Graph Plugin Tech Spec

[I. Plugin Overview 2](#_Toc1538822873)

[A. Introduction 2](#_Toc279628844)

[B. Chart.js 2](#_Toc1039940415)

[C. chartjs-plugin-datalabels 2](#_Toc809690989)

[D. chartjs-plugin-dragdata 3](#_Toc2035847272)

[II. Functions of Graph Component 3](#_Toc2116182098)

[A. BuildChartData 3](#_Toc1766496391)

[B. ScaleLabelOptionsChange 4](#_Toc589290994)

[C. Axis Label Size 5](#_Toc692431702)

[D. Axis Line Size 6](#_Toc719270114)

[E. Graph Event Handler 7](#_Toc769410415)

[F. Snap To Grid Point 7](#_Toc1845549218)

[G. Extended Line 8](#_Toc29960962)

[H. Point Rotation 11](#_Toc649789473)

[I. Find Axis 12](#_Toc162749235)

[J. Axis line scale labels 13](#_Toc2004078406)

[K. Math Logic for calculating the upper and lower points in line items 15](#_Toc282717707)

[III. Graph Orientation in UI 16](#_Toc1347625129)

[A. Titles & Labels Layout 16](#_Toc1719563452)

# Plugin Overview

## Introduction

* + - This Graph plugin is used for line items correct response. Straight line, Single Line, and Placing Point items.
    - We are using three different plugins to form a Graph component, that provides the UI for the user to draw lines and points.

## Chart.js

* We are using Chart.js as plugin to draw our graph with 3.8.0 as version. We are drawing the basic graph using the plugin by providing the respective parameters as input for the canvas to draw the graph.

import Chart from 'chart.js/auto';

chart = new Chart(ctx.current, { type: 'scatter', data: chartData, options: scatterOptions });

<div style={styles}>

<canvas id={ctx?.current?.id || uuid()} ref={ctx} />

</div>

* Reference link for the Chart.js document

<https://www.chartjs.org/docs/latest/getting-started/installation.html>

## chartjs-plugin-datalabels

* We are using chartjs-plugin-datalabels, to show the axis labels and other datas while customizing the graph. Currently using chartjs-plugin-datalabels with 2.0.0.

**Configuration of Chartjs plugin datalabels in Graph Component.**

import ChartDataLabels from 'chartjs-plugin-datalabels';

Chart.register(ChartDataLabels);

## chartjs-plugin-dragdata

* To Enable the drag functionality for the plotted points we are using chartjs-plugin-dragdata version 2.2.4 with our graph component.

**Configuration of Chartjs Drag Functionality in Graph Component.**

import 'chartjs-plugin-dragdata';

* With the latest version of chart.js we do not need external configuration for the drag data.
* By using this plugin, we can drag the plotted points and lines around the graph.

# Functions of Graph Component

## BuildChartData

* This function is completely responsible for the building the graph with data, it helps to prepare chart with the graph settings, template and saved values.
* To Utilize this function we need to pass all the below mentioned parameters to build the chart data.

/\*

\* @param {object} - data - saved graph data

\* @param {boolean} - showLine - determine whether we need to connect the points as a line, or not

\* @param {number} - limit - max lines/points

\* @param {object} - content - item related data

\* @param {boolean} - drawLine - manually draw the line with slope and a point

\* @param {boolean} - extendedLines - add extended line for the line

\* @param {boolean} - native - represent original or duplciate graph

\*/

const buildChartData = (data, showLine, limit, content, incrementLabel, drawLine, extendedLines, native) => {

.......

.......

}

## ScaleLabelOptionsChange

* To show the Axis labels for the axis line we are using the below function based of Incremental label drop-down value.

const scaleLabelOptionsChange = (scale, bottomLeft, upperRight, xAxisScale, yAxisScale, chartData, incrementLabel) => {

if (scale === 0 || scale === 1) {

for (let ind = 0; ind >= (scale === 1 ? parseFloat(bottomLeft[0]) : parseFloat(bottomLeft[1])); ind = parseInt(ind) - (scale === 0 ? yAxisScale : xAxisScale)) {

if (incrementLabel && ((incrementLabel.code === "ALL" || incrementLabel.code === "NONE" || incrementLabel.code === "SELECT") ||

(incrementLabel.code === "LEAST\_AND\_GREAT" && (parseFloat(upperRight[scale]) === ind)))) {

chartData[scale].data.push({ x: (scale === 0 ? 0 : ind), y: (scale === 0 ? ind : 0) });

}

};

chartData[scale].data.reverse();

for (let ind = 0; ind <= (scale === 1 ? parseFloat(upperRight[0]) : parseFloat(upperRight[1])); ind = parseInt(ind) + (scale === 0 ? yAxisScale : xAxisScale)) {

if (incrementLabel && ((incrementLabel.code === "ALL" || incrementLabel.code === "NONE" || incrementLabel.code === "SELECT") ||

(incrementLabel.code === "LEAST\_AND\_GREAT" && (parseFloat(bottomLeft[scale]) === ind)))) {

chartData[scale].data.push({ x: (scale === 0 ? 0 : ind), y: (scale === 0 ? ind : 0) });

}

};

// Include the axes end points to the list if they are missing while scaling.

if (scale === 0) {

// Y Axis scale

if (undefined === chartData[scale].data.find(point => point.x === 0 && point.y === parseInt(upperRight[1]))) {

chartData[scale].data.push({ x: 0, y: parseInt(upperRight[1]) });

}

if (undefined === chartData[scale].data.find(point => point.x === 0 && point.y === parseInt(bottomLeft[1]))) {

chartData[scale].data.push({ x: 0, y: parseInt(bottomLeft[1]) });

}

} else if (scale === 1) {

// X Axis scale

if (undefined === chartData[scale].data.find(point => point.x === parseInt(upperRight[0]) && point.y === 0)) {

chartData[scale].data.push({ x: parseInt(upperRight[0]), y: 0 });

}

if (undefined === chartData[scale].data.find(point => point.x === parseInt(bottomLeft[0]) && point.y === 0)) {

chartData[scale].data.push({ x: parseInt(bottomLeft[0]), y: 0 });

}

}

}

}

## Axis Label Size

* The user given value to update the Axis label size will be given while configuring the axis line and label data of the graph.

chartData[scale].datalabels = {

align: (data) => {

const value = data.dataset.data[data.dataIndex];

let cursorAt = (value.y && value.y !== 0 ? value.y : null) || (value.x && value.x !== 0 ? value.x : null);

let position = 130;

if (cursorAt !== null) {

position = (scale === 0 ? ((cursorAt > 0) ? 180 : -180) : ((cursorAt > 0) ? 90 : 110));

}

return position;

},

offset: 4,

font: {

size: content?.axisLabelSize || 10

},

color: 'black',

}

## Axis Line Size

* Axis line size helps to configure the line size of the axis line inside the graph. We will be configuring this based on the quadrants in the graph.
* Configuring the AxisLine size inside the buildchart data function.

const buildChartData = (data, showLine, limit, content, incrementLabel, drawLine, extendedLines, native) => {

chartData[scale].borderWidth = content?.axisLineSize;

}

* To Configure the Axis line based on the quadrant, need to configure inside the line width of grid lines, which we defined for each Quadrants as X, Y, X1 and Y1.

grid: { //gridLines

lineWidth: function (context) {

if (context.tick.value > 0) {

return (content?.showGrid ? 1 : 0);

} else if (context.tick.value < 0) {

return (content?.showGrid ? 1 : 0);

}

return (content?.showGrid ? content?.axisLineSize : 0);

},

drawTicks: false,

…...

}

## Graph Event Handler

* This Function is responsible for the events in graph. Click event , Drag event, On Mouse hover event of the Graph UI.

const graphEventHandler = (e, element, isDragging) => {

.............

.........

..........

}

* Using this function we will be capturing the co-ordinates of the plotting points and draw a point in green color as choosed by user.

## Snap To Grid Point

* To Achieve the snap to grid functionality for our graph we are using the below function to get the nearest snap to grid point of x/y position from the chart.
* We need to pass the chartData and Boolean value of whether it is X axis or Y Axis and the point to be plotted.

const snapToGridPoint = (chartData, isXAxis, point) => {

let closest = point;

if (chartData?.datasets && chartData?.datasets[0]?.data?.length > 0 && chartData?.datasets[1]?.data?.length > 0) {

const xy = [...chartData?.datasets[0].data, ...chartData?.datasets[1]?.data];

let data = [];

if (isXAxis) {

data = xy.filter(d => d.x !== 0).map(d => d.x);

} else {

data = xy.filter(d => d.y !== 0).map(d => d.y);

}

if (data?.length > 0) {

data.push(0);

const reduced = data.reduce(function (prev, curr) {

return (Math.abs(curr - point) < Math.abs(prev - point) ? curr : prev);

});

if (reduced !== null && Number.isNaN(Number(reduced)) === false) {

closest = reduced;

}

}

}

return closest;

}

## Extended Line

* This function is helpful for the user to extend the line upto the end of the axis. This functionality is applicable only for Straight Line and Single Line item.

const buildChartData = (data, showLine, limit, content, incrementLabel, drawLine, extendedLines, native) => {

.......

if (extendedLines) {

const ePoints = infiniteLine(pointA, pointB, bottomLeft, upperRight);

points = ePoints ? ePoints : points;

if (points?.length === 4) {

dummyPoints = true;

}

}

.......

}

* The Infiniteline is a helper function to find the end of axis and denote it as dummypoint with Arrow marks.
* We are using the mathematical formulas for calculating the boundary of the graph.
* It identifies the graph boundary and will calculate the dummy points to draw arrowheads. Based on the points given end point will be calculated.

const infiniteLine = (pointA, pointB, bottomLeft, upperRight) => {

let points = null;

if (pointA && pointB) {

let finiteA = { dummyPoint: true }, finiteB = { dummyPoint: true };

let pointAB = Math.sqrt(Math.abs(Math.pow(pointA.x - pointB.x, 2.0) - Math.pow(pointA.y - pointB.y, 2.0)));

if (isNaN(pointAB) === false) {

if (pointAB === 0) {

pointAB = 1;

}

//Calculating Extended line points based on end point of graph

let midPoint = null; //To find mid point of selected two points

let extendPoint = null; //Actual Mid point which used calculating positive / negative points

let checkData = null;//line equation calculation

//Assiging the value to variable if the points not equal

if (pointB.x !== pointA.x && pointA.y !== pointB.y) {

extendPoint = (pointB.y - pointA.y) / (pointB.x - pointA.x);

midPoint = extendPoint;

checkData = pointA.y - (extendPoint \* (pointA.x));

}

// calculating angle is 0 to 90 and 180 to 270

if (midPoint > 0 && pointA.x !== pointB.x && pointA.y !== pointB.y) {

let y = (extendPoint \* (upperRight[0])) + checkData;

if (y <= upperRight[1]) {

finiteA.x = upperRight[0];

finiteA.y = y;

} else {

let x = (upperRight[1] - checkData) / extendPoint;

if (x <= upperRight[0]) {

finiteA.x = x;

finiteA.y = upperRight[1];

}

}

let y1 = (extendPoint \* (bottomLeft[0])) + checkData;

if (y1 >= bottomLeft[1]) {

finiteB.x = bottomLeft[0];

finiteB.y = y1;

} else {

let x1 = (bottomLeft[1] - checkData) / extendPoint;

if (x1 >= bottomLeft[0]) {

finiteB.x = x1;

finiteB.y = bottomLeft[1];

}

}

} else if (midPoint < 0 && pointA.x !== pointB.x && pointA.y !== pointB.y) { // calculating angle is 90 to 180 and 270 to 360

let y = (extendPoint \* (bottomLeft[0])) + checkData;

if (y <= upperRight[1]) {

finiteA.x = bottomLeft[0];

finiteA.y = y;

} else {

let x = (upperRight[1] - checkData) / extendPoint;

if (x >= bottomLeft[0]) {

finiteA.x = x;

finiteA.y = upperRight[1];

}

}

let y1 = (extendPoint \* (upperRight[0])) + checkData;

if (y1 >= bottomLeft[1]) {

finiteB.x = upperRight[0];

finiteB.y = y1;

} else {

let x1 = (bottomLeft[1] - checkData) / extendPoint;

if (x1 <= upperRight[0]) {

finiteB.x = x1;

finiteB.y = bottomLeft[1];

}

}

} else if (pointA.x === pointB.x && pointA.y !== pointB.y) { //if x points are same and y are different this condition will be executed

finiteA.x = pointA.x;

finiteA.y = upperRight[1];

finiteB.x = pointB.x;

finiteB.y = bottomLeft[1];

} else if (pointA.y === pointB.y && pointA.x !== pointB.x) { //if y points are same and x are different this condition will be executed

finiteA.x = upperRight[0];

finiteA.y = pointA.y;

finiteB.x = bottomLeft[0];

finiteB.y = pointB.y;

} else { // if All the points are same

finiteA.x = upperRight[0];

finiteA.y = pointA.y;

finiteB.x = bottomLeft[0];

finiteB.y = pointB.y;

}

points = Object.assign([], [finiteA, pointA, pointB, finiteB]);

}

}

return points;

}

## Point Rotation

* This method helps to rotate the Arrow points with the line flow is drawn when the extended lines is enabled.

pointRotation: (data) => {

let rotation = 0;

if (showLine && extendedLines && data && data.dataset && data.dataset.data && data.dataset.data.length > 0

&& ((data.dataset.data[data.dataIndex].dummyPoint) || (!data.dataset.data[data.dataIndex].dummyPoint))) {

let line = data.dataset.data;

let index = data.dataIndex;

line = data.dataset.data.filter((d) => d.dummyPoint);

index = data.dataIndex === 0 ? 0 : 1;

if (line && line.length === 2) {

let firstPoints = Object.assign({}, line[index]);

let secondPoint = Object.assign({}, line[index === 0 ? 1 : 0]);

if (firstPoints && Object.keys(firstPoints).length >= 2 && secondPoint && Object.keys(secondPoint).length >= 2) {

if (firstPoints.x !== undefined && secondPoint.x !== undefined && firstPoints.y !== undefined && secondPoint.y !== undefined) {

let switched = false;

if (index === 0 && firstPoints.y >= secondPoint.y) {

switched = true;

} else if (index === 1 && firstPoints.y <= secondPoint.y) {

switched = true;

}

if (switched) {

firstPoints = Object.assign({}, secondPoint);

secondPoint = Object.assign({}, line[index]);

}

let rotation1 = Math.atan2((firstPoints.x - secondPoint.x), (firstPoints.y - secondPoint.y)) \* (180 / Math.PI) + 360;

rotation1 = (rotation1 + 180) % 360;

rotation = parseInt((switched) ? rotation1 : (rotation1 + 180) % 360);

if (index === 0) {

if (lastRotation) {

lastRotation.rotation = rotation;

}

} else {

if (lastRotation.rotation === rotation) {

rotation = parseInt((parseInt(rotation1) !== rotation) ? rotation1 : ((rotation1 - 180) % 360));

}

}

}

}

}

}

return rotation;

}

## Find Axis

* This method is responsible for identifying the axis based on the mouse event on the chart get the x, and y position.
* We need to pass the event as a parameter along with our chart data and need to mention snap to grid in Boolean value.

const findAxis = (event, myChart, snapToGrid) => {

let x = null, y = null;

try {

if (snapToGrid === true) {

const pointX = myChart.getElementsAtEventForMode(event, 'nearest', { intersect: false, axis: 'x' }, false);

const pointY = myChart.getElementsAtEventForMode(event, 'nearest', { intersect: false, axis: 'y' }, false);

x = pointX?.[0]?.element?.$context?.parsed?.x;

y = pointY?.[0]?.element?.$context?.parsed?.y;

} else {

let xAxisID = 'x0';

let yAxisID = 'y0';

if (event.touches) {

x = myChart.scales[xAxisID].getValueForPixel((event?.native?.touches?.[0]?.clientX || event?.touches?.[0]?.clientX) - myChart.canvas.getBoundingClientRect().left);

y = myChart.scales[yAxisID].getValueForPixel((event?.native?.touches?.[0]?.clientY || event?.touches?.[0]?.clientY) - myChart.canvas.getBoundingClientRect().top);

} else {

x = myChart.scales[xAxisID].getValueForPixel((event?.native?.clientX || event?.clientX) - myChart.canvas.getBoundingClientRect().left);

y = myChart.scales[yAxisID].getValueForPixel((event?.native?.clientY || event?.clientY) - myChart.canvas.getBoundingClientRect().top);

}

if (Number.isNaN(x) === false) {

x = x.toFixed(1);

}

if (Number.isNaN(y) === false) {

y = y.toFixed(1);

}

}

} catch (e) {

console.error(e);

}

return { x, y };

}

## Axis line scale labels

* + - This method helps to form a datasets for the axis line labels based on the axis line drawn and x axis scale and y axis scale.

const scaleLabelOptionsChange = (scale, bottomLeft, upperRight, xAxisScale, yAxisScale, chartData, incrementLabel) => {

if (scale === 0 || scale === 1) {

for (let ind = 0; ind >= (scale === 1 ? parseFloat(bottomLeft[0]) : parseFloat(bottomLeft[1])); ind = parseInt(ind) - (scale === 0 ? yAxisScale : xAxisScale)) {

if (incrementLabel && ((incrementLabel.code === "ALL" || incrementLabel.code === "NONE" || incrementLabel.code === "SELECT") ||

(incrementLabel.code === "LEAST\_AND\_GREAT" && (parseFloat(upperRight[scale]) === ind)))) {

chartData[scale].data.push({ x: (scale === 0 ? 0 : ind), y: (scale === 0 ? ind : 0) });

}

};

chartData[scale].data.reverse();

for (let ind = 0; ind <= (scale === 1 ? parseFloat(upperRight[0]) : parseFloat(upperRight[1])); ind = parseInt(ind) + (scale === 0 ? yAxisScale : xAxisScale)) {

if (incrementLabel && ((incrementLabel.code === "ALL" || incrementLabel.code === "NONE" || incrementLabel.code === "SELECT") ||

(incrementLabel.code === "LEAST\_AND\_GREAT" && (parseFloat(bottomLeft[scale]) === ind)))) {

chartData[scale].data.push({ x: (scale === 0 ? 0 : ind), y: (scale === 0 ? ind : 0) });

}

};

// Include the axes end points to the list if they are missing while scaling.

if (scale === 0) {

// Y Axis scale

if (undefined === chartData[scale].data.find(point => point.x === 0 && point.y === parseInt(upperRight[1]))) {

chartData[scale].data.push({ x: 0, y: parseInt(upperRight[1]) });

}

if (undefined === chartData[scale].data.find(point => point.x === 0 && point.y === parseInt(bottomLeft[1]))) {

chartData[scale].data.push({ x: 0, y: parseInt(bottomLeft[1]) });

}

} else if (scale === 1) {

// X Axis scale

if (undefined === chartData[scale].data.find(point => point.x === parseInt(upperRight[0]) && point.y === 0)) {

chartData[scale].data.push({ x: parseInt(upperRight[0]), y: 0 });

}

if (undefined === chartData[scale].data.find(point => point.x === parseInt(bottomLeft[0]) && point.y === 0)) {

chartData[scale].data.push({ x: parseInt(bottomLeft[0]), y: 0 });

}

}

}

}

## Math Logic for calculating the upper and lower points in line items

* + - To Find the slope

m = ((y2-y1)/(x2-x1));

* + - Find Y-Intercept

c = (y1 - mx1);

Line equation - y=mx+c;

* + - Calculate Upper point - ex upper right is (10,20)
      * First calculate upper point with x value

yu = m\*10 + c;

* + - * If yu is less than or equal to 20, then upper point is (10,yu), else calculate upper point with y value

xu = (20-c)/m;

* + - * If xu is less than or equal to 10, then upper point is (xu, 20)
    - Calculate Lower point - ex lower left is (-10,-20)
      * First calculate lower point with x value

yl = m\*-10 + c;

* + - * If yl is greater than or equal to -20, then lower point is (-10, yl), else calculate lower point with y value

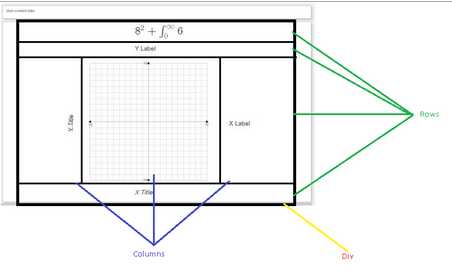
xl = (-20-c)/m;

* + - * If xl is greater than or equal to -10, then the lower point is (xl, -20)
    - Finally, once you get to the upper point and lower point - you can place an arrow at that location.

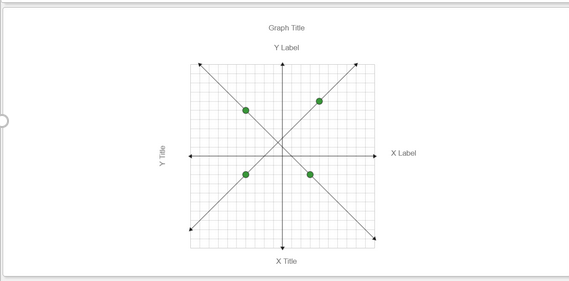
# Graph Orientation in UI

## Titles & Labels Layout

* To show Labels and Titles across the four side of graph we are going to split the graph along with div structure as given in below image.



* We are not using the chart.js existing functionalities to show titles and labels, because it will not accept the math plugin data from CK-Editor html content.
* A graph will look like below after drawing a line in the SP preview.
* We are parsing the CK-Editor 5 content and showing inside the div.
* With the help of CSS classes we are placing the titles and labels as we like in the UI.



* + - With latest CP user can able to move the Y Axis label and X Axis label along with the axis line.