Frequently Asked Questions

FAQs

1. What is the relationship between **QCChart2D for JavaScript/TypeScript** and **QCSPCChart for JavaScript/TypeScript** and **QCRTGraph for JavaScript/TypeScript** ?
2. How do you create a chart with multiple coordinate systems and axes?
3. Can I add new axes, text objects, plot objects, and images to a chart after it is already displayed; or must I create a new chart from scratch taking into account the additional objects?
4. How do you zoom charts that use multiple coordinate systems?
5. How do you select a chart object and create a dialog panel that permits editing of that objects properties?
6. How do you handle missing data points in a chart?
7. How do you update a chart in real-time?
8. How do I prevent flicker when updating my charts on real-time?
9. How do you implement drill down, or data tool tips in a chart?
10. I do not want to my graph to auto-scale. How do I setup the graph axes for a specific range?
11. How do I update my data, and auto-rescale the chart scales and axes to reflect the new data, after it has already been drawn?
12. When I use the auto-scale and auto-axis routines my semi-log chart has the logarithmic axis scaled using powers of 10 (1, 10,100, 1000, etc.) as the starting and ending values, or as the major tick interval for labeling. How do I make my log graphs start at 20 and end at 50,000, with major tick marks at 20, 200, 2000 and 20000?
13. How do I create and use custom, multi-line string labels as the axis labels for my graph?
14. How do I place more than one graph in a view?
15. How do I use your software to generate GIF files?
16. Sometimes the major tick marks of an axis are missing the associated tick mark label ?

17. How do I change the order the chart objects are drawn? For example, I want one of my grid objects to be drawn under the charts line plot objects, and another grid object to be drawn top of the charts line plot objects.

18. How to I use a Forms scrollbar object to control horizontal scrolling of the data in my chart?

19. I am trying to plot 100,000,000 data points and it takes too long to draw the graph. What is wrong with the software and what can I do to make it faster?

20. How do I get data from my database into a chart?

21. How do I use this charting software to generate chart images "on-the-fly"?

22. Can **QCChart2D for JavaScript/TypeScript** be used to create web application using frameworks other than explicitly described I this manual.

1. What is the relationship between **QCChart2D for JavaScript/TypeScript** and **QCSPCChart for JavaScript/TypeScript** and **QCRTGraph for JavaScript/TypeScript** ?

QCChart2D is a subset (in its entirety) of both QCSPCChart and QCRTGraph. In order to minimize size, and maximize loading speed, the QCSPCChart library (qcspcchartts.js) and the QCRTGraph library (qcrtgraphts.js) contain their own copies of the QCChart2D library.

2. How do you create a chart with multiple coordinate systems and axes?

A chart can have as many coordinate systems and axes as you want. A single coordinate system can have one or more x- and/or y-axes. The most common use for multiple axes in a single coordinate system is to place y-axes on both the left and the right sides of a chart, and x-axes above and below. The left and bottom axes usually have numeric or date labels, and the top and right axes just tick marks. This does not have to be the case though; every axis can have axis labels if you want. In general, the axis position in the chart is determined by its intercept. The default value of the intercept is set to the minimums of the coordinate system that the axis is placed in. Adjusting the intercept using the **setAxisIntercept** method changes the position of the axis in the chart. The axis intercept value is set using units of the coordinate system at right angles to the axis. The example below, extracted from the LineFill example, places y-axes on both the left and right of the chart.

[TypeScript]

```
let xAxis: QCChartTS.TimeAxis =  QCChartTS.TimeAxis.newTimeAxis(pTransform1);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);
let yAxis: QCChartTS.LinearAxis =  QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
    //  Default places y-axis at miniumum of x-coordinate scale
yAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis);
let yAxis2: QCChartTS.LinearAxis =  QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
```

```
yAxis2.setAxisIntercept(xAxis.getAxisMax());
yAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);
yAxis2.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis2);;
```

[JavaScript]

```
let xAxis = QCChartTS.TimeAxis.newTimeAxis(pTransform1);
xAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(xAxis);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
//  Default places y-axis at miniumum of x-coordinate scale
yAxis.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis);
let yAxis2 = QCChartTS.LinearAxis.newLinearAxis(pTransform1, QCChartTS.ChartConstants.Y_AXIS);
yAxis2.setAxisIntercept(xAxis.getAxisMax());
yAxis2.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);
yAxis2.setColor(QCChartTS.ChartColor.WHITE);
chartVu.addChartObject(yAxis2);
```

The other common reason to have multiple axes in a chart is to delineate the simultaneous use of different coordinate systems in the chart. In this case each coordinate system has an x- and/or y-axis to differentiate it from the other coordinate systems. When the different coordinate systems are created, they usually overlay the same area of the chart. The default positioning of the axes for each coordinate system will all overlay one another, making the axes unreadable. In the y-axis case you will want to offset additional axes to the left, or to the right of the default axis position, using the **setAxisIntecept** method. When using the **setAxisIntercept** method, make sure you specify the position using the units of the coordinate system scale at right angles to the axis. Specify an intercept value outside of the normal scale range to offset the axes so that they do not overlap. The example below, extracted from the MultipleAxes.BuildMultiAxes example, creates one x-axis, common to all of the charts because the x-scaling for all of the coordinate systems match, and five y-axes, one for each of the five different coordinate systems.

[TypeScript]

```
let pTransform1: QCChartTS.CartesianCoordinates;
let pTransform2: QCChartTS.CartesianCoordinates;
let pTransform3: QCChartTS.CartesianCoordinates;
let pTransform4: QCChartTS.CartesianCoordinates;
let pTransform5: QCChartTS.CartesianCoordinates;

//  Initialize datasets, coordinate system ranges
//  The x-scale range for pTransform1 to pTransform5 are all the same, 0 - 100
//  The y-scale range for pTransform1 to pTransform5 are all different
//  The plotting area for each pTransform is indentical, leaving a large open
//  to the left for extra axes.

pTransform1.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform2.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform3.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform4.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform5.setGraphBorderDiagonal(0.35, .15, .9, 0.65);

let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 2,
QCChartTS.ChartConstants.LS_SOLID);
let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 2,
QCChartTS.ChartConstants.LS_SOLID);
let attrib3: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.GREEN, 2,
QCChartTS.ChartConstants.LS_SOLID);
```

```
let attrib4: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.ORANGE, 2,
QCChartTS.ChartConstants.LS_SOLID);
let attrib5: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.MAGENTA, 2,
QCChartTS.ChartConstants.LS_SOLID);

this.xAxis = QCChartTS.LinearAxis.newLinearAxis(this.pTransform1,
QCChartTS.ChartConstants.X_AXIS);
this.xAxis.setLineWidth(2);
this.chartVu.addChartObject(this.xAxis);
this.yAxis1 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis1.setAxisIntercept(0);
this.yAxis1.setChartObjAttributes(attrib1);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis1);
this.yAxis2 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform2,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis2.setAxisIntercept(-18);
this.yAxis2.setChartObjAttributes(attrib2);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis2);
this.yAxis3 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform3,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis3.setAxisIntercept(-35);
this.yAxis3.setChartObjAttributes(attrib3);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis3);
this.yAxis4 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform4,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis4.setAxisIntercept(-52);
this.yAxis4.setChartObjAttributes(attrib4);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis4);
this.yAxis5 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform5,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis5.setAxisIntercept(this.xAxis.getAxisMax());
this.yAxis5.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);
this.yAxis5.setChartObjAttributes(attrib5);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis5);
let xAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.xAxis);
xAxisLab.setTextFont(theFont);
this.chartVu.addChartObject(xAxisLab);
let yAxisLab1: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis1);
yAxisLab1.setTextFont(theFont);
yAxisLab1.setAxisLabelsFormat(QCChartTS.ChartConstants.BUSINESSFORMAT);
this.chartVu.addChartObject(yAxisLab1);
let yAxisLab2: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis2);
yAxisLab2.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab2);
let yAxisLab3: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis3);
yAxisLab3.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab3);
let yAxisLab4: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis4);
yAxisLab4.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab4);
let yAxisLab5: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis5);
yAxisLab5.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab5);
let axisTitleFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans
Serif",  QCChartTS.ChartFont.BOLD, 12);
let xaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(this.xAxis, axisTitleFont,
"Event Partition");
```

```
this.chartVu.addChartObject(xaxistitle);
let xgrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(this.xAxis, this.yAxis1,
QCChartTS.ChartConstants.X_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
this.chartVu.addChartObject(xgrid);
let thePlot1: QCChartTS.SimpleLinePlot
=QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform1, Dataset1, attrib1);
thePlot1.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot1);
let thePlot2: QCChartTS.SimpleLinePlot
=QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform2, Dataset2, attrib2);
thePlot2.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot2);
let thePlot3: QCChartTS.SimpleLinePlot
=QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform3, Dataset3, attrib3);
thePlot3.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot3);
let thePlot4: QCChartTS.SimpleLinePlot
=QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform4, Dataset4, attrib4);
thePlot4.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot4);
let thePlot5: QCChartTS.SimpleLinePlot
=QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform5, Dataset5, attrib5);
thePlot5.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot5);
```

[JavaScript]

```
let pTransform1;
let pTransform2;
let pTransform3;
let pTransform4;
let pTransform5;
//  Initialize datasets, coordinate system ranges
//  The x-scale range for pTransform1 to pTransform5 are all the same, 0 - 100
//  The y-scale range for pTransform1 to pTransform5 are all different
//  The plotting area for each pTransform is indentical, leaving a large open
//  to the left for extra axes.
pTransform1.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform2.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform3.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform4.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
pTransform5.setGraphBorderDiagonal(0.35, .15, .9, 0.65);
let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 2,
QCChartTS.ChartConstants.LS_SOLID);
let attrib2 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 2,
QCChartTS.ChartConstants.LS_SOLID);
let attrib3 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.GREEN, 2,
QCChartTS.ChartConstants.LS_SOLID);
let attrib4 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.ORANGE, 2,
QCChartTS.ChartConstants.LS_SOLID);
let attrib5 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.MAGENTA, 2,
QCChartTS.ChartConstants.LS_SOLID);


this.xAxis = QCChartTS.LinearAxis.newLinearAxis(this.pTransform1,
QCChartTS.ChartConstants.X_AXIS);
this.xAxis.setLineWidth(2);
this.chartVu.addChartObject(this.xAxis);
this.yAxis1 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis1.setAxisIntercept(0);
this.yAxis1.setChartObjAttributes(attrib1);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis1);
this.yAxis2 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform2,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis2.setAxisIntercept(-18);
this.yAxis2.setChartObjAttributes(attrib2);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis2);
```

```
this.yAxis3 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform3,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis3.setAxisIntercept(-35);
this.yAxis3.setChartObjAttributes(attrib3);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis3);
this.yAxis4 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform4,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis4.setAxisIntercept(-52);
this.yAxis4.setChartObjAttributes(attrib4);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis4);
this.yAxis5 = QCChartTS.LinearAxis.newLinearAxis(this.pTransform5,
QCChartTS.ChartConstants.Y_AXIS);
this.yAxis5.setAxisIntercept(this.xAxis.getAxisMax());
this.yAxis5.setAxisTickDir(QCChartTS.ChartConstants.AXIS_MAX);
this.yAxis5.setChartObjAttributes(attrib5);
//  axis color matches line color
this.chartVu.addChartObject(this.yAxis5);
let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.xAxis);
xAxisLab.setTextFont(theFont);
this.chartVu.addChartObject(xAxisLab);
let yAxisLab1 = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis1);
yAxisLab1.setTextFont(theFont);
yAxisLab1.setAxisLabelsFormat(QCChartTS.ChartConstants.BUSINESSFORMAT);
this.chartVu.addChartObject(yAxisLab1);
let yAxisLab2 = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis2);
yAxisLab2.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab2);
let yAxisLab3 = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis3);
yAxisLab3.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab3);
let yAxisLab4 = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis4);
yAxisLab4.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab4);
let yAxisLab5 = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis5);
yAxisLab5.setTextFont(theFont);
this.chartVu.addChartObject(yAxisLab5);
let axisTitleFont = QCChartTS.ChartFont.newChartFont3("Microsoft Sans Serif",
QCChartTS.ChartFont.BOLD, 12);
let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(this.xAxis, axisTitleFont, "Event Partition");
this.chartVu.addChartObject(xaxistitle);
let xgrid = QCChartTS.Grid.newGrid(this.xAxis, this.yAxis1, QCChartTS.ChartConstants.X_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
this.chartVu.addChartObject(xgrid);
let thePlot1 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform1, Dataset1, attrib1);
thePlot1.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot1);
let thePlot2 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform2, Dataset2, attrib2);
thePlot2.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot2);
let thePlot3 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform3, Dataset3, attrib3);
thePlot3.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot3);
let thePlot4 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform4, Dataset4, attrib4);
thePlot4.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot4);
let thePlot5 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(this.pTransform5, Dataset5, attrib5);
thePlot5.setFastClipMode(QCChartTS.ChartConstants.FASTCLIP_X);
this.chartVu.addChartObject(thePlot5);
```

3. Can I add new axes, text objects, plot objects, and images to a chart after it is already displayed; or must I create a new chart from scratch taking into account the additional objects?

There are two ways to add new objects to a chart. The first way is to create all objects when the chart is initially created, but disable the ones that you do not want to show up when the chart is initially

rendered. Enable the objects when you want them to show up. Use the chart objects s**etChartObjEnable** method to enable/disable the object. This is useful if you are creating an animated chart where you want the chart to sequence through a predefined series of steps. The second way you add new chart objects to the **ChartView** using the **ChartView.addChartObject** method. In both cases you need to call the **ChartView.updateDraw()** method after any changes are made.

The example below, extracted from the **CustomChartDataCursor** class, creates a new **Marker** object and **NumericLabel** object each time a mouse button clicked.

[TypeScript]

```
//  create marker object at place it at the nearest point
let amarker: QCChartTS.Marker = QCChartTS.Marker.newMarker(this.getChartObjScale(),
QCChartTS.ChartConstants.MARKER_BOX, nearestPoint.getX(), nearestPoint.getY(), 10,
QCChartTS.ChartConstants.PHYS_POS);
chartview.addChartObject(amarker);
this.rNumericLabelCntr += 1;
//  Add a numeric label the identifies the marker
this.pointLabel = QCChartTS.NumericLabel.newNumericLabel(this.getChartObjScale(),
this.textCoordsFont, this.rNumericLabelCntr, nearestPoint.getX(), nearestPoint.getY(),
QCChartTS.ChartConstants.PHYS_POS, QCChartTS.ChartConstants.DECIMALFORMAT, 0);
//  Nudge text to the right and up so that it does not write over marker
this.pointLabel.setTextNudge(5, -5);
chartview.addChartObject(this.pointLabel);
chartview.updateDraw();
```

[JavaScript]

```
//  create marker object at place it at the nearest point
let amarker = QCChartTS.Marker.newMarker(this.getChartObjScale(),
QCChartTS.ChartConstants.MARKER_BOX, nearestPoint.getX(), nearestPoint.getY(), 10,
QCChartTS.ChartConstants.PHYS_POS);
chartview.addChartObject(amarker);
this.rNumericLabelCntr += 1;
//  Add a numeric label the identifies the marker
this.pointLabel = QCChartTS.NumericLabel.newNumericLabel(this.getChartObjScale(),
this.textCoordsFont, this.rNumericLabelCntr, nearestPoint.getX(), nearestPoint.getY(),
QCChartTS.ChartConstants.PHYS_POS, QCChartTS.ChartConstants.DECIMALFORMAT, 0);
//  Nudge text to the right and up so that it does not write over marker
this.pointLabel.setTextNudge(5, -5);
chartview.addChartObject(this.pointLabel);
chartview.updateDraw();
```

4. How do you zoom charts that use multiple coordinate systems?

The **ChartZoom** class will zoom one or more simultaneous coordinate systems. The example program ZoomExamples zooms a chart that has one x-axis and five y-axes. Use the **ChartZoom** constructor that accepts an array of coordinate system objects.

5. How do you select a chart object and create a dialog panel that permits editing of that objects properties?

The **QCChart2D for JavaScript/TypeScript** library does not include predefined dialogs for editing chart object properties. The look, feel and details of such dialogs are application and culture specific and it is up to the application programmer to provide these.

You can add your own dialogs that edit the characteristics important to your end users. If you want to select the chart object by pressing a mouse button while the cursor is on the object, use the **FindObj** class. Override the **onMouseDown** method and invoke the appropriate dialog panel there. We use the BootStrap HTML library for creating pop-up browser dialogs, but you can use whatever technique you are familiar with. The following code is extracted from the EditChartExample example program.

[HTML]

```
<!DOCTYPE html>

<head>
    <meta charset="utf-8" />
    <title>Edit Chart Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
    <link href="https://gitcdn.github.io/bootstrap-toggle/2.2.2/css/bootstrap-toggle.min.css"
rel="stylesheet">
    <script src="https://gitcdn.github.io/bootstrap-toggle/2.2.2/js/bootstrap-toggle.min.js"></
script>
</head>

<body>
    <div id="buttondiv">
        <button type="button" id="editchart_menuitem">Simple Chart</button>
    </div>



    <div id="canvasdiv">
        <canvas id="chartCanvas1" width="800" height="600"></canvas>
    </div>
    <script type="module">


        import { EditChartExample } from './EditChartExample.js';

        var editchartex = new EditChartExample();


        function editchart() {
            editchartex.BuildEditChart("chartCanvas1");
        }

        // Since loading of script module is asynchronous, need to assign onclick event handlers
after module loaded.
        document.getElementById("editchart_menuitem").onclick = editchart;

    </script>

<script >

function showTextEditModal() {
    $("#charttextsetup_modal").modal();
  }
  function showLineAttribModal() {
    $("#lineattribsetup_modal").modal();
```

```
  }
</script>

    <br>
    <br>
    <div class="container" >
        <div class="modal fade" id="charttextsetup_modal" role="dialog"  >
            <div class="modal-dialog">
                <!-- Modal content-->
                <div class="modal-content">
                    <div class="modal-header">
                        <button type="button" class="close" data-dismiss="modal">&times;</button>
                        <h4 id="charttextsetup_title" class="modal-title">Text Dialog</h4>
                    </div>
                    <div class="form-group"

                    <div class="form-group"
                        style="display: flex; flex-direction: row; justify-content: flex-start;
align-items: flex-start">
                        <div class="row col-sm-2">
                            <label id="textinput_label" for="text_input"> Text</label>
                        </div>
                        <div class="col-sm-8">
                            <input class="form-control" type="text" name="textinput_name"
id="text_input">
                        </div>
                    </div>
                    <div class="form-group"
                        style="display: flex; flex-direction: row; justify-content: flex-start;
align-items: flex-start">
                        <div class="row col-sm-2">
                            <label id="sizeinput_label" for="size_input"> Size</label>
                        </div>
                        <div class="col-sm-2">
                            <input class="form-control" type="text" name="sizeinput_name"
id="textsize_input">
                        </div>
                    </div>

                    <div class="modal-footer">
                        <button type="button" class="btn btn-default" id="charttextsetup_cancel"
                            data-dismiss="modal">Cancel</button>
                        <button type="button" class="btn btn-sucess" id="charttextsetup_ok"
                            data-dismiss="modal">OK</button>
                    </div>
                </div>

            </div>
        </div>

    </div>
    <div class="container" >
        <div class="modal fade" id="lineattribsetup_modal" role="dialog"  >
            <div class="modal-dialog">
                <!-- Modal content-->
                <div class="modal-content">
                    <div class="modal-header">
                        <button type="button" class="close" data-dismiss="modal">&times;</button>
                        <h4 id="lineattribsetup_title" class="modal-title">Line Attributes</h4>
                    </div>
                    <div class="form-group"

                    <div class="form-group"
                        style="display: flex; flex-direction: row; justify-content: flex-start;
align-items: flex-start">
                        <div class="row col-sm-3">
                            <label id="linewidth_label" for="linewidth_input"> Line width</label>
                        </div>
                        <div class="col-sm-2">
```

```
                            <input class="form-control" type="text" name="linewidth_name"
id="linewidth_input">
                        </div>
                    </div>
                    <div class="form-group"
                        style="display: flex; flex-direction: row; justify-content: flex-start;
align-items: flex-start">
                        <div class="row col-sm-3">
                            <label id="linecolor_label" for="linecolor_input"> Color</label>
                        </div>
                        <div class="col-sm-3">
                            <input class="form-control" type="text" name="linecolorinput_name"
id="linecolor_input">
                        </div>
                    </div>

                    <div class="modal-footer">
                        <button type="button" class="btn btn-default" id="lineattribsetup_cancel"
                            data-dismiss="modal">Cancel</button>
                        <button type="button" class="btn btn-sucess" id="lineattribsetup_ok"
                            data-dismiss="modal">OK</button>
                    </div>
                </div>

            </div>
        </div>

    </div>

</body>


</html>
```

[TypeScript]

```
export class EditChartExample {
    xAxis: QCChartTS.LinearAxis = new QCChartTS.LinearAxis();

    yAxis: QCChartTS.LinearAxis = new QCChartTS.LinearAxis();

    chartvu: QCChartTS.ChartView | null = null;
    public constructor() {

    }


    public async  BuildEditChart(canvasid: string) {

        let htmlcanvas: QCChartTS.Canvas = <QCChartTS.Canvas>document.getElementById(canvasid);

        let chartVu: QCChartTS.ChartView = QCChartTS.ChartView.newChartView(htmlcanvas);
        if (!chartVu) return;
        else
            this.chartvu = chartVu;
        chartVu.setPreferredSize(800, 600);
        let theFont: QCChartTS.ChartFont;

        let numPoints: number = 95;
        let x1: number[] = new Array(numPoints);
        let y1: number[] = new Array(numPoints);
        let y2: number[] = new Array(numPoints);
        let y3: number[] = new Array(numPoints);
        let i: number;
        for (i = 0; i < numPoints; i++) {
            x1[i] = (i + 1);
            y1[i] = 20.0 + 50.0 * (1 - Math.exp(-x1[i] / 20.0));
            y2[i] = y1[i] + ((20 + 0.2 * x1[i]) * (0.6 -
QCChartTS.ChartSupport.getRandomDouble()));
```

```
            y3[i] = y1[i] + ((20 + 0.4 * x1[i]) * (0.4 -
QCChartTS.ChartSupport.getRandomDouble()));
        }

        theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
        let Dataset1: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("First",
x1, y1);
        let Dataset2: QCChartTS.SimpleDataset =
QCChartTS.SimpleDataset.newSimpleDataset("Second", x1, y2);
        let Dataset3: QCChartTS.SimpleDataset = QCChartTS.SimpleDataset.newSimpleDataset("Third",
x1, y3);
        let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
        pTransform1.autoScaleDatasetRoundMode(Dataset3, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
        pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.725);
        let background: QCChartTS.Background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
        chartVu.addChartObject(background);
        this.xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
        chartVu.addChartObject(this.xAxis);
        this.yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
        chartVu.addChartObject(this.yAxis);
        let xAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.xAxis);
        xAxisLab.setTextFont(theFont);
        chartVu.addChartObject(xAxisLab);
        let yAxisLab: QCChartTS.NumericAxisLabels =
QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis);
        yAxisLab.setTextFont(theFont);
        chartVu.addChartObject(yAxisLab);
        let titleFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 12);
        let yaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(this.yAxis,
titleFont, "Measurable work output");
        chartVu.addChartObject(yaxistitle);
        let xaxistitle: QCChartTS.AxisTitle = QCChartTS.AxisTitle.newAxisTitle(this.xAxis,
titleFont, "# MBAs/1000 employees");
        chartVu.addChartObject(xaxistitle);
        let xgrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(this.xAxis, this.yAxis,
QCChartTS.ChartConstants.X_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(xgrid);
        let ygrid: QCChartTS.Grid = QCChartTS.Grid.newGrid(this.xAxis, this.yAxis,
QCChartTS.ChartConstants.Y_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(ygrid);
        let attrib1: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
        attrib1.setFillColor(QCChartTS.ChartColor.BLUE);
        attrib1.setFillFlag(true);
        attrib1.setSymbolSize(10);
        let thePlot1: QCChartTS.SimpleScatterPlot =
QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset2,
QCChartTS.ChartConstants.CROSS, attrib1);
        chartVu.addChartObject(thePlot1);
        let attrib2: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.GREEN, 3,
QCChartTS.ChartConstants.LS_SOLID);
        let thePlot2: QCChartTS.SimpleLinePlot =
QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1, attrib2);
        chartVu.addChartObject(thePlot2);
        let attrib3: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID);
        attrib3.setFillColor(QCChartTS.ChartColor.RED);
        attrib3.setFillFlag(true);
        attrib3.setSymbolSize(6);
```

```
        let thePlot3: QCChartTS.SimpleScatterPlot =
QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset3,
QCChartTS.ChartConstants.CIRCLE, attrib3);
        chartVu.addChartObject(thePlot3);
        let legendFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 12);
        let legendAttributes: QCChartTS.ChartAttribute =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GRAY, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.LIGHTBLUE);
        legendAttributes.setFillFlag(true);
        legendAttributes.setLineFlag(true);
        let legend: QCChartTS.StandardLegend =
QCChartTS.StandardLegend.newStandardLegendPosWHAttribLayout(0.1, 0.875, 0.4, 0.075,
legendAttributes, QCChartTS.ChartConstants.HORIZ_DIR);
        legend.addLegendItemStringSymbolNumGraphObjFont("Energy Companies",
QCChartTS.ChartConstants.CROSS, thePlot1, legendFont);
        //  legend.addLegendItemStringSymbolNumGraphObjFont("Software Companies",
QCChartTS.ChartConstants.CIRCLE, thePlot3, legendFont);
        //  legend.addLegendItemStringSymbolNumGraphObjFont("Predicted",
QCChartTS.ChartConstants.LINE, thePlot2, legendFont);
        legend.setLegendItemUniformTextColor(QCChartTS.ChartColor.BLACK);
        chartVu.addChartObject(legend);
        let theTitleFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 14);
        let mainTitle: QCChartTS.ChartTitle =
QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1, theTitleFont, "Theoretical vs.
Experimental Data ");
        mainTitle.setTitleType(QCChartTS.ChartConstants.CHART_HEADER);
        mainTitle.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
        chartVu.addChartObject(mainTitle);
        let titleLine: QCChartTS.GPath = new QCChartTS.GPath();
        titleLine.addLineXY(0.1, 0.1, 0.9, 0.1);
        let titleLineShape: QCChartTS.ChartShape =
QCChartTS.ChartShape.newChartShape(pTransform1, titleLine,
QCChartTS.ChartConstants.NORM_GRAPH_POS, 0, 0, QCChartTS.ChartConstants.NORM_GRAPH_POS, 0);
        titleLineShape.setLineWidth(3);
        chartVu.addChartObject(titleLineShape);
        let theFooterFont: QCChartTS.ChartFont = QCChartTS.ChartFont.newChartFont3("Arial",
QCChartTS.ChartFont.BOLD, 12);
        let footer: QCChartTS.ChartTitle =
QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1, theFooterFont, "Scatter plots
usually display some form of sampled data.");
        footer.setTitleType(QCChartTS.ChartConstants.CHART_FOOTER);
        footer.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
        footer.setTitleOffset(8);
        chartVu.addChartObject(footer);
        chartVu.setResizeMode(QCChartTS.ChartConstants.AUTO_RESIZE_OBJECTS);


        let editObject: CustomFindObj = CustomFindObj.newCustomFindObj(this);
        editObject.addMouseListener();
        editObject.setEnable(true);

        chartVu.updateDraw();

    }


}


class CustomFindObj extends QCChartTS.FindObj {
    currentObj: EditChartExample | null = null;

    public constructor() {
        super();
    }


    public static newCustomFindObj(component: EditChartExample): CustomFindObj {
        let result: CustomFindObj = new CustomFindObj();
```

```
        result.currentObj = component;
        result.ChartObjComponent = component.chartvu;
        return result;
    }

    public invokeLineDialog(graphobj: QCChartTS.GraphObj | null) {
        let textdlgsetup: LineAttributesDialogSetup =
LineAttributesDialogSetup.newLineAttributesDialogSetup(graphobj);
        (window as any).showLineAttribModal();
    }

    public invokeTextDialog(textobj: QCChartTS.ChartText | null) {

        let textdlgsetup: TextDialogSetup = TextDialogSetup.newTextDialogSetup(textobj);
        (window as any).showTextEditModal();

    }

    public onMouseDown(mouseevent: MouseEvent) {
        super.onMouseDown(mouseevent);
        let selectedObj: QCChartTS.GraphObj | null = this.SelectedObject;
        if (selectedObj && this.currentObj) {
            // Check for a specific object
            if ((selectedObj == this.currentObj.xAxis) ||
                (selectedObj == this.currentObj.yAxis) ||
                // or check for for all classes inheriting from a specific type
                (QCChartTS.ChartSupport.isKindOf(selectedObj, "SimpleLinePlot")) ||
                // or Check for a specific object type
                (QCChartTS.ChartSupport.isKindOf(selectedObj, "SimpleScatterPlot"))) {
                this.invokeLineDialog(selectedObj);
            } else
                if (QCChartTS.ChartSupport.isKindOf(selectedObj, "ChartText"))
                    this.invokeTextDialog(<QCChartTS.ChartText>this.SelectedObject);
            if (this.ChartObjComponent)
                this.ChartObjComponent.updateDraw();
        }
    }
}

export class TextDialogSetup {

    public OKButton: HTMLInputElement =
<HTMLInputElement>document.getElementById("charttextsetup_ok");
    public cancelButton: HTMLInputElement =
<HTMLInputElement>document.getElementById("charttextsetup_cancel");
    public dialogText: HTMLInputElement =
<HTMLInputElement>document.getElementById("text_input");
    public dialogTextSize: HTMLInputElement =
<HTMLInputElement>document.getElementById("textsize_input");

    public textString: string | null = "";
    public textSize: number = 12;
    public textObj: QCChartTS.ChartText | null = null;

    // Need to save these references for consistant use with addEventListener and
removeEventListener
    public fOK: any = (e: Event) =>  this.setupOKAction(e);
    public fCancel: any = (e: Event) =>  this.setupCancelAction(e);

    public static stringToInt(s: string, defaultvalue: number): number {
        let result: number = parseInt(s, 10);
        if (isNaN(result)) {
            result = defaultvalue;
        }
        return result;
    }


    public copyChartToDialog() {
        if (this.textObj) {
            this.textString = this.textObj.TextString;
```

```
            this.textSize = this.textObj.TextFont.getFontSize();
        }
    }



    public copyDialogToChart() {
        if (this.textObj && this.textString) {
            this.textObj.TextString = this.textString;
            let f: QCChartTS.ChartFont = this.textObj.TextFont;
            this.textObj.TextFont = QCChartTS.ChartFont.newChartFont3(f.fontName, f.fontStyle,
this.textSize);

        }
    }

    public copyControlsToDialog() {
        this.textString = this.dialogText.value;
        this.textSize = TextDialogSetup.stringToInt(this.dialogTextSize.value, 12);

    }

    public copyDialogToControls() {
        if (this.textString) {
            this.dialogText.value = this.textString;
            this.dialogTextSize.value = this.textSize.toString();

        }
    }

    public addEventListeners()
    {

       this.OKButton.addEventListener("click",this.fOK);
       this.cancelButton.addEventListener("click",this.fCancel);

    }

    public removeEventListeners()
    {
        this.OKButton.removeEventListener("click", this.fOK);
        this.cancelButton.removeEventListener("click",  this.fCancel);
    }

    public OKButtonClicked(e: Event) {
        this.copyControlsToDialog();
        this.copyDialogToChart();
        if (this.textObj && this.textObj.ChartObjComponent)
            this.textObj.ChartObjComponent.updateDraw();
        this.removeEventListeners();

    }

    public cancelButtonClicked(e: Event) {
        this.removeEventListeners();

    }



    setupOKAction(e: Event) {
        this.OKButtonClicked(e);
    }
    setupCancelAction(e: Event) {
        this.cancelButtonClicked(e);

    }


    public onDialogLoad() {
```

```
        this.copyChartToDialog();
        this.copyDialogToControls();
    }

    initDefaultsTextDialogSetup() {

       this.addEventListeners();
        // jquery call assigning event listener to bootstrap modal dialog
        //    (window as any)['$'](this.spcGeneralSetupModalDialog).on("shown.bs.modal", (e:
Event) => this.onDialogLoad());


    }

    public constructor() {

    }

    public static newTextDialogSetup(textobj: QCChartTS.ChartText | null): TextDialogSetup {
        let result: TextDialogSetup = new TextDialogSetup();
        if (textobj) {
            result.textObj = textobj;
            result.initDefaultsTextDialogSetup();
            result.copyChartToDialog();
            result.copyDialogToControls();
        }

        return result;
    }
}


export class LineAttributesDialogSetup {

    public OKButton: HTMLInputElement =
<HTMLInputElement>document.getElementById("lineattribsetup_ok");
    public cancelButton: HTMLInputElement =
<HTMLInputElement>document.getElementById("lineattribsetup_cancel");
    public dialogLineWidth: HTMLInputElement =
<HTMLInputElement>document.getElementById("linewidth_input");
    public dialogLineColor: HTMLInputElement =
<HTMLInputElement>document.getElementById("linecolor_input");

    public lineWidth: number = 1;
    public lineColor: number = QCChartTS.ChartColor.RED;
    public graphObj: QCChartTS.GraphObj | null = null;

    public fOK: any = (e: Event) =>  this.setupOKAction(e);
    public fCancel: any = (e: Event) =>  this.setupCancelAction(e);

    public static stringToInt(s: string, defaultvalue: number): number {
        let result: number = parseInt(s, 10);
        if (isNaN(result)) {
            result = defaultvalue;
        }
        return result;
    }

    public static stringToColor(s: string, defaultvalue: number): number {
        let result: number = parseInt(s);
        if (isNaN(result)) {
            result = defaultvalue;
        }
        return result;
    }

    public static decimalToHexString(d: number): string {
        let result:string = ""
        if (d < 0) {
            d = 0xFFFFFFFF + d + 1;
        }
```

```
        result = "0x" + d.toString(16).toUpperCase();
        return result;
    }

    public copyChartToDialog() {
        if (this.graphObj) {
            this.lineWidth = this.graphObj.getLineWidth();
            this.lineColor = this.graphObj.getLineColor();
        }
    }



    public copyDialogToChart() {
        if (this.graphObj) {
            this.graphObj.setLineColor(this.lineColor);
            this.graphObj.setLineWidth(this.lineWidth);

        }
    }

    public copyControlsToDialog() {
        this.lineWidth = LineAttributesDialogSetup.stringToInt(this.dialogLineWidth.value, 1);
        this.lineColor = LineAttributesDialogSetup.stringToColor(this.dialogLineColor.value,
QCChartTS.ChartColor.RED);

    }

    public copyDialogToControls() {

        this.dialogLineWidth.value = this.lineWidth.toString();
        this.dialogLineColor.value = LineAttributesDialogSetup.decimalToHexString
(this.lineColor);

    }

    public addEventListeners()
    {
        this.OKButton.addEventListener("click",this.fOK);
        this.cancelButton.addEventListener("click",this.fCancel);

    }

    public removeEventListeners()
    {
        this.OKButton.removeEventListener("click", this.fOK);
        this.cancelButton.removeEventListener("click",  this.fCancel);
    }

    public OKButtonClicked(e: Event) {
        this.copyControlsToDialog();
        this.copyDialogToChart();
        if (this.graphObj && this.graphObj.ChartObjComponent)
            this.graphObj.ChartObjComponent.updateDraw();
        this.removeEventListeners();
    }

    public cancelButtonClicked(e: Event) {
        this.removeEventListeners();
    }



    setupOKAction(e: Event) {
        this.OKButtonClicked(e);
    }
    setupCancelAction(e: Event) {
        this.cancelButtonClicked(e);

    }
```

```
    public onDialogLoad() {

        this.copyChartToDialog();
        this.copyDialogToControls();
    }

    initDefaultsLineAttributesDialogSetup() {

      this.addEventListeners();
         // jquery call assigning event listener to bootstrap modal dialog
        //    (window as any)['$'](this.spcGeneralSetupModalDialog).on("shown.bs.modal", (e:
Event) => this.onDialogLoad());


    }

    public constructor() {

    }

    public static newLineAttributesDialogSetup(graphobj: QCChartTS.GraphObj | null):
LineAttributesDialogSetup {
        let result: LineAttributesDialogSetup = new LineAttributesDialogSetup();
        if (graphobj) {
            result.graphObj = graphobj;
            result.initDefaultsLineAttributesDialogSetup();
            result.copyChartToDialog();
            result.copyDialogToControls();
        }

        return result;
    }

}
```

[JavaScript]

```
export class EditChartExample {
    constructor() {
        this.xAxis = new QCChartTS.LinearAxis();
        this.yAxis = new QCChartTS.LinearAxis();
        this.chartvu = null;
    }
    async BuildEditChart(canvasid) {
        let htmlcanvas = document.getElementById(canvasid);
        let chartVu = QCChartTS.ChartView.newChartView(htmlcanvas);
        if (!chartVu)
            return;
        else
            this.chartvu = chartVu;
        chartVu.setPreferredSize(800, 600);
        let theFont;
        let numPoints = 95;
        let x1 = new Array(numPoints);
        let y1 = new Array(numPoints);
        let y2 = new Array(numPoints);
        let y3 = new Array(numPoints);
        let i;
        for (i = 0; i < numPoints; i++) {
            x1[i] = (i + 1);
            y1[i] = 20.0 + 50.0 * (1 - Math.exp(-x1[i] / 20.0));
            y2[i] = y1[i] + ((20 + 0.2 * x1[i]) * (0.6 -
QCChartTS.ChartSupport.getRandomDouble()));
            y3[i] = y1[i] + ((20 + 0.4 * x1[i]) * (0.4 -
QCChartTS.ChartSupport.getRandomDouble()));
        }
        theFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
        let Dataset1 = QCChartTS.SimpleDataset.newSimpleDataset("First", x1, y1);
```

```
        let Dataset2 = QCChartTS.SimpleDataset.newSimpleDataset("Second", x1, y2);
        let Dataset3 = QCChartTS.SimpleDataset.newSimpleDataset("Third", x1, y3);
        let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LINEAR_SCA
LE, QCChartTS.ChartConstants.LINEAR_SCALE);
        pTransform1.autoScaleDatasetRoundMode(Dataset3, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
        pTransform1.setGraphBorderDiagonal(0.15, .15, .90, 0.725);
        let background = QCChartTS.Background.newBackground(pTransform1,
QCChartTS.ChartConstants.PLOT_BACKGROUND, QCChartTS.ChartColor.WHITE);
        chartVu.addChartObject(background);
        this.xAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
        chartVu.addChartObject(this.xAxis);
        this.yAxis = QCChartTS.LinearAxis.newLinearAxis(pTransform1,
QCChartTS.ChartConstants.Y_AXIS);
        chartVu.addChartObject(this.yAxis);
        let xAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.xAxis);
        xAxisLab.setTextFont(theFont);
        chartVu.addChartObject(xAxisLab);
        let yAxisLab = QCChartTS.NumericAxisLabels.newNumericAxisLabels(this.yAxis);
        yAxisLab.setTextFont(theFont);
        chartVu.addChartObject(yAxisLab);
        let titleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD, 12);
        let yaxistitle = QCChartTS.AxisTitle.newAxisTitle(this.yAxis, titleFont, "Measurable work
output");
        chartVu.addChartObject(yaxistitle);
        let xaxistitle = QCChartTS.AxisTitle.newAxisTitle(this.xAxis, titleFont, "# MBAs/1000
employees");
        chartVu.addChartObject(xaxistitle);
        let xgrid = QCChartTS.Grid.newGrid(this.xAxis, this.yAxis,
QCChartTS.ChartConstants.X_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(xgrid);
        let ygrid = QCChartTS.Grid.newGrid(this.xAxis, this.yAxis,
QCChartTS.ChartConstants.Y_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
        chartVu.addChartObject(ygrid);
        let attrib1 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.BLUE, 1,
QCChartTS.ChartConstants.LS_SOLID);
        attrib1.setFillColor(QCChartTS.ChartColor.BLUE);
        attrib1.setFillFlag(true);
        attrib1.setSymbolSize(10);
        let thePlot1 = QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset2,
QCChartTS.ChartConstants.CROSS, attrib1);
        chartVu.addChartObject(thePlot1);
        let attrib2 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.GREEN, 3,
QCChartTS.ChartConstants.LS_SOLID);
        let thePlot2 = QCChartTS.SimpleLinePlot.newSimpleLinePlot(pTransform1, Dataset1,
attrib2);
        chartVu.addChartObject(thePlot2);
        let attrib3 = QCChartTS.ChartAttribute.newChartAttribute3(QCChartTS.ChartColor.RED, 1,
QCChartTS.ChartConstants.LS_SOLID);
        attrib3.setFillColor(QCChartTS.ChartColor.RED);
        attrib3.setFillFlag(true);
        attrib3.setSymbolSize(6);
        let thePlot3 = QCChartTS.SimpleScatterPlot.newSimpleScatterPlot(pTransform1, Dataset3,
QCChartTS.ChartConstants.CIRCLE, attrib3);
        chartVu.addChartObject(thePlot3);
        let legendFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD,
12);
        let legendAttributes =
QCChartTS.ChartAttribute.newChartAttribute4(QCChartTS.ChartColor.GRAY, 1,
QCChartTS.ChartConstants.LS_SOLID, QCChartTS.ChartColor.LIGHTBLUE);
        legendAttributes.setFillFlag(true);
        legendAttributes.setLineFlag(true);
        let legend = QCChartTS.StandardLegend.newStandardLegendPosWHAttribLayout(0.1, 0.875, 0.4,
0.075, legendAttributes, QCChartTS.ChartConstants.HORIZ_DIR);
        legend.addLegendItemStringSymbolNumGraphObjFont("Energy Companies",
QCChartTS.ChartConstants.CROSS, thePlot1, legendFont);
        //  legend.addLegendItemStringSymbolNumGraphObjFont("Software Companies",
QCChartTS.ChartConstants.CIRCLE, thePlot3, legendFont);
```

```
        //  legend.addLegendItemStringSymbolNumGraphObjFont("Predicted",
QCChartTS.ChartConstants.LINE, thePlot2, legendFont);
        legend.setLegendItemUniformTextColor(QCChartTS.ChartColor.BLACK);
        chartVu.addChartObject(legend);
        let theTitleFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD,
14);
        let mainTitle = QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1,
theTitleFont, "Theoretical vs. Experimental Data ");
        mainTitle.setTitleType(QCChartTS.ChartConstants.CHART_HEADER);
        mainTitle.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
        chartVu.addChartObject(mainTitle);
        let titleLine = new QCChartTS.GPath();
        titleLine.addLineXY(0.1, 0.1, 0.9, 0.1);
        let titleLineShape = QCChartTS.ChartShape.newChartShape(pTransform1, titleLine,
QCChartTS.ChartConstants.NORM_GRAPH_POS, 0, 0, QCChartTS.ChartConstants.NORM_GRAPH_POS, 0);
        titleLineShape.setLineWidth(3);
        chartVu.addChartObject(titleLineShape);
        let theFooterFont = QCChartTS.ChartFont.newChartFont3("Arial", QCChartTS.ChartFont.BOLD,
12);
        let footer = QCChartTS.ChartTitle.newChartTitleCoordsFontString(pTransform1,
theFooterFont, "Scatter plots usually display some form of sampled data.");
        footer.setTitleType(QCChartTS.ChartConstants.CHART_FOOTER);
        footer.setTitlePosition(QCChartTS.ChartConstants.CENTER_GRAPH);
        footer.setTitleOffset(8);
        chartVu.addChartObject(footer);
        chartVu.setResizeMode(QCChartTS.ChartConstants.AUTO_RESIZE_OBJECTS);
        let editObject = CustomFindObj.newCustomFindObj(this);
        editObject.addMouseListener();
        editObject.setEnable(true);
        chartVu.updateDraw();
    }
}
class CustomFindObj extends QCChartTS.FindObj {
    constructor() {
        super();
        this.currentObj = null;
    }
    static newCustomFindObj(component) {
        let result = new CustomFindObj();
        result.currentObj = component;
        result.ChartObjComponent = component.chartvu;
        return result;
    }
    invokeLineDialog(graphobj) {
        let textdlgsetup = LineAttributesDialogSetup.newLineAttributesDialogSetup(graphobj);
        window.showLineAttribModal();
    }
    invokeTextDialog(textobj) {
        let textdlgsetup = TextDialogSetup.newTextDialogSetup(textobj);
        window.showTextEditModal();
    }
    onMouseDown(mouseevent) {
        super.onMouseDown(mouseevent);
        let selectedObj = this.SelectedObject;
        if (selectedObj && this.currentObj) {
            // Check for a specific object
            if ((selectedObj == this.currentObj.xAxis) ||
                (selectedObj == this.currentObj.yAxis) ||
                // or check for for all classes inheriting from a specific type
                (QCChartTS.ChartSupport.isKindOf(selectedObj, "SimpleLinePlot")) ||
                // or Check for a specific object type
                (QCChartTS.ChartSupport.isKindOf(selectedObj, "SimpleScatterPlot"))) {
                this.invokeLineDialog(selectedObj);
            }
            else if (QCChartTS.ChartSupport.isKindOf(selectedObj, "ChartText"))
                this.invokeTextDialog(this.SelectedObject);
            if (this.ChartObjComponent)
                this.ChartObjComponent.updateDraw();
        }
    }
}
```

```
export class TextDialogSetup {
    constructor() {
        this.OKButton = document.getElementById("charttextsetup_ok");
        this.cancelButton = document.getElementById("charttextsetup_cancel");
        this.dialogText = document.getElementById("text_input");
        this.dialogTextSize = document.getElementById("textsize_input");
        this.textString = "";
        this.textSize = 12;
        this.textObj = null;
        this.fOK = (e) => this.setupOKAction(e);
        this.fCancel = (e) => this.setupCancelAction(e);
    }
    static stringToInt(s, defaultvalue) {
        let result = parseInt(s, 10);
        if (isNaN(result)) {
            result = defaultvalue;
        }
        return result;
    }
    copyChartToDialog() {
        if (this.textObj) {
            this.textString = this.textObj.TextString;
            this.textSize = this.textObj.TextFont.getFontSize();
        }
    }
    copyDialogToChart() {
        if (this.textObj && this.textString) {
            this.textObj.TextString = this.textString;
            let f = this.textObj.TextFont;
            this.textObj.TextFont = QCChartTS.ChartFont.newChartFont3(f.fontName, f.fontStyle,
this.textSize);
        }
    }
    copyControlsToDialog() {
        this.textString = this.dialogText.value;
        this.textSize = TextDialogSetup.stringToInt(this.dialogTextSize.value, 12);
    }
    copyDialogToControls() {
        if (this.textString) {
            this.dialogText.value = this.textString;
            this.dialogTextSize.value = this.textSize.toString();
        }
    }
    addEventListeners() {
        this.OKButton.addEventListener("click", this.fOK);
        this.cancelButton.addEventListener("click", this.fCancel);
    }
    removeEventListeners() {
        this.OKButton.removeEventListener("click", this.fOK);
        this.cancelButton.removeEventListener("click", this.fCancel);
    }
    OKButtonClicked(e) {
        this.copyControlsToDialog();
        this.copyDialogToChart();
        if (this.textObj && this.textObj.ChartObjComponent)
            this.textObj.ChartObjComponent.updateDraw();
        this.removeEventListeners();
    }
    cancelButtonClicked(e) {
        this.removeEventListeners();
    }
    setupOKAction(e) {
        this.OKButtonClicked(e);
    }
    setupCancelAction(e) {
        this.cancelButtonClicked(e);
    }
    onDialogLoad() {
        this.copyChartToDialog();
        this.copyDialogToControls();
    }
```

```
    initDefaultsTextDialogSetup() {
        this.addEventListeners();
        // jquery call assigning event listener to bootstrap modal dialog
        //    (window as any)['$'](this.spcGeneralSetupModalDialog).on("shown.bs.modal", (e:
Event) => this.onDialogLoad());
    }
    static newTextDialogSetup(textobj) {
        let result = new TextDialogSetup();
        if (textobj) {
            result.textObj = textobj;
            result.initDefaultsTextDialogSetup();
            result.copyChartToDialog();
            result.copyDialogToControls();
        }
        return result;
    }
}
export class LineAttributesDialogSetup {
    constructor() {
        this.OKButton = document.getElementById("lineattribsetup_ok");
        this.cancelButton = document.getElementById("lineattribsetup_cancel");
        this.dialogLineWidth = document.getElementById("linewidth_input");
        this.dialogLineColor = document.getElementById("linecolor_input");
        this.lineWidth = 1;
        this.lineColor = QCChartTS.ChartColor.RED;
        this.graphObj = null;
        this.fOK = (e) => this.setupOKAction(e);
        this.fCancel = (e) => this.setupCancelAction(e);
    }
    static stringToInt(s, defaultvalue) {
        let result = parseInt(s, 10);
        if (isNaN(result)) {
            result = defaultvalue;
        }
        return result;
    }
    static stringToColor(s, defaultvalue) {
        let result = parseInt(s);
        if (isNaN(result)) {
            result = defaultvalue;
        }
        return result;
    }
    static decimalToHexString(d) {
        let result = "";
        if (d < 0) {
            d = 0xFFFFFFFF + d + 1;
        }
        result = "0x" + d.toString(16).toUpperCase();
        return result;
    }
    copyChartToDialog() {
        if (this.graphObj) {
            this.lineWidth = this.graphObj.getLineWidth();
            this.lineColor = this.graphObj.getLineColor();
        }
    }
    copyDialogToChart() {
        if (this.graphObj) {
            this.graphObj.setLineColor(this.lineColor);
            this.graphObj.setLineWidth(this.lineWidth);
        }
    }
    copyControlsToDialog() {
        this.lineWidth = LineAttributesDialogSetup.stringToInt(this.dialogLineWidth.value, 1);
        this.lineColor = LineAttributesDialogSetup.stringToColor(this.dialogLineColor.value,
QCChartTS.ChartColor.RED);
    }
    copyDialogToControls() {
        this.dialogLineWidth.value = this.lineWidth.toString();
```

```
        this.dialogLineColor.value =
LineAttributesDialogSetup.decimalToHexString(this.lineColor);
    }
    addEventListeners() {
        this.OKButton.addEventListener("click", this.fOK);
        this.cancelButton.addEventListener("click", this.fCancel);
    }
    removeEventListeners() {
        this.OKButton.removeEventListener("click", this.fOK);
        this.cancelButton.removeEventListener("click", this.fCancel);
    }
    OKButtonClicked(e) {
        this.copyControlsToDialog();
        this.copyDialogToChart();
        if (this.graphObj && this.graphObj.ChartObjComponent)
            this.graphObj.ChartObjComponent.updateDraw();
        this.removeEventListeners();
    }
    cancelButtonClicked(e) {
        this.removeEventListeners();
    }
    setupOKAction(e) {
        this.OKButtonClicked(e);
    }
    setupCancelAction(e) {
        this.cancelButtonClicked(e);
    }
    onDialogLoad() {
        this.copyChartToDialog();
        this.copyDialogToControls();
    }
    initDefaultsLineAttributesDialogSetup() {
        this.addEventListeners();
        // jquery call assigning event listener to bootstrap modal dialog
        //    (window as any)['$'](this.spcGeneralSetupModalDialog).on("shown.bs.modal", (e:
Event) => this.onDialogLoad());
    }
    static newLineAttributesDialogSetup(graphobj) {
        let result = new LineAttributesDialogSetup();
        if (graphobj) {
            result.graphObj = graphobj;
            result.initDefaultsLineAttributesDialogSetup();
            result.copyChartToDialog();
            result.copyDialogToControls();
        }
        return result;
    }
}
```

The EditChartExample **LineDialogSetup** and **TextDialogSetup** classes reference the HTML elements of the BootStrap blocks in the EditChartExample.html file.

6. How do you handle missing data points in a chart?

There are two ways to handle missing, or bad data. The first is to mark the data point in the dataset invalid, using the datasets s**etValidData** method. The second is to set the x- and/or y- value of the bad data point to the designated bad data value, QCChartTS.ChartConstants**.rBadDataValue**. Currently this value is set equal to the value of Number.Max_Value. Either method will prevent the data point from being displayed in a chart. If the bad data value is part of a line plot, a gap will appear in the line plot at that point. Bad data points are not deleted from a dataset.

7. How do you update a chart in real-time?

In general, real-time updates involve adding new objects to a chart, or modifying existing objects that are already in the chart. Once the object is added or changed, call the **ChartView.updateDraw()** method to force the chart to update using the new values. Objects can be added or modified based on some external event, or in response to a timer event created using **the HTML window.setInterval** timer. Make all changes for a given event and call the **ChartView.updateDraw** method once. The position of most **GraphObj** derived objects is set or modified using one of the objects s**etLocation** methods. New data points can be added to an existing dataset using one of the datasets **addDataPoint**, **addTimeDataPoint**, **addGroupDataPoints** or a**ddTimeGroupDataPoints** methods. **ChartPlot** derived objects that use datasets will update to reflect the new values when the **ChartView.updateDraw** method is called. If the coordinates of the new data points are outside of the x- and y-limits of the current coordinate system it may be necessary to rescale the coordinate system so that the new points show up; otherwise the new data points will be clipped. The new scale values can be set explicitly, or calculated using one of the auto-scale methods. The program DynamicCharts for different ways to update a chart in realtime.

If you want to change points in an existing dataset, but not the size of the dataset, call the datasets appropriate **setXDataValue**, **setYDataValue**, or **setDataPoint** methods. The dataset has its own copy of the data so you must change these values, not the original values you used to initialize the dataset. If you plan to change every value in the dataset, you can do that point by point, or create a new dataset and swap that in for the old dataset using the plot objects **setDataset** or **setGroupDataset** method. Call the **ChartView.updateDraw** method to force the chart to update using the new values.

8. How do I prevent flicker when updating my charts on real-time?

Flicker can result because of erasing and redrawing all or part of a chart in the current display buffer. The **ChartView** class does the actual work of rendering a chart image to the underlying **Canvas** display object. It is the function of the browser to implement the **Canvas**. It appears that the **Canvas** object in all of the major browsers use and efficient form of double buffering which minimized screen flicker, so nothing further needs to be done.

9. How do you implement drill down, or data tool tips in a chart?

Implementing drill down or tool tips consists of three major parts:

- Trapping a mouse event and determining the mouse cursor position in device and physical coordinates.
- Identifying the chart object that intersects the mouse event.

- Displaying appropriate information about the chart object.

There are many classes that aid in one or more of these functions. The **MouseListener** class will trap a mouse event in the chart view. The **FindObj** class will filter and return the chart object, if any, that intersects the mouse cursor when a mouse button is pressed. The **MoveObj** class will filter, select and move a chart object as the mouse is dragged across the chart. The **DataToolTip** class will find the data point in a chart nearest the mouse cursor and display xy information about the data point as a popup **ChartText** display. The **DataToolTip** can also be customized for the display of custom information about the selected data point. It only takes a one line to add a simple y-value tool tip to an existing chart.

[TypeScript]

```
let datatooltip: QCChartTS.DataToolTip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
```

[JavaScript]

```
let datatooltip = QCChartTS.DataToolTip.newDataToolTip(chartVu);
```

You will find examples of data tooltips in almost every example. Every chart in the SimpleLinePlots example includes a data tooltip. Chapter 16 covers data tooltips in more detail..

10. I do not want to my graph to auto-scale. How do I setup the graph axes for a specific range?

Auto-scaling has two parts. The first is the auto-scaling of the coordinate system based on one or more datasets. The second part is the auto-scaling of the axes that reside in the coordinate system. Manually scale the coordinate system and axes by calling the appropriate constructors. For example:

[TypeScript]

```
let xMin: Date  = new Date(1996, QCChartTS.ChartConstants.FEBRUARY,  5);
let xMax: Date  = new Date(2002, QCChartTS.ChartConstants.JANUARY,  5);
let yMin: number = 0;
let yMax: number =  105;

let simpleTimeScale: QCChartTS.TimeCoordinates  =
QCChartTS.TimeCoordinates.newTimeCoordinates(xMin, yMin, xMax, yMax);
// Create the time axis (x-axis is assumed)
let xAxis: QCChartTS.TimeAxis  = QCChartTS.TimeAxis.newTimeAxis(simpleTimeScale);
// Create the linear y-axis
let yAxis: QCChartTS.LinearAxis  = QCChartTS.LinearAxis.newLinearAxis(simpleTimeScale,
QCChartTS.ChartConstants.Y_AXIS);


// Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
```

[JavaScript]

```
let xMin = new Date(1996, QCChartTS.ChartConstants.FEBRUARY, 5);
let xMax = new Date(2002, QCChartTS.ChartConstants.JANUARY, 5);
let yMin = 0;
let yMax = 105;
let simpleTimeScale = QCChartTS.TimeCoordinates.newTimeCoordinates(xMin, yMin, xMax, yMax);
// Create the time axis (x-axis is assumed)
let xAxis = QCChartTS.TimeAxis.newTimeAxis(simpleTimeScale);
// Create the linear y-axis
let yAxis = QCChartTS.LinearAxis.newLinearAxis(simpleTimeScale, QCChartTS.ChartConstants.Y_AXIS);
// Add the x- and y-axes to the chartVu object
chartVu.addChartObject(xAxis);
chartVu.addChartObject(yAxis);
```

The documentation for the various coordinate system and axis classes includes examples of manual scaling.

11. How do I update my data, and auto-rescale the chart scales and axes to reflect the new data, after it has already been drawn?

Updating data was discussed in FAQ # 6. If you want the chart to rescale based on the new data, call the appropriate coordinate systems auto-scale method, followed by the auto-axis methods of all related axes. Then call the **ChartView.updateDraw** method. For example:

[TypeScript]

```
let Dataset1: QCChartTS.TimeSimpleDataset =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", x1, y1);
let simpleTimeCoordinates: QCChartTS.TimeCoordinates = new QCChartTS.TimeCoordinates();
simpleTimeCoordinates.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);

let xAxis: QCChartTS.TimeAxis = QCChartTS.TimeAxis.newTimeAxis(simpleTimeCoordinates);
let yAxis: QCChartTS.LinearAxis = QCChartTS.LinearAxis.newLinearAxis(simpleTimeCoordinates,
QCChartTS.ChartConstants.Y_AXIS);

.
.
.

//  The following code would be in the code handling the rescale event
//  Rescale chart based on a modified Dataset1 datset
simpleTimeCoordinates.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
xAxis.calcAutoAxis();
yAxis.calcAutoAxis();
//  Redraw the chart using the rescaled coordinate system and axes
chartVu.updateDraw();
```

[JavaScript]

```
let Dataset1 = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Sales", x1, y1);
let simpleTimeCoordinates = new QCChartTS.TimeCoordinates();
simpleTimeCoordinates.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
```

```
let xAxis = QCChartTS.TimeAxis.newTimeAxis(simpleTimeCoordinates);
let yAxis = QCChartTS.LinearAxis.newLinearAxis(simpleTimeCoordinates,
QCChartTS.ChartConstants.Y_AXIS);
.
.
.

//  The following code would be in the code handling the rescale event
//  Rescale chart based on a modified Dataset1 datset
simpleTimeCoordinates.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
xAxis.calcAutoAxis();
yAxis.calcAutoAxis();
//  Redraw the chart using the rescaled coordinate system and axes
chartVu.updateDraw();
```

12. When I use the auto-scale and auto-axis routines my semi-log chart has the log axis scaled using powers of 10 (1, 10,100, 1000, etc.) as the starting and ending values, or as the major tick interval for labeling. How do I make my log graphs start at 20 and end at 50,000, with major tick marks at 20, 200, 2000 and 20000?

The auto-scale routines for logarithmic coordinate systems will always select a power of 10 for the minimum and maximum value of the scale. You can use the auto-scale routine and then override the minimum and/or maximum values for the logarithmic scale. The default **LogAxis** constructor will pick up on the minimum of the coordinate system and use that as the axis tick mark origin. Or you can leave the coordinate system unchanged, and change the starting point of the axis tick marks using the axis **SetAxisTickOrigin** method. The example below is derived from the Logarithmic example code.

[TypeScript]

```
let Dataset1: QCChartTS.GroupDataset = QCChartTS.GroupDataset.newGroupDataset("First", x1, y1);
let pTransform1: QCChartTS.CartesianCoordinates =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LOG_SCALE,
QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartX(20);
let xAxis: QCChartTS.LogAxis =  QCChartTS.LogAxis.newLogAxis(pTransform1,
QCChartTS.ChartConstants.X_AXIS);
xAxis.setAxisTickOrigin(20);
chartVu.addChartObject(xAxis);
```

[JavaScript]

```
let Dataset1 = QCChartTS.GroupDataset.newGroupDataset("First", x1, y1);
let pTransform1 =
QCChartTS.CartesianCoordinates.newCartesianCoordinatesScaleXY(QCChartTS.ChartConstants.LOG_SCALE,
QCChartTS.ChartConstants.LINEAR_SCALE);
pTransform1.autoScaleDatasetRoundMode(Dataset1, QCChartTS.ChartConstants.AUTOAXES_FAR,
QCChartTS.ChartConstants.AUTOAXES_FAR);
pTransform1.setScaleStartX(20);
let xAxis = QCChartTS.LogAxis.newLogAxis(pTransform1, QCChartTS.ChartConstants.X_AXIS);
xAxis.setAxisTickOrigin(20);
chartVu.addChartObject(xAxis);
```

13. How do I create and use custom, multi-line string labels as the axis labels for my graph?

The **StringAxisLabels** class should be used to create multi-line axis labels. Insert the "\n" new line character to add additional lines to each string used to define the string axis labels. The example below is from the AxisLabels example program.

[TypeScript]

```
let xstringlabels: string [] =
[
    "",
    "Western"+"\n"+"Sales"+"\n"+"Region",
    "Eastern"+"\n"+"Sales"+"\n"+"Region",
    "Southern"+"\n"+"Sales"+"\n"+"Region",
    "Northern"+"\n"+"Sales"+"\n"+"Region"
];

let xAxisLab5: QCChartTS.StringAxisLabels  =
QCChartTS.StringAxisLabels.newStringAxisLabels(xAxis5);
xAxisLab5.setAxisLabelsStrings(xstringlabels,5);
xAxisLab5.setTextFont(graph5Font);
chartVu.addChartObject(xAxisLab5);
```

[JavaScript]

```
let xstringlabels = [
            "",
            "Western" + "\n" + "Sales" + "\n" + "Region",
            "Eastern" + "\n" + "Sales" + "\n" + "Region",
            "Southern" + "\n" + "Sales" + "\n" + "Region",
            "Northern" + "\n" + "Sales" + "\n" + "Region"
        ];
let xAxisLab5 = QCChartTS.StringAxisLabels.newStringAxisLabels(xAxis5);
xAxisLab5.setAxisLabelsStrings(xstringlabels, 5);
xAxisLab5.setTextFont(graph5Font);
chartVu.addChartObject(xAxisLab5);
```

14. How do I place more than one graph in a view?

One way to create multiple charts is to place multiple HTML Canvas elements (each with a unique ID) and then place a **ChartView** object in each one. You can layout the HTML canvas elements using whatever technique you want, placing them in different elements. The browser manages the position and size of each **Canvas**, and resulting ChartView. Another way is to place multiple charts in the same **ChartView** object. This makes it easier to guarantee alignment between the axes of separate graphs. The trick to doing this is to create separate coordinate system objects (**CartesianCoordinates**, **TimeCoordinates, EventCoordinates for example**) for each chart, and to position the plot area of each coordinate system so that they do not overlap. Use one of the coordinate systems **setGraphBorder**… methods. Many of the examples use this technique, including Bargraphs.BuildDoubleBarPlot, Bargraphs.BuildGroupBars, FinincialExamples.BuildOHLCChart, FinancialExamples.BuildFinancialOptions, DynamicCharts.BuildDynPieChart, PieCharts.BuildPieAndLineChart and PieCharts.BuildPieAndBarChart.

[TypeScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.6) ;

pTransform2 = new QCChartTS.TimeCoordinates();
pTransform2.setGraphBorderDiagonal(0.1, .7, .90, 0.875) ;
```

[JavaScript]

```
pTransform1 = new QCChartTS.TimeCoordinates();
pTransform1.setGraphBorderDiagonal(0.1, .15, .90, 0.6)

pTransform2 = new QCChartTS.TimeCoordinates();
pTransform2.setGraphBorderDiagonal(0.1, .7, .90, 0.875)
```

15. **How do I use your software to generate image files?**

You can right click on any of the charts and invoke the browser copy image menu. You can copy the image to the clipboard and past it into most other programs. If you require a specific format, paste the image into the a drawing program, such as Paint under Windows. From there you can use **File | Save As** to save the image to any format supported by the drawing program. Paint supports TIF, JPEG, BMP, PNG and GIF. You can also use the technique described in Chapter 23, where the BufferedImage class is used to generate a URL to an image already drawn in in a ChartView.

[TypeScript]

```
chartVu.updateDraw();
.
.
.

let image: HTMLImageElement | null = <HTMLImageElement> document.getElementById("imageElement1");
let buffimage: QCChartTS.BufferedImage = QCChartTS.BufferedImage.newBufferedImage(chartVu);
image.src = buffimage.toImageURL();
```

[JavaScript]

```
chartVu.updateDraw();
.
.
.

let image = document.getElementById("imageElement1");
let buffimage = QCChartTS.BufferedImage.newBufferedImage(chartVu);
image.src = buffimage.toImageURL();
```

16. Sometimes the major tick marks of an axis are missing the associated tick mark label ?

The axis labeling routines are quite intelligent. Before the label is drawn at its calculated position, the software does a check to see if the bounding box of the new axis label intersects the bounding box of the previous axis label. If the new label is going to overlap the previous label, the label is skipped. You can override this default behavior by calling the objects **setOverlapLabelMode** method.

```
setOverlapLabelMode (QCChartTS.ChartConstants.OVERLAP_LABEL_DRAW);
```

Another option, for horizontal axes only, is to stagger the tick mark labels. A stagger automatically alternates the line on which the tick mark label is placed.

```
setOverlapLabelMode (QCChartTS.ChartConstants.OVERLAP_LABEL_STAGGER);
```

17. How do I change the order the chart objects are drawn? For example, I want one of my grid objects to be drawn under the charts line plot objects, and another grid object to be drawn top of the charts line plot objects.

There are two ordering methods used to render chart objects. The first method renders the objects in order, as added to the **ChartView** object. Objects added to the view last are drawn on top of objects added first. The second method renders the objects according to their z-order. Objects with the lowest z-order values are rendered first. Objects with equal z-order values are rendered in the ordered they are added to the **ChartView** object. The second method (z-order rendering) is the default method of object rendering used by the **ChartView** class. This default behavior can be changed by call the **ChartView.setZOrderSortEnable(false**) method.

You can change the default z-order value on an object-by-object basis. Call the **GraphObj.setZOrder** method to change the z-order for any given object.

See the section in the manual titled ***Rendering Order of GraphObj Objects*** for information about the default z-values for all chart objects

The example below sets the z-order value of grid1 to something less than the default value (50) of **ChartPlot** objects, and the z-order value of grid2 to something greater than the default value.

[TypeScript]

```
let grid1: QCChartTS.Grid  =  QCChartTS.Grid.newGrid(xAxis, yAxis,
QCChartTS.ChartConstants.Y_AXIS, QCChartTS.ChartConstants.GRID_MAJOR);
grid1.setZOrder(40); // This is actually the default value for the grid z-order
chartVu.addChartObject(grid1);

let  grid2: QCChartTS.Grid  =  QCChartTS.Grid.newGrid(xAxis, yAxis,
QCChartTS.ChartConstants.Y_AXIS, QCChartTS.ChartConstants.GRID_MINOR);
grid2.setZOrder(150); // Grid is drawn after ChartPlot objects
     // which have default z-value of 50
chartVu.addChartObject(grid2);
```

[JavaScript]

```
let grid1 = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MAJOR);
grid1.setZOrder(40); // This is actually the default value for the grid z-order
chartVu.addChartObject(grid1);
let grid2 = QCChartTS.Grid.newGrid(xAxis, yAxis, QCChartTS.ChartConstants.Y_AXIS,
QCChartTS.ChartConstants.GRID_MINOR);
grid2.setZOrder(150); // Grid is drawn after ChartPlot objects
// which have default z-value of 50
chartVu.addChartObject(grid2);
```

18. How to I use a ScrollBar object to control horizontal scrolling of the data in my chart?

There are no standardized scroll bars for use in HTML Canvas objects. So we created our own Canvas based scroll bar class (ScrollbarArea) to use instead. A ScrollbarArea can either be a horizontal or vertical scroll bar. You will find a standalone version demonstrated in the OHLCChart example program. The ScrollbarArea (and ButtonArea) are designed to work with mouse and touch interaction.

The ChartView window has a horizontal and vertical ScrollbarArea already added to it, positioned at the right side (vertical scroll bar) and bottom (horizontal scrollbar). They can be enabled using setEnableHScrollbar (setEnableVScrollbar) methods of the ChartView.

```
chartVu.setEnableHScrollbar(true);
chartVu.setEnableVScrollbar(true);

if ( chartVu.HScrollbar)
{
  chartVu.HScrollbar.setScrollbarAreaEventListener(this);
  chartVu.HScrollbar.setThumbType(QCChartTS.ChartConstants.SQUARE);
  chartVu.HScrollbar.Maximum = numPoints;
  chartVu.HScrollbar.Value = 100;
  this.UpdateXScaleAndAxes(chartVu.HScrollbar.Value);
}
if ( chartVu.VScrollbar)
{
  chartVu.VScrollbar.setScrollbarAreaEventListener(this);
  chartVu.VScrollbar.Minimum = 1;
  chartVu.VScrollbar.Maximum = 50;
  chartVu.VScrollbar.Value =25;
  this.UpdateYScaleAndAxes(chartVu.VScrollbar.Value);

}

chartVu.updateDraw();
```

The example program MouseListeners.BuildLinePlotScrollBar uses two scroll bars, a horizontal scroll bar to control scrolling of the x-axis, and a vertical scroll bar that controls the magnitude of the y-axis. The scroll bar events are sent to the ScrollbarEvent method of the class (both horizontal and vertical scroll bar events), so you need to figure out which one is generating the event and vector to the correct processing function.

[TypeScript]

```
UpdateXScaleAndAxes( index: number)
    {
        let startindex: number = index;
        if (!this.pTransform1) return;

        this.pTransform1.setScaleStartX( startindex);
        this.pTransform1.setScaleStopX( (startindex + 100));
        if (this.xAxis) this.xAxis.calcAutoAxis();
        if (this.yAxis) this.yAxis.calcAutoAxis();
        if (this.xAxisLab) this.xAxisLab.calcAutoAxisLabels();
```

```
        if (this.yAxisLab) this.yAxisLab.calcAutoAxisLabels();
        if (this.chartvu)
          this.chartvu.updateDraw();
    }

UpdateYScaleAndAxes( index: number)
    {
        let startindex: number = Math.max(1,index);
        if (!this.pTransform1) return;


        this.pTransform1.setScaleStartY( -startindex);
        this.pTransform1.setScaleStopY( startindex);
        if (this.xAxis) this.xAxis.calcAutoAxis();
        if (this.yAxis) this.yAxis.calcAutoAxis();
        if (this.xAxisLab) this.xAxisLab.calcAutoAxisLabels();
        if (this.yAxisLab) this.yAxisLab.calcAutoAxisLabels();
        if (this.chartvu)
          this.chartvu.updateDraw();
    }


hScrollBar1_Scroll(args: QCChartTS.ScrollbarAreaEventArgs)
    {
        this.UpdateXScaleAndAxes(args.getScrollValue());
    }

vScrollBar1_Scroll(args: QCChartTS.ScrollbarAreaEventArgs)
    {
        this.UpdateYScaleAndAxes(args.getScrollValue());
    }

ScrollbarEvent(args: QCChartTS.ScrollbarAreaEventArgs)
    {
        if (!this.chartvu) return;
        if (!this.chartvu.HScrollbar) return;
        if (!this.chartvu.VScrollbar) return;

        if (args.getScrollbarArea() == this.chartvu.HScrollbar)
          this.hScrollBar1_Scroll(args);
        else if (args.getScrollbarArea() == this.chartvu.VScrollbar)
          this.vScrollBar1_Scroll(args);
    }
```

[JavaScript]

```
UpdateXScaleAndAxes(index) {
        let startindex = index;
        if (!this.pTransform1)
            return;
        this.pTransform1.setScaleStartX(startindex);
        this.pTransform1.setScaleStopX((startindex + 100));
        if (this.xAxis)
            this.xAxis.calcAutoAxis();
        if (this.yAxis)
            this.yAxis.calcAutoAxis();
        if (this.xAxisLab)
            this.xAxisLab.calcAutoAxisLabels();
        if (this.yAxisLab)
            this.yAxisLab.calcAutoAxisLabels();
        if (this.chartvu)
            this.chartvu.updateDraw();
    }
UpdateYScaleAndAxes(index) {
        let startindex = Math.max(1, index);
        if (!this.pTransform1)
            return;
```

```
        this.pTransform1.setScaleStartY(-startindex);
        this.pTransform1.setScaleStopY(startindex);
        if (this.xAxis)
            this.xAxis.calcAutoAxis();
        if (this.yAxis)
            this.yAxis.calcAutoAxis();
        if (this.xAxisLab)
            this.xAxisLab.calcAutoAxisLabels();
        if (this.yAxisLab)
            this.yAxisLab.calcAutoAxisLabels();
        if (this.chartvu)
            this.chartvu.updateDraw();
    }
hScrollBar1_Scroll(args) {
        this.UpdateXScaleAndAxes(args.getScrollValue());
    }
vScrollBar1_Scroll(args) {
        this.UpdateYScaleAndAxes(args.getScrollValue());
    }
ScrollbarEvent(args) {
        if (!this.chartvu)
            return;
        if (!this.chartvu.HScrollbar)
            return;
        if (!this.chartvu.VScrollbar)
            return;
        if (args.getScrollbarArea() == this.chartvu.HScrollbar)
            this.hScrollBar1_Scroll(args);
        else if (args.getScrollbarArea() == this.chartvu.VScrollbar)
            this.vScrollBar1_Scroll(args);
    }
```

There are other example of our own Canvas-based controls. The NewDemosR2.BuildBoxAndWhisker example program uses a **ButtonArea** objects to specify which Box and Whisker chart options. The OHLCChart example specifies the use of an independent ScrollbarArea object.

19. I am trying to plot 100,000,000 data points and it takes too long to draw the graph. What is wrong with the software and what can I do to make it faster?

The software runs as fast as we can make it. We do not have any hidden switches that will speed up the software. What you need to do is to step back and think about the best way to display your data.

A fundamental issue that many programmers fail to consider is the relationship between the resolution of the rasterized screen image of the plot and the resolution of the data. A typical chart image will have 500-1000 pixels as the horizontal resolution of the plotting area. This would imply that in the 100M data point example above, every horizontal pixel would represent 50K to 100K data points. Obviously this is a terrible mismatch. In fact it is a bad match for datasets that have more than a couple of thousands points. We make this point over and over again in customer support.

So what you do is compress the data before it is displayed. Take the 100M data points and compress them down to 2K data points. The data compression can take several forms. You can take an average of every N points. The resulting dataset will be reduced by a factor of N. You can also find the sum for every N points, the minimum value of every N points, the maximum of every N points, or both the minimum and maximum of every 2N points. The last compression method, minimum and

maximum, will always capture any minimums and maximum in the data. The result is that a 2000 point compressed dataset, where there are at least two data points per pixel of horizontal resolution, will look just like the 100,000,000 point dataset, only display hundreds of times faster. The **Datset** classes all include compression methods (**SimpleDataset.compressSimpleDataset**, **GroupDataset.compressGroupDataset**, **TimeSimpleDataset.compressTimeSimpleDataset** and **TimeGroupDataset.compressTimeGroupDataset, TimeGroupDataset.compressTimeFieldSimpleDataset, TimeGroupDataset.compressTimeFieldGroupDataset**) that operate on the existing dataset and return a new, compressed dataset. The **compressTimeFieldSimpleData** and **compressTimeFieldGroupDataset** are particular useful because they do not use a fixed sample size of N, instead they compress data so that adjacent time values are an increment of a specific time field (ChartObj.DAY_OF_YEAR, QCChartTS.ChartConstants.WEEK_OF_YEAR, QCChartTS.ChartConstants.MONTH, QCChartTS.ChartConstants.Year). Compressing data by month and year obviously requires a varying sample size.

Once created, connect the compressed dataset to the **ChartPlot** object used to display the dataset.

[TypeScript]

```
let nNumPnts: number = 1000000;

let RawDataset: QCChartTS.TimeSimpleDataset  =
QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Raw", xtimedata,  ydata,nNumPnts);
let compressXmode: number  = QCChartTS.ChartConstants.DATACOMPRESS_AVERAGE;
let compressYmode: number  = QCChartTS.ChartConstants.DATACOMPRESS_MINMAX;
let compressTimeField: number  = QCChartTS.ChartConstants.MONTH;
let CompressedDataset: QCChartTS.TimeSimpleDataset  =
RawDataset.compressTimeFieldSimpleDataset( compressXmode,
      compressYmode,
      compressTimeField,
      0, nNumPnts,"Compressed");
```

[JavaScript]

```
 let nNumPnts = 1000000;
 let RawDataset = QCChartTS.TimeSimpleDataset.newTimeSimpleDataset("Raw", xtimedata, ydata,
nNumPnts);
 let compressXmode = QCChartTS.ChartConstants.DATACOMPRESS_AVERAGE;
 let compressYmode = QCChartTS.ChartConstants.DATACOMPRESS_MINMAX;
 let compressTimeField = QCChartTS.ChartConstants.MONTH;
 let CompressedDataset = RawDataset.compressTimeFieldSimpleDataset(compressXmode, compressYmode,
compressTimeField, 0, nNumPnts, "Compressed");
```

20. How do I get data from my database into a chart?

The real question is: How do you get data from your database into JavaScript in a browser, storing sequential data values in data array variables. This is up to you and is independent of the charting software. You can access the database on the server and transmit to the client side browser using JSON or one of many other techniques. Or you can access your database using URL based data requests. What you can't do is read it from the client-side computer, since that would be a security violation. Once you can read individual data elements of your data base it is a trivial matter to place the numeric and calendar data into simple JavaScript variables and from there plot the data.

21. How do I use this charting software to generate chart images "on-the-fly" from a server?

You can generate images on-the-fly using the BufferedImage class, describe in FAQ #15.

22. Can QCChart2D for JavaScript/TypeScript be used to create web application using frameworks other than explicitly described I this manual, Asp.Net for eample.

Yes. Even though we wrote the software using TypeScript, your interface to it is pure client-side JavaScript. Any language that you can interact with using JavaScript you can use this software with. The only requirement is that the environment supports the browser ES6 module specification. We have supplied four examples which demonstrate one way to combine the software with Asp.Net programs. Anybody who is an expert in Asp.Net programming can probably do better than we did.