 [Chart.js](http://docs.google.com/docs/3.9.1/)

[Home](http://docs.google.com/docs/3.9.1/)

[API](http://docs.google.com/docs/3.9.1/api/)

[Samples](http://docs.google.com/docs/3.9.1/samples/)

Ecosystem Ecosystem

* [Awesome (opens new window)](https://github.com/chartjs/awesome)
* [Slack (opens new window)](https://chartjs-slack.herokuapp.com/)
* [Stack Overflow (opens new window)](https://stackoverflow.com/questions/tagged/chart.js)

[GitHub (opens new window)](https://github.com/chartjs/Chart.js)

[Home](http://docs.google.com/docs/3.9.1/)

[API](http://docs.google.com/docs/3.9.1/api/)

[Samples](http://docs.google.com/docs/3.9.1/samples/)

Ecosystem Ecosystem

* [Awesome (opens new window)](https://github.com/chartjs/awesome)
* [Slack (opens new window)](https://chartjs-slack.herokuapp.com/)
* [Stack Overflow (opens new window)](https://stackoverflow.com/questions/tagged/chart.js)

[GitHub (opens new window)](https://github.com/chartjs/Chart.js)

* [Chart.js](http://docs.google.com/docs/3.9.1/)
* Getting Started
* General
* Configuration
* Chart Types
* Axes
* Developers
  + [Developers](http://docs.google.com/docs/3.9.1/developers/)
  + [API](http://docs.google.com/docs/3.9.1/developers/api.html)
  + [New Axes](http://docs.google.com/docs/3.9.1/developers/axes.html)
  + [New Charts](http://docs.google.com/docs/3.9.1/developers/charts.html)
  + [Contributing](http://docs.google.com/docs/3.9.1/developers/contributing.html)
  + [Plugins](http://docs.google.com/docs/3.9.1/developers/plugins.html)
  + [Publishing an extension](http://docs.google.com/docs/3.9.1/developers/publishing.html)
  + [TypeDoc](http://docs.google.com/docs/3.9.1/api/)
  + [Updating Charts](http://docs.google.com/docs/3.9.1/developers/updates.html)

[**#**](#gjdgxs) API

For each chart, there are a set of global prototype methods on the shared chart type 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:  
const myLineChart = new Chart(ctx, config);

## [**#**](#30j0zll) .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();

## [**#**](#1fob9te) .update(mode?)

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.

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.

A mode string can be provided to indicate transition configuration should be used. Core calls this method using any of 'active', 'hide', 'reset', 'resize', 'show' or undefined. 'none' is also a supported mode for skipping animations for single update. Please see [animations](http://docs.google.com/docs/3.9.1/configuration/animations.html) docs for more details.

Example:

myChart.update('active');

See [Updating Charts](http://docs.google.com/docs/3.9.1/developers/updates.html) for more details.

## [**#**](#3znysh7) .reset()

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

myLineChart.reset();

## [**#**](#2et92p0) .render()

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

## [**#**](#tyjcwt) .stop()

Use this to stop any current animation. 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

## [**#**](#3dy6vkm) .resize(width?, height?)

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.

You can call .resize() with no parameters to have the chart take the size of its container element, or you can pass explicit dimensions (e.g., for [printing](http://docs.google.com/docs/3.9.1/configuration/responsive.html#printing-resizable-charts)).

// Resizes & redraws to fill its container element  
myLineChart.resize();  
// => returns 'this' for chainability  
// With an explicit size:  
myLineChart.resize(width, height);

## [**#**](#1t3h5sf) .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

## [**#**](#4d34og8) .toBase64Image(type?, quality?)

This returns a base 64 encoded string of the chart in its current state.

myLineChart.toBase64Image();  
// => returns png data url of the image on the canvas  
myLineChart.toBase64Image('image/jpeg', 1)  
// => returns a jpeg data url in the highest quality of the canvas

## [**#**](#2s8eyo1) .getElementsAtEventForMode(e, mode, options, useFinalPosition)

Calling getElementsAtEventForMode(e, mode, options, useFinalPosition) on your Chart instance passing an event and a mode will return the elements that are found. The options and useFinalPosition arguments are passed through to the handlers.

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

function clickHandler(evt) {  
 const points = myChart.getElementsAtEventForMode(evt, 'nearest', { intersect: true }, true);  
 if (points.length) {  
 const firstPoint = points[0];  
 const label = myChart.data.labels[firstPoint.index];  
 const value = myChart.data.datasets[firstPoint.datasetIndex].data[firstPoint.index];  
 }  
}

## [**#**](#17dp8vu) .getSortedVisibleDatasetMetas()

Returns an array of all the dataset meta's in the order that they are drawn on the canvas that are not hidden.

const visibleMetas = chart.getSortedVisibleDatasetMetas();

## [**#**](#3rdcrjn) .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, bar, etc. depending on the chart type.

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

const meta = myChart.getDatasetMeta(0);  
const x = meta.data[0].x;

## [**#**](#26in1rg) getVisibleDatasetCount

Returns the amount of datasets that are currently not hidden.

const numberOfVisibleDatasets = chart.getVisibleDatasetCount();

## [**#**](#lnxbz9) setDatasetVisibility(datasetIndex, visibility)

Sets the visibility for a given dataset. This can be used to build a chart legend in HTML. During click on one of the HTML items, you can call setDatasetVisibility to change the appropriate dataset.

chart.setDatasetVisibility(1, false); // hides dataset at index 1  
chart.update(); // chart now renders with dataset hidden

## [**#**](#35nkun2) toggleDataVisibility(index)

Toggles the visibility of an item in all datasets. A dataset needs to explicitly support this feature for it to have an effect. From internal chart types, doughnut / pie, polar area, and bar use this.

chart.toggleDataVisibility(2); // toggles the item in all datasets, at index 2  
chart.update(); // chart now renders with item hidden

## [**#**](#1ksv4uv) getDataVisibility(index)

Returns the stored visibility state of an data index for all datasets. Set by [toggleDataVisibility](#44sinio). A dataset controller should use this method to determine if an item should not be visible.

const visible = chart.getDataVisibility(2);

## [**#**](#2jxsxqh) hide(datasetIndex, dataIndex?)

If dataIndex is not specified, sets the visibility for the given dataset to false. Updates the chart and animates the dataset with 'hide' mode. This animation can be configured under the hide key in animation options. Please see [animations](http://docs.google.com/docs/3.9.1/configuration/animations.html) docs for more details.

If dataIndex is specified, sets the hidden flag of that element to true and updates the chart.

chart.hide(1); // hides dataset at index 1 and does 'hide' animation.  
chart.hide(0, 2); // hides the data element at index 2 of the first dataset.

## [**#**](#z337ya) show(datasetIndex, dataIndex?)

If dataIndex is not specified, sets the visibility for the given dataset to true. Updates the chart and animates the dataset with 'show' mode. This animation can be configured under the show key in animation options. Please see [animations](http://docs.google.com/docs/3.9.1/configuration/animations.html) docs for more details.

If dataIndex is specified, sets the hidden flag of that element to false and updates the chart.

chart.show(1); // shows dataset at index 1 and does 'show' animation.  
chart.show(0, 2); // shows the data element at index 2 of the first dataset.

## [**#**](#3j2qqm3) setActiveElements(activeElements)

Sets the active (hovered) elements for the chart. See the "Programmatic Events" sample file to see this in action.

chart.setActiveElements([  
 {datasetIndex: 0, index: 1},  
]);

## [**#**](#1y810tw) Static: getChart(key)

Finds the chart instance from the given key. If the key is a string, it is interpreted as the ID of the Canvas node for the Chart. The key can also be a CanvasRenderingContext2D or an HTMLDOMElement. This will return undefined if no Chart is found. To be found, the chart must have previously been created.

const chart = Chart.getChart("canvas-id");

## [**#**](#4i7ojhp) Static: register(chartComponentLike)

Used to register plugins, axis types or chart types globally to all your charts.

import { Chart, Tooltip, LinearScale, PointElement, BubbleController } from 'chart.js';  
Chart.register(Tooltip, LinearScale, PointElement, BubbleController);

## [**#**](#2xcytpi) Static: unregister(chartComponentLike)

Used to unregister plugins, axis types or chart types globally from all your charts.

Last Updated: 8/3/2022, 12:46:38 PM

←  [Developers](http://docs.google.com/docs/3.9.1/developers/)   [New Axes](http://docs.google.com/docs/3.9.1/developers/axes.html)  →