Straight Line Item Type Tech Spec

[I. Item Type Overview 2](#_Toc2090916754)

[A. JSON Structure 2](#_Toc1265818784)

[B. Other Database Fields Used 4](#_Toc1327211610)

[C. Scoring Types used 4](#_Toc1337092987)

[D. Item Validation Rules 4](#_Toc241480439)

[E. Third Party Libraries Used 4](#_Toc985259977)

[II. Item Content and Response View 5](#_Toc1139052637)

[III. Technical Updates 11](#_Toc618241607)

[A. Outstanding Defects 11](#_Toc1263712491)

[B. Business Flow 12](#_Toc1527721810)

[C. Implementation 13](#_Toc614552954)

[D. Software Updates Needed to Align with Project Standards 19](#_Toc2023943102)

[E. How to consume 19](#_Toc616585714)

[F. Click History 21](#_Toc735364738)

[IV. Test Scenarios 21](#_Toc1811627265)

[V. Final Effort Estimate 24](#_Toc2091681425)

[VI. Questions 24](#_Toc1053091725)

# Item Type Overview

## JSON Structure

* + Straight Line item JSON structure below.

{

"uuid": "522c10-4adc-03c2-4be-361fbb304127",

"width": 500,

"height": 500,

"maxLines": 2,

"bottomLeft": "-10, -10",

"upperRight": "10, 10",

"slope": true,

"showGrid": true,

"lineArrow": false,

"snapToGrid": true,

"extendedLines": true,

"incrementLabel": "ALL",

"xAxisScale": 1,

"yAxisScale": 1,

"graphTitle": "<p id=\"cb-378090-8-42\">Graph Title</p>\n",

"xAxisTitle": "<p id=\"cb-185883-1-72\">X Axis Title</p>\n",

"yAxisTitle": "<p id=\"cb-143479-2-63\">Y Axis Title</p>\n",

"yAxisLabel": "<p id=\"cb-623541-8-54\">Y Axis Label</p>\n",

"xAxisLabel": "<p id=\"cb-600598-6-52\">X Axis Label</p>\n",

"stemContent": "<p id=\"cb-696779-6-68\">Stem content for straight line</p>\n",

"itemTypeCode": "SL",

"minItemWidth": 0,

"minItemHeight": 0,

"orderPairOne": true,

"orderPairTwo": true,

"xIntercept": false,

"yIntercept": false,

"lineRelationships": false,

"relationships": {

"parallel": true,

"intersecting": false,

"perpendicular": false

},

" axisLineSize ":1,

" axisLabelSize ":9,

"correctResponse": [

{

"id": "response-id-1",

"value": {

"line": [

{

"x": -5,

"y": 6,

"scoringSelection": " Ordered Pair 1"

},

{

"x": 0,

"y": 1,

"scoringSelection": " X Intercept "

}

],

"slopeValue": -1

}

},

{

"id": "response-id-2",

"value": {

"line": [

{

"x": 6,

"y": 6

"scoringSelection": " Ordered Pair 1"

},

{

"x": -7,

"y": 0,

"scoringSelection": "X Intercept"

}

]

}

}

]

}

* + - There is no optionList for straight line item.

## Other Database Fields Used

* + We are not using any other specific database fields for this item type other than the standard field for an item.

## Scoring Types used

* + - The Scoring types for Straight Line are
* No Scoring needed
* Correct Only
* Partial Credit

## Item Validation Rules

These are the validations used in this item

* CORRECT\_ANSWER\_REQUIRED\_LEN- The correct Answer is required.
* SCORE\_METHOD\_REQUIRED- Scoring Method is required.
* MAX\_SCORE\_REQUIRED - Maximum Score is required and it cannot be zero.
* SCORE\_EXCEED - Total score cannot be greater than maximum score.
* SCORE\_DECEED - Total score cannot be less than maximum score.
* SCORE\_NEGATIVE - Score must be between zero and the maximum.
* CONTENT\_CODE\_SUGGESTED – At least one primary content code should be selected.

**Common Validations**

* In Attributes and Metadata accordion Content Area, Grade, Framework, Taxonomy, Cognitive Category, Testing program, Usage and Estimate of difficulty validations are included for this item.

## Third Party Libraries Used

* The Chart.js, ~~react-chartjs-2,~~ chartjs-plugin-datalabels, chartjs-plugin-dragdata are used in this Item.

<https://www.npmjs.com/package/chart.js> -- Chart.js

[~~https://www.npmjs.com/package/react-chartjs-2~~](https://www.npmjs.com/package/react-chartjs-2) ~~-- React Chartjs-2~~

<https://www.npmjs.com/package/chartjs-plugin-datalabels>-- Chartjs-plugin-datalabels.

<https://www.npmjs.com/package/chartjs-plugin-dragdata> -- Chartjs - plugin-dragdata.

* + - Earlier in CP we have used react-chartjs-2 version 2.11.2 which is wrapper component of chart.js for react.js.
    - Chart.js over the react-chartjs-2 plugin.

***Pros***

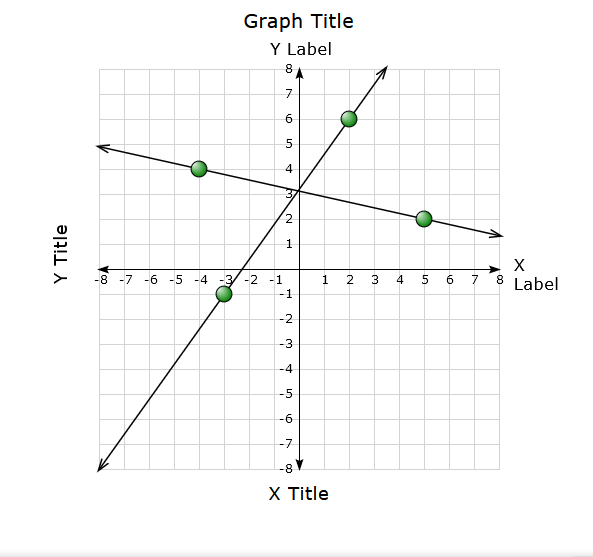
* + - * By using chart js directly we can use all the customization and utilize the version updates immediately.
      * Chart.js with latest version fixed bugs.

***Cons***

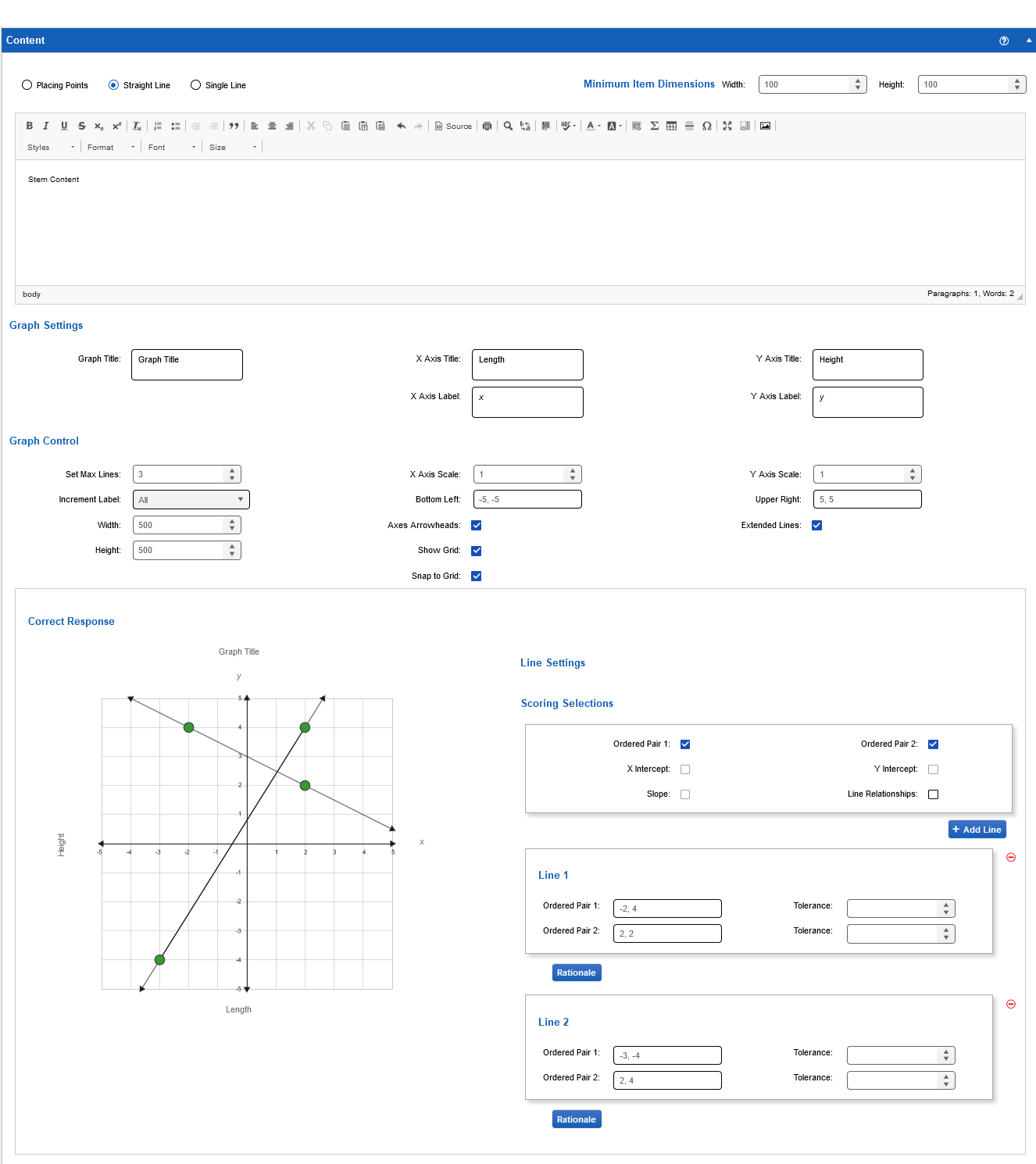
* + - * Wrapper class of react-chartjs-2 that needs to release new version with latest chart.js version.
      * By using of react-chartjs-2 we couldn't utilize all the customization of chart.js features.
    - Based on above points, plan to use the chart.js plugin directly instead of using wrapper component (react-chartjs-2).

# Item Content and Response View

* CB1 SP Preview for Straight Line

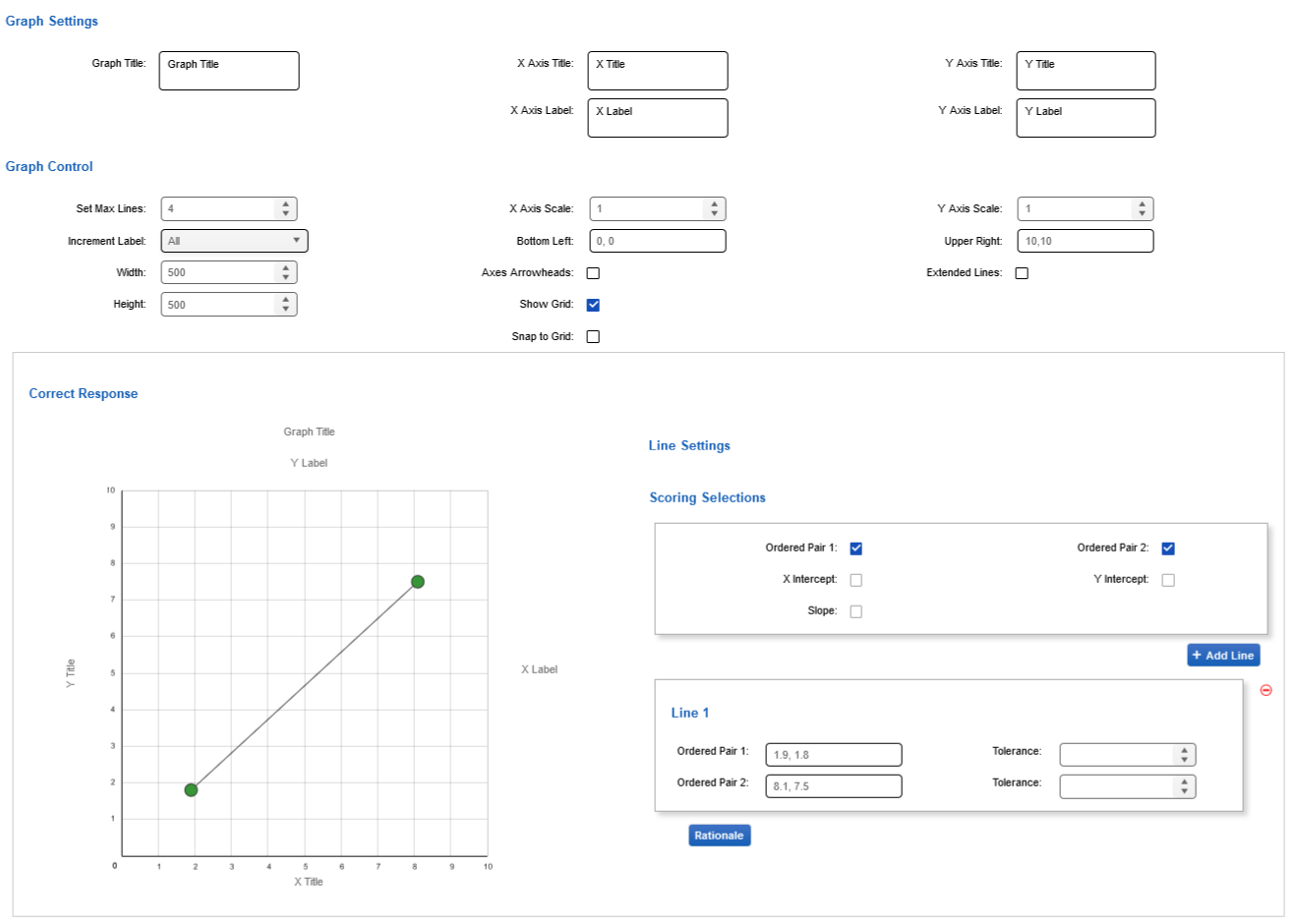


* Content Portal – Current Content Page (Straight Line item)



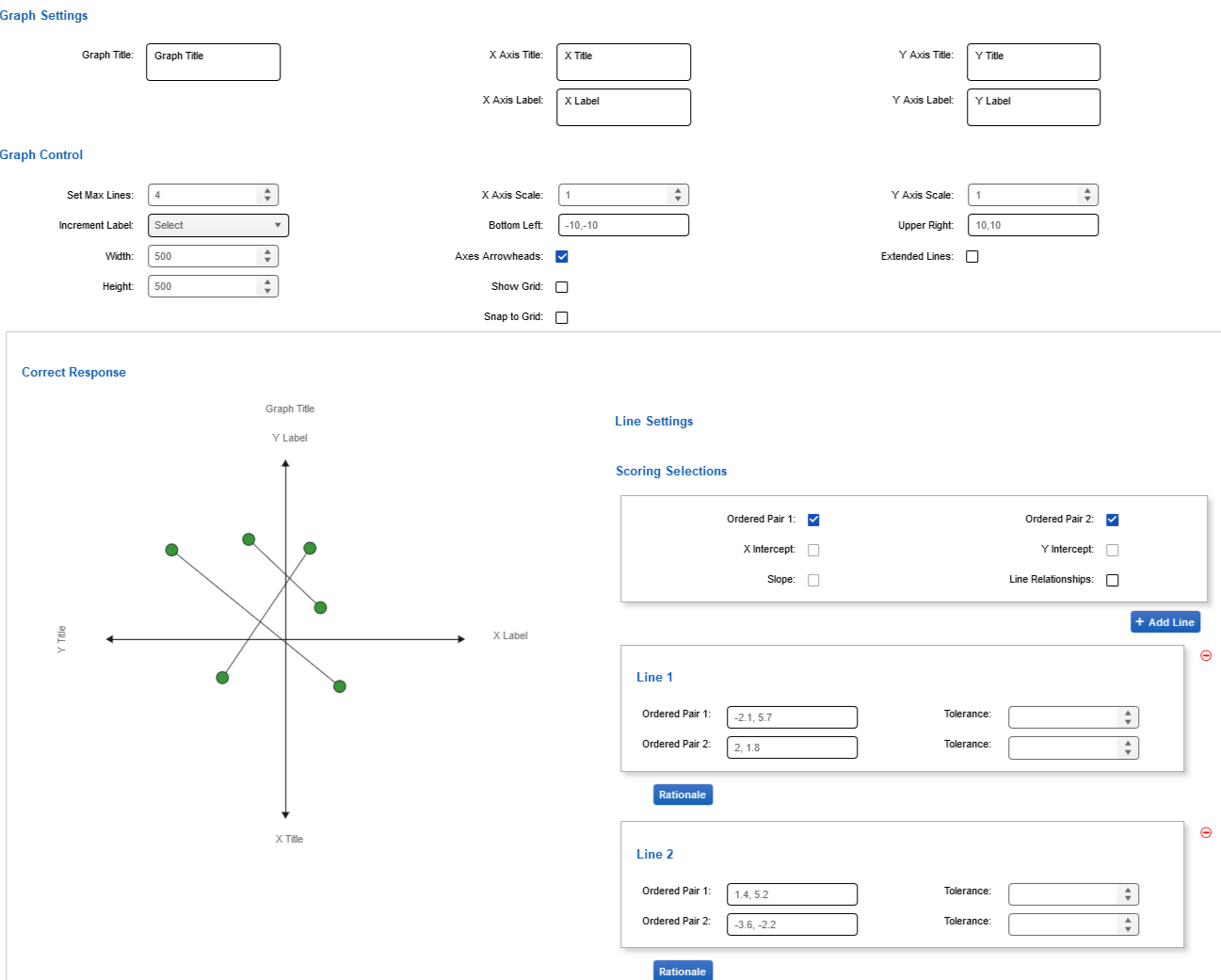
* Content Portal – Current Content Page (Straight Line Item)

Single line with incremental label as ALL, Enabled Show Grid and Disabled Axes Arrowheads, Snap to grid and extended lines.



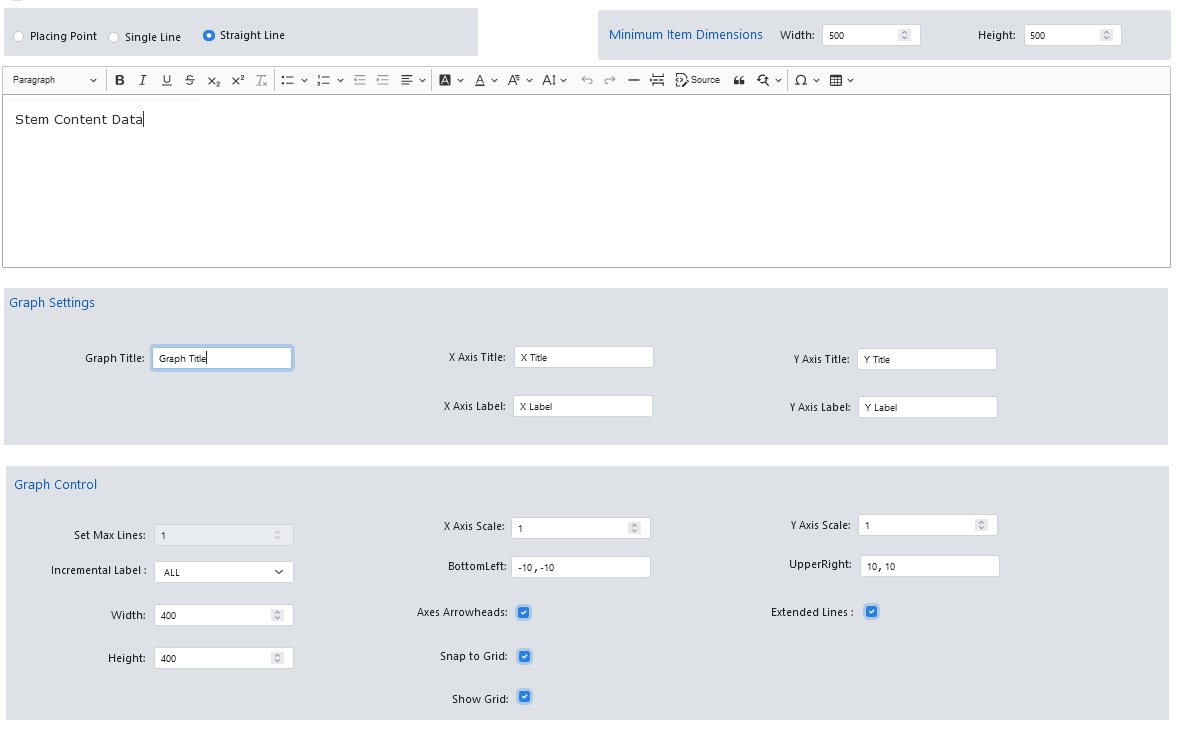
* Content Portal – Current Content Page (Straight Line Item)

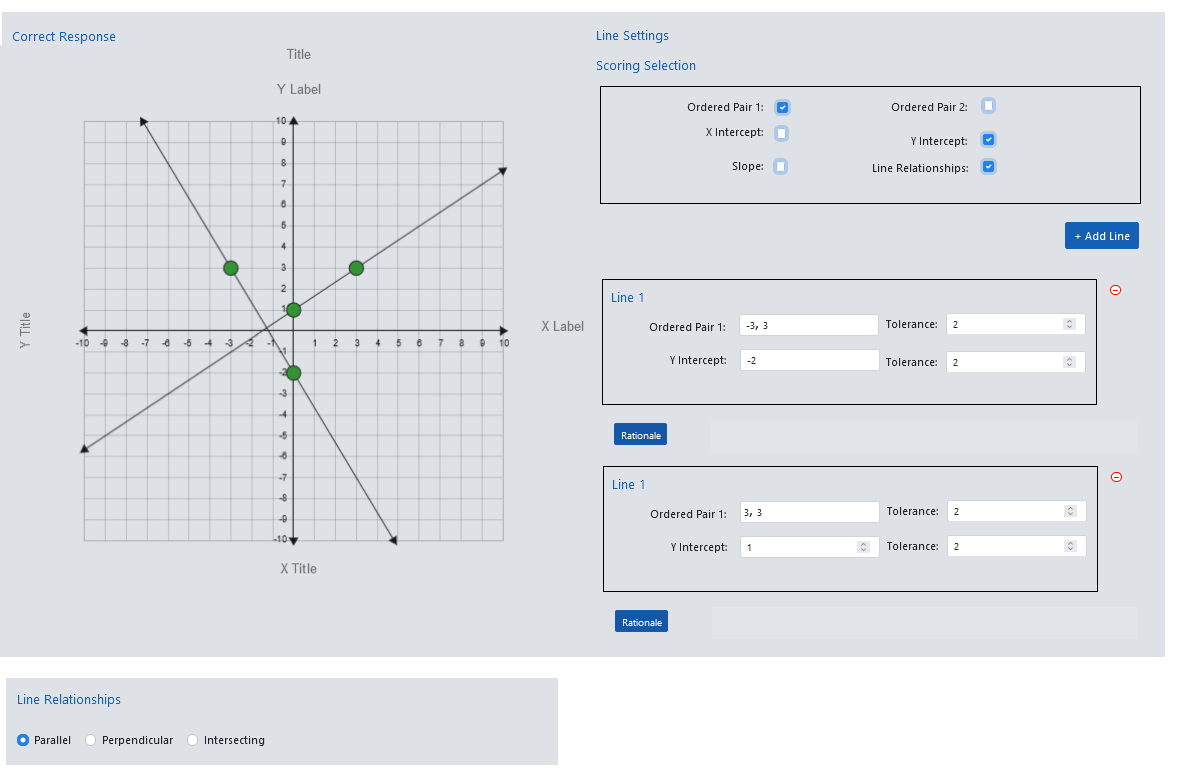
Multiple lines with Disabled show grid and extended line, Axes arrowheads as enabled with no labels for axis lines.



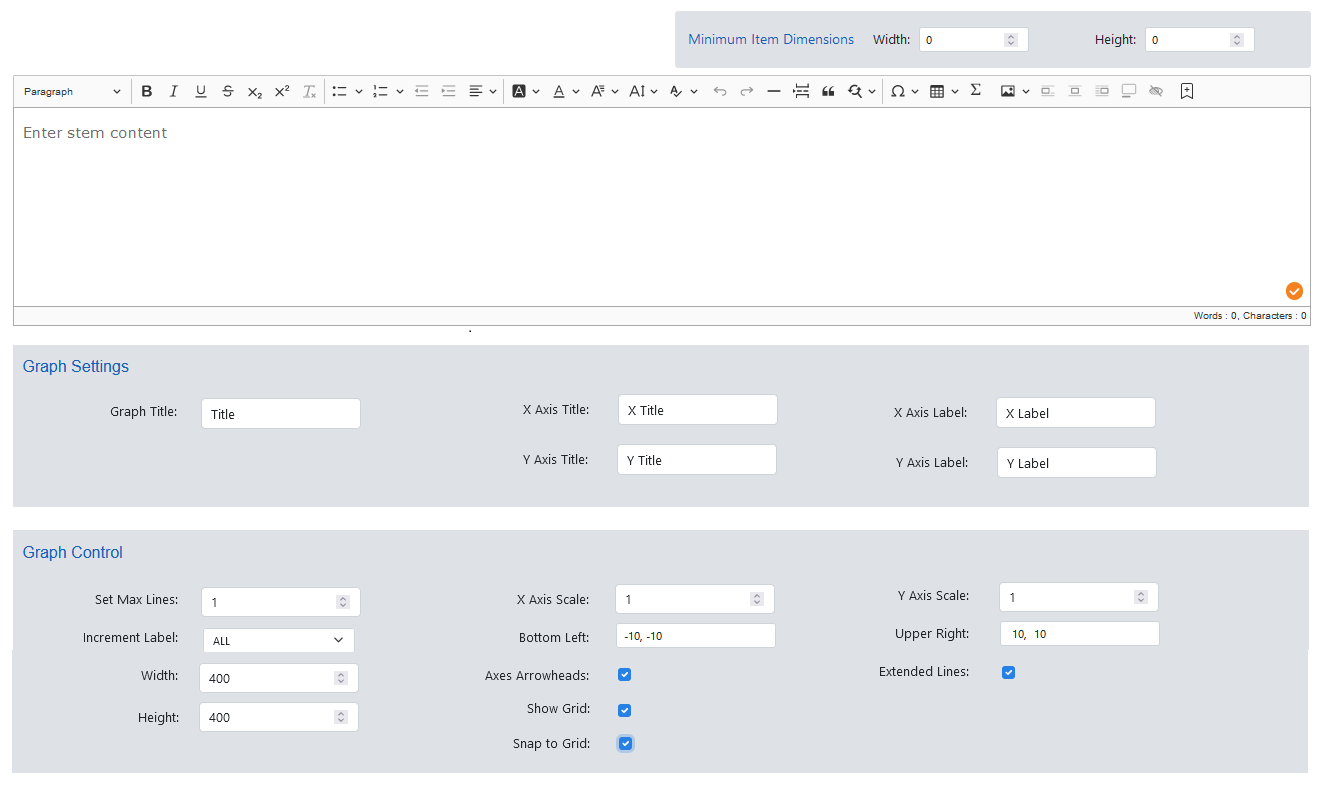
* + Content Portal – New Content Page (Straight Line)

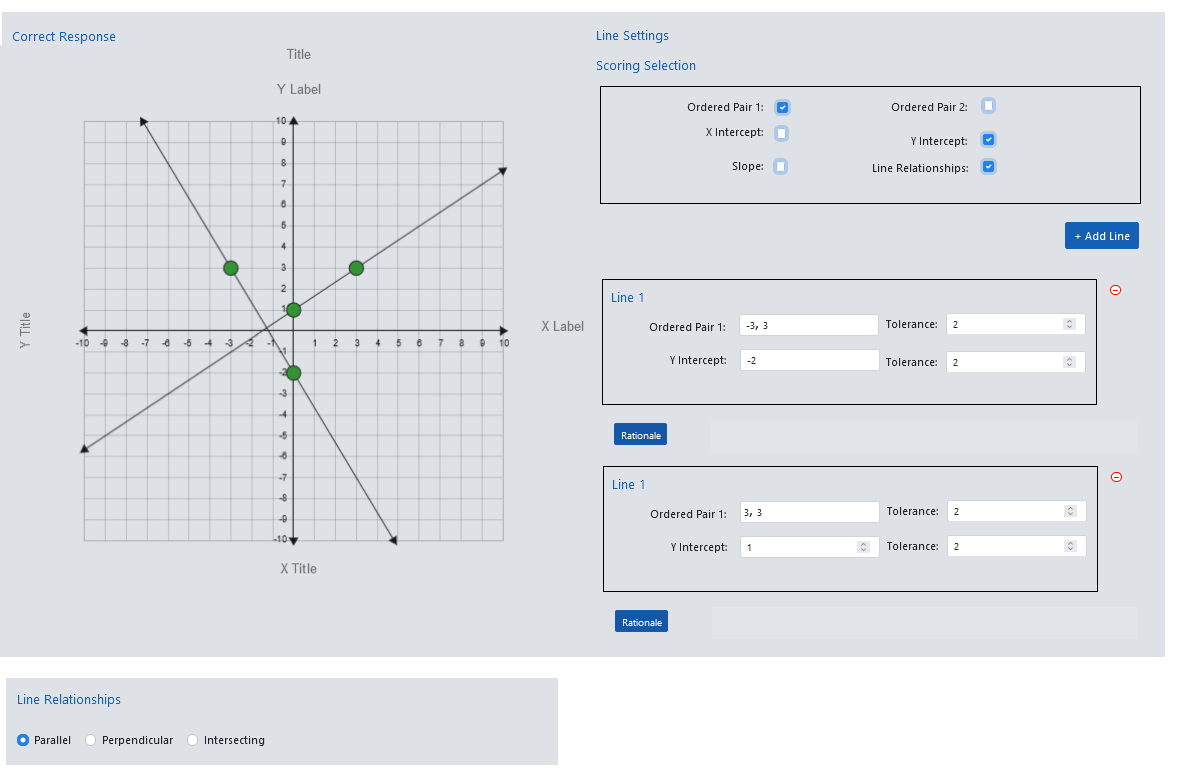
Multiple Lines with Enabled show grid and snap to grid, Axes arrowheads as enabled with ALL labels for axis lines. Enabled Line Relationships and parallel chosen as the relation for the line.



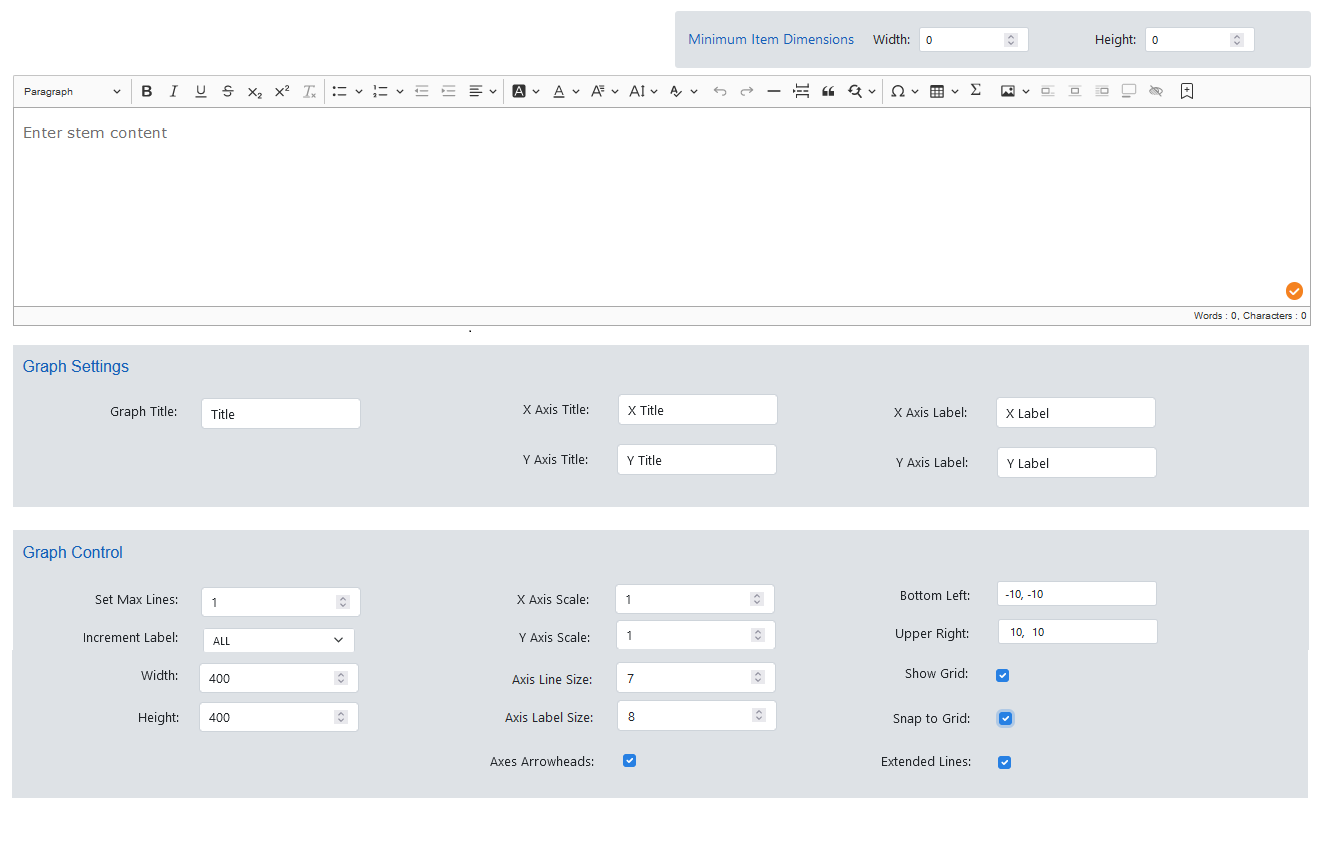


* + Updated Content UI for Straight Line (After removing item swap functionality)

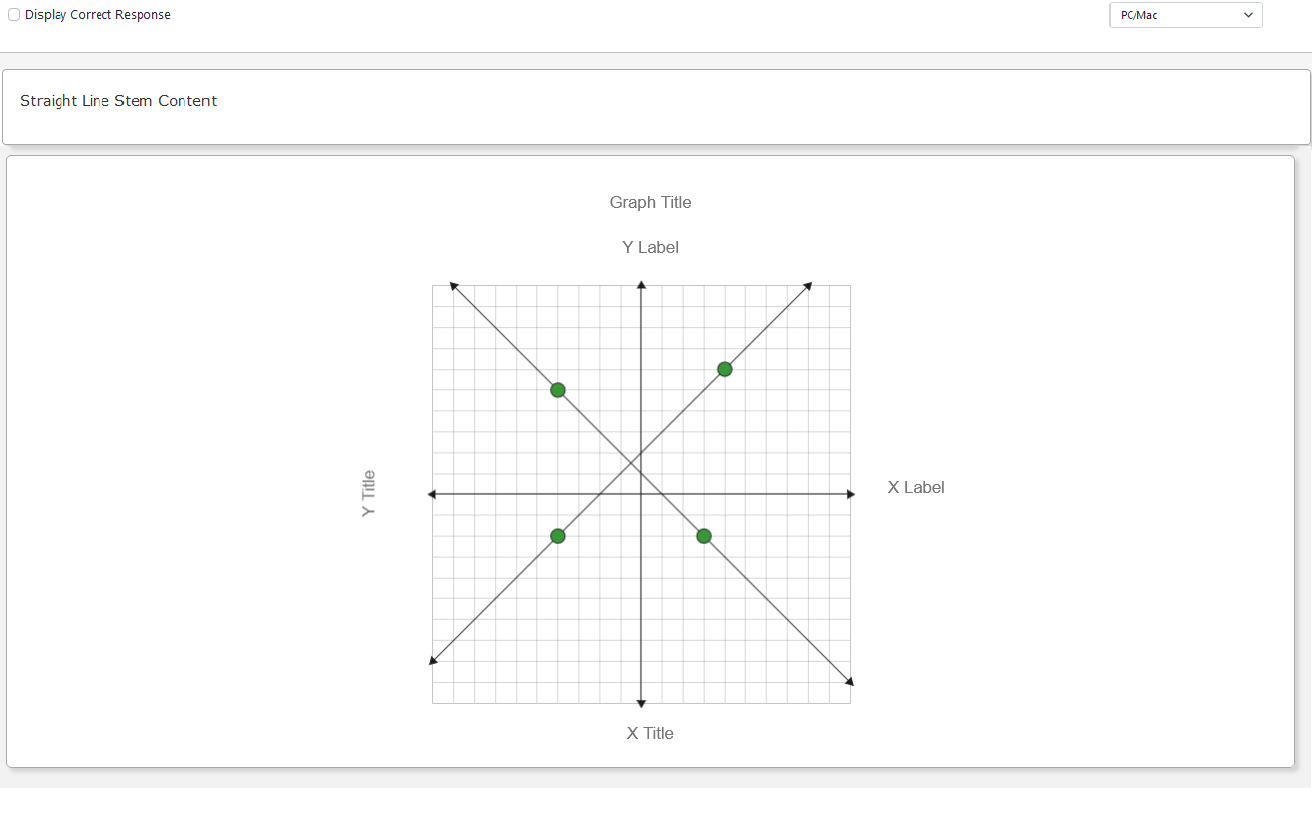




* + Latest Content UI after adding two new fields ( Axis Line Size & Axis Label Size)



* + Updated New Content Preview for Straight Line.



# Technical Updates

## Outstanding Defects

**Backlog defects**:

* X and Y Axis Titles are not following movement of the axis when bottom left and upper right are adjusted. Title should be along the corresponding axis, and the labels should be at the end of the x and y axis lines.
* Points placed at the end of an axis and at the corner of the graph cannot be removed. This seems like a lower priority because I don't think items are usually set up with the correct response having a point at the end of the axis. I added the issue because you can add & remove a point in these locations in SP.
* Extended line arrow point rotation is not straight when bottom left and upper right have different values (i.e: BottomLeft – (-5,-10) and UpperRight – (10, 6)).
* Console error while click over the graph (plugin error).
* Overall general function good, difficulty reading the slope, x, or y intercept, and ordered pairs without the numbers in the graph - 1229
* US19753 Unable to adjust the thickness of the axes to differentiate them from the grid lines – we need separate field to increase the thickness of the axis line.
* US19754 Unable to format the increment labels to adjust sizing – plugin having some restriction.

## Business Flow

* The Min item dimensions functionality will be same as every item.
* Stem content will have the questions.
* Users can set the titles and labels for the graph.
* Based on titles and labels given in Graph setting area same will be populated in the correct response area/preview.
* The user can change the graph controls options from default values (i.e., X Axis scale = 1, Y Axis Scale = 1).
* User can set Maximum Lines to be specified using set max lines.
* We can change the X Axis and Y Axis scale values.
* We can choose labels for the graph from the incremental label dropdown option.
* The bottom left and upper right can be changed (I.e., Bottom left = [-10, -10], Upper right = [10,10]), bottom left is to specify the left and bottom side of the axis line of the graph and the upper right is to specify top and right side axis line of the graph.
* The height and width of the graph can be changed (I.e., Height = 400 and width = 430).
* The User can enable / disable Axes Arrowheads, show grid, snap to grid options, which needs to be shown in the graph.
* Users can avail themselves of the extended line option by enabling the checkbox.
* User can adjust the thickness of the axis line using Axis Line Size field. Default value will be 1.
* User can adjust the thickness of the axis label using Axis Label Size field. Default value will be 9 pt.
* User can choose two points to draw a line.
* User can drag the points to change the co-ordinate of the point.
* User can choose any one of the line relationships if the number of lines drawn is 2 or more.
* The User will be able to plot the points in the preview.
* If display correct response is enabled, then correct response will be set in the Preview and user will not be able do any action on it.

## Implementation

***Create:***

* + - As part of Straight Line item creation, will create the StraightLine.jsx file inside the components/create/straightline folder.
    - This StraightLine.jsx is completely responsible for rendering the item content creation except for the correct response render.
    - Shared components ItemDimension and StemContent will be used for this item.

import ItemDimensions from '../shared/ItemDimensions';

import StemContent from '../shared/StemContent';

//ItemDimension will be called by passing all the required parameters

<ItemDimensions

minWidth={item?.item\_json?.minItemWidth || 0}

minHeight={item?.item\_json?.minItemHeight || 0}

onChange={(dimension) => {

if (dimension?.minWidth !== item?.item\_json?.minItemWidth) {

updateItemJson('minItemWidth', dimension.minWidth);}

if (dimension?.minHeight !== item.item\_json.minItemHeight){

updateItemJson('minItemHeight', dimension.minHeight);}

}}

/>

//StemContent will be called by passing all the required parameters

<StemContent

data={item.item\_json?.stemContent}

onUpdate={updateItemJson}

fieldName={'stemContent'}

/>

* + - ~~Common bootstrap/Input component will be used for Graph Width, Height, BottomLeft, UpperRight, SetMaxPoints, Y-Axis Scale, and X-axis scale.~~

~~import Input from '../../common/bootstrap/Input';~~

~~//Input will be called by passing all the required parameters~~

~~//Graph Width~~

~~<Input~~

~~id={'li\_width'}~~

~~type={'number'}~~

~~name={'li\_width'}~~

~~label={{code: 'width',~~

~~addColon: true,~~

~~className: 'text-right',~~

~~style: {marginTop: '5px'}~~

~~}}~~

~~inline={true}~~

~~/>~~

~~//Graph Height~~

~~<Input~~

~~id={'li\_height'}~~

~~type={'number'}~~

~~name={'li\_height'}~~

~~label={{code: 'height',~~

~~addColon: true,~~

~~className: 'text-right',~~

~~style: {marginTop: '5px'}~~

~~}}~~

~~inline={true}~~

~~/>~~

~~//Set Max Lines~~

~~<Input~~

~~id={'max\_lines'}~~

~~type={'number'}~~

~~name={'max\_lines'}~~

~~label={{code: 'max\_lines',~~

~~addColon: true,~~

~~className: 'text-center',~~

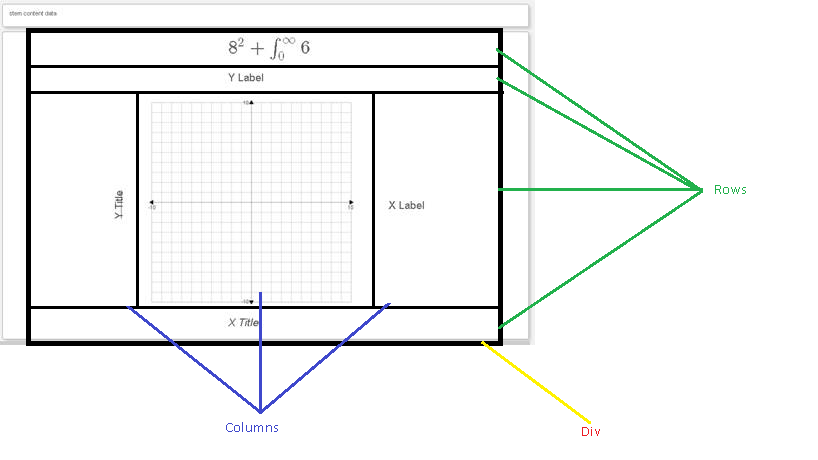
~~style: {marginTop: '5px'}~~

~~}}~~

~~inline={true}~~

~~/>~~

* + - For Labels and Titles we are going to split the graph along with the div structure as given in the below image.



* + - GraphSettings.jsx will be created inside the components/create/shared/lineitem folder and used for rendering the Titles and labels field.

import GraphSettings from '../shared/lineitem/GraphSettings';

//Graph Settings will be called by passing all the required parameters

<GraphSettings

Content={item?.item\_json}

config={config}

/>

* + - GraphControl.jsx will be created inside the components/create/shared/lineitem folder and used on graph input operations and values.

import GraphControl from '../shared/lineitem/GraphControl';

//Graph Control will be called by passing all the required parameters

<GraphControl

content={item?.item\_json}

config={config}

/>

* + - LineRelationship.jsx will be created inside the components/create/shared/lineitem folder and used on Line Relationship operations and values.

import LineRelationship from '../shared/lineitem/ LineRelationship;

//Line Relationship will be called by passing all the required parameters

<LineRelationship

content={item?.item\_json}

config={config}

/>

* + - Required constants, labels and messages will be added in the appropriate files.
    - Test files will be created for all the jsx files and located in the same straightline folder itself.
    - The new component StraightLine will be imported and mapped with the unique item code in the CreateInterface.jsx.

***Response:***

* + - One new folder ‘straightline’ will be created in components/display/response/ for creating the main response component for line item.
    - StraightLineResponses.jsx is the main component of responses that will receive the inputs and pass that to line item response sub-components.

import StraightLineResponses from './StraightLineResponses;

// Line Item Responses will be called by passing with all the required parameters.

<StraightLineResponses

content={item?.item\_json}

config={config}

showCorrectResponse={showCorrectResponse}

/>

* + - The Following method will Identify the graph boundary and calculate the dummy points to draw arrowheads. Based on the points given, the endpoint 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;

}

* + - To Handle the Extended Lines Axes Arrowheads rotation

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;

}

* + - ~~One new folder ‘lineitem’ will be created in components/display/response/lineitem/responses for creating the response component for line item.~~
    - ~~Line.jsx is the subcomponent that will handle the response of straight line and single line items.~~

~~import Line from './Line;~~

~~// Line Responses will be called by passing with all the required parameters.~~

~~<Line~~

~~content={item?.item\_json}~~

~~config={config}~~

~~showCorrectResponse={showCorrectResponse}~~

~~/>~~

* + - GraphResponse.jsx component is the subcomponent that will handle the graph operations and configurations.

import GraphResponse from './GraphResponse;

// Graph will be called by passing with all the required parameters.

<GraphResponse

item={item}

stemComponent={null}

config={config}

showCorrectResponse={true}

graphNative = {false}

/>

* + - Inside the Graph response the Drag functionality for the plotted points will be enabled using the 'chartjs-plugin-dragdata' plugin.

//Adding chartDragData plugin while creating the chart and enabling the Drag Functionality.

Importing the point draggable plugin for chart js

import ChartDragData from 'chartjs-plugin-dragdata';

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

* + - This same StraightLineResponses component will be called from both the StraightLine(Create) and StraightLinePreview(Preview) component.
    - The readable response for the correct response will be done with following structure.

"readable\_response": "\"\"Ordered Pair 1\n\n\"\"\"X intercept\n\n\""

* + - The readable response will be generated using following method

const generateReadableResponse = (itemData) => {

let readableResponse = '';

let responseValue = [];

let responseValue1 = [];

for (let i = 0; i < itemData.correctResponse.numberOfResponses; i++) {

itemData.correctResponse.responses.forEach((res) => {

if (res && res.point && res.point[0]) {

responseValue.push(

res.point[0].x + "," + res.point[0].y

);

}

if (res && res.point && res.point[1]) {

responseValue1.push(res.point[1].x + "," + res.point[1].y)

}

});

let singlePoint = responseValue.join()

let singlePoint1 = responseValue1.join()

let responseRow = singlePoint;

let responseRow1 = singlePoint1;

if (i === 0) {

readableResponse = `"${responseRow}"Ordered Pair 1\n\n""${responseRow1}"X intercept\n\n"`;

}

}

return readableResponse;

}

**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 lower point is (xl, -20)

* + - Finally, once you get the upper point and lower point - you can place an arrow at that location.

***Preview:***

* + - New folder ‘straightline’ will be created in components/display/item for the Straight-Line item preview.
    - StraightLinePreview.jsx is the component responsible to rendering the item preview.
    - StraightLineResponses component will be called from the StraightLinePreview.
    - While calling the StraightLineResponses will be passing the stem content with help of StemFormatter.

import StraightLineResponses from '../../response/straightline/StraightLineResponses;

<div className='row'>

{itemJson?.stemContent ? itemJson.stemContent: null}

</div>

// StraightLineResponses will be called by passing with all the required parameters.

<StraightLineResponses

item={item}

config={config}

onUpdate={handleResponse}

showCorrectResponse={showCorrectResponse}

/>

***Stories:***

* + - For the straight Line Item components Create, Response and Preview will have stories to render and check the UI.
    - Will have a folder called ‘sl’ inside the stories/assets which contain json files with all the possible combinations as mentioned below.
      * straight\_line\_single\_quadrant
      * straight\_line\_four\_quadrant
      * straight\_line\_four\_quadrant\_line\_relationship
      * straight\_line\_four\_quadrant\_extended\_lines
    - StraightLineNew.stories.jsx will be created to call the create line item component with default UI.
    - StraightLine.stories.jsx will be created under the stories/create/alternate to check the create straightline component with alternate UI.
    - Same as for other styles, separate story files will also be created and used.
    - StraightLineDisplay.stories.jsx will be created under the stories/display for StraightLine item preview check with default UI.
    - Inside the StraightLineDisplay.stories will be using all the json that we have created under the asset which help us to check the preview in all the combinations.
    - Display stories will also have a separate story for all the available styles.
    - For the Straight Line item response alone will have a separate story StraightLineResponses.stories.jsx under stories/response.

## Software Updates Needed to Align with Project Standards

* chart.js plugin version needs to be updated from 2.9.4 to 3.7.0.
* chartjs-plugin-datalabels version needs to be updated from 0.7.0 to 2.0.0.
* chartjs-plugin-dragdata version needs to be updated from 1.1.3 to 2.2.4.

## How to consume

* + - * This kite-react-items component will be used in the cp-prototype application, and CreateInterface and PreviewInterface will be called.

<CreateInterface

item={item}

onUpdate={updateItem}

config={config}

/>

<PreviewInterface

item={itemObj}

onUpdate={onUpdate}

config={config}

/>

* + - * In the kite-react-items chart-item response component is common for both Create and Preview components.
      * In the Create component it will be called without stemComponent like below.

<StraightLineResponses

item={item}

config={config}

showCorrectResponse={true}

/>

* + - * In StraightLineResponses component, a sub component will be called based on the item type.
      * In each response component a graph component will be called to display the graph and enable the graph operations.

<GraphResponse

item={item}

stemComponent={null}

config={config}

showCorrectResponse={true}

graphNative = {false}

/>

* + - * In the Display component it will be called with the stemComponent like below.

<StraightLineResponses

item={item}

stemComponent={<StemFormatter stemContent={itemJson?.stemContent} />}

config={config}

onUpdate={handleResponse}

showCorrectResponse={showCorrectResponse}

selected={selected}

/>

## Click History

* + - Straight Line Click history structure below

"responses": [

{

"responseoption": [ {

"line": [

{"x": 0, "y": 6.4, "scoringSelection": "Y Intercept"},

{"x": -2.3, "y": -2.8, "scoringSelection": "Slope"}

],

"slopeValue": 4

} ],

"ts": "Thu Feb 25 2021 09:05:26 GMT-0600 (Central Standard Time)",

"score": 0,

"referencescore": null,

"isCorrectResponse": false

}

]

# Test Scenarios

* Below is the Straight Line Item Test Scenarios.

**Content**

1. The User will be able to select Straight Line item and open the content accordion.
2. Users will be able to save the Item by giving the required fields. The user will be able to edit the Item after saving it in Draft/Final/Approved/Complete.
3. Users will be able to save the item in Draft status without completing all required fields. Once saved, those fields can be edited and saved again.
4. The User will be able to set Item dimensions for the item in item dimensions field.
5. The User will be able enter content in stem.
6. User can able to edit the Graph Settings section consists of graph title, x-axis title, y-axis title and x-axis label, y-axis label that will reflect on correct response.
7. User can set the Set Max Lines that will reflect on Number of lines in correct response graph.
8. User can change the x-axis scale, y-axis scale fields for changing the graph scale.
9. User can change the incremental label, bottom-left, upper-right fields, snap grid for changing the graph properties in the correct response.
10. While changing the incremental label as ALL user can see all the labels of axis line in the graph. If it is NONE –no labels will be displayed and for First & Last Starting and ending label of the axis line.
11. User can change the default values of Bottom Left and Upper Right, based on this axis line will be drawn in the graph.
12. If the user enables snap to grid option, the decimal value of plotted point will get round off based on the scale value.
13. User can adjust the thickness of the axis line using Axis Line Size field.
14. User can adjust the thickness of the axis label using Axis Label Size field.
15. User can adjust the dimensions of graph by providing values in width and height fields in graph control section.
16. User can check the Axes Arrowheads for showing axis line arrows in graph.
17. User can check the show grid for showing the grid lines in graph. If show grid is unchecked, axes and increment labels will still appear.
18. User can check the extended line to extend the line till the edges of the graph and arrow heads will be shown at the end of the line.
19. User can add the add lines button on correct response panel. Also, you can enter content in rationale box below it.
20. User can line by single click on the graph.
21. User will be able to adjust the plotted points in graph by changing x, y in correct response or drag the points in the graph.
22. User can remove the plotted points by clicking over it or clicking on the remove icon on the right side of the graph.
23. User needs to choose any two-scoring selection option to draw a line in the graph.
24. For straight line users can enable line relationship checkbox if the number of lines is more than one.
25. User can choose line relationship as parallel / perpendicular/ intersecting by clicking on radio button.
26. User can add the Tolerance value for each point plotted in the graph.

**Preview**

1. The User will be able to open the preview, by clicking on the preview button in common header.
2. The Stem content that we created should be shown in CP (Content Portal) Preview.
3. The empty graph will be populated in the preview.
4. When the user doesn’t enable the display correct response checkbox, the User will be able to plot the points and draw the line.
5. If display correct response is enabled, then correct response will be set in the Preview what we have set in content page and user will not be able do any action on it.
6. Item response should render based on the configuration of what user set in the content page.

**Scoring**

1. User can set the scores in scoring accordion.
2. The scoring method for Straight line item are Partial credit, No scoring needed and correct only.
3. User will be able to choose the scoring method from the drop-down menu.
4. For Straight Line user can able to see the selected line relationship if he added more than two lines.
5. When a user selected no scoring needed as scoring methods, no additional fields will be displayed.
6. When the user selects correct only as scoring methods, the correct response table will be displayed.
7. When user selected Partial credit as scoring methods, the correct response and partial credit table will be displayed, and the max score is divided between the correct responses but is editable.

**Attributes and Metadata**

1. Users will be able to open the attribute and metadata accordion and will be able to select Item metadata, content codes, cognitive settings, tags, and item details.
2. These are mandatory fields and should be filled before saving the Item in a Final /Approved state.

# Final Effort Estimate

|  |  |
| --- | --- |
| **Tasks** | **Effort in Hours** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Questions

1. Currently we have combined the UI and code for Single line, Straight Line and Placing point.

- Are we going to separate the UI and code?

**Byron** - *One vote for yes – I think each item type should be independent and you shouldn’t switch types via a group like that.  If we want, we can try to preserve relevant data when the user switches the item type using the regular menu.*

**Karen** - *Agree – independent item types.*

1. Graph settings fields for line graph, placing point, single line and straight line are same.

- Can we create Graph setting as a common component?

Yes, this should be common component.

1. Is it required to provide the functionality of drawing lines using keyboard in preview.

- This need to be planned in future release.

**Questions about Line Items Scoring Selections – Expected Functionality**

1. When Ordered Pairs are used to set up the correct response, does the order the points are selected matter for scoring the item correctly? For example, in Editor they’re called “Starting” and “Ending” points and you must select the starting point first in order to get the item correct.
2. When Slope and either x or y intercept are selected for the Scoring Selection, you enter the intercept and the slope value in the Content Accordion. The Correct Response graph shows the intercept and then tries to plot another point at the intercept on the other axis. These two points are what also shows up when you display the Correct Response in the Preview. In the Scoring Accordion, the Slope appears as the coordinates for that second point.

a. Is the slope showing up as a coordinate the expected functionality?

b. An example I ran into when recreating an item we have in CB that seemed odd was a line with y intercept= 8 and slope=3. In the Scoring Accordion, the slope was displaying as the coordinates for the x intercept for the line (-2.666………, it never showed the “, 0”). This seems odd because the graph is set to Snap to Grid so that point of -2.666 isn’t selectable for test taker.

1. If the slope is supposed to display as a coordinate point in the Scoring Accordion, will the item still score any line created with the same slope and intercept as correct?
2. When you have a y-intercept and a slope of 0, it appears in the Scoring Accordion as “-Infinity, 0” –is that the expected functionality?
3. In CP, for Scoring Selection, when you pick “slope” you’re also able to pick either ordered pair 1 or ordered pair 2. If the ordered pairs work like they do in Editor today (as the point that needs to be selected first and the point that needs to be selected second), will they work to set up with Slope?
4. We do not have any items set up in Editor/CB with Line Relationships so we did not have anything to compare against when testing. You are able to set up a key with Line Relationships selected, but it doesn’t appear to function any differently if you have Line Relationships selected or not.