 [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 Axes

Axes in Chart.js can be individually extended. Axes should always derive from Chart.Scale but this is not a mandatory requirement.

class MyScale extends Chart.Scale {  
 /\* extensions ... \*/  
}  
MyScale.id = 'myScale';  
MyScale.defaults = defaultConfigObject;  
// MyScale is now derived from Chart.Scale

Once you have created your scale class, you need to register it with the global chart object so that it can be used.

Chart.register(MyScale);  
// If the new scale is not extending Chart.Scale, the prototype can not be used to detect what  
// you are trying to register - so you need to be explicit:  
// Chart.registry.addScales(MyScale);

To use the new scale, simply pass in the string key to the config when creating a chart.

const lineChart = new Chart(ctx, {  
 data: data,  
 type: 'line',  
 options: {  
 scales: {  
 y: {  
 type: 'myScale' // this is the same id that was set on the scale  
 }  
 }  
 }  
});

## [**#**](#30j0zll) Scale Properties

Scale instances are given the following properties during the fitting process.

{  
 left: number, // left edge of the scale bounding box  
 right: number, // right edge of the bounding box  
 top: number,  
 bottom: number,  
 width: number, // the same as right - left  
 height: number, // the same as bottom - top  
 // Margin on each side. Like css, this is outside the bounding box.  
 margins: {  
 left: number,  
 right: number,  
 top: number,  
 bottom: number  
 },  
 // Amount of padding on the inside of the bounding box (like CSS)  
 paddingLeft: number,  
 paddingRight: number,  
 paddingTop: number,  
 paddingBottom: number  
}

## [**#**](#1fob9te) Scale Interface

To work with Chart.js, custom scale types must implement the following interface.

{  
 // Determines the data limits. Should set this.min and this.max to be the data max/min  
 determineDataLimits: function() {},  
 // Generate tick marks. this.chart is the chart instance. The data object can be accessed as this.chart.data  
 // buildTicks() should create a ticks array on the axis instance, if you intend to use any of the implementations from the base class  
 buildTicks: function() {},  
 // Get the label to show for the given value  
 getLabelForValue: function(value) {},  
 // Get the pixel (x coordinate for horizontal axis, y coordinate for vertical axis) for a given value  
 // @param index: index into the ticks array  
 getPixelForTick: function(index) {},  
 // Get the pixel (x coordinate for horizontal axis, y coordinate for vertical axis) for a given value  
 // @param value : the value to get the pixel for  
 // @param [index] : index into the data array of the value  
 getPixelForValue: function(value, index) {},  
 // Get the value for a given pixel (x coordinate for horizontal axis, y coordinate for vertical axis)  
 // @param pixel : pixel value  
 getValueForPixel: function(pixel) {}  
}

Optionally, the following methods may also be overwritten, but an implementation is already provided by the Chart.Scale base class.

{  
 // Adds labels to objects in the ticks array. The default implementation simply calls this.options.ticks.callback(numericalTick, index, ticks);  
 generateTickLabels: function() {},  
 // Determine how much the labels will rotate by. The default implementation will only rotate labels if the scale is horizontal.  
 calculateLabelRotation: function() {},  
 // Fits the scale into the canvas.  
 // this.maxWidth and this.maxHeight will tell you the maximum dimensions the scale instance can be. Scales should endeavour to be as efficient as possible with canvas space.  
 // this.margins is the amount of space you have on either side of your scale that you may expand in to. This is used already for calculating the best label rotation  
 // You must set this.minSize to be the size of your scale. It must be an object containing 2 properties: width and height.  
 // You must set this.width to be the width and this.height to be the height of the scale  
 fit: function() {},  
 // Draws the scale onto the canvas. this.(left|right|top|bottom) will have been populated to tell you the area on the canvas to draw in  
 // @param chartArea : an object containing four properties: left, right, top, bottom. This is the rectangle that lines, bars, etc will be drawn in. It may be used, for example, to draw grid lines.  
 draw: function(chartArea) {}  
}

The Core.Scale base class also has some utility functions that you may find useful.

{  
 // Returns true if the scale instance is horizontal  
 isHorizontal: function() {},  
 // Returns the scale tick objects ({label, major})  
 getTicks: function() {}  
}

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

←  [API](http://docs.google.com/docs/3.9.1/developers/api.html)   [New Charts](http://docs.google.com/docs/3.9.1/developers/charts.html)  →