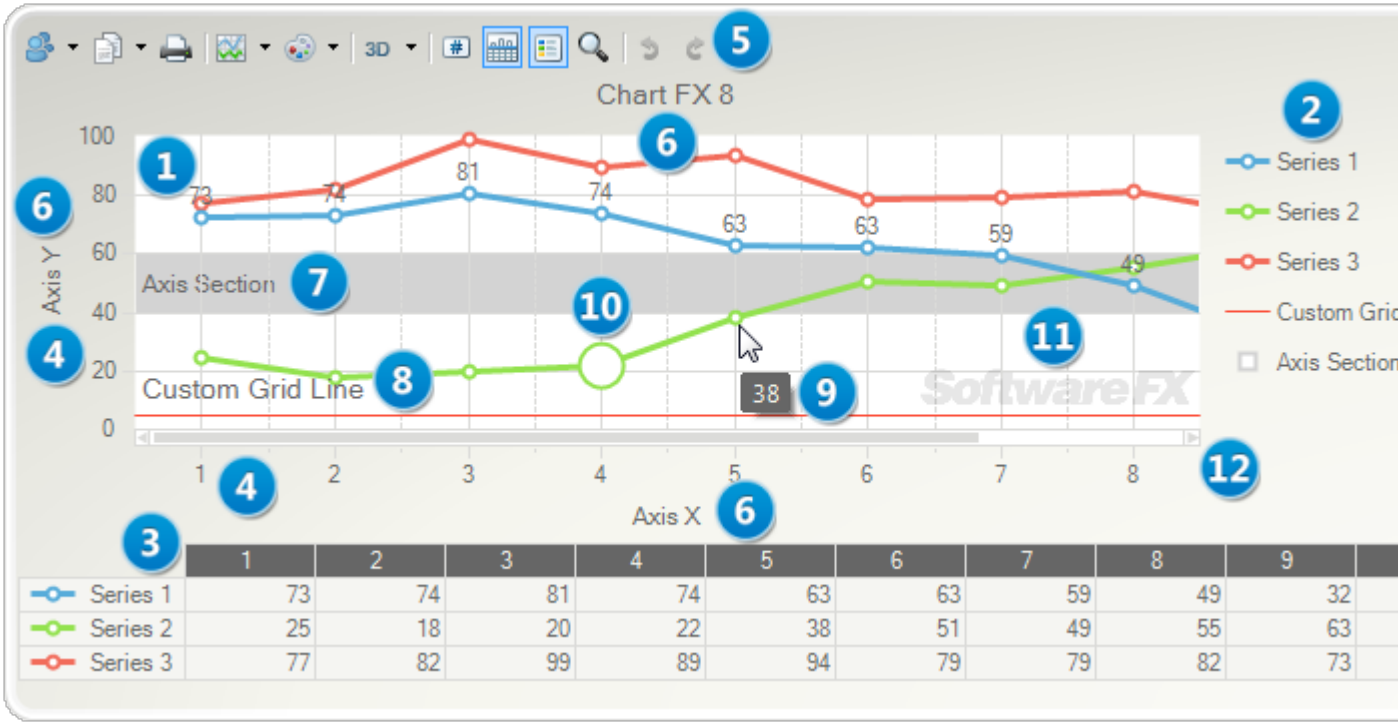
Definitions



1. Plot Area
2. Legend Box
3. Data Grid
4. Axes
5. Toolbar
6. Titles
7. Axis Sections
8. Grid Lines
9. Tooltips
10. Markers
11. Annotations
12. Axis Scrolling

DataFields

One of the biggest advances of Chart FX 8 for Java Server is in data analysis. The use of this new feature begins when the data is first passed to the chart. But before we analyze the different

ways data can be passed to the chart, we need to understand how the chart stores the data. Chart FX 8 for Java Server stores data in DataField.

DataFields

Each column of data passed to the chart will correspond to a data field, although a data field does not necessarily have to come from a column (or from the data at all). Data fields can populate series, axis and even tooltips. It also contains information to be used by the chart such as aggregation method or group membership. You can use the chart to create the data fields automatically, by invoking the fillFromSchema method:

Java

```
chart1.getDataSourceSettings().fillFromSchema();
```

fillFromSchema will create data fields obeying the data type associated with it. DateTime columns will generate DateTimeDataField, numeric columns will generate NumericDataField, string columns will generate StringDataField, and so on.

The types of DataField available are:

- BooleanDataField
- CountDataField
- DateTimeDataField
- ImageDataField
- NumericDataField
- StringDataField
- TimeSpanDataField

Note: If you use fillFromSchema, the key for the datafield will come from your data (column name). If there is not a column name, the keys will be "Field1", "Field2", "Field3", etc.

To create a data field manually, you need to match the data field type with the data, set its data path (column name for example) and add it to the collection of data fields.

Java

```
NumericDataField dfField1;

dfField1 = new NumericDataField();

dfField1.setDataPath("Field_1");

chart1.getDataSourceSettings().getDataFields().add(dfField1);
```

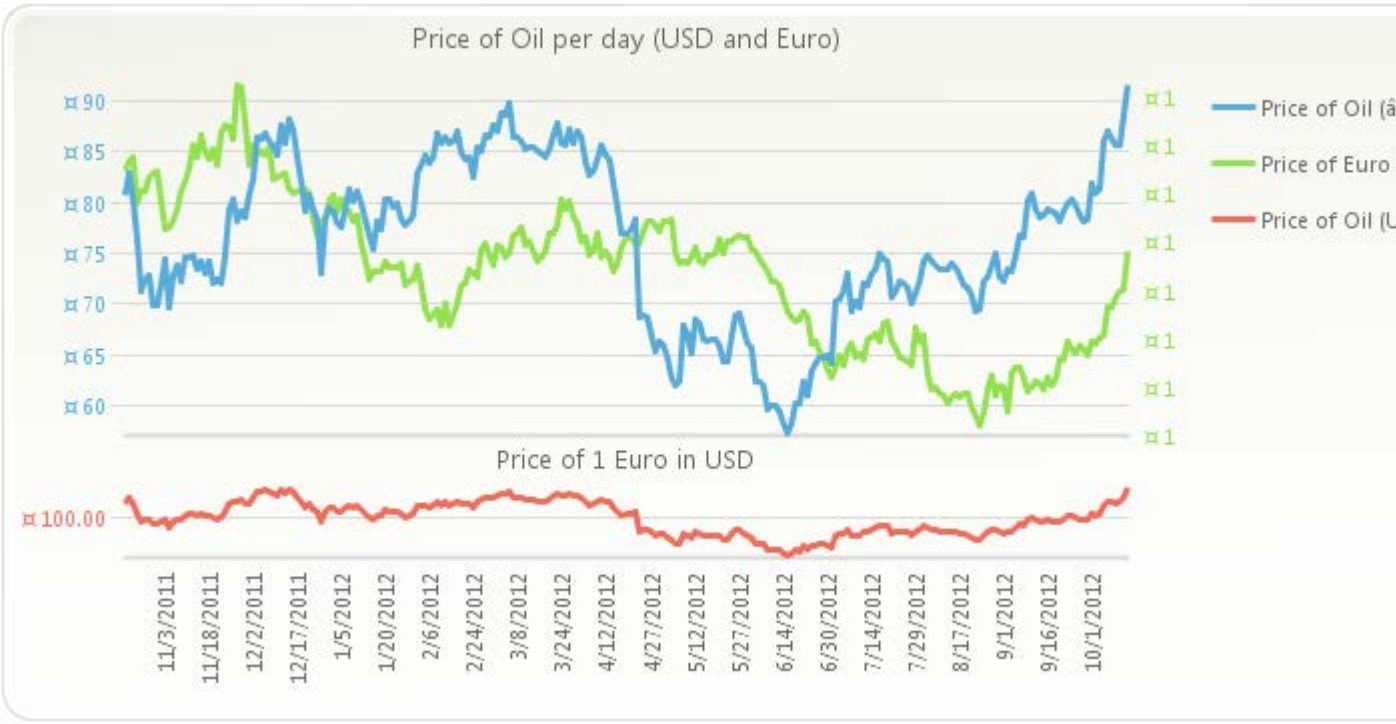
Once the data fields are created they can be configured. For instance, to set up so a data field gets aggregated by adding its values if grouped, you need to set its aggregation method to sum:

Java

```
chart1.getDataSourceSettings().getDataFields().getTValue("Field_1").setAggregation
Method(NumericDataField.AggregationMethods.getSum());
```

Formula Data Fields

While most getDataFields come straight from your data, sometimes it is necessary to perform calculations on the data being imported before displaying it on the chart. Or even creating a new getDataFields, populated by calculated values from other data. The image below shows how to achieve this in Chart FX.



Business (CLR) Objects

Using Business Objects with Chart FX

In modern day applications employing various business objects throughout, there may be times where the contents of a business object must be represented via a chart. Lets first assume a very basic Department object existed with the methods Name, Budget & Expenses as defined below:

Java

```
public class Department

{

    private String _name;

    private double _budget;

    private double _expenses;



    public Department(String name, double budget, double expenses)

    {

            _name = name;

            _budget = budget;

            _expenses = expenses;

    }



    public final String getName()

    {

            return _name;

    }

    public final void setName(String value)

    {

            _name = value;

    }



    public final double getBudget()
```

```
    {
            return _budget;
    }
    public final void setBudget(double value)
    {
            _budget = value;
    }

    public final double getExpenses()
    {
            return _expenses;
    }
```

```
    public final void setExpenses(double value)
```

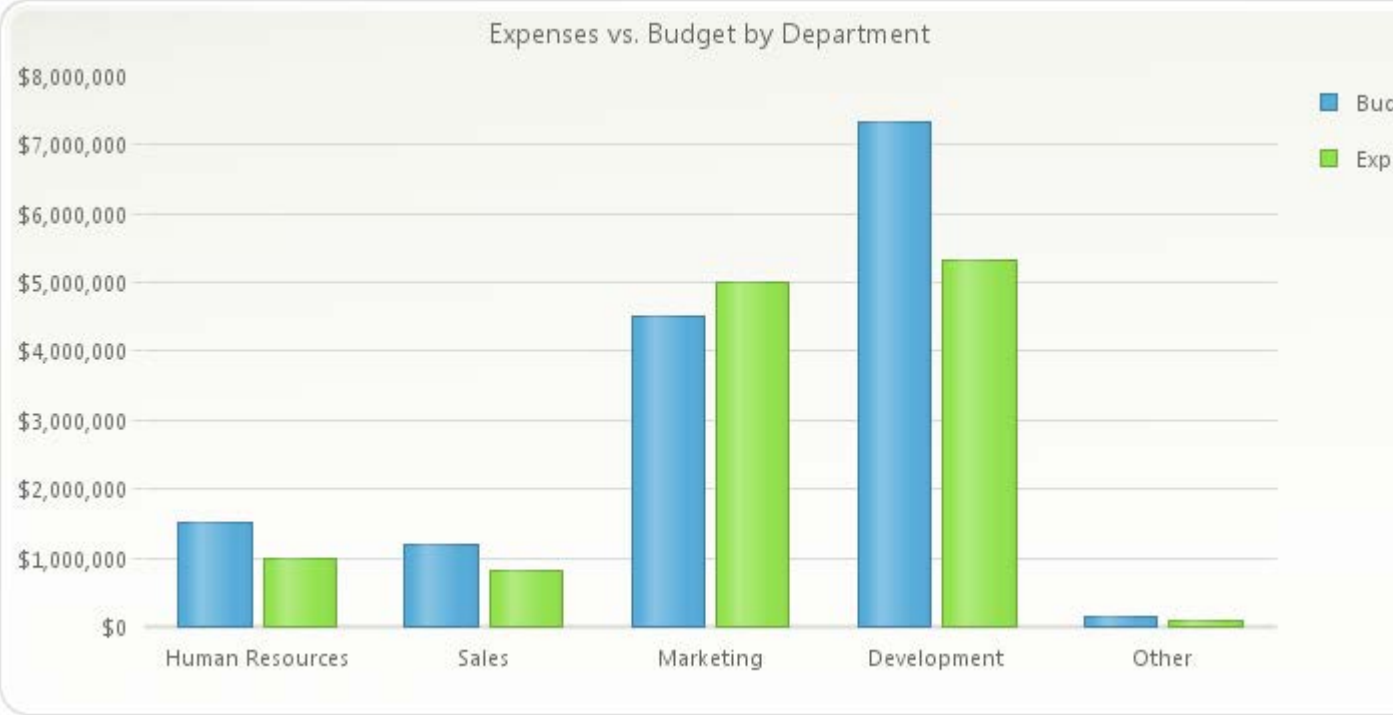
```
    {
            _expenses = value;
    }
 }
public static class Department
 {
    private String _name;
    private double _budget;
    private double _expenses;
```

```
    public Department(String name, double budget, double expenses)
    {
            _name = name;
            _budget = budget;
            _expenses = expenses;
    }

    public final String getName()
    {
            return _name;
    }
    public final void setName(String value)
    {
            _name = value;
    }

    public final double getBudget()
    {
            return _budget;
    }
    public final void setBudget(double value)
    {
            _budget = value;
```

```
    }


    public final double getExpenses()
    {
            return _expenses;
    }
    public final void setExpenses(double value)
    {
            _expenses = value;
    }
}
```

Now in this sample application, we have an array of business objects, say the Departments of the company for example, we would like to plot on a chart. Instead of iterating through the Chart's API value by value to do the necessary plot, you can simply databind the array Department objects itself as shown below.

Chart FX also supports reading data from a Business Object. In this case, we have created a class containing the data (Department) and used the ListProvider to populate Chart FX with multiple instances of the class.

- 


```
chart1.setGallery(Gallery.BAR);

chart1.getTitles().add(new TitleDockable("Expenses vs. Budget by Department"))
;

chart1.getAxisY().getLabelsFormat().setFormat(AxisFormat.CURRENCY);



Department[] departments = new Department[5];

departments[0] = new Department("Human Resources", 1500000, 1000000);

departments[1] = new Department("Sales", 1200000, 800000);

departments[2] = new Department("Marketing", 4500000, 5000000);

departments[3] = new Department("Development", 7320000, 5310000);

departments[4] = new Department("Other", 150000, 100000);



ListProvider lstProvider = new ListProvider(departments);

chart1.getDataSourceSettings().setDataSource(lstProvider);
```

```
public static class Department

 {

     private String _name;

     private double _budget;

     private double _expenses;



     public Department(String name, double budget, double expenses)

     {

              _name = name;

              _budget = budget;

              _expenses = expenses;

     }



     public final String getName()

     {

              return _name;

     }

     public final void setName(String value)

     {

              _name = value;

     }



     public final double getBudget()
```

```
    {
            return _budget;
    }
    public final void setBudget(double value)
    {
            _budget = value;
    }

    public final double getExpenses()
    {
            return _expenses;
    }
    public final void setExpenses(double value)
    {
            _expenses = value;
    }
}
```

Notice, we simply databind the object to the chart as we would any other data such as XML or DataSet. At this point however, you would need to use the DataField to specify the getFields which are important to your chart. If the Person object had many attributes, then this would be the way to indicate which attribute you are interested in displaying on the chart.

Passing Data via XML File

A chart can be populated via XML using the XmlDataProvider. The XML can be loaded as a string, as a file (using the file name) or as a stream. For the example below, we load a file called "data.xml". Here is what is inside the file:

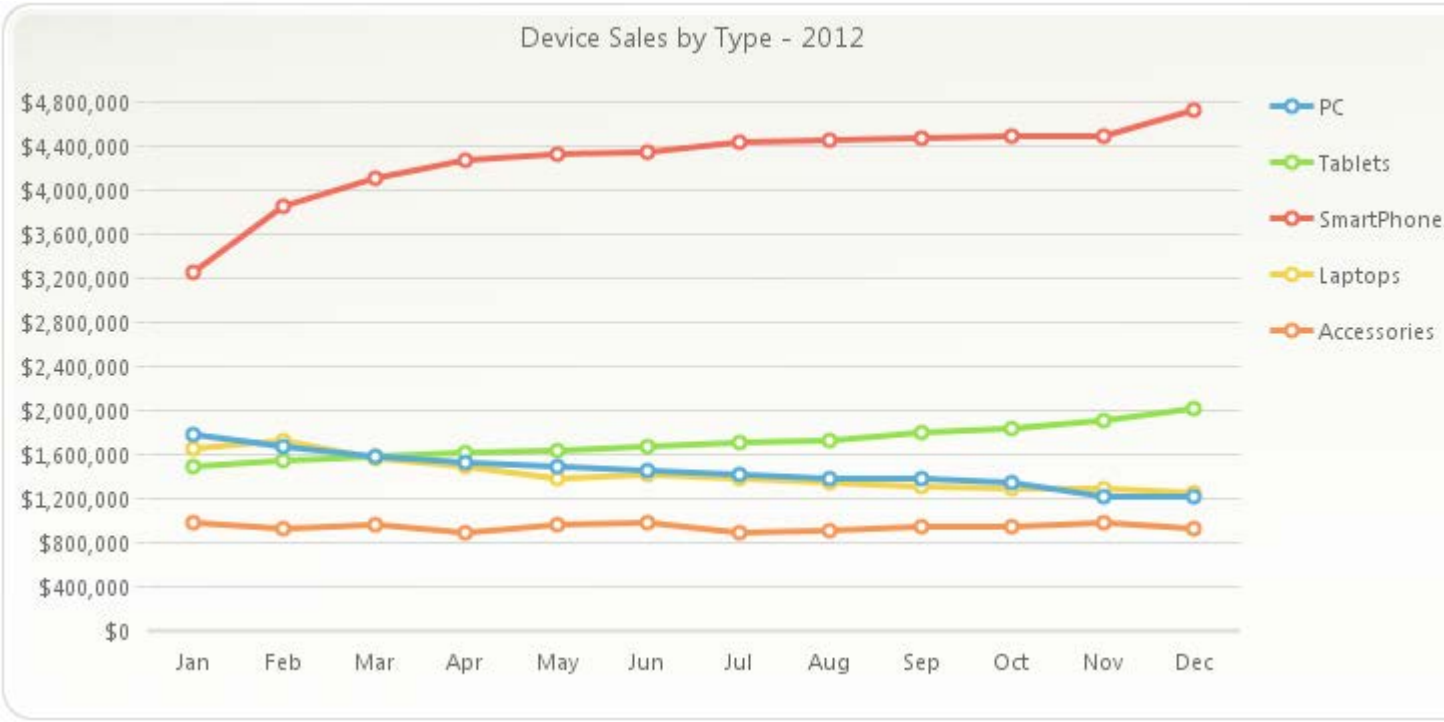
```
<?xml version="1.0"?>
<CHARTFX>
 <COLUMNS>
 <COLUMN NAME="Month" TYPE="String"/>
 <COLUMN NAME="PC" TYPE="Integer"/>
 <COLUMN NAME="Tablets" TYPE="Integer"/>
 <COLUMN NAME="SmartPhones" TYPE="Integer"/>
 <COLUMN NAME="Laptops" TYPE="Integer"/>
 <COLUMN NAME="Accessories" TYPE="Integer"/>
 </COLUMNS>
 <ROW Month="Jan" PC="1780000" Tablets="1490000" SmartPhones="3258000" Laptops="16
50000" Accessories="975000"></ROW>
 <ROW Month="Feb" PC="1670000" Tablets="1545000" SmartPhones="3855000" Laptops="17
25000" Accessories="923000"></ROW>
 <ROW Month="Mar" PC="1580000" Tablets="1586000" SmartPhones="4105000" Laptops="15
64000" Accessories="956000"></ROW>
 <ROW Month="Apr" PC="1520000" Tablets="1612000" SmartPhones="4275000" Laptops="14
98000" Accessories="890000"></ROW>
 <ROW Month="May" PC="1490000" Tablets="1643000" SmartPhones="4320000" Laptops="13
75000" Accessories="966000"></ROW>
 <ROW Month="Jun" PC="1450000" Tablets="1675000" SmartPhones="4338000" Laptops="14
23000" Accessories="983000"></ROW>
 <ROW Month="Jul" PC="1410000" Tablets="1701000" SmartPhones="4442000" Laptops="13
83000" Accessories="894000"></ROW>
 <ROW Month="Aug" PC="1390000" Tablets="1733000" SmartPhones="4455000" Laptops="13
43000" Accessories="905000"></ROW>
 <ROW Month="Sep" PC="1375000" Tablets="1794000" SmartPhones="4465000" Laptops="13
11000" Accessories="946000"></ROW>
 <ROW Month="Oct" PC="1348000" Tablets="1840000" SmartPhones="4483000" Laptops="12
98000" Accessories="951000"></ROW>
 <ROW Month="Nov" PC="1225000" Tablets="1901000" SmartPhones="4494000" Laptops="12
85000" Accessories="973000"></ROW>
 <ROW Month="Dec" PC="1210000" Tablets="2010000" SmartPhones="4725000" Laptops="12
56000" Accessories="933000"></ROW>
</CHARTFX>
```

Note each column has a name and a type. They will be converted in DataField correspondingly. The Date column can be either Date or String, depending whether you want to use it as data or just labels. More details on data driven axis can be found on the documentation of Axis.

The load method of the XmlDataProvider can be used by giving it the file name. We also set the format for our Date column so that it is interpreted correctly.

image

Java




```
chart1.setGallery(Gallery.LINES);

chart1.getTitles().add(new TitleDockable("Device Sales by Type - 2012"));

chart1.getAxisY().getLabelsFormat().setFormat(AxisFormat.CURRENCY);

XmlDataProvider xdp = new XmlDataProvider();

xdp.load(System.getProperty("user.dir") + "/data/ProductSales.xml");

xdp.setDateFormat("MMM d, yyyy");

chart1.getDataSourceSettings().setDataSource(xdp);

public static class ProductSales {


    public ProductSales(String month, double white, double red, double sparkling) {

        this.setMonth(month);

        this.setWhite(white);
```

```
        this.setRed(red);
        this.setSparkling(sparkling);
    }
    private String privateMonth;

    public final String getMonth() {
        return privateMonth;
    }

    public final void setMonth(String value) {
        privateMonth = value;
    }
    private double privateWhite;

    public final double getWhite() {
        return privateWhite;
    }

    public final void setWhite(double value) {
        privateWhite = value;
    }
    private double privateRed;
```

```
    public final double getRed() {
        return privateRed;
    }

    public final void setRed(double value) {
        privateRed = value;
    }
    private double privateSparkling;

    public final double getSparkling() {
        return privateSparkling;
    }

    public final void setSparkling(double value) {
        privateSparkling = value;
    }
}
```

```
Additional Required Files: Click to Download
```

Since we kept the DATE column as string it will be used for X labels instead of value data for all the series.

Note: This function reads from an external XML File using the load method of the Xml Data Provider. You need to add a reference to the Chart FX Data Providers assembly in order to use this mechanism.

Although the data contains volume information, for this chart we want to display open, high, low and close values only. Therefore we need to create and bind our DataField manually. We use the OpenHighLowClose enumeration to make sure each series gets the correct data. And we bind the DATE column to the X value of every series.

Java

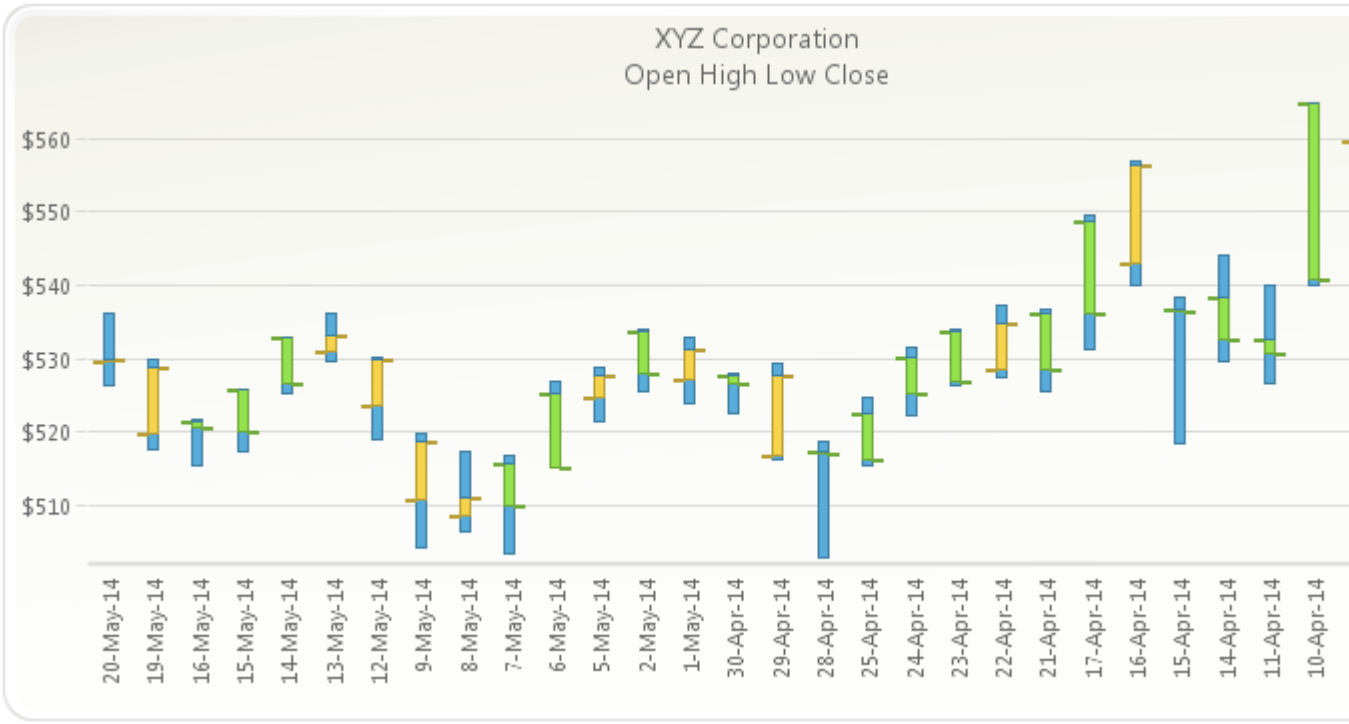
```
DateTimeDataField dfDate = new DateTimeDataField();

dfDate.setDataPath("DATE");

NumericDataField dfOpen = new NumericDataField();

dfOpen.setDataPath("OPEN");

NumericDataField dfHigh = new NumericDataField();

dfHigh.setDataPath("HIGH");

NumericDataField dfLow = new NumericDataField();

dfLow.setDataPath("LOW");

NumericDataField dfClose = new NumericDataField();

dfClose.setDataPath("CLOSE");



DataFieldCollection dataFieldCollection = chart1.getDataSourceSettings().getDa
taFields();

dataFieldCollection.add(dfDate);

dataFieldCollection.add(dfOpen);

dataFieldCollection.add(dfHigh);

dataFieldCollection.add(dfLow);

dataFieldCollection.add(dfClose);



chart1.getData().setSeries(4);
```

```
chart1.getSeries().get(0).getBindings().setY(dfLow);



List<SeriesAttributes> series = chart1.getSeries();

series.get((int)OpenHighLowClose.getUnderlyingValue(OpenHighLowClose.OPEN)).ge
tBindings().setY(dfOpen);

series.get((int)OpenHighLowClose.getUnderlyingValue(OpenHighLowClose.HIGH)).ge
tBindings().setY(dfHigh);

series.get((int)OpenHighLowClose.getUnderlyingValue(OpenHighLowClose.LOW)).get
Bindings().setY(dfLow);

series.get((int)OpenHighLowClose.getUnderlyingValue(OpenHighLowClose.CLOSE)).g
etBindings().setY(dfClose);



chart1.getAxisX().getBindings().setLabel(dfDate);
```

And here is the final product. Below using loadXml method of the XmlDataProvider for portability.

image

Java



Plain Text Files

Plain text files can be used to populate a chart. Chart FX supports both comma separated values (CSV) and tab separated values (TSV). Each column of the documented will be translated into a DataField, each row a value of the DataField (with exception of the first row, which will be used as DataField ID).

Here is an example of a TSV that could be used to populate a chart, followed by the code necessary and the chart created. Note it is necessary to call the readData method of the DataSourceSettings class before closing the TextProvider.

Temperature Pressure

-99.7 0.1

-94 0.13

-90.7 0.2

-81.7 0.3

-72.7 0.4

-63.7 0.6

-54.7 0.9

-45.9 1.33

-45.7 1.4

-36.7 2

-27.7 2.8

-18.7 3.8

-9.67 5.3

-8.14 5.33

-0.67 7.1

8.33 9.5

17.3 12.4

20.7 13.33

26.3 16.1

35.3 20.7

44.3 26.3

53.3 33

62.3 41.1

71.3 50.8

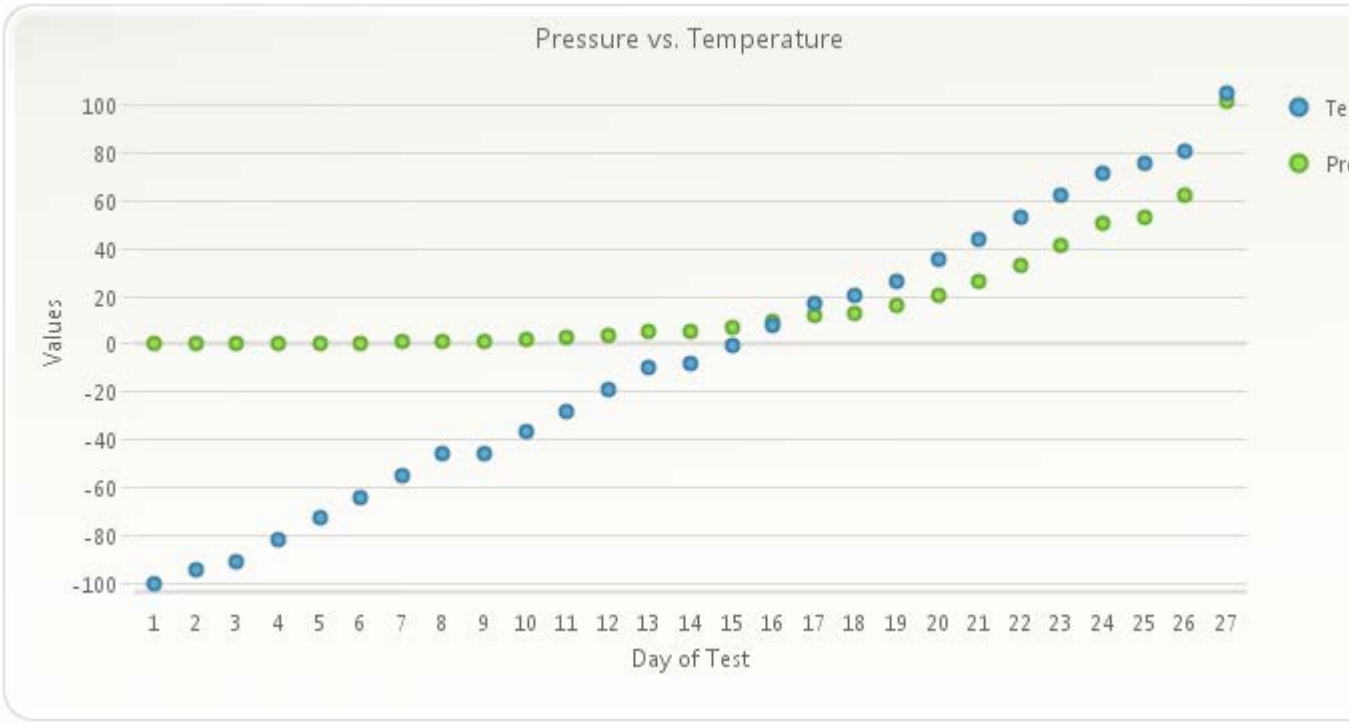
75.4 53.33

80.3 62.1

105 101.3

image

Java



Passing Data Through API

Data can be passed to the chart point per point, through the chart's API. Data works as a two dimensional array, the first index being for the series and the second for the point.

In order to use that API you should first use the setSeries and setPoints methods of DataValues, indicating how many of which will be passed to the chart. Then, use the DataValues class set method for different types of data: double, DateTime TimeSpan. The setters also take as parameters the index for the series and point you would like to set. There are three different getters, each returning different types: getDouble, getDateTime and getTimeSpan.

image

Java

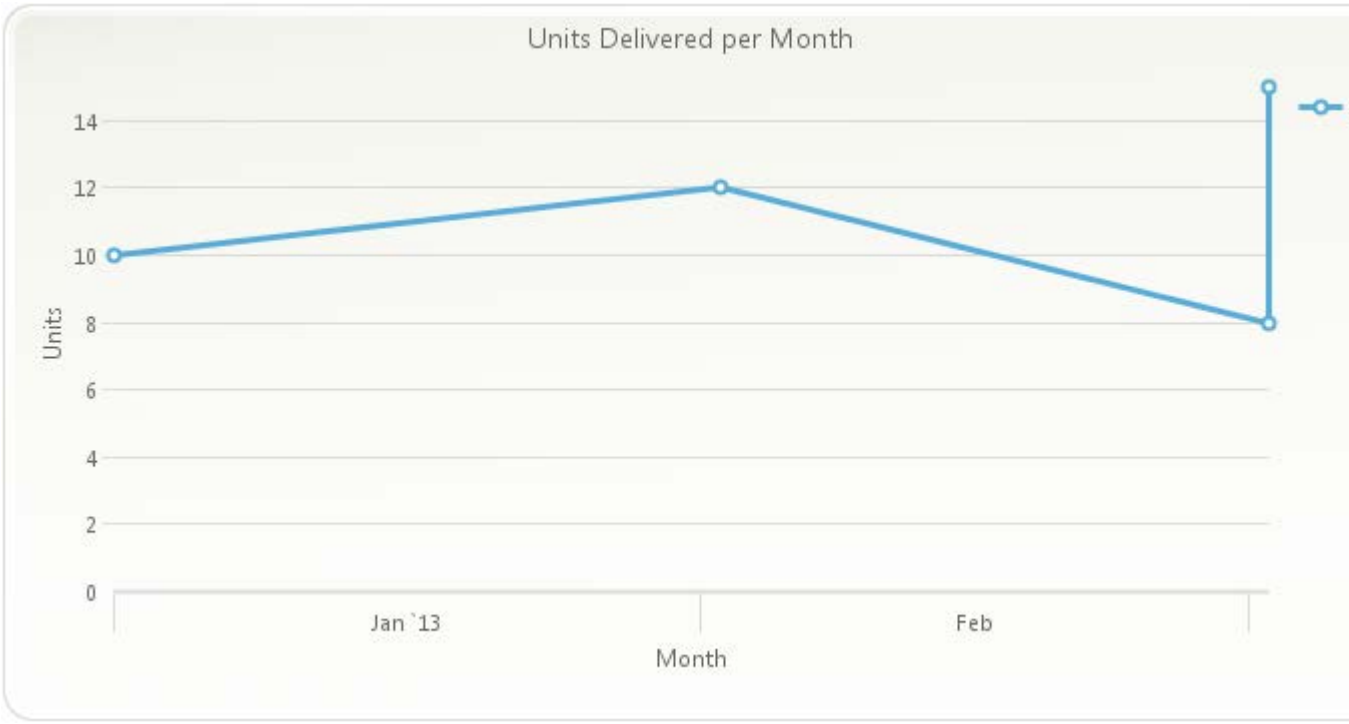

```
chart1.getTitles().add(new TitleDockable("Wine Sales by Type"));

chart1.getAxisY().getLabelsFormat().setFormat(AxisFormat.CURRENCY);

chart1.setGallery(Gallery.CURVE);

chart1.getData().setSeries(2);

chart1.getData().setPoints(12);

chart1.getSeries().get(0).setText("Red Wine");

chart1.getSeries().get(1).setText("White Wine");



DataValues dataValues = chart1.getData();

dataValues.set(0, 0, 12560);

dataValues.set(0, 1, 13400);

dataValues.set(0, 2, 16700);

dataValues.set(0, 3, 12000);

dataValues.set(0, 4, 15800);
```

```
dataValues.set(0, 5, 9800);

dataValues.set(0, 6, 17800);

dataValues.set(0, 7, 19800);

dataValues.set(0, 8, 23200);

dataValues.set(0, 9, 16700);

dataValues.set(0, 10, 11800);

dataValues.set(0, 11, 13400);

dataValues.set(1, 0, 21000);

dataValues.set(1, 1, 17000);

dataValues.set(1, 2, 19020);

dataValues.set(1, 3, 26500);

dataValues.set(1, 4, 27800);

dataValues.set(1, 5, 29820);

dataValues.set(1, 6, 17800);

dataValues.set(1, 7, 32000);

dataValues.set(1, 8, 26500);

dataValues.set(1, 9, 23000);

dataValues.set(1, 10, 23400);

dataValues.set(1, 11, 15400);
```

Here is an example of setting Date as data (using java.util.GregorianCalendar).

image

Java




```
chart1.getTitles().add(new TitleDockable("Units Delivered per Month"));



DataValues dataValues = chart1.getData();

dataValues.setSeries(1);

dataValues.setPoints(2);



IDataArray yDataValues = dataValues.getY();

yDataValues.set(0, 0, 10);

yDataValues.set(0, 1, 12);

yDataValues.set(0, 2, 8);

yDataValues.set(0, 3, 15);



IDataArray xDataValues = dataValues.getX();

xDataValues.set(0, 0, new java.util.GregorianCalendar(2013, 0, 1).getTime());
```

```
xDataValues.set(0, 1, new java.util.GregorianCalendar(2013, 1, 1).getTime());

xDataValues.set(0, 2, new java.util.GregorianCalendar(2013, 2, 1).getTime());

xDataValues.set(0, 3, new java.util.GregorianCalendar(2013, 2, 1).getTime());



chart1.getSeries().get(0).setText("Units Sold");

chart1.getAxisY().getTitle().setText("Units");



AxisX axisX = chart1.getAxisX();

axisX.getTitle().setText("Month");

axisX.getDataFormat().setFormat(AxisFormat.DATE_TIME);

axisX.getLabelsFormat().setFormat(AxisFormat.DATE_TIME);
```

Skipped points will take the value of null, and not be displayed. This can also be accomplished explicitly, as below.

image

Java


```
chart1.getTitles().add(new TitleDockable("Wine Sales by Type"));

chart1.getAxisY().getLabelsFormat().setFormat(AxisFormat.CURRENCY);

chart1.setGallery(Gallery.LINES);

List<SeriesAttributes> series = chart1.getSeries();

series.get(0).setText("Red Wine");

series.get(1).setText("White Wine");



DataValues dataValues = chart1.getData();

dataValues.setSeries(2);

dataValues.setPoints(12);

dataValues.set(0, 0, 12560);

dataValues.set(0, 1, 13400);

dataValues.set(0, 2, (DataUnit)null);

dataValues.set(0, 3, 12000);
```

```
dataValues.set(0, 4, 15800);

dataValues.set(0, 5, 9800);

dataValues.set(0, 6, 17800);

dataValues.set(0, 7, 19800);

dataValues.set(0, 8, (DataUnit)null);

dataValues.set(0, 9, 16700);

dataValues.set(0, 10, 11800);

dataValues.set(0, 11, 13400);

dataValues.set(1, 0, 21000);

dataValues.set(1, 1, (DataUnit)null);

dataValues.set(1, 2, 19020);

dataValues.set(1, 3, 26500);

dataValues.set(1, 4, 27800);

dataValues.set(1, 5, 29820);

dataValues.set(1, 6, 17800);

dataValues.set(1, 7, 32000);

dataValues.set(1, 8, (DataUnit)null);

dataValues.set(1, 9, 23000);

dataValues.set(1, 10, 23400);

dataValues.set(1, 11, 15400);
```

Note: Since setDouble() cannot take null values we use set() for those.

As is the case no matter how you pass data to the chart, hidden points can be interpolated to avoid the gap on the line.

image

Java


```
chart1.getTitles().add(new TitleDockable("Wine Sales by Type"));

chart1.getAxisY().getLabelsFormat().setFormat(AxisFormat.CURRENCY);

chart1.setGallery(Gallery.LINES);



List<SeriesAttributes> series = chart1.getSeries();

series.get(0).setText("Red Wine");

series.get(1).setText("White Wine");



DataValues dataValues = chart1.getData();

dataValues.setSeries(2);

dataValues.setPoints(12);

dataValues.set(0, 0, 12560);
```

```
dataValues.set(0, 1, 13400);

dataValues.set(0, 2, (DataUnit)null);

dataValues.set(0, 3, 12000);

dataValues.set(0, 4, 15800);

dataValues.set(0, 5, 9800);

dataValues.set(0, 6, 17800);

dataValues.set(0, 7, 19800);

dataValues.set(0, 8, (DataUnit)null);

dataValues.set(0, 9, 16700);

dataValues.set(0, 10, 11800);

dataValues.set(0, 11, 13400);

dataValues.set(1, 0, 21000);

dataValues.set(1, 1, (DataUnit)null);

dataValues.set(1, 2, 19020);

dataValues.set(1, 3, 26500);

dataValues.set(1, 4, 27800);

dataValues.set(1, 5, 29820);

dataValues.set(1, 6, 17800);

dataValues.set(1, 7, 32000);

dataValues.set(1, 8, (DataUnit)null);

dataValues.set(1, 9, 23000);

dataValues.set(1, 10, 23400);

dataValues.set(1, 11, 15400);
```

```
chart1.getData().setInterpolateHidden(true);
```

Data with X and Y values can also be plotted manually using the API. It is recommended that you set your X axis to Date or DateTime format when using date values as data for the X axis.

image

Java


```
chart1.getTitles().add(new TitleDockable("Wine Sales by Type"));

chart1.getAxisY().getLabelsFormat().setFormat(AxisFormat.CURRENCY);

chart1.setGallery(Gallery.LINES);

DataValues dataValues = chart1.getData();

dataValues.setSeries(2);

dataValues.setPoints(12);



IDataArray yDatavalues = dataValues.getY();
```

```
yDatavalues.set(0, 0, 12560);

yDatavalues.set(0, 1, 13400);

yDatavalues.set(0, 2, 16700);

yDatavalues.set(0, 3, 12000);

yDatavalues.set(0, 4, 15800);

yDatavalues.set(0, 5, 9800);

yDatavalues.set(0, 6, 17800);

yDatavalues.set(0, 7, 19800);

yDatavalues.set(0, 8, 23200);

yDatavalues.set(0, 9, 16700);

yDatavalues.set(0, 10, 11800);

yDatavalues.set(0, 11, 13400);

yDatavalues.set(1, 0, 21000);

yDatavalues.set(1, 1, 17000);

yDatavalues.set(1, 2, 19020);

yDatavalues.set(1, 3, 26500);

yDatavalues.set(1, 4, 27800);

yDatavalues.set(1, 5, 29820);

yDatavalues.set(1, 6, 17800);

yDatavalues.set(1, 7, 32000);

yDatavalues.set(1, 8, 26500);

yDatavalues.set(1, 9, 23000);

yDatavalues.set(1, 10, 23400);
```

```
yDatavalues.set(1, 11, 15400);



IDataArray xDataValues = dataValues.getX();

xDataValues.set(0, 0, (new java.util.GregorianCalendar(2013, 1, 1).getTime()))
;

xDataValues.set(0, 1, (new java.util.GregorianCalendar(2013, 2, 11).getTime())
);

xDataValues.set(0, 2, (new java.util.GregorianCalendar(2013, 3, 1).getTime()))
;

xDataValues.set(0, 3, (new java.util.GregorianCalendar(2013, 4, 1).getTime()))
;

xDataValues.set(0, 4, (new java.util.GregorianCalendar(2013, 5, 1).getTime()))
;

xDataValues.set(0, 5, (new java.util.GregorianCalendar(2013, 6, 1).getTime()))
;

xDataValues.set(0, 6, (new java.util.GregorianCalendar(2013, 7, 1).getTime()))
;

xDataValues.set(0, 7, (new java.util.GregorianCalendar(2013, 8, 1).getTime()))
;

xDataValues.set(0, 8, (new java.util.GregorianCalendar(2013, 9, 1).getTime()))
;

xDataValues.set(0, 9, (new java.util.GregorianCalendar(2013, 10, 1).getTime())
);

xDataValues.set(0, 10,(new java.util.GregorianCalendar(2013, 11, 1).getTime())
);

xDataValues.set(0, 11,(new java.util.GregorianCalendar(2013, 12, 1).getTime())
);

xDataValues.set(1, 0, (new java.util.GregorianCalendar(2013, 1, 1).getTime()))
;
```

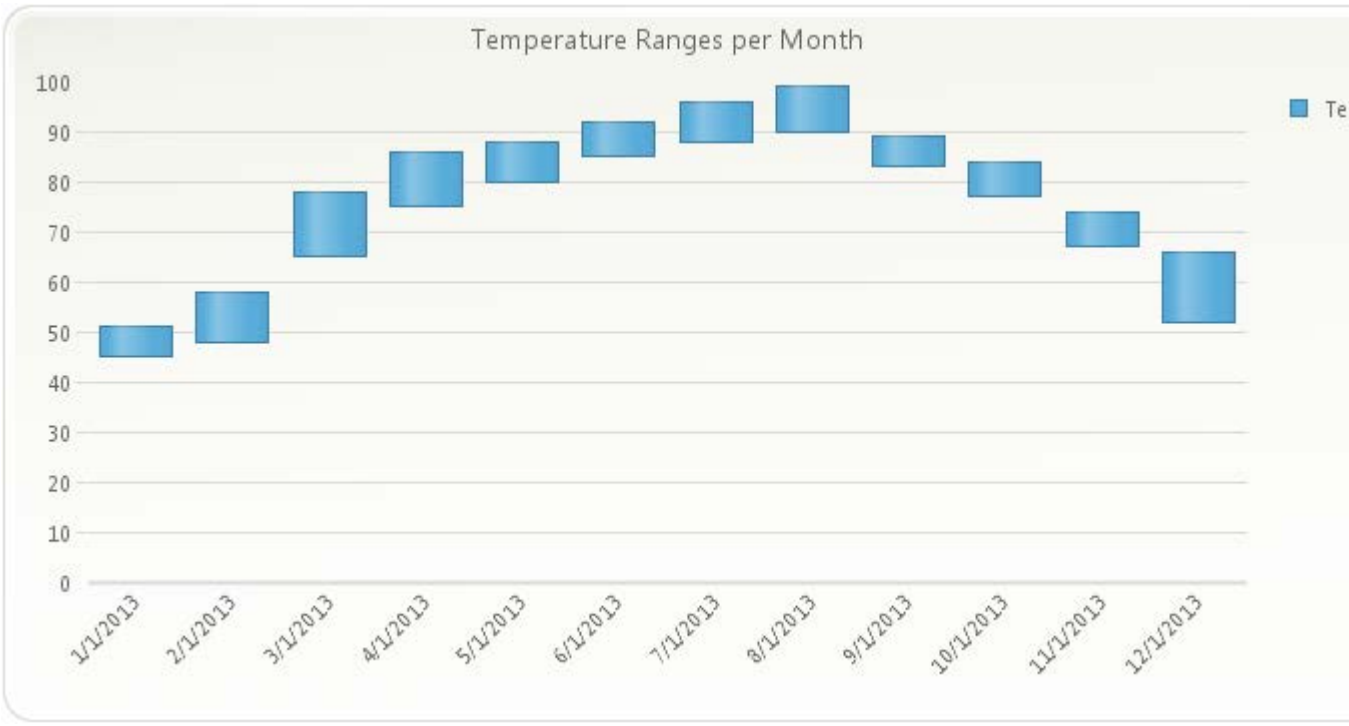
```
xDataValues.set(1, 1, (new java.util.GregorianCalendar(2013, 2, 11).getTime())
);

xDataValues.set(1, 2, (new java.util.GregorianCalendar(2013, 3, 1).getTime()))
;

xDataValues.set(1, 3, (new java.util.GregorianCalendar(2013, 4, 1).getTime()))
;

xDataValues.set(1, 4, (new java.util.GregorianCalendar(2013, 5, 1).getTime()))
;

xDataValues.set(1, 5, (new java.util.GregorianCalendar(2013, 6, 1).getTime()))
;

xDataValues.set(1, 6, (new java.util.GregorianCalendar(2013, 7, 1).getTime()))
;

xDataValues.set(1, 7, (new java.util.GregorianCalendar(2013, 8, 1).getTime()))
;

xDataValues.set(1, 8, (new java.util.GregorianCalendar(2013, 9, 1).getTime()))
;

xDataValues.set(1, 9, (new java.util.GregorianCalendar(2013, 10, 1).getTime())
);

xDataValues.set(1, 10,(new java.util.GregorianCalendar(2013, 11, 1).getTime())
);

xDataValues.set(1, 11,(new java.util.GregorianCalendar(2013, 12, 1).getTime())
);



List<SeriesAttributes> series = chart1.getSeries();

series.get(0).setText("Red Wine");

series.get(1).setText("White Wine");



AxisX axisX = chart1.getAxisX();

axisX.getDataFormat().setFormat(AxisFormat.DATE);
```

```
axisX.getLabelsFormat().setFormat(AxisFormat.DATE);

axisX.setLabelAngle((short)45);
```

There are also setters and getters for YFrom (also specific for Double, DateTime and TimeSpan), for bars with initial values.

image

Java



Real Time Charts

Chart FX has an API dedicated to real time data. There are two modes available: Loop and Scroll. The chart will scroll by default. Here is how to change to loop.

Java

```
chart1.getRealTime().setMode(RealTimeMode.LOOP);
```

While in loop mode you can add a loop marker as an indication of what was the last point updated.

Java

```
Line realTimeLoopMarker = chart1.getRealTime().getLoopMarker();

realTimeLoopMarker.setColor(new java.awt.Color(255,0,0,255));

realTimeLoopMarker.setWidth((short)3);

realTimeLoopMarker.setStyle(DashStyle.DASH);
```

There are some things to observe when using a real time chart. First, the chart will be rendered without any data at first, so you should set the Y axis ranges manually. It is also a good idea to have your buffer size the same as your points. Here is the basic setup for a real time chart.

Java

```
DataValues dataValues = chart1.getData();

dataValues.setSeries(1);

dataValues.setPoints(10);



chart1.getRealTime().setBufferSize(10);



AxisY axisY = chart1.getAxisY();

axisY.setMinDouble(0);

axisY.setMaxDouble(100);
```

To add data to a real time chart, you must invoke the beginAddData method, that takes the number of points (independently of series) and how the point should be added (inserted or appended). You then pass the points to the chart as you would manually, using the getData method. Finally, you must invoke the endAddData method, that takes two booleans as parameters: whether the labels should scroll with the points and whether the chart should scroll to new point if zoomed.

Here is a sample of inserting a point of value 10 to the chart. Note the index of the point added should be always 0.

Java

```
chart1.getRealTime().beginAddData(1, RealTimeAction.INSERT);

chart1.getData().setDouble(0, 0, 10);

chart1.getRealTime().endAddData(true, true);
```

This sample illlustrates the use of Chart FX to handle real-time scenarios. We simulate the activity of an Electrocardiogram (EKG) measuring a patient's heartbeat under different conditions. This sample shows the difference between scrolling and rotating when a new value is added to the chart.

image

Java

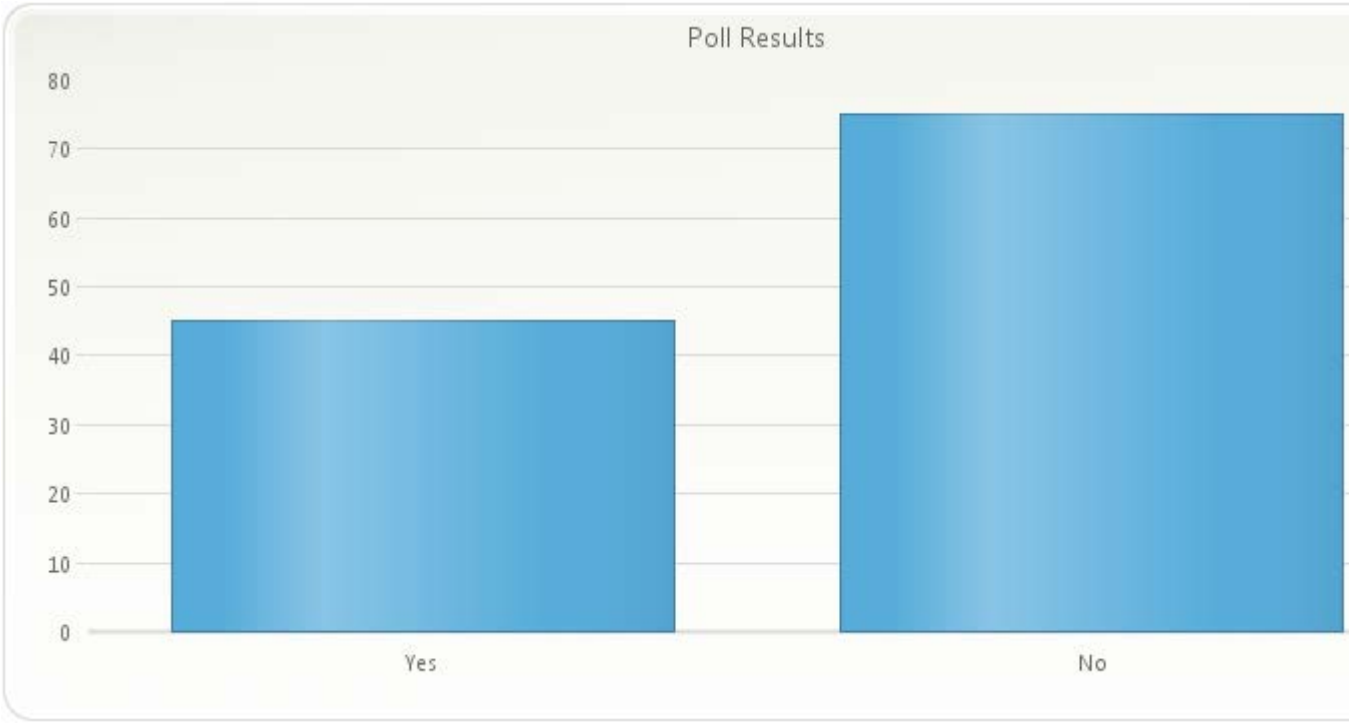
Series and Points

In order to render a chart, Chart FX must have the data organized in a meaningful way. Numerical data in Chart FX is organized in terms of series and points.

Providing data for simple charts is as simple as having a simple set of values to plot. For example, a bar chart representing the total yes/no votes of a poll is a single series with two values as shown below.

image

Java



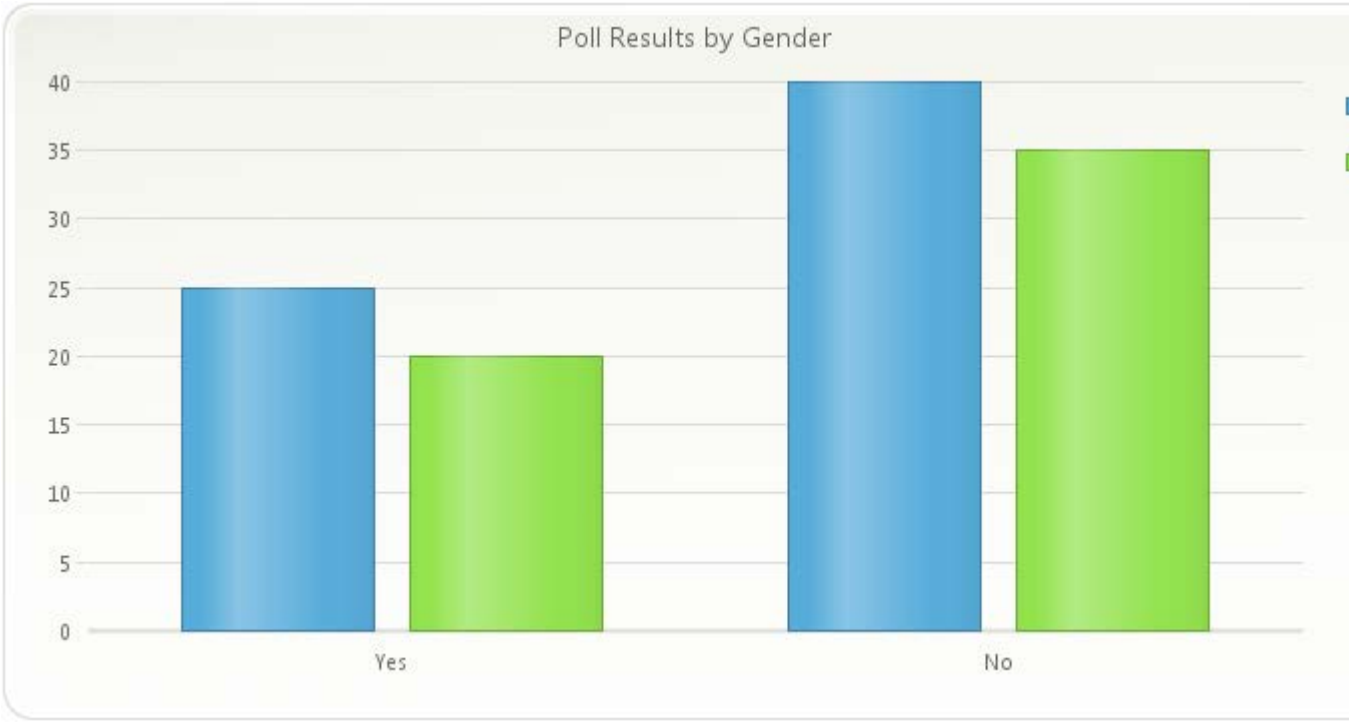

```
int nSeries;

nSeries = 1;

int nPoints;

nPoints = 2;

chart1.getData().setSeries(nSeries);

chart1.getData().setPoints(nPoints);

chart1.getData().set(0, 0, 45);

chart1.getData().set(0, 1, 75);

chart1.getAxisX().getLabels().set((short)0, "Yes");

chart1.getAxisX().getLabels().set((short)1, "No");
```

```
chart1.setGallery(Gallery.BAR);

chart1.getTitles().add(new TitleDockable("Poll Results"));

chart1.getLegendBox().setVisible(false);
```

The most basic chart contains one series with any number of points (in this case, two). This can be translated to a bar chart with one bar for each point. Many times however, data can be more complex and multidimensional. For example, we now receive the same polling data from the example above, but now broken down by gender. This would result now in a chart with two series and two points:

image

Java



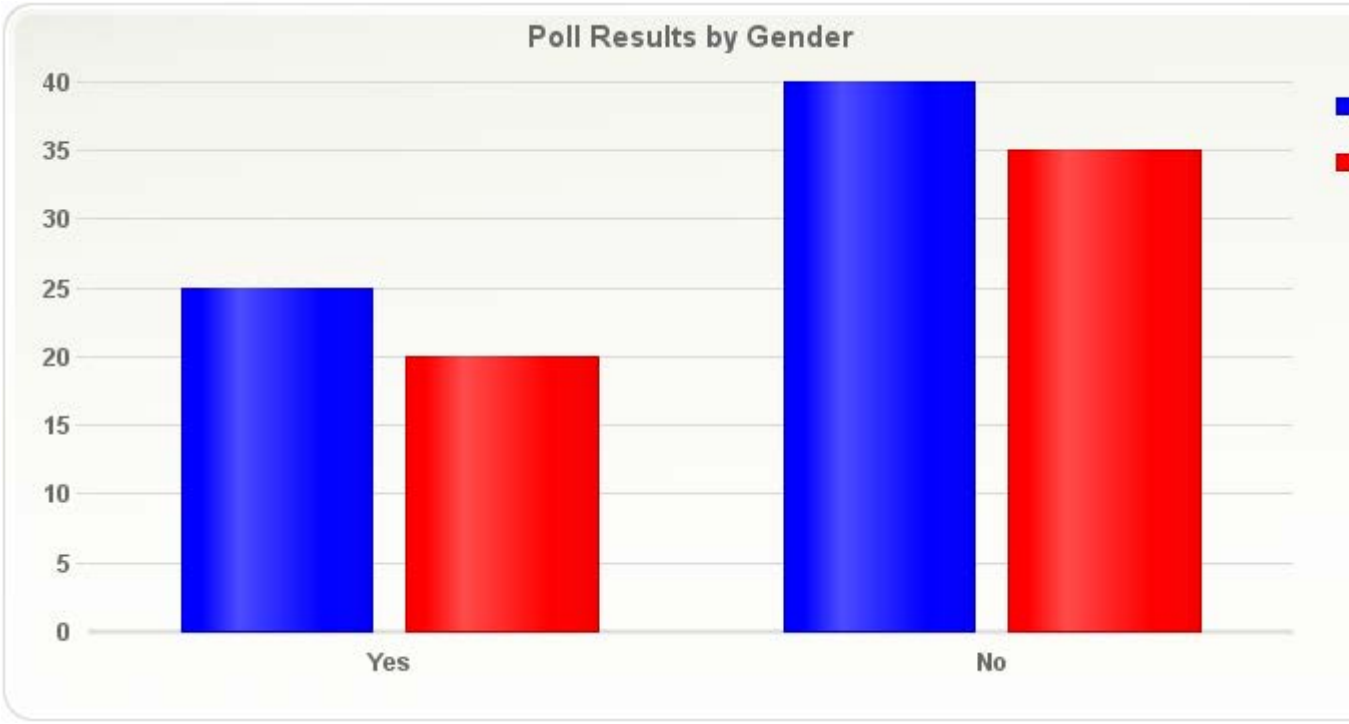

```
int nSeries;

nSeries = 2;

int nPoints;

nPoints = 2;

chart1.getData().setSeries(nSeries);

chart1.getData().setPoints(nPoints);
```

```
chart1.getData().set(0, 0, 25);

chart1.getData().set(0, 1, 40);

chart1.getData().set(1, 0, 20);

chart1.getData().set(1, 1, 35);

chart1.getSeries().get(0).setText("Female");

chart1.getSeries().get(1).setText("Male");

chart1.getAxisX().getLabels().set((short)0, "Yes");

chart1.getAxisX().getLabels().set((short)1, "No");

chart1.setGallery(Gallery.BAR);

chart1.getTitles().add(new TitleDockable("Poll Results by Gender"));
```

Additionally, a series can be distinguished by modifying the methods such as color, etc. To do this you would simply reference the index of the series you wish to modify using the indexer of the series collection.

image

Java



```
int nSeries = 2;

int nPoints = 2;



DataValues dataValues = chart1.getData();

dataValues.setSeries(nSeries);

dataValues.setPoints(nPoints);

dataValues.setDouble(0, 0, 25);

dataValues.setDouble(0, 1, 40);

dataValues.setDouble(1, 0, 20);

dataValues.setDouble(1, 1, 35);



List<SeriesAttributes> seriesAttributesList = chart1.getSeries();



SeriesAttributes seriesAttributes0 = seriesAttributesList.get(0);

seriesAttributes0.setText("Female");

seriesAttributes0.setColor(java.awt.Color.BLUE);



SeriesAttributes seriesAttributes1 = seriesAttributesList.get(1);

seriesAttributes1.setText("Male");

seriesAttributes1.setColor(java.awt.Color.RED);



List<String> axisXLabels = chart1.getAxisX().getLabels();

axisXLabels.set((short)0, "Yes");
```

```
axisXLabels.set((short)1, "No");



chart1.setGallery(Gallery.BAR);

chart1.getTitles().add(new TitleDockable("Poll Results by Gender"));
```

Unlike the simple bar chart example above, some gallery types require additional data for rendering. Bubble charts for example, aside from the regular X and Y values, require an additional dimension of data specifying the size of each bubble. This additional dimension is passed to the chart as a third series. An Open-High-Low-Close chart requires a total of four series to plot each point. In a pie chart, each series will be represented as a pie, with every point being represented as a slice. Using the concept of series Chart FX is able to capture the necessary data to produce charts containing multidimensional data.

Reordering Series

Chart FX allows series reordering. In some cases, depending on the gallery type being used, a series may be hidden by the values present in another series. Although there are multiple solutions to this scenario such as transparency or multiple panes, the better solution may be simply a reordering of the series.

CAUTION: Some charts require multiple series of data and require those series to be in the correct order. Reordering series in such cases may produce unexpected results.

There are various methods for reording the series in your chart such as the common seen sendToBack() and bringToFront() methods:

Java

```
chart1.getSeries().get(1).bringToFront();   /*moves the second series to the front
*/
```

Additionally, you can reorder the series by using the remove() and insert() methods. When removing a serie, the remaining series to the right (or series with a higher index number than the one removed) are shifted to the left and have their index number decreased accordingly. Below for example, we demonstrate how you can swap two series utilizing this method.

Java

```
/*this code swaps the fourth series with the third series*/

List<SeriesAttributes> seriesAttributes = chart1.getSeries();

SeriesAttributes series2 = seriesAttributes.get(2);
```

```
seriesAttributes.remove(series2);

seriesAttributes.add(3,series2);
```