DataTables and DataViews

This page discusses the internal data representation used by charts, the `DataTable` and `DataView` classes used to pass data into a chart, and the various ways of instantiating and populating a `DataTable`.

Contents

How Data is Represented in a Chart

All charts store their data in a table. Here's a simplified representation of a populated two-column data table:

index: 0 **type: string** **label: 'Task'**	**index: 1** **type: number** **label: 'Hours per Day'**
'Work'	11
'Eat'	2
'Commute'	2
'Watch TV'	2
'Sleep'	7

Data is stored in cells referenced as (*row*, *column*), where *row* is a zero-based row index, and *column* is either a zero-based column index or a unique ID that you can specify.

Here is a more complete list of the supported elements and properties of the table; see the Format of the Constructor's JavaScript Literal Parameter (https://google-developers.appspot.com/chart/interactive/docs/reference#dataparam) for more details:

- **Table** - An array of columns and rows, plus an optional map of arbitrary name/value pairs that you can assign. Table-level properties aren't currently used by charts.
- **Columns** - Each column supports a required data type, plus an optional string label, ID, pattern, and map of arbitrary name/value properties. The label is a user-friendly string

that can be displayed by the chart; the ID is an optional identifier that can be used in place of a column index. A column can be referred to in code either by zero-based index, or by the optional ID. See the `DataTable.addColumn()` (https://google-developers.appspot.com/chart/interactive/docs/reference#DataTable_addColumn) for a list of supported data types.

- **Rows** - A row is an array of cells, plus an optional map of arbitrary name/value pairs that you can assign.
- **Cells** - Each cell is an object containing an actual value of the column type, plus an optional string-formatted version of the value that you provide. For example: a numeric column might be assigned the value 7 and the formatted value "seven". If a formatted value is supplied, a chart will use the actual value for calculations and rendering, but might show the formatted value where appropriate, for example if the user hovers over a point. Each cell also has an optional map of arbitrary name/value pairs.

What Table Schema Does My Chart Use?

Different charts use tables in different formats: for example, a pie chart expects a two-column table with a string column and a number column, where each row describes a slice, and the first column is the slize label and the second column is the slice value. A scatter chart, however, expects a table consisting of two numeric columns, where each row is a point, and the two columns are the X and Y values of the point. Read your chart's documentation to learn what data format it requires.

DataTables and DataViews

A chart data table is represented in JavaScript by either a `DataTable` (https://google-developers.appspot.com/chart/interactive/docs/reference#DataTable) object or a `DataView` (https://google-developers.appspot.com/chart/interactive/docs/reference#DataView) object. In some cases, you might see a JavaScript literal or JSON version of a DataTable used, for instance when data is sent over the Internet by a Chart Tools Datasource, or as a possible input value for a `ChartWrapper` (https://google-developers.appspot.com/chart/interactive/docs/reference#chartwrapperobject).

A `DataTable` is used to create the original data table. A `DataView` is a convenience class that provides a read-only view of a `DataTable`, with methods to hide or reorder rows or columns quickly without modifying the linked, original data. Here is a brief comparison of the two classes:

DataTable	DataView
Read/Write	Read-only
Can be created empty and then populated	Is a reference to an existing **`DataTable`**. Cannot be populated from scratch; must be instantiated with a reference to an existing **`DataTable`**.
Data takes storage space.	Data is a reference to an existing **`DataTable`**, and does not consume space.
Can add/edit/delete rows, columns, and data, and all changes are persistent.	Can sort or filter rows without modifying the underlying data. Rows and columns can be hidden and revealed repeatedly.
Can be cloned	Can return a **`DataTable`** version of the view
Is source data; does not contain references	A live reference to a **`DataTable`**; any changes in the **`DataTable`** data is immediately reflected in the view.
Can be passed into a chart as a data source	Can be passed into a chart as a data source
Does not support calculated columns	Supports calculated columns, which are columns with a value calculated on the fly by combining or manipulating other columns.
No row or column hiding	Can hide or show selected columns

Creating and Populating a DataTable

There are several different ways of creating and populating a DataTable:

- Create a new DataTable, then call addColumn()/addRows()/addRow()/setCell() (#emptytable)
- arrayToDataTable() (#arraytodatatable)
- JavaScript literal initializer (#javascriptliteral)
- Sending a Datasource Query (#query)

Empty DataTable + addColumn()/addRows()/addRow()/setCell()

Steps:

1. Instantiate a new `DataTable`

2. Add columns
3. Add one or more rows, optionally populated with data. You can add empty rows and populate them later. You can also add or remove rows additional rows or edit cell values individually.

Advantages:

- You can specify the data type and label of each column.
- Good for generating the table in the browser, and less prone to typos than the JSON literal method.

Disadvantages:

- Not as useful as building a JSON literal string to pass into a DataTable constructor when programmatically generating the page on a web server.
- Depends on the speed of the browser, and can be slower than JSON literal strings with larger tables (about 1,000+ cells).

Examples:

Here are a few examples of creating the same data table using different variations of this technique:

```
// ------- Version 1------------
// Add rows + data at the same time
// ----------------------------
var data = new google.visualization.DataTable();

// Declare columns
data.addColumn('string', 'Employee Name');
data.addColumn('datetime', 'Hire Date');

// Add data.
data.addRows([
  ['Mike', {v:new Date(2008,1,28), f:'February 28, 2008'}], // Example of spe
  ['Bob', new Date(2007,5,1)],                              // More typically
  ['Alice', new Date(2006,7,16)],                           // formatter (https:
  ['Frank', new Date(2007,11,28)],
  ['Floyd', new Date(2005,3,13)],
  ['Fritz', new Date(2011,6,1)]
]);



// ------- Version 2------------
```

```
// Add empty rows, then populate
// ----------------------------
var data = new google.visualization.DataTable();
  // Add columns
  data.addColumn('string', 'Employee Name');
  data.addColumn('date', 'Start Date');

  // Add empty rows
  data.addRows(6);
  data.setCell(0, 0, 'Mike');
  data.setCell(0, 1, {v:new Date(2008,1,28), f:'February 28, 2008'});
  data.setCell(1, 0, 'Bob');
  data.setCell(1, 1, new Date(2007, 5, 1));
  data.setCell(2, 0, 'Alice');
  data.setCell(2, 1, new Date(2006, 7, 16));
  data.setCell(3, 0, 'Frank');
  data.setCell(3, 1, new Date(2007, 11, 28));
  data.setCell(4, 0, 'Floyd');
  data.setCell(4, 1, new Date(2005, 3, 13));
  data.setCell(5, 0, 'Fritz');
  data.setCell(5, 1, new Date(2007, 9, 2));
```

arrayToDataTable()

This helper function creates and populates a `DataTable` using a single call.

Advantages:

- Very simple and readable code executed in the browser.
- You can either explicitly specify the data type of each column, or let Google Charts infer the type from the data passed in.
    - To explicitly specify the data type of a column, specify an object in the header row with the `type` property.
    - To let Google Charts infer the type, use a string for the column label.

Examples:

```
var data = google.visualization.arrayToDataTable([
       ['Employee Name', 'Salary'],
       ['Mike', {v:22500, f:'22,500'}], // Format as "22,500".
       ['Bob', 35000],
       ['Alice', 44000],
       ['Frank', 27000],
       ['Floyd', 92000],
```

```
       ['Fritz', 18500]
      ],
      false); // 'false' means that the first row contains labels, not data.
```

```
var data = google.visualization.arrayToDataTable([
       [ {label: 'Year', id: 'year'},
         {label: 'Sales', id: 'Sales', type: 'number'}, // Use object notatio
         {label: 'Expenses', id: 'Expenses', type: 'number'} ],
       ['2014', 1000, 400],
       ['2015', 1170, 460],
       ['2016', 660, 1120],
       ['2017', 1030, 540]]);
```

JavaScript Literal Initializer

You can pass a JavaScript literal object into your table constructor, defining the table schema and optionally data as well.

Advantages:

- Useful when generating data on your web server.
- Processes faster than other methods for larger tables (about 1,000+ cells)

Disadvantages:

- Syntax is tricky to get right, and prone to typos.
- Not very readable code.
- Temptingly similar, but not identical, to JSON.

Example:

```
var data = new google.visualization.DataTable(
   {
     cols: [{id: 'task', label: 'Employee Name', type: 'string'},
            {id: 'startDate', label: 'Start Date', type: 'date'}],
     rows: [{c:[{v: 'Mike'}, {v: new Date(2008, 1, 28), f:'February 28, 2008'
            {c:[{v: 'Bob'}, {v: new Date(2007, 5, 1)}]},
            {c:[{v: 'Alice'}, {v: new Date(2006, 7, 16)}]},
            {c:[{v: 'Frank'}, {v: new Date(2007, 11, 28)}]},
            {c:[{v: 'Floyd'}, {v: new Date(2005, 3, 13)}]},
            {c:[{v: 'Fritz'}, {v: new Date(2011, 6, 1)}]}
           ]
   }
)
```

Sending a Datasource Query

When you send a query to a Chart Tools Datasource (https://google-developers.appspot.com/chart/interactive/docs/queries), a successful reply is a DataTable instance. This returned DataTable can be copied, modified, or copied into a DataView the same as any other DataTable.

```
function drawVisualization() {
    var query = new google.visualization.Query(
        'http://spreadsheets.google.com/tq?key=pCQbetd-CptGXxxQIG7VFIQ&pub=1'

    // Apply query language statement.
    query.setQuery('SELECT A,D WHERE D > 100 ORDER BY D');

    // Send the query with a callback function.
    query.send(handleQueryResponse);
  }

  function handleQueryResponse(response) {
    if (response.isError()) {
      alert('Error in query: ' + response.getMessage() + ' ' + response.getDe
      return;
    }

    var data = response.getDataTable();
    visualization = new google.visualization.LineChart(document.getElementById
    visualization.draw(data, {legend: 'bottom'});
  }
```

dataTableToCsv()

The helper function `google.visualization.dataTableToCsv(data)` returns a CSV string with the data from the data table.

The input to this function can be either a DataTable or a DataView.

It uses the formatted values of the cells. Column lables are ignored.

Special characters such as "`,`" and "`\n`" are escaped using standard CSV escaping rules.

The following code will display

```
Ramanujan,1729
Gauss,5050
```

in your browser's JavaScript console:

```
<html>
  <head>
  <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js
  <script type="text/javascript">
   google.charts.load("current", {packages:['corechart']});
   google.charts.setOnLoadCallback(drawChart);
   function drawChart() {
      var data = google.visualization.arrayToDataTable([
        ['Name', 'Number'],
        ['Ramanujan', 1729],
        ['Gauss', 5050]
      ]);
    var csv = google.visualization.dataTableToCsv(data);
    console.log(csv);
  }
  </script>
  </head>
</html>
```

More Information

- Querying a Chart Tools datasource (https://google-developers.appspot.com/chart/interactive/docs/queries)
- DataTable JavaScript literal syntax (https://google-developers.appspot.com/chart/interactive/docs/reference#dataparam)
- `DataTable` (https://google-developers.appspot.com/chart/interactive/docs/reference#DataTable) reference
- `DataView` (https://google-developers.appspot.com/chart/interactive/docs/reference#DataView) reference
- `arrayToDataTable()` (https://google-developers.appspot.com/chart/interactive/docs/reference#google.visualization.arraytodatatable)

reference

上次更新日期：二月 8, 2016