Controls and Dashboards

On this page, you'll see how to combine multiple charts into dashboards and give users controls to manipulate what data they show.

Overview

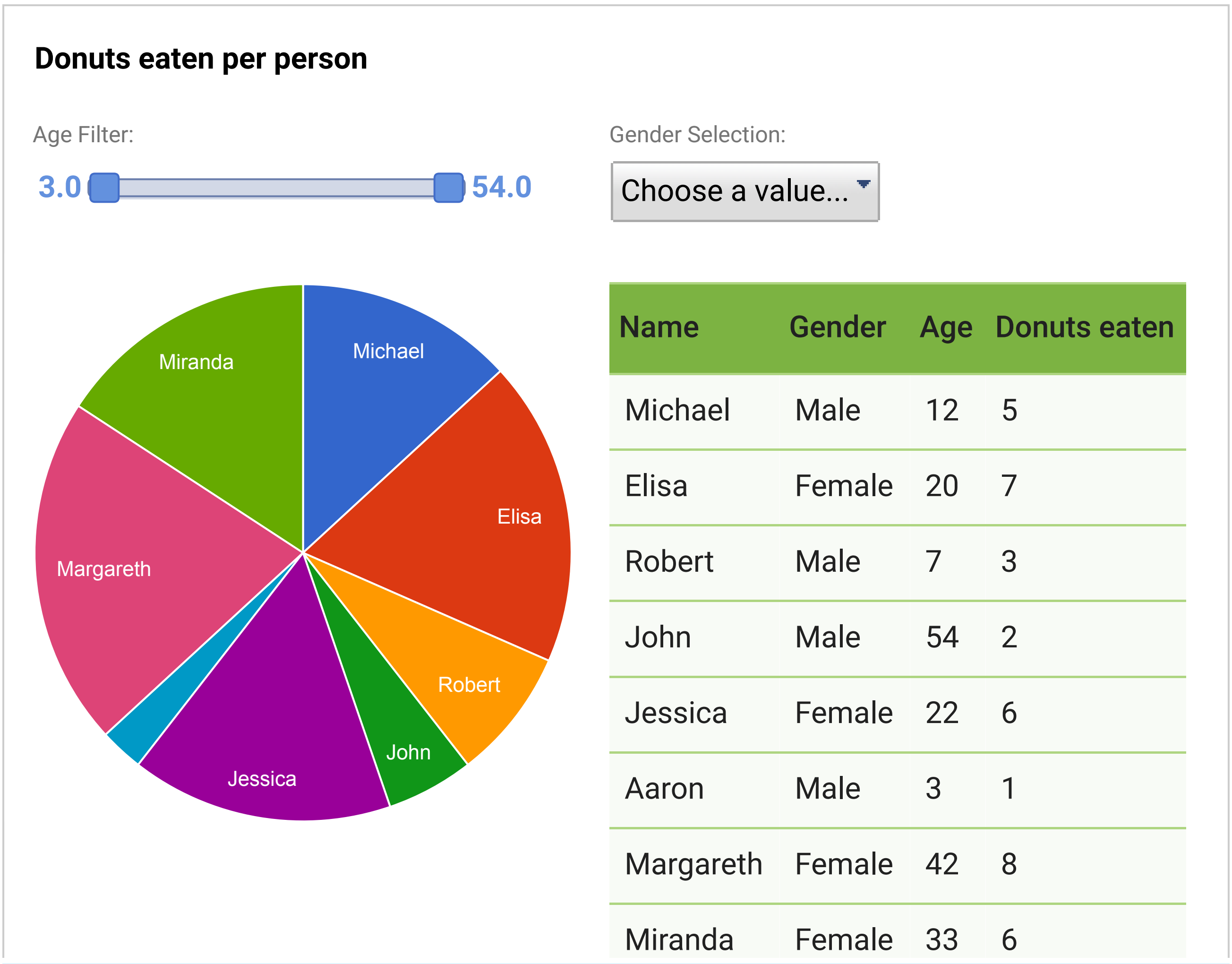
Dashboards are a simple way to organize together and manage multiple charts that share the same underlying data. By using the APIs described in this page, you can free yourself from the burden of wiring together and coordinating all the charts that are part of a dashboard.

Dashboards are defined using `google.visualization.Dashboard` (#dashboardobject) classes. `Dashboard` instances receive a `DataTable` containing the data to visualize and take care of drawing and distributing the data to all the charts that are part of the dashboard.

Controls are user interface widgets (such as category pickers, range sliders, autocompleters...) you interact with in order to drive the data managed by a dashboard and the charts that are part of it.

Controls are defined using `google.visualization.ControlWrapper` (#controlwrapperobject) classes. You can add `ControlWrapper` instances to a dashboard, where they behave like pipes and valves in a plumbing system. They collect user input and use the information to decide which of the data the dashboard is managing should be made available to the charts that are part of it.

Have a look at the following example where a category picker and a range slider are used to drive the data visualized by a pie chart.



Donuts eaten per person

Age Filter: 3.0 – 54.0

Gender Selection: Choose a value...

Name	Gender	Age	Donuts eaten
Michael	Male	12	5
Elisa	Female	20	7
Robert	Male	7	3
John	Male	54	2
Jessica	Female	22	6
Aaron	Male	3	1
Margareth	Female	42	8
Miranda	Female	33	6

Note: The dashboard is interactive. Try operating the controls and see the chart change in real time.

Using Controls and Dashboards

Here are the key steps for creating a dashboard and embedding it in your page. You'll find a code snippet demonstrating all these steps below, followed by detailed information about each step.

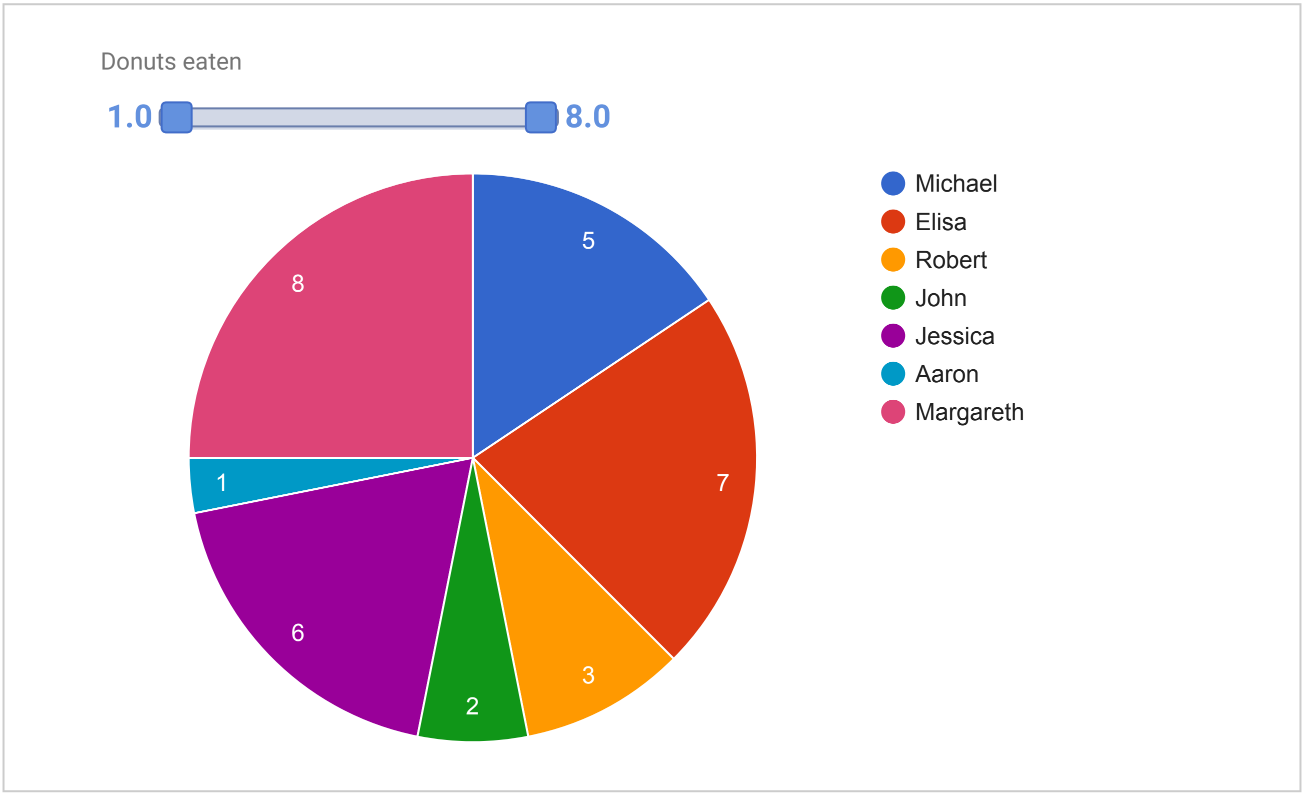
1. **Create an HTML skeleton for your dashboard** (#skeleton)**.** Your page must have as many HTML elements as needed to hold every member of a dashboard. This includes the dashboard itself and all the controls and charts that are part of it. Typically you'll use a <div> for each one.
2. **Load your libraries** (#load_your_libraries)**.** A dashboard requires only two libraries to be included or loaded on the page: the Google AJAX API and the Google Visualization `controls` package.

3. **Prepare your data** (#preparedata). You'll need to prepare the data to visualize; this means either specifying the data yourself in code, or querying a remote site for data.
4. **Create a dashboard instance** (#create_dashboard). Instantiate your dashboard by calling its constructor and passing in a reference to the <div> element that will hold it.
5. **Create as many controls and charts instances as you need** (#create_controls). Create `google.visualization.ChartWrapper` and `google.visualization.ControlWrapper` instances to describe each chart and control that the dashboard manages.
6. **Establish dependencies** (#establish_dependencies). Call `bind()` on your dashboard and pass in the control and chart instances to let the dashboard know what to manage. Once a control and chart are bound together, the dashboard updates the chart to match the constraints the control enforces over the data.
7. **Draw your dashboard** (#draw_dashboard). Call `draw()` on your dashboard and pass in your data to draw the entire dashboard on the page.
8. **Programmatic changes after draw** (#programmatic_change). Optionally, after the initial draw you can programmatically drive the controls that are part of the dashboard, and have the dashboard update the charts in response to that.

Here's a simple example of a page that creates a simple dashboard with a range slider driving a pie chart. The resulting dashboard is shown below the snippet.

```
<html>
  <head>
    <!--Load the AJAX API-->
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader
    <script type="text/javascript">

      // Load the Visualization API and the controls package.
      google.charts.load('current', {'packages':['corechart', 'controls']});

      // Set a callback to run when the Google Visualization API is loaded.
      google.charts.setOnLoadCallback(drawDashboard);

      // Callback that creates and populates a data table,
      // instantiates a dashboard, a range slider and a pie chart,
      // passes in the data and draws it.
      function drawDashboard() {

        // Create our data table.
        var data = google.visualization.arrayToDataTable([
          ['Name', 'Donuts eaten'],
          ['Michael' , 5],
          ['Elisa', 7],
          ['Robert', 3],
          ['John', 2],
          ['Jessica', 6],
          ['Aaron', 1],
          ['Margareth', 8]
        ]);

        // Create a dashboard.
        var dashboard = new google.visualization.Dashboard(
            document.getElementById('dashboard_div'));

        // Create a range slider, passing some options
        var donutRangeSlider = new google.visualization.ControlWrapper({
          'controlType': 'NumberRangeFilter',
          'containerId': 'filter_div',
          'options': {
            'filterColumnLabel': 'Donuts eaten'
          }
        });

        // Create a pie chart, passing some options
        var pieChart = new google.visualization.ChartWrapper({
          'chartType': 'PieChart',
          'containerId': 'chart_div',
          'options': {
            'width': 300,
            'height': 300,
            'pieSliceText': 'value',
            'legend': 'right'
          }
        });

        // Establish dependencies, declaring that 'filter' drives 'pieChart',
        // so that the pie chart will only display entries that are let throu
        // given the chosen slider range.
        dashboard.bind(donutRangeSlider, pieChart);

        // Draw the dashboard.
        dashboard.draw(data);
      }
    </script>
  </head>

  <body>
    <!--Div that will hold the dashboard-->
    <div id="dashboard_div">
      <!--Divs that will hold each control and chart-->
      <div id="filter_div"></div>
      <div id="chart_div"></div>
    </div>
  </body>
</html>
```

Here's the dashboard that this code creates.



1. Create An HTML Skeleton For Your Dashboard

Your page must have as many HTML elements (typically <div>s) to hold both the dashboard itself and all the controls and charts part of it. When instantiating dashboard, control, and chart instances, you must pass a reference to their element, so assign an ID to each HTML element.

```
    <!--Div that will hold the dashboard-->
    <div id="dashboard_div">
      <!--Divs that will hold each control and chart-->
      <div id="filter_div"></div>
      <div id="chart_div"></div>
    </div>
```

You are free to position each HTML element however you want your dashboard to look.

2. Load Your Libraries

A dashboard requires only two libraries to be included or loaded on the page: the Google AJAX API and the Google Visualization `controls` package. The API (in particular, `google.visualization.ChartWrapper`) automatically identifies the other packages needed (for example, `gauge` if you are using a Gauge chart) and loads them on the fly without further intervention from you.

You must use `google.charts.load()` to fetch the control library.

```
<!--Load the AJAX API-->
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"
<script type="text/javascript">

  // Load the Visualization API and the controls package.
  // Packages for all the other charts you need will be loaded
  // automatically by the system.
  google.charts.load('current', {'packages':['corechart', 'controls']});

  // Set a callback to run when the Google Visualization API is loaded.
  google.charts.setOnLoadCallback(drawDashboard);

  function drawDashboard() {
    // Everything is loaded. Assemble your dashboard...
  }
</script>
```

3. Prepare Your Data

When the Visualization API has been loaded, `google.charts.setOnLoadCallback()` will call your handler function, so you know that all the required helper methods and classes will be ready for you to start preparing your data.

Dashboards accepts data in a DataTable, the same as charts.

4. Create A Dashboard Instance

After you have created your data, you can instantiate your dashboard object. A dashboard constructor takes one parameter: a reference to the DOM element in which to draw the dashboard.

```
  var dashboard = new google.visualization.Dashboard(document.getElementById(
```

Dashboards are exposed as a Javascript class. After instantiating your dashboard, you can perform a few optional steps such as adding event listeners (for example, to be notified once the dashboard is 'ready'). Dashboards handle events in the same way charts do, so refer to Handling Visualization Events (https://google-developers.appspot.com/chart/interactive/docs/events) or Displaying Errors Nicely (https://google-developers.appspot.com/chart/interactive/docs/reference#errordisplay) in the chart section for more information.

5. Create Control And Chart Instances

Define as many instances you need for each control and chart that will be part of the dashboard. Use `google.visualization.ChartWrapper` (https://google-developers.appspot.com/chart/interactive/docs/reference#chartwrapperobject) and `google.visualization.ControlWrapper` (#controlwrapperobject) to define charts and controls respectively.

```
// Create a range slider, passing some options
var donutRangeSlider = new google.visualization.ControlWrapper({
  'controlType': 'NumberRangeFilter',
  'containerId': 'filter_div',
  'options': {
    'filterColumnLabel': 'Donuts eaten'
  }
});

// Create a pie chart, passing some options
var pieChart = new google.visualization.ChartWrapper({
  'chartType': 'PieChart',
  'containerId': 'chart_div',
  'options': {
    'width': 300,
    'height': 300,
    'pieSliceText': 'label'
  }
});
```

When creating `ChartWrapper` and `ControlWrapper` instances, do not specify either the `dataTable` or the `dataSourceUrl` parameter. The dashboard takes care of feeding each one with the appropriate data. However, be sure to specify the required parameters: `chartType` and `containerId` for charts, `controlType` and `containerId` for controls.

A few tips about configuring controls and charts:

- You must give all controls a `filterColumnIndex` (or `filterColumnLabel`) to specify which column of your `google.visualization.DataTable` the control operates on (in the example above, the control operates on the column labeled *Donuts eaten*),
- Use the `state` configuration option on controls to initialize them with an explicit state when they are first drawn. For example, use this setting to fix the initial positions of the thumbs of a range slider control.

  ```
  var donutRangeSlider = new google.visualization.ControlWrapper({
    'controlType': 'NumberRangeFilter',
    'containerId': 'filter_div',
    'options': {
      'filterColumnLabel': 'Donuts eaten',
      'minValue': 1,
      'maxValue': 10
    },
    // Explicitly positions the thumbs at position 3 and 8,
    // out of the possible range of 1 to 10.
    'state': {'lowValue': 3, 'highValue': 8}
  });
  ```

- All the charts that are part of a dashboard share the same underlying dataTable you prepared in the *Prepare Your Data* (#preparedata) step. However, charts often require a specific arrangement of columns to display correctly: for example, a pie chart requires a string column for the label, followed by a number column for the value.

  Use the `view` option while configuring each `ChartWrapper` instance to declare which columns are relevant for the chart, as in the following example.

  ```
  var data = google.visualization.arrayToDataTable([
    ['Name', 'Gender', 'Age', 'Donuts eaten'],
    ['Michael' , 'Male', 12, 5],
    ['Elisa', 'Female', 20, 7],
    ['Robert', 'Male', 7, 3],
    ['John', 'Male', 54, 2],
    ['Jessica', 'Female', 22, 6],
    ['Aaron', 'Male', 3, 1],
    ['Margareth', 'Female', 42, 8]
  ]);

  var pieChart = new google.visualization.ChartWrapper({
    'chartType': 'PieChart',
    'containerId': 'chart_div',
    'options': {
      'width': 300,
  ```

```
      'height': 300,
      'title': 'Donuts eaten per person'
    },
    // The pie chart will use the columns 'Name' and 'Donuts eaten'
    // out of all the available ones.
    'view': {'columns': [0, 3]}
  });

  // The rest of dashboard configuration follows
  // ...
```

6. Establish Dependencies

Once you have instantiated both the dashboard and all the controls and charts that will be part it, use the `bind()` method to tell the dashboard about the dependencies that exist between controls and charts.

Once a control and chart are bound together, the dashboard updates the chart to match the constraints the control enforces over the data. In the example dashboard you are building, the range slider and the pie chart are bound together, so whenever you interact with the former, the latter updates to display only the data that matches the selected range.

```
// 'pieChart' will update whenever you interact with 'donutRangeSlider'
// to match the selected range.
dashboard.bind(donutRangeSlider, pieChart);
```

You can bind controls and charts in many different configurations: one-to-one, one-to-many, many-to-one and many-to-many. Whenever multiple controls are bound to a chart, the dashboard updates the chart to match the combined constraints enforced by all the bound controls. At the same time, a control can drive multiple charts concurrently. To specify multiple bindings at the same time, pass in arrays to the `bind()` method instead of single instances. You can also chain multiple `bind()` calls together. Here are some examples.

```
// Many-to-one binding where 'ageSelector' and 'salarySelector' concurrentl
// participate in selecting which data 'ageVsSalaryScatterPlot' visualizes.
dashboard.bind([agePicker, salaryPicker], ageVsSalaryScatterPlot);

// One-to-many binding where 'ageSelector' drives two charts.
dashboard.bind(agePicker, [ageVsSalaryScatterPlot, ageBarChart]);

// bind() chaining where each control drives its own chart.
dashboard.bind(agePicker, ageBarChart).bind(salaryRangePicker, salaryPieCha
```

For advanced usages, you can also bind controls to other controls to establish chains of dependencies .

```
dashboard.bind(countryPicker, regionPicker).bind(regionPicker, cityPicker);
```

7. Draw Your Dashboard

Call the `draw()` method on the dashboard instance to render the entire dashboard. The `draw()` method takes only one parameter: the `DataTable` (or `DataView`) that powers the dashboard.

You should call `draw()` every time you change the composition of the dashboard (for example by adding new controls or charts to it) or you change the overall `DataTable` that powers it.

The dashboard instance fires a `ready` event whenever `draw()` terminates drawing all the controls and charts that are part of it. An `error` event is fired if any of the managed controls or chart fails to draw. To learn more about handling events, see Handling Events (https://google-developers.appspot.com/chart/interactive/docs/events).

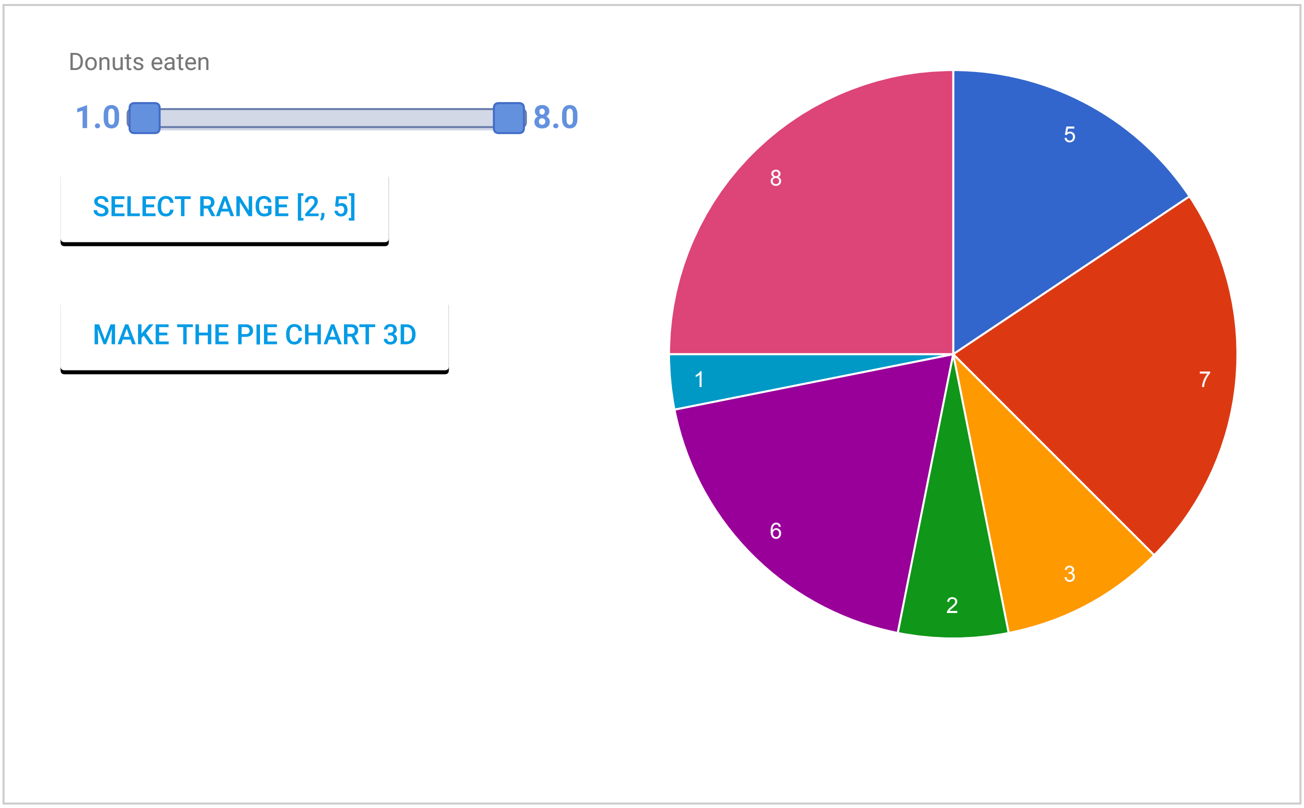
Note: You should listen for the `ready` event before you try to change the dashboard composition or draw it again.

8. Programmatic Changes After Draw

Once the dashboard completes the initial `draw()` it will be *live* and respond automatically to any action you perform on it (such as changing the selected range of a control slider that is part of the dashboard).

If you need to programmatically alter the dashboard state, you can do so by operating directly on the `ControlWrapper` and `ChartWrapper` instances that are part of it. The rule of thumb is to perform any change you need directly on the specific `ControlWrapper` (or `ChartWrapper`) instance: for example, change a control option or state via `setOption()` and `setState()` respectively, and call its `draw()` method afterward. The dashboard will then update to match the requested changes.

The following example shows such a case.



CODE IT YOURSELF ON JSFIDDLE

FULL WEB PAGE

```
<html>
  <head>
    <script type="text/javascript" src="https://www.gstatic.com/charts/loade
    <script type="text/javascript">
      google.charts.load('current', {'packages':['corechart', 'controls']});
      google.charts.setOnLoadCallback(drawStuff);

      function drawStuff() {

        var dashboard = new google.visualization.Dashboard(
          document.getElementById('programmatic_dashboard_div'));

        // We omit "var" so that programmaticSlider is visible to changeRang
        var programmaticSlider = new google.visualization.ControlWrapper({
          'controlType': 'NumberRangeFilter',
          'containerId': 'programmatic_control_div',
          'options': {
            'filterColumnLabel': 'Donuts eaten',
            'ui': {'labelStacking': 'vertical'}
          }
        });

        var programmaticChart  = new google.visualization.ChartWrapper({
          'chartType': 'PieChart',
          'containerId': 'programmatic_chart_div',
          'options': {
            'width': 300,
            'height': 300,
            'legend': 'none',
            'chartArea': {'left': 15, 'top': 15, 'right': 0, 'bottom': 0},
            'pieSliceText': 'value'
          }
        });

        var data = google.visualization.arrayToDataTable([
          ['Name', 'Donuts eaten'],
          ['Michael' , 5],
          ['Elisa', 7],
          ['Robert', 3],
          ['John', 2],
          ['Jessica', 6],
          ['Aaron', 1],
          ['Margareth', 8]
        ]);

        dashboard.bind(programmaticSlider, programmaticChart);
        dashboard.draw(data);

        changeRange = function() {
          programmaticSlider.setState({'lowValue': 2, 'highValue': 5});
          programmaticSlider.draw();
        };

        changeOptions = function() {
          programmaticChart.setOption('is3D', true);
          programmaticChart.draw();
        };
      }

    </script>
  </head>
  <body>
    <div id="programmatic_dashboard_div" style="border: 1px solid #ccc">
      <table class="columns">
        <tr>
          <td>
            <div id="programmatic_control_div" style="padding-left: 2em; min
            <div>
              <button style="margin: 1em 1em 1em 2em" onclick="changeRange()
                Select range [2, 5]
              </button><br />
              <button style="margin: 1em 1em 1em 2em" onclick="changeOptions
                Make the pie chart 3D
              </button>
            </div>
            <script type="text/javascript">
              function changeRange() {
                programmaticSlider.setState({'lowValue': 2, 'highValue': 5})
                programmaticSlider.draw();
              }

              function changeOptions() {
                programmaticChart.setOption('is3D', true);
                programmaticChart.draw();
              }
            </script>
          </td>
          <td>
            <div id="programmatic_chart_div"></div>
          </td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

API Reference

This section lists the objects exposed by the Controls and Dashboards API, and the standard methods exposed by all controls.

Dashboard

Represents a collection of collaborating controls and charts that share the same underlying data.

Constructor

```
Dashboard(containerRef)
```

containerRef

A reference to a valid container element on the page that will hold the dashboard contents.

Methods

Dashboard exposes the following methods:

Method	Return Type	Description
`bind(`*`controls`*`, `*`charts`*`)`	google.visualization.Dashboard	Binds one or more Controls to one or more other dashl controls), so that all of the latter are redrawn wheneve user interaction that affects the data managed by the itself for chaining multiple **`bind()`** calls together. • *controls* - Either a single one or an array of **`google`** instances defining the controls to bind. • *charts* - Either a single one or an array of **`google.v`** defining the charts the that will be driven the by the
`draw(`*`dataTable`*`)`	None	Draws the dashboard. • *dataTable* - Any one of the following: a **`DataTable`** representation of a DataTable; or an array following google.visualization.arrayToDataTable() (https://google-developers.appspot.com/chart/interactive/docs/re .
`getSelection()`	Array of objects	Returns an array of the selected visual entities of the c **`getSelection()`** method, when called on the dashb selections made on all of the charts within it, allowing with chart selections. **Note:** Event listeners for the select event (https://google-developers.appspot.com/chart/intera need to be attached to each chart to which you wish to Extended description (https://google-developers.appspot.com/chart/intera

Events

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

Name	Description	Properties
`error`	Fired when an error occurs when attempting to render the dashboard. One or more of the controls and charts that are part of the dashboard may have failed rendering.	id, message
`ready`	The dashboard has completed drawing and is ready to accept changes. All the controls and charts that are part of the dashboard are ready for external method call and user interaction. If you want to change the dashboard (or the data it displays) after you draw it, you should set up a listener for this event *before* you call the `draw` method, and then apply your changes only after the event was fired. The `ready` event will also fire: • after the completion of a dashboard refresh triggered by a user or programmatic interaction with one of the controls, • after a programmatic call to the `draw()` method of any chart part of the dashboard.	None

ControlWrapper

A ControlWrapper object is a wrapper around a JSON representation of a configured control instance. The class exposes convenience methods for defining a dashboard control, drawing it and programmatically changing its state.

Constructor

```
ControlWrapper(opt_spec)
```

opt_spec

[*Optional*] - Either a JSON object defining the control, or a serialized string version of that object. The supported properties of the JSON object are shown in the following table. If not specified, you must set all the appropriate properties using the *set...* methods exposed by ControlWrapper.

Property	Type	Required	Default	Description
controlType	String	Required	none	The class name of the control. The `google.visualization` package name can be omitted for Google controls. **Examples:** `CategoryFilter`, `NumberRangeFilter`.
containerId	String	Required	none	The ID of the DOM element on your page that will host the control.
options	Object	Optional	none	An object describing the options for the control. You can use either

				JavaScript literal notation, or provide a handle to the object. **Example**: `"options": {"filterColumnLabel": "Age", "minValue": 10, "maxValue": 80}`
state	Object	Optional	none	An object describing the state of the control. The state collects all the variables that the user operating the control can affect. For example, a range slider state can be described in term of the positions that the low and high thumb of the slider occupy. You can use either Javascript literal notation, or provide a handle to the object.**Example**: `"state": {"lowValue": 20, "highValue": 50}`

Methods

ControlWrapper exposes the following additional methods:

Method	Return Type	Description
`draw()`	None	Draws the control. Normally the dashboard holding the control takes care of drawing it. You should call `draw()` to force programmatic redraws of the control after you change any of its other settings, like options or state.
`toJSON()`	String	Returns a string version of the JSON representation of the control.
`clone()`	ControlWrapper (#controlwrapperobject)	Returns a deep copy of the control wrapper.
`getControlType()`	String	The class name of the control. If this is a Google control, the name will not be qualified with `google.visualization`. So, for example, if this were a CategoryFilter control, it would return "CategoryFilter" rather than "google.visualization.CategoryFilter".
`getControlName()`	String	Returns the control name assigned by `setControlName()`.
`getControl()`	Control object reference	Returns a reference to the control created by this ControlWrapper. This will return null until after you have called `draw()` on the ControlWrapper object (or on the dashboard holding it), and it throws a ready event. The returned object only exposes one method: `resetControl()`, which resets the control state to the one it was initialized with (like resetting an HTML form element).
`getContainerId()`	String	The ID of the control's DOM container element.

`getOption(`*`key`*`, `*`opt_default_val`*`)`	Any type	Returns the specified control option value • *key* - The name of the option to retrieve. May be a qualified name, such as `'vAxis.title'`. • *opt_default_value* [*Optional*] - If the specified value is undefined or null, this value will be returned.
`getOptions()`	Object	Returns the options object for this control.
`getState()`	Object	Returns the control state.
`setControlType(`*`type`*`)`	None	Sets the control type. Pass in the class name of the control to instantiate. If this is a Google control, do not qualify it with `google.visualization`. So, for example, for a range slider over a numeric column, pass in "NumberRangeFilter".
`setControlName(`*`name`*`)`	None	Sets an arbitrary name for the control. This is not shown anywhere on the control, but is for your reference only.
`setContainerId(`*`id`*`)`	None	Sets the ID of the containing DOM element for the control.
`setOption(`*`key`*`, `*`value`*`)`	None	Sets a single control option value, where *key* is the option name and *value* is the value. To unset an option, pass in null for the value. Note that *key* may be a qualified name, such as `'vAxis.title'`.
`setOptions(`*`options_obj`*`)`	None	Sets a complete options object for a control.
`setState( state_obj)`	None	Sets the control state. The state collects all the variables that the user operating the control can affect. For example, a range slider state can be described in term of the positions that the low and high thumb of the slider occupy.

Events

The ControlWrapper object throws the following events. Note that you must call `ControlWrapper.draw()` (or draw the dashboard holding the control) before any events will be thrown.

Name	Description	Properties
`error`	Fired when an error occurs when attempting to render the control.	id, message

`ready`	The control is ready to accept user interaction and for external method calls. If you want to interact with the control, and call methods after you draw it, you should set up a listener for this event *before* you call the `draw` method, and call them only after the event was fired. Alternatively, you can listen for a `ready` event on the dashboard holding the control and call control methods only after the event was fired.	None
`statechange`	Fired when the user interacts with the control, affecting its state. For example, a `statechange` event will fire whenever you move the thumbs of a range slider control. To retrieve an updated control state after the event fired, call `ControlWrapper.getState()`.	None

(#chartwrapperobject)ChartWrapper

Refer to `google.visualization.ChartWrapper` (https://google-developers.appspot.com/chart/interactive/docs/reference#chartwrapperobject) documentation in the visualizations' API reference section.

The following notes apply when using a `ChartWrapper` as part of a dashboard:

- Do not set the `dataTable`, `query`, `dataSourceUrl` and `refreshInterval` attributes explicitly. The dashboard holding the chart takes care of feeding it the data it needs.
- Do set its `view` attribute to declare which columns, out of all the ones present in the `dataTable` given to the dashboard, are relevant for the chart, as shown in *Create Control and Chart Instances* (#create_controls).

Controls Gallery

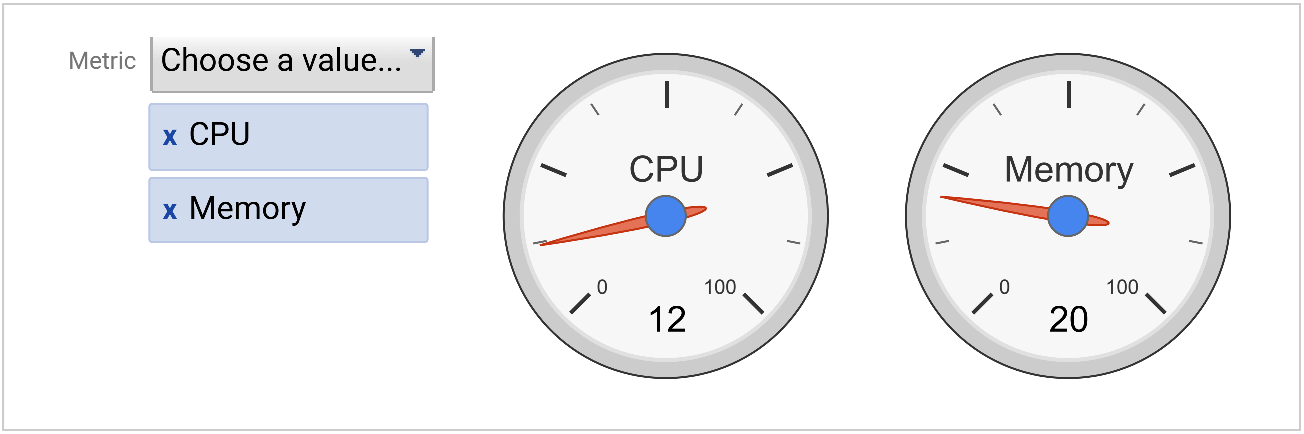
Filters are graphical elements that people can use to interactively select which data is displayed on your chart. This section describes the Google Chart filters: *CategoryFilter*, *ChartRangeFilter*, *DateRangeFilter*, *NumberRangeFilter*, and *StringFilter*.

You can use any of them as a parameter to `ControlWrapper.setControlType()`.

Note: In describing options, the dot notation is used to describe nested object attributes. For example an option named `'ui.label'` should be declared in an options object as `var options = {"ui": {"label": "someLabelValue"} };`

CategoryFilter

A picker to choose one or more between a set of defined values.



State

Name	Type	Default	Description
selectedValues	Array of objects or primitive types	none	The set of values currently selected. The selected values must be a set of the overall selectable values defined by the `values` option (any extraneous one will be ignored). If the `CategoryFilter` does not allow multiple choice, only the first value of the array is retained.

Options

Name	Type	Default	Description
filterColumnIndex	number	none	The column of the datatable the filter should operate upon. It is mandatory to provide either this option or `filterColumnLabel`. If both present, this option takes precedence.
filterColumnLabel	string	none	The label of the column the filter should operate upon. It is mandatory to provide either this option or `filterColumnIndex`. If both present,

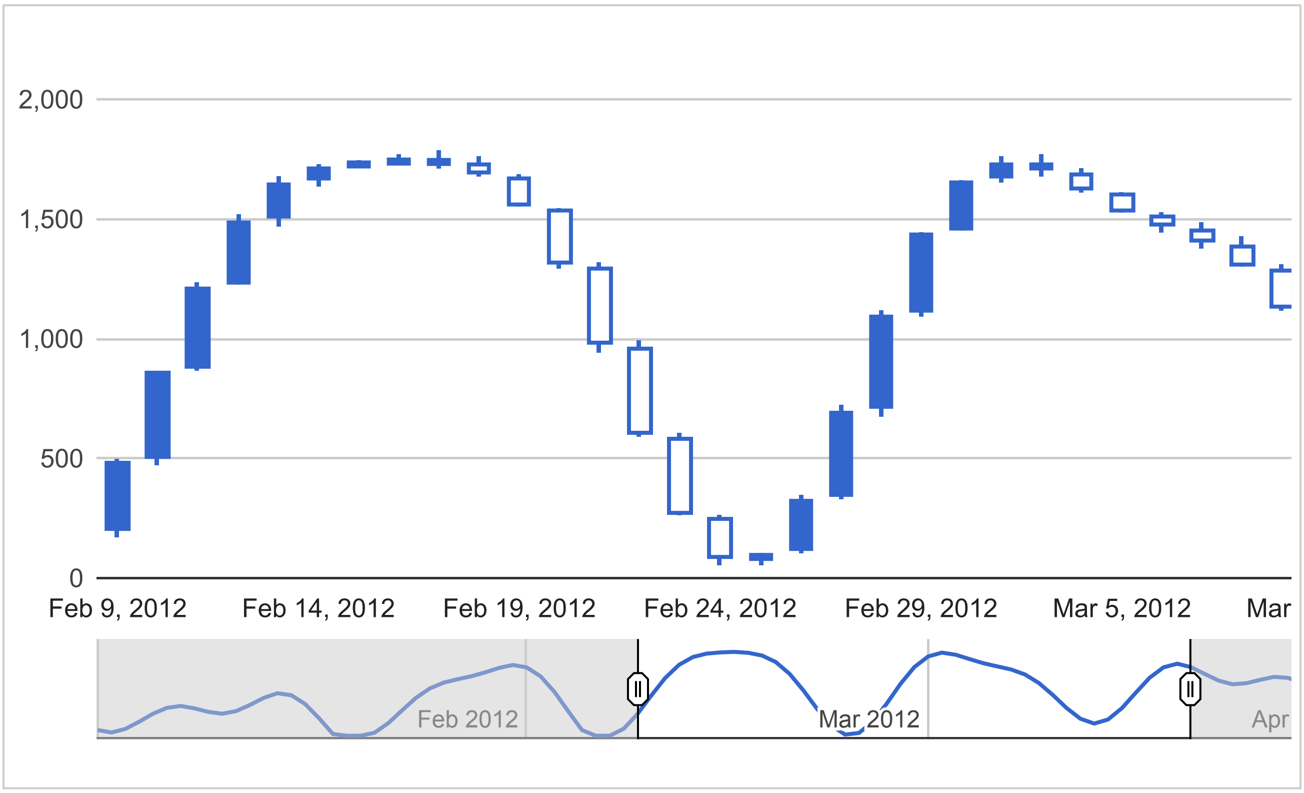
			`filterColumnIndex` takes precedence.
values	Array	auto	List of values to choose from. If an array of Objects is used, they should have a suitable **`toString()`** representation for display to the user. If null or undefined, the list of values will be automatically computed from the values present in the DataTable column this control operates on.
useFormattedValue	boolean	false	When populating the list of selectable values automatically from the DataTable column this filter operates on, whether to use the actual cell values or their formatted values.
ui	Object	null	An object with members to configure various aspects of the control's UI. To specify properties of this object, you can use object literal notation, as shown here: `{label: 'Metric', la`
ui.caption	string	'Choose a value...'	The caption to display inside the value picker widget when no item is selected.
ui.sortValues	boolean	true	Whether the values to choose from should be sorted.
ui.selectedValuesLayout	string	'aside'	How to display selected values, when multiple selection is allowed. Possible values are:

			<ul><li>`'aside'`: selected values will display in a single text line next to the value picker widget,</li><li>`'below'`: selected values will display in a single text line below the widget,</li><li>`'belowWrapping'`: similar to `below`, but entries that cannot fit in the picker will wrap to a new line,</li><li>`'belowStacked'`: selected values will be displayed in a column below the widget.</li></ul>
ui.allowNone	boolean	true	Whether the user is allowed not to choose any value. If `false` the user must choose at least one value from the available ones. During control initialization, if the option is set to `false` and no `selectedValues` state is given, the first value from the avaiable ones is automatically seleted.
ui.allowMultiple	boolean	true	Whether multiple values can be selected, rather than just one.
ui.allowTyping	boolean	true	Whether the user is allowed to type in a text field to narrow down the list of possible choices (via an autocompleter), or not.
ui.label	string	auto	The label to display next to the category picker. If unspecified, the label of the column the control operates on will be used.

ui.labelSeparator	string	none	A separator string appended to the label, to visually separate the label from the category picker.
ui.labelStacking	string	'horizontal'	Whether the label should display above (vertical stacking) or beside (horizontal stacking) the category picker. Use either `'vertical'` or `'horizontal'`.
ui.cssClass	string	'google-visualization-controls-categoryfilter'	The CSS class to assign to the control, for custom styling.

ChartRangeFilter

A slider with two thumbs superimposed onto a chart, to select a range of values from the continuous axis (https://google-developers.appspot.com/chart/interactive/docs/customizing_axes#Discrete_vs_Continuous) of the chart.



State

Name	Type	Default	Description
range.start	number, date, datetime or timeofday	Range start value	The start of the selected range, inclusive.
range.end	number, date, datetime or timeofday	Range end value	The end of the selected range, inclusive.

Options

Name	Type	Default	Description
filterColumnIndex	number	none	The index of the column in the data table the filter operates on. It is mandatory to provide either this option or **`filterColumnLabel`**. If both are present, this option takes precedence. Note that it only makes sense to specify an index of a domain (https://google-developers.appspot.com/chart/interactive/docs/roles#whatrolesavailable) column that is embodied in the continuous axis of the chart drawn inside the control.
filterColumnLabel	string	none	The label of the column of the data table the filter operates on. It is mandatory to provide either this option or **`filterColumnIndex`**. If both are present, **`filterColumnIndex`** takes precedence.

			Note that it only makes sense to specify an label of a domain (https://google-developers.appspot.com/chart/interactive/docs/roles#whatrolesavailable) column that is embodied in the continuous axis of the chart drawn inside the control.
ui	Object	null	An object with members to configure various aspects of the control's UI. To specify properties of this object, you can use object literal notation, as shown here: `{chartType: 'Scatte`
ui.chartType	string	'ComboChart'	The type of the chart drawn inside the control. Can be one of: 'AreaChart', 'LineChart', 'ComboChart' or 'ScatterChart'.

ui.chartOptions	Object	`{`
		` 'enableInteractivi`
		` 'chartArea': {'hei`
		` 'legend': {'positi`
		` 'hAxis': {'textPos`
		` 'vAxis': {`
		`  'textPosition': '`
		`  'gridlines': {'co`
		` }`
		`}`
			The configuration options of the chart drawn inside the control. Allows the same level of configuration as any chart in the dashboard, and complies with the same format as accepted by ChartWrapper.setOptions() (https://google-developers.appspot.com/chart/interactive/docs/reference#chartwrapperobject) .
			
			You can specify additional options or override the default ones (note that the defaults have been carefully chosen for

			optimal appearance, though).
ui.chartView	Object or string (serialized Object)	null	Specification of the view to apply to the data table used to draw the chart inside the control. Allows the same level of configuration as any chart in the dashboard, and complies with the same format as accepted by ChartWrapper.setView() (https://google-developers.appspot.com/chart/interactive/docs/reference#chartwrapperobject) . If not specified, the data table itself is used to draw the chart. Please note that the horizontal axis of the drawn chart must be continuous (https://google-developers.appspot.com/chart/interactive/docs/customizing_axes#Discrete_vs_Continuous) , so be careful to specify **`ui.chartView`** accordingly.
ui.minRangeSize	number	Data value difference interpreted as 1 pixel	The minimum selectable range size (**`range.end - range.start`**), specified in data value units. For a numeric axis, it is a number (not necessarily an integer). For a date, datetime or timeofday axis, it is an integer that specifies the difference in milliseconds.
ui.snapToData	boolean	false	If true, range thumbs are

		snapped to the nearest data points. In this case, the end points of the range returned by **`getState()`** are necessarily values in the data table.

Events

Additions to ControlWrapper (#controlwrapperobject) events:

Name	Description	Properties
`statechange`	Same as documented for every ControlWrapper, only has an extra boolean property, **`inProgress`**, that specifies whether the state is currently being changed (either one of the thumbs or the range itself is being dragged).	inProgress

DateRangeFilter

A dual-valued slider for selecting ranges of dates.

Try moving the slider to change which rows are shown in the table below.

Extrasolar planet	Comment	Year
amma Cephei Ab	Deduced from radial velocity variations of the star Gamma Cephei	Jul 13, 1988
D 114762 b	At least 11 times the mass of Jupiter	May 4, 198
SR B1257+12	First confirmed discovery of an extrasolar planet	Jan 22, 199
1 Pegasi b	Hot Jupiter with a 4.2 day orbit	Oct 6, 1995
7 Ursae Majoris b	First long-period planet discovered	Jan 17, 199
psilon Andromedae	First multiple planetary system around a main sequence star	Aug 12, 199
liese 876 b	First planet found orbiting a red dwarf	Jun 23, 199
D 209458 b	First exoplanet seen transiting its parent star	Nov 5, 1999
ta Draconis b	Provided evidence that planets can exist around giant stars	Jan 8, 2002
SR B1620-26 b	12.7 billion year old planet orbiting a binary star system	Jul 10, 2003
M1207 b	First planet found orbiting a brown dwarf	Jul 22, 2004
u Arae c	Hot Neptune	Aug 25, 200
rES-1 and HD 209458 b	First detection of light from exoplanets	Mar 22, 200
GLE-2005-BLG-390Lb	Detected used gravitational microlensing	Feb 25, 200
liese 581 c	Inhospitable due to runaway greenhouse effect	Apr 4, 2007
omalhaut b	First exoplanet directly imaged by optical telescope	Nov 13, 200
J 1214 b	Might be 75% water and 25% rock	Dec 16, 200
D 10180	Seven planets orbiting a sun-like star	Aug 24, 201
5 Cancri e	Orbital period of just 0.73 days	Apr 27, 2011
lpha Centauri Bb	Earth-mass planet in the star system closest to ours	Oct 16, 2012
H2 b	Potentially habitable Jupiter-sized planet	Jan 13, 201
epler-69c	First potentially habitable Earth-sized planet orbiting a sun-sized star	Apr 18, 201

State

Name	Type	Default	Description
lowValue	number	none	The lower extent of the selected range, inclusive.
highValue	number	none	The higher extent of the selected range, inclusive.
lowThumbAtMinimum	boolean	none	Whether the lower thumb of the slider is locked at the minimum allowed range. If set, overrides `lowValue`.
highThumbAtMaximum	boolean	none	Whether the higher thumb of the slider is locked at the maximum allowed range. If set, overrides `highValue`.

Options

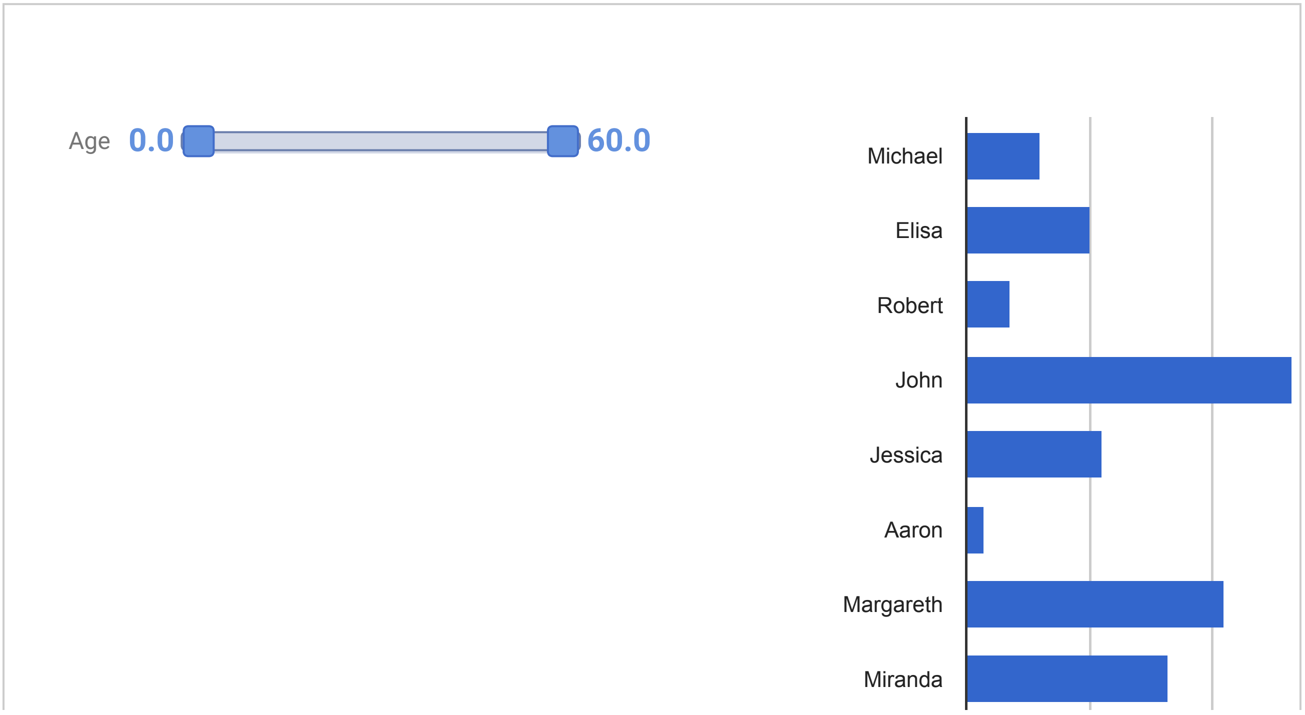
Name	Type	Default	Description
filterColumnIndex	number	none	The column of the datatable the filter should operate upon. It is mandatory to provide either this option or **`filterColumnLabel`**. If both present, this option takes precedence. Must point to a column with type **`number`**.
filterColumnLabel	string	none	The label of the column the filter should operate upon. It is mandatory to provide either this option or **`filterColumnIndex`**. If both present, **`filterColumnIndex`** takes precedence. Must point to a column with type **`number`**.
minValue	Date	auto	Minimum allowed value for the range lower extent. If undefined, the value will be inferred from the contents of the DataTable managed by the control.
maxValue	Date	auto	Maximum allowed value for the range higher extent. If undefined, the value will be inferred from the contents of the DataTable managed by the control.
ui	Object	null	An object with members to configure various aspects of the control's UI. To specify properties of this object, you can use object literal notation, as shown here: `{label: 'Age', label`
ui.format	Object	none	How to represent the date

			as a string. Accepts any valid date format (https://developers.google.com/chart/interactive/docs/reference?hl=en#dateformatter) .
ui.step	string	day	The minimum possible change when dragging the slider thumbs: can be any time unit up to "day". ("month" and "year" aren't yet supported.)
ui.ticks	number	auto	The number of ticks (fixed positions in the range bar) the slider thumbs can occupy.
ui.unitIncrement	string	'1'	The amount to increment for unit increments of the range extents. A unit increment is equivalent to using the arrow keys to move a slider thumb.
ui.blockIncrement	number	10	The amount to increment for block increments of the range extents. A block increment is equivalent to using the pgUp and pgDown keys to move the slider thumbs.
ui.showRangeValues	boolean	true	Whether to have labels next to the slider displaying extents of the selected range.
ui.orientation	string	'horizontal'	The slider orientation. Either `'horizontal'` or `'vertical'`.
ui.label	string	auto	The label to display next to the slider. If unspecified, the label of the column the control operates on will be used.

ui.labelSeparator	string	none	A separator string appended to the label, to visually separate the label from the slider.
ui.labelStacking	string	'horizontal'	Whether the label should display above (vertical stacking) or beside (horizontal stacking) the slider. Use either `'vertical'` or `'horizontal'`.
ui.cssClass	string	'google-visualization-controls-rangefilter'	The CSS class to assign to the control, for custom styling.

NumberRangeFilter

A dual-valued slider for selecting ranges of numeric values.



State

Name	Type	Default	Description
lowValue	number	none	The lower extent of the selected range, inclusive.
highValue	number	none	The higher extent of the selected range, inclusive.

lowThumbAtMinimum	boolean	none	Whether the lower thumb of the slider is locked at the minimum allowed range. If set, overrides `lowValue`.
highThumbAtMaximum	boolean	none	Whether the higher thumb of the slider is locked at the maximum allowed range. If set, overrides `highValue`.

Options

Name	Type	Default	Description
filterColumnIndex	number	none	The column of the datatable the filter should operate upon. It is mandatory to provide either this option or `filterColumnLabel`. If both present, this option takes precedence. Must point to a column with type `number`.
filterColumnLabel	string	none	The label of the column the filter should operate upon. It is mandatory to provide either this option or `filterColumnIndex`. If both present, `filterColumnIndex` takes precedence. Must point to a column with type `number`.
minValue	number	auto	Minimum allowed value for the range lower extent. If undefined, the value will be inferred from the contents of the DataTable managed by the control.
maxValue	number	auto	Maximum allowed value for the range higher extent. If undefined, the value will be inferred from the contents of the DataTable managed by the control.

ui	Object	null	An object with members to configure various aspects of the control's UI. To specify properties of this object, you can use object literal notation, as shown here: `{label: 'Age', label`
ui.format	Object	none	How to represent the number as a string. Accepts any valid number format (https://developers.google.com/chart/interactive/docs/reference?hl=en#numberformatter).
ui.step	number	1, or computed from `ticks` if defined	The minimum possible change when dragging the slider thumbs.
ui.ticks	number	auto	The number of ticks (fixed positions in the range bar) the slider thumbs can occupy.
ui.unitIncrement	number	1	The amount to increment for unit increments of the range extents. A unit increment is equivalent to using the arrow keys to move a slider thumb.
ui.blockIncrement	number	10	The amount to increment for block increments of the range extents. A block increment is equivalent to using the pgUp and pgDown keys to move the slider thumbs.
ui.showRangeValues	boolean	true	Whether to have labels next to the slider displaying extents of the selected range.
ui.orientation	string	'horizontal'	The slider orientation.

			Either `'horizontal'` or `'vertical'`.
ui.label	string	auto	The label to display next to the slider. If unspecified, the label of the column the control operates on will be used.
ui.labelSeparator	string	none	A separator string appended to the label, to visually separate the label from the slider.
ui.labelStacking	string	'horizontal'	Whether the label should display above (vertical stacking) or beside (horizontal stacking) the slider. Use either `'vertical'` or `'horizontal'`.
ui.cssClass	string	'google-visualization-controls-rangefilter'	The CSS class to assign to the control, for custom styling.

StringFilter

A simple text input field that lets you filter data via string matching. It updates after every keypress: try typing `j` to narrow the table below to John and Jessica.

Name

Name	Age
ichael	1
lisa	2
obert	
ohn	5
essica	2
aron	
argareth	4
iranda	3

State

Name	Type	Default	Description
value	string or object	none	The text currently entered in the control input field.

Options

Name	Type	Default	Description
filterColumnIndex	number	none	The column of the datatable the filter should operate upon. It is mandatory to provide either this option or **`filterColumnLabel`**. If both present, this option takes precedence.
filterColumnLabel	string	none	The label of the column the filter should operate upon. It is mandatory to provide either this option or **`filterColumnIndex`**. If both present, **`filterColumnIndex`** takes precedence.
matchType	string	'prefix'	Whether the control should match exact values only (**`'exact'`**), prefixes starting from the beginning of the value (**`'prefix'`**) or any substring (**`'any'`**).
caseSensitive	boolean	false	Whether matching should be case sensitive or not.
useFormattedValue	boolean	false	Whether the control should match against cell formatted values or againt actual values.
ui	Object	null	An object with members to configure various

			aspects of the control's UI. To specify properties of this object, you can use object literal notation, as shown here: `{label: 'Name', labe`
ui.realtimeTrigger	boolean	true	Whether the control should match any time a key is pressed or only when the input field 'changes' (loss of focus or pressing the Enter key).
ui.label	string	auto	The label to display next to the input field. If unspecified, the label of the column the control operates on will be used.
ui.labelSeparator	string	none	A separator string appended to the label, to visually separate the label from the input field.
ui.labelStacking	string	'horizontal'	Whether the label should display above (vertical stacking) or beside (horizontal stacking) the input field. Use either **`'vertical'`** or **`'horizontal'`**.
ui.cssClass	string	'google-visualization-controls-stringfilter'	The CSS class to assign to the control, for custom styling.

上次更新日期：二月 23, 2017