

Google Visualization API Reference

This page lists the objects exposed by the Google Visualization API, and the standard methods exposed by all visualizations.

Development Tip

You can download the `google.visualization` 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 provide autocompletion. Note that this file might not always be up to date, so be sure to test your code. You should update this file after each release.

Note: The Google Visualization API namespace is `google.visualization.*`

A Note on Arrays

Some browsers don't properly handle trailing commas in JavaScript arrays, so don't use them. Empty values in the middle of an array are fine. So, for example:

```
data = ['a','b','c', ,]; // BAD
data = ['a','b','c'];    // OK
data = ['a','b', , 'd']; // Also OK. The third value is undefined.
```

DataTable Class

Represents a two-dimensional, mutable table of values. To make a read-only copy of a `DataTable` (optionally filtered to show specific values, rows, or columns), create a [DataView](#) (`#DataView`).

Each column is assigned a data type, plus several optional properties including an ID, label, and pattern string.

In addition, you can assign custom properties (name/value pairs) to any cell, row, column, or the entire table. Some visualizations support specific custom properties; for example the [Table visualization](#)

(<https://developers.google.com/chart/interactive/docs/gallery/table#customproperties>) supports a

cell property called 'style', which lets you assign an inline CSS style string to the rendered table cell. A visualization should describe in its documentation any custom properties that it supports.

See also: [QueryResponse.getDataTable](#) (#QueryResponse_getDataTable)

Constructor

Syntax

`DataTable(opt_data, opt_version)`

opt_data

[*Optional*] Data used to initialize the table. This can either be the JSON returned by calling [DataTable.toJSON\(\)](#) (#DataTable_toJSON) on a populated table, or a JavaScript object containing data used to initialize the table. The structure of the JavaScript literal object is described [here](#) (#dataparam). If this parameter is not supplied, a new, empty data table will be returned.

opt_version

[*Optional*] A numeric value specifying the version of the wire protocol used. This is only used by [Chart Tools Datasource implementors](#) (https://developers.google.com/chart/interactive/docs/dev/implementing_data_source). *The current version is 0.6.*

Details

The `DataTable` object is used to hold the data passed into a visualization. A `DataTable` is a basic two-dimensional table. All data in each column must have the same data type. Each column has a descriptor that includes its data type, a label for that column (which might be displayed by a visualization), and an ID, which can be used to refer to a specific column (as an alternative to using column indexes). The `DataTable` object also supports a map of arbitrary properties assigned to a specific value, a row, a column, or the whole `DataTable`. Visualizations can use these to support additional features; for example, the [Table visualization](#) (<https://developers.google.com/chart/interactive/docs/gallery/table#customproperties>) uses custom properties to let you assign arbitrary class names or styles to individual cells.

Each cell in the table holds a value. Cells can have a null value, or a value of the type specified by its column. Cells optionally can take a "formatted" version of the data; this is a string version of the data, formatted for display by a visualization. A visualization can (but is not required to) use the formatted version for display, but will always use the data itself for

any sorting or calculations that it makes (such as determining where on a graph to place a point). An example might be assigning the values "low" "medium", and "high" as formatted values to numeric cell values of 1, 2, and 3.

To add data rows after calling the constructor, you can call either `addRow()` (`#addrow`) for a single row, or `addRows()` (`#DataTable_addRows`) for multiple rows. You can add columns as well by calling the `addColumn()` (`#DataTable_addColumn`) methods. There are removal methods for rows and columns as well, but rather than removing rows or columns, consider creating a `DataView` that is a selective view of the `DataTable`.

If you change values in a `DataTable` after it is passed into a visualization's `draw()` method, the changes will not immediately change the chart. You must call `draw()` again to reflect any changes.

Note: Google Charts does not perform any validation on datatables. (If it did, charts would be slower to render.) If you provide a number where your column is expecting a string, or vice versa, Google Charts will do its level best to interpret the value in a way that makes sense, but will not flag mistakes.

Examples

The following example demonstrates instantiating and populating a `DataTable` with a literal string, with the same data as shown in the JavaScript example above:

```
var dt = new google.visualization.DataTable({
  cols: [{id: 'task', label: 'Task', type: 'string'},
        {id: 'hours', label: 'Hours per Day', type: 'number'}],
  rows: [{c:[{v: 'Work'}, {v: 11}]},
        {c:[{v: 'Eat'}, {v: 2}]},
        {c:[{v: 'Commute'}, {v: 2}]},
        {c:[{v: 'Watch TV'}, {v:2}]},
        {c:[{v: 'Sleep'}, {v:7, f:'7.000'}]}]
}, 0.6);
```

The following example creates a new, empty `DataTable` and then populates it manually with the same data as above:

```
var data = new google.visualization.DataTable();
data.addColumn('string', 'Task');
data.addColumn('number', 'Hours per Day');
data.addRows([
  ['Work', 11],
  ['Eat', 2],
  ['Commute', 2],
  ['Watch TV', 2],
```

```
['Sleep', {v:7, f:'7.000'}]  
]);
```

Should I create my DataTable in JavaScript or object literal notation?

You can create a **DataTable** either by calling the constructor without parameters and then adding values by calling the [addColumn\(\)](#) (`#DataTable_addColumn`)/ [addRows\(\)](#) (`#DataTable_addRows`) methods listed below, or by passing in a populated JavaScript literal object to initialize it. Both methods are described below. Which one should you use?

- Creating and populating a table in JavaScript by calling [addColumn\(\)](#), [addRow\(\)](#), and [addRows\(\)](#) is very readable code. This method is useful when you'll be entering code by hand. It is slower than using object literal notation (described next), but in smaller tables (say, 1,000 cells) you probably won't notice much difference.
- Direct declaration of the **DataTable** object using object-literal notation is considerably faster in large tables. However, it can be a tricky syntax to get right; use this if you can generate the object literal syntax in code, which reduces possibility of errors.

Methods

Method	Return Value	Description
<code>addColumn(type, opt_label, opt_id)</code> OR <code>addColumn(description_object)</code>	Number	<p>Adds a new column to the data table, and returns the index of the new column. All the cells of the new column are assigned a null value. This method has two signatures:</p> <p>First signature has the following parameters:</p> <ul style="list-style-type: none">• type - A string with the data type of the values of the column. The type can be one of the following: <code>'string'</code>, <code>'number'</code>, <code>'boolean'</code>, <code>'date'</code>, <code>'datetime'</code>, and <code>'timeofday'</code>.• opt_label - [<i>Optional</i>] A string with the label of the column.

		<p>The column label is typically displayed as part of the visualization, for example as a column header in a table, or as a legend label in a pie chart. If not value is specified, an empty string is assigned.</p> <ul style="list-style-type: none"> • opt_id - [<i>Optional</i>] A string with a unique identifier for the column. If not value is specified, an empty string is assigned. <p>Second signature has a single object parameter with the following members:</p> <ul style="list-style-type: none"> • type - A string describing the column data type. Same values as type above. • label - [<i>Optional, string</i>] A label for the column. • id - [<i>Optional, string</i>] An ID for the column. • role - [<i>Optional, string</i>] A <u>role</u> (https://developers.google.com/chart/interactive/docs/roles) for the column. • pattern - [<i>Optional, string</i>] A number (or date) format string specifying how to display the column value. <p>See also: <u>getColumnId</u> (#DataTable_getColumnId), <u>getColumnLabel</u> (#DataTable_getColumnLabel), <u>getColumnType</u> (#DataTable_getColumnType), <u>insertColumn</u> (#DataTable_insertColumn), <u>getColumnRole</u> (#DataTable_getColumnRole)</p>
addRow(opt_cellArray)	Number	Adds a new row to the data table, and returns the index of the new

		<p>row.</p> <ul style="list-style-type: none"> • opt_cellArray [<i>optional</i>] A row object, in JavaScript notation, specifying the data for the new row. If this parameter is not included, this method will simply add a new, empty row to the end of the table. This parameter is an array of cell values: if you only want to specify a value for a cell, just give the cell value (e.g. 55 or 'hello'); if you want to specify a formatted value and/or properties for the cell, use a <u>cell object</u> (<code>#cell_object</code>) (e.g., <code>{v:55, f:'Fifty-five'}</code>). You can mix simple values and cell objects in the same method call). Use null or an empty array entry for an empty cell. <p>Examples:</p> <pre>data.addRow(); // Add an data.addRow(['Hermione', n // Add a row with two cell data.addRow(['Hermione', {]); data.addRow(['Col1Val', nu data.addRow(['Col1Val', ,</pre>
addRows(numOrArray)	Number	<p>Adds new rows to the data table, and returns the index of the last added row. You can call this method to create new empty rows, or with data used to populate the rows, as described below.</p> <ul style="list-style-type: none"> • numOrArray - Either a number or an array: <ul style="list-style-type: none"> • Number - A number specifying how many

		<p>new, unpopulated rows to add.</p> <ul style="list-style-type: none"> • Array - An array of <u>row</u> (<code>#rowsproperty</code>) objects used to populate a set of new rows. Each row is an object as described in <code>addRow()</code>. Use <code>null</code> or an empty array entry for an empty cell. <p>Example:</p> <pre>data.addRow([['Ivan', new Date(1977,2 ['Igor', new Date(1962,7 ['Felix', new Date(1983, ['Bob', null] // No date]);</pre> <p>See also: <u>insertRows</u> (<code>#DataTable_insertRows</code>)</p>
<code>clone()</code>	<u>DataTable</u> (<code>#DataTable</code>)	<p>Returns a clone of the data table. The result is a deep copy of the data table except for the <u>cell properties</u> (<code>#DataTable_getProperty</code>), <u>row properties</u> (<code>#DataTable_getRowProperty</code>), <u>table properties</u> (<code>#DataTable_getTableProperty</code>) and <u>column properties</u> (<code>#DataTable_getColumnProperty</code>), which are shallow copies; this means that non-primitive properties are copied by reference, but primitive properties are copied by value.</p>
<code>getColumnId(columnIndex)</code>	String	<p>Returns the identifier of a given column specified by the column index in the underlying table. For data tables that are retrieved by queries, the column identifier is set by the data source, and can be used to refer to columns when using the <u>query language</u></p>

		<p>(https://developers.google.com/chart/interactive/docs/querylanguage)</p> <p>.</p> <p>See also: Query.setQuery (#Query_setQuery)</p>
getColumnLabel(columnIndex)	String	<p>Returns the label of a given column specified by the column index in the underlying table. The column label is typically displayed as part of the visualization. For example the column label can be displayed as a column header in a table, or as the legend label in a pie chart. For data tables that are retrieved by queries, the column label is set by the data source, or by the label clause of the query language (https://developers.google.com/chart/interactive/docs/querylanguage#Label)</p> <p>.</p> <p>See also: setColumnLabel (#DataTable_setColumnLabel)</p>
getColumnPattern(columnIndex)	String	<p>Returns the formatting pattern used to format the values of the specified column.</p> <ul style="list-style-type: none"> • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. <p>For data tables that are retrieved by queries, The column pattern is set by the data source, or by the format clause of the query language. An example of a pattern is '#,##0.00'. For more on patterns see the query language reference</p>

		<p>(https://developers.google.com/chart/interactive/docs/querylanguage#Format)</p> <p>.</p>
getColumnProperties(columnIndex)	Object	<p>Returns a map of all properties for the specified column. Note that the properties object is returned by reference, so changing values in the retrieved object changes them in the DataTable.</p> <ul style="list-style-type: none"> • columnIndex is the numeric index of the column to retrieve properties for.
getColumnProperty(columnIndex, name)	Auto	<p>Returns the value of a named property, or null if no such property is set for the specified column. The return type varies, depending on the property.</p> <ul style="list-style-type: none"> • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • name is the property name, as a string. <p>See also: setColumnProperty (#DataTable_setColumnProperty) setColumnProperties (#DataTable_setColumnProperties)</p>
getColumnRange(columnIndex)	Object	<p>Returns the minimal and maximal values of values in a specified column. The returned object has properties min and max. If the range has no values, min and max will contain null.</p> <p>columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the</p>

		getNumberOfColumns() method.
getColumnRole(columnIndex)	String	Returns the <u>role</u> (https://developers.google.com/chart/interactive/docs/roles) of the specified column.
getColumnType(columnIndex)	String	Returns the type of a given column specified by the column index. <ul style="list-style-type: none"> columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. <p>The returned column type can be one of the following: 'string', 'number', 'boolean', 'date', 'datetime', and 'timeofday'</p>
getDistinctValues(columnIndex)	Array of objects	Returns the unique values in a certain column, in ascending order. <ul style="list-style-type: none"> columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. <p>The type of the returned objects is the same as that returned by the <u>getValue</u> (#DataTable_getValue) method.</p>
getFilteredRows(filters)	Array of objects	Returns the row indexes for rows that match all of the given filters. The indexes are returned in ascending order. The output of this method can be used as input to <u>DataRow.setRows()</u> (#DataRow_setRows) to change the displayed set of rows in a visualization.

filters - An array of objects that describe an acceptable cell value. A row index is returned by this method if it matches *all* of the given filters. Each filter is an object with a numeric **column** property that specifies the index of the column in the row to assess, plus one of the following:

- A **value** property with a value that must be matched exactly by the cell in the specified column. The value must be the same type as the column; or
- One or both of the following properties, the same type as the column being filtered:
 - **minValue** - A minimum value for the cell. The cell value in the specified column must be greater than or equal to this value.
 - **maxValue** - A maximum value for the cell. The cell value in the specified column must be less than or equal to this value.

A null or undefined value for **minValue** (or **maxValue**) means that the lower (or upper) bound of the range will not be enforced.

Another optional property, **test**, specifies a function to be combined with value or range filtering. The function is called with the cell value, the row and column indices, and the datatable. It should return false if the row should be excluded from the result.

		<p>Example:</p> <p><code>getFilteredRows([{column: 3, value: 42}, {column: 2, minValue: 'bar', maxValue: 'foo' }])</code> returns an array containing, in ascending order, the indexes of all rows for which the fourth column (column index 3) is exactly 42, and the third column (column index 2) is between 'bar' and 'foo' (inclusive).</p>
<code>getFormattedValue(rowIndex, columnIndex)</code>	String	<p>Returns the formatted value of the cell at the given row and column indexes.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the <code>getNumberOfRows()</code> method. • ColumnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the <code>getNumberOfColumns()</code> method. <p>For more on formatting column values see the query language reference (https://developers.google.com/chart/interactive/docs/querylanguage#Format).</p> <p>See also: setFormattedValue (#DataTable_setFormattedValue)</p>
<code>getNumberOfColumns()</code>	Number	Returns the number of columns in the table.
<code>getNumberOfRows()</code>	Number	Returns the number of rows in the table.
<code>getProperties(rowIndex, columnIndex)</code>	Object	Returns a map of all the properties for the specified cell. Note that the properties object is returned by

		<p>reference, so changing values in the retrieved object changes them in the DataTable.</p> <ul style="list-style-type: none"> • rowIndex is the cell's row index. • columnIndex is the cell's column index.
getProperty(rowIndex, columnIndex, name)	Auto	<p>Returns the value of a named property, or null if no such property is set for the specified cell. The return type varies, depending on the property.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • name is a string with the property name. <p>See also: setCell (#DataTable_setCell) setProperties (#DataTable_setProperties) setProperty (#DataTable_setProperty)</p>
getRowProperties(rowIndex)	Object	<p>Returns a map of all properties for the specified row. Note that the properties object is returned by reference, so changing values in the retrieved object changes them in the DataTable.</p> <ul style="list-style-type: none"> • rowIndex is the index of the row to retrieve properties for.
getRowProperty(rowIndex, name)	Auto	<p>Returns the value of a named property, or null if no such</p>

		<p>property is set for the specified row. The return type varies, depending on the property.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • name is a string with the property name. <p>See also: setRowProperty (#DataTable_setRowProperty) setRowProperties (#DataTable_setRowProperties)</p>
getSortedRows(sortColumns)	Array of numbers	<p>Returns a sorted version of the table without modifying the order of the underlying data. To permanently sort the underlying data, call sort() (#DataTable_sort). You can specify sorting in a number of ways, depending on the type you pass in to the sortColumns parameter:</p> <ul style="list-style-type: none"> • A single number specifies the index of the column to sort by. Sorting will be in ascending order. Example: sortColumns(3) will sort by the 4th column, in ascending order. • A single object that contains the number of the column index to sort by, and an optional boolean property desc. If desc is set to true, the specific column will be sorted in descending order; otherwise, sorting is in ascending order. Examples: sortColumns({column: 3}) will sort by the 4th column, in ascending order;

`sortColumns({column: 3, desc: true})` will sort by the 4th column, in descending order.

- **An array of numbers** of the column indexes by which to sort. The first number is the primary column by which to sort, the second one is the secondary, and so on. This means that when two values in the first column are equal, the values in the next column are compared, and so on. **Example:** `sortColumns([3, 1, 6])` will sort first by the 4th column (in ascending order), then by the 2nd column (in ascending order), and then by the 7th column (in ascending order).
- **An array of objects**, each one with the number of the column index to sort by, and an optional boolean property `desc`. If `desc` is set to true, the specific column will be sorted in descending order (the default is ascending order). The first object is the primary column by which to sort, the second one is the secondary, and so on. This means that when two values in the first column are equal, the values in the next column are compared, and so on. **Example:** `sortColumn([{column: 3}, {column: 1, desc: true}, {column: 6, desc: true}])` will sort first by the 4th column (in ascending order), then column 2 in descending order, and then column 7 in descending order.

The returned value is an array of numbers, each number is an index of a **DataTable** row. Iterating on the **DataTable** rows by the order

of the returned array will result in rows ordered by the specified **sortColumns**. The output can be used as input to **[DataView.setRows\(\)](#)** (**[#DataView_setRows](#)**) to quickly change the displayed set of rows in a visualization.

Note that the sorting is guaranteed to be stable: this means that if you sort on equal values of two rows, the same sort will return the rows in the same order every time.
See also: **[sort](#)** (**[#DataTable_sort](#)**)

Example: To iterate on rows ordered by the third column, use:

```
var rowInds = data.getSort
for (var i = 0; i < rowInds.length; i++) {
    var v = data.getValue(rowInds[i], 2)
}
```

getTableProperties	Object	Returns a map of all properties for the table.
getTableProperty(name)	Auto	<p>Returns the value of a named property, or null if no such property is set for the table. The return type varies, depending on the property.</p> <ul style="list-style-type: none">• name is a string with the property name. <p>See also: <u>setTableProperties</u> (<u>#DataTable_setTableProperties</u>) <u>setTableProperty</u> (<u>#DataTable_setTableProperty</u>)</p>
getValue(rowIndex, columnIndex)	Object	<p>Returns the value of the cell at the given row and column indexes.</p> <ul style="list-style-type: none">• rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the

		<p>getNumberOfRows() method.</p> <ul style="list-style-type: none"> columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. <p>The type of the returned value depends on the column type (see <u>getColumnType</u> (#DataTable_getColumnType)):</p> <ul style="list-style-type: none"> If the column type is 'string', the value is a string. If the column type is 'number', the value is a number. If the column type is 'boolean', the value is a boolean. If the column type is 'date' or 'datetime', the value is a Date object. If the column type is 'timeofday', the value is an array of four numbers: [hour, minute, second, milliseconds]. If the cell value is a null value, it returns null.
<p>insertColumn(columnIndex, None type [,label [,id]])</p>		<p>Inserts a new column to the data table, at the specifid index. All existing columns at or after the specified index are shifted to a higher index.</p> <ul style="list-style-type: none"> columnIndex is a number with the required index of the new column. type should be a string with the data type of the values of the column. The type can be one of the following: 'string', 'number', 'boolean', 'date',

		<p>'datetime', and 'timeofday'.</p> <ul style="list-style-type: none"> • label should be a string with the label of the column. The column label is typically displayed as part of the visualization, for example as a column header in a table, or as a legend label in a pie chart. If no value is specified, an empty string is assigned. • id should be a string with a unique identifier for the column. If no value is specified, an empty string is assigned. <p>See also: addColumn (#DataTable_addColumn)</p>
insertRows(rowIndex, numberOrArray)	None	<p>Insert the specified number of rows at the specified row index.</p> <ul style="list-style-type: none"> • rowIndex is the index number where to insert the new row(s). Rows will be added, starting at the index number specified. • numberOrArray is either a number of new, empty rows to add, or an array of one or more populated rows to add at the index. See addRows() (#DataTable_addRows) for the syntax for adding an array of row objects. <p>See also: addRows (#DataTable_addRows)</p>
removeColumn(columnIndex)	None	<p>Removes the column at the specified index.</p> <ul style="list-style-type: none"> • columnIndex should be a number with a valid column index. <p>See also: removeColumns (#DataTable_removeColumns)</p>
removeColumns(None	<p>Removes the specified number of</p>

<code>columnIndex, numberOfColumns)</code>		<p>columns starting from the column at the specified index.</p> <ul style="list-style-type: none"> • numberOfColumns is the number of columns to remove. • columnIndex should be a number with a valid column index. <p>See also: removeColumn (#DataTable_removeColumn)</p>
<code>removeRow(rowIndex)</code>	None	<p>Removes the row at the specified index.</p> <ul style="list-style-type: none"> • rowIndex should be a number with a valid row index. <p>See also: removeRows (#DataTable_removeRows)</p>
<code>removeRows(rowIndex, numberOfRows)</code>	None	<p>Removes the specified number of rows starting from the row at the specified index.</p> <ul style="list-style-type: none"> • numberOfRows is the number of rows to remove. • rowIndex should be a number with a valid row index. <p>See also: removeRow (#DataTable_removeRow)</p>
<code>setCell(rowIndex, columnIndex [, value [, formattedValue [, properties]]])</code>	None	<p>Sets the value, formatted value, and/or properties, of a cell.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method.

- **value** [*Optional*] is the value assigned to the specified cell. To avoid overwriting this value, set this parameter to **undefined**; to clear this value, set it to **null**. The type of the value depends on the column type (see [getColumnType\(\)](#) (#DataTable_getColumnType)):
 - If the column type is 'string', the value should be a string.
 - If the column type is 'number', the value should be a number.
 - If the column type is 'boolean', the value should be a boolean.
 - If the column type is 'date' or 'datetime', the value should be a Date object.
 - If the column type is 'timeofday', the value should be an array of four numbers: [hour, minute, second, milliseconds].
- **formattedValue** [*Optional*] is a string with the value formatted as a string. To avoid overwriting this value, set this parameter to **undefined**; to clear this value and have the API apply default formatting to **value** as needed, set it to **null**; to explicitly set an empty formatted value, set it to an empty string. The formatted value is typically used by visualizations to display value labels. For example the formatted value can appear as a label text within a pie chart.
- **properties** [*Optional*] is an **Object** (a name/value map)

		<p>with additional properties for this cell. To avoid overwriting this value, set this parameter to undefined; to clear this value, set it to null. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p> <p>See also: setValue() (#DataTable_setValue), setFormattedValue() (#DataTable_setFormattedValue), setProperty() (#DataTable_setProperty), setProperties() (#DataTable_setProperties).</p>
setColumnLabel(columnIndex, label)	None	<p>Sets the label of a column.</p> <ul style="list-style-type: none"> • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • label is a string with the label to assign to the column. The column label is typically displayed as part of the visualization. For example the column label can be displayed as a column header in a table, or as the legend label in a pie chart. <p>See also: getColumnLabel (#DataTable_getColumnLabel)</p>
setColumnProperty(columnIndex, name, value)	None	<p>Sets a single column property. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p>

		<ul style="list-style-type: none"> • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • name is a string with the property name. • value is a value of any type to assign to the specified named property of the specified column. <p>See also: setColumnProperties (#DataTable_setColumnProperties) getColumnProperty (#DataTable_getColumnProperty)</p>
setColumnProperties(columnIndex, properties)	None	<p>Sets multiple column properties. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p> <ul style="list-style-type: none"> • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • properties is an Object (name/value map) with additional properties for this column. If null is specified, all additional properties of the column will be removed. <p>See also: setColumnProperty (#DataTable_setColumnProperty) getColumnProperty (#DataTable_getColumnProperty)</p>
setFormattedValue(rowIndex, columnIndex, formattedValue)	None	<p>Sets the formatted value of a cell.</p>

		<ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • formattedValue is a string with the value formatted for display. To clear this value and have the API apply default formatting to the cell value as needed, set it <i>formattedValue</i> null; to explicitly set an empty formatted value, set it to an empty string. <p>See also: getFormattedValue (#DataTable_getFormattedValue)</p>
setProperty(rowIndex, columnIndex, name, value)	None	<p>Sets a cell property. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • name is a string with the property name.

		<ul style="list-style-type: none"> • value is a value of any type to assign to the specified named property of the specified cell. <p>See also: setCell (#DataTable_setCell) setProperties (#DataTable_setProperties) getProperty (#DataTable_getProperty)</p>
setProperties(rowIndex, columnIndex, properties)	None	<p>Sets multiple cell properties. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. • properties is an Object (name/value map) with additional properties for this cell. If null is specified, all additional properties of the cell will be removed. <p>See also: setCell (#DataTable_setCell) setProperty (#DataTable_setProperty) getProperty (#DataTable_getProperty)</p>
setRowProperty(rowIndex, name, value)	None	<p>Sets a row property. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization</p>

		<p>documentation to see what properties are supported.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • name is a string with the property name. • value is a value of any type to assign to the specified named property of the specified row. <p>See also: setRowProperties (#DataTable_setRowProperties) getRowProperty (#DataTable_getRowProperty)</p>
setRowProperties(rowIndex, properties)	None	<p>Sets multiple row properties. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method. • properties is an Object (name/value map) with additional properties for this row. If null is specified, all additional properties of the row will be removed. <p>See also: setRowProperty (#DataTable_setRowProperty) getRowProperty (#DataTable_getRowProperty)</p>
setTableProperty(name, value)	None	<p>Sets a single table property. Some visualizations support table, row,</p>

		<p>column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p> <ul style="list-style-type: none"> • name is a string with the property name. • value is a value of any type to assign to the specified named property of the table. <p>See also: setTableProperties (#DataTable_setTableProperties) getTableProperty (#DataTable_getTableProperty)</p>
setTableProperties(properties)	None	<p>Sets multiple table properties. Some visualizations support table, row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported.</p> <ul style="list-style-type: none"> • properties is an Object (name/value map) with additional properties for the table. If null is specified, all additional properties of the table will be removed. <p>See also: setTableProperty (#DataTable_setTableProperty) getTableProperty (#DataTable_getTableProperty)</p>
setValue(rowIndex, columnIndex, value)	None	<p>Sets the value of a cell. In addition to overwriting any existing cell value, this method will also clear out any formatted value and properties for the cell.</p> <ul style="list-style-type: none"> • rowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method.

- **columnIndex** should be a number greater than or equal to zero, and less than the number of columns as returned by the **getNumberOfColumns()** method. This method does not let you set a formatted value for this cell; to do that, call **setFormattedValue()** (`#DataTable_setFormattedValue`).
- **value** is the value assigned to the specified cell. The type of the returned value depends on the column type (see **getColumnType** (`#DataTable_getColumnType`)):
 - If the column type is 'string', the value should be a string.
 - If the column type is 'number', the value should be a number.
 - If the column type is 'boolean', the value should be a boolean.
 - If the column type is 'date' or 'datetime', the value should be a Date object.
 - If the column type is 'timeofday', the value should be an array of four numbers: [hour, minute, second, milliseconds].
 - For any column type, the value can be set to **null**.

See also: **setCell** (`#DataTable_setCell`), **setFormattedValue** (`#DataTable_setFormattedValue`), **setProperty** (`#DataTable_setProperty`),

		<u>setProperties</u> (#DataTable_setProperties)
sort(sortColumns)	None	<p>Sorts the rows, according to the specified sort columns. The DataTable is modified by this method. See <u>getSortedRows()</u> (#DataTable_getSortedRows) for a description of the sorting details. This method does not return the sorted data.</p> <p>See also: <u>getSortedRows</u> (#DataTable_getSortedRows)</p> <p>Example: To sort by the third column and then by the second column, use: <code>data.sort([{column: 2}, {column: 1}])</code>;</p>
toJSON()	String	<p>Returns a JSON representation of the DataTable that can be passed into the DataTable constructor (#DataTable_literal_constructor_example). For example:</p> <pre>{ "cols": [{ "id": "Col1", "label": "rows": [{ "c": [{ "v": "a" }, { "v": "D { "c": [{ "v": "b" }, { "v": "D] } }</pre>

Format of the Constructor's JavaScript Literal *data* Parameter

You can initialize a **DataTable** by passing a JavaScript string literal object into the *data* parameter. We'll call this object the *data* object. You can code this object by hand, according to the description below, or you can use a helper Python library (https://developers.google.com/chart/interactive/docs/dev/gviz_api_lib) if you know how to use Python, and your site can use it. However, if you want to construct the object by hand, this section will describe the syntax.

First, let's show an example of a simple JavaScript object describing a table with three rows and three columns (String, Number, and Date types):

```

{
  cols: [{id: 'A', label: 'NEW A', type: 'string'},
        {id: 'B', label: 'B-label', type: 'number'},
        {id: 'C', label: 'C-label', type: 'date'}
  ],
  rows: [{c:[{v: 'a'},
             {v: 1.0, f: 'One'},
             {v: new Date(2008, 1, 28, 0, 31, 26), f: '2/28/08 12:31 AM'}
          ]},
        {c:[{v: 'b'},
             {v: 2.0, f: 'Two'},
             {v: new Date(2008, 2, 30, 0, 31, 26), f: '3/30/08 12:31 AM'}
          ]},
        {c:[{v: 'c'},
             {v: 3.0, f: 'Three'},
             {v: new Date(2008, 3, 30, 0, 31, 26), f: '4/30/08 12:31 AM'}
          ]}
  ],
  p: {foo: 'hello', bar: 'world!'}
}

```

Now let's describe the syntax:

The *data* object consists of two required top-level properties, `cols` and `rows`, and an optional `p` property that is a map of arbitrary values.

Note: All property names and string constants shown are case-sensitive. Also, properties described as taking a string value should have their value enclosed in quotation marks. For example, if you wish to specify the type property as being number, it would be expressed like this: `type: 'number'` but the value itself, as numeric, would be expressed like this: `v: 42`

cols Property

`cols` is an array of objects describing the ID and type of each column. Each property is an object with the following properties (case-sensitive):

- **type** [Required] Data type of the data in the column. Supports the following string values (examples include the `v:` property, described later):
 - 'boolean' - JavaScript boolean value ('true' or 'false'). *Example value:* `v: 'true'`
 - 'number' - JavaScript number value. *Example values:* `v: 7`, `v: 3.14`, `v: -55`
 - 'string' - JavaScript string value. *Example value:* `v: 'hello'`
 - 'date' - JavaScript Date object (zero-based month), with the time truncated. *Example value:* `v: new Date(2008, 0, 15)`

- 'datetime' - JavaScript Date object including the time. *Example value:* `v: new Date(2008, 0, 15, 14, 30, 45)`
- 'timeofday' - Array of three numbers and an optional fourth, representing hour (0 indicates midnight), minute, second, and optional millisecond. *Example values:* `v: [8, 15, 0], v: [6, 12, 1, 144]`
- **id** [Optional] String ID of the column. Must be unique in the table. Use basic alphanumeric characters, so the host page does not require fancy escapes to access the column in JavaScript. Be careful not to choose a JavaScript keyword. *Example:* `id: 'col_1'`
- **label** [Optional] String value that some visualizations display for this column. *Example:* `label: 'Height'`
- **pattern** [Optional] String pattern that was used by a data source to format numeric, date, or time column values. This is for reference only; you probably won't need to read the pattern, and it isn't required to exist. The Google Visualization client does not use this value (it reads the cell's formatted value). If the **DataTable** has come from a data source in response to a query with a [format](https://developers.google.com/chart/interactive/docs/querylanguage#Format) (<https://developers.google.com/chart/interactive/docs/querylanguage#Format>) clause, the pattern you specified in that clause will probably be returned in this value. The recommended pattern standards are the ICU [DecimalFormat](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/DecimalFormat.html) (<http://icu-project.org/apiref/icu4j/com/ibm/icu/text/DecimalFormat.html>) and [SimpleDateFormat](http://icu-project.org/apiref/icu4j/com/ibm/icu/text/SimpleDateFormat.html) (<http://icu-project.org/apiref/icu4j/com/ibm/icu/text/SimpleDateFormat.html>).
- **p** [Optional] An object that is a map of custom values applied to the cell. These values can be of any JavaScript type. If your visualization supports any cell-level properties, it will describe them; otherwise, this property will be ignored. **Example:** `p: {style: 'border: 1px solid green;'}`.

cols Example

```
cols: [{id: 'A', label: 'NEW A', type: 'string'},
       {id: 'B', label: 'B-label', type: 'number'},
       {id: 'C', label: 'C-label', type: 'date'}]
```

rows Property

The **rows** property holds an array of row objects.

Each row object has one required property called **c**, which is an array of cells in that row. It also has an optional **p** property that defines a map of arbitrary custom values to assign to

the whole row. If your visualization supports any row-level properties it will describe them; otherwise, this property will be ignored.

Cell Objects

Each cell in the table is described by an object with the following properties:

- **v** [Optional] The cell value. The data type should match the column data type. If the cell is null, the **v** property should be null, though it can still have **f** and **p** properties.
- **f** [Optional] A string version of the **v** value, formatted for display. Typically the values will match, though they do not need to, so if you specify `Date(2008, 0, 1)` for **v**, you should specify "January 1, 2008" or some such string for this property. This value is not checked against the **v** value. The visualization will not use this value for calculation, only as a label for display. If omitted, a string version of **v** will be automatically generated using the default formatter. The **f** values can be modified using your own formatter, or set with `setFormattedValue()` or `setCell()`, or retrieved with `getFormattedValue()`.
- **p** [Optional] An object that is a map of custom values applied to the cell. These values can be of any JavaScript type. If your visualization supports any cell-level properties, it will describe them. These properties can be retrieved by the `getProperty()` and `getProperties()` methods. **Example:** `p:{style: 'border: 1px solid green;'}`.

Cells in the row array should be in the same order as their column descriptions in `cols`. To indicate a null cell, you can specify `null`, leave a blank for a cell in an array, or omit trailing array members. So, to indicate a row with null for the first two cells, you could specify `[, , {cell_val}]` or `[null, null, {cell_val}]`.

Here is a sample table object with three columns, filled with three rows of data:

```
{
  cols: [{id: 'A', label: 'NEW A', type: 'string'},
          {id: 'B', label: 'B-label', type: 'number'},
          {id: 'C', label: 'C-label', type: 'date'}
  ],
  rows: [{c:[{v: 'a'},
              {v: 1.0, f: 'One'},
              {v: new Date(2008, 1, 28, 0, 31, 26), f: '2/28/08 12:31 AM'}
            ]},
          {c:[{v: 'b'},
              {v: 2.0, f: 'Two'},
              {v: new Date(2008, 2, 30, 0, 31, 26), f: '3/30/08 12:31 AM'}
            ]},
          {c:[{v: 'c'},
              {v: 3.0, f: 'Three'},
              {v: null, f: null}
            ]}
  ]
}
```

```
        {v: new Date(2008, 3, 30, 0, 31, 26), f: '4/30/08 12:31 AM'}
    ]}
  ]
}
```

p Property

The table-level **p** property is a map of custom values applied to the whole **DataTable**. These values can be of any JavaScript type. If your visualization supports any datatable-level properties, it will describe them; otherwise, this property will be available for application use. **Example:** **p**: {className: 'myDataTable'}.

DataView Class

A read-only view of an underlying **DataTable** (**#DataTable**). A **DataView** allows selection of only a subset of the columns and/or rows. It also allows reordering columns/rows, and duplicating columns/rows.

A view is a live window on the underlying **DataTable**, not a static snapshot of data. However, you still must be careful when changing the *structure* of the table itself, as described here:

- Adding or removing *columns* from the underlying table will not be reflected by the view, and might cause unexpected behavior in the view; you will have to create a new **DataView** from the **DataTable** to pick up these changes.
- Adding or removing *rows* from the underlying table is safe and changes will be propagated to the view immediately (but you must call **draw()** on any visualizations after this change to have the new row set rendered). Note that if your view has filtered out rows by calling one of the **setRows()** or **hideRows()** methods, and you add or remove rows from the underlying table, the behavior is unexpected; you must create a new **DataView** to reflect the new table.
- Changing *cell values* in existing cells is safe, and changes are immediately propagated to the **DataView** (but you must call **draw()** on any visualizations after this change to have the new cell values rendered).

It is also possible to create a **DataView** from another **DataView**. Note that whenever an *underlying table or view* is mentioned, it refers to the level immediately below this level. In other words, it refers to the data object used to construct this **DataView**.

DataView also supports calculated columns; these are columns whose value is calculated on the fly using a function that you supply. So, for example, you can include a column that is

a sum of two preceding columns, or a column that calculates and shows the calendar quarter of a date from another column. See [setColumns\(\)](#) (#DataView_setColumns) for more details.

When you modify a `DataView` by hiding or showing rows or columns, the visualization will not be affected until you call `draw()` on the visualization again.

You can combine `DataView.getFilteredRows()` with `DataView.setRows()` to create a `DataView` with an interesting subset of data, as shown here:

```
var data = new google.visualization.DataTable();
data.addColumn('string', 'Employee Name');
data.addColumn('date', 'Start Date');
data.addRows(6);
data.setCell(0, 0, 'Mike');
data.setCell(0, 1, new Date(2008, 1, 28));
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));

// Create a view that shows everyone hired since 2007.
var view = new google.visualization.DataView(data);
view.setRows(view.getFilteredRows([{column: 1, minValue: new Date(2007, 0, 1)}]);
var table = new google.visualization.Table(document.getElementById('test_data'));
table.draw(view, {sortColumn: 1});
```

Constructors

There are two ways to create a new `DataView` instance:

Constructor 1

```
var myView = new google.visualization.DataView(data)
```

data

A **DataTable** or **DataRowView** used to initialize the view. By default, the view contains all the columns and rows in the underlying data table or view, in the original order. To hide or show rows or columns in this view, call the appropriate `set...()` or `hide...()` methods.

See also:

[setColumns\(\)](#) (#DataRowView_setColumns), [hideColumns\(\)](#) (#DataRowView_hideColumns), [setRows\(\)](#) (#DataRowView_setRows), [hideRows\(\)](#) (#DataRowView_hideRows).

Constructor 2

This constructor creates a new **DataRowView** by assigning a serialized **DataRowView** to a **DataTable**. It helps you recreate the **DataRowView** that you serialized using **DataRowView.toJSON()**.

```
var myView = google.visualization.DataRowView.fromJSON(data, viewAsJson)
```

data

The **DataTable** object that you used to create the **DataRowView**, on which you called **DataRowView.toJSON()**. If this table is any different from the original table, you will get unpredictable results.

viewAsJson

The JSON string returned by **DataRowView.toJSON()** (#DataRowView_toJSON). This is a description of which rows to show or hide from the *data* **DataTable**.

Methods

Method	Return Value	Description
<ul style="list-style-type: none">• getColumnId (#DataRowView_getColumnId)• getColumnLabel (#DataRowView_getColumnLabel)• getColumnPattern (#DataRowView_getColumnPattern)	See descriptions in DataTable .	Same as the equivalent DataTable methods, except that row/column indexes refer to the index in the view and not in the underlying table/view.

<ul style="list-style-type: none"> • <u>getColumnProperty</u> (#DataTable_getColumnProperty) • <u>getColumnRange</u> (#DataTable_getColumnRange) • <u>getColumnType</u> (#DataTable_getColumnType) • <u>getDistinctValues</u> (#DataTable_getDistinctValues) • <u>getFilteredRows</u> (#DataTable_getFilteredRows) • <u>getFormattedValue</u> (#DataTable_getFormattedValue) • <u>getNumberOfColumns</u> (#DataTable_getNumberOfColumns) • <u>getNumberOfRows</u> (#DataTable_getNumberOfRows) • <u>getProperty</u> (#DataTable_getProperty) • <u>getProperties</u> (#DataTable_getProperties) • <u>getRowProperty</u> (#DataTable_getRowProperty) • <u>getSortedRows</u> (#DataTable_getSortedRows) • <u>getTableProperty</u> (#DataTable_getTableProperty) • <u>getValue</u> (#DataTable_getValue) 		
getTableColumnIndex(viewColumnIndex)	Number	Returns the index in the underlying table (or view) of a given column specified by its index in this view. viewColumnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the

		<p>getNumberOfColumns() method. Returns -1 if this is a <u>generated column</u> (#DataView_setColumns).</p> <p>Example: If setColumns([3, 1, 4]) was previously called, then getTableColumnIndex(2) will return 4.</p>
getTableRowIndex(viewRowIndex)	Number	<p>Returns the index in the underlying table (or view) of a given row specified by its index in this view. viewRowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method.</p> <p>Example: If setRows([3, 1, 4]) was previously called, then getTableRowIndex(2) will return 4.</p>
getViewColumnIndex(tableColumnIndex)	Number	<p>Returns the index in this view that maps to a given column specified by its index in the underlying table (or view). If more than one such index exists, returns the first (smallest) one. If no such index exists (the specified column is not in the view), returns -1. tableColumnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method of the underlying table/view.</p> <p>Example: If setColumns([3, 1, 4]) was previously called, then getViewColumnIndex(4) will return 2.</p>
getViewColumns()	Array of numbers	<p>Returns the columns in this view, in order. That is, if you call setColumns with some array, and then call getViewColumns() you should get an identical array.</p>

getViewRowIndex(tableRowIndex)	Number	<p>Returns the index in this view that maps to a given row specified by its index in the underlying table (or view). If more than one such index exists, returns the first (smallest) one. If no such index exists (the specified row is not in the view), returns -1. tableRowIndex should be a number greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method of the underlying table/view.</p> <p>Example: If setRows([3, 1, 4]) was previously called, then getViewRowIndex(4) will return 2.</p>
getViewRows()	Array of numbers	<p>Returns the rows in this view, in order. That is, if you call setRows with some array, and then call getViewRows() you should get an identical array.</p>
hideColumns(columnIndexes)	none	<p>Hides the specified columns from the current view. columnIndexes is an array of numbers representing the indexes of the columns to hide. These indexes are the index numbers in the <i>underlying table/view</i>. The numbers in columnIndexes do not have to be in order (that is, [3,4,1] is fine). The remaining columns retain their index order when you iterate through them. Entering an index number for a column already hidden is not an error, but entering an index that does not exist in the underlying table/view will throw an error. To unhide columns, call setColumns().</p> <p>Example: If you have a table with 10 columns, and you call setColumns([2, 7, 1, 7, 9]),</p>

		and then <code>hideColumns([7,9])</code> , the columns in the view will then be [2,1].
<code>hideRows(min, max)</code>	None	Hides all rows with indexes that lie between min and max (inclusive) from the current view. This is a convenience syntax for <code>hideRows(rowIndexes)</code> above. For example, <code>hideRows(5, 10)</code> is equivalent to <code>hideRows([5, 6, 7, 8, 9, 10])</code> .
<code>hideRows(rowIndexes)</code>	None	<p>Hides the specified rows from the current view. rowIndexes is an array of numbers representing the indexes of the rows to hide. These indexes are the index numbers in the <i>underlying table/view</i>. The numbers in rowIndexes do not have to be in order (that is, [3,4,1] is fine). The remaining rows retain their index order. Entering an index number for a row already hidden is not an error, but entering an index that does not exist in the underlying table/view will throw an error. To unhide rows, call <code>setRows()</code>.</p> <p>Example: If you have a table with 10 rows, and you call <code>setRows([2,7,1,7,9])</code>, and then <code>hideRows([7,9])</code>, the rows in the view will then be [2,1].</p>
<code>setColumns(columnIndexes)</code>	None	Specifies which columns are visible in this view. Any columns not specified will be hidden. This is an array of column indexes in the underlying table/view, or calculated columns. If you don't call this method, the default is to show all columns. The array can also contain duplicates, to show the same column multiple times. Columns will be shown in the order specified.

- **columnIndexes** - An array of numbers and/or objects (can be mixed):
 - **Numbers** specify the index of the source data column to include in the view. The data is brought through unmodified. If you need to explicitly define a role or additional column properties, specify an object with a **sourceColumn** property.
 - **Objects** specify a *calculated column*. A calculated column creates a value on the fly for each row and adds it to the view. The object must have the following properties:
 - **calc** [*function*] - A function that will be called for each row in the column to calculate a value for that cell. The function signature is **func(*dataTable*, *row*)**, where **dataTable** is the source **DataTable**, and **row** is the index of the source data row. The function should return a single value of the type specified by **type**.
 - **type** [*string*] - The JavaScript type of the value that the

		<p>calc function returns.</p> <ul style="list-style-type: none">• label [<i>Optional, string</i>] - An optional label to assign to this generated column. If not specified, the view column will have no label.• id [<i>Optional, string</i>] - An optional ID to assign to this generated column.• sourceColumn - [<i>Optional, number</i>] The source column to use as a value; if specified, do not specify the calc or the type property. This is similar to passing in a number instead of an object, but enables you to specify a role and properties for the new column.• properties [<i>Optional, object</i>] - An object containing any arbitrary properties to assign to this column. If not specified, the view column will have no properties.• role [<i>Optional, string</i>] - A <u>role</u>
--	--	--

		<p>(https://developer.s.google.com/chart/interactive/docs/roles) to assign to this column. If not specified, the existing role will not be imported.</p> <p>Examples:</p> <pre>// Show some columns direc // Shows column 3 twice. view.setColumns([3, 4, 3, // Underlying table has a // The view imports this d // that converts the value view.setColumns([1, {calc:c function cmToInches(dataTa return Math.floor(dataTa }</pre>
setRows(min, max)	None	<p>Sets the rows in this view to be all indexes (in the underlying table/view) that lie between min and max (inclusive). This is a convenience syntax for setRows(rowIndexes) below. For example, setRows(5, 10) is equivalent to setRows([5, 6, 7, 8, 9, 10]).</p>
setRows(rowIndexes)	None	<p>Sets the visible rows in this view, based on index numbers from the underlying table/view. rowIndexes should be an array of index numbers specifying which rows to show in the view. The array specifies the order in which to show the rows, and rows can be duplicated. Note that <i>only</i> the rows specified in rowIndexes will be shown; this method clears all other rows from the view. The array can also contain duplicates, effectively duplicating the specified row in</p>

		<p>this view (for example, <code>setRows([3, 4, 3, 2])</code> will cause row 3 to appear twice in this view). The array thus provides a mapping of the rows from the underlying table/view to this view. You can use <code><u>getFilteredRows()</u></code> (<code>#DataTable_getFilteredRows</code>) or <code><u>getSortedRows()</u></code> (<code>#DataTable_getSortedRows</code>) to generate input for this method.</p> <p>Example: To create a view with rows three and zero of an underlying table/view: <code>view.setRows([3, 0])</code></p>
<code>toDataTable()</code>	DataTable	Returns a DataTable object populated with the visible rows and columns of the DataView .
<code>toJSON()</code>	string	<p>Returns a string representation of this DataView. This string does not contain the actual data; it only contains the DataView-specific settings such as visible rows and columns. You can store this string and pass it to the static <code><u>DataView.fromJSON()</u></code> (<code>#DataView_fromJSON</code>) <u>constructor</u> (<code>#DataView_fromJSON</code>) to recreate this view. This won't include <u>generated columns</u> (<code>#DataView_setColumns</code>).</p>

ChartWrapper Class

A **ChartWrapper** class is used to wrap your chart and handle all loading, drawing, and Datasource querying for your chart. The class exposes convenience methods for setting values on the chart and drawing it. This class simplifies reading from a data source, because you do not have to create a query callback handler. You can also use it to save a chart easily for reuse.

Another bonus of using `ChartWrapper` is that you can reduce the number of library loads by using dynamic loading. Additionally, you don't need to load the packages explicitly since `ChartWrapper` will handle looking up and loading the chart packages for you. See the examples below for details.

However, `ChartWrapper` currently only propagates a subset of events thrown by charts: `select`, `ready`, and `error`. Other events are not transmitted through the `ChartWrapper` instance; to get other events, you must call `getChart()` and subscribe to events directly on the chart handle, as shown here:

```
var wrapper;
function drawVisualization() {

    // Draw a column chart
    wrapper = new google.visualization.ChartWrapper({
        chartType: 'ColumnChart',
        dataTable: [['Germany', 'USA', 'Brazil', 'Canada', 'France', 'RU'],
                    [700, 300, 400, 500, 600, 800]],
        options: {'title': 'Countries'},
        containerId: 'visualization'
    });

    // Never called.
    google.visualization.events.addListener(wrapper, 'onmouseover', uselessHandler);

    // Must wait for the ready event in order to
    // request the chart and subscribe to 'onmouseover'.
    google.visualization.events.addListener(wrapper, 'ready', onReady);

    wrapper.draw();

    // Never called
    function uselessHandler() {
        alert("I am never called!");
    }

    function onReady() {
        google.visualization.events.addListener(wrapper.getChart(), 'onmouseover'
    )

    // Called
    function usefulHandler() {
        alert("Mouseover event!");
    }
}
```

Constructor

`ChartWrapper(opt_spec)`

opt_spec

[*Optional*] - Either a JSON object defining the chart, or a serialized string version of that object. The format of this object is shown in the [drawChart\(\) documentation](#) (`#google.visualization.drawchart`). If not specified, you must set all the appropriate properties using the `set...` methods exposed by this object.

Methods

ChartWrapper exposes the following additional methods:

Method	Return Type	Description
<code>draw(<i>opt_container_ref</i>)</code>	None	Draws the chart. You must call this method after the chart or data has been loaded. <ul style="list-style-type: none"><code>opt_container_ref</code> [<i>Optional</i>] - A reference to the page. If specified, the chart will be drawn in the element with ID specified by <code>cc</code>.
<code>toJSON()</code>	String	Returns a string version of the JSON representation of the chart.
<code>clone()</code>	ChartWrapper (<code>#chartwrapperobject</code>)	Returns a deep copy of the chart wrapper.
<code>getDataSourceUrl()</code>	String	If this chart gets its data from a data source (https://developers.google.com/chart/interact), returns the URL for this data source. Otherwise, returns null.
<code>getDataTable()</code>	google.visualization.DataTable (<code>#DataTable</code>)	If this chart gets its data from a locally-defined reference to the chart's <code>DataTable</code> . If this chart is not a data source, it will return null. Any changes that you make to the returned object will be reflected in the chart the next time you call <code>ChartWrapper.draw()</code> .
<code>getChartType()</code>	String	The class name of the wrapped chart. If this is not qualified with <code>google.visualization</code> , it will be qualified. For example, if it is a Treemap chart, it would return "Treemap" rather than "google.visualization.treemap".
<code>getChartName()</code>	String	Returns the chart name assigned by <code>setChartName()</code> .

getChart()	Chart object reference	Returns a reference to the chart created by this <u>google.visualization.BarChart</u> (https://developers.google.com/chart/interact) or a <u>google.visualization.ColumnChart</u> (https://developers.google.com/chart/interact). This will return null until after you have called <code>draw()</code> on the chart object, and it throws a ready event. Methods can be reflected on the page.
getContainerId()	String	The ID of the chart's DOM container element.
getQuery()	String	The query string for this chart, if it has one and
getRefreshInterval()	Number	Any <u>refresh interval</u> (#Query) for this chart, if it indicates no refresh.
getOption(<i>key</i>, <i>opt_default_val</i>)	Any type	Returns the specified chart option value <ul style="list-style-type: none"> <i>key</i> - The name of the option to retrieve. May be <code>'vAxis.title'</code>. <i>opt_default_value</i> [Optional] - If the specified value will be returned.
getOptions()	Object	Returns the options object for this chart.
getView()	Object OR Array	Returns the DataView initializer object, in the form of <u>toJSON()</u> (#DataView_toJSON), or an array of
setDataSourceUrl(<i>url</i>)	None	Sets the URL of a <u>data source</u> (https://developers.google.com/chart/interact) for this chart. If you also set a data table for this chart, the URL will be ignored.
setDataTable(<i>table</i>)	None	Sets the DataTable for the chart. Pass in one of the following: a DataTable object; a JSON representation of a DataTable object; or the syntax of <u>arrayToDataTable()</u> (#google.visualization.arrayToDataTable).
setChartType(<i>type</i>)	None	Sets the chart type. Pass in the class name of the Google chart, do not qualify it with google.visualization . For example, for a pie chart, pass in "PieChart".
setChartName(<i>name</i>)	None	Sets an arbitrary name for the chart. This is not used for the chart, unless a custom chart is explicitly designed.
setContainerId(<i>id</i>)	None	Sets the ID of the containing DOM element for the chart.
setQuery(<i>query_string</i>)	None	Sets a query string, if this chart queries a data source. If you also set a data source URL, the query string will be ignored.
setRefreshInterval(<i>interval</i>)	None	Sets the <u>refresh interval</u> (#Query) for this chart. You must also set a data source URL if specifying a refresh interval.

		no refresh.
<code>setOption(key, value)</code>	None	Sets a single chart option value, where <i>key</i> is the value. To unset an option, pass in null for the value. To set a qualified name, such as <code>'vAxis.title'</code> .
<code>setOptions(options_obj)</code>	None	Sets a complete options object for a chart.
<code>setView(view_spec)</code>	None	Sets a DataView initializer object, which acts as a data source. The chart wrapper must have underlying data to apply this view to. You can pass a DataView initializer object, like that returned by <code>(#DataView_toJSON)</code> . You can also pass in an array of objects, in which case the first DataView in the array is used to create a new data table, and applied to the data table resulting from applying the other DataView objects and so on.

Events

The `ChartWrapper` object throws the following events. Note that you must call `ChartWrapper.draw()` before any events will be thrown.

Name	Description	Properties
error	Fired when an error occurs when attempting to render the chart.	id, message
ready	The chart is ready for external method calls. If you want to interact with the chart, and call methods after you draw it, you should set up a listener for this event <i>before</i> you call the <code>draw</code> method, and call them only after the event was fired.	None
select	Fired when the user clicks a bar or legend. When a chart element is selected, the corresponding cell in the data table is selected; when a legend is selected, the corresponding column in the data table is selected. To learn what has been selected, call <code>ChartWrapper.getChart().getSelection()</code> (<code>#visgetselection</code>). Note that this will only be thrown when the underlying chart type throws a selection event.	None

Example

The following two snippets create an equivalent [line chart](https://developers.google.com/chart/interactive/docs/gallery/linechart) (<https://developers.google.com/chart/interactive/docs/gallery/linechart>). The first example uses JSON literal notation to define the chart; the second uses `ChartWrapper` methods to set these values.

```

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Google Visualization API Sample</title>
<!--
    One script tag loads all the required libraries! Do not specify any chart type
    autoload statement.
-->
<script type="text/javascript"
    src='https://www.gstatic.com/charts/loader.js'></script>
<script type="text/javascript">
    google.charts.load('current');
    google.charts.setOnLoadCallback(drawVisualization);

    function drawVisualization() {
        var wrap = new google.visualization.ChartWrapper({
            'chartType': 'LineChart',
            'dataSourceUrl': 'http://spreadsheets.google.com/tq?key=pCQbetd-CptGXxxl',
            'containerId': 'visualization',
            'query': 'SELECT A,D WHERE D > 100 ORDER BY D',
            'options': {'title': 'Population Density (people/km^2)', 'legend': 'none'}
        });
        wrap.draw();
    }
</script>
</head>
<body>
    <div id="visualization" style="height: 400px; width: 400px;"></div>
</body>
</html>

```

Same chart, now using setter methods:

```

<!DOCTYPE html>
<html>
<head>
<meta http-equiv='content-type' content='text/html; charset=utf-8'/>
<title>Google Visualization API Sample</title>
<!-- One script tag loads all the required libraries! Do not specify any chart type
    the autoload statement.
-->
<script type="text/javascript"
    src='https://www.gstatic.com/charts/loader.js'></script>
<script type="text/javascript">
    google.charts.load('current');
    google.charts.setOnLoadCallback(drawVisualization);

```

```
function drawVisualization() {
  // Define the chart using setters:
  var wrap = new google.visualization.ChartWrapper();
  wrap.setChartType('LineChart');
  wrap.setDataSourceUrl('http://spreadsheets.google.com/tq?key=pCQbetd-CptG');
  wrap.setContainerId('visualization');
  wrap.setQuery('SELECT A,D WHERE D > 100 ORDER BY D');
  wrap.setOptions({'title':'Population Density (people/km^2)', 'legend':'none'});
  wrap.draw();
}
</script>
</head>
<body>
  <div id='visualization' style='height: 400px; width: 400px;'></div>
</body>
</html>
```

ChartEditor Class

The `ChartEditor` class is used to open an in-page dialog box that enables a user to customize a visualization on the fly.

To use `ChartEditor`:

1. **Load the `charteditor` package.** In `google.charts.load()`, load the package 'charteditor'. You do not need to load the packages for the chart type that you render in the editor; the chart editor will load any package for you as needed.
2. **Create a `ChartWrapper` object that defines the chart for the user to customize.** This chart will be shown in the dialog, and the user uses the editor to redesign the chart, change chart types, or even change the source data.
3. **Create a new `ChartEditor` instance, and register to listen for the "ok" event.** This event is thrown when the user clicks the "OK" button on the dialog. When received, you should call `ChartEditor.getChartWrapper()` to retrieve the user-modified chart.
4. **Call `ChartEditor.openDialog()`, passing in the `ChartWrapper`.** This opens the dialog. The dialog buttons enable the user to close the editor. The `ChartEditor` instance is available as long as it is in scope; it is not automatically destroyed after the user closes the dialog.
5. **To update the chart in code, call `setChartWrapper()`.**

Methods

Method	Return Value	Description
<code>openDialog(chartWrapper, opt_options)</code>	null	<p>Opens the chart editor as an embedded dialog box on the page. The function returns immediately; it does not wait for the dialog to be closed. If you do not lose scope of the instance, you can call <code>openDialog()</code> again to reopen the dialog, although you must pass in a <code>ChartWrapper</code> object again.</p> <ul style="list-style-type: none"> • <i>chartWrapper</i> - A ChartWrapper (#chartwrapperobject) object defining the initial chart to render in the window. The chart must either have a populated DataTable, or be connected to a valid data source. This wrapper is copied internally to the chart editor, so any later changes that you make to your <code>ChartWrapper</code> handle will not be reflected in the chart editor's copy. • <i>opt_options</i> - [Optional] An object containing any options for the chart editor. See the options table below.
<code>getChartWrapper()</code>	ChartWrapper (#chartwrapperobject)	Returns a ChartWrapper representing the chart, with user modifications.
<code>setChartWrapper(null chartWrapper)</code>		<p>Use this method to update the rendered chart on the editor.</p> <p><i>chartWrapper</i> - A ChartWrapper object representing the new chart to render. The chart must either have a populated DataTable, or be connected to a valid data source.</p>
<code>closeDialog()</code>	null	Closes the chart editor dialog box.

Options

The chart editor supports the following options:

Name	Type	Default	Description
<code>dataSourceInput</code>	Element	null	Use this to enable the user to specify a data source for the chart.

handle or 'urlbox'	<p>This property can be one of two values:</p> <ul style="list-style-type: none"> • 'urlbox' - Show the chart's data source URL on the dialog in an editable textbox. The user can modify this, and the chart will be redrawn, based on the new data source. • DOM element - Enables you to provide a custom HTML element to use to select a data source. Pass in a handle to an HTML element, either one created in code or copied from the page. This element will be displayed on the dialog. Use this as a way to let the user choose the chart's data source. For example, create a listbox containing several data source URLs, or user-friendly names that the user can choose from. The element must implement a selection handler and use it to change the chart's data source: for example, either change the underlying DataTable, or modify the chart's dataSourceUrl field.
--------------------------	--

Events

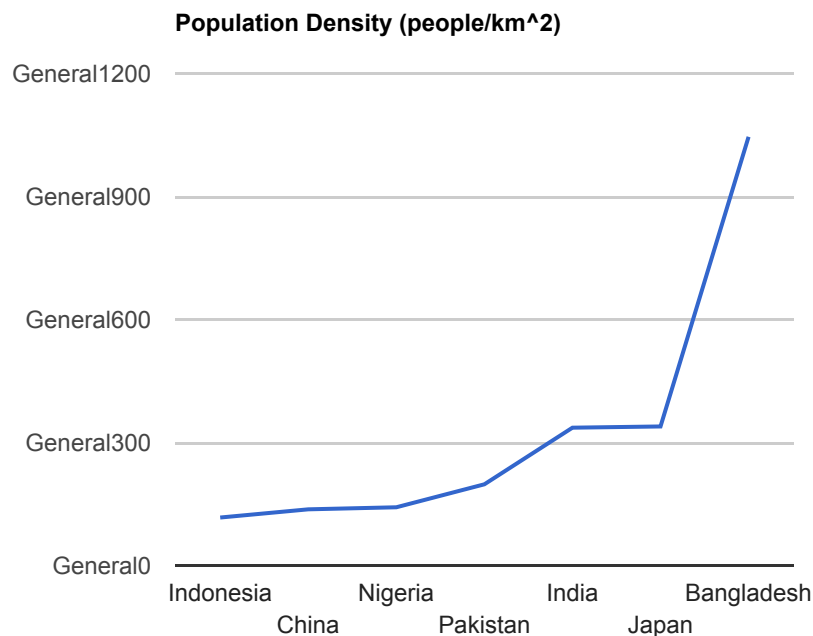
The chart editor throws the following events:

Name	Description	Properties
ok	Fired when the user clicks the "OK" button on the dialog. After receiving this method, none you should call getChartWrapper() to retrieve the user-configured chart.	
cancel	Fired when the user clicks the "Cancel" button on the dialog.	none

Example

Try it out!

[EDIT ME!](#)



The following example code opens a chart editor dialog with a populated line chart. If the user clicks "OK", the edited chart will be saved to the specified `<div>` on the page.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
  <title>
    Google Visualization API Sample
  </title>
  <script type="text/javascript"
    src="https://www.gstatic.com/charts/loader.js"></script>
  <script type="text/javascript">
    google.charts.load('current', {packages: ['charteditor']});
  </script>
  <script type="text/javascript">
    google.charts.setOnLoadCallback(loadEditor);
    var chartEditor = null;
```

```

function loadEditor() {
    // Create the chart to edit.
    var wrapper = new google.visualization.ChartWrapper({
        'chartType': 'LineChart',
        'dataSourceUrl': 'http://spreadsheets.google.com/tq?key=pCQbetd-CptGX:
        'query': 'SELECT A,D WHERE D > 100 ORDER BY D',
        'options': {'title': 'Population Density (people/km^2)', 'legend': 'no
    });

    chartEditor = new google.visualization.ChartEditor();
    google.visualization.events.addListener(chartEditor, 'ok', redrawChart)
    chartEditor.openDialog(wrapper, {});
}

// On "OK" save the chart to a <div> on the page.
function redrawChart(){
    chartEditor.getChartWrapper().draw(document.getElementById('vis_div'));
}

</script>
</head>
<body>
    <div id="vis_div" style="height: 400px; width: 600px;"></div>
</body>
</html>

```

Data Manipulation Methods

The `google.visualization.data` namespace holds static methods to perform SQL-like operations on `DataTable` objects, for example joining them or grouping by column value.

The `google.visualization.data` namespace exposes the following methods:

Method	Description
<u>google.visualization.data.group</u> (#google_visualization_data_group)	Performs a SQL GROUP BY action to return a table grouped by values in specified columns.
<u>google.visualization.data.join</u> (#google_visualization_data_join)	Joins two data tables on one or more key columns.

`group()`

Takes a populated `DataTable` object and performs a SQL-like GROUP BY operation, returning a table with rows grouped by the specified column values. Note that this does not modify the input `DataTable`.

The returned table includes one row for each combination of values in the specified key columns. Each row includes the key columns, plus one column with an aggregated column value over all rows that match the key combination (for example, a sum or count of all values in the specified column).

The `google.visualization.data` namespace includes several useful aggregation values (for example, `sum` (`#data_aggregation_functions`) and `count` (`#data_aggregation_functions`)), but you can define your own (for example, `standardDeviation` or `secondHighest`). Instructions on how to define your own aggregator are given after the method description.

Syntax

```
google.visualization.data.group(data_table, keys, columns)
```

data_table

The input `DataTable`. This will not be modified by calling `group()`.

keys

An array of numbers and/or objects specifying which columns to group by. The result table includes every column in this array, as well as every column in `columns`. If a number, this is a column index of the input `DataTable` to group by. If an object, it will include a function that can modify the specified column (for example, add 1 to the value in that column). The object must have the following properties:

- **column** - A number that is a column index from `dt` to apply the transformation to.
- **modifier** - A function that accepts one value (the cell value in that column for each row), and returns the modified value. This function is used to modify the column value to assist in the grouping: for example, by calling a `whichQuarter` function that calculates a quarter from a date column, so the API can group rows by quarter. The calculated value is displayed in the key columns in the returned table. This function can be declared inline inside this object, or it can be a function that you define elsewhere in your code (it must be within the calling scope). The API provides [one simple modifier function](#) (`#data_modifier_functions`); [here are instructions](#) (`#creating_a_modifier_function`) on how to create your own, more useful functions. You must know the data type that this function can

accept, and call it only on columns of that type. You must also know the return type of this function, and declare it in the **type** property described next.

- **type** - The type returned by the function *modifier*. This should be a JavaScript string type name, for example: 'number' or 'boolean'.
- **label** - *[Optional]* A string label to assign this column in the returned **DataTable**.
- **id** - *[Optional]* A string ID to assign this column in the returned **DataTable**.

Examples: `[0], [0,2], [0, {column:1, modifier:myPlusOneFunc, type:'number'},2]`

columns

[Optional] Lets you specify which columns, in addition to key columns, to include in the output table. Because all rows in the row group are compressed into a single output row, you must determine what value to display for that row group. For example, you could choose to show the column value from the first row in the set, or an average of all rows in that group. *columns* is an array of objects, with the following properties:

- **column** - A number specifying the index of the column to show.
- **aggregation** - A function that accepts an array of all values of this column in this row group and returns a single value to display in the result row. The return value must be of the type specified by the object's **type** property. Details on creating your own aggregation function are given below. You must know what data types this method accepts and only call it on columns of the appropriate type. The API provides several useful aggregation functions. See [Provided Aggregation Functions](#) (#data_aggregation_functions) below for a list, or [Creating an aggregation function](#) (#creating_an_aggregation_function) to learn how to write your own aggregation function.
- **type** - The return type of the aggregation function. This should be a JavaScript string type name, for example: 'number' or 'boolean'.
- **label** - *[Optional]* A string label to apply to this column in the returned table.
- **id** - *[Optional]* A string ID to apply to this column in the returned table.

Return Value

A **DataTable** with one column for each column listed in *keys* and one column for each column listed in *columns*. The table is sorted by key rows, from left to right.

Example

```
// This call will group the table by column 0 values.
// It will also show column 3, which will be a sum of
// values in that column for that row group.
var result = google.visualization.data.group(
    dt,
    [0],
    [{ 'column': 3, 'aggregation': google.visualization.data.sum, 'type': 'number' }]);

*Input table*
1  'john'  'doe'           10
1  'jane'  'doe'          100
3  'jill'  'jones'         50
3  'jack'  'jones'         75
5  'al'    'weisenheimer' 500

*Output table*
1  110
3  125
5  500
```

Provided Modifier Functions

The API provides the following modifier functions that you can pass into the *keys*. **modifier** parameter to customize grouping behavior.

Function	Input Array Type	Return Type	Description
<code>google.visualization.data.month</code>	Date	number	Given a date, it will return the zero-based month value (0, 1, 2, and so on).

Provided Aggregation Functions

The API provides the following aggregation functions that you can pass into the *columns*. **aggregation** parameter array.

Function	Input Array Type	Return Type	Description
<code>google.visualization.string</code>	number, number, string,		The average value of the array passed in.

data.avg	Date	
google.visualization.data.count	any type	number The count of rows in the group. Null and duplicate values are counted.
google.visualization.data.max	number, string, Date	number, string, Date The maximum value in the array. For strings, this is the first item in an lexicographically sorted list; for Date values, it is the latest date. Nulls are ignored.
google.visualization.data.min	number, string, Date	number, string, Date The minimum value in the array. For strings, this is the last item in an lexicographically sorted list; for Date values, it is the earliest date. Nulls are ignored.
google.visualization.data.sum	number, string, Date	number The sum of all values in the array.

Creating a modifier function

You can create a modifier function to perform a simple transformation on key values before the `group()` function groups your rows. This function takes a single cell value, performs an action on it (for example, adds 1 to the value), and returns it. The input and return types need not be the same type, but the caller must know the input and output types. Here's an example of a function that accepts a date and returns the quarter:

```
// Input type: Date
// Return type: number (1-4)
function getQuarter(someDate) {
  return Math.floor(someDate.getMonth()/3) + 1;
}
```

Creating an aggregation function

You can create an aggregation function that accepts a set of column values in a row group and returns a single number: for example, returning a count or average of values. Here is an implementation of the provided count aggregation function, which returns a count of how many rows are in the row group:

```
// Input type: Array of any type
// Return type: number
function count(values) {
  return values.length;
}
```


join()

This method joins two data tables (**DataTable** or **DataRowView** objects) into a single results table, similar to a SQL JOIN statement. You specify one or more column pairs (*key* columns) between the two tables, and the output table includes the rows according to a join method that you specify: only rows where both keys match; all rows from one table; or all rows from both tables, whether or not the keys match. The results table includes only the key columns, plus any additional columns that you specify. Note that *dt2* cannot have duplicate keys, but *dt1* can. The term "key" means the combination of all key column values, not a specific key column value; so if a row has cell values A | B | C and columns 0 and 1 are key columns, then the key for that row is AB.

Syntax

```
google.visualization.data.join(dt1, dt2, joinMethod,  
                              keys, dt1Columns, dt2Columns);
```

dt1

A populated **DataTable** to join with *dt2*.

dt2

A populated **DataTable** to join with *dt1*. This table cannot have multiple identical keys (where a key is a combination of key column values).

joinMethod

A string specifying the join type. If *dt1* has multiple rows that match a *dt2* row, the output table will include all matching *dt1* rows. Choose from the following values:

- 'full' - The output table includes all rows from both tables, regardless of whether keys match. Unmatched rows will have null cell entries; matched rows are joined.
- 'inner' - The full join filtered to include only rows where the keys match.
- 'left' - The output table includes all rows from *dt1*, whether or not there are any matching rows from *dt2*.
- 'right' - The output table includes all rows from *dt2*, whether or not there are any matching rows from *dt1*.

keys

An array of key columns to compare from both tables. Each pair is a two element array, the first is a key in *dt1*, the second is a key in *dt2*. Columns must be the same type in both tables. All specified keys must match according to the rule given by *joinMethod* in order to include a row from the table. Key columns are always included in the output table. Only *dt1*, the left-hand table, can include duplicate keys; keys in *dt2* must be unique. The term "key" here means a unique set of key columns, not individual column values. For example, if your key columns were A and B, the following table would have only unique key values (and could thus be used as *dt2*):

A	B
Jen	Red
Jen	Blue
Fred	Red

Example: `[[0, 0], [2, 1]]` compares values from the first column in both tables as well as the third column from *dt1* with the second column from *dt2*.

dt1Columns

An array of columns from *dt1* to include in the output table, in addition to *dt1*'s key columns. This is an array of column indexes.

dt2Columns

An array of columns from *dt2* to include in the output table, in addition to *dt2*'s key columns. This is an array of column indexes.

Return Value

A `DataTable` with the key columns, *dt1Columns*, and *dt2Columns*. This table is sorted by the key columns, from left to right. When *joinMethod* is 'inner', all key cells should be populated. For other join methods, if no matching key is found, the table will have a null for any unmatched key cells.

Examples

Tables

<u>dt1</u>	<u>dt2</u>
bob 111 red	bob 111 point
bob 111 green	ellyn 222 square
bob 333 orange	jane 555 circle
fred 555 blue	jane 777 triangle

```
jane | 777 | yellow          fred | 666 | dodecahedron
* Note that right table has duplicate Jane entries, but the key we will use is
* columns 0 and 1. The left table has duplicate key values, but that is
* allowed.
```

```
*Inner join* google.visualization.data.join(dt1, dt2, 'inner', [[0,0],[1,1]],
bob | 111 | red | point
bob | 111 | green | point
jane | 777 | yellow | triangle
* Note that both rows from dt1 are included and matched to
* the equivalent dt2 row.
```

```
*Full join* google.visualization.data.join(dt1, dt2, 'full', [[0,0],[1,1]], [
bob | 111 | red | point
bob | 111 | green | point
bob | 333 | orange | null
ellyn | 222 | null | square
fred | 555 | blue | null
fred | 666 | null | dodecahedron
jane | 555 | null | circle
jane | 777 | yellow | triangle
```

```
*Left join* google.visualization.data.join(dt1, dt2, 'left', [[0,0],[1,1]],
bob | 111 | red | point
bob | 111 | green | point
bob | 333 | orange | null
fred | 555 | blue | null
jane | 777 | yellow | triangle
```

```
*Right join* google.visualization.data.join(dt1, dt2, 'right', [[0,0],[1,1]]
bob | 111 | red | point
bob | 111 | green | point
ellyn | 222 | null | square
fred | 666 | null | dodecahedron
jane | 555 | null | circle
jane | 777 | yellow | triangle
```

Formatters

The Google Visualization API provides formatters that can be used to reformat data in a visualization. These formatters change the formatted value of the specified column in all rows. Note that:

- Formatters modify only the formatted values, not the underlying values. For example, the displayed value would be "\$1,000.00" but the underlying value would still be "1000".
- Formatters only affect one column at a time; to reformat multiple columns, apply a formatter to each column that you want to change.
- If you also use user-defined formatters, certain Google Visualization formatters will override all user-defined formatters.

The actual formatting applied to the data is derived from the locale the API has been loaded with. For more details, see [loading charts with a specific locale](https://developers.google.com/chart/interactive/docs/library_loading_enhancements#loadwithlocale) (https://developers.google.com/chart/interactive/docs/library_loading_enhancements#loadwithlocale)



Important: Formatters can only be used with a **DataTable**; they cannot be used with a **DataView** (**DataView** objects are read-only).

Here are the general steps for using a formatter:

1. Get your populated **DataTable** object.
 2. For each column that you want to reformat:
 - a. Create an object that specifies all the options for your formatter. This is a basic JavaScript object with a set of properties and values. Look at your formatter's documentation to see what properties are supported. (Optionally, you can pass in an object literal notation object specifying your options.)
 - b. Create your formatter, passing in your options object.
 - c. Call `formatter.Format(table, colIndex)`, passing in the **DataTable** and the (zero-based) column number of the data to reformat. Note that you can only apply a single formatter to each column; applying a second formatter will simply overwrite the effects of the first.
- Important:** Many formatters require HTML tags to display special formatting; if your visualization supports an `allowHtml` option, you should set it to true.

Here is an example of changing the formatted date values of a date column to use a long date format ("January 1, 2009"):

```

var data = new google.visualization.DataTable();
data.addColumn('string', 'Employee Name');
data.addColumn('date', 'Start Date');
data.addRows(3);
data.setCell(0, 0, 'Mike');
data.setCell(0, 1, new Date(2008, 1, 28));
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));

// Create a formatter.
// This example uses object literal notation to define the options.
var formatter = new google.visualization.DateFormat({formatType: 'long'});

// Reformat our data.
formatter.format(data, 1);

// Draw our data
var table = new google.visualization.Table(document.getElementById('dataTable'));
table.draw(data, {showRowNumber: true});

```

Most formatters expose the following two methods:

Method	Description
google.visualization. formatter_name(options)	<p>Constructor, where <i>formatter_name</i> is a specific formatter class name.</p> <ul style="list-style-type: none"> <i>options</i> - A generic JavaScript object that specifies the options for that formatter. This object is a generic object with property/value pairs with properties specific to that formatter. See the documentation for your specific formatter to learn what options are supported. Here are two example ways to call the constructor for the DateFormat object, passing in two properties: <pre> // Object literal technique var formatter = new google.visualization.DateFormat({formatType: 'long', timeZone: -5}); // Equivalent property setting technique var options = new Object(); options['formatType'] = 'long'; options['timeZone'] = -5; var formatter = new google.visualization.DateFormat(options); </pre>

<code>format(<i>data</i>, <i>colIndex</i>)</code>	Reformats the data in the specified column. <ul style="list-style-type: none"> <i>data</i> - A <u>DataTable</u> (#DataTable) containing the data to reformat. You cannot use a <u>DataRowView</u> here. <i>colIndex</i> - The zero-based index of the column to format. To format multiple columns, you must call this method multiple times, with different <i>colIndex</i> values.
---	--

The Google Visualization API provides the following formatters:

Formatter Name	Description
<u>ArrowFormat</u> (#arrowformatter)	Adds an up or down arrow, indicating whether the cell value is above or below a specified value.
<u>BarFormat</u> (#barformatter)	Adds a colored bar, the direction and color of which indicates whether the cell value is above or below a specified value.
<u>ColorFormat</u> (#colorformatter)	Colors a cell according to whether the values fall within a specified range.
<u>DateFormat</u> (#dateformatter)	Formats a Date or DateTime value in several different ways, including "January 1, 2009," "1/1/09" and "Jan 1, 2009."
<u>NumberFormat</u> (#numberformatter)	Formats various aspects of numeric values.
<u>PatternFormat</u> (#patternformatter)	Concatenates cell values on the same row into a specified cell, along with arbitrary text.

ArrowFormat

Adds an up or down arrow to a numeric cell, depending on whether the value is above or below a specified base value. If equal to the base value, no arrow is shown.

	Department	Revenues Change
1	Shoes	▲ 12.0%
2	Sports	▼ -7.3%
3	Toys	0%
4	Electronics	▼ -2.1%
5	Food	▲ 22.0%

Options

ArrowFormat supports the following options, passed in to the constructor:

OptionDescription

base A number indicating the base value, used to compare against the cell value. If the cell value is higher, the cell will include a green up arrow; if the cell value is lower, it will include a red down arrow; if the same, no arrow.

Sample code

```
var data = new google.visualization.DataTable();
data.addColumn('string', 'Department');
data.addColumn('number', 'Revenues Change');
data.addRows([
  ['Shoes', {v:12, f:'12.0%'}],
  ['Sports', {v:-7.3, f:'-7.3%'}],
  ['Toys', {v:0, f:'0%'}],
  ['Electronics', {v:-2.1, f:'-2.1%'}],
  ['Food', {v:22, f:'22.0%'}]
]);

var table = new google.visualization.Table(document.getElementById('arrow'));


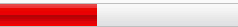

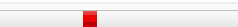
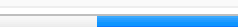
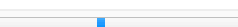
var formatter = new google.visualization.ArrowFormat();
formatter.format(data, 1); // Apply formatter to second column

table.draw(data, {allowHtml: true, showRowNumber: true});
```

BarFormat

Adds a colored bar to a numeric cell indicating whether the cell value is above or below a specified base value

	Department	Revenues
--	------------	----------

	Department		Revenue
1	Shoes		10,700
2	Sports		-15,400
3	Toys		12,500
4	Electronics		-2,100
5	Food		22,600
6	Art		1,100

Options

BarFormat supports the following options, passed in to the constructor:

Option	Example	Description
base		A number that is the base value to compare the cell value against. If the cell value is higher, it will be drawn to the right of the base; if lower, it will be drawn to the left. Default value is 0.
colorNegative		A string indicating the negative value section of bars. Possible values are 'red', 'green' and 'blue'; default value is 'red'.
colorPositive		A string indicating the color of the positive value section of bars. Possible values are 'red', 'green' and 'blue'. Default is 'blue'.
drawZeroLine		A boolean indicating if to draw a 1 pixel dark base line when negative values are present. The dark line is there to enhance visual scanning of the bars. Default value is 'false'.
max		The maximum number value for the bar range. Default value is the highest value in the table.
min		The minimum number value for the bar range. Default value is the lowest value in the table.
showValue		If true, shows values and bars; if false, shows only bars. Default value is true.
width		Thickness of each bar, in pixels. Default value is 100.

Sample code


```

var data = new google.visualization.DataTable();
data.addColumn('string', 'Department');
data.addColumn('number', 'Revenues');
data.addRows([
    ['Shoes', 10700],
    ['Sports', -15400],
    ['Toys', 12500],
    ['Electronics', -2100],
    ['Food', 22600],
    ['Art', 1100]
]);

var table = new google.visualization.Table(document.getElementById('barfo

var formatter = new google.visualization.BarFormat({width: 120});
formatter.format(data, 1); // Apply formatter to second column

table.draw(data, {allowHtml: true, showRowNumber: true, width: '100%', he

```

ColorFormat

Assigns colors to the foreground or background of a numeric cell, depending on the cell value. This formatter is unusual, in that it doesn't take its options in the constructor. Instead, you should call `addRange()` or `addGradientRange()` as many times as you want, to add color ranges, before calling `format()`. Colors can be specified in any acceptable HTML format, for example "black", "#000000", or "#000".

	Department	Revenues
1	Shoes	10,70
2	Sports	-15,40
3	Toys	12,50
4	Electronics	-2,10
5	Food	22,60
6	Art	1,10

Methods

ColorFormat supports the following methods:

Method	Description
google.	Constructor. Takes no arguments.

visualization. ColorFormat()	
addRange(<i>from</i>, <i>to</i>, <i>color</i>, <i>bgcolor</i>)	<p>Specifies a foreground color and/or background color to a cell, depending on the cell value. Any cell with a value in the specified <i>from</i>–<i>to</i> range will be assigned <i>color</i> and <i>bgcolor</i>. It is important to realize that the range is non-inclusive, because creating a range from 1–1,000 and a second from 1,000– 2,000 will not cover the value 1,000!</p> <ul style="list-style-type: none"> • <i>from</i> - [<i>String</i>, <i>Number</i>, <i>Date</i>, <i>DateTime</i>, or <i>TimeOfDay</i>] The lower boundary (inclusive) of the range, or null. If null, it will match $-\infty$. String boundaries are compared alphabetically against string values. • <i>to</i> - [<i>String</i>, <i>Number</i>, <i>Date</i>, <i>DateTime</i>, or <i>TimeOfDay</i>] The high boundary (non-inclusive) of the range, or null. If null, it will match $+\infty$. String boundaries are compared alphabetically against string values. • <i>color</i> - The color to apply to text in matching cells. Values can be either '#RRGGBB' values or defined color constants, (example: '#FF0000' or 'red'). • <i>bgcolor</i> - The color to apply to the background of matching cells. Values can be either '#RRGGBB' values or defined color constants, (example: '#FF0000' or 'red').
addGradientRange(<i>from</i>, <i>to</i>, <i>color</i>, <i>fromBgColor</i>, <i>toBgColor</i>)	<p>Assigns a background color from a range, according to the cell value. The <i>color</i> is scaled to match the cell's value within a range from a lower boundary color to an upper boundary color. Note that this method cannot compare string values, as addRange() can. <i>Tip</i>: Color ranges are often hard for viewers to gauge accurately; the simplest and easiest to read range is from a fully saturated color to white (e.g., #FF0000–FFFFFF).</p> <ul style="list-style-type: none"> • <i>from</i> - [<i>Number</i>, <i>Date</i>, <i>DateTime</i>, or <i>TimeOfDay</i>] The lower boundary (inclusive) of the range, or null. If null, it will match $-\infty$. • <i>to</i> - [<i>Number</i>, <i>Date</i>, <i>DateTime</i>, or <i>TimeOfDay</i>] The higher boundary (non-inclusive) of the range, or null. If null, it will match $+\infty$. • <i>color</i> - The color to apply to text in matching cells. This color is the same for all cells, no matter what their value. • <i>fromBgColor</i> - The background color for cells holding values at the low end of the gradient. Values can be either '#RRGGBB' values or defined color constants, (example: '#FF0000' or 'red'). • <i>toBgColor</i> - The background color for cells holding values at the high end of the gradient. Values can be either '#RRGGBB' values or defined color constants, (example: '#FF0000' or 'red').
format(<i>dataTable</i>, <i>columnIndex</i>)	<p>The standard format() method to apply formatting to the specified column.</p>

Sample code

```
var data = new google.visualization.DataTable();
data.addColumn('string', 'Department');
data.addColumn('number', 'Revenues');
data.addRows([
  ['Shoes', 10700],
  ['Sports', -15400],
  ['Toys', 12500],
  ['Electronics', -2100],
  ['Food', 22600],
  ['Art', 1100]
]);

var table = new google.visualization.Table(document.getElementById('color

var formatter = new google.visualization.ColorFormat();
formatter.addRange(-20000, 0, 'white', 'orange');
formatter.addRange(20000, null, 'red', '#33ff33');
formatter.format(data, 1); // Apply formatter to second column

table.draw(data, {allowHtml: true, showRowNumber: true, width: '100%', he
```

DateFormat

Formats a JavaScript `Date` value in a variety of ways, including "January 1, 2016," "1/1/16" and "Jan 1, 2016".

	Employee Name	Start Date (Long)	Start Date (Medium)	Start Date (Short)
1	Mike	February 28, 2008	Feb 28, 2008	2/28/08
2	Bob	June 1, 2007	Jun 1, 2007	6/1/07
3	Alice	August 16, 2006	Aug 16, 2006	8/16/06

Options

`DateFormat` supports the following options, passed in to the constructor:

Option	Description
<code>formatType</code>	A quick formatting option for the date. The following string values are supported, reformatting the date February 28, 2016 as shown:

- 'short' - Short format: e.g., "2/28/16"
- 'medium' - Medium format: e.g., "Feb 28, 2016"
- 'long' - Long format: e.g., "February 28, 2016"

You cannot specify both **formatType** and **pattern**.

pattern A custom format pattern to apply to the value, similar to the [ICU date and time format](http://icu-project.org/userguide/formatDateTime.html) (http://icu-project.org/userguide/formatDateTime.html). For example: **var formatter3 = new google.visualization.DateFormat({pattern: "EEE, MMM d, 'yy"});**

You cannot specify both **formatType** and **pattern**. You can read more details about patterns in the next section.

timeZone The time zone in which to display the date value. This is a numeric value, indicating GMT + this number of time zones (can be negative). Date object are created by default with the assumed time zone of the computer on which they are created; this option is used to display that value in a different time zone. For example, if you created a Date object of 5pm noon on a computer located in Greenwich, England, and specified **timeZone** to be -5 (**options['timeZone'] = -5**, or Eastern Pacific Time in the US), the value displayed would be 12 noon.

Methods

DateFormat supports the following methods:

Method	Description
google.visualization.DateFormat(<i>options</i>)	Constructor. See the options section above for more info.
format(<i>dataTable</i>, <i>columnIndex</i>)	The standard format() method to apply formatting to the specified column.
formatValue(<i>value</i>)	Returns the formatted value of a given value. This method does not require a DataTable .

Sample code

```
function drawDateFormatTable() {
  var data = new google.visualization.DataTable();
  data.addColumn('string', 'Employee Name');
  data.addColumn('date', 'Start Date (Long)');
  data.addColumn('date', 'Start Date (Medium)');
  data.addColumn('date', 'Start Date (Short)');
```

```

data.addRow([
  ['Mike', new Date(2008, 1, 28, 0, 31, 26),
    new Date(2008, 1, 28, 0, 31, 26),
    new Date(2008, 1, 28, 0, 31, 26)],
  ['Bob', new Date(2007, 5, 1, 0),
    new Date(2007, 5, 1, 0),
    new Date(2007, 5, 1, 0)],
  ['Alice', new Date(2006, 7, 16),
    new Date(2006, 7, 16),
    new Date(2006, 7, 16)]
]);

// Create three formatters in three styles.
var formatter_long = new google.visualization.DateFormat({formatType: 'long'});
var formatter_medium = new google.visualization.DateFormat({formatType: 'medium'});
var formatter_short = new google.visualization.DateFormat({formatType: 'short'});

// Reformat our data.
formatter_long.format(data, 1);
formatter_medium.format(data, 2);
formatter_short.format(data, 3);

// Draw our data
var table = new google.visualization.Table(document.getElementById('dataTable'));
table.draw(data, {showRowNumber: true, width: '100%', height: '100%'});
}

```

More About Date Patterns

Here are some more details on what patterns are supported:

The patterns are similar to the [ICU date and time format](http://icu-project.org/userguide/formatDateTime.html)

(<http://icu-project.org/userguide/formatDateTime.html>), but the following patterns are not yet supported: A e D F g Y u w W. To avoid collision with patterns, any literal text you want to appear in the output should be surrounded by single quotes, except for the single quote, which should be doubled: e.g., "K 'o' 'clock.'".

Pattern Description		Example Output
GG	Era designator.	"AD"
yy or yyyy	year.	1996
M	Month in year. For January:	"July"

	<ul style="list-style-type: none"> • M produces 1 • MM produces 01 • MMM produces Jan • MMMM produces January 	"07"
d	Day in month. Extra 'd' values will add leading zeros.	10
h	Hour in 12 hour scale. Extra 'h' values will add leading zeros.	12
H	Hour in 24 hour scale. Extra 'H' values will add leading zeros.	0
m	Minute in hour. Extra 'm' values will add leading zeros.	30
s	Second in minute. Extra 's' values will add leading zeros.	55
S	Fractional second. Extra 'S' values will be padded on the right with zeros.	978
E	Day of week. Following outputs for "Tuesday": <ul style="list-style-type: none"> • E produces T • EE or EEE Produce Tu or Tues • EEEE Produces Tuesday 	"Tues" "Tuesday"
aa	AM/PM	"PM"
k	Hour in day (1~24). Extra 'k' values will add leading zeros.	24
K	Hour in AM/PM (0~11). Extra 'k' values will add leading zeros.	0
z	Time zone. For time zone 5, produces "UTC+5"	"UTC+5"
Z	Time zone in RFC 822 format. For time zone -5: Z, ZZ, ZZZ produce -0500 ZZZZ and more produce "GMT -05:00"	"-0800" "GMT -05:00"
v	Time zone (generic).	"Etc/GMT-5"
'	escape for text	'Date='
"	single quote	"yy

NumberFormat

Describes how numeric columns should be formatted. Formatting options include specifying a prefix symbol (such as a dollar sign) or the punctuation to use as a

thousands marker.

	Department	Revenues
1	Shoes	\$10,700.0
2	Sports	(\$15,400.00
3	Toys	\$12,500.0
4	Electronics	(\$2,100.00
5	Food	\$22,600.0
6	Art	\$1,100.0

Options

`NumberFormat` supports the following options, passed in to the constructor:

Option	Description
decimalSymbol	A character to use as the decimal marker. The default is a dot (.).
fractionDigits	A number specifying how many digits to display after the decimal. The default is 2. If you specify more digits than the number contains, it will display zeros for the smaller values. Truncated values will be rounded (5 rounded up).
groupingSymbol	A character to be used to group digits to the left of the decimal into sets of three. Default is a comma (,).
negativeColor	The text color for negative values. No default value. Values can be any acceptable HTML color value, such as "red" or "#FF0000".
negativeParens	A boolean, where true indicates that negative values should be surrounded by parentheses. Default is true.
pattern	<p>A format string. When provided, all other options are ignored, except negativeColor.</p> <p>The format string is a subset of the ICU pattern set (http://icu-project.org/apiref/icu4c/classDecimalFormat.html#_details). For instance, {pattern: '#,###%' } will result in output values "1,000%", "750%", and "50%" for values 10, 7.5, and 0.5.</p>
prefix	A string prefix for the value, for example "\$".
suffix	A string suffix for the value, for example "%".

Methods

NumberFormat supports the following methods:

Method	Description
<code>google.visualization.NumberFormat(<i>options</i>)</code>	Constructor. See the options section above for more info.
<code>format(<i>dataTable</i>, <i>columnIndex</i>)</code>	The standard <code>format()</code> method to apply formatting to the specified column.
<code>formatValue(<i>value</i>)</code>	Returns the formatted value of a given value. This method does not require a <code>DataTable</code> .

Sample code

```
var data = new google.visualization.DataTable();
data.addColumn('string', 'Department');
data.addColumn('number', 'Revenues');
data.addRows([
  ['Shoes', 10700],
  ['Sports', -15400],
  ['Toys', 12500],
  ['Electronics', -2100],
  ['Food', 22600],
  ['Art', 1100]
]);

var table = new google.visualization.Table(document.getElementById('numberTable'));

var formatter = new google.visualization.NumberFormat(
  {prefix: '$', negativeColor: 'red', negativeParens: true});
formatter.format(data, 1); // Apply formatter to second column

table.draw(data, {allowHtml: true, showRowNumber: true, width: '100%', height: 400});
```

PatternFormat

Enables you to merge the values of designated columns into a single column, along with arbitrary text. So, for example, if you had a column for first name and a column for last name, you could populate a third column with {last name}, {first name}. This formatter does not follow the conventions for the constructor and the `format()` method. See the Methods section below for instructions.

	name
1	John Lennon
2	Paul McCartney
3	George Harrison
4	Ringo Starr

Methods

`PatternFormat` supports the following methods:

Method	Description
<code>google.visualization.PatternFormat(<i>pattern</i>)</code>	<p>Constructor. Does not take an options object. Instead, it takes a string <i>pattern</i> parameter. This is a string that describes which column values to put into the destination column, along with any arbitrary text. Embed placeholders in your string to indicate a value from another column to embed. The placeholders are <code>{#}</code>, where <code>#</code> is a the index of a source column to use. The index is an index in the <i>srcColumnIndices</i> array from the <code>format()</code> method below. To include a literal <code>{</code> or <code>}</code> character, escape it like this: <code>\{</code> or <code>\}</code>. To include a literal <code>\</code> mark, escape it as <code>\\</code>.</p> <p>Sample code</p> <p>The following example demonstrates a constructor for a pattern that creates an anchor element, with the first and second elements taken from the <code>format()</code> method:</p> <pre>var formatter = new google.visualization.FormatTableFormatter({ pattern: '{0}' });</pre>
<code>format(<i>dataTable</i>, <i>srcColumnIndices</i>, <i>opt_dstColumnIndex</i>)</code>	<p>The standard formatting call, with a few additional parameters:</p> <ul style="list-style-type: none"> <i>dataTable</i> - The DataTable on which to operate. <i>srcColumnIndices</i> - An array of one or more (zero-based) column indices to pull as the

sources from the underlying DataTable. This will be used as a data source for the *pattern* parameter in the constructor. The column numbers do *not* have to be in sorted order.

- *opt_dstColumnIndex* - [optional] The destination column to place the output of the *pattern* manipulation. If not specified, the first element in *srcColumnIndices* will be used as the destination.

See the formatting examples after the table.

Here are a few example inputs and outputs for a four-column table.

Row before formatting (4 columns, last is blank):

John | Paul | Jones | *[empty]*

```
var formatter = new google.visualization.PatternFormat("{0} {1} {2}");  
formatter.format(data, [0,1,2], 3);
```

Output:

John | Paul | Jones | John Paul Jones

```
var formatter = new google.visualization.PatternFormat("{1}, {0}");  
formatter.format(data, [0,2], 3);
```

Output:

John | Paul | Jones | Jones, John

Sample code

The following example demonstrates how to combine data from two columns to create an email address. It uses a [DataView](#) (#DataView) object to hide the original source columns:

```
var data = new google.visualization.DataTable();  
data.addColumn('string', 'Name');  
data.addColumn('string', 'Email');  
data.addRows([  
    ['John Lennon', 'john@beatles.co.uk'],  
    ['Paul McCartney', 'paul@beatles.co.uk'],  
    ['George Harrison', 'george@beatles.co.uk'],  
    ['Ringo Starr', 'ringo@beatles.co.uk']  
]);  
  
var table = new google.visualization.Table(document.getElementById('patte
```

```

var formatter = new google.visualization.PatternFormat(
    '<a href="mailto:{1}">{0}</a>');
// Apply formatter and set the formatted value of the first column.
formatter.format(data, [0, 1]);

var view = new google.visualization.DataView(data);
view.setColumns([0]); // Create a view with the first column only.

table.draw(view, {allowHtml: true, showRowNumber: true, width: '100%', he

```

GadgetHelper

A helper class to simplify writing Gadgets (<https://developers.google.com/gadgets/>) that use the Google Visualization API.

Constructor

```
google.visualization.GadgetHelper()
```

Methods

Method	Return Value	Description
createQueryFromPrefs (<u>google.visualization.QueryStatic</u> . Create a new instance of google.v prefs)	(#Query)	its properties according to values from the parameter prefs is <u>_IG_Prefs</u> (https://developers.google.com/gadgets/d 1. Preference _table_query_url is used 2. Preference _table_query_refresh_ refresh interval (in seconds).
validateResponse (response)	Boolean	Static. Parameter response is of type <u>goog</u> (#QueryResponse). Returns true if the res false if the query execution failed and the an error occurred, this method displays an e

Query Classes

The following objects are available to send queries for data to an external data source, such as Google Spreadsheets.

- Query (#Query) - Wraps the outgoing data request.
- QueryResponse (#QueryResponse) - Handles the response from the data source.

Query

Represents a query that is sent to a data source.

Constructor

```
google.visualization.Query(dataSourceUrl, opt_options)
```

Parameters

dataSourceUrl

[*Required, String*] URL to send the query to. See the [Charts and Spreadsheets documentation](https://developers.google.com/chart/interactive/docs/spreadsheets) (<https://developers.google.com/chart/interactive/docs/spreadsheets>) for Google Spreadsheets.

opt_options

[*Optional, Object*] A map of options for the request. **Note:** If you are accessing a [restricted data source](https://developers.google.com/chart/interactive/docs/dev/implementing_data_source#security_considerations) (https://developers.google.com/chart/interactive/docs/dev/implementing_data_source#security_considerations)

, you should not use this parameter. Here are the supported properties:

- **sendMethod** - [*Optional, String*] Specifies the method to use to send the query. Choose one of the following string values:
 - **'xhr'** - Send the query using [XmlHttpRequest](http://en.wikipedia.org/wiki/XMLHttpRequest) (<http://en.wikipedia.org/wiki/XMLHttpRequest>).
 - **'scriptInjection'** - Send the query using script injection.
 - **'makeRequest'** - [*Available only for gadgets, which are deprecated*] Send the query using the Gadget API [makeRequest\(\)](https://developers.google.com/gadgets/docs/reference/#gadgets.io.makeRequest) (<https://developers.google.com/gadgets/docs/reference/#gadgets.io.makeRequest>) method. If specified, you should also specify **makeRequestParams**.

- **'auto'** - Use the method specified by the `tqrt` URL parameter from the data source URL. `tqrt` can have the following values: 'xhr', 'scriptInjection', or 'makeRequest'. If `tqrt` is missing or has an invalid value, the default is 'xhr' for same-domain requests and 'scriptInjection' for cross-domain requests.
- **makeRequestParams** - *[Object]* A map of parameters for a `makeRequest()` query. Used and required only if **sendMethod** is 'makeRequest'.

Methods

Method	Return Value	Description
<code>abort()</code>	None	Stops the automated query sending that was started with <code>setRefreshInterval()</code> .
<code>setRefreshInterval(seconds)</code>	None	<p>Sets the query to automatically call the <code>send</code> method every specified duration (number of seconds), starting from the first explicit call to <code>send</code>. <code>seconds</code> is a number greater than or equal to zero.</p> <p>If you use this method, you should call it before calling the <code>send</code> method.</p> <p>Cancel this method either by calling it again with zero (the default) or by calling <code>abort()</code>.</p>
<code>setTimeout(seconds)</code>	None	<p>Sets the number of seconds to wait for the data source to respond before raising a timeout error. <code>seconds</code> is a number greater than zero. The default timeout is 30 seconds. This method, if used, should be called before calling the <code>send</code> method.</p>
<code>setQuery(string)</code>	None	<p>Sets the query string. The value of the <code>string</code> parameter should be a valid query.</p> <p>This method, if used, should be called before calling the <code>send</code> method.</p> <p>Learn more about the Query language (https://developers.google.com/chart/interactive/docs/querylanguage)</p>
<code>send(callback)</code>	None	<p>Sends the query to the data source. <code>callback</code> should be a function that will be called when the data source responds. The callback function will receive a single parameter of type <code>google.visualization.QueryResponse</code> (<code>#QueryResponse</code>).</p>

QueryResponse

Represents a response of a query execution as received from the data source. An instance of this class is passed as an argument to the callback function that was set when [Query.send](#) (#Query_send) was called.

Methods

Method	Return Value	Description
<code>getDataTable()</code>	DataTable (#DataTable)	Returns the data table as returned by the data source. Returns null if the query execution failed and no data was returned.
<code>getDetailedMessage()</code>	String	Returns a detailed error message for queries that failed. If the query execution was successful, this method returns an empty string. The message returned is a message that is intended for developers, and may contain technical information, for example 'Column {salary} does not exist'.
<code>getMessage()</code>	String	Returns a short error message for queries that failed. If the query execution was successful, this method returns an empty string. The message returned is a short message that is intended for end users, for example 'Invalid Query' or 'Access Denied'.
<code>getReasons()</code>	Array of strings	Returns an array of zero or more entries. Each entry is a short string with an error or warning code that was raised while executing the query. Possible codes: <ul style="list-style-type: none">• access_denied The user does not have permissions to access the data source.• invalid_query The specified query has a syntax error.• data_truncated One or more data rows that match the query selection were not returned due to output size limits. (warning).• timeout The query did not respond within the expected time.
<code>hasWarning()</code>	Boolean	Returns true if the query execution has any warning messages.
<code>isError()</code>	Boolean	Returns true if the query execution failed, and the response does not contain any data table. Returns

<false> if the query execution was successful and the response contains a data table.

Error Display

The API provides several functions to help you display custom error messages to your users. To use these functions, provide a container element on the page (typically a `<div>`), into which the API will draw a formatted error message. This container can be either the visualization container element, or a container just for errors. If you specify the visualization container element, the error message will be displayed above the visualization. Then call the appropriate function below to show, or remove, the error message.

All functions are static functions in the namespace `google.visualization.errors`.

Many visualizations can throw an error event; see error event below to learn more about that.

You can see an example custom error in the [Query Wrapper Example](https://developers.google.com/chart/interactive/docs/examples#querywrapper) (<https://developers.google.com/chart/interactive/docs/examples#querywrapper>).

Function	Return Value	Description
<code>addError(<i>container</i>, <i>message</i>, <i>opt_detailedMessage</i>, <i>opt_options</i>)</code>	String ID value that identifies the error object created. This is a unique value on the page, and can be used to remove the error or find its containing element.	<p>Adds an error display block to the specified page element, with specified text and formatting.</p> <ul style="list-style-type: none">• <i>container</i> - The DOM element into which to insert the error message. If the container cannot be found, the function will throw a JavaScript error.• <i>message</i> - A string message to display.• <i>opt_detailedMessage</i> - An optional detailed message string. By default, this is mouseover text, but that can be changed in the <i>opt_options.showInToolTip</i> property described below.• <i>opt_options</i> - An optional object with properties that set various display options for the message. The following options are supported:

		<ul style="list-style-type: none"> • showInTooltip - A boolean value where true shows the detailed message only as tooltip text, and false shows the detailed message in the container body after the short message. Default value is true. • type - A string describing the error type, which determines which css styles should be applied to this message. The supported values are 'error' and 'warning'. Default value is 'error'. • style - A style string for the error message. This style will override any styles applied to the warning type (<code>opt_options.type</code>). Example: <code>'background-color: #33ff99; padding: 2px;'</code> Default value is an empty string. • removable - A boolean value, where true means that the message can be closed by a mouse click from the user. Default value is false.
addErrorFromQueryResponse (<i>String ID value container, response</i>)	<p>that identifies the error object created, or null if the response didn't indicate an error. This is a unique value on the page, and can be used to remove the style of the display appropriate to the error or find its containing element.</p>	<p>Pass a query response and error message container to this method: if the query response indicates a query error, displays an error message in the specified page element. If the query response is null, the method will throw a JavaScript error. Pass your <u>QueryResponse</u> (#QueryResponse) received in your query handler to this message to display an error. It will also set the style of the display appropriate to the type (error or warning, similar to <code>addError(<i>opt_options.type</i>)</code>)</p> <ul style="list-style-type: none"> • <i>container</i> - The DOM element into which to insert the error message. If the container cannot be found, the function will throw a JavaScript error. • <i>response</i> - A <u>QueryResponse</u> (#QueryResponse) object received by your query handler in response to a

		<u>query</u> (#Query). If this is null, the method will throw a JavaScript error.
removeError(<i>id</i>)	Boolean: true if the error was removed; false otherwise.	Removes the error specified by ID from the page. <ul style="list-style-type: none"> <i>id</i> - The string ID of an error created using addError() or addErrorFromQueryResponse().
removeAll(<i>container</i>)	None	Removes all error blocks from a specified container. If the specified container does not exist, this will throw an error. <ul style="list-style-type: none"> <i>container</i> - The DOM element holding the error strings to remove. If the container cannot be found, the function will throw a JavaScript error.
getContainer(<i>errorId</i>)	Handle to a DOM element holding the error specified, or null if it could not be found.	Retrieves a handle to the container element holding the error specified by <i>errorID</i> . <ul style="list-style-type: none"> <i>errorId</i> - String ID of an error created using addError() or addErrorFromQueryResponse().

Events

Most visualizations fire events to indicate something has occurred. As a user of the chart, you would often want to listen to these events. If you [code your own visualization](https://developers.google.com/chart/interactive/docs/dev/index) (<https://developers.google.com/chart/interactive/docs/dev/index>), you might also want to trigger such events on your own.

The following methods enable developers to listen to events, remove existing event handlers or trigger events from inside a visualization.

- [google.visualization.events.addListener\(\)](#) (#addlistener) and [google.visualization.events.addOneTimeListener\(\)](#) (#addonetimelistener) listen for events.
- [google.visualization.events.removeListener\(\)](#) (#removelistener) removes an existing listener
- [google.visualization.events.removeAllListeners\(\)](#) (#removealllisteners) removes all listeners of a specific chart
- [google.visualization.events.trigger\(\)](#) (#trigger) fires an event.

addListener()

Call this method to register to receive events fired by a visualization hosted on your page. You should document what event arguments, if any, will be passed to the handling function.

```
google.visualization.events.addListener(source_visualization,  
    event_name, handling_function)
```

source_visualization

A handle to the source visualization instance.

event_name

The string name of the event to listen for. A visualization should document which events it throws.

handling_function

The name of the local JavaScript function to call when *source_visualization* fires the *event_name* event. The handling function will be passed any event arguments as parameters.

Returns

A listener handler for the new listener. The handler can be used to later remove this listener if needed by calling [google.visualization.events.removeListener\(\)](#) (#removelistener).

Example

Here is an example of registering to receive the selection event

```
var table = new google.visualization.Table(document.getElementById('table'));  
table.draw(data, options);  
  
google.visualization.events.addListener(table, 'select', selectHandler);  
  
function selectHandler() {  
    alert('A table row was selected');  
}
```

addOneTimeListener()

This is identical to `addListener()`, but is intended for events that should only be listened to once. Subsequent throws of the event will not invoke the handling function.

An example of when this is useful: every draw causes a `ready` event to be thrown. If you want only the first `ready` to execute your code, you'll want `addOneTimeListener` rather than `addListener`.

removeListener()

Call this method to unregister an existing event listener.

```
google.visualization.events.removeListener(listener_handler)
```

listener_handler

The listener handler to remove, as returned by `google.visualization.events.addListener\(\)` (`#addListener`).

removeAllListeners()

Call this method to unregister all event listeners of a specific visualization instance.

```
google.visualization.events.removeAllListeners(source_visualization)
```

source_visualization

A handle to the source visualization instance from which all event listeners should be removed.

trigger()

Called by visualization *implementers*. Call this method from your visualization to fire an event with an arbitrary name and set of values.

```
google.visualization.events.trigger(source_visualization, event_name,  
                                   event_args)
```

source_visualization

A handle to the source visualization instance. If you are calling this function from within a method defined by the sending visualization, you can simply pass in the `this` keyword.

`event_name`

A string name to call the event. You can choose any string value that you want.

`event_args`

[*optional*] A map of name/value pairs to pass to the receiving method. For example: {message: "Hello there!", score: 10, name: "Fred"}. You can pass null if no events are needed; the receiver should be prepared to accept null for this parameter.

Example

Here is an example of a visualization that throws a method named "select" when its onclick method is called. It does not pass back any values.

```
MyVisualization.prototype.onclick = function(rowIndex) {
  this.highlightRow(this.selectedRow, false); // Clear previous selection
  this.highlightRow(rowIndex, true); // Highlight new selection

  // Save the selected row index in case getSelection is called.
  this.selectedRow = rowIndex;

  // Trigger a select event.
  google.visualization.events.trigger(this, 'select', null);
}
```

Standard Visualization Methods and Properties

Every visualization *should* expose the following set of required and optional methods and properties. However, note that there is no type checking to enforce these standards, so you should read the documentation for each visualization.



Note: These methods are in the namespace of the visualization, *not* the `google.visualization` namespace.

Constructor

The constructor should have the name of your visualization class, and return an instance of that class.

```
visualization_class_name(dom_element)
```

dom_element

A pointer to a DOM element where the visualization should be embedded.

Example

```
var org = new google.visualization.OrgChart(document.getElementById('org_
```

draw()

Draws the visualization on the page. Behind the scenes this can be fetching a graphic from a server or creating the graphic on the page using the linked visualization code. You should call this method every time the data or options change. The object should be drawn inside the DOM element passed into the constructor.

```
draw(data[, options])
```

data

A **`DataTable`** (`#DataTable`) or **`DataView`** (`#DataView`) holding the data to use to draw the chart. There is no standard method for extracting a **`DataTable`** from a chart.

options

[*Optional*] A map of name/value pairs of custom options. Examples include height and width, background colors, and captions. The visualization documentation should list which options are available, and should support default options if you do not specify this parameter. You can use the JavaScript object literal syntax to pass in an options map: e.g., `{x:100, y:200, title:'An Example'}`

Example

```
chart.draw(myData, {width: 400, height: 240, is3D: true, title: 'My Daily
```

getAction()

This is optionally exposed by visualizations that have tooltips

(https://developers.google.com/chart/interactive/docs/customizing_tooltip_content) and allow tooltip actions

(https://developers.google.com/chart/interactive/docs/customizing_tooltip_content#tooltip_actions)

.

Returns the tooltip action object with the requested `actionID`.

Example:

```
// Returns the action object with the ID 'alertAction'.
chart.getAction('alertAction');
```

getSelection()

This is optionally exposed by visualizations that want to let you access the currently selected data in the graphic.

```
selection_array getSelection()
```

Returns

selection_array An array of selected objects, each one describing a data element in the underlying table used to create the visualization (a `DataView` or a `DataTable`).

Each object has properties `row` and/or `column`, with the index of the row and/or column of the selected item in the underlying `DataTable`. If the `row` property is null, then the selection is a column; if the `column` property is null, then the selection is a row; if both are non-null, then it is a specific data item. You can call the `DataTable.getValue()` (`#DataTable_getValue`) method to get the value of the selected item. The retrieved array can be passed into `setSelection()` (`#vissetselection`).

Example

```
function myClickHandler(){
  var selection = myVis.getSelection();
  var message = '';

  for (var i = 0; i < selection.length; i++) {
    var item = selection[i];
    if (item.row != null && item.column != null) {
```

```

        message += '{row:' + item.row + ',column:' + item.column + '}';
    } else if (item.row != null) {
        message += '{row:' + item.row + '}';
    } else if (item.column != null) {
        message += '{column:' + item.column + '}';
    }
}
if (message == '') {
    message = 'nothing';
}
alert('You selected ' + message);
}

```

removeAction()

This is optionally exposed by visualizations that have tooltips

(https://developers.google.com/chart/interactive/docs/customizing_tooltip_content) and allow tooltip actions

(https://developers.google.com/chart/interactive/docs/customizing_tooltip_content#tooltip_actions)

.

Removes the tooltip action object with the requested **actionID** from your chart.

Example:

```

// Removes an action from chart with the ID of 'alertAction'.
chart.removeAction('alertAction');

```

setAction()

This is optionally exposed by visualizations that have tooltips

(https://developers.google.com/chart/interactive/docs/customizing_tooltip_content) and allow tooltip actions

(https://developers.google.com/chart/interactive/docs/customizing_tooltip_content#tooltip_actions)

. It works only for core charts (bar, column, line, area, scatter, combo, bubble, pie, donut, candlestick, histogram, stepped area).

Sets a tooltip action to be executed when the user clicks on the action text.

```

setAction(action object)

```

The `setAction` method takes an object as its action parameter. This object should specify 3 properties: `id`— the ID of the action being set, `text` —the text that should appear in the tooltip for the action, and `action` — the function that should be run when a user clicks on the action text.

Any and all tooltip actions should be set prior to calling the chart's `draw()` method.

Example:

```
// Sets a tooltip action which will pop an alert box on the screen when t
chart.setAction({
  id: 'alertAction',
  text: 'Trigger Alert',
  action: function() {
    alert('You have triggered an alert');
  }
});
```

The `setAction` method can also define two additional properties: `visible` and `enabled`. These properties should be functions that return `boolean` values indicating if the tooltip action will be visible and/or enabled.

Example:

```
// The visible/enabled functions can contain any logic to determine their
// as long as they return boolean values.
chart.setAction({
  id: 'alertAction',
  text: 'Trigger Alert',
  action: function() {
    alert('You have triggered an alert');
  },
  visible: function() {
    return true;
  },
  enabled: function() {
    return true;
  }
});
```

setSelection()

Optionally selects a data entry in the visualization—for example, a point in an area chart, or a bar in a bar chart. When this method is called, the visualization should

visually indicate what the new selection is. The implementation of `setSelection()` *should not* fire a "select" event. Visualizations may ignore part of the selection. For example, a table that can show only selected rows may ignore cell or column elements in its `setSelection()` implementation, or it can select the entire row.

Every time this method is called, all selected items are deselected, and the new selection list passed in should be applied. There is no explicit way to deselect individual items; to deselect individual items, call `setSelection()` with the items to *remain selected*; to deselect all elements, call `setSelection()`, `setSelection(null)`, or `setSelection([])`.

```
setSelection(selection_array)
```

selection_array

An array of objects, each with a numeric row and/or column property. `row` and `column` are the zero-based row or column number of an item in the data table to select. To select a whole column, set `row` to null; to select a whole row, set `column` to null. **Example:** `setSelection([{row:0, column:1}, {row:1, column:null}])` selects the cell at (0,1) and the entire row 1.

Assorted Static Methods

This section contains various useful methods exposed in the `google.visualization` namespace.

`arrayToDataTable()`

This method takes in a two-dimensional array and converts it to a `DataTable`.

The column data types are determined automatically by the data provided. Column data types can also be specified using the object literal notation in the first row (the column header row) of the array (i.e. `{label: 'Start Date', type: 'date'}`). [Optional data roles](https://developers.google.com/chart/interactive/docs/roles) (<https://developers.google.com/chart/interactive/docs/roles>) may be used as well, but they must be defined explicitly using object literal notation. Object literal notation may also be used for any cell, allowing you to define [Cell Objects](#) (`#cell_object`)).

Syntax

```
google.visualization.arrayToDataTable(twoDArray, opt_firstRowIsData)
```

twoDArray

A two-dimensional array, where each row represents a row in the data table. If *opt_firstRowsData* is false (the default), the first row will be interpreted as header labels. The data types of each column are interpreted automatically from the data given. If a cell has no value, specify a null or empty value as appropriate.

opt_firstRowsData

Whether the first row defines a header row or not. If true, all rows are assumed to be data. If false, the first row is assumed to be a header row, and the values are assigned as column labels. *Default is false.*

Returns

A new **DataTable**.

Examples

The following code demonstrates three ways to create the same **DataTable** object:

```
// Version 1: arrayToDataTable method
var data2 = google.visualization.arrayToDataTable([
  [{label: 'Country', type: 'string'},
   {label: 'Population', type: 'number'},
   {label: 'Area', type: 'number'},
   {type: 'string', role: 'annotation'}],
  ['CN', 1324, 9640821, 'Annotated'],
  ['IN', 1133, 3287263, 'Annotated'],
  ['US', 304, 9629091, 'Annotated'],
  ['ID', 232, 1904569, 'Annotated'],
  ['BR', 187, 8514877, 'Annotated']
]);

// Version 2: DataTable.addRows
var data3 = new google.visualization.DataTable();
data3.addColumn('string', 'Country');
data3.addColumn('number', 'Population');
data3.addColumn('number', 'Area');
data3.addRows([
  ['CN', 1324, 9640821],
  ['IN', 1133, 3287263],
  ['US', 304, 9629091],
  ['ID', 232, 1904569],
  ['BR', 187, 8514877]
```

```

]);

// Version 3: DataTable.setValue
var data = new google.visualization.DataTable();
data.addColumn('string', 'Country');
data.addColumn('number', 'Population');
data.addColumn('number', 'Area');
data.addRows(5);
data.setValue(0, 0, 'CN');
data.setValue(0, 1, 1324);
data.setValue(0, 2, 9640821);
data.setValue(1, 0, 'IN');
data.setValue(1, 1, 1133);
data.setValue(1, 2, 3287263);
data.setValue(2, 0, 'US');
data.setValue(2, 1, 304);
data.setValue(2, 2, 9629091);
data.setValue(3, 0, 'ID');
data.setValue(3, 1, 232);
data.setValue(3, 2, 1904569);
data.setValue(4, 0, 'BR');
data.setValue(4, 1, 187);
data.setValue(4, 2, 8514877);

```

drawChart()

This method creates a chart in a single call. The advantage of using this method is that it requires slightly less code, and you can serialize and save visualizations as text strings for reuse. This method does not return a handle to the created chart, so you cannot assign method listeners to catch chart events.

Syntax

```
google.visualization.drawChart(chart_JSON_or_object)
```

chart_JSON_or_object

Either a JSON (<http://www.json.org/>) literal string or a JavaScript object, with the following properties (case-sensitive):

Property	Type	Required	Default	Description
chartType	String	Required	none	The class name of the visualization. The <code>google.visua</code> package name can be omitted for Google charts. If the <code>ap</code> visualization library has not already been loaded, this met

				the library for you if this is a Google visualization; you must load the library explicitly. Examples: <code>Table</code> , <code>PieChart</code> , <code>com.CrazyChart</code> .
containerId	String	Required	none	The ID of the DOM element on your page that will host the visualization.
options	Object	Optional	none	An object describing the options for the visualization. You can use either JavaScript literal notation, or provide a handle to the <code>google.visualization</code> namespace. Example: <code>"options": {"width": 400, "height": 300, "is3D": true, "title": "Company Performance"}</code>
dataTable	Object	Optional	None	A DataTable used to populate the visualization. This can be a JSON string representation of a DataTable , as described in DataTable (#dataparam), or a handle to a populated <code>google.visualization.DataTable</code> object, or a 2-dimensional array like that accepted by <code>arrayToDataTable(opt_firstRowIsData=false)</code> (<code>#google.visualization.arraytodatatable</code>). You must specify either the <code>dataSourceUrl</code> property or the <code>dataTable</code> property.
dataSourceUrl	String	Optional	None	A data source query to populate the chart data (for example, Spreadsheet (<code>https://developers.google.com/chart/interactive/docs/s</code>). You must specify either this property or the <code>dataTable</code> property.
query	String	Optional	None	If specifying <code>dataSourceUrl</code> , you can optionally specify a query string using the Visualization query language (<code>https://developers.google.com/chart/interactive/docs/c</code>) to filter or manipulate the data.
refreshInterval	Number	Optional	None	How often, in seconds, the visualization should refresh its data. Use this only when specifying <code>dataSourceUrl</code> .
view	Object OR Array	Optional	None	Sets a DataView initializer object, which acts as a filter on the underlying data, as defined by either the <code>dataTable</code> or <code>dataSourceUrl</code> parameter. You can pass in either a string or a DataView initializer object, like that returned by <code>dataView.fromDataTable</code> (#DataView_toJSON). Example: <code>"view": {"columns": ["A", "D"]}</code> . You can also pass in an array of DataView initializer objects; in this case the first DataView in the array is applied to the underlying data to create a new data table, and the second DataView is applied to that data table resulting from application of the first DataView .

Examples

Creates a table chart based on a spreadsheet data source, and includes the query `SELECT A,D WHERE D > 100 ORDER BY D`

```
<script type="text/javascript">
  google.charts.load('current'); // Note: No need to specify chart package
```

```

function drawVisualization() {
  google.visualization.drawChart({
    "containerId": "visualization_div",
    "dataSourceUrl": "https://spreadsheets.google.com/a/google.com/tq?k",
    "query": "SELECT A,D WHERE D > 100 ORDER BY D",
    "refreshInterval": 5,
    "chartType": "Table",
    "options": {
      "alternatingRowStyle": true,
      "showRowNumber" : true
    }
  });
}
google.charts.setOnLoadCallback(drawVisualization);
</script>

```

This next example creates the same table, but creates a **DataTable** locally:

```

<script type='text/javascript'>
  google.charts.load('current');
  function drawVisualization() {
    var dataTable = [
      ["Country", "Population Density"],
      ["Indonesia", 117],
      ["China", 137],
      ["Nigeria", 142],
      ["Pakistan", 198],
      ["India", 336],
      ["Japan", 339],
      ["Bangladesh", 1045]
    ];
    google.visualization.drawChart({
      "containerId": "visualization_div",
      "dataTable": dataTable,
      "refreshInterval": 5,
      "chartType": "Table",
      "options": {
        "alternatingRowStyle": true,
        "showRowNumber" : true,
      }
    });
  }
  google.charts.setOnLoadCallback(drawVisualization);
</script>

```

This example passes in a JSON string representation of the chart, which you might have loaded from a file:

```

<script type="text/javascript">
  google.charts.load('current');
  var myStoredString = '{"containerId": "visualization_div",' +
    '"dataSourceUrl": "https://spreadsheets.google.com/a/google.com/tq?ke'
    '"query": "SELECT A,D WHERE D > 100 ORDER BY D",' +
    '"refreshInterval": 5,' +
    '"chartType": "Table",' +
    '"options": {' +
    '  "alternatingRowStyle": true,' +
    '  "showRowNumber" : true' +
    '}' +
    '}'';
  function drawVisualization() {
    google.visualization.drawChart(myStoredString);
  }
  google.charts.setOnLoadCallback(drawVisualization);
</script>

```

drawToolbar()

This is the constructor for the toolbar element that can be attached to many visualizations. This toolbar enables the user to export the visualization data into different formats, or to provide an embeddable version of the visualization for use in different places. See the [toolbar page](#)

(<https://developers.google.com/chart/interactive/docs/gallery/toolbar>) for more information and a code example.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](http://creativecommons.org/licenses/by/3.0/) (<http://creativecommons.org/licenses/by/3.0/>), and code samples are licensed under the [Apache 2.0 License](http://www.apache.org/licenses/LICENSE-2.0) (<http://www.apache.org/licenses/LICENSE-2.0>). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (<https://developers.google.com/terms/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

上次更新日期: 二月 23, 2017