* [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)

[**New Charts**](http://docs.google.com/)

New Charts

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

Chart.controllers.MyType = Chart.DatasetController.extend({  
  
});  
  
  
// 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  
});

## Dataset Controller Interface

Dataset controllers must implement the following interface.

{  
 // Create elements for each piece of data in the dataset. Store elements in an array on the dataset as dataset.metaData  
 addElements: function() {},  
  
 // Create a single element for the data at the given index and reset its state  
 addElementAndReset: function(index) {},  
  
 // Draw the representation of the dataset  
 // @param ease : if specified, this number represents how far to transition elements. See the implementation of draw() in any of the provided controllers to see how this should be used  
 draw: function(ease) {},  
  
 // Remove hover styling from the given element  
 removeHoverStyle: function(element) {},  
  
 // Add hover styling to the given element  
 setHoverStyle: function(element) {},  
  
 // Update the elements in response to new data  
 // @param reset : if true, put the elements into a reset state so they can animate to their final values  
 update: function(reset) {},  
}

The following methods may optionally be overridden by derived dataset controllers

{  
 // Initializes the controller  
 initialize: function(chart, datasetIndex) {},  
  
 // 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() {},  
  
 // Called by the main chart controller when an update is triggered. The default implementation handles the number of data points changing and creating elements appropriately.  
 buildOrUpdateElements: function() {}  
}

## 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:

* Chart.controllers.line
* Chart.controllers.bar
* Chart.controllers.radar
* Chart.controllers.doughnut
* Chart.controllers.polarArea
* Chart.controllers.bubble

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

// Sets the default config for 'derivedBubble' to be the same as the bubble defaults.  
// We look for the defaults by doing Chart.defaults[chartType]  
// It looks like a bug exists when the defaults don't exist  
Chart.defaults.derivedBubble = Chart.defaults.bubble;  
  
// I think the recommend using Chart.controllers.bubble.extend({ extensions here });  
var custom = Chart.controllers.bubble.extend({  
 draw: function(ease) {  
 // Call super method first  
 Chart.controllers.bubble.prototype.draw.call(this, ease);  
  
 // Now we can do some custom drawing for this dataset. Here we'll draw a red box around the first point in each dataset  
 var meta = this.getMeta();  
 var pt0 = meta.data[0];  
 var radius = pt0.\_view.radius;  
  
 var ctx = this.chart.chart.ctx;  
 ctx.save();  
 ctx.strokeStyle = 'red';  
 ctx.lineWidth = 1;  
 ctx.strokeRect(pt0.\_view.x - radius, pt0.\_view.y - radius, 2 \* radius, 2 \* radius);  
 ctx.restore();  
 }  
});  
  
// Stores the controller so that the chart initialization routine can look it up with  
// Chart.controllers[type]  
Chart.controllers.derivedBubble = custom;  
  
// Now we can create and use our new chart type  
new Chart(ctx, {  
 type: 'derivedBubble',  
 data: data,  
 options: options,  
});

### Bar Controller

The bar controller has a special property that you should be aware of. To correctly calculate the width of a bar, the controller must determine the number of datasets that map to bars. To do this, the bar controller attaches a property bar to the dataset during initialization. If you are creating a replacement or updated bar controller, you should do the same. This will ensure that charts with regular bars and your new derived bars will work seamlessly.

results matching ""

No results matching ""