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

New Axes

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

let MyScale = Chart.Scale.extend({  
 /\* extensions ... \*/  
});  
  
// 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. A default config for the scale may be provided when registering the constructor. The first parameter to the register function is a string key that is used later to identify which scale type to use for a chart.

Chart.scaleService.registerScaleType('myScale', MyScale, defaultConfigObject);

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

var lineChart = new Chart(ctx, {  
 data: data,  
 type: 'line',  
 options: {  
 scales: {  
 yAxes: [{  
 type: 'myScale' // this is the same key that was passed to the registerScaleType function  
 }]  
 }  
 }  
})

## 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,  
}

## 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 value to show for the data at the given index of the the given dataset, ie this.chart.data.datasets[datasetIndex].data[index]  
 getLabelForIndex: function(index, datasetIndex) {},  
  
 // Get the pixel (x coordinate for horizontal axis, y coordinate for vertical axis) for a given value  
 // @param index: index into the ticks array  
 // @param includeOffset: if true, get the pixel halfway between the given tick and the next  
 getPixelForTick: function(index, includeOffset) {},  
  
 // 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  
 // @param datasetIndex : index of the dataset the value comes from  
 // @param includeOffset : if true, get the pixel halfway between the given tick and the next  
 getPixelForValue: function(value, index, datasetIndex, includeOffset) {}  
  
 // 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.

// Transform the ticks array of the scale instance into strings. The default implementation simply calls this.options.ticks.callback(numericalTick, index, ticks);  
 convertTicksToLabels: function() {},  
  
 // Determine how much the labels will rotate by. The default implementation will only rotate labels if the scale is horizontal.  
 calculateTickRotation: 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() {},  
  
 // Get the correct value from the value from this.chart.data.datasets[x].data[]  
 // If dataValue is an object, returns .x or .y depending on the return of isHorizontal()  
 // If the value is undefined, returns NaN  
 // Otherwise returns the value.  
 // Note that in all cases, the returned value is not guaranteed to be a Number  
 getRightValue: function(dataValue) {},  
}

results matching ""

No results matching ""