

VAADIN TRAINING

Charts



Get started

Maven dependency

First, add maven dependency in your pom file.


XML

```
<dependency>
  <groupId>com.vaadin</groupId>
  <artifactId>vaadin-charts</artifactId>
  <version>4.0.4</version>
</dependency>
```


Licence

Vaadin Charts is a commercial tool, which means that you need to get a licence.

Go to <https://vaadin.com/pro/licenses> to download your license key, and copy the file to your home directory.

 **Vaadin Charts**
The most comprehensive visualization library available for Vaadin.
Supported Vaadin versions: 6, 7, 8.

License key



c90d47da-9379-4450-9b4c-42df3f41db8a [Download](#)

Copy-paste the license key to a file called `vaadin.charts.developer.license` or download it directly.
After downloading you need to copy the file to your home directory
(`/Users/<username>/`).

(Re)Compile Widgetset

Vaadin Charts has client side code, which means that you need to (Re)Compile the widgetset after you the charts dependency is added. To (re)compile widgetset, run

```
mvn clean package
```

Basic Usage

Instantiate a Chart

- Usually you start by creating a Chart instance.
- Chart is just a regular Vaadin Components.
- Chart type can be given in the constructor or later in the chart model.
- A chart has a height of 400 pixels and width of 100%.

```
XML      Chart chart = new Chart(ChartType.COLUMN);
           chart.setWidth("400px"); // 100% by default
           chart.setHeight("300px"); // 400px by default
           ...
           layout.addComponent(chart);
```

Chart Types

- All the available chart types can be found at <https://vaadin.com/components/browse#charts>
- The chart type is normally specified in the constructor of the Chart object.
- The available chart types are defined in the ChartType enum.
- Chart type can also be read/write with chartType property in Chart model, which you can get with getConfiguraton().getChart().
- Each chart type has its specific plot options and support its specific collection of chart features. They also have specific requirements for the data series.

Configuration

- The Chart object handles basic Vaadin components properties.
- All the chart-specific properties are in a separate Configuration object, which you can access with the getConfiguration() method.

Configure title and subtitle

```
JAVA    Configuration conf = chart.getConfiguration();
        conf.setTitle("Reindeer Kills by Predators");
        conf.setSubTitle("Kills Grouped by Counties");
```

Configure Legend

The legend is a box that describes the data series shown in the chart. Properties can be customised for Legend are:

- **title**
Specifies the title of the legend
- **enabled**
Enables or disables the legend. Defaults to true.
- **alignment**
Specifies the horizontal alignment of the **legend box** within the chart area. Defaults to `HorizontalAlign.CENTER`
- **verticalAlign**
Specifies the vertical alignment of the **legend box** with the chart area. Defaults to `VerticalAlign.BOTTOM`.
- **layout**
Specifies the layout direction of the **legend items**. Defaults to `LayoutDirection.HORIZONTAL`

```
JAVA    Legend legend = configuration.getLegend();
        legend.getTitle().setText("City");
        legend.setLayout(LayoutDirection.VERTICAL);
        legend.setAlign(HorizontalAlign.LEFT);
        legend.setVerticalAlign(VerticalAlign.MIDDLE);
```

Adding Data to Chart

- The data displayed in a chart is stored as a list of `Series` objects in the chart configuration.
- New data series is added to a chart with the `addSeries()` method.

```
JAVA    conf.addSeries(Series series);
```

- The data can be specified with a number of different series types: `DataSetSeries`, `ListSeries`, `RangeSeries`, `TreeSeries`, `DataProviderSeries` and `HeatSeries`.

DataSeries

The `DataSeries` can represent a sequence of data points. Each data point is represented as `DataSeriesItem` object.

- To add a `DataSeriesItem`, use `DataSeries.add()`

- To remove a `DataSetItem`, use `DataSet.remove()`
- After updating a `DataSetItem` object, you need to call `DataSet.update()`

JAVA

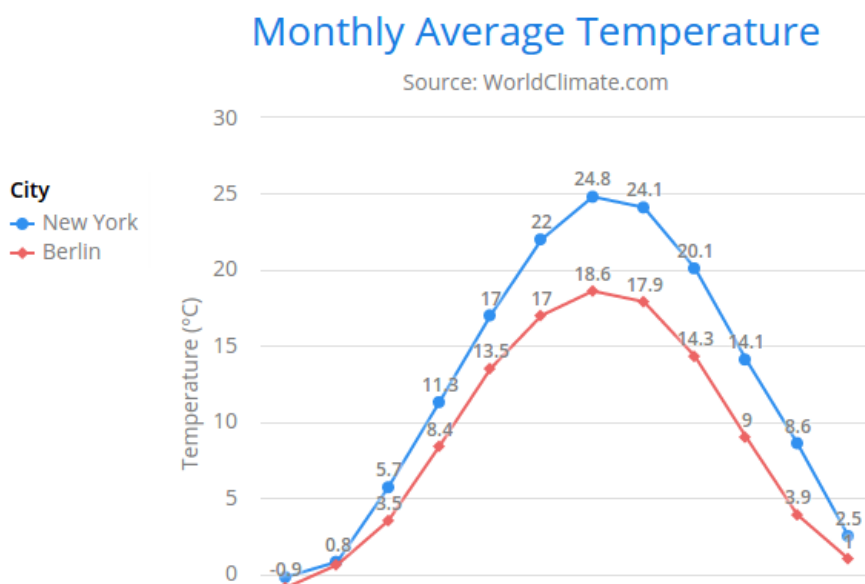
```
DataSet series = new DataSet();

DataSetItem item = new DataSetItem(1959, 181091)
//add item
series.add(item);
series.add(new DataSetItem(1960, 201485));
series.add(new DataSetItem(1961, 188105));
series.add(new DataSetItem(1962, 177206));

//update item
item.setY(181092);
series.update(item);

//remove item
series.remove(item);
```

- Range charts expect the Y values to be specified as minimum-maximum value pairs.
- The `DataSetItem` provides `setLow()` and `setHigh()` methods to set the minimum and maximum values for a data point.
- It's also possible to specify the minimum and maximum values in the constructor.



```

JAVA      RangeSeries series =
              new RangeSeries("Temperature Extremes");

              // Give low-high values in constructor
              series.add(new DataSeriesItem(0, -51.5, 10.9));
              series.add(new DataSeriesItem(1, -49.0, 11.8));

              // Set low-high values with setters
              DataSeriesItem point = new DataSeriesItem();
              point.setX(2);
              point.setLow(-44.3);
              point.setHigh(17.5);
              series.add(point);

```

RangeSeries

- The RangeSeries is a helper class that extends DataSeries to allow specifying interval data a bit easier.
 - Y axis values are provided as a list of minimum-maximum values ranges.
-

```

JAVA      //Y axis values are given as an array of minimum-maximum value pairs
              RangeSeries series =
                  new RangeSeries("Temperature Ranges",
                      new Double[]{-51.5,10.9},
                      new Double[]{-49.0,11.8},
                      ...
                      new Double[]{-47.0,10.8});
              conf.addSeries(series);

```

ListSeries

The ListSeries is essentially a helper type that makes handling of simple sequential data easier than with DataSeries.

- The data points are assumed to be at a constant interval on the X axis.
- pointStart property specifies the value to start from, default is 0.
- pointInterval property specifies the interval, default is 1.
- Both pointStart and pointInterval can be configured in PlotOptions.

- The Y axis values are given as constructor parameters

```
JAVA    //Y axis values are given as constructor parameters
ListSeries series = new ListSeries("Total Reindeer Population",
    181091, 201485, 188105);
```

- or using setData() method

```
JAVA    ListSeries series = new ListSeries("Diameter");
series.setData(4900, 12100, 12800,
    6800, 143000, 125000,
    51100, 49500);
```

Multiple Data Series

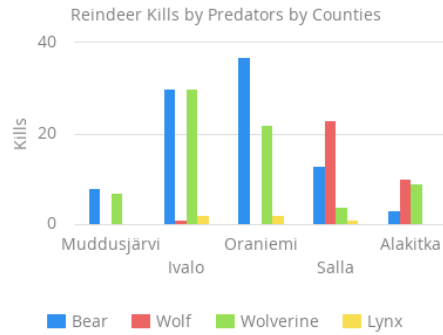
- Multiple series can be added to chart, this can be achieved by adding series to chart configuration with `conf.addSeries()`
- Most chart types support multiple data series.
 - In line charts you can have multiple lines.
 - In column charts the columns for different series are grouped by category.

```
JAVA    String predators[] = {"Bear", "Wolf", "Wolverine", "Lynx"};
    int kills[][] = {           // Location:
        {8, 0, 7, 0}, // Muddusjarvi
        {30, 1, 30, 2}, // Ivalo
        {37, 0, 22, 2}, // Oraniemi
        {13, 23, 4, 1}, // Salla
        {3, 10, 9, 0}, // Alakitka
    };

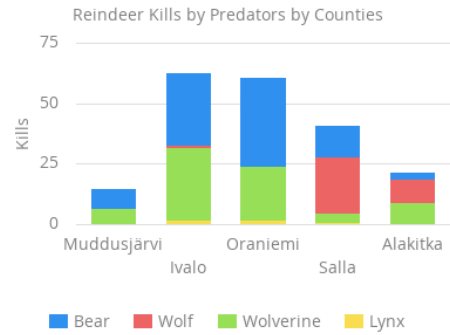
    // Create a data series for each numeric column in the table
    for (int predator = 0; predator < 4; predator++) {
        ListSeries series = new ListSeries();
        series.setName(predators[predator]);

        for (int location = 0; location < kills.length; location++)
            series.addData(kills[location][predator]);
        conf.addSeries(series);
    }
```

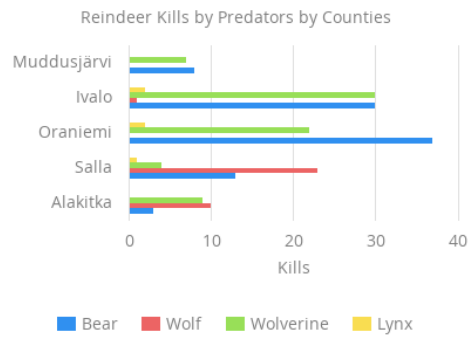
Column



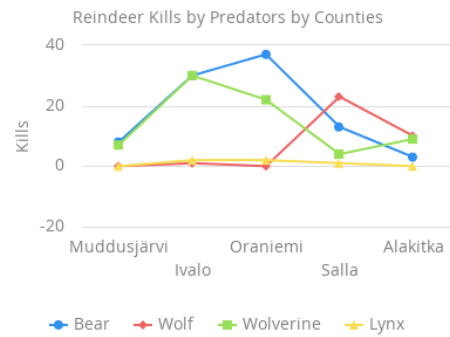
Stacked Column



Bar



Line



More Advanced Usages

DataProvider

DataProvider provides unified API for fetching data. It has methods for getting items, getting size etc. There are In-Memory and Lazy Load variants. For this training, we only use In-Memory data providers, as most likely you would just want to show all the data in the chart.

The most commonly used In-Memory data provider is ListDataProvider. A ListDataProvider can be created from a collection.

```
JAVA      DataProvider listDataProvider = DataProvider.ofCollection(items);
```

One of the powerful feature of DataProvider is that you can do filtering. The API for filtering is:

```
JAVA      public void setFilter(SerializablePredicate<T> filter)
```

A SerializablePredicate is just a normal Java Predicate which implements Serializable interface. Here is an example of filtering with ListDataProvider

```
JAVA      dataProvider.setFilter(person -> person.getEmail() != null);
```

DataProviderSeries

The DataProviderSeries is an adapter for using a Vaadin DataProvider as a DataSeries in a chart. A DataProvider can be wrapped into a DataProviderSeries in the constructor of DataProviderSeries.

```
JAVA      // The bean
      public class Order {
          private String description;
          private int quantity;
          private double unitPrice;
      }
```

```

JAVA    // The data
          List<Order> orders = new ArrayList<>();
          orders.add(new Order("Domain Name", 3, 7.99));
          ...
          DataProvider<Order, ?> dataProvider = new ListDataProvider<>(orders);

```

```

JAVA    // The data series
          DataProviderSeries<Order> series = new DataProviderSeries<>(dataProvider);

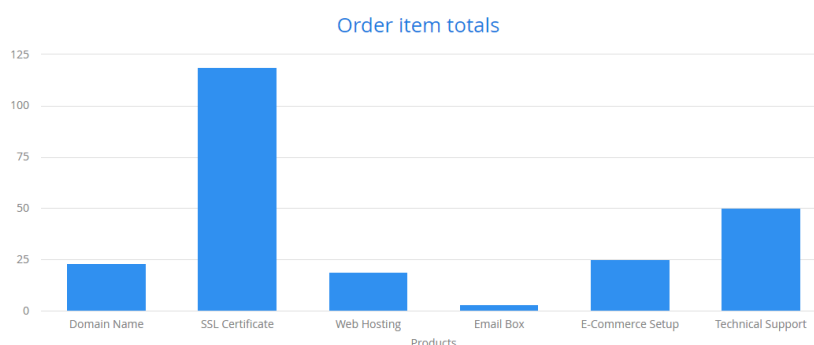
```

setPointName(), setX(), setY() can define which parts of the bean in the DataProvider are used in the chart.

```

JAVA    //configure x and y
          series.setX(Order::getDescription);
          series.setY(Order::getUnitPrice);

```



Configure Axes

Axes are accessible from Configuration with `getXAxis()` and `getYAxis()`

```

JAVA    Configuration conf = chart.getConfiguration();
          XAxis xaxis = conf.getXAxis();
          YAxis yaxis = conf.getYAxis();

```

One of the most common tasks for chart is customising its axes. For example, set the axis titles. When an axis is categorical rather than numeric, you can define category label for the items

```

JAVA    XAxis xaxis = conf.getXAxis();
          xaxis.setCategories("Mercury", "Venus", "Earth", "Mars", "Jupiter",
                             "Saturn", "Uranus", "Neptune");
          xaxis.setTitle("Planet");

```

AXES TYPES

Axes can be one of the following types, which you can set with `setType()`. The axis types are enumerated under `AxisType`. `LINEAR` is the default.

- **LINEAR**(default)
For numeric values in linear scale.
- **LOGARITHMIC**
For numerical values, as in the linear axis, but the axis will be scaled in the logarithmic scale. The minimum for the axis must be a positive non-zero value ($\log(0)$ is not defined, as it has limit at negative infinity when the parameter approaches zero).
- **DATETIME**
Enables date/time mode in the axis. The date/time values are expected to be given either as a **Date** object or in milliseconds since the Java (or Unix) date epoch on January 1st 1970 at 00:00:00 GMT. You can get the millisecond representation of Java **Date** with `getTime()`.
- **CATEGORY**
Enables using categorical data for the axis, as described in more detail later. With this axis type, the category labels are determined from the labels of the data points in the data series, without need to set them explicitly with `setCategories()`.

LABEL

The format and styles of labels in an axis is defined in the `Labels` object, which you can get with `getLabels()` from the axis. Properties can be defined for a label are:

- **rotation**
defines the rotation of the labels in degrees
- **step**
defines tick interval for showing labels, so that labels are shown at every *nth* tick
- **style**
defines the style for labels. The property is a **Style** Object.

```
JAVA    Labels xlabels = xaxis.getLabels();
        xlabels.setAlign(HorizontalAlign.CENTER); // Default
        xlabels.getStyle().setColor(SolidColor.GREEN);
        xlabels.getStyle().setFontWeight(FontWeight.BOLD);
        xlabels.setRotation(-45);
        xlabels.setStep(2); // Every 2 major tick
```

PlotOptions

- `PlotOptions` is used to configure the data series in the chart.
- Each chart type has its own `PlotOptions`, like: `PlotOptionsLine`, `PlotOptionsColumn`, `PlotOptionsGauge` etc.
- `PlotOptions` can be configured for entire Chart

```
JAVA      PlotOptionsColumn plotOptions = new PlotOptionsColumn();
            Chart chart = new Chart();
            Configuration configuration = chart.getConfiguration();
            configuration.setPlotOptions(plotOptions);
```

- or separate data series.

```
JAVA      PlotOptionsColumn plotOptions = new PlotOptionsColumn();
            ListSeries series = new ListSeries(50, 60, 70, 80);
            series.setPlotOptions(plotOptions);
```

PlotOptions are grouped as below:

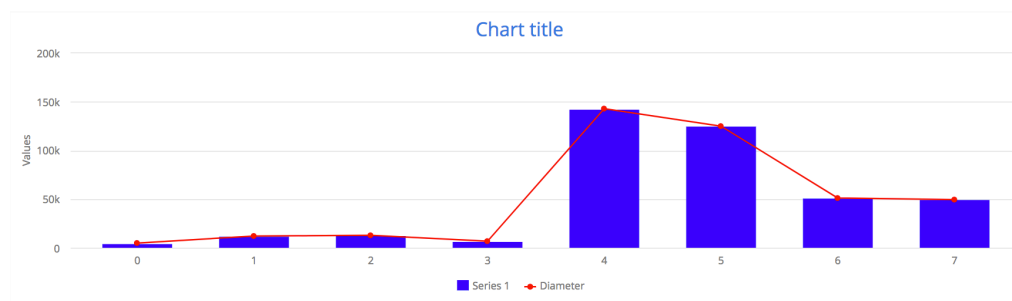
- AreaOptions: PlotOptionsArea, PlotOptionsArearange, PlotOptionsAreaspline, PlotOptionsAreasplinerange
- ColumnOptions: PlotOptionsBar, PlotOptionsColumn, PlotOptionsColumnrange
- GaugeOptions: PlotOptionsGauge, PlotOptionsSolidgauge
- PointOptions: PlotOptionsLine, PlotOptionsSpline, PlotOptionsScatter
- PyramidOptions: PlotOptionsPyramid, PlotOptionsFunnel
- OhlcOptions: PlotOptionsOhlc, PlotOptionsCandlestick

Mixed chart types

By setting different PlotOptions for multiple Dataseries, you can have a Chart with mixed types.

```
JAVA      DataSeries series1 = new DataSeries();
            PlotOptionsColumn options1 = new PlotOptionsColumn();
            series1.setPlotOptions(options1);
            series1.setData(4900, 12100, 12800, 6800, 143000, 125000, 51100, 49500);
            conf.addSeries(series1);

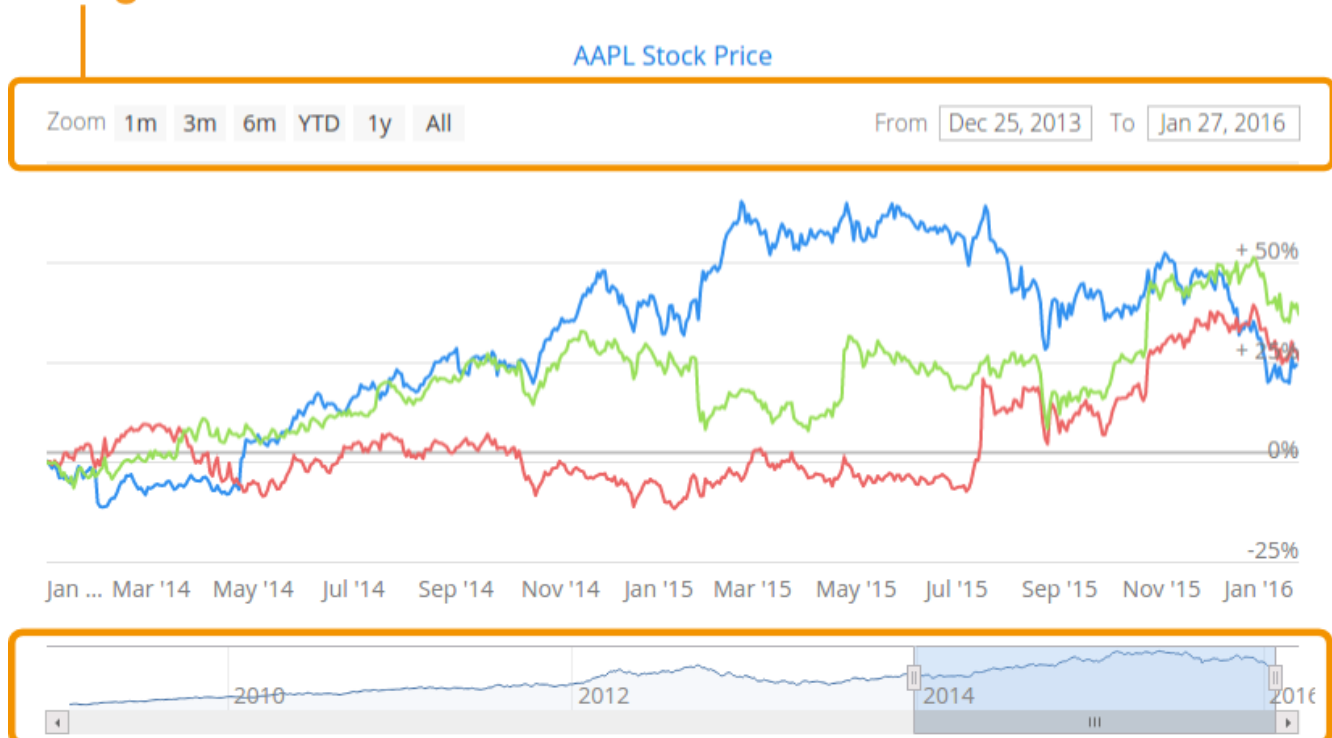
            // A data series as line graph
            ListSeries series2 = new ListSeries("Diameter");
            PlotOptionsLine options2 = new PlotOptionsLine();
            series2.setPlotOptions(options2);
            series2.setData(4900, 12100, 12800, 6800, 143000, 125000, 51100, 49500);
            conf.addSeries(series2);
```



Timeline

- A chart's timeline feature allows selecting different time ranges for which to display the chart data, as well as navigating between such ranges.
- Adding a timeline to your chart is very easy, just call `chart.setTimeline(true)`.
- The X values of the corresponding data-series should be of the type `Date`.
- Integer values can be used, they will be interpreted as milliseconds since the 01/01/1070 epoch.
- By default, navigator is shown, to hide the navigator, call `chart.getConfiguration().getNavigator().setEnabled(false)`.
- By default, range selector is shown, to hide the range selector, call `chart.getConfiguration().getRangeSelector().setEnabled(false)`.

Range selector



Navigator

Export

- Sometimes you might export your charts as images, this can be achieved rather easily with `SVGGenerator` provided by Vaadin.
- Internally, `SVGGenerator` uses `PhantomJS`, so you have to install `PhantomJS`, and make it available on your system path.
- To get the svg, you just call: `String svg = SVGGenerator.getInstance().generate(chart.getConfiguration());`
- `SVGGenerator` depends on [PhantomJS](#), so you need to first download and install [PhantomJS](#) before using `SVGGenerator`.

```

JAVA Button downloadButton = new Button("Download");
FileDownloader fileDownloader = new FileDownloader(new StreamResource(() -> {
    try {
        String svg = SVGGenerator.getInstance().generate(chart.getConfiguration());
        return new ByteArrayInputStream(svg.getBytes(StandardCharsets.UTF_8.name()));
    } catch (UnsupportedEncodingException e1) {
        return null;
    }
}, "chart.svg"));
fileDownloader.extend(downloadButton);

```

Chart Language

Lang class provides an opportunity for internationalising charts. You can specify a custom decimal point, names of months, weekdays and their abbreviated forms. You can also specify the text to display when the chart contains no data.

Lang option is global and can be set with `setLang()` in the `ChartOptions`:

`ChartOptions.get().setLang(fi)`

```

JAVA final Lang fi = new Lang();

// Set language properties
fi.setDecimalPoint(",");

fi.setMonths(new String[] { "Tammikuu", "Helmikuu", "Maaliskuu",
    "Huhtikuu", "Toukokuu", "Kesäkuu",
    "Heinäkuu", "Elokuu", "Syyskuu",
    "Lokakuu", "Marraskuu", "Joulukuu" });

fi.setShortMonths(new String[] { "Tammi", "Helmi", "Maalis",
    "Huhti", "Touko", "Kesä",
    "Heinä", "Elo", "Syys",
    "Loka", "Marras", "Joulu" });

fi.setWeekdays(new String[] { "Ma", "Ti", "Ke", "To", "Pe", "La", "Su" });
fi.setNoData("Data puuttuu");

```