* [Chart.js](http://docs.google.com/)
* [Getting Started](http://docs.google.com/getting-started/)
  + [Installation](http://docs.google.com/getting-started/installation.html)
  + [Integration](http://docs.google.com/getting-started/integration.html)
  + [Usage](http://docs.google.com/getting-started/usage.html)
* [General](http://docs.google.com/general/)
  + [Responsive](http://docs.google.com/general/responsive.html)
  + [Pixel Ratio](http://docs.google.com/general/device-pixel-ratio.html)
  + [Interactions](http://docs.google.com/general/interactions/)
    - [Events](http://docs.google.com/general/interactions/events.html)
    - [Modes](http://docs.google.com/general/interactions/modes.html)
  + [Options](http://docs.google.com/general/options.html)
  + [Colors](http://docs.google.com/general/colors.html)
  + [Fonts](http://docs.google.com/general/fonts.html)
* [Configuration](http://docs.google.com/configuration/)
  + [Animations](http://docs.google.com/configuration/animations.html)
  + [Layout](http://docs.google.com/configuration/layout.html)
  + [Legend](http://docs.google.com/configuration/legend.html)
  + [Title](http://docs.google.com/configuration/title.html)
  + [Tooltip](http://docs.google.com/configuration/tooltip.html)
  + [Elements](http://docs.google.com/configuration/elements.html)
* [Charts](http://docs.google.com/charts/)
  + [Line](http://docs.google.com/charts/line.html)
  + [Bar](http://docs.google.com/charts/bar.html)
  + [Radar](http://docs.google.com/charts/radar.html)
  + [Doughnut & Pie](http://docs.google.com/charts/doughnut.html)
  + [Polar Area](http://docs.google.com/charts/polar.html)
  + [Bubble](http://docs.google.com/charts/bubble.html)
  + [Scatter](http://docs.google.com/charts/scatter.html)
  + [Area](http://docs.google.com/charts/area.html)
  + [Mixed](http://docs.google.com/charts/mixed.html)
* [Axes](http://docs.google.com/axes/)
  + [Cartesian](http://docs.google.com/axes/cartesian/)
    - [Category](http://docs.google.com/axes/cartesian/category.html)
    - [Linear](http://docs.google.com/axes/cartesian/linear.html)
    - [Logarithmic](http://docs.google.com/axes/cartesian/logarithmic.html)
    - [Time](http://docs.google.com/axes/cartesian/time.html)
  + [Radial](http://docs.google.com/axes/radial/)
    - [Linear](http://docs.google.com/axes/radial/linear.html)
  + [Labelling](http://docs.google.com/axes/labelling.html)
  + [Styling](http://docs.google.com/axes/styling.html)
* [Developers](http://docs.google.com/)
  + [Chart.js API](http://docs.google.com/api.html)
  + [Updating Charts](http://docs.google.com/updates.html)
  + [Plugins](http://docs.google.com/plugins.html)
  + [New Charts](http://docs.google.com/charts.html)
  + [New Axes](http://docs.google.com/axes.html)
  + [Contributing](http://docs.google.com/contributing.html)
* [Additional Notes](http://docs.google.com/notes/)
  + [Comparison Table](http://docs.google.com/notes/comparison.html)
  + [Popular Extensions](http://docs.google.com/notes/extensions.html)
  + [License](http://docs.google.com/notes/license.html)
* [Published with GitBook](https://www.gitbook.com)

[**Chart.js API**](http://docs.google.com/)

Chart Prototype Methods

For each chart, there are a set of global prototype methods on the shared ChartType which you may find useful. These are available on all charts created with Chart.js, but for the examples, let's use a line chart we've made.

// For example:  
var myLineChart = new Chart(ctx, config);

## .destroy()

Use this to destroy any chart instances that are created. This will clean up any references stored to the chart object within Chart.js, along with any associated event listeners attached by Chart.js. This must be called before the canvas is reused for a new chart.

// Destroys a specific chart instance  
myLineChart.destroy();

## .update(config)

Triggers an update of the chart. This can be safely called after updating the data object. This will update all scales, legends, and then re-render the chart.

// duration is the time for the animation of the redraw in milliseconds  
// lazy is a boolean. if true, the animation can be interrupted by other animations  
myLineChart.data.datasets[0].data[2] = 50; // Would update the first dataset's value of 'March' to be 50  
myLineChart.update(); // Calling update now animates the position of March from 90 to 50.

**Note:** replacing the data reference (e.g. myLineChart.data = {datasets: [...]} only works starting **version 2.6**. Prior that, replacing the entire data object could be achieved with the following workaround: myLineChart.config.data = {datasets: [...]}.

A config object can be provided with additional configuration for the update process. This is useful when update is manually called inside an event handler and some different animation is desired.

The following properties are supported:

* **duration** (number): Time for the animation of the redraw in milliseconds
* **lazy** (boolean): If true, the animation can be interrupted by other animations
* **easing** (string): The animation easing function. See [Animation Easing](http://docs.google.com/configuration/animations.html) for possible values.

Example:

myChart.update({  
 duration: 800,  
 easing: 'easeOutBounce'  
})

See [Updating Charts](http://docs.google.com/updates.html) for more details.

## .reset()

Reset the chart to it's state before the initial animation. A new animation can then be triggered using update.

myLineChart.reset();

## .render(config)

Triggers a redraw of all chart elements. Note, this does not update elements for new data. Use .update() in that case.

See .update(config) for more details on the config object.

// duration is the time for the animation of the redraw in milliseconds  
// lazy is a boolean. if true, the animation can be interrupted by other animations  
myLineChart.render({  
 duration: 800,  
 lazy: false,  
 easing: 'easeOutBounce'  
});

## .stop()

Use this to stop any current animation loop. This will pause the chart during any current animation frame. Call .render() to re-animate.

// Stops the charts animation loop at its current frame  
myLineChart.stop();  
// => returns 'this' for chainability

## .resize()

Use this to manually resize the canvas element. This is run each time the canvas container is resized, but you can call this method manually if you change the size of the canvas nodes container element.

// Resizes & redraws to fill its container element  
myLineChart.resize();  
// => returns 'this' for chainability

## .clear()

Will clear the chart canvas. Used extensively internally between animation frames, but you might find it useful.

// Will clear the canvas that myLineChart is drawn on  
myLineChart.clear();  
// => returns 'this' for chainability

## .toBase64Image()

This returns a base 64 encoded string of the chart in it's current state.

myLineChart.toBase64Image();  
// => returns png data url of the image on the canvas

## .generateLegend()

Returns an HTML string of a legend for that chart. The legend is generated from the legendCallback in the options.

myLineChart.generateLegend();  
// => returns HTML string of a legend for this chart

## .getElementAtEvent(e)

Calling getElementAtEvent(event) on your Chart instance passing an argument of an event, or jQuery event, will return the single element at the event position. If there are multiple items within range, only the first is returned. The value returned from this method is an array with a single parameter. An array is used to keep a consistent API between the get\*AtEvent methods.

myLineChart.getElementAtEvent(e);  
// => returns the first element at the event point.

To get an item that was clicked on, getElementAtEvent can be used.

function clickHandler(evt) {  
 var firstPoint = myChart.getElementAtEvent(evt)[0];  
  
 if (firstPoint) {  
 var label = myChart.data.labels[firstPoint.\_index];  
 var value = myChart.data.datasets[firstPoint.\_datasetIndex].data[firstPoint.\_index];  
 }  
}

## .getElementsAtEvent(e)

Looks for the element under the event point, then returns all elements at the same data index. This is used internally for 'label' mode highlighting.

Calling getElementsAtEvent(event) on your Chart instance passing an argument of an event, or jQuery event, will return the point elements that are at that the same position of that event.

canvas.onclick = function(evt){  
 var activePoints = myLineChart.getElementsAtEvent(evt);  
 // => activePoints is an array of points on the canvas that are at the same position as the click event.  
};

This functionality may be useful for implementing DOM based tooltips, or triggering custom behaviour in your application.

## .getDatasetAtEvent(e)

Looks for the element under the event point, then returns all elements from that dataset. This is used internally for 'dataset' mode highlighting

myLineChart.getDatasetAtEvent(e);  
// => returns an array of elements

## .getDatasetMeta(index)

Looks for the dataset that matches the current index and returns that metadata. This returned data has all of the metadata that is used to construct the chart.

The data property of the metadata will contain information about each point, rectangle, etc. depending on the chart type.

Extensive examples of usage are available in the [Chart.js tests](https://github.com/chartjs/Chart.js/tree/master/test).

var meta = myChart.getDatasetMeta(0);  
var x = meta.data[0].\_model.x

results matching ""

No results matching ""