optimizing Your Request (#optimizingrequests) below for information about how to use this.
			• **`out`** - [*Optional in request; Data source must handle*] A string describing the format for the returned data. It can be any of the following values:
			  • **json** - [*Default value*] A JSON response string (described below).
			  • **html** - A basic HTML table with rows and columns. If this is used, the *only* thing that should be returned is an HTML table with the data; this is useful for debugging, as described in the Response Format (#responseformat) section below.
			  • **csv** - Comma-separated values. If this is used, the *only* thing returned is a CSV data string. You can request a custom name for the file in the response headers by specifying a **`outFileName`** parameter.
			  • **tsv-excel** - Similar to **`csv`**, but using tabs instead of commas, and all data is utf-16 encoded.
			
			Note that the only data type that a chart built on the Google Visualization API will ever request is json. See Response Format

			(#responseformat) below for details on each type. • **`responseHandler`** - [*Optional in request; Data source must handle*] The string name of the JavaScript handling function on the client page that will be called with the response. If not included in the request, the value is "google.visualization.Query.setResponse". This will be sent back as part of the response; see Response Format (#responseformat) below to learn how. • **`outFileName`** - [*Optional in request; Optional for data source to handle*] If you specify **`out:csv`** or **`out:tsv-excel`**, you can optionally request the file name specified here. **Example:** **`outFileName=results.csv`**. **Example:** **`tqx=version:0.6;reqId:1;sig:5277771;out:json`**; **`responseHandler:myQueryHandler`**
tqrt	No	No	*Reserved: ignore this parameter.* The method that was used to send the query.

Response Format

The format of the response depends on the request's **`out`** parameter, which specifies the type of response expected. See the following sections to learn how to respond to each request type:

- JSON (#jsondatatable) - Returns a JSON response that includes the data in a JavaScript object that can be passed directly into a **`DataTable`** constructor to populate it. This is by far the most common request type, and the most important to implement properly.
- CSV (#csvdatatable) - Returns a flat comma-separated value list, to be handled by the browser.
- TSV (#tsvdatatable) - Returns a tab-separated value list, to be handled by the browser.
- HTML (#htmldatatable) - Returns an HTML table to be rendered by the browser.

You can use the Google Visualization Data Source Library (https://google-developers.appspot.com/chart/interactive/docs/dev/dsl_about) (java) or the visualization python library (https://google-developers.appspot.com/chart/interactive/docs/dev/gviz_api_lib) to generate these output formats for you.

JSON Response Format

The default response format is JSON (http://en.wikipedia.org/wiki/JSON) if the request includes an "X-DataSource-Auth" header, and JSONP (http://en.wikipedia.org/wiki/JSON#JSONP) otherwise. Note that the Google chart client actually supports a modified version of JSON and JSONP; if you are using the Java (https://google-developers.appspot.com/chart/interactive/docs/dev/dsl_about) or Python (https://google-developers.appspot.com/chart/interactive/docs/dev/gviz_api_lib) helper libraries, they will put out the proper code for you; if you are parsing responses by hand, see JSON Modifications (#json_modifications) below.

If you are enforcing same-domain requests, you should verify the presence of the "X-DataSource-Auth" header in the request and use authorization cookies.

This is the only response format specified by the Google Visualization API method `google.visualization.Query.send()` (https://google-developers.appspot.com/chart/interactive/docs/reference#Query). You can see some example JSON requests and responses at the end of this page in Examples (#examples). You can use the Java (https://google-developers.appspot.com/chart/interactive/docs/dev/dsl_about) or Python (https://google-developers.appspot.com/chart/interactive/docs/dev/gviz_api_lib) helper libraries to create this response string for you.

This response format is a UTF-8 encoded JSON object (an object wrapped by by braces { } with each property separated by a comma) that includes the properties in the table below (the data is assigned to the `table` property). This JSON object should be wrapped inside the `responseHandler` parameter value from the request. So, if the request's `responseHandler` value was "myHandler", you should return a string like this (only one property shown for brevity):

```
"myHandler({status:ok, ...})"
```

If the request did not include a `responseHandler` value, the default value is "google.visualization.Query.setResponse", so you should return a string like this (only one property shown for brevity):

```
"google.visualization.Query.setResponse({status:ok, ...})"
```

Here are the available response object members:

Property	Required?	Description
version	No	A string number giving the Google Visualization wire protocol version numb specified, the client assumes the latest version.

		Example: `version=0.6`
reqId	Yes*	A string number indicating the ID of this request for this client. If this was in request, return the same value. See the **`reqId`** description in the request se (#requestformat) for more details. ***** If this parameter was not specified in the request, you don't have to set it i response.
status	Yes	A string describing the success or failure of this operation. Must be one and one of the following values: • **`ok`** - Successful request. A table *must* be included in the **`table`** property • **`warning`** - Success, but with issues. A table *must* be included in the **`tab`** property. • **`error`** - There was a problem. If you return this, you should *not* return **`ta`** and must return **`errors`**. **Example:** `status:'warning'`

warnings	Only if `status=warning`	An array of one or more objects, each describing a non-fatal problem. Requi **`status=warning`**, not allowed otherwise. Each object has the following st properties (return only one value for each property):
		
		• **`reason`** - [*Required*] A one-word string description of the warning. The pr defines the following values, but you can define your own values, if you r (however, the client will not process custom values in any special way). Y only have one **`reason`** value:
		  • **`data_truncated`** - The returned **`DataTable`** had some rows rer either because the user included a query string that trimmed the r or the data source did not want to return complete results for som reason.
		  • **`other`** - A generic unspecified warning.
		• **`message`** - [*Optional*] A short quoted string explaining the problem, possi as a title for an alert box. This might be displayed to the user. HTML is n supported.
		• **`detailed_message`** - [*Optional*] A detailed quoted string message expl the problem, and any possible solutions. The only HTML supported is th element with a single **`href`** attribute. Unicode encoding is supported. Th be displayed to the user.
		
		Example: `warnings:[{reason:'data_truncated',message:'Retr data was truncated'}]`

errors	Required if `status=error`	An array of one or more objects, each describing an error. Required if **`status=error`**, not allowed otherwise. This is similar to the **`warnings`** va Note that the **`not_modified`** error, though an error, is not actually an error client.

		The array has the following string members (return only one value for each member):
		• **`reason`** - [*Required*] Same as the **`warnings.reason`**, but the following are defined:
		  • **`not_modified`** - The data has not changed since the last reques is the reason for the error, you should not have a value for **`table`**.
		  • **`user_not_authenticated`** - If the data source requires authen and it has not been done, specify this value. The client will then di alert with the value of **`message`**.
		  • **`unknown_data_source_id`**
		  • **`access_denied`**
		  • **`unsupported_query_operation`**
		  • **`invalid_query`**
		  • **`invalid_request`**
		  • **`internal_error`**
		  • **`not_supported`**
		  • **`illegal_formatting_patterns`**
		  • **`other`** - A generic, unspecified error.
		• **`message`** - [*Optional*] Same as **`warnings.message`**. **Note:** It is possible malicious user could use a detailed data string as a means for accessin unauthorized data, or even using it to attack your data or your site. If you data that should be secure, or process user queries directly, consider no returning a detailed error message that could provide information to an a instead, give a generic message, such as "Bad query string."
		• **`detailed_message`** - [*Optional*] Same as **`warnings.detailed_mess`** See the warning for overly detailed **`message`** information.
		
		Example: **`status:'error',errors:[{reason:'not_modified',message:'Data not modified'}]`**

sig	No	A hashed value of the table object. Useful for optimizing data transfer betw client and the data source. You can choose any hash algorithm you want. If support this property, you should return the value passed in by the client if n is returned, or return a new hash if new data is returned. **Example:** **`sig:'5982206968295329967'`**
table	No	A [DataTable](https://google-developers.appspot.com/chart/interactive/docs/reference#dataparam) object in Javascript literal notation, with your data. See the linked reference for details on the format of this object. Here's an example of a simple data t

```
{cols:[{id:'Col1',label:'',type:'number'}],
 rows:[{c:[{v:1.0,f:'1'}]},
       {c:[{v:2.0,f:'2'}]},
       {c:[{v:3.0,f:'3'}]},
       {c:[{v:1.0,f:'1'}]}
      ]
}
```

The **`table`** property should only be present if **`status=ok`** or **`status=warn`** you are not returning data, do not include this property (that is, don't pass ba property with an empty string value).

Example: See *Examples* (#examples) below.

JSON Modifications

Google's helper libraries, and all queries sent to Google, return a slightly non-standard version of JSON/JSONP. If you are not parsing the returned code yourself, this should not matter to you. The Visualization API client supports both standard and the modified versions of JSON. Here is a summary of the differences:

- JSON does not support JavaScript Date values (for example, "new Date(2008,1,28,0,31,26)"; the API implementation does. However, the API does now support a custom valid JSON representation of dates as a string in the following format: `Date(year, month, day[,hour, minute, second[, millisecond]])` where everything after day is optional, and months are zero-based.
- JSON uses double-quotes for dictionary keys; the API implementation uses unquoted keys.
- JSON requires double-quotes around string values; the API implementation uses single quotes.

Optimizing JSON Responses

If a client makes two requests, and the data has not changed between requests, it makes sense not to resend the data--doing so would waste bandwidth. To make requests more efficient, the protocol supports caching the data on the client, and sending a signal in the response if the data has not changed since the last request. Here's how this works:

1. The client sends a request to the data source.

2. The data source generates a `DataTable` as well as a hash of the `DataTable` object, and returns both in its response (the hash is returned in the `tqx.sig` parameter). The Google Visualization API client caches the `DataTable` and `sig` value.
3. The client sends another request for data, including the cached `tqx.sig` value.
4. The data source can respond in one of two ways:
    - If the data has changed from the previous request, the data source sends back the new `DataTable` and new `sig` value hash.
    - If the data has not changed from the previous request, the data source sends back `status=error`, `reason=not_modified`, `sig=`*`old_sig_value`*.
5. In either case, the page hosting the chart gets a successful response, and can retrieve the `DataTable` by calling `QueryResponse.getDataTable()` (https://google-developers.appspot.com/chart/interactive/docs/reference#QueryResponse). If the data is the same, it will simply be the cached version of the table.

Note that this only works for JSON requests from charts built on the Google Visualization API.

CSV Response Format

If the request specifies `out:csv`, the response includes no metadata, but simply a CSV representation of the data. A CSV table is typically a comma-separated list, where each row of data is a comma-separated list of values, ending in a UNIX newline character (\n). The cell values should have the same type for each column. The first row is the column labels. Here's an example of a three-row, three-column table:

```
A, B, C
1.0, "yes", true
2.0, "no", false
3.0, "maybe", true
```

The CSV format is not specified by this protocol; the data source is responsible for defining its CSV format. However, a common format is a set of values separated by commas (with no intervening spaces), and a newline (\n) at the end of every row. When a browser receives a CSV string reply, it might ask the user what application to use to open the string, or might simply render it on the screen. The Java (http://code.google.com/p/google-visualization-java/) and Python (http://code.google.com/p/google-visualization-python/) open source libraries provide a method to convert a DataTable to a CSV string.

If the request includes an `outFileName` member of the `tqx` parameter, you should try to include the specified file name in the response headers.

The `google.visualization.Query` (https://google-developers.appspot.com/chart/interactive/docs/reference#Query) object does not support a request for a CSV response. If a client wants to request CSV, you could embed a Visualization Toolbar gadget (https://google-developers.appspot.com/chart/interactive/docs/gallery/toolbar) on your page, or they could use custom code to create the request, or you could supply a link that sets the `out:csv` property of `tqx` explicitly, as shown in the following request URL:

Request

```
http://www.example.com/mydatasource?tqx=reqId:1;out:csv
```

Response

```
Label 1,Label2\n1,a\n2,b\n3,c\n4,d
```

TSV Response Format

If the request specifies `out:tsv-excel`, the response includes no metadata, but simply a tab-separated representation of the data, utf-16 encoded (http://en.wikipedia.org/wiki/UTF-16). If the request includes an `outFileName` member of the `tqx` parameter, you should try to include the specified file name in the response headers.

HTML Response Format

If the request specifies `out:html`, the response should be an HTML page defining an HTML table with the data. This is useful for debugging your code, because the browser can render your result in a readable format directly. You cannot send a query for an HTML response using the `google.visualization.Query` (https://google-developers.appspot.com/chart/interactive/docs/reference#Query) object. You must make a query for an HTML response using custom code, or by typing a URL similar to this one in your browser:

Request

```
http://www.example.com/mydatasource?tqx=reqId:1;out:html
```

Response

```
<html><body><table border='1' cellpadding='2' cellspacing='0'><tr style='font
```

Examples

Here are some example requests and responses. Note that requests have not been url-escaped; that is typically done by either the browser, or the `google.visualization.Query` object.

Simple request: Returns the basic information with a three column, four row table.

Request:
```
http://www.example.com/mydatasource
```

Response
```
google.visualization.Query.setResponse({version:'0.6',reqId:'0',status:'ok',s
```

Simple request with a response handler: Returns a three column, three row table with different data types.

Request:
```
http://www.example.com/mydatasource?tqx=responseHandler:myHandlerFunction
```

Response
```
myHandlerFunction({version:'0.6',reqId:'0',status:'ok',sig:'46419827968340631
```

Query with a simple query string: Request for a single column, returns a single column with four rows.

Request:
```
http://www.example.com/mydatasource?tq=select Col1
```

Response:
```
google.visualization.Query.setResponse({version:'0.6',reqId:'0',status:'ok',s
```

Data not modified error: Example of a `not_modified` error.

Request:
```
http://www.example.com/mydatasource?tqx=reqId:0;sig:4641982796834063168
```

Response:
```
google.visualization.Query.setResponse({version:'0.6',reqId:'0',status:'error
```

Data truncated warning: Example of a `data_truncated` warning. Notice that the request still returns data.

```
Request:
http://www.example.com/mydatasource?tq=limit 1

Response:
google.visualization.Query.setResponse({version:'0.6',reqId:'0',status:'warni
```

Access denied error: Example of an `access_denied` error.

```
Request:
http://www.example.com/mydatasource

Response:
google.visualization.Query.setResponse({version:'0.6',reqId:'0',status:'error
```

Invalid query string: Example of a request with an invalid query string. Note that the detailed message is a generic message, rather than the actual error message.

```
Request:
http://www.example.com/mydatasource?tq=select A

Response:
google.visualization.Query.setResponse({version:'0.6',reqId:'0',status:'error
```

Development Tools

- Java Datasource Library (https://google-developers.appspot.com/chart/interactive/docs/dev/dsl_intro) (from Google) - Handles the request and response, creates the response table from the data you give it, and implements the Google Chart Tools SQL query language.
- Python Datasource Library (https://google-developers.appspot.com/chart/interactive/docs/dev/gviz_api_lib) (from Google) - Creates a response table generates the response syntax. Does not handle parsing the request (#requestformat) or implementing the Google Chart Tools SQL query language.
- MC-Google_Visualization (http://code.google.com/p/mc-goog-visualization/wiki/UserDocumentation) (Third-party) -

This is a PHP server-side library you can use to implement a Chart Tools Datasource for MySQL, SQLite, and PostgreSQL database engines using PDO.

- bortosky-google-visualization (http://code.google.com/p/bortosky-google-visualization/wiki/ReadMe) (Third-party) - This is a helper library to create a Google Visualization API Datatable for .NET users.
- GV Streamer (http://www.gvstreamer.com/) (Third-party) - GV Streamer is a server-side tool that can convert data from different sources into valid query responses to Google charts. GV Streamer supports several languages (for example, PHP, Java, .NET) and several raw data sources (for example, MySql).
- TracGViz (http://mac.softpedia.com/get/Development/HTML/TracGViz.shtml) (Third-party) - TracGViz is a free and open source tool that provides components so that Trac will be able to use chart gadgets as well implements data managed by Trac as a Google Chart Tools Datasource source.
- vis-table (http://code.google.com/p/vis-table/) (Third-party) - A library implementing a Google Chart Tools Datasource in PHP. It has three main parts. The datatable implementation itself, the query language parser, and the formatters.
- Google Datasource Implementation in Oracle PL/SQL (http://code.google.com/p/oragoods/w/list) (Third-party) - An Oracle PL/SQL package that enables Oracle to server Data Sources directly from the database. So basically you can use any Oracle query as a Google Chart Tools Datasource (the package will return a JSON file with the data). It has almost-full support for the Google Query Language.

上次更新日期：九月 27, 2012