 [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) New Charts

Chart.js 2.0 introduced the concept of controllers for each dataset. Like scales, new controllers can be written as needed.

class MyType extends Chart.DatasetController {  
}  
Chart.register(MyType);  
// Now we can create a new instance of our chart, using the Chart.js API  
new Chart(ctx, {  
 // this is the string the constructor was registered at, ie Chart.controllers.MyType  
 type: 'MyType',  
 data: data,  
 options: options  
});

## [**#**](#30j0zll) Dataset Controller Interface

Dataset controllers must implement the following interface.

{  
 // Defaults for charts of this type  
 defaults: {  
 // If set to `false` or `null`, no dataset level element is created.  
 // If set to a string, this is the type of element to create for the dataset.  
 // For example, a line create needs to create a line element so this is the string 'line'  
 datasetElementType: string | null | false,  
 // If set to `false` or `null`, no elements are created for each data value.  
 // If set to a string, this is the type of element to create for each data value.  
 // For example, a line create needs to create a point element so this is the string 'point'  
 dataElementType: string | null | false,  
 }  
 // ID of the controller  
 id: string;  
 // Update the elements in response to new data  
 // @param mode : update mode, core calls this method using any of `'active'`, `'hide'`, `'reset'`, `'resize'`, `'show'` or `undefined`  
 update: function(mode) {}  
}

The following methods may optionally be overridden by derived dataset controllers.

{  
 // Draw the representation of the dataset. The base implementation works in most cases, and an example of a derived version  
 // can be found in the line controller  
 draw: function() {},  
 // Initializes the controller  
 initialize: function() {},  
 // Ensures that the dataset represented by this controller is linked to a scale. Overridden to helpers.noop in the polar area and doughnut controllers as these  
 // chart types using a single scale  
 linkScales: function() {},  
 // Parse the data into the controller meta data. The default implementation will work for cartesian parsing, but an example of an overridden  
 // version can be found in the doughnut controller  
 parse: function(start, count) {},  
}

## [**#**](#1fob9te) Extending Existing Chart Types

Extending or replacing an existing controller type is easy. Simply replace the constructor for one of the built in types with your own.

The built in controller types are:

* BarController
* BubbleController
* DoughnutController
* LineController
* PieController
* PolarAreaController
* RadarController
* ScatterController

These controllers are also available in the UMD package, directly under Chart. Eg: Chart.BarController.

For example, to derive a new chart type that extends from a bubble chart, you would do the following.

import {BubbleController} from 'chart.js';  
class Custom extends BubbleController {  
 draw() {  
 // Call bubble controller method to draw all the points  
 super.draw(arguments);  
 // Now we can do some custom drawing for this dataset. Here we'll draw a red box around the first point in each dataset  
 const meta = this.getMeta();  
 const pt0 = meta.data[0];  
 const {x, y} = pt0.getProps(['x', 'y']);  
 const {radius} = pt0.options;  
 const ctx = this.chart.ctx;  
 ctx.save();  
 ctx.strokeStyle = 'red';  
 ctx.lineWidth = 1;  
 ctx.strokeRect(x - radius, y - radius, 2 \* radius, 2 \* radius);  
 ctx.restore();  
 }  
};  
Custom.id = 'derivedBubble';  
Custom.defaults = BubbleController.defaults;  
// Stores the controller so that the chart initialization routine can look it up  
Chart.register(Custom);  
// Now we can create and use our new chart type  
new Chart(ctx, {  
 type: 'derivedBubble',  
 data: data,  
 options: options  
});

## [**#**](#3znysh7) TypeScript Typings

If you want your new chart type to be statically typed, you must provide a .d.ts TypeScript declaration file. Chart.js provides a way to augment built-in types with user-defined ones, by using the concept of "declaration merging".

When adding a new chart type, ChartTypeRegistry must contains the declarations for the new type, either by extending an existing entry in ChartTypeRegistry or by creating a new one.

For example, to provide typings for a new chart type that extends from a bubble chart, you would add a .d.ts containing:

import { ChartTypeRegistry } from 'chart.js'  
declare module 'chart.js' {  
 interface ChartTypeRegistry {  
 derivedBubble: ChartTypeRegistry['bubble']  
 }  
}

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

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