ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA186

JAVAFX CHARTS: IMPLEMENTATION OF MISSING FEATURES

G.Kruk, O.Alves, L.Molinari, CERN, Geneva, Switzerland

Abstract

JavaFX, the GUI toolkit included in the standard JDK, provides charting components with commonly used chart types, a simple API and wide customization possibilities via CSS. Nevertheless, while the offered functionality is easy to use and of high quality, it lacks a number of features that are crucial for scientific or controls GUIs. Examples are the possibility to zoom and pan the chart content, superposition of different plot types, data annotations, decorations or a logarithmic axis. The standard charts also show performance limitations when exposed to large data sets or high update rates.

The article will describe how we have implemented the missing features and overcome the performance problems.

JAVAFX CHARTING PACKAGE

JavaFX is a software platform enabling creation of rich client applications. It includes a charting package with Pie Chart and a set of most commonly used XY charts such as Area Chart, Bar Chart, Line Chart or Scatter Chart. Each chart is represented by a class and can be styled using Cascading Style Sheets (CSS) and dedicated style classes [1], in a similar way as all the other JavaFX components.

Although the JavaFX charting package allows developers to create sophisticated and well-looking charts that are sufficient for typical business applications, it misses a number of built-in features that are essential for scientific tools used for data visualisation and analysis.

Missing Features

One missing piece of functionality is an easy, graphical way of zooming, i.e. by drawing a bounding box with mouse cursor. For charts containing a significant number of points, it is usually a *must have* feature that users just expect to be there.

Also, JavaFX does not provide an easy way for developers to add custom graphical elements on top of their charts. Examples are data point annotations, lines and rectangles defining limits, text labels or other decorations that help in interpretation and enrich the displayed data.

Other missing features that are commonly used in the controls domain include logarithmic scales, the possibility of mixing different plot types on a single chart or a heatmap chart used to display particle beam images.

Most of the aforementioned features are present in the list of possible enhancements for OpenJFX [2] but up to and including Java 10 there are no plans of the JavaFX team to actually implement them.

CERN CHART EXTENSIONS

Motivation

Lacking the required functionality, we faced a choice between either implementing it ourselves or using a 3rd party library that would support it.

As of today, the only suitable open source library is *JFreeChart* [3] in combination with *FXGraphics2D* [4]. The rich set of functionality offered by *JFreeChart* comes with a relatively steep learning curve for its API. Most of our non-professional software developers (like physicists or operators) preferred to use the standard JavaFX charting package and simpler APIs that they already knew. We wanted to avoid a hybrid approach, with JavaFX charting used for some applications and *JFreeChart* for others, as it would make long-term support and maintenance of these applications more difficult and expensive. Since the estimated development effort was not very high, we decided to implement it, with an idea of making it open sourced and available for the JavaFX community.

XYChartPane

The central class of the package is *XYChartPane*. In several aspects, it is similar to the standard JavaFX *StackPane*, which lays out its children in a back-to-front stack, but is specialized to lay out instances of *XYChart* and to manage nodes belonging to custom chart plugins (see next section for details).

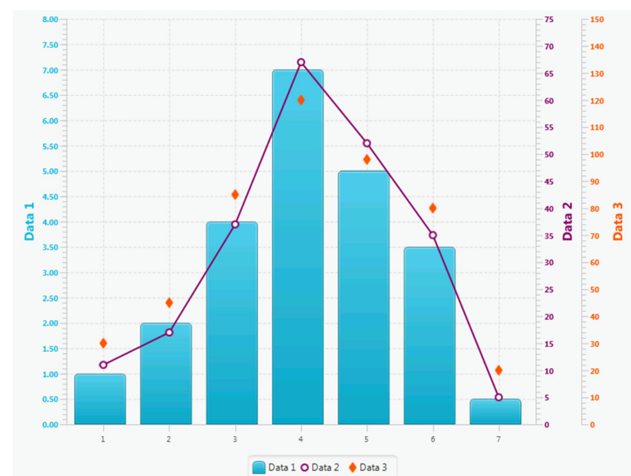

Figure 1: XYChartPane with different chart types.

The main (base) chart must be specified at the construction of *XYChartPane* but the additional charts, drawn on top of each other (see Fig. 1), can be added and removed at any moment via exposed observable list of overlay charts.

All the overlay charts use the X-axis of the base chart. They may also share the common Y-axis of the base chart

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA186

or have their own independent ones with distinct data ranges and drawn on either left or right side of the chart.

To make the overlay layout possible, certain features of charts added to the *XYChartPane* must be deactivated. For instance, the background is made transparent, the grid lines and individual legends are hidden as well as title labels. Some of these properties are configurable only on the base chart e.g. background or grid, while the others, such as title or legend, are configurable for all charts together using properties of the *XYChartPane*.

Chart Plugins

Chart plugins, instances of *XYChartPlugin* class, represent add-ons to the standard charts that can be added to the *XYChartPane* to interact with the chart content and/or to insert graphical components that are rendered on top of charts.

At the moment the extension provides the following plugins:

- ***ChartOverlay*** - allows adding to the chart area any instance of *javax.scene.Node* that will be laid out on top of charts. Typically the plugin would be given an instance of *javax.scene.layout.Pane* (e.g. *AnchorPane*) containing child nodes at arbitrary locations.
- ***CrosshairIndicator*** – a cross (horizontal and vertical line) following the mouse cursor and displaying its coordinates.
- ***DataPointTooltip*** – a tooltip label displaying coordinates of the data point hovered by the mouse cursor.
- ***Zoomer*** – zooms the visible chart area to the rectangle drawn by dragging the cursor. It keeps a stack of zoom windows (X and Y ranges), allowing zoom-out to the previous or to the origin window.
- ***Panner*** – allows dragging the visible chart area.
- ***XValueIndicator, YValueIndicator*** – a vertical and horizontal line (respectively), indicating specified X or Y value, with an optional text label describing the indicated value.
- ***XRangeIndicator, YRangeIndicator*** – a rectangle indicating vertical or horizontal range (respectively) between specified X or Y values, with an optional text label describing the indicated range.

The plugin can access the *XYChartPane* via a dedicated observable property (set when the plugin is added to the pane) and therefore it can register mouse or keyboard event handlers that it should react on. For instance, the *Zoomer* plugin registers a handler intercepting mouse events to draw zoom-in rectangles and to change the X and Y ranges at the end of the interaction.

Every plugin can provide an optional list of nodes that should be rendered on the *XYChartPane*, e.g. the *Zoomer* provides a *javax.scene.shape.Rectangle* node that is used to draw zooming box.

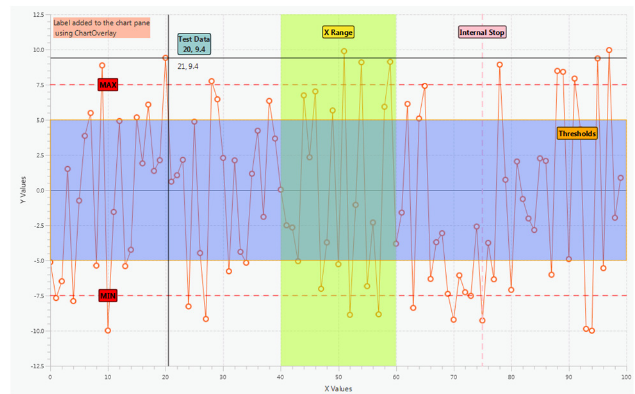

Figure 2: *XYChartPane* with components of different plugins.

Most of the plugins are configurable via dedicated properties. Also all the graphical elements used by plugins (lines, rectangles, labels, etc.) have their own style classes and therefore can be customized in a standard way using CSS.

NumericAxis

Along with *CategoryAxis*, JavaFX provides *NumberAxis* class that is used to display numerical values such as *Long*, *Double* or *BigDecimal*. It contains a boolean property called *autoRanging* that determines the behaviour of the class. With *autoRanging* set to true, the axis automatically calculates visible range and tick units based on data. If the property is set to false, the axis relies on the range and tick unit specified programmatically by the user.

This behaviour is incompatible with the requirements of *Zoomer* and *Panner* plugins that need to change the axis range programmatically but at the same time require the axis to adjust the tick unit automatically. The *NumberAxis* class being final prevents anyone from extending it and adding zooming and panning capabilities.

For this reason we implemented a *NumericAxis* class providing all the original functionality and in addition supporting automatic tick unit calculation with auto-range being off. Therefore it is necessary to use *NumericAxis* (rather than *NumberAxis*) for the *Zoomer* and *Panner* plugins to work properly.

In addition, the *NumericAxis* offers few other features:

- *autoRangePadding* – a double property representing a fraction of the range to be applied as padding on both sides of the axis range e.g. if set to 0.1 (10%) on axis with data range <10, 20>, the new automatically calculated range will be <9, 21>.
- *autoRangeRounding* – a boolean property, indicating if the automatically calculated range should be extended to the major tick unit value e.g. with data range <3, 74> and major tick unit 5, the range will be extended to <0, 75>.
- *tickUnitSupplier* – a property holding a strategy (*TickUnitSupplier*) responsible for calculation of major tick units, with default implementation equivalent to the one of *NumberAxis*.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA186

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

LogarithmicAxis

In addition to the *NumericAxis*, we also implemented *LogarithmicAxis*, with configurable logarithm base and the number of minor ticks. An example is presented on Fig. 3.

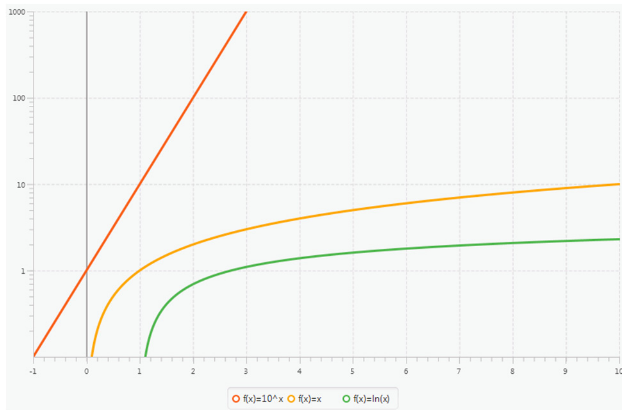

Figure 3: Logarithmic Y axis with base 10.

Similarly to the *NumericAxis*, it also supports the zooming and panning operations.

HeatMapChart

HeatMapChart is a specialised chart that uses colours to represent data values contained in a matrix. At CERN, it is typically used by applications displaying beam images (like on Fig. 4) or time trends of signals, with a single line of pixels representing a single acquisition.

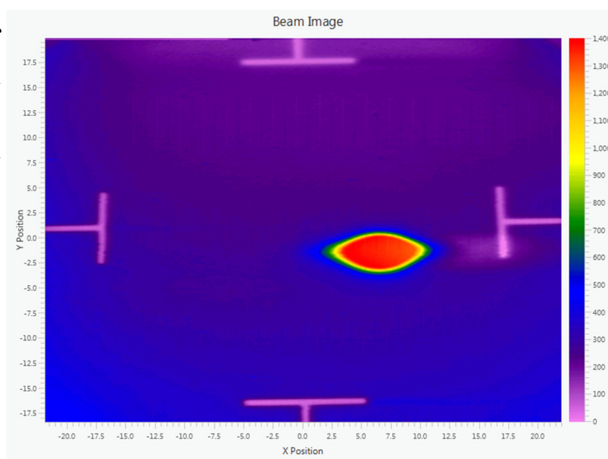

Figure 4: *HeatMapChart* showing the LHC beam image.

The class uses its own *Data* type that represents the matrix and allows retrieving X-Y coordinates and the corresponding values.

By default, the *HeatMapChart* uses a *rainbow* colours gradient but this can be changed using *colorGradient* property to either one of the predefined gradients (e.g. white-black, black-white, sunrise) or to a custom gradient specified by the user.

DEALING WITH LARGE DATA SETS

In scientific applications, it is a relatively common requirement to plot signals containing large number of data points. Some instruments at CERN measuring particle beam properties produce waveforms with more than 100,000 data points. Physicists and equipment specialists need to plot the entire waveform and to zoom-in to specific segments.

The JavaFX charting package performs well with series containing up to a few thousands data points, with rendering time below one second (on a decent desktop computer). However, drawing series containing tens of thousands points takes several seconds, blocking the FX thread and making the application unresponsive.

To address this issue, we developed the *DataReducingObservableList,* a specialised implementation of the *ObservableList* interface (as used by the *XYChart Series* class), performing data reduction the specified number of most significant points. It is a wrapper over a source list (containing all points) that triggers execution of the reduction algorithm on every change of the source data or X-axis range, exposing to the chart *Series* reduced number of points from the current X range.

By default, *DataReducingObservableList* uses *Ramer-Douglas-Peucker* [5] reduction algorithm that is fast and suitable for the vast majority of cases. The reduction of 100,000 points to 500 takes around 50-60ms, preserving the original shape of the signal. It is also possible to use another algorithm by providing its implementation and configuring it via a dedicated property.

CONCLUSION

The implemented extension fulfils the substantial set of features missing in the JavaFX charting package, enabling its usage for all controls applications.

All added components follow JavaFX design principles and API style, making their usage simple and intuitive.

The application of data reduction algorithm addresses the performance issues, allowing visualisation of large data sets flawlessly.

Currently we are in the process of making the extension open sourced.

REFERENCES

[1] CSS Reference, https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html#charts
[2] OpenJDK issue tracker, https://bugs.openjdk.java.net
[3] JFreeChart, http://www.jfree.org/jfreechart
[4] FXGraphics2D, http://www.jfree.org/fxgraphics2d
[5] Ramer-Douglas-Peucker algorithm, https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm