Creating Chart Types

This page describes how to develop your own chart type and make it available to users.

Audience

This page assumes that you have read the Using Charts (https://google-developers.appspot.com/chart/interactive/docs/index) page. It also assumes that you are familiar with JavaScript (http://www.ecma-international.org/publications/standards/Ecma-262.htm) and object-oriented programming. There are many JavaScript tutorials (http://www.google.com/search?q=javascript+tutorials) available on the Web.

Creating a Chart

Charts are exposed to the user through a JavaScript library that you create. Here are the steps for creating a chart library:

1. **Choose a namespace for your code.** (#Avoiding_Namespace_Conflicts) Other pages will be hosting your code; you should try to avoid naming conflicts.
2. **Implement your chart object.** (#implementthevisualization) Implement a JavaScript object that exposes the following:
    - A constructor,
    - A `draw()` method to draw your object inside the DOM element passed to the constructor,
    - Any other optional standard methods for a client to use, such as `getSelection()` (https://google-developers.appspot.com/chart/interactive/docs/reference#visgetselection), and
    - Any custom methods that you want to expose to your clients.
3. **[*Optional*] Implement any events** (#Documentation_and_Samples) that you want to fire for the client to catch.
4. **Write the documentation for your chart.** (#Documentation_and_Samples) If you don't document it, people probably won't be able to embed it.
5. **Post your chart in the Chart Gallery.** (#post)

Top Tips

- You can develop a chart using the Google Web Toolkit! See the GWT chart page (http://code.google.com/p/gwt-google-apis/wiki/VisualizationGettingStarted) for more information.
- You can download the **`goog.visualization`** API class and method definitions to enable autocompletion in your IDE (code editor). Download the file from http://www.google.com/uds/modules/gviz/gviz-api.js (http://www.google.com/uds/modules/gviz/gviz-api.js) and save it to your project. Most IDEs will index it automatically and enable autocompletion, though your IDE might be different. Note that the file might not always be up to date; check the reference pages (https://google-developers.appspot.com/chart/interactive/docs/reference) for the most up to date API reference.

Choose a Namespace

Your chart can be embedded on a page that hosts other charts or other unrelated JavaScript. To avoid naming conflicts with other code or CSS class names, you should choose a unique namespace for your chart code. A good choice for a namespace is the URL that you will use to host your script (minus the WWW and any extensions). So, for example, if your chart will be posted at www.example.com, you would use **`example`** as your unique namespace. You can add additional suffixes, separated by . marks, to further group your chart (all of Google's charts have the namespace **`google.visualization`**). Use your namespace object to store your chart object, as well as any global variables that you might need.

Here is an example of creating a namespace object to hold a chart class called MyTable, as well as any global variables needed:

```
// Namespace, implemented as a global variable.
var example = {};

// MyTable class constructor.
example.MyTable = function(container) {
  // ...
}

// MyTable.draw() method.
example.MyTable.prototype.draw = function(data, options) {
  // ...
}
```

Avoiding CSS Conflicts

If you use CSS, make sure not to write CSS rules that can affect other charts on the page. For example, a rule such as `td {color: blue;}` is highly discouraged, since it will affect any other `<td>` element on the page, not only within your chart. One way to overcome this is to enclose your entire chart in a `<div>` with a class name, and have all your CSS rules apply only to elements that are descendants of an element with that class name. For example, The following CSS rule will affect only `<td>` elements that have an element with the class name "example" as an ancestor.

```
td.example {color: blue;}
```

Then you can wrap your chart in a `<div>` with :

```
<div class="example">
  ...
</div>
```

Implement your Chart

You'll need to implement your chart as a JavaScript object that exposes the standard methods described in the Reference Section (https://google-developers.appspot.com/chart/interactive/docs/reference#standardproperties). The two required methods are the constructor and the draw() methods. You can also expose any additional methods to your user that are appropriate for your chart. Just remember that easier to use is better.

The Constructor

Your chart should expose a single constructor that takes a single parameter, a DOM element into which you will draw your chart. Typically, charts store a local copy of this element in the constructor for later use:

```
function example.MyTable(container) {
  this.container = container
}
```

The `draw()` Method

Your chart class should have a method `draw()` defined in the prototype of your chart class. The `draw()` method accepts two parameters:

1. A `DataTable` (https://google-developers.appspot.com/chart/interactive/docs/reference#DataTable) that holds the data to display.
2. An optional map of name/value options for your chart. The names and value types of the options are defined by you for your specific chart. For example, in the Hello Chart example below, the chart supports an option named 'showLineNumber' with a value of type Boolean. You should support a default value for each option, in case the user does not pass a value for a specific option. This parameter is optional, so you should also be prepared to use all default values if the user does not pass in this parameter (*more information* (https://google-developers.appspot.com/chart/interactive/docs/reference#visdraw)).

```
example.MyTable.prototype.draw = function(data, options) {
  // Process data and options and render output into the container element.
  ...
}
```

Hello Chart!

Here's a simple chart that displays a `DataTable` data as an HTML table:

	Name	Manager
1	Mary	
2	John	Mary
3	Steve	Mary
4	Ellen	Steve
5	Robert	Steve

And here's the implementation code:

```
// Declare a unique namespace.
var example = {};

// Class constructor. Parameter container is a DOM elementon the client that
```

```
// that will contain the chart.
example.MyTable = function(container) {
  this.containerElement = container;
}

// Main drawing logic.
// Parameters:
//   data is data to display, type google.visualization.DataTable.
//   options is a name/value map of options. Our example takes one option.
example.MyTable.prototype.draw = function(data, options) {

  // Create an HTML table
  var showLineNumber = options.showLineNumber; // Boolean configuration optio

  var html = [];
  html.push('<table border="1">');

  // Header row
  html.push('<tr>');
  if (showLineNumber) {
    html.push('<th>Seq</th>');
  }
  for (var col = 0; col < data.getNumberOfColumns(); col++) {
    html.push('<th>' + this.escapeHtml(data.getColumnLabel(col)) + '</th>');
  }
  html.push('</tr>');

  for (var row = 0; row < data.getNumberOfRows(); row++) {
    html.push('<tr>');
    if (showLineNumber) {
      html.push('<td align="right">', (row + 1), '</td>');
    }

    for (var col = 0; col < data.getNumberOfColumns(); col++) {
      html.push(data.getColumnType(col) == 'number' ? '<td align="right">' :
      html.push(this.escapeHtml(data.getFormattedValue(row, col)));
      html.push('</td>');
    }
    html.push('</tr>');
  }
  html.push('</table>');

  this.containerElement.innerHTML = html.join('');
}

// Utility function to escape HTML special characters
example.MyTable.prototype.escapeHtml = function(text) {
  if (text == null)
```

```
    return '';
  return text.replace(/&/g, '&').replace(/</g, '<')
      .replace(/>/g, '>').replace(/"/g, '"');
}
```

Including Your Chart in a Web Page

To use the previous chart, save it in a .js file accessible from your browser. Then save the following code, changing the `<script>` source parameter to point to your JavaScript file:

```
<html>
  <head>
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader
    <script type="text/javascript" src="mytablevis.js"></script>
    <script type="text/javascript">
      google.charts.load("current");

      // Set callback to run when API is loaded
      google.charts.setOnLoadCallback(drawVisualization);

      // Called when the Chart API is loaded.
      function drawVisualization() {

        // Create and populate a data table.
        var data = new google.visualization.DataTable();
        data.addColumn('string', 'Task');
        data.addColumn('number', 'Daily Hours');
        data.addRows(5);
        data.setCell(0, 0, 'Work');
        data.setCell(0, 1, 11);
        // Add more data rows and cells here

        // Instantiate our table object.
        var vis = new example.MyTable(document.getElementById('mydiv'));

        // Draw our table with the data we created locally.
        vis.draw(data, {showLineNumber: true});
       }
   </script>
  <title>MyTable example page</title></head>
  <body>
    <div id="mydiv"></div>
    <p>This page demonstrates hosting a table visualization.</p>
  </body>
</html>
```

Implement Your Events

If you want your chart to fire useful events (for example, timer events, or user-initiated events such as clicks), you'll need to call the `google.visualization.events.trigger` function with the details of your event (name, properties to send, etc.). You can find details on the Events (https://google-developers.appspot.com/chart/interactive/docs/dev/events) page. You can either expose your event details to the client by adding properties to the event object, or you can expose a get...() method of some type on your chart, and the client can call that method to get the event details. In either case, document your methods or event properties well.

Document Your Chart

If you don't properly document your chart, you probably won't get many users. Be sure to document the following:

- Describe all the methods that you support. The `draw()` method is common to all charts, but each chart can support its own additional methods. (You probably don't need to document your constructor, unless it has non-standard behavior.) You can find a list of expected methods on the Reference Page (https://google-developers.appspot.com/chart/interactive/docs/reference#standardproperties).
- Describe all the options that your `draw()` method supports. This includes the name of each option, the expected value type, and its the default value.
- Describe all the events that you trigger. This means the name and properties of each event, and when each event is triggered.
- List the URL of your chart library that should be used in the client's `<script>` include statement, and give the URL for your documentation.
- Give the fully-qualified name of your chart.
- Create sample pages that demonstrate how to use your chart with the options it supports, its events, and custom methods.
- Clearly describe the data policy of your chart. Most charts process the data within the browser, but some may send data to a server, for example to create an image of a chart or a map. If you do send data to a server:
    - Clearly define which data is sent.
    - Note how long the data will be saved on the server.

- Document which entities will have access to the data. For example, Company XYZ, etc.
- Specify if the data will be logged and for how long.

Your documentation will be hosted in the same place as your chart code (see *Submit Your Chart to the Gallery* below).

Submit Your Chart to the Gallery

After you've written your chart, submit (https://google-developers.appspot.com/chart/interactive/docs/submit) it for posting in the "Additional Charts" section of our gallery (https://google-developers.appspot.com/chart/interactive/docs/more_charts). Submitting a chart means that you'll have to sign an agreement with us agreeing not to create malicious software, or misuse user data. The gallery is just a list of *pointers* to charts that we've created, or that we've reviewed; you can choose to *host* the actual JavaScript library and documentation on your own site, or you can have Google host the library and documentation for you. Specify whether you want us to host your chart when you post it to the gallery.

Troubleshooting

If your code doesn't seem to be working, here are some approaches that might help you solve your problems:

- Look for typos. Remember that JavaScript is a case-sensitive language.
- Use a JavaScript debugger. In Firefox, you can use the JavaScript console, the Venkman Debugger (http://www.mozilla.org/projects/venkman/), or the Firebug add-on (https://addons.mozilla.org/en-US/firefox/addon/1843). In IE, you can use the Microsoft Script Debugger (http://en.wikipedia.org/wiki/Microsoft_Script_Debugger).
- Search the Google Chart API discussion group (http://groups.google.com/group/google-visualization-api). If you can't find a post that answers your question, post your question to the group along with a link to a web page that demonstrates the problem.

Localization

If you expect your chart to be used by people in a variety of countries, you might want to design your chart to be localized for different languages and cultures. The most basic localization is to translate the standard text strings in the UI according to the user's browser settings. A more advanced form of localization would be to change number formats depending on localization, or possibly even UI design. If you decide to localize your chart, list the languages that your chart supports in your documentation, and provide a default setting of a commonly used language. It is also useful to include a "change language" button in the UI of your chart, in case you get the language wrong. The common way to detect browser language is to look at the Accept-Language (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html) HTML header.

上次更新日期：二月 23, 2017