**Google Visualization API Reference**

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

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

1. [**DataTable**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable)
2. [**DataView**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataView)
3. [**Formatters**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#formatters)
   1. [ArrowFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#arrowformatter)
   2. [BarFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#barformatter)
   3. [ColorFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#colorformatter)
   4. [DateFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#dateformatter)
   5. [NumberFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#numberformatter)
   6. [PatternFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#patternformatter)
4. [**GadgetHelper**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#GadgetHelper)
5. [**Query**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#Query)
6. [**QueryResponse**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#QueryResponse)
7. [**drawToolbar()**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#drawToolbar)
8. [**Errors**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#errordisplay)
9. [**Events**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#events)
10. [**Standard Visualization Methods and Properties**](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#standardproperties)

**google.visualization.DataTable**

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](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataView).

Each column is assigned a data type, plus several optional properties including an ID and label. In addition, you can assign custom properties (name/value pairs) to any cell, row, column, or the entire table. These custom properties are defined and consumed by some visualizations, as described in their documentation. For an example of a custom property, see the className property consumed by the [Table visualization](http://code.google.com/intl/zh-CN/apis/visualization/documentation/gallery/table.html#customproperties).

**See also:** [QueryResponse.getDataTable](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "QueryResponse_getDataTable)

**Constructors**

The DataTable class has two constructors:

google.visualization.DataTable() // Create an empty DataTable instance

google.visualization.DataTable(*data*, *version*) // Create a populated DataTable instance

**Parameters**

***data***

[*Required*] A JavaScript object containing data used to initialize the table. The structure of this object is described later in this section. If you do not use this parameter, you can populate the table manually using the various add or remove methods.

***version***

[*Required*] A numeric value specifying the version of the wire protocol used. *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](http://code.google.com/intl/zh-CN/apis/visualization/documentation/gallery/table.html#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()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "addrow) for a single row, or [addRows()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_addRows) for multiple rows. You can add columns as well by calling the [addColumn()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "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.

**So 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()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_addColumn)/[addRows()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "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 setValue() 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.

**Example: Creating and Populating a DataTable in JavaScript**

The following example demonstrates creating and populating a two column, three row DataTable in JavaScript. You might use this on a page hosting a visualization to create the data for that visualization.

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'}]

]);

**Example: Creating a Table using JavaScript Object Literal Notation**

You can also populate a table by passing in an object that hosts data when you instantiate it by specifying the *data* object. This object is a JavaScript object formatted in a specific way (described below in [Format of the *data* Parameter Object](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#dataparam)). The following example demonstrates instantiating and populating a DataTablewith 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

)

**Format of the *data* Parameter Object**

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](http://code.google.com/intl/zh-CN/apis/visualization/documentation/dev/gviz_api_lib.html) 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 theDataTable has come from a data source in response to a query with a [format](http://code.google.com/intl/zh-CN/apis/visualization/documentation/querylanguage.html#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) and [SimpleDateFormat](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.

**Cells**

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 null, the whole object should be empty and have neither v nor f properties.
* f [*Optional*] A string version of the v value, formatted for display. The values should match, 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 used.
* 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;'}.

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

]

}

**p property**

The 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 ignored.**Example:** p:{className: 'myDataTable'}.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Value** | **Description** |
| addColumn(type [,label [,id]]) | number | 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.   * 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' 'datetime' 'timeofday'. * 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 not value is specified, an empty string is assigned. * id should be a string with a unique identifier for the column. If not value is specified, an empty string is assigned.   **See also:** [getColumnId](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getColumnId) [getColumnLabel](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getColumnLabel) [getColumnType](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getColumnType) [insertColumn](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_insertColumn) |
| addRow(opt\_cellArray) | number | Adds a new row to the data table, and returns the index of the new row.   * opt\_cellArray [*optional*] A row object, in JavaScript notation, specifying the data for the new row. If not included, this method will simply add a new, empty row to the end of the table. This parameter is an array of simple cell values, and/or [cell objects](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#cell_object) (you can mix simple values and objects in the same method call)   **Examples:**  data.addRow(); // Add an empty row  data.addRow(['Hermione', new Date(1999,0,1)]); // Add a row with a string and a date value.  // Add a row with two cells, the second with a formatted value.  data.addRow(['Hermione', {v: new Date(1999,0,1),  f: 'January First, Nineteen ninety-nine'}]); |
| addRows(numOrArray) | number | 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.   * numOrArray - Either a number or an array:   + **Number** - A number specifying how many new, unpopulated rows to add.   + **Array** - An array of [row](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#rowsproperty) objects used to populate a set of new rows. Each row is an object as described in addRow().   **Example:**  data.addRows([['Ivan', new Date(1977,2,28)],  ['Igor', new Date(1962,7,5)],  ['Felix', new Date(1983,11,17)]]);  **See also:** [insertRows](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_insertRows) |
| clone() | [DataTable](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable) | Returns a clone of the data table. The result is a deep copy of the data table except for the [cell properties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getProperty), [row properties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getRowProperty), [table properties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getTableProperty) and [column properties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getColumnProperty), which are shallow copies. |
| getColumnId(columnIndex) | string | 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 [query language](http://code.google.com/intl/zh-CN/apis/visualization/documentation/querylanguage.html).  **See also:** [Query.setQuery](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "Query_setQuery) |
| getColumnLabel(columnIndex) | string | 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 thelabel clause of the [query language](http://code.google.com/intl/zh-CN/apis/visualization/documentation/querylanguage.html#Label).  **See also:** [setColumnLabel](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setColumnLabel) |
| getColumnPattern(columnIndex) | string | Returns the formatting pattern used to format the values of the specified column.   * columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method.   For data tables that are retrieved by queries, The column pattern is set by the data source, or by theformat clause of the query language. An example of a pattern is '#,##0.00'. For more on patterns see the [query language reference](http://code.google.com/intl/zh-CN/apis/visualization/documentation/querylanguage.html#Format). |
| getColumnProperty(columnIndex, name) | Object | Returns the value of a named property, or null if no such property is set for the specified column.   * 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.   **See also:** [setColumnProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setColumnProperty) [setColumnProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setColumnProperties) |
| getColumnRange(columnIndex) | Object | 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.  columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method. |
| getColumnType(columnIndex) | string | Returns the type of a given column specified by the column index.   * columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method.   The returned column type can be one of the following: 'string' 'number' 'boolean' 'date' 'datetime' 'timeofday' |
| getDistinctValues(columnIndex) | Array of Objects | Returns the unique values in a certain column, in ascending order.   * columnIndex should be a number greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method.   The type of the returned objects is the same as that returned by the [getValue](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getValue) method. |
| getFilteredRows(filters) | Array of numbers | Returns the row indexes for rows that match all of the given filters. The indexes are returned in ascending order.  filters - An array of filter objects that describe an acceptable row. 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, and 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.   **Example**: getFilteredRows([{column: 3, value: 42}, {column: 2, minValue: 'bar', maxValue: 'foo'}]) 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). |
| getFormattedValue(rowIndex, columnIndex) | string | Returns the formatted value of the cell at the given row and column indexes.   * 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.   For more on formatting column values see the [query language reference](http://code.google.com/intl/zh-CN/apis/visualization/documentation/querylanguage.html#Format).  **See also:** [setFormatedValue](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setFormattedValue) |
| getNumberOfColumns() | number | Returns the number of columns in the table. |
| getNumberOfRows() | number | Returns the number of rows in the table. |
| getProperty(rowIndex, columnIndex, name) | Object | Returns the value of a named property, or null if no such property is set for the specified cell.   * 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.   **See also:** [setCell](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setCell) [setProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setProperties) [setProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setProperty) |
| getRowProperty(rowIndex, name) | Object | Returns the value of a named property, or null if no such property is set for the specified row.   * 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.   **See also:** [setRowProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setRowProperty) [setRowProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setRowProperties) |
| getSortedRows(sortColumns) | Array of numbers | Returns a sorted version of the table without modifying the order of the underlying data. To permanently sort the underlying data, call [sort()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_sort). You can specify sorting in a number of ways, depending on the type you pass in to the sortColumns parameter:   * **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 specifiedsortColumns.  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](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_sort)  **Example**: To iterate on rows ordered by the third column, use:  var rowInds = data.getSortedRows([{column: 2}]);  for (var i = 0; i < rowInds.length; i++) {  var v = data.getValue(rowInds[i], 2);  } |
| getTableProperty(name) | Object | Returns the value of a named property, or null if no such property is set for the table.   * name is a string with the property name.   **See also:** [setTableProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setTableProperties) [setTableProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setTableProperty) |
| getValue(rowIndex, columnIndex) | object | Returns the value of the cell at the given row and column indexes.   * 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.   The type of the returned value depends on the column type (see [getColumnType](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getColumnType)):   * 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, millisenconds]. * If the column value is a null value, regardless of the column type, the returned value is null. |
| insertColumn(columnIndex, type [,label [,id]]) | None | 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.   * 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' 'datetime' 'timeofday'. * 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.   **See also:** [addColumn](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_addColumn) |
| insertRows(rowIndex, numberOrArray) | None | Insert the specified number of rows at the specified row index.   * 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()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_addRows) for the syntax for adding an array of row objects.   **See also:** [addRows](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_addRows) |
| removeColumn(columnIndex) | None | Removes the column at the specified index.   * columnIndex should be a number with a valid column index.   **See also:** [removeColumns](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_removeColumns) |
| removeColumns(columnIndex, numberOfColumns) | None | Removes the specified number of columns starting from the column at the specified index.   * numberOfColumns is the number of columns to remove. * columnIndex should be a number with a valid column index.   **See also:** [removeColumn](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_removeColumn) |
| removeRow(rowIndex) | None | Removes the row at the specified index.   * rowIndex should be a number with a valid row index.   **See also:** [removeRows](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_removeRows) |
| removeRows(rowIndex, numberOfRows) | None | Removes the specified number of rows starting from the row at the specified index.   * numberOfRows is the number of rows to remove. * rowIndex should be a number with a valid row index.   **See also:** [removeRow](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_removeRow) |
| setCell(rowIndex, columnIndex, value [, formattedValue [, properties]]) | None | Sets the value, and optionally the formatted value and properties, of a cell. To simply change the cell value, use [setValue](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setValue)   * 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 is the value assigned to the specified cell. The type of the returned value depends on the column type (see [getColumnType](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "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, millisenconds].   + For any column type, the value can be set to null. * formattedValue is a string with the value formatted as a string. If null is specified, or if this parameter is omitted, the default formatting will be applied. 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 is an optional Object (name/value map) with additional properties for this cell. If nullis specified, or if this parameter is omitted, no additional properties are assigned to this cell. Some visualizations support row, column, or cell properties to modify their display or behavior; see the visualization documentation to see what properties are supported. |
| setColumnLabel(columnIndex, label) | None | Sets the label of a column.   * 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.   **See also:** [getColumnLabel](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getColumnLabel) |
| setColumnProperty(columnIndex, name, value) | None | 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.   * 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.   **See also:** [setColumnProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setColumnProperties) [getColumnProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getColumnProperty) |
| setColumnProperties(columnIndex, properties) | None | 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.   * 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.   **See also:** [setColumnProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setColumnProperty) [getColumnProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getColumnProperty) |
| setFormattedValue(rowIndex, columnIndex, formattedValue) | None | Sets the formatted value of a cell.   * 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.   **See also:** [getFormattedValue](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getFormattedValue) |
| setProperty(rowIndex, columnIndex, name, value) | None | 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.   * 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. * value is a value of any type to assign to the specified named property of the specified cell.   **See also:** [setCell](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setCell) [setProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setProperties) [getProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getProperty) |
| setProperties(rowIndex, columnIndex, properties) | None | 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.   * 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.   **See also:** [setCell](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setCell) [setProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setProperty) [getProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getProperty) |
| setRowProperty(rowIndex, name, value) | None | Sets a row 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.   * 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.   **See also:** [setRowProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setRowProperties) [getRowProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getRowProperty) |
| setRowProperties(rowIndex, properties) | None | 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.   * 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.   **See also:** [setRowProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setRowProperty) [getRowProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getRowProperty) |
| setTableProperty(name, value) | None | Sets a single table property. 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.   * 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.   **See also:** [setTableProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setTableProperties) [getTableProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getTableProperty) |
| setTableProperties(properties) | None | Sets multiple table table. 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.   * 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.   **See also:** [setTableProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setTableProperty) [getTableProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getTableProperty) |
| setValue(rowIndex, columnIndex, value) | None | 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.   * 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()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setFormattedValue). * value is the value assigned to the specified cell. The type of the returned value depends on the column type (see [getColumnType](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "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, millisenconds].   + For any column type, the value can be set to null.   See also: [setCell](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setCell), [setFormattedValue](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setFormattedValue), [setProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setProperty), [setProperties](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_setProperties) |
| sort(sortColumns) | None | Sorts the rows, according to the specified sort columns. The DataTable is modified by this method. See [getSortedRows()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getSortedRows) for a description of the sorting details. This method does not return the sorted data. **See also:** [getSortedRows](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getSortedRows)  **Example**: To sort by the third column and then by the second column, use: data.sort([{column: 2}, {column: 1}]); |

**google.visualization.DataView**

A read-only view of an underlying [DataTable](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "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 should 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 in the view, and might cause unexpected behavior in the view.
* Adding or removing *rows* from the underlying table is safe and you will see changes immediately. However, if you have filtered out rows (by calling one of thesetRows() or hideRows() methods) your behavior might be unexpected.
* Changing *cell values* in existing cells is safe, and is immediately visible in the view.

It is also possible to create a DataView on an underlying 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.

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\_dataview'));

table.draw(view, {sortColumn: 1});

**Constructor**

google.visualization.DataView(*data*)

**Parameters**

***data***

A DataTable or DataView 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()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataView_setColumns), [hideColumns()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataView_hideColumns), [setRows()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataView_setRows), [hideRows()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataView_hideRows).

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Value** | **Description** |
| * [getColumnId](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getColumnId) * [getColumnLabel](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getColumnLabel) * [getColumnPattern](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getColumnPattern) * [getColumnProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getColumnProperty) * [getColumnRange](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getColumnRange) * [getColumnType](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getColumnType) * [getDistinctValues](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getDistinctValues) * [getFilteredRows](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getFilteredRows) * [getFormattedValue](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getFormattedValue) * [getNumberOfColumns](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getNumberOfColumns) * [getNumberOfRows](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getNumberOfRows) * [getProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getProperty) * [getRowProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getRowProperty) * [getSortedRows](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getSortedRows) * [getTableProperty](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getTableProperty) * [getValue](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable_getValue) | See descriptions inDataTable. | 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. |
| 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 getNumberOfColumns() method.  **Example**: If setColumns([3, 1, 4]) was previously called, then getTableColumnIndex(2)will return 4. |
| getTableRowIndex(viewRowIndex) | number | 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.  **Example**: If setRows([3, 1, 4]) was previously called, then getTableRowIndex(2) will return4. |
| getViewColumnIndex(tableColumnIndex) | number | 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. tableColumnIndexshould 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.  **Example**: If setColumns([3, 1, 4]) was previously called, then getViewColumnIndex(4) will return 2. |
| getViewColumns() | Array of numbers | 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. |
| getViewRowIndex(tableRowIndex) | number | 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 thegetNumberOfRows() method of the underlying table/view.  **Example**: If setRows([3, 1, 4]) was previously called, then getViewRowIndex(4) will return2. |
| getViewRows() | Array of numbers | Returns the rows in this view, in order. That is, if you call setRows with some array, and then callgetViewRows() you should get an identical array. |
| hideColumns(columnIndexes) | none | 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*underlying table/view*. 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().  **Example**: If you have a table with 10 columns, and you call setColumns([2,7,1,7,9]), and then hideColumns([7,9]), the columns in the view will then be [2,1]. |
| hideRows(min, max) | none | Hides all rows with indexes that lie between min and max (inclusive) from the current view. This is a convenience syntax for hideRows(rowIndexes) above. For example, hideRows(5, 10) is equivalent to hideRows([5, 6, 7, 8, 9, 10]). |
| hideRows(rowIndexes) | none | 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 *underlying table/view*. 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 setRows().  **Example**: If you have a table with 10 rows, and you call setRows([2,7,1,7,9]), and thenhideRows([7,9]), the rows in the view will then be [2,1]. |
| setColumns(columnIndexes) | None | Sets the columns in this view based on indexes from the underlying table/view. columnIndexesshould be an array of numbers, greater than or equal to zero, and less than the number of columns as returned by the getNumberOfColumns() method of the underlying table/view. The specified column indexes are the indexes in the underlying table/view, which will be in the view, in the specified order. Note that *only* the columns specified in columnIndexes will be shown; this method clears all other columns from the view. The array can also contain duplicates, effectively duplicating the specified column in this view (for example, setColumns([3, 4, 3, 2]) will cause column 3 to appear twice in the view). The array thus provides a mapping of the columns from the underlying table/view to this view.  **Example**: To create a view with column three and zero of an underlying table/view:view.setColumns([3, 0]) |
| setRows(min, max) | none | 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) above. For example, setRows(5, 10) is equivalent to setRows([5, 6, 7, 8, 9, 10]) |
| setRows(rowIndexes) | None | Sets the rows in this view based on indexes from the underlying table/view. rowIndexes should be an array of numbers, greater than or equal to zero, and less than the number of rows as returned by the getNumberOfRows() method of the underlying table/view. The specified row indexes are the indexes in the underlying table/view, which will be in the view, in the specified order. Note that *only* the rows specified in rowIndexes will bw shown; this method clears all other rows from the view. The array can also contain duplicates, effectively duplicating the specified row in this view (for example, setRows([3, 4, 3, 2]) 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.  **Example**: To create a view with rows three and zero of an underlying table/view:view.setRows([3, 0]) |

**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 it does not modify the underlying values; just the formatted values. So, for example, the displayed value would be "$1,000.00" but the underlying value would still be "1000". Formatters can only affect one column at a time; to reformat multiple columns, apply a formatter to each column that you want to change.

**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:
   1. 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.)
   2. Create your formatter, passing in your options object.
   3. Call *formatter*.Format(*table*, *colIndex*), passing in the DataTable and the (zero-based) column number of the data to reformat.  
      ***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('dateformat\_div'));  
table.draw(data, {showRowNumber: true});

Most formatters expose the following two methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| google.visualization.*formatter\_name*(*options*) | Constructor, where *formatter\_name* is a specfic formatter class name.   * *options* - 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:   // 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); |
| format(*data*, *colIndex*) | Reformats the data in the specified column.   * *data* - A [DataTable](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable) containing the data to reformat. You cannot use a DataView here. * *colIndex* - The zero-based index of the column to format. To format multiple columns, you must call this method multiple times, with different colIndex values. |

The Google Visualization API provides the following formatters:

|  |  |
| --- | --- |
| **Formatter Name** | **Description** |
| [ArrowFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#arrowformatter) | Adds an up or down arrow, indicating whether the cell value is above or below a specified value. |
| [BarFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#barformatter) | Adds a colored bar, the direction and color of which indicates whether the cell value is above or below a specified value. |
| [ColorFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#colorformatter) | Colors a cell according to whether the values fall within a specified range. |
| [DateFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#dateformatter) | Formats a Date or DateTime value in several different ways, including "January 1, 2009," "1/1/09" and "Jan 1, 2009." |
| [NumberFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#numberformatter) | Formats various aspects of numeric values. |
| [PatternFormat](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#patternformatter) | Concatenates cell values on the same row into a specified cell, along with arbitrary text. |
|  |  |

**google.visualization.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.

**Options**

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

|  |  |
| --- | --- |
| **Option** | **Description** |
| 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. |

**Example**

ArrowFormat Example

  var data = new google.visualization.DataTable();  
  data.addColumn('string', 'Department');  
  data.addColumn('number', 'Revenues Change');  
  data.addRows(5);  
  data.setCell(0, 1, 12, '12.0%');  
  data.setCell(1, 0, 'Sports');  
  data.setCell(1, 1, -7.3, '-7.3%');  
  data.setCell(2, 0, 'Toys');  
  data.setCell(2, 1, 0, '0%');  
  data.setCell(3, 0, 'Electronics');  
  data.setCell(3, 1, -2.1, '-2.1%');  
  data.setCell(4, 0, 'Food');  
  data.setCell(4, 1, 22, '22.0%');  
  
  var table = new google.visualization.Table(document.getElementById('arrowformat\_div'));  
   **var formatter = new google.visualization.ArrowFormat();  
  formatter.format(data, 1); // Apply formatter to second column**   
  table.draw(data, {**allowHtml: true**, showRowNumber: true});

**google.visualization.BarFormat**

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

**Options**

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

|  |  |
| --- | --- |
| **Option** | **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 is 'false'. |
| max | The maximum number value for the bar range. By default, it is the highest value in the table. |
| min | The minimum number value for the bar range. By default, it is the lowest value in the table. |
| showValue | If true, shows values and bars; if false, shows only bars. Default, it is true. |
| width | Thickness of each bar, in pixels. By default, it is 100. |

**Example**

BarFormat Example

  var data = new google.visualization.DataTable();  
  data.addColumn('string', 'Department');  
  data.addColumn('number', 'Revenues');  
  data.addRows(6);  
  data.setCell(0, 0, 'Shoes');  
  data.setCell(0, 1, 10700);  
  data.setCell(1, 0, 'Sports');  
  data.setCell(1, 1, -15400);  
  data.setCell(2, 0, 'Toys');  
  data.setCell(2, 1, 12500);  
  data.setCell(3, 0, 'Electronics');  
  data.setCell(3, 1, -2100);  
  data.setCell(4, 0, 'Food');  
  data.setCell(4, 1, 22600);  
  data.setCell(5, 0, 'Art');  
  data.setCell(5, 1, 1100);  
  
  var table = new google.visualization.Table(document.getElementById('barformat\_div'));  
   **var formatter = new google.visualization.BarFormat({width: 120});  
  formatter.format(data, 1); // Apply formatter to second column**   
  table.draw(data, {**allowHtml: true**, showRowNumber: true});

**google.visualization.ColorFormat**

Assigns colors to the foreground or background of a numeric cell, depending on the cell value. This formatter is an 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".

**Methods**

ColorFormat supports the following methods:

|  |  |
| --- | --- |
| **Option** | **Description** |
| addRange(*from*, *to*,*color*, *bgcolor*) | Specifies a foreground color and/or background color to a cell, depending on the cell value. Any cell with a value in the specified*from*—*to* range (**non-inclusive**) will be assigned *color* and *bgcolor*. 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!   * *from* - [*String, Number, Date, DateTime, or TimeOfDay*] The lower boundary (non-inclusive) of the range, or null. If null, it will match -∞. String boundaries are compared alphabetically against string values. * *to* - [*String, Number, Date, DateTime, or TimeOfDay*] The high boundary (non-inclusive) of the range, or null. If null, it will match +∞. String boundaries are compared alphabetically against string values. * *color* - The color to apply to text in matching cells. Values can be either '#RRGGBB' values or defined color constants, (example: '#FF0000' or 'red'). * *bgcolor* - 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(*from*,*to,*c*olor,fromBgColor*,*toBgColor*) | Assigns a background color from a range, according to the cell value. The color 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.*Tip*: 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).   * *from* - [*Number, Date, DateTime, or TimeOfDay*] The lower boundary (non-inclusive) of the range, or null. If null, it will match -∞. * *to* - [*Number, Date, DateTime, or TimeOfDay*] The higher boundary (non-inclusive) of the range, or null. If null, it will match +∞. * *color* - The color to apply to text in matching cells. This color is the same for all cells, no matter what their value. * *fromBgColor* - 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'). * *toBgColor* - 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(*dataTable*,*columnIndex*) | The standard format() method to apply formatting to the specified column. |
| ColorFormat() | Constructor. Takes no arguments. |

**Example**

ColorFormat Example

  var data = new google.visualization.DataTable();  
  data.addColumn('string', 'Department');  
  data.addColumn('number', 'Revenues');  
  data.addRows(6);  
  data.setCell(0, 0, 'Shoes');  
  data.setCell(0, 1, 10700);  
  data.setCell(1, 0, 'Sports');  
  data.setCell(1, 1, -15400);  
  data.setCell(2, 0, 'Toys');  
  data.setCell(2, 1, 12500);  
  data.setCell(3, 0, 'Electronics');  
  data.setCell(3, 1, -2100);  
  data.setCell(4, 0, 'Food');  
  data.setCell(4, 1, 22600);  
  data.setCell(5, 0, 'Art');  
  data.setCell(5, 1, 1100);  
  
  var table = new google.visualization.Table(document.getElementById('colorformat\_div'));  
   **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});

**google.visualization.DateFormat**

Formats a JavaScript Date value in a variety of ways, including "January 1, 2009," "1/1/09" and "Jan 1, 2009.

**Options**

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

|  |  |
| --- | --- |
| **Option** | **Description** |
| formatType | A quick formatting option for the date. The following string values are supported, reformatting the date February 28, 2008 as shown:   * 'short' - Short format: e.g., "2/28/08" * 'medium' - Medium format: e.g., "Feb 28, 2008" * 'long' - Long format: e.g., "February 28, 2008"   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). 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. |

**Example**

DateFormat Example

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.addRows(3);  
  data.setCell(0, 0, 'Mike');  
  data.setCell(0, 1, new Date(2008, 1, 28, 0, 31, 26));  
  data.setCell(0, 2, new Date(2008, 1, 28, 0, 31, 26));  
  data.setCell(0, 3, new Date(2008, 1, 28, 0, 31, 26));  
  data.setCell(1, 0, 'Bob');  
  data.setCell(1, 1, new Date(2007, 5, 1, 0));  
  data.setCell(1, 2, new Date(2007, 5, 1, 0));  
  data.setCell(1, 3, new Date(2007, 5, 1, 0));  
  data.setCell(2, 0, 'Alice');  
  data.setCell(2, 1, new Date(2006, 7, 16));  
  data.setCell(2, 2, new Date(2006, 7, 16));  
  data.setCell(2, 3, 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('dateformat\_div'));  
  table.draw(data, {showRowNumber: true});  
}

**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), 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:   * M produces 1 * MM produces 01 * MMM produces Jan * MMMM produces January | "July"  "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 Hk' 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":   * 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 |

**google.visualization.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.

**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. |
| prefix | A string prefix for the value, for example "$". |
| suffix | A string suffix for the value, for example "%". |

**Example**

NumberFormat Example

  var data = new google.visualization.DataTable();  
  data.addColumn('string', 'Department');  
  data.addColumn('number', 'Revenues');  
  data.addRows(6);  
  data.setCell(0, 0, 'Shoes');  
  data.setCell(0, 1, 10700);  
  data.setCell(1, 0, 'Sports');  
  data.setCell(1, 1, -15400);  
  data.setCell(2, 0, 'Toys');  
  data.setCell(2, 1, 12500);  
  data.setCell(3, 0, 'Electronics');  
  data.setCell(3, 1, -2100);  
  data.setCell(4, 0, 'Food');  
  data.setCell(4, 1, 22600);  
  data.setCell(5, 0, 'Art');  
  data.setCell(5, 1, 1100);  
  
  var table = new google.visualization.Table(document.getElementById('numberformat\_div'));  
   **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});

**google.visualization.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 theformat() method. See the Methods section below for instructions.

**Methods**

PatternFormat supports the following methods:

|  |  |
| --- | --- |
| **Option** | **Description** |
| google.visualization.PatternFormat(*pattern*) | Constructor. Does not take an options object. Instead, it takes a string *pattern* 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 {#}, where # is a the index of a source column to use. The index is an index in the*srcColumnIndices* array from the format() method below. To include a literal { or } character, escape it like this: \{ or \}. To include a literal \ mark, escape it as \\.  **Example:**The following example demonstrates a constructor for a pattern that creates an anchor element, with the first and second elements taken from the format() method:  var formatter = new google.visualization.PatternFormat('<a href="mailto:{1}">{0}</a>'); |
| format(*dataTable*, *srcColumnIndices*,*opt\_dstColumnIndex*) | The standard formatting call, with a few additional parameters:   * *dataTable* - The DataTable on which to operate. * *srcColumnIndices* - 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 *srcColumIndices* 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 columnss, 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

**Example**

The following example demonstrates how to combine data from two columns to create an email address. It uses a [DataView](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataView) object to hide the original source columns:

PatternFormat Example

  var data = new google.visualization.DataTable();  
  data.addColumn('string', 'Name');  
  data.addColumn('string', 'Email');  
  data.addRows(4);  
  data.setCell(0, 0, 'John Lennon');  
  data.setCell(0, 1, 'john@beatles.co.uk');  
  data.setCell(1, 0, 'Paul McCartney');  
  data.setCell(1, 1, 'paul@beatles.co.uk');  
  data.setCell(2, 0, 'George Harrison');  
  data.setCell(2, 1, 'george@beatles.co.uk');  
  data.setCell(3, 0, 'Ringo Starr');  
  data.setCell(3, 1, 'ringo@beatles.co.uk');  
  
  var table = new google.visualization.Table(document.getElementById('patternformat\_div'));  
   **var formatter = new google.visualization.PatternFormat('<a href="mailto:{1}">{0}</a>');  
  formatter.format(data, [0, 1]); // Apply formatter and set the formatted value of the first column.  
  
  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});

**google.visualization.GadgetHelper**

A helper class to simplify writing [Gadgets](http://code.google.com/apis/gadgets/) that use the Google Visualization API.

**Constructor**

google.visualization.GadgetHelper()

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Value** | **Description** |
| createQueryFromPrefs(prefs) | [google.visualization.Query](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#Query) | Static. Create a new instance of google.visualization.Query and set its properties according to values from the gadget preferences. The type of parameter prefs is [\_IG\_Prefs](http://code.google.com/apis/gadgets/docs/reference.html#Core)   1. Preference \_table\_query\_url is used to set the Query data source URL. 2. Preference \_table\_query\_refresh\_interval is used to set the Query refresh interval (in seconds). |
| validateResponse(response) | boolean | Static. Parameter response is of type [google.visualization.QueryResponse](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "QueryResponse). Returns true if the response contains data. Returns false if the query execution failed and the response does not contain data. If an error occured, this method displays an error message. |

**google.visualization.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. The data source should expose this URL in some way; for example, to get the dataSourceUrl from a [Google Spreadsheet](http://docs.google.com), do the following:

1. In your spreadsheet, select the range of cells.
2. Select 'Insert' and then 'Gadget' from the menu.
3. Open the gadget's menu by clicking on the top-right selector.
4. Select menu option 'Get data source URL'.

***opt\_options***

[*Optional, Object*] A map of options for the request. **Note:**If you are accessing a [restricted data source](http://code.google.com/intl/zh-CN/apis/visualization/documentation/dev/implementing_data_source.html#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).
  + **'scriptInjection'** - Send the query using script injection.
  + **'makeRequest'** - [*Available only for gadgets*] Send the query using the Gadget API  [makeRequest()](http://code.google.com/apis/gadgets/docs/reference/" \l "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** |
| setRefreshInterval(seconds) | None | Sets the query to automatically call the send method every specified duration (number of seconds), starting from the first explicit call to send. seconds is a number greater than or equal to zero.  If set to zero (the default), the query will not be automatically resent. This method, if used, should be called before calling the send method. |
| setTimeout(seconds) | None | Sets the number of seconds to wait for the data source to respond before raising a timeout error. seconds is a number greater than zero.  The default timeout is 30 seconds. This method, if used, should be called before calling the send method. |
| setQuery(string) | None | Sets the query string. The value of the string parameter should be a valid query.  This method, if used, should be called before calling the send method. [Learn more about the Query language](http://code.google.com/intl/zh-CN/apis/visualization/documentation/querylanguage.html) |
| send(callback) | None | Sends the query to the data source. callback should be a function that will be called when the data source responds. The callback function will receive a single parameter of type [google.visualization.QueryResponse](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "QueryResponse). |

**google.visualization.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](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "Query_send) was called.

**See also:** [Query.send](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "Query_send)

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Value** | **Description** |
| getDataTable() | [DataTable](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#DataTable) | Returns the data table as returned by the data source. Returns null if the query execution failed and no data was returned. |
| getDetailedMessage() | 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'. |
| getMessage() | 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'. |
| getReasons() | Array of strings | Returns an array of zero of more entries. Each entry is a short string with an error or warning code that was raised while executing the query. Possible codes:   * 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. |
| hasWarning() | boolean | Returns true if the query execution has any warning messages. |
| isError() | 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. |

**google.visualization.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](http://code.google.com/intl/zh-CN/apis/visualization/documentation/gallery/toolbar.html) for more information and a code example.

**google.visualization.errors**

The API provides several functions to help you display nicely-formatted 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.

|  |  |  |
| --- | --- | --- |
| **Function** | **Return Value** | **Description** |
| addError(*container*, *message*,*opt\_detailedMessage*, *opt\_options*) | 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. | Adds an error display block to the specified page element, with specified text and formatting.   * *container* - The DOM element into which to insert the error message. If the container cannot be found, the function will throw a JavaScript error. * *message* - A string message to display. * *opt\_detailedMessage* - An optional detailed message string. By default, this is mouseover text, but that can be changed in the*opt\_options*.showInToolTip property described below. * *opt\_options* - An optional object with properties that set various display options for the message. The following options are supported:   + **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 (opt\_options.type). Example: 'background-color: #33ff99; padding: 2px;' 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(*container*,*response*) | String ID value 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 error or find its containing element. | 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 [QueryResponse](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "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 toaddError(*opt\_options*.type))   * *container* - The DOM element into which to insert the error message. If the container cannot be found, the function will throw a JavaScript error. * *response* - A [QueryResponse](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "QueryResponse) object received by your query handler in response to a [query](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#Query). If this is null, the method will throw a JavaScript error. |
| removeError(*id*) | Boolean: true if the error was removed; false otherwise. | Removes the error specified by ID from the page.   * *id* - The string ID of an error created using addError() oraddErrorFromQueryResponse(). |
| removeAll(*container*) | None | Removes all error blocks from a specified container. If the specified container does not exist, this will throw an error.   * *container* - The DOM element holding the error strings to remove. If the container cannot be found, the function will throw a JavaScript error. |
| getContainer(*errorId*) | 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*errorID*.   * *errorId* - String ID of an error created using addError() oraddErrorFromQueryResponse(). |

**Events**

**Registering to Catch Events**

Your visualizations can fire and receive events, and exposes the following two methods to enable you to do so:

* [google.visualization.events.trigger()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#trigger)
* [google.visualization.events.addListener()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#addlistener)

**Commonly Exposed Events**

Visualizations can fire a number of events. Every visualization can define the details of the events it fires, but the following event should be implemented in a standard way:

* [ready event](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#readyevent)
* [select event](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#selectevent)

**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 thethis 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);

};

**addListener()**

Used by visualization *users*. Call this method to register to receive events fired by a visualization hosted on your page. Note that this will not work for gadget visualizations. The visualization 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.

**Example**

Here is an example of registering to receive the selection event

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

table.draw(data, options);

google.visualization.events.addListener(table, 'select', selectHandler);

function selectHandler() {

  alert('A table row was selected');

}

**ready Event**

If your visualization is not expected to be ready for interaction immediately after it returns from its draw() implementation, consider implementing a ready event. This event should be fired when the visualization is ready to begin responding to user interaction and method calls. This event typically does not provide any parameters to the receiver. See the [Firing Events](http://code.google.com/intl/zh-CN/apis/visualization/documentation/events.html#The_Ready_Event) page for more information.

**select Event**

If your visualization can fire events to the client when selected, consider implementing the select event. Details on the standard way to implement this event are given on the [Firing Events](http://code.google.com/intl/zh-CN/apis/visualization/documentation/dev/events.html#The_Select_Event) page.

**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.

* [Constructor](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#visconstructor)
* [draw()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#visdraw)
* [getSelection()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#visgetselection) [*optional*]
* [setSelection()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html#vissetselection) [*optional*]

**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\_div'));

**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](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable) holding the data to use to draw this graphic. There is no standard method for getting the underlying DataTable passed into a visualization.

***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 Activities'});

**getSelection() [*optional*]**

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. 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()](http://code.google.com/intl/zh-CN/apis/visualization/documentation/reference.html" \l "DataTable_getValue) method to get the value of that element.

**Example**

function myClickHandler(){

var selection = myVis.getSelection();

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);

}

**setSelection() [*optional*]**

Sets the current selection in the graphic. 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.

setSelection(*selection\_array*)

***selection\_array***

An array of objects, each with a row and/or column property, with zero-based numeric values representing the index in the underlying data table to select. Setrow=null to select a whole column, or column=null to select a whole row.